Ατομική Διπλωματική Εργασία

ΓΡΑΦΙΚΗ ΑΝΑΠΑΡΑΣΤΑΣΗ ΠΑΡΑΛΛΗΛΩΝ ΑΛΓΟΡΙΘΜΩΝ ΜΕ ΤΟ ΕΡΓΑΛΕΙΟ UNITY

Παντελής Στυλιανίδης

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΓΡΑΦΙΚΗ ΑΝΑΠΑΡΑΣΤΑΣΗ ΠΑΡΑΛΛΗΛΩΝ ΑΛΓΟΡΙΘΜΩΝ ΜΕ ΤΟ ΕΡΓΑΛΕΙΟ UNITY

Παντελής Στυλιανίδης

Επιβλέπων Καθηγητής
Χρύσης Γεωργίου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2016

Ευχαριστίες

Με την ολοκλήρωση της διπλωματικής μου εργασίας θα ήταν παράλειψη μου να μην ευχαριστήσω κάποια άτομα τα οποία με την βοήθεια τους συνέβαλαν στην ολοκλήρωση αυτού του δύσκολου έργου που ανέλαβα. Πρώτο από όλους θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου Δρ. Χρύση Γεωργίου για την ευκαιρία που μου έδωσε να εργαστώ στη συγκεκριμένη εργασία , για την καθοδήγηση και την μεγάλη βοήθεια που μου πρόσφερε από την αρχή μέχρι την ολοκλήρωση τις εργασίας.

Επίσης θα ήθελα να ευχαριστήσω τον Χρήστο Βασιλείου για την βοήθεια στην κατανόηση του εργαλείου Unity .

Τέλος θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου για την συνεχή στήριξη που μου πρόσφεραν και την πίστη προς εμένα.

Περίληψη

Το γενικό θέμα με το οποίο ασχολείται η παρούσα διπλωματική είναι ο παραλληλισμός και οι παράλληλοι αλγόριθμοι. Δηλαδή αντί να έχουμε μόνο ένα επεξεργαστή ο οποίος θα εκτελεί κάποιες εργασίες σειριακά για την επίλυση ενός προβλήματος, έχουμε περισσότερους οι οποίοι συνεργάζονται ταυτόχρονα με σκοπό την γρηγορότερη επίλυση αυτού του προβλήματος. Ο παραλληλισμός έχει ακμάσει πολύ τα τελευταία χρόνια και αυτό οφείλετε στο ότι έχει δώσει λύσεις σε πολλά πολύπλοκα προβλήματα και έχει ελαχιστοποιήσει τεράστιους χρόνους επεξεργασίας. Σκοπός της διπλωματικής αυτής ήταν να αναπαρασταθούν γραφικά κάποιοι σημαντικοί αλγόριθμοι ώστε να είναι πιο εύκολη η εκμάθηση τους.

Η εργασία αυτή δημιουργήθηκε εξολοκλήρου στο Game Machine Unity σε δυο διαστάσεις (2D). Πιο παλιά σε κάποια άλλη ΑΔΕ είχε γίνει μια ακόμη προσπάθεια σε Java Swing που φάνηκε πως δεν είχε τις δυνατότητες του Unity

Σαν πρώτο στάδιο έγινε η μελέτη των παράλληλων αλγορίθμων ώστε να εξακριβωθούν όλες οι λεπτομέρειες για να μπορούν να υλοποιηθούν. Στη συνέχεια η εκμάθηση και εξοικείωση του Unity και τέλος η γραφική αναπαράσταση των αλγόριθμων σε αυτό.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός και Κίνητρο Διπλωματικές Εργασίας

Η Διπλωματική αυτή εργασία αναφέρεται στον παραλληλισμό, ο οποίος δίνει γρήγορες λύσεις σε πολύπλοκα προβλήματα που σε σειριακούς υπολογισμούς είναι ανέφικτη η επίλυση τους.[6]

Ο κάθε επεξεργαστής κάνει μια συγκεκριμένη εργασία και στο τέλος ενώνοντας τα αποτελέσματα τους οι επεξεργαστές φτάνουν στο αρχικό πρόβλημα όπου ψάχναμε λύση. Για απλά προβλήματα η διδασκαλία τον αλγόριθμων δεν είναι πολύ δύσκολη αλλά για πιο δύσκολα προβλήματα η υλοποίηση και ο τρόπος σκέψης γίνεται πιο δύσκολος και το ίδιο και η διδασκαλία τους. Έτσι σκοπός αυτής της διπλωματικής ήταν να βοηθήσει στην εκμάθηση κάποιων παράλληλων αλγόριθμων κυρίως σε φοιτητές, αφού όπως έχω προαναφέρει η ακμή που υπάρχει στον παραλληλισμό είναι τεράστια και δεν ξέρουμε μέχρι που μπορεί να φτάσει. Έτσι κατά την άποψη μου για έναν φοιτητή της Πληροφορικής θα ήταν σημαντικό να έχει γνώσεις σε αυτό τον τομέα.

Η όλη αναπαράσταση των αλγορίθμων έχει γίνει στη μηχανή παιχνιδιών Unity[1] σε δισδιάστατο περιβάλλον. Συγκεκριμένα προσφέρει στον χρήστη τις δυνατότητες να επιλέγει διάφορα σενάρια για κάθε πρόβλημα όπως το μέγεθος του προβλήματος και τον αριθμό επεξεργαστών και ποιοι από αυτούς θα είναι ενεργοί και ποιοι όχι. Επίσης ο

χρήστης έχει την δυνατότητα να παρακολουθεί βήμα προς βήμα τι γίνεται στον αλγόριθμο και να μπορεί να επέμβει.

Επιπλέον στόχος ήταν το τελικό προϊόν να είναι όσο πιο λειτουργικό και ευκατανόητο στον οποιοδήποτε χρήστη που θα το χρησιμοποιούσε για να διδάξει ή να διδαχτεί από αυτό.

1.2 Μεθοδολογία Υλοποίησης

Η μεθοδολογία η οποία ακολούθησα για την υλοποίηση αυτής της διπλωματικής εργασίας ήταν η εξής:

- Εγκατάσταση και μελέτη του εργαλείου Unity υλοποιώντας διάφορα παραδείγματα από την ιστοσελίδα του Unity ώστε να εξοικειωθώ καλύτερα με τον έλεγχο της κάμερας και των αντικειμένων των οποίων θα χρησιμοποιούσα στα επόμενα στάδια της διπλωματικής μου.
- Στη συνέχεια έγινε μελέτη και κατανόηση ξεχωριστά του κάθε αλγόριθμου ξεκινώντας από τον αλγόριθμο παράλληλης άθροισης και ακολούθως η υλοποίηση του ως γραφική αναπαράσταση στο Unity.
- Γινόταν συνεχή αξιολόγηση των υλοποιήσεων από τον διδάσκοντα και τους φοιτητές του ΕΠΛ431 του τρέχοντος εξαμήνου για περαιτέρω βελτίωση της λειτουργικότητας και ευχρηστίας.

1.3 Δομή Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία αποτελείται από οκτώ κεφάλαια. Το παρών κεφάλαιο περιγράφει το σκοπό το κίνητρο και την μεθοδολογία η οποία ακολουθήθηκε για την ολοκλήρωση της εργασίας. Το υπόλοιπο της αναφοράς θα ακολουθήσει την εξής δομή.

Στο Κεφάλαιο 2 θα γίνει μια ανασκόπηση των δυο προηγούμενων εργασιών η οποίες είναι σχετικές με την παρούσα διπλωματική εργασία και περιγραφή του υπόβαθρου της εργασίας. Συγκεκριμένα θα αναφερθεί η εργασία της Νάταλης Τεμενέ[4] η οποία ήταν γραφική αναπαράσταση αλγορίθμων στο Unity οι οποίοι είχαν σχέση με κίνηση ρομπότ

και η εργασία της Ελένης Σκιττίδου όπου το θέμα ήταν το ίδιο με την παρούσα διπλωματική με την παράλειψη κάποιων από τους αλγορίθμους και η υλοποίηση είχε γίνει σε Java Swing[3]. Θα γίνουν συγκρίσεις στις δυσκολίες οι οποίες αντιμετωπίσθηκαν, στα αποτελέσματα και στα πλεονεκτήματα μεταξύ των δυο εργαλείων. Επίσης θα γίνει αναφορά στον παραλληλισμό και τα μοντέλα του.

Στη συνέχεια στο Κεφάλαιο 3 θα γίνει μια σύντομη περιγραφή για το Unity[1] και αναφορά σε βασικές γνώσεις για την δημιουργία ενός νέου έργου σε αυτό.

Ακολούθως τα Κεφάλαια 4-8 παρουσιάζονται οι αλγόριθμοι (σε κάθε κεφάλαιο) που υλοποιήθηκαν. Αρχικά περιγράφεται ο αλγόριθμος και ακολουθούν λεπτομέρειες της υλοποίησης του στο Unity. Τέλος συζητούνται διάφορα προβλήματα που προέκυψαν και πως αντιμετωπιστήκαν.

Πιο συγκεκριμένα στο Κεφάλαιο 4 θα παρουσιαστεί ο Αλγόριθμος Παράλληλης Άθροισης[2,5] , στο Κεφάλαιο 5 ο Αλγόριθμος Παράλληλης Αναζήτησης[2,5] , στο Κεφάλαιο 6 ο Αλγόριθμος Παράλληλης Συγχώνευσης[2,5] και ο Αλγόριθμος Παράλληλης Ταξινόμησης[2,5], στο Κεφάλαιο 7 ο Αλγόριθμος W[2,5,6] και στο Κεφάλαιο 8 ο αλγόριθμος X[2,5,6].

Τέλος στο 9ο Κεφάλαιο θα παρουσιαστούν τα συμπεράσματα στα οποία έχω καταλήξει με το πέρας αυτής της διπλωματικής εργασίας δηλαδή μια σύνοψη, προβλήματα τα οποία παρουσιάσθηκαν κατά την υλοποίηση, τα οφέλη που αποκόμισα από την εργασία αυτή και οι τρόποι με τους οποίους θα μπορούσε να βελτιωθεί ως μελλοντική εργασία.

Κεφάλαιο 2

Προηγούμενες Εργασίες Και Υπόβαθρο

2.1 Προηγούμενες Εργασίες

Για την παρούσα εργασία υπήρχαν δύο άλλες εργασίες οι οποίες βοήθησαν στην υλοποίηση αυτής που ήταν οι εξής:
1)Γραφική Αναπαράσταση Παράλληλων Εύρωστων Αλγορίθμων Με Το Εργαλείο Java Swing της Ελένης Σκιττίδου[3]
Στην διπλωματική αυτή υπήρχαν πολλές δυσκολίες από πλευράς υλοποίησης και το τελικό αποτέλεσμα αν και ήταν σωστό δεν ήταν τόσο εύχρηστο στην διδασκαλία και στην κατανόηση όσο θα θέλαμε. Αρχικά η προσπάθεια της Ελένης είχε γίνει χωρίς την χρήση γραφικών Graphics API της Java που ήταν ακόμα πιο δύσκολο και πολύπλοκο. Η εμφάνιση και αισθητική που έδινε δεν ήταν τόσο καλή και η αναπαράσταση της κίνησης δεν ήταν εφικτή. Με την προσθήκη των Graphics υπήρξε μια αισθητή βελτίωση αλλά οι δυνατότητες του εργαλείου αυτού είναι σαφώς πολύ πιο περιορισμένες από μια κονσόλα όπως το Unity που χρησιμοποιείται στην παρούσα εργασία.
Ακολούθως υπήρξε δυσκολία στην δημιουργία ασύγχρονων μοντέλων κάτι που στην παρούσα διπλωματική αντιμετωπίστηκε εύκολα βάζοντας μια καθυστέρηση μεταξύ των βημάτων. Επίσης υπήρχε το πρόβλημα στον περιορισμό των επεξεργαστών και του αριθμού *n* που χρησιμοποιούταν σε κάθε παράδειγμα το οποίο υπήρχε και στην παρούσα

διπλωματική. Στην εργασία της Ελένης είχαν υλοποιηθεί ο Αλγόριθμος Παράλληλης Άθροισης (βέλτιστος και μη-βέλτιστος), ο Αλγόριθμος W και ο Αλγόριθμος X.

2)Implementation And Visual Representation Of A Fault-Tolerant Algorithm For Gathering Fat Crash-Prone Robots On A Plane της Νάταλης Τεμενέ[4]
Η εργασία αυτή χρησιμοποίησε το ίδιο εργαλείο το οποίο χρησιμοποιήθηκε και στην παρούσα εργασία, δηλαδή το Unity[1]. Η εργασία αυτή αναπαριστούσε γραφικά πώς κάποια αυτόνομα ρομπότ μαζεύονταν σε κάποια συγκεκριμένα σημεία ή μαζεύονταν για να φτιάξουν ένα συγκεκριμένο σχηματισμό που πίσω από αυτό βρίσκονταν υλοποιημένοι αλγόριθμοι. Η εργασία αυτή βοήθησε δίνοντας μια πρώτη όψη στο πώς περίπου θα φαίνονται υλοποιημένοι οι αλγόριθμοι της παρούσας εργασίας και τις δυνατότητες που προσφέρει η κονσόλα Unity.

2.2 Υπόβαθρο Μελέτης

Στο υποκεφάλαιο αυτό θα δωθούν λεπτομέριες για τον παράλληλο υπολογισμό και τα μοντέλα του.

2.2.1 Παράλληλος Υπολογισμός

Στην επίλυση προβλημάτων υπάρχουν δύο κύριοι τρόποι επίλυσης τους. Ο σειριακός και ο παράλληλος υπολογισμός. Η κύρια διαφορά μεταξύ των δύο αυτών τρόπων είναι ότι ο σειριακός χρησιμοποιεί μόνο ένα επεξεργαστή ενώ ο παράλληλος τουλάχιστον δύο.

Συγκεκριμένα παράλληλος υπολογισμός είναι η συνεταιριστική και παράλληλη επεξεργασία δεδομένων από περισσότερους από ένα επεξεργαστές που αποσκοπεί στη γρήγορη επίλυση σύνθετων υπολογιστικών προβλημάτων[2,5]. Για μεγαλύτερη αποδοτικότητα και για να είναι εφικτή η επίλυση κάποιων προβλημάτων οι επεξεργαστές πρέπει να είναι του ίδιου τύπου (ομοιογενείς), να βρίσκονται σε κοντινή απόσταση και να έχουν ένα κοινό πρόβλημα το οποίο προσπαθούν να επιλύσουν. Με αυτό τον τρόπο παίρνουμε μια τεράστια υπολογιστική δύναμη[5]

Με την χρήση του σειριακού υπολογισμού η επιστήμη έφτασε σε ένα πολύ ψηλό επίπεδο αφού κατάφερε να επιλύσει πάρα πολλά προβλήματα σε ικανοποιητικούς χρόνους χωρίς πάρα πολύ ψηλό κόστος. Τα τελευταία χρόνια όμως ο παράλληλος υπολογισμός έχει εξελιχθεί ραγδαία και σε πολλά προβλήματα έχει αντικαταστήσει τον σειριακό με πολύ καλύτερους χρόνους, καλύτερα κόστη και σε πολύ πιο απλή μορφή. Επίσης ο παράλληλος υπολογισμός έχει επιλύσει προβλήματα που πριν λίγα χρόνια θεωρούσαμε ακατόρθωτα γιατί υπήρχαν φυσικοί περιορισμοί που ένας σειριακός αλγόριθμος δεν μπορούσε να ξεπεράσει ενώ ο παράλληλος έχει παρακάμψει αυτούς τους περιορισμούς. Κάποιες εφαρμογές του παράλληλου υπολογισμού είναι η πρόβλεψη καιρικών συνθηκών, σεισμολογία , γενετική και άλλα πολλά.

Όλα αυτά έχουν σκοπό την λύση ενός αλγόριθμου, δηλαδή μιας πεπερασμένης ακολουθίας από εντολές εκτελεσμένες σε πεπερασμένο χρόνο για την επίλυση ενός προβλήματος. Ένας παράλληλος αλγόριθμος είναι όταν ο αλγόριθμος αυτός είναι σχεδιασμένος για παράλληλους επεξεργαστές. Για να είναι ένας παράλληλος αλγόριθμος αποδοτικός όμως πρέπει να υπολογίσουμε την πολυπλοκότητα και το κόστος του.

Κάποιες έννοιες οι οποίες χρησιμοποιήσουμε για να τον υπολογισμό αν ένας αλγόριθμος είναι αποδοτικός είναι οι εξής[2]:

- Παράλληλος Χρόνος Εκτέλεσης-T(n)

  Ο χρόνος που χρειάζονται οι επεξεργαστές να λύσουν ένα πρόβλημα μεγέθους n.

- Πλήθος Επεξεργαστών-P(n)

  Ο αριθμός των επεξεργαστών που χρειάστηκαν να λύσουν το πρόβλημα μεγέθους n.

- Κόστος-C(n)

  Το γινόμενο του παράλληλου χρόνου εκτέλεσης T(n) επί το πλήθος επεξεργαστών P(n) που χρησιμοποιήθηκαν να επιλύσουν ένα πρόβλημα μεγέθους n. C(n)=T(n)*P(n)

- Επιτάχυνση S(n)

  Ο χρόνος εκτέλεσης του καλύτερου σειριακού αλγορίθμου που επιλύει ένα πρόβλημα μεγέθους n δια το χρόνο εκτέλεσης του παράλληλου αλγορίθμου που επιλύει το ίδιο πρόβλημα ίδιου μεγέθους.

- Χρόνος εκτέλεσης καλύτερου σειριακού αλγορίθμου $T^*(n)$

  Ο χρόνος που χρειάζεται να επιλύσει το πρόβλημα μεγέθους n ο καλύτερος αλγόριθμος.

Ένας παράλληλος αλγόριθμος θεωρείται βέλτιστος αν για ένα πρόβλημα μεγέθους n το κόστος του να είναι της τάξης του χρόνου επίλυσης του βέλτιστου σειριακού αλγόριθμου.

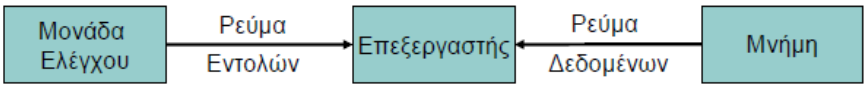
$$C(n) = O(T^*(n))$$

Επιπλέον είναι χρόνου-βέλτιστος αν το $T(n)$ δεν μπορεί να βελτιωθεί.

2.2.2 Μοντέλα Παράλληλου Υπολογισμού

Στον παράλληλο υπολογισμό υπάρχουν πολλά διαφορετικά μοντέλα όπου το καθένα προγραμματίζεται με διαφορετικό τρόπο, ώστε να μπορούμε να επιλέξουμε το κατάλληλο μοντέλο και πιο αποδοτικό σε κάθε περίπτωση. Ο σχεδιαστής πρέπει να έχει ένα συγκεκριμένο μοντέλο υπόψη του όταν σχεδιάζει ένα αλγόριθμο. Αυτό γίνεται με βάση τον αριθμό των ρευμάτων εντολών-Stream of Instructions(τι θα κάνει ο υπολογιστής σε κάθε βήμα – Αλγόριθμος) και τον αριθμό των ρευμάτων δεδομένων-Stream of Data(Δεδομένα εισόδου του Αλγόριθμου) που έχει ο κάθε επεξεργαστής σε κάθε βήμα.

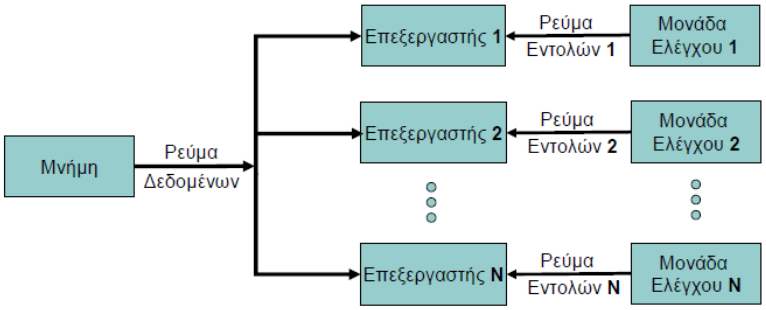
Τα μοντέλα που υπάρχουν είναι τέσσερα με βάση τον Flynn[5]:



Σχήμα 2.1

- **SISD**: **S**ingle **I**nstruction stream, **S**ingle **D**ata stream (σειριακό) (Σχήμα 2.1)

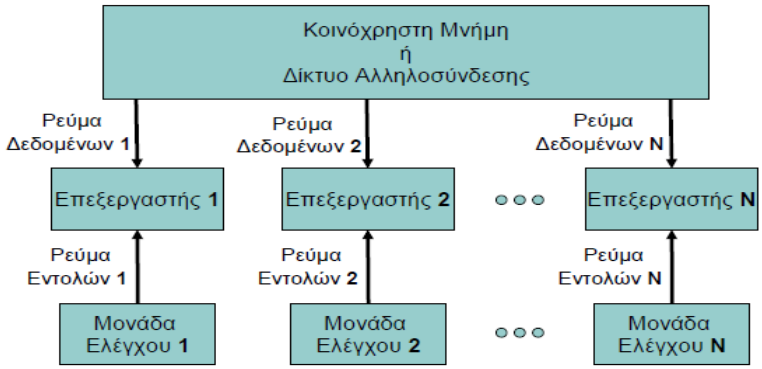
  Ένας επεξεργαστής εκτελεί σειριακά όλους τους υπολογισμούς



Σχήμα 2.2

- **MISD**: **M**ultiple **I**nstruction stream, **S**ingle **D**ata stream (Σχήμα 2.2)

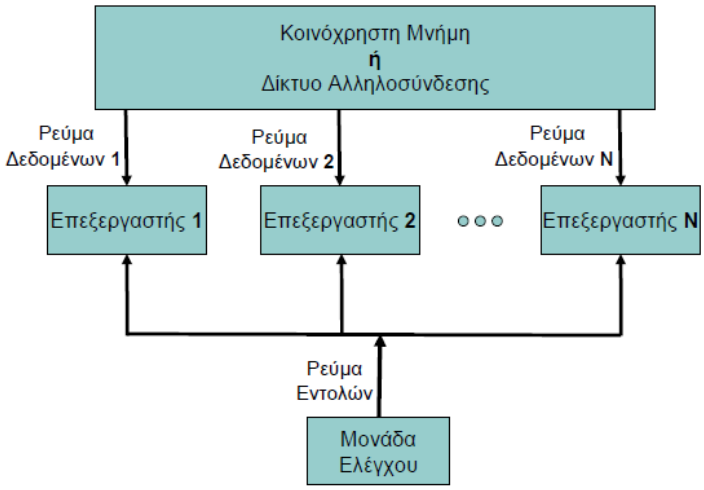
  Έχουμε N επεξεργαστές οι οποίοι έχουν κοινή μνήμη αλλά ο κάθε ένας έχει δική του μονάδα ελέγχου. Σε κάθε βήμα, οι επεξεργαστές μπορούν να δώσουν διαφορετική εντολή πάνω όμως στο ίδιο δεδομένο



Σχήμα 2.3

- **MIMD**: **M**ultiple **I**nstruction stream, **M**ultiple **D**ata stream (Σχήμα 2.3)

  Έχουμε N επεξεργαστές και αντίστοιχα ρεύματα εντολών και δεδομένων . Οι επεξεργαστές μπορούν να εκτελέσουν διαφορετικά προγράμματα επιλύοντας υπο-προβλήματα του προβλήματος ή αλλά προβλήματα



Σχήμα 2.4

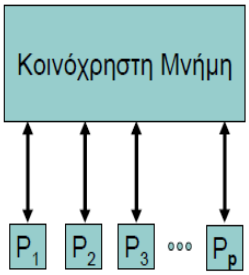
- **SIMD**: **S**ingle **I**nstruction stream, **M**ultiple **D**ata stream (Σχήμα 2.4)

  Ν πανομοιότυποι επεξεργαστές όπου ο κάθε ένας έχει τοπική μνήμη και διαφορετικό ρεύμα δεδομένων

Στην εργασία αυτή θα επικεντρωθούμε στο υπομοντέλο PRAM: **P**arallel **R**andom **A**ccess **M**achine (Παράλληλη Μηχανή Τυχαίας Προσπέλασης) , F-PRAM και A-PRAM.

2.2.3 Μοντέλο PRAM

Το μοντέλο PRAM[5] είναι ένα μοντέλο κοινόχρηστης μνήμης (Σχήμα 2.5) όπου έχουμε p πανομοιότυπους συγχρονισμένους επεξεργαστές και ο κάθε ένας διαθέτει τοπική μνήμη. Χρησιμοποιούν την κοινόχρηστη μνήμη ως μέσο επικοινωνίας όπου μπορούν να πάρουν τα δεδομένα εισόδου από αυτήν και να καταγράψουν ενδιάμεσα αποτελέσματα ή δεδομένα εξόδου σε αυτή . Στο βασικό μοντέλο δεν μπορούν δύο επεξεργαστές να γράψουν ή να διαβάσουν από κοινή κυψελίδα μνήμης. Ο χρόνος πρόσβασης που χρειάζεται ο κάθε επεξεργαστής προς την κοινόχρηστη μνήμη είναι Θ(1). Το μοντέλο αυτό κατατάσσεται στην κατηγορία SIMD. Το μοντέλο αποτελείτε από συγχρονισμένα βήματα. Δηλαδή σε ένα βήμα δεν μπορούν δύο επεξεργαστές να εκτελούν δύο διαφορετικές εργασίες, είτε θα εκτελούν την ίδια είτε ο ένας από τους δύο θα μένει αδρανές και θα εκτελεί την εντολή no-op μέχρι το επόμενο βήμα για την σωστή επίλυση του αλγορίθμου.



Σχήμα 2.5

Όπως προαναφέραμε στο μοντέλο αυτό η μνήμη είναι κοινόχρηστη. Έτσι στην περίπτωση που δύο επεξεργαστές προσπαθήσουν να γράψουν ή να διαβάσουν από μια κοινή κυψελίδα μνήμης τι θα γίνει; Η απάντηση σε αυτό το ερώτημα καθορίζεται από τον τύπο του PRAM που χρησιμοποιείται. Έχουμε 4 διαφορετικούς τύπους PRAM, με βάση των περιορισμών στην ταυτόχρονη πρόσβαση στη μνήμη:

- **EREW: E**xclusive **R**ead, **E**xclusive **W**rite

Δεν επιτρέπεται περισσότεροι από ένα επεξεργαστές να διαβάσουν ή να γράψουν ταυτόχρονα στην ίδια κυψελίδα της κοινόχρηστης τους μνήμης.

- **CREW: C**oncurrent **R**ead, **E**xclusive **W**rite

Μπορούν όλοι οι επεξεργαστές να διαβάσουν από την ίδια κυψελίδα ταυτόχρονα, αλλά μόνο ένας μπορεί να γράψει στην ίδια κυψελίδα της κοινόχρηστης τους μνήμης.

- **ERCW: E**xclusive **R**ead, **C**oncurrent **W**rite

Μπορούν όλοι οι επεξεργαστές να γράψουν στην ίδια κυψελίδα ταυτόχρονα, αλλά μόνο ένας μπορεί να διαβάσει από την ίδια κυψελίδα της κοινόχρηστης τους μνήμης.

- **CRCW: C**oncurrent **R**ead, **C**oncurrent **W**rite

Δεν υπάρχουν περιορισμοί στην ταυτόχρονη πρόσβαση στη κοινόχρηστη μνήμη

Το ταυτόχρονο διάβασμα από περισσότερους από ένα επεξεργαστές σε μια κοινή κυψελίδα μνήμης δεν παρουσιάζει κανένα πρόβλημα αφού απλά διαβάζουν την τιμή αυτή και την κρατούν ανέπαφη, έτσι δεν έχει σημασία με ποια σειρά θα την διαβάσουν. Αντιθέτως στην ταυτόχρονη γραφή σε μια κυψελίδα μνήμης μπορεί να παρουσιαστεί πρόβλημα αφού δεν θα ξέρουμε ποια είναι η τελική τιμή που υπάρχει στην κυψελίδα

τώρα. Γι' αυτό τον λόγο το CRCW χωρίζεται σε τρία υπομοντέλα ώστε αυτά να χειριστούν την ταυτόχρονη γραφή. Τα μοντέλα αυτά είναι τα εξής:

- **Common(κοινό) CRCW PRAM**

  Στο μοντέλο αυτό όλοι οι επεξεργαστές πρέπει να γράψουν την ίδια τιμή στη συγκεκριμένη κυψελίδα κοινόχρηστης μνήμης
- **Arbitrary(αυθαίρετο) CRCW PRAM**

  Ένας από τους επεξεργαστές από αυτούς που προσπαθούν να γράψουν στην ίδια κυψελίδα μνήμης επιλέγεται αυθαίρετα και καταφέρνει να γράψει αυτός την δική του τιμή.
- **Priority CRCW PRAM**

  Σε αυτό το μοντέλο θέτουμε μια προτεραιότητα με την οποία καθορίζουμε τον επεξεργαστή τον οποίο θα επικρατήσει η τιμή του μετά την εγγραφή στη συγκεκριμένη κυψελίδα μνήμης. Για παράδειγμα η επιλογή μπορεί να γινεί γράφοντας πάντα την τιμή του επεξεργαστή με τον μικρότερο προσωπικό αριθμό από αυτούς που προσπαθούν να γράψουν την συγκεκριμένη χρονική στιγμή.

Το κάθε μοντέλο έχει τους δικούς τους περιορισμούς με το EREW PRAM να είναι το πιο περιοριστικό μοντέλο και το Priority CRCW PRAM το πιο ελαστικό μοντέλο. Έτσι κατατάσσουμε τα μοντέλα σε μια ιεραρχία δυναμικότητας με την εξής σειρά:

EREW → CREW → Common CRCW → Arbitrary CRCW → Priority CRCW

Ένας αλγόριθμος ο οποίος σχεδιάστηκε για ένα πιο αδύνατο μοντέλο μπορεί να επανασχεδιαστεί για ένα πιο δυνατό μοντέλο με την ίδια πολυπλοκότητα χρόνου και κόστους σε ένα πιο δυνατό μοντέλο. Δεν συμβαίνει το ίδιο και στην αντίθετη περίπτωση αφού για να προσομοιωθεί ένας αλγόριθμος από ένα δυνατό μοντέλο σε ένα πιο αδύνατο θα χρειαστεί μεγαλύτερο αριθμό επεξεργαστών άρα και κόστος ή μεγαλύτερο χρόνο εκτέλεσης.

Όταν υλοποιούμε ένα αλγόριθμο σε κάποιο μοντέλο πρέπει να έχουμε υπόψη τους περιορισμούς. Για παράδειγμα αν το μοντέλο μας δεν επιτρέπει ταυτόχρονη γραφή τότε σε κανένα σημείο δεν επιτρέπεται να χρησιμοποιήσουμε ταυτόχρονη γραφή.

Ένα παράδειγμα αλγόριθμου που χρησιμοποιεί PRAM και υλοποιείται στην παρούσα εργασία είναι η παράλληλη άθροιση

2.2.4 Μοντέλο F-PRAM

Το F-PRAM είναι πανομοιότυπο με το PRAM όπου κληρονομεί σχεδόν όλα τα χαρακτηριστικά του[5]. Δηλαδή είναι ένα συγχρονισμένο μοντέλο όπου οι επεξεργαστές μεταβαίνουν από το ένα βήμα στο επόμενο με τον ίδιο ρυθμό και έχουν μια κοινή μνήμη στην οποία έχουν πρόσβαση σε Θ(1) χρόνο. Στο PRAM όμως οι επεξεργαστές δεν υπόκεινται σε σφάλματα κατάρρευσης ενώ στο F-PRAM υπόκεινται . Αυτή είναι η κύρια διαφορά των δυο μοντέλων όπου στο μοντέλο F-PRAM οι επεξεργαστές παρουσιάζουν σφάλματα κατάρρευσης και το μοντέλο είναι σχεδιασμένο με αυτό τον τρόπο έτσι ώστε να αντιμετωπίζονται και να συνεχίζουν τον αλγόριθμο οι εν λειτουργεία επεξεργαστές και να δίνουν λύση.

Σφάλμα κατάρρευσης ορίζουμε όταν ένας επεξεργαστής σε οποιαδήποτε σημείο του υπολογισμού σταματήσει την λειτουργεία του και δεν επιστρέψει πίσω στον υπολογισμό. Για ένα υπολογισμό λέμε ότι υπόκειται σε f σφάλματα κατάρρευσης αν από την αρχή του υπολογισμού μέχρι το τέλος μπορούν να καταρρεύσουν το πολύ μέχρι f επεξεργαστές. Απαραίτητη προϋπόθεση για την ολοκλήρωση του προβλήματος είναι να υπάρχει τουλάχιστον ένας επεξεργαστής σε λειτουργία σε όλα τα σημεία του αλγόριθμου.

Ένας αλγόριθμος υλοποιημένος σε F-PRAM ο οποίος λύνει ένα συγκεκριμένο πρόβλημα αποδοτικά και ανέχεται σφάλματα κατάρρευσης επεξεργαστών ονομάζεται εύρωστος. Ο σχεδιασμός τους είναι αρκετά δύσκολος και πολύπλοκος επειδή για να αυξηθεί η απόδοση πρέπει να μειωθεί ο χρόνος εκτέλεσης ή ο αριθμός των επεξεργαστών και για να ανεχτούμε σφάλματα πρέπει να γίνει ακριβώς το αντίθετο, δηλαδή να μειωθούν.

Ένα παράδειγμα εύρωστου αλγόριθμου το οποίο υλοποιείται στην παρούσα εργασία είναι το Write all (Algorithm W)[5]

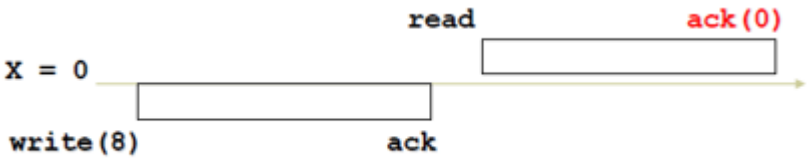
2.2.5 Μοντέλο A-PRAM

Το μοντέλο A-PRAM είναι υπομοντέλο του MIMD. Η κύρια διαφορά με το απλό PRAM είναι ότι οι επεξεργαστές ενεργούν εντελώς ασυγχρόνιστα[5]. Στο μοντέλο αυτό οι επεξεργαστές μπορούν να διαβάσουν ή να γράψουν σε οποιαδήποτε κυψελίδα της μνήμης όμως η πρόσβαση στην μνήμη μπορεί να πάρει διαφορετικό χρόνο. Ο ένας επεξεργαστής μπορεί να είναι πιο γρήγορος από τον άλλο και το μοντέλο αντιμετωπίζει σφάλματα κατάρρευσης. Ο χρόνος δεν παίζει κανένα ουσιαστικό ρόλο στο σχεδιασμό αλγορίθμων με το μοντέλο αυτό και είναι αδύνατο να διακρίνεις αν ένας επεξεργαστής έχει καταρρεύσει ή είναι πολύ αργός.

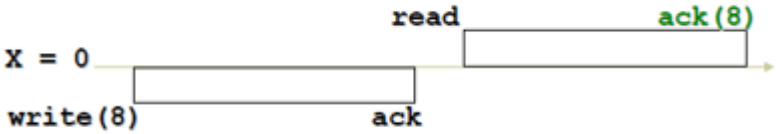
Σε αυτό το μοντέλο δεν υπάρχει η έννοια της ταυτόχρονης πρόσβασης στην μνήμη αλλά υπάρχει η έννοια της ατομικότητας. Δηλαδή κάθε ακολουθία ασυγχρόνιστων προσβάσεων από τον ίδιο ή και διαφορετικούς επεξεργαστές σε μια συγκεκριμένη κυψελίδα μνήμης μπορεί να τοποθετηθεί σε μερική διάταξη. Συγκεκριμένα η τιμή που θα επιστρέφεται με την ανάγνωση από μια κυψελίδας μνήμης να είναι η τελευταία ολοκληρωμένη γραφής ή η τιμή που συμβαίνει το ίδιο χρονικό διάστημα χωρίς να έχει ολοκληρωθεί. Ακόμη ισχύει η ατομικότητα αν η τιμή που θα επιστρέψει η επόμενη ανάγνωση της κυψελίδας είτε θα είναι ίδια με την προηγούμενη ανάγνωση είτε θα έχει την τιμή μιας γραφής που έγινε μετά την προηγούμενη ανάγνωση

Στη συνέχεια δίνονται κάποια παραδείγματα για τις περιπτώσεις παραβίασης της ατομικότητας και πιο θα έπρεπε να είναι το σωστό αποτέλεσμα.

Στην περίπτωση αυτή (Σχήμα 2.6) βλέπουμε παραβίαση της πρώτης περίπτωσης της ατομικότητας αφού η γραφή έχει ολοκληρωθεί με την τιμή 8. Έτσι στη συνέχεια που διαβάζει την τιμή ο επεξεργαστής θα έπρεπε να διαβάζει την τιμή 8 που είναι η τιμή της τελευταίας ολοκληρωμένης γραφής και όχι την τιμή 0 που διάβασε(Σχήμα 2.7).

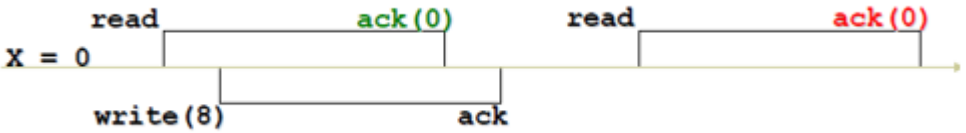


Σχήμα 2.6



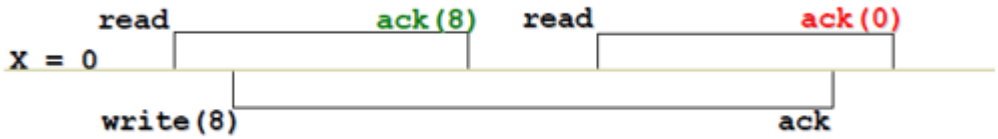
Σχήμα 2.7

Στο επόμενο παράδειγμα (Σχήμα 2.8) βλέπουμε ότι στην πρώτη περίπτωση σωστά ενέργησε ο επεξεργαστής που διάβασε την τιμή 0 γιατί ήταν η τελευταία ολοκληρωμένη γραφή. Αλλά στην δεύτερη περίπτωση θα έπρεπε να διαβάσει την τιμή 8 αφού η γραφή έχει ολοκληρωθεί πριν την ανάγνωση της τιμής. Έτσι λανθασμένα διαβάζει την τιμή 0.



Σχήμα 2.8

Στο τελευταίο παράδειγμα (Σχήμα 2.9) ο επεξεργαστής έχει διαβάσει την τιμή 8 της γραφής που γίνεται την ίδια στιγμή μαζί με την ανάγνωση που είναι σωστό με βάση τον πρώτο κανόνα ατομικότητας που αναφέραμε. Στην δεύτερη περίπτωση λανθασμένα διαβάζει την τιμή 0 ο επεξεργαστής αφού διαφωνεί με τον δεύτερο κανόνα που αναφέραμε αφού δεν έχει την τιμή της προηγούμενης ανάγνωσης ούτε την τιμή μιας μεταγενέστερης γραφής. Άρα θα έπρεπε η τιμή να είναι 8.



Σχήμα 2.9

Κεφάλαιο 3

Unity

3.1 Εισαγωγή

Το Unity [1,7] είναι ένα εργαλείο με τεράστιες δυνατότητες το οποίο παρέχεται δωρεάν, απλά χρειάζεται να το κατεβάσεις από την κεντρική του ιστοσελίδα. Είναι μια μηχανή παιχνιδιών η οποία μπορεί να δημιουργήσει 2D και 3D παιχνίδια αλλά και γραφικές απεικονίσεις. Το Unity είναι πολύ εύκολο στην χρήση, δίνοντας την ευκαιρία σε άπειρους χρήστες να μπορούν να δημιουργήσουν τα δικά τους απλά παιχνίδια αλλά και σε πιο έμπειρους να δημιουργήσουν παιχνίδια τα οποία βγαίνουν και στην αγορά.

Επίσης η κοινότητα του Unity στην ιστοσελίδα είναι πολύ ενεργή και πρόθυμη να δώσει βοήθεια σε αρχάριους χρήστες. Η κεντρική ιστοσελίδα παρέχει πολλά παραδείγματα τα οποία μπορεί κάποιος να μάθει βασικά πράγματα όπως μετακίνηση της κάμερας μέχρι και παραδείγματα για υλοποίηση έτοιμων παιχνιδιών τα οποία είναι πολύ βοηθητικά στην εκμάθηση[1].

Οι περισσότερες ενέργειες μπορούν να γίνουν τόσο χειροκίνητα αλλά και με την δημιουργία τους με κώδικα σε μορφή scripts τα οποία με μεταφορά και απόθεση τα προορίζεις στα αντικείμενα τα οποία θα τα χρησιμοποιήσουν. Το scripting υποστηρίζει τις γλώσσες προγραμματισμού JavaScript, C# και Boo[1,7]. Στην παρούσα εργασία χρησιμοποιήθηκε C#.

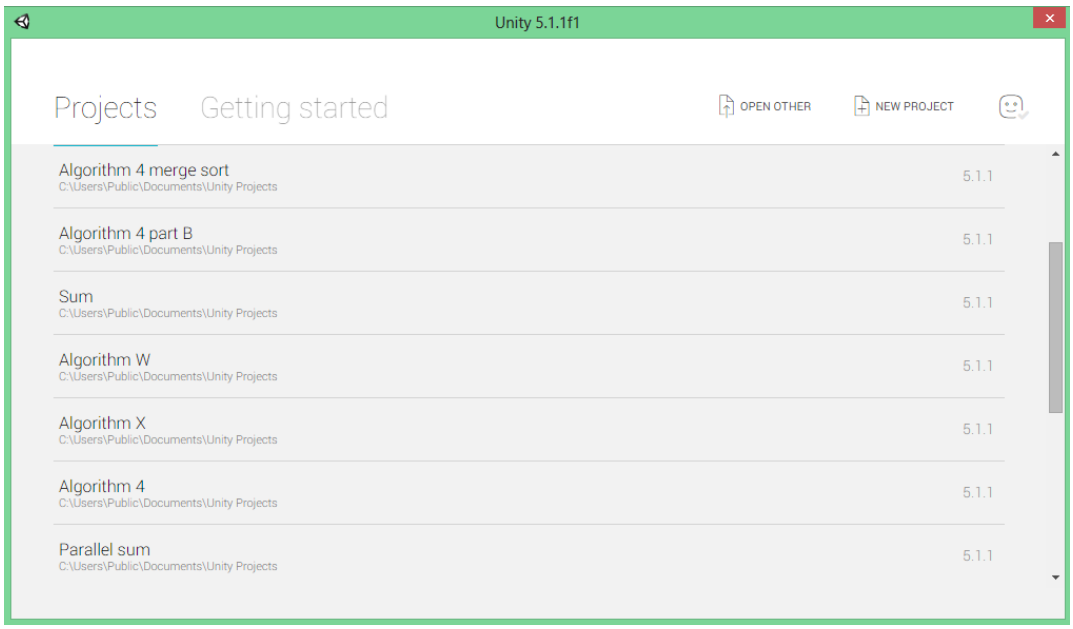
Μια τεράστια δυνατότητα η οποία παρέχει είναι ότι μετά την ολοκλήρωση του έργου σου, το αρχείο το οποίο δημιουργείται μπορείς να επιλέξεις για ποια κονσόλα

προορίζεται και οι επιλογές σου είναι πάρα πολλές. Πιο συγκεκριμένα μπορείς να επιλέξεις λειτουργικό Windows, Mac, Android και σε iOS, ή ακόμα και σε παιχνιδομηχανές όπως το Xbox 360, το PlayStation 3 και το Wii.[7]

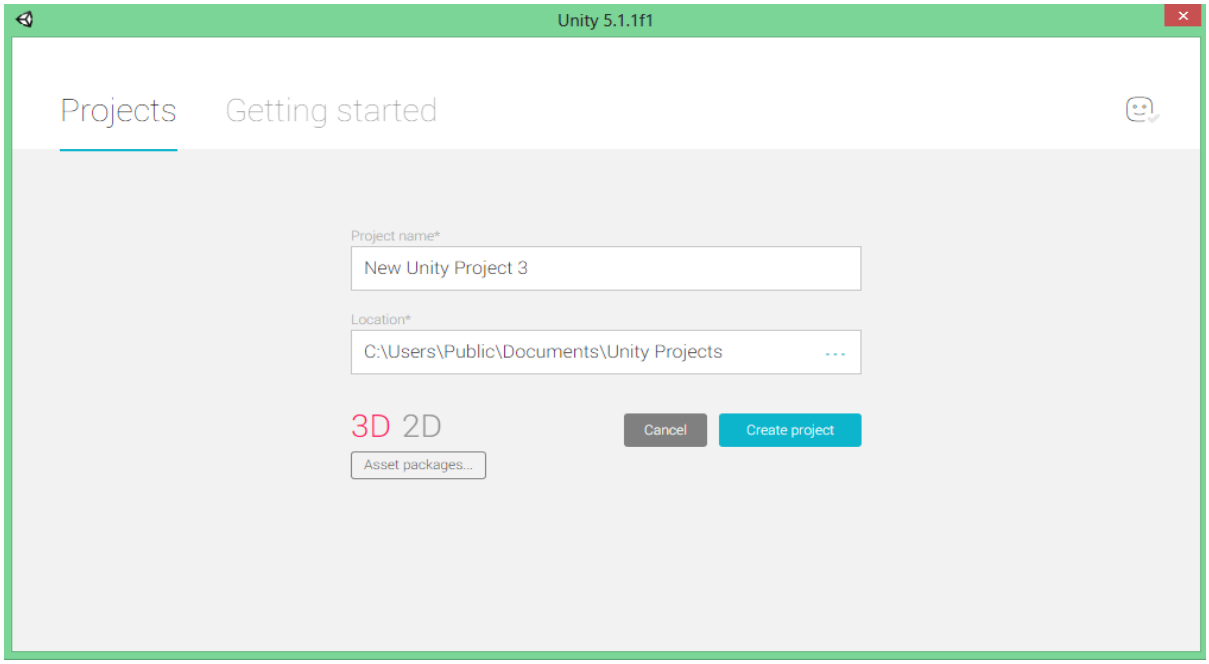
Το Unity επίσης αν και καινούργιο στην αγορά έχει βραβευτεί με αξιέπαινα βραβεία όπως το 2006 που διαγωνίστηκε στο Apple Design Awards και βγήκε δεύτερο στην καλύτερη χρήση γραφικών ή όπως το 2010 που κέρδισε το βραβείο Wall Street Journal στην κατηγορία καινοτόμου τεχνολογίας

3.2 Βασικές Γνώσεις Για Την Δημιουργία Ενός Έργου

Ξεκινώντας το Unity, εμφανίζεται ένα αρχικό παράθυρο (Σχήμα 3.1) όπου ο χρήστης καλείται να επιλέξει αν θα τροποποιήσει ένα παλιό έργο ή αν θα δημιουργήσει ένα νέο. Αν επιλέξει ένα υπάρχον έργο αυτό ανοίγει και το επεξεργάζεται, αν επιλέξει να δημιουργήσει ένα νέο τότε καλείται να επιλέξει το όνομα του έργου, που θα αποθηκευτεί, ποια πακέτα με "assets" θα φορτωθούν και αν θα είναι της μορφής 2D ή 3D (Σχήμα 3.2).



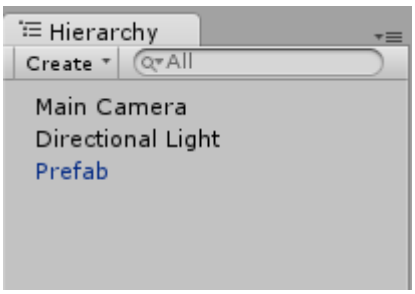
Σχήμα 3.1: Πρώτη Οθόνη



Σχήμα 3.2: Επιλογή Create new Project

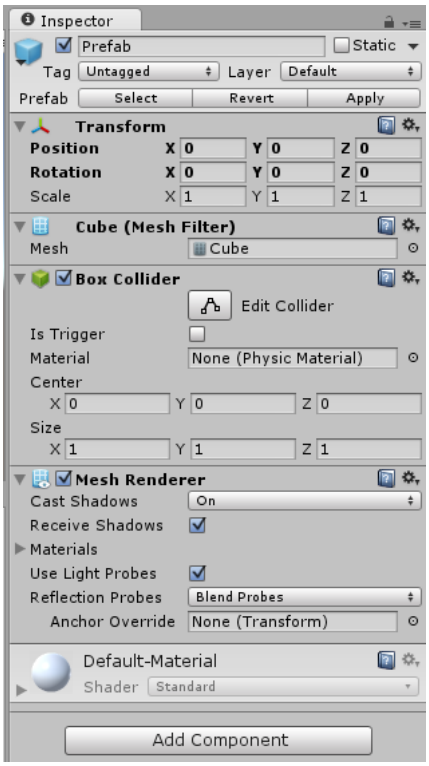
Στη συνέχεια ανοίγει ένα νέο παράθυρο στο οποίο εδώ γίνονται τα πάντα. Στο κέντρο βρίσκονται δύο οθόνες. Μια στην οποία επεξεργάζεσαι την συγκεκριμένη σκήνη την "scene panel" όπου τοποθετούνται όλα τα αντικείμενα, δημιουργούνται τα "game levels" και το περιβάλλον. και μια στην οποία φαίνεται τι θα δείχνει η κάμερα στην πρώτη φάση του παιχνιδιού όταν το τρέξεις πατώντας το κουμπί "play" .

Στα αριστερά βρίσκεται το "hierarchy"(Σχήμα 3.3) όπου βρίσκονται όλα τα αντικείμενα που υπάρχουν την συγκεκριμένη στιγμή στη σκηνή. Για να δημιουργήσεις ένα αντικείμενο απλά επιλέγεις το "create" το οποίο βρίσκεται στο "hierarchy" και επιλέγεις τι ακριβώς θέλεις να δημιουργήσεις.



Σχήμα 3.3: Επιλογή Create new Project

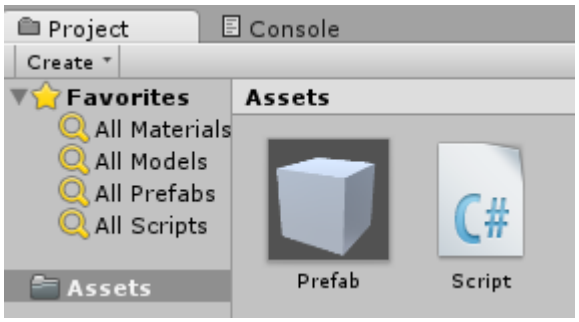
Στη συνέχεια αφού δημιουργήσεις τα αντικείμενα που χρειάζεσαι όταν επιλέξεις ένα οποιοδήποτε αντικείμενο, στα δεξιά εμφανίζεται το inspector (Σχήμα 3.4) όπου βρίσκονται όλα τα στοιχεία για το συγκεκριμένο αντικείμενο όπου μπορείς με το "transform" να το τοποθετήσεις όπου θέλεις στη σκηνή σου και με το "add component" να προσθέσεις ότι θέλεις σε αυτό. Επίσης υπάρχει το "tag" το οποίο βοηθά στην ομαδοποίηση των αντικειμένων σου.



Σχήμα 3.4: Inspector

Μετά την τροποποίηση του αντικειμένου σου μπορείς να το θέσεις ως "prefab" όπως έγινε στο συγκεκριμένο έργο στην περίπτωση των επεξεργαστών σε κάθε αλγόριθμο. Με αυτό τον τρόπο μπορείς με κώδικα να καλείς την δημιουργία αυτών των αντικειμένων και στη συνέχεια τα τροποποιείς όπως εσύ θέλεις. Ο κώδικας αυτός γίνεται με την δημιουργία ενός script και γράφοντας εκεί εντολές σε μια από τις γλώσσες προγραμματισμού C#, Javascript ή Boo και βάζοντας το script αυτό σέρνοντας το πάνω στο prefab.

Κάτω βρίσκεται το “project” και το “console” (Σχήμα 3.5) όπου στο “project” βρίσκονται τα πάντα τα οποία αποθήκευσες για το παρών έργο για παράδειγμα σκηνές, “scripts”, “prefabs” και στο “console” όπου δείχνει όλα τα “errors”, “warnings” και “debug” μηνύματα τα οποία υπάρχουν πριν την εκτέλεση του παιχνιδιού και κατά την διάρκεια της εκτέλεσης



Σχήμα 3.5: Project and Console

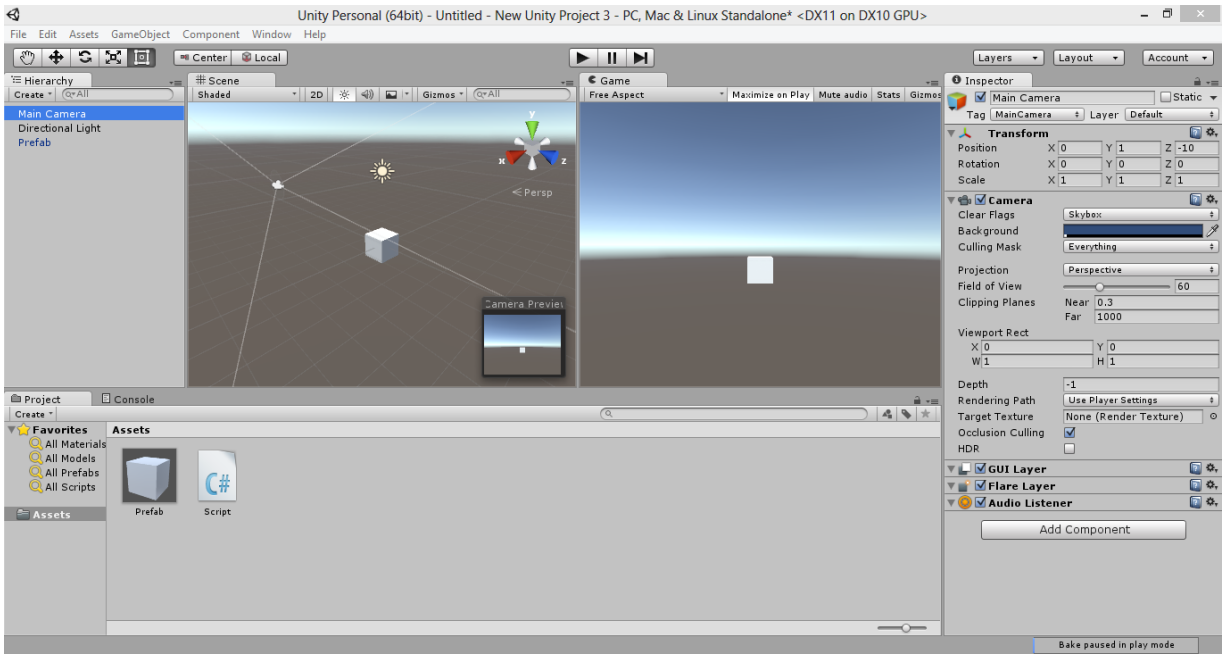
Πάνω βρίσκονται τα κουμπιά “Play”, “Pause”, “Frame Step” για την εξομοίωση της σκηνής (Σχήμα 3.6). Με το πάτημα του κουμπιού “Play” μπορείς να τρέξεις το παιχνίδι για να δεις πως φαίνεται μέχρι στιγμής και το Unity περνά απευθείας στο “Game View”. Υπάρχει και η επιλογή του “Maximize on play” που όταν είναι επιλεγμένο και πατήσεις “Play” τρέχει σε πλήρης οθόνη. Επίσης μπορείς να το κάνεις “Pause” ή με το πάτημα του τρίτου κουμπιού να προχωρήσεις κατά ένα “frame” και να γίνει αυτόματα pause το παιχνίδι σε αυτό το “frame”. Κατά την διάρκεια που τρέχει το παιχνίδι μπορούμε να κάνουμε ότι αλλαγές θέλουμε για να κάνουμε ελέγχους αφού οι αλλαγές είναι προσωρινές και εξαφανίζονται μόλις ξαναπατήσουμε το κουμπί “Play”

Σχήμα 3.6: Play,pause,next frame

Ακόμη επιλέγοντας τα “assets” πάνω στο μενού μπορείς να κάνεις εγκαταστήσεις έτοιμα πακέτα με αντικείμενα για παράδειγμα αυτοκίνητα, δέντρα , έδαφος ή οτιδήποτε άλλο χρειάζεσαι για το έργο σου.

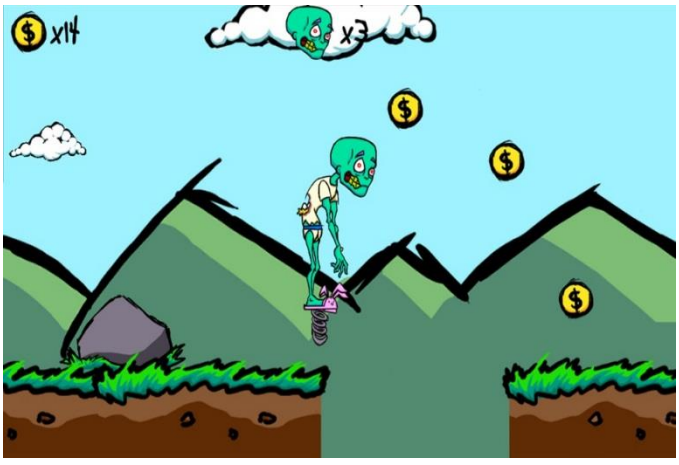
Με την ολοκλήρωση της σκηνής σου μπορείς να την αποθηκεύσεις και να ξεκινήσεις μια νέα σκήνη από την αρχή και όλα τα αντικείμενα του έργου που είναι δημιουργημένα μπορούν να χρησιμοποιηθούν. Ένα έργο μπορεί να αποτελείται από πολλές σκηνές όπου αν το έργο είναι ένα παιχνίδι αυτές οι σκηνές μπορεί να είναι τα στάδια του.

Επίσης την διαμόρφωση για το που θα βρίσκονται όλα αυτά που περίγραψα την διαμορφώνει ο κάθε χρήστης όπως τον βολεύει αυτόν με μια απλή μεταφορά και απόθεση. Στο Σχήμα 3.7 μπορούμε να δούμε μια ολοκληρωμένη εικόνα από την φάση δημιουργίας του έργου και ακολούθως στο Σχήμα 3.8 και Σχήμα 3.9 ένα παιχνίδι 3D και ένα 2D αντίστοιχα υλοποιημένα στο Unity.



Σχήμα 3.7:Ολοκληρωμένη εικόνα στην φάση δημιουργίας του έργου

Σχήμα 3.8:Παράδειγμα 3D στο Unity



Σχήμα 3.9:Παράδειγμα 2D στο Unity

Κεφάλαιο 4

Αλγόριθμος Παράλληλης Άθροισης

4.1 Πρόβλημα

Το πρόβλημα της παράλληλης άθροισης είναι το εξής: *Δεδομένου μιας λίστας από n αριθμούς πρέπει να υπολογίσουμε το άθροισμα S τους.[2,5]*

4.2 Σειριακός Αλγόριθμος

Στην περίπτωση του σειριακού υπολογισμού, όπου έχουμε δηλαδή μόνο ένα επεξεργαστή αυτός ο επεξεργαστής θα κάνει σειριακά τα αθροίσματα για να μας δώσει την λύση. Συγκεκριμένα θα διαβάζει ένα προς ένα τους n αριθμούς και θα τους προσθέτει στο ολικό άθροισμα και θα χρειαστεί *n-1* αθροίσματα. Πιο κάτω φαίνεται το κύριο κομμάτι του κώδικα και η πολυπλοκότητα του.

```
Set S = α1
For j = 2 to n do
        S = S + αj
end do
```

Πολυπλοκότητα: $T_1(n) = C_1(n) = \Theta(n)$

Ο χρόνος εκτέλεσης του αλγόριθμου αυτού είναι Θ(*n*). Όπως φαίνεται και από το πιο πάνω κομμάτι κώδικα στο πρώτο βήμα ο αλγόριθμος θέτει το άθροισμα ως τον πρώτο αριθμό της λίστας που αυτό απαιτεί σταθερό χρόνο Θ(1). Στη συνέχεια μπαίνει σε ένα βρόγχο όπου προσθέτει τον αριθμό που διαβάζει στο άθροισμα και αυτό απαιτεί επίσης σταθερό χρόνο Θ(1). Η διαδικασία του βρόγχου επαναλαμβάνεται *n-1* φορές για να διαβαστούν τα υπόλοιπα *n-1* στοιχεία που δεν είχαν διαβαστεί και να προστεθούν στο αρχικό μας άθροισμα. Έτσι ο χρόνος εκτέλεσης του σειριακού αλγόριθμου άθροισης αποδεικνύεται πως είναι Θ(*n*) , όπως και το κόστος του αφού χρησιμοποιούμε μόνο ένα επεξεργαστή .

4.3 Παράλληλος Αλγόριθμος

Ακολούθως γίνεται περιγραφή του αλγόριθμου παράλληλης άθροισης και λεπτομέρειες για την γραφική αναπαράσταση που υλοποιήθηκε στην εργασία.

4.3.1 Περιγραφή Αλγορίθμου

Το πρόβλημα όπως και στη σειριακή μας επίλυση παραμένει το ίδιο, η άθροιση δηλαδή *n* αριθμών. Η ουσιαστική διαφορά με τον σειριακό αλγόριθμο είναι η χρησιμοποίηση περισσότερους από ένα επεξεργαστές οι οποίοι θα κάνουν την άθροιση ανά ζεύγη με σκοπό η επίλυση να γίνεται γρηγορότερα.

Για τον αλγόριθμο αυτό οι αριθμοί μας θα βρίσκονται σε n συνεχόμενες κυψελίδες της κοινής μνήμης και θα χρησιμοποιήσουμε *n/2* επεξεργαστές. Στο πρώτο βήμα ο κάθε ένας θα ανατεθεί στην κυψελίδα της κοινόχρηστης μνήμης στην θέση με τον προσωπικό του αριθμό επί δύο τον οποίο θα διαβάσει και θα τον προσθέσει με τον επόμενο αριθμό δηλαδή τον προσωπικό του αριθμό επί δύο συν ένα (KM[(*i*2-1*)]+KM[(*i*2*)]). Για παράδειγμα ο επεξεργαστής με προσωπικό αριθμό 0 θα ανατεθεί στην θέση (1*2-1) και

στην θέση (1*2) άρα στις θέσεις ένα και δύο αντίστοιχα στην κοινόχρηστη μνήμη. Αυτό γίνεται από όλους τους *n/2* επεξεργαστές και μετά την πρόσθεση των δύο αριθμών αποθηκεύουν το άθροισμα στην θέση με τον προσωπικό τους αριθμό στην κοινόχρηστη μνήμη και αντικαθιστούν την παλιά τιμή.

Μετά την ολοκλήρωση του πρώτου βήματος οι αριθμοί που μας μένουν να υπολογίσουμε έχουν μείνει οι μισοί αλλά οι επεξεργαστές μας παραμένουν οι ίδιοι. Για τον λόγο αυτό επειδή η διαδικασία που θα ακολουθήσουμε στα επόμενα βήματα είναι η ίδια με το πρώτο βήμα οι μισοί επεξεργαστές δεν θα κάνουν καμία λειτουργία (θα είναι σε κατάσταση no-op). Συγκεκριμένα οι πρώτοι μισοί επεξεργαστές από τους οποίους αρχίσαμε. Δηλαδή οι επεξεργαστές με προσωπικό αριθμό 1<=*n/4* θα διαβάζουν τους αριθμούς της κοινόχρηστης μνήμης στην θέση με προσωπικό αριθμό διπλάσιο από το δικό τους και θα τους προσθέσουν με αυτούς που είναι στην θέση με διπλάσιο συν ένα όπως το πρώτο βήμα αποθηκεύοντας τους στη θέση με τον προσωπικό τους αριθμό. Η διαδικασία αυτή θα επαναλαμβάνεται και οι θα μένουν μισοί κάθε φορά μέχρι να φτάσουμε στο τελευταίο βήμα όπου θα έχουμε μόνο δύο αριθμούς.

Όταν φτάσουμε στο τελευταίο βήμα που έχουμε μόνο δύο αριθμούς ο επεξεργαστής με προσωπικό αριθμό ένα θα προσθέσει τους δύο αριθμούς στις θέσεις ένα και δύο της κοινόχρηστης μνήμης και θα αποθηκεύσει το τελικό αποτέλεσμα στην θέση ένα αυτής, το οποίο θα είναι η λύση του προβλήματος μας.

Όλα τα βήματα εκτελούνται συγχρονισμένα και παράλληλα από τους επεξεργαστές για τον λόγο αυτό αναμφισβήτητα χρησιμοποιείται μοντέλο PRAM για την ανάπτυξη του αλγορίθμου. Πιο συγκεκριμένα χρησιμοποιείται το μοντέλο EREW-PRAM. Αυτό γίνεται επειδή σε καμιά περίπτωση στην παράλληλη εκδοχή του αλγόριθμου που χρησιμοποιήσαμε δεν χρειάστηκε ταυτόχρονη εγγραφή ή ανάγνωση σε κάποιο βήμα. Συγκεκριμένα σε κάθε βήμα οι επεξεργαστές διαβάζουν τον αριθμό στην θέση με τον προσωπικό τους αριθμό επί δύο και τον προηγούμενο αριθμό από αυτόν έτσι είμαστε σίγουροι ότι δεν θα υπάρξουν δυο επεξεργαστές οι οποίοι θα προσπαθήσουν να διαβάσουν ή να γράψουν σε κοινή κυψελίδα της μνήμης. Για τον λόγο αυτό θα επιλέξουμε το EREW που είναι το πιο περιοριστικό μοντέλο που όπως αναφέραμε είναι αρκετά εύκολο από ένα περιοριστικό μοντέλο να το μετατρέψουμε σε ένα πιο δυνατό.

Γι’ αυτό επιλέγουμε πάντα το μοντέλο με τις πιο λίγες δυνατότητες το οποίο όμως μας ικανοποιεί.

Στο ακόλουθο κομμάτι κώδικα φαίνεται η διαδικασία η οποία περιγράφεται πιο πάνω για την παράλληλη άθροιση n αριθμών με *n/2* επεξεργαστές και ακολουθεί η πολυπλοκότητα της.

```
Processors i = 1 to n/2 do in parallel
        shared array sum [1…n]
        private j, a, b
        set j = 1
        while ( i ≤ n/2j )
                a = sum[2i - 1]
                b = sum[2i]
                sum[i]=a + b
                j = j+1
        end while
end do
```

Χρόνος εκτέλεσης αλγορίθμου: $T_{n/2}(n) = \Theta(log\ n) \cdot \Theta(1) = \Theta(log\ n)$
Κόστος αλγορίθμου: $C_{n/2}(n) = \Theta\ (log\ n) \cdot n/2 = \Theta(n\ log\ n)$

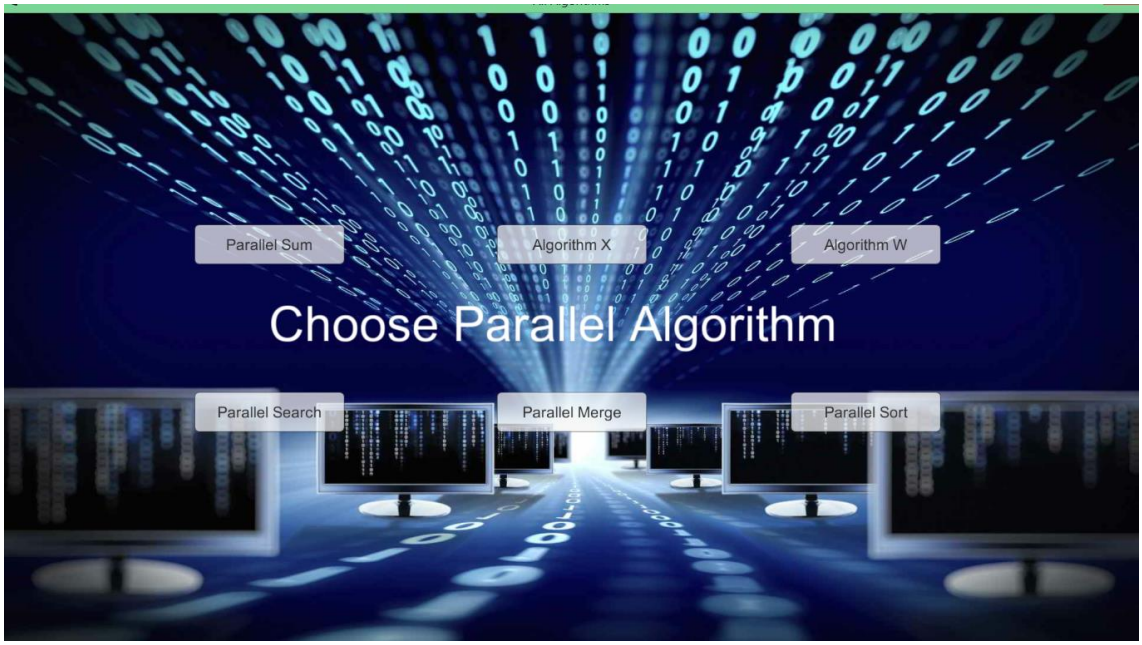
Ο χρόνος εκτέλεσης του αλγορίθμου είναι Θ(*logn*). Αυτό οφείλεται ότι σε κάθε βήμα οι επεξεργαστές χρειάζονται σταθερό χρόνο για την ανάγνωση των αριθμών, για τον υπολογισμό του αθροίσματος και για την αποθήκευση τους στην κοινόχρηστη μνήμη
Σε κάθε βήμα έχουμε το πολύ *n/2* επεξεργαστές που αυτό δεν παίζει κάποια σημασία αλλά αυτό που κρίνει τον χρόνο είναι ο αριθμός των βημάτων. Σε κάθε επόμενο βήμα ο αριθμός των αριθμών που μένουν για να προστεθούν είναι κάθε φορά ο μισός από το προηγούμενο. Έτσι για την ολοκλήρωση των βημάτων θα πρέπει να γίνουν logn επαναλήψεις ώστε να υπολογιστούν όλοι οι αριθμοί όπως φαίνεται και από τον βρόγχο στον πιο πάνω κώδικα. Έτσι ο συνολικός χρόνος του αλγορίθμου θα είναι ο αριθμός των βημάτων επί τον χρόνο εκτέλεσης του κάθε βήματος άρα Θ(*logn*) * Θ(*1*) = Θ(*logn*).

Το κόστος του αλγόριθμου δίνεται από το γινόμενο των επεξεργαστών επί τον χρόνο εκτέλεσης του αλγορίθμου , άρα $(n/2)^* \Theta(logn) = \Theta(nlogn)$.

Όπως είχαμε δει ο καλύτερος σειριακός αλγόριθμος άθροισης χρειαζόταν κόστος $\Theta(n)$ και όπως είχαμε εξηγήσει ο παράλληλος αλγόριθμος για να είναι βέλτιστος πρέπει το κόστος του να είναι ίσο με το κόστος του καλύτερου σειριακού αλγόριθμου. Άρα στην περίπτωση μας ο αλγόριθμος δεν είναι κόστους βέλτιστος αλλά θα μιλήσουμε για αυτόν στη συνέχεια του κεφαλαίου.

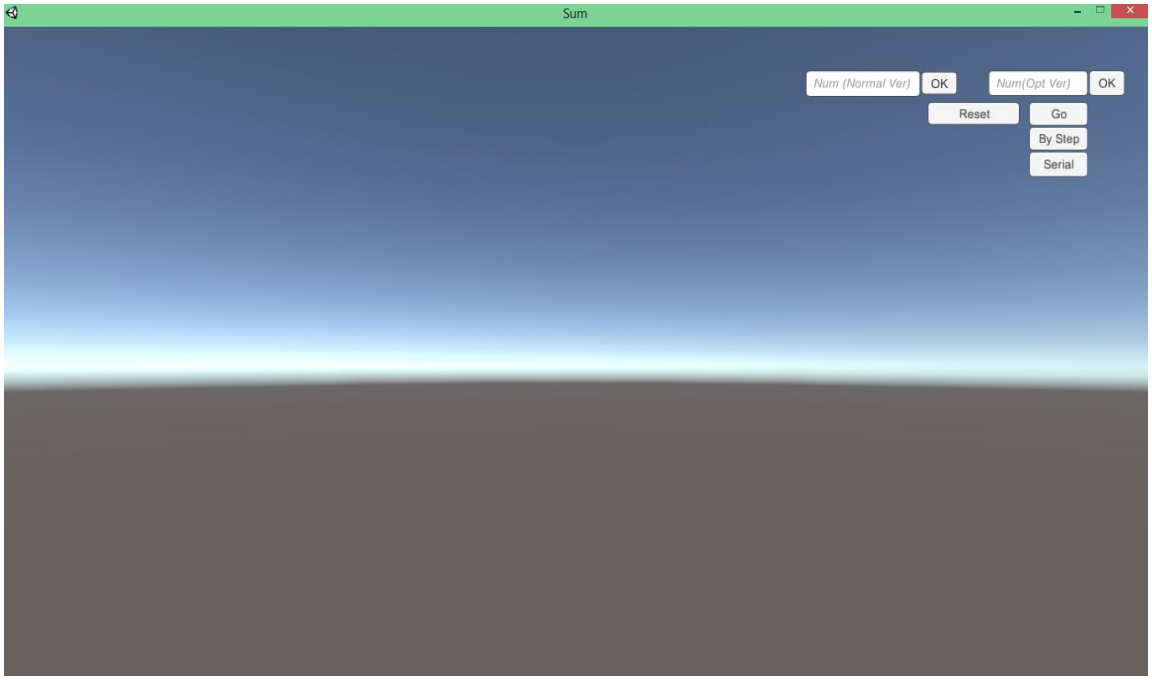
4.3.2 Γραφική Αναπαράσταση Αλγορίθμου

Στο σημείο αυτό θα δοθούν λεπτομέρειες για το πως υλοποιήθηκε ο μη βέλτιστος αλγόριθμος παράλληλης άθροισης στο Unity και βήμα προς βήμα πως τρέχει ο αλγόριθμος. Καταρχάς τρέχουμε το εκτελέσιμο. Η παρακάτω εικόνα(Σχήμα 4.1) είναι η πρώτη εικόνα που βλέπουμε μόλις τρέξουμε το εκτελέσιμο



Σχήμα 4.1: Πρώτη εικόνα εκτελέσιμου

Στη συνέχεια επιλέγουμε το πρώτο κουμπί του Parallel Sum όπου είναι ο αλγόριθμος ο οποίος μελετάμε στο συγκεκριμένο κεφάλαιο.



Σχήμα 4.2: Αρχική Εικόνα

Ακολούθως εμφανίζεται η πιο πάνω αρχική εικόνα(Σχήμα 4.2)

Σε οποιαδήποτε σημείο που τρέχει το πρόγραμμα πατώντας το κουμπί ESC πηγαίνουμε στην αρχική επιλογή αλγορίθμων και πατώντας το P γίνεται παύση.
Η αρχική εικόνα αποτελείται από:

- Δύο input fields τα όποια ο χρήστης περνά την τιμή των αριθμών που έχουμε να προσθέσουμε (το ένα για τη κανονική περίπτωση του αλγόριθμου και το άλλο για την βέλτιστη) με τα αντίστοιχα κουμπιά για να καταχωρηθεί ο αριθμός σε μια μεταβλητή
- Ένα κουμπί το κουμπί Reset το οποίο κάνει reset την σκηνή στην αρχική της μορφή για να εκτελέσει νέο παράδειγμα ο χρήστης
- Το κουμπί Go το οποίο τρέχει τον αλγόριθμο ολόκληρο με μια μικρή καθυστέρηση μεταξύ των βημάτων για να διακρίνονται αυτά.
- Το κουμπί By Step με το οποίο ο αλγόριθμος προχωρά στο επόμενο ακριβώς βήμα
- Το κουμπί Serial με το οποίο εκτελεί ένα βήμα του αλγόριθμου αν είχαμε μόνο ένα επεξεργαστή.
- Το canvas που μέσα σε αυτό ήταν όλα τα πιο πάνω.

Τα κουμπιά είναι ανεξάρτητα το ένα με το άλλο δηλαδή αν πρώτα εκτελέσουμε ένα βήμα με το κουμπί By Step πατώντας ακολούθως το Go θα ολοκληρωθεί ο αλγόριθμος από εκεί όπου είχε μείνει.

Για την δημιουργία του αλγόριθμου αυτού χρησιμοποιήθηκε ένα prefab το οποίο είχε μέσα ένα ακόμα αντικείμενο το οποίο αποτελούταν από ένα text mesh όπου στη συνέχεια σε αυτό το text mesh αποθηκεύεται και εμφανίζεται ο αριθμός.

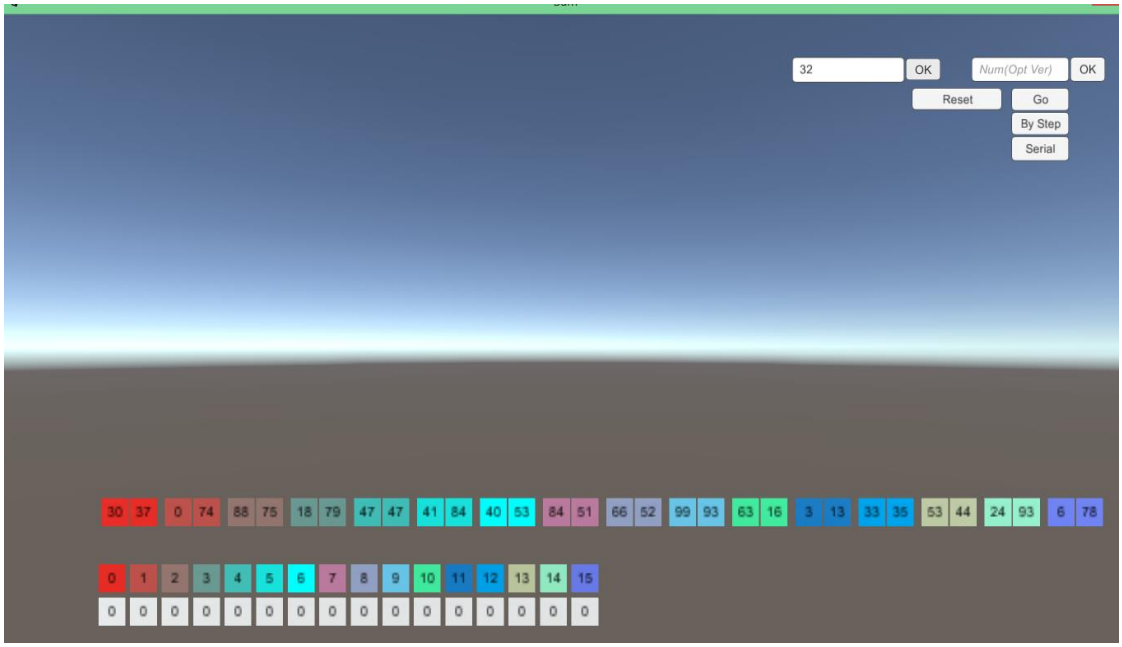
Για την δημιουργία του κώδικα χρησιμοποιήθηκε μόνο ένα script το οποίο ήταν ενσωματωμένο στο canvas και το κάθε κουμπί καλούσε την αντίστοιχη συνάρτηση με αυτή που έπρεπε να εκτελέσει.

Αν ο χρήστης προσπαθήσει να πατήσει ένα από τα κουμπιά χωρίς να έχει πρώτα εισάγει τον αριθμό και πατήσει ok το πρόγραμμα δεν κάνει κάτι μέχρι να εισαχθεί σωστά ο αριθμός. Σε αρχικά στάδια το πρόγραμμα λειτουργούσε εσφαλμένα και για τον λόγο αυτό αντιμετωπίσθηκε έτσι για καλύτερη αλληλεπίδραση με τον χρήστη.

Το μέγεθος της κάμερας για μεγαλύτερο αριθμό *n* μικραίνει και μεγαλώνει αντίστοιχα ώστε να υποστηρίζει μέχρι 64 αριθμούς και οι αριθμοί αυτοί να παραμένουν διακριτοί στο μάτι.
Στη συνέχεια θα αναλυθεί ένα παράδειγμα με 32 αριθμούς και 16 επεξεργαστές.

Ο χρήστης γράφει τον αριθμό 32 στην περίπτωση μας στο πρώτο input field (normal version) και ακολούθως πατά το Ok button. Στη συνέχεια παρατηρεί να εμφανίζεται στην οθόνη του η ακόλουθη εικόνα. (Σχήμα 4.3)

Όπως παρατηρείται στο κάτω μέρος εμφανίζονται 16 κουτιά αριθμημένα από το 0 μέχρι το 15 που είναι ο αριθμός n που έχουμε βάλει δια 2 δηλαδή το 32 που μας δίνει τον αριθμό των επεξεργαστών. Κάθε ένα κουτί αντιστοιχεί με ένα επεξεργαστή και του έχει ανατεθεί ένα χρώμα το οποίο θα εμφανίζεται κατά όλη την διάρκεια του αλγορίθμου χρωματίζοντας όποια πράξη εκτελέσει ο επεξεργαστής αυτός.



Σχήμα 4.3: Αρχικό Βήμα με 32 αριθμούς

Το κάθε χρώμα αντικατοπτρίζεται από ένα βαθμό κόκκινου, ένα βαθμό μπλε και ένα βαθμό πράσινου που παίρνουν τιμή από 0-255. Στην κονσόλα μας οι αριθμοί για την δημιουργία ενός χρώματος ήταν από το 0 μέχρι το 1 γι' αυτό δημιουργήθηκαν τα όρια αυτά. Για την δημιουργία των χρωμάτων αρχικοποιήθηκαν τρείς μεταβλητές με κάποιες τιμές. Για κάθε 2 αριθμούς οι οποίοι δημιουργούνταν, μια τιμή προθετόταν σε αυτές τις μεταβλητές και έτσι άλλαζε το χρώμα του επόμενου επεξεργαστή. Τα χρώματα αυτά διατηρούνταν σε ένα πίνακα ανάλογα με την αντίστοιχη θέση του επεξεργαστή.

```
if ( i % 2 == 0) {//new color to each processor
        if (col1 < 0)
                col1 = 1;
        col1 = col1 - 0.20f;
        if (col2 > 1)
                col2 = 0.30f;
        col2 = col2 + 0.18f;
        if (col3 > 1)
                col3 = 0.47f;
        col3 = col3 + 0.18f;
        Color newColor = new Color (col1, col2, col3, 1);
        colortable [colorcount] = newColor;
        colorcount++;}
```

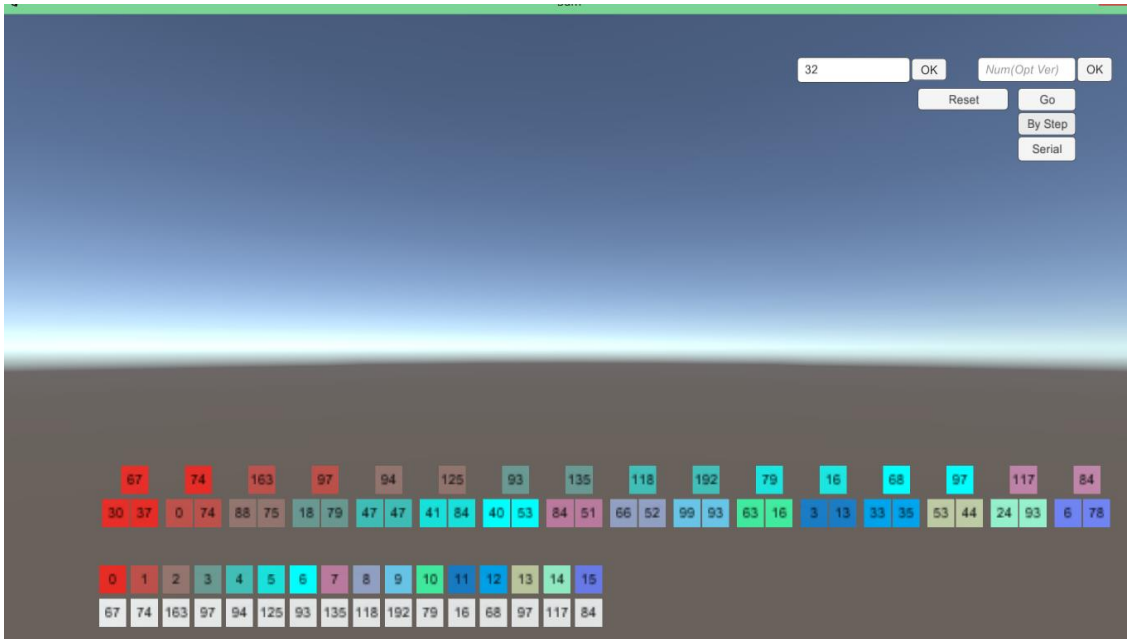
Κάτω από τον κάθε επεξεργαστή είναι ο αριθμός ο οποίος έχει αθροίσει στο βήμα αυτό που για το πρώτο βήμα σε όλους είναι το μηδέν αφού ακόμα δεν έχουν κάνει κάποια πράξη

Πάνω από τους επεξεργαστές μπορούμε να διακρίνουμε 32 κουτιά με τυχαίους αριθμούς όσο δηλαδή το n το οποίο έχουμε εισάγει στην αρχή χρωματισμένα ανά δύο με ένα μικρό κενό ενδιάμεσα τους. Αυτό γίνεται γιατί ο κάθε επεξεργαστής διαβάζει την τιμή που βρίσκεται στον προσωπικό του αριθμό επί 2 και την επόμενη. Πιο πάνω έχει αναφερθεί ότι διαβάζουμε την τιμή του προσωπικού αριθμού επί 2 και την προηγούμενη από αυτή αλλά αυτό γινόταν στην περίπτωση που η αρίθμηση των επεξεργαστών γινόταν από το 1. Στην περίπτωση της υλοποίησης μας τους αριθμήσαμε από το 0 .

Η δημιουργία όλων των κουτιών έγινε δυναμικά με την εντολή instantiate δημιουργώντας το prefab που είχαμε αναφέρει πιο πάνω. Για την επιλογή της τοποθεσίας που θα εμφανίζονταν χρησιμοποιηθήκαν μια σταθερή τιμή η οποία αυξανόταν σε κάθε βήμα για να προχωρά στον άξονα y και η τιμή του βρόγχου για να προχωρά στον άξονα x
Παράδειγμα κώδικα για την δημιουργία ενός κουτιού.

```
int cont = 0;
Vector3 position = new Vector3 ((cont * stepSize) - 10.6f + temp, (steps * stepSize), 0);
vectors [cont] = (cont * stepSize - 10 + temp);
GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
table [cont] = 0;
go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table [cont];
go.GetComponent <MeshRenderer> ().material.color = colortable [colorcount - 1];
cont++;
```

Στη συνέχεια πατώντας μια φορά το By Step βλέπουμε την ακόλουθη εικόνα όπου είναι ακριβώς το επόμενο βήμα. (Σχήμα 4.4)

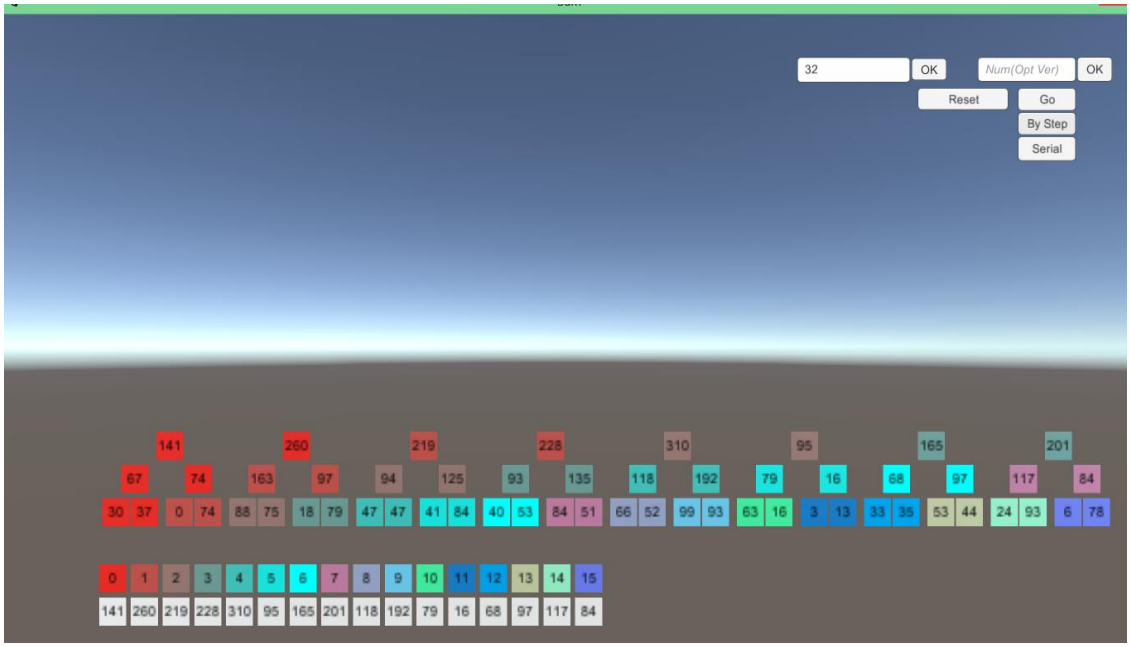


Σχήμα 4.4: Βήμα 1

Στο βήμα αυτό βλέπουμε πως οι επεξεργαστές στο κάτω μέρος έχουν κάνει όλοι από μια πρόσθεση και έχουν υπολογίσει ένα άθροισμα. Αυτό το άθροισμα αναγράφεται κάτω από κάθε επεξεργαστή στο κάτω μέρος και πάνω από τις δυάδες στο πάνω μέρος.
Μετά από τον υπολογισμό του νέου αθροίσματος και η καταγραφή στις αντίστοιχες θέσεις στην μνήμη μας με τον προσωπικό τους αριθμό (στην περίπτωση μας 0-15) , οι επεξεργαστές ανατίθενται ξανά σε δυάδες όπου θα υπολογίσουν νέο άθροισμα. Για τον υπολογισμό του επόμενου βήματος έχουμε 16 αριθμούς όπου τους αναλαμβάνουν οι πρώτοι 8 επεξεργαστές (0-7 προσωπικό αριθμό) και οι υπόλοιποι 8 (8-15 προσωπικό αριθμό) εκτελούν πράξη no-op.

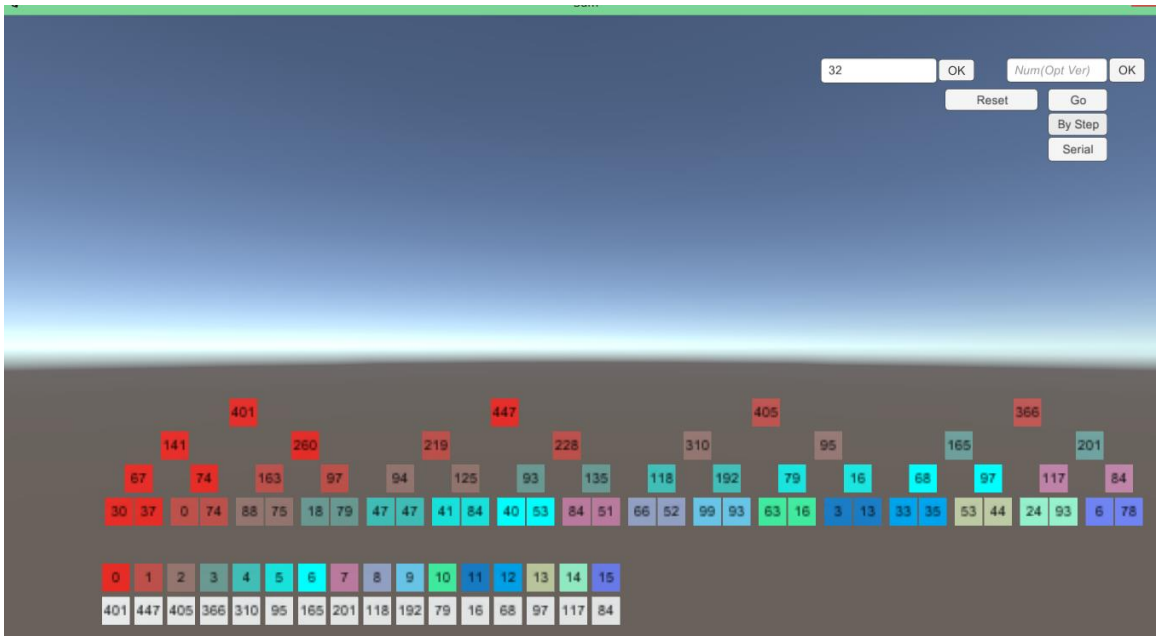
Έτσι πατώντας ξανά το By Step προχωράμε στο επόμενο βήμα.(Σχήμα 4.5)

Στο βήμα αυτό οι αριθμοί έχουν υποδιπλασιαστεί όπως και σε κάθε βήμα. Στο συγκεκριμένο βήμα οι αριθμοί έχουν γίνει 8 και όπως βλέπουμε στο κάτω μέρος οι αριθμοί των επεξεργαστών με προσωπικό αριθμό 8 μέχρι 15 έχουν παραμείνει οι ίδιοι γιατί δεν εκτέλεσαν κάποια λειτουργία. Μετά τον υπολογισμό των 8 αυτών αριθμών και την δημιουργία των νέων κουτιών αυτά χρωματίστηκαν ανάλογα με τους επεξεργαστές που θα τους αναλάβουν στο επόμενο βήμα οι πρώτοι 4 επεξεργαστές δηλαδή.



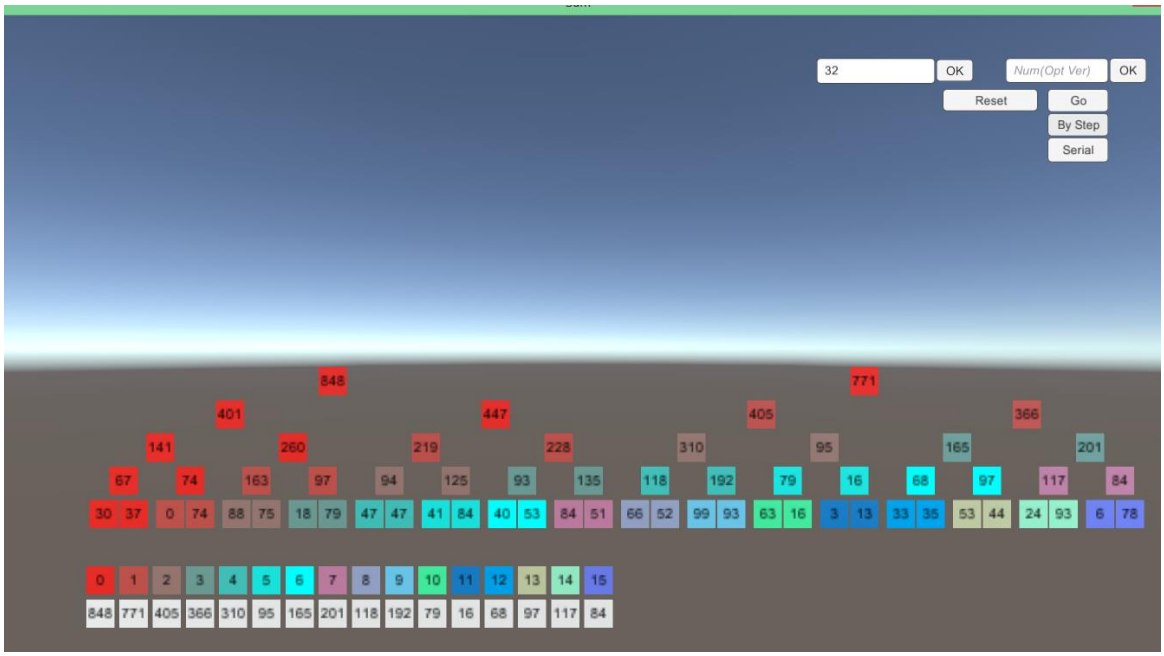
Σχήμα 4.5: Βήμα 2

Στο βήμα 3 (Σχήμα 4.6) οι αριθμοί έχουν υποδιπλασιαστεί ξανά. Στο συγκεκριμένο βήμα οι αριθμοί έχουν γίνει 4 και στο κάτω μέρος οι αριθμοί των επεξεργαστών με προσωπικό αριθμό 4 μέχρι 15 έχουν παραμείνει οι ίδιοι γιατί δεν εκτέλεσαν κάποια λειτουργία. Έτσι στο παρών βήμα παραμένουν 4 αριθμοί οι οποίοι θα γίνουν 2 αθροίσματα από τους πρώτους 2 επεξεργαστές



Σχήμα 4.6: Βήμα 3

Στο Βήμα 4 (Σχήμα 4.7) έχουν υπολογιστεί 2 αθροίσματα από τους 2 πρώτους επεξεργαστές και οι υπόλοιποι 14 επεξεργαστές παραμένουν αδρανείς. Έτσι στο τελικό βήμα μένουν οι 2 αριθμοί που δοθήκαν από τα 2 υπό-δένδρα του τελικού που θα δώσουν το τελικό αποτέλεσμα



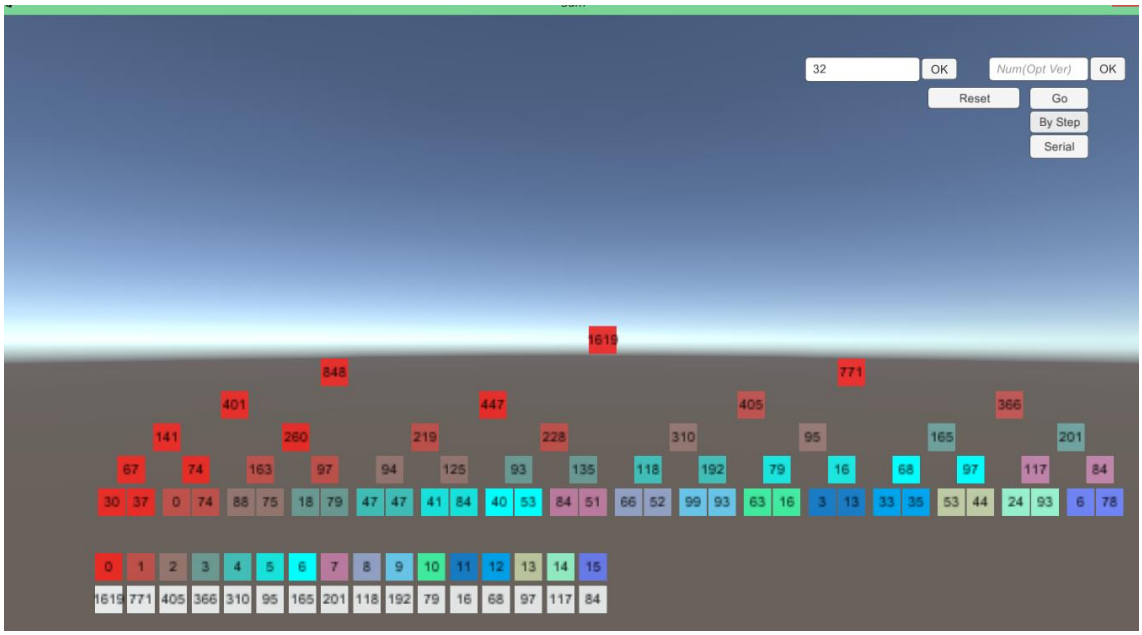
Σχήμα 4.7: Βήμα 4

Στο τελικό βήμα (Σχήμα 4.8) ο επεξεργαστής με προσωπικό αριθμό 0 παίρνει τους αριθμούς στις θέσεις 0 και 1 της κοινόχρηστης μνήμης και υπολογίζει το τελικό αποτέλεσμα και το αναγράφει στην θέση 0 της κεντρικής μνήμης.

Το κομμάτι κώδικα που δημιουργεί ένα κουτί σε ένα βήμα, υπολογίζει το άθροισμα και του αναθέτει χρώμα δίνεται πιο κάτω

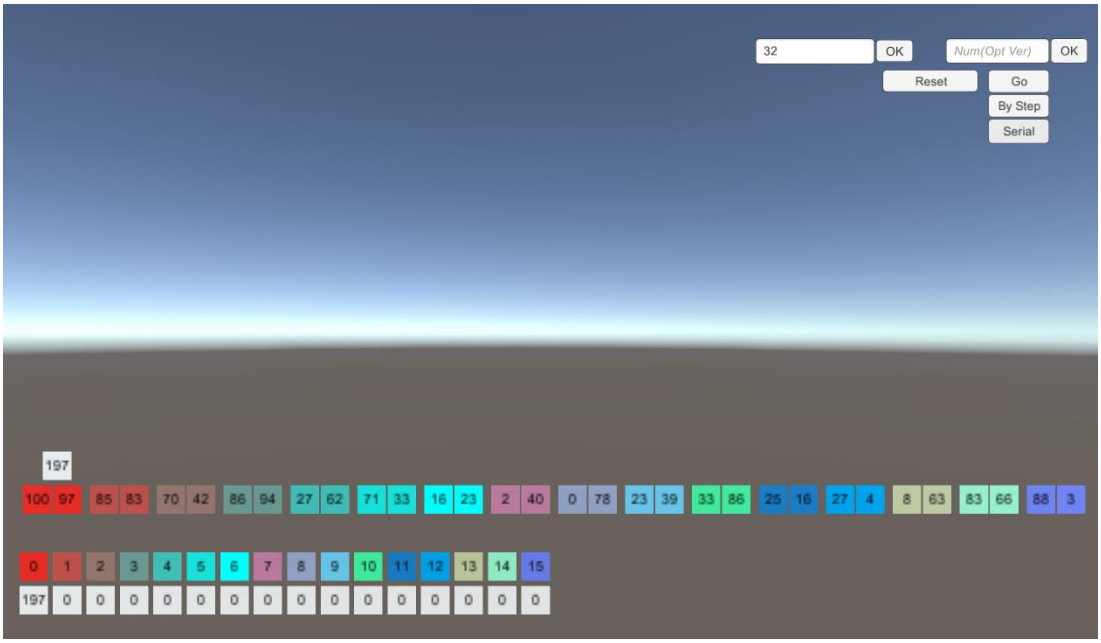
```
Vector3 position = new Vector3 ((vectors [i]), (steps * stepSize), 0);
int a = table [2 * i];
int b = table [2 * i + 1];
table [i] = a + b;
if (i % 2 == 0)
        colorcount++;
StartCoroutine (waitonesecond (position, table [i], se, colortable [colorcount-1]));
StartCoroutine (waittransf (tableobject [count], table [i], se));
```

*Με την εντολή StartCoroutine δημιουργούμε μια καθυστέρηση για να γίνεται πιο αντιληπτό στα μάτια του χρήστη πως λειτουργά ο αλγόριθμος.



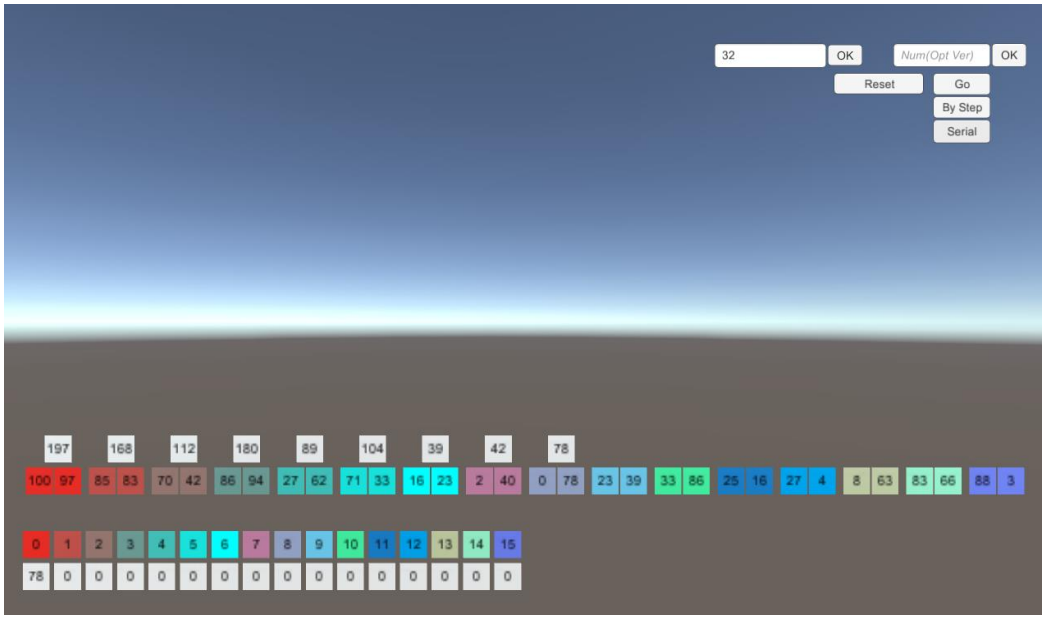
Σχήμα 4.8: Τελικό Βήμα Και Αποτέλεσμα

Στη συνέχεια θα δούμε το κουμπί serial(Σχήμα 4.9) πως λειτουργεί χρησιμοποιώντας ξανά 32 αριθμούς. Το αρχικό βήμα είναι το ίδιο δηλαδή βάζουμε τον αριθμό στο input field και πατάμε OK. Στη συνέχεια πατώντας το serial βλέπουμε πως θα λειτουργούσε αν ήταν μόνο ο επεξεργαστής στην θέση 0. Ο οποίος απλά υπολογίζει το άθροισμα των πρώτον 2 αριθμών και το αναγράφει.



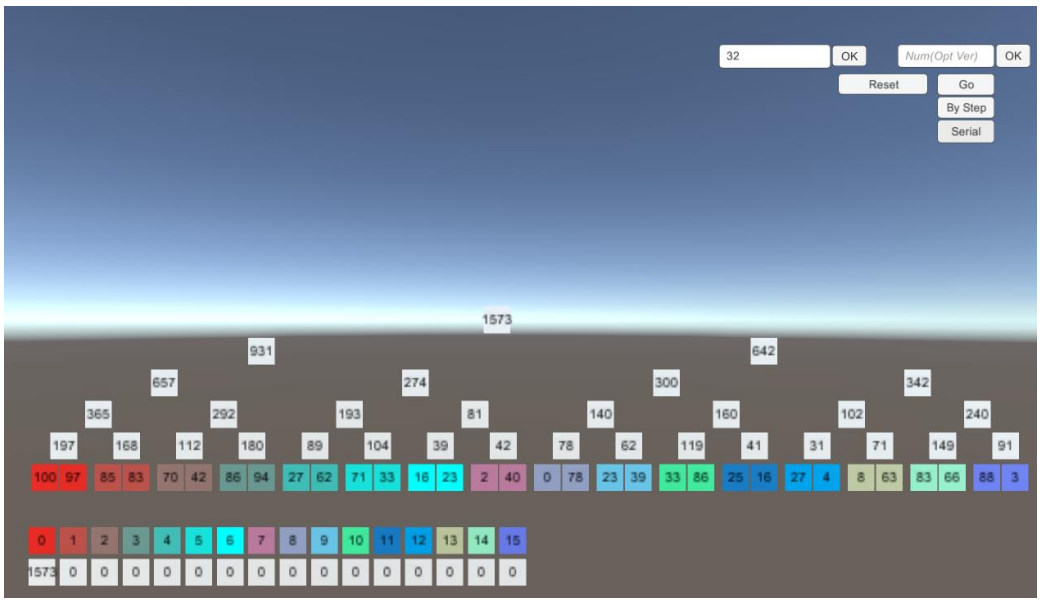
Σχήμα 4.9: Σειριακός αλγόριθμος βήμα 1

Στη συνέχεια για σκοπούς ευχρηστίας δίνεται το παράδειγμα μετά από 8 βήματα όπου έχουν υπολογιστεί 9 αριθμοί και παρατηρείται ότι μόνο ο επεξεργαστής 0 έχει υπολογίσει κάτι στο κάτω μέρος.



Σχήμα 4.10: Σειριακός αλγόριθμος βήμα 9

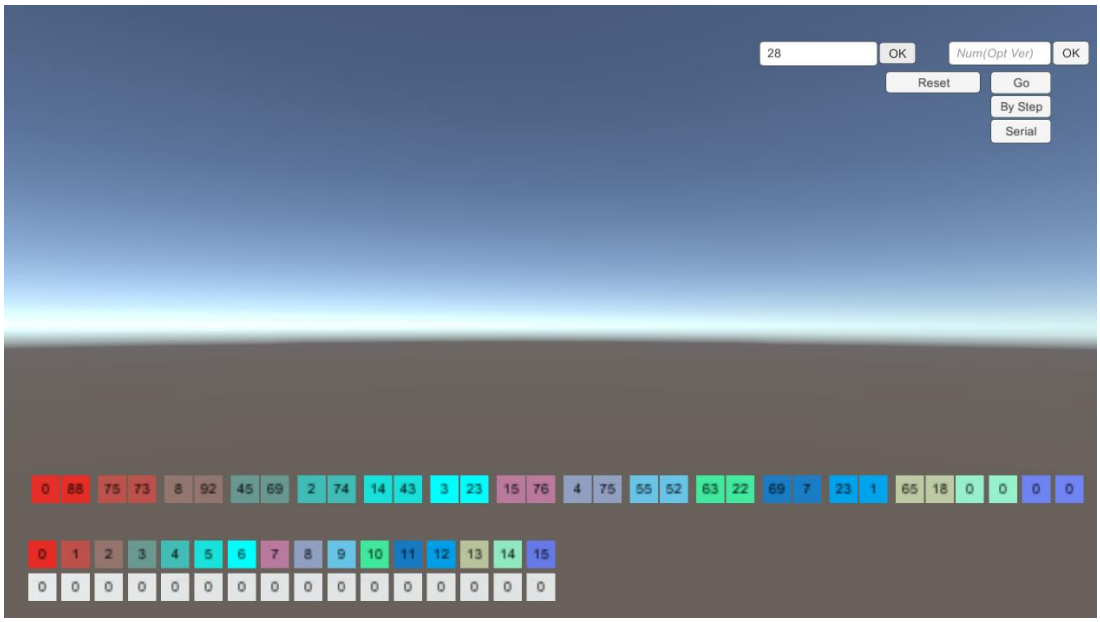
Τέλος δίνεται το τελικό αποτέλεσμα αν πατήσεις το κουμπί serial αρκετές φορές ώστε ο επεξεργαστής 0 να ολοκληρώσει τον αλγόριθμο.



Σχήμα 4.11: Σειριακός αλγόριθμος τελικό αποτέλεσμα

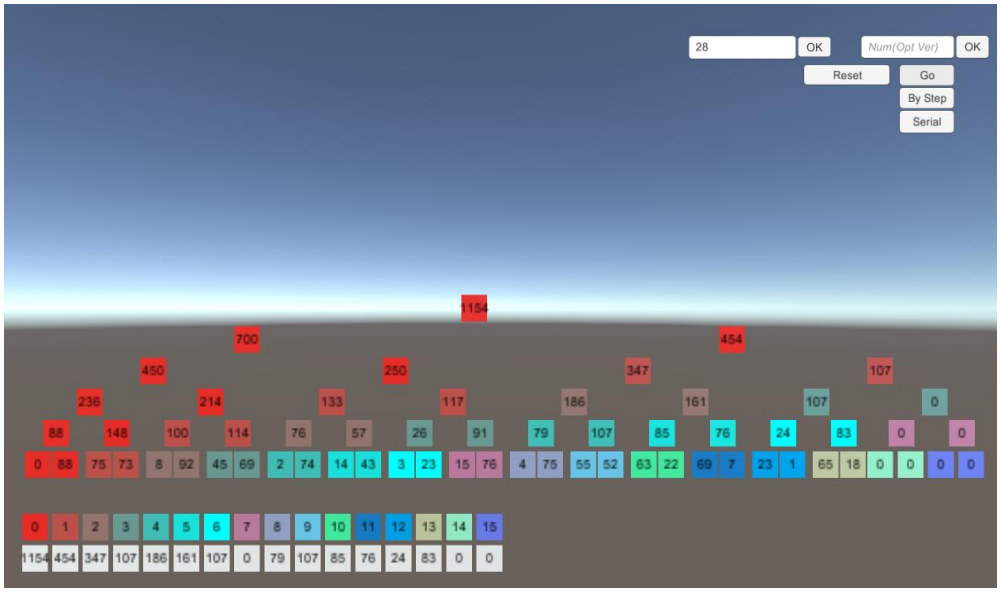
Το κουμπί Go μπορεί να συμπληρώσει τον αλγόριθμό σε οποιοδήποτε στάδιο μέχρι την ολοκλήρωση του.

Στη συνέχεια θα αναφερθούμε στο **Patching.** Patching είναι η συμπλήρωση που γίνεται όταν ο αριθμός δεν είναι πολλαπλάσιο του δύο. Ο αλγόριθμος αυτός δημιουργήθηκε με τρόπο ώστε να συμπληρώνει με μηδενικά μέχρι την πλησιέστερη δύναμη του δύο.
Στη συνέχεια θα δούμε ένα παράδειγμα όπου ο αριθμός που δίνεται είναι το 28.



Σχήμα 4.12: Patching Αρχικό Βήμα

Παρατηρούμε στο Σχήμα 4.12 πως οι πρώτοι 28 αριθμοί έχουν συμπληρωθεί με τυχαίους αριθμούς και οι τελευταίοι 4 με μηδενικά όπως αναφέρθηκε πιο πάνω. Στη συνέχεια ο αλγόριθμος τρέχει κανονικά όπως εξηγήσαμε και πατώντας το κουμπί Go παίρνουμε το τελικό αποτέλεσμα (Σχήμα 4.13) με μια μικρή καθυστέρηση ενδιάμεσα των βημάτων για να ξεχωρίζουν.



Σχήμα 4.13: Τελικό Αποτέλεσμα Με Patching

4.4 Βέλτιστος Αλγόριθμος Παράλληλης Άθροισης

Ακολούθως γίνεται περιγραφή του βέλτιστου αλγόριθμου παράλληλης άθροισης όπου επιλύει το ίδιο πρόβλημα με τον αλγόριθμο στο Κεφάλαιο 4.3 αλλά με το καλύτερο δυνατό κόστος. Επίσης δίνονται λεπτομέρειες για την γραφική αναπαράσταση που υλοποιήθηκε στην εργασία.

4.4.1 Περιγραφή Αλγορίθμου

Όπως αναφέρθηκε σε προηγούμενα στάδια ο αλγόριθμος που παρουσιάστηκε πριν δεν ήταν ο βέλτιστος αλγόριθμος για τον λόγο ότι το κόστος του ήταν μεγαλύτερο από το κόστος του βέλτιστου σειριακού. Το κόστος του προηγούμενου αλγόριθμου ήταν $\Theta(nlogn)$ και του καλύτερου σειριακού $\Theta(n)$. Για την μείωση του κόστους σε $\Theta(n)$ πρέπει οι επεξεργαστές να μειωθούν γιατί όπως φάνηκε στον προηγούμενο αλγόριθμο σε πάρα πολλά βήματα οι επεξεργαστές μας δεν εκτελούσαν κάποια λειτουργία. Έτσι στον αλγόριθμο αυτό θα χρησιμοποιηθούν n/logn επεξεργαστές.

Ο αλγόριθμος αυτός για να λειτουργήσει στον ίδιο χρόνο με τον προηγούμενο με λιγότερους επεξεργαστές θα αποτελείται από δύο φάσεις. Στην πρώτη φάση ο κάθε επεξεργαστής θα αναλάβει ένα εκτελέσει ένα σειριακό άθροισμα σε ένα σύνολο της κοινόχρηστης μνήμης και στην δεύτερη φάση θα εκτελεστεί με τα αποτελέσματα της πρώτης φάσης ο αλγόριθμος παράλληλης άθροισης όπως περιγράφηκε πιο πάνω.

Συγκεκριμένα στην πρώτη φάση ο κάθε επεξεργαστής θα αναλάβει από logn αριθμούς τους οποίους θα προσθέσει σειριακά. Ο πρώτος επεξεργαστής θα αναλάβει τους πρώτους logn αριθμούς ο δεύτερος τους επομένους και αυτό θα συνεχιστεί μέχρι τον n/logn επεξεργαστή, που αυτό θα μας δώσει *n/logn* αθροίσματα.

Η δεύτερη φάση θα είναι ακριβώς η ίδια με τον αλγόριθμο παράλληλης άθροισης αλλά με n/logn αριθμούς τώρα και θα χρησιμοποιηθούν όπως περιγράφηκε οι μισοί τους επεξεργαστές δηλαδή (*n/logn*)/2.

Στη πρώτη φάση δεν υπάρχει ταυτόχρονο διάβασμα σε ίδια κυψελίδα της μνήμης η γραφή αφού ο κάθε επεξεργαστής αναλαμβάνει n/logn διαφορετικούς αριθμούς. Η φάση δύο παραμένει η ίδια με πριν. Άρα δεν θα χρειαστούμε πιο δυνατό μοντέλο PRAM και θα χρησιμοποιηθεί όπως πριν μοντέλο EREW-PRAM

Ο χρόνος εκτέλεσης της πρώτης φάσης είναι Θ(*logn*) όπου χρειάζεται ο κάθε επεξεργαστής για τον υπολογισμό του σειριακού αθροίσματος logn αριθμών. Η δεύτερη φάση χρειάζεται Θ(*log(n/logn*)). Αυτό οφείλεται γιατί ο αλγόριθμος της δεύτερης φάσης για n στοιχεία χρειάζεται Θ(*logn*) έτσι αντίστοιχα για *n/logn* στοιχεία θα χρειαστεί Θ(*log(n/logn)).*

Έτσι ο συνολικός χρόνος εκτέλεσης θα είναι αυτός της πρώτης φάσης αφού είναι μεγαλύτερος από αυτό της δεύτερης. Το κόστος θα είναι ο χρόνος αυτός επί τους επεξεργαστές που χρησιμοποιήθηκαν. Δηλαδή Θ(*logn*) * *n/logn* = Θ(*n*).

Έτσι παρατηρούμε ότι ο χρόνος παραμένει ο ίδιος με τον αρχικό αλγόριθμο αλλά το κόστος έχει μειωθεί όσο του καλύτερου σειριακού αλγόριθμου. Έτσι ο αλγόριθμος μας είναι βέλτιστος.

$T(n) = \Theta(\log(n/\log n))$
$C(n) = \Theta(n)$

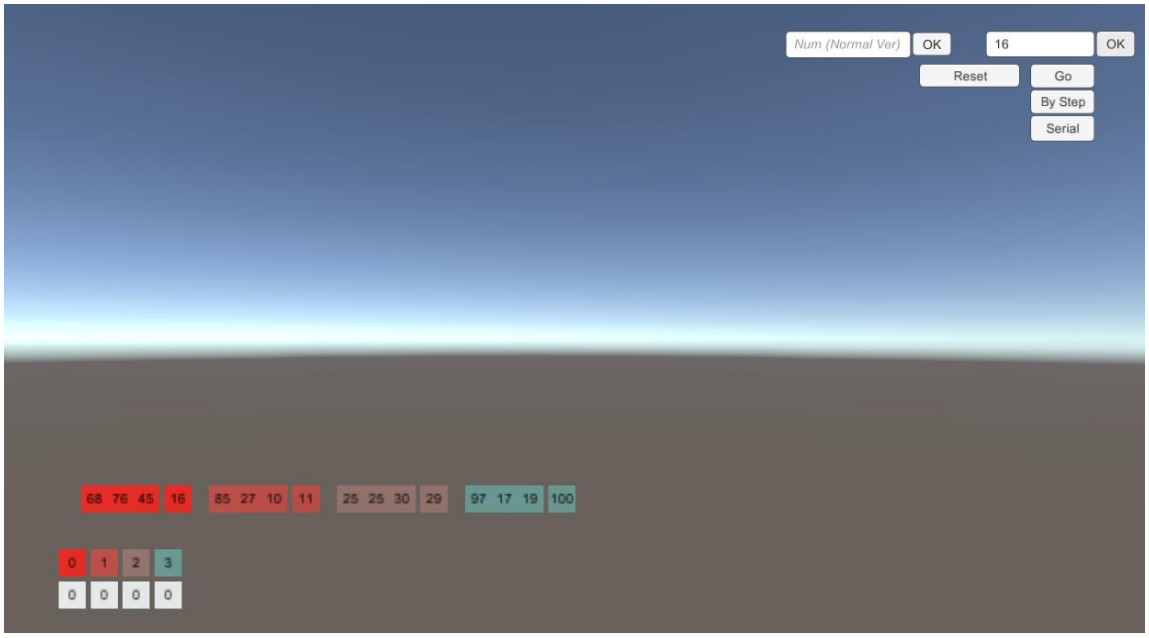
4.4.2 Γραφική Αναπαράσταση Αλγορίθμου

Στο σημείο αυτό θα δοθούν λεπτομέρειες για το πώς υλοποιήθηκε ο βέλτιστος αλγόριθμος παράλληλης άθροισης και βήμα προς βήμα πως τρέχει ο αλγόριθμος.
Το εκτελέσιμο το οποίο χρησιμοποιήθηκε είναι το ίδιο με του αρχικού αλγόριθμου παράλληλης άθροισης αλλά τώρα θα χρησιμοποιηθεί το δεξιά input box.
Ο αλγόριθμος αυτός όπως αναφέρθηκε αποτελείται από 2 φάσεις.

Η υλοποίηση υποστηρίζει μέχρι τον αριθμό 32. Τα χρώματα των επεξεργαστών και ο τρόπος με τον οποίο τοποθετήθηκαν τα κουτιά είναι με τον ίδιο τρόπο και χρησιμοποιώντας το ίδιο prefab με το αρχικό μας παράδειγμα. Επίσης τα κουμπιά τα

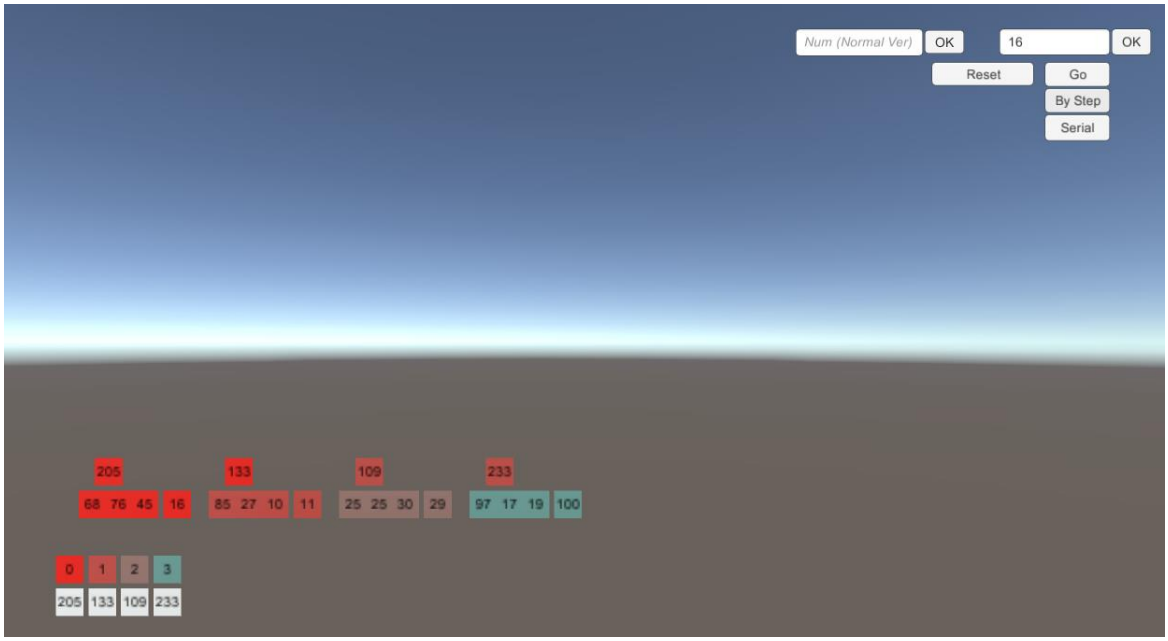
οποία υλοποιήθηκαν για τον αρχικό αλγόριθμο (Go, By step) λειτουργούν με τον ίδιο τρόπο για τον βέλτιστο αλγόριθμο με την προσθήκη μιας μεταβλητής που ενεργοποιεί διαφορετικά κομμάτια κώδικα αν ο αριθμός που έχει δοθεί είναι από το αριστερά input box και διαφορετικά κομμάτια κώδικα αν είναι από το δεξιά.
Στη συνέχεια δίνεται ένα παράδειγμα με τον αριθμό 16.

Όπως φαίνεται στο Σχήμα 4.14 οι αριθμοί χωρίστηκαν διαφορετικά σε σχέση με τον αρχικό αλγόριθμο. Μπορούμε να παρατηρήσουμε στο κάτω μέρος πως τώρα έχουμε μόνο n/logn επεξεργαστές, όπου είναι 4 στην περίπτωση μας. Ο κάθε επεξεργαστής ανέλαβε logn αριθμούς δηλαδή στο παράδειγμα αυτό έχει αναλάβει 4 αριθμούς. Έτσι πατώντας το κουμπί By Step θα δούμε πώς ο κάθε επεξεργαστής θα βρει άθροισμα για τους αριθμούς που έχει αναλάβει



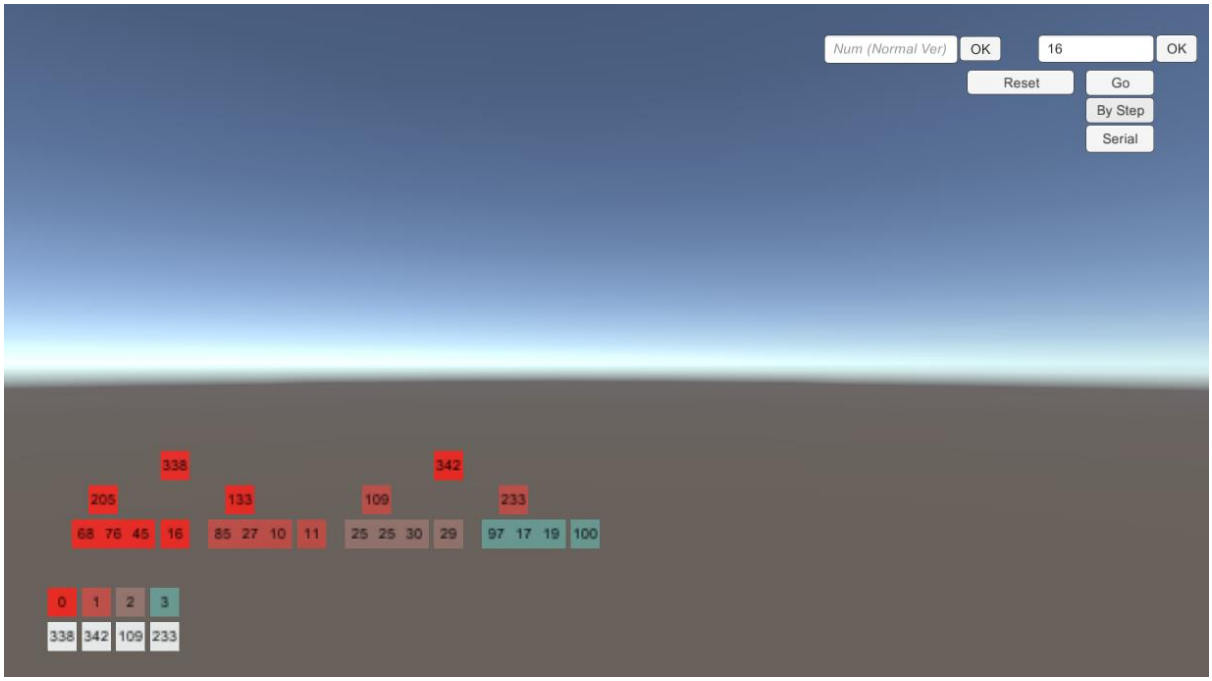
Σχήμα 4.14: Αρχική εικόνα βέλτιστου αλγόριθμου με n=16

Με αυτό τον τρόπο έχουν υπολογιστεί 4(n/logn) αθροίσματα τα οποία έχουν πάρει την μορφή κουτιού και έχουν αναγραφεί στις πρώτες 4 θέσεις της κοινόχρηστης μνήμης(Σχήμα 4.15). Έτσι οι επεξεργαστές που θα χρησιμοποιηθούν στο επόμενο βήμα είναι μόνο 2 ((n/logn)/2) και θα συνεχίσουν την εκτέλεση όπως τον αρχικό αλγόριθμο δηλαδή χρωματίζοντας ανά δύο και εκτελώντας τα αθροίσματα που είναι η φάση 2 του αλγόριθμου

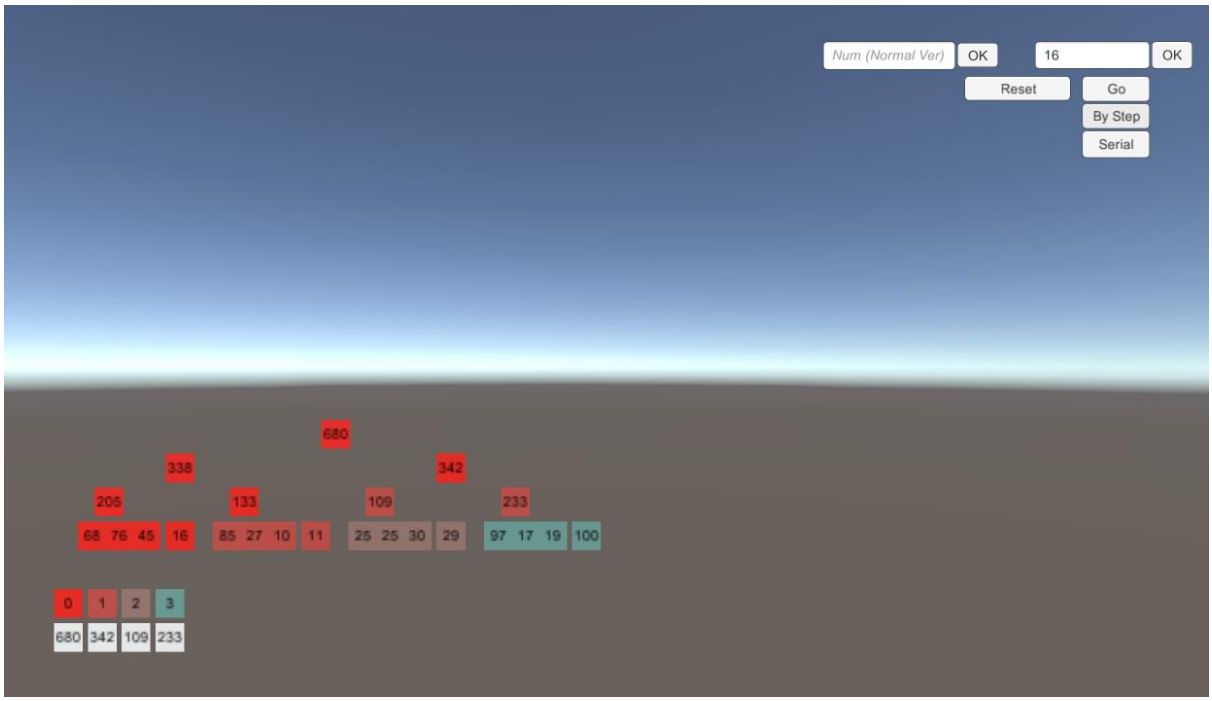


Σχήμα 4.15: Φάση 1 Βέλτιστου

Στη συνέχεια φαίνονται τα βήματα για την ολοκλήρωση της φάσης 2. Στο Σχήμα 4.16 γίνεται το άθροισμα για τους 4 αριθμούς που έχουν μείνει από τους 2 πρώτους επεξεργαστές. Στο Σχήμα 4.17 παρατηρείται το τελικό βήμα όπου ο επεξεργαστής με προσωπικό αριθμό 0 κάνει το άθροισμα των 2 πρώτον αριθμών στην κοινόχρηστη μνήμη και το αποθηκεύει στην πρώτη θέση, που είναι και το τελικό μας αποτέλεσμα.



Σχήμα 4.16: Φάση 2 Βέλτιστου – Βήμα 1



Σχήμα 4.17: Φάση 2 Βέλτιστου – Τελικό Βήμα

Έτσι όπως παρατηρείται στο τελικό βήμα το αποτέλεσμα της άθροισης είναι το 680 που προσθέτοντας τους πρώτους 16 αριθμούς που δόθηκαν είναι σωστό.

Κεφάλαιο 5

Αλγόριθμος Παράλληλης Αναζήτησης

5.1 Πρόβλημα

Το πρόβλημα της παράλληλης αναζήτησης είναι το εξής : *Δεδομένου μιας ταξινομημένης λίστας από n αριθμούς* $X_1,X_2,..,X_n$ *και ενός αριθμού y πρέπει να βρεθεί ο δείκτης* k *ο οποίος* $X_k <= y < X_{k+1}$ *(*$X_0 = -\infty$, $X_{n+1} = +\infty$*). [2,5]*

5.2 Σειριακός Αλγόριθμος

Στην περίπτωση του σειριακού υπολογισμού, όπου έχουμε δηλαδή μόνο ένα επεξεργαστή αυτός ο επεξεργαστής αρχικά θα συγκρίνει το μεσαίο στοιχείο της λίστας με το y:

- Αν είναι ίσο επιστρέφει την θέση του στοιχείου αυτού
  if (X_μ == *y*) Return *y*
- Αν είναι μικρότερο (X_μ>*y*) τότε η διαδικασία επαναλαμβάνεται στο μικρότερο μέρος της λίστας.
- Αν είναι μεγαλύτερο (X_μ<*y*) τότε η διαδικασία επαναλαμβάνεται στο μεγαλύτερο μέρος της λίστας.
- Η διαδικασία τερματίζει αν βρεθεί το στοιχείο σε κάποια στιγμή η αν το μέγεθος της λίστας φτάσει στο 1 και δεν έχουμε βρει το στοιχείο που ψάχναμε.

Πολυπλοκότητα: $T_1(n) = C_1(n) = O(log\ n)$

Ο χρόνος εκτέλεσης του αλγόριθμου αυτού είναι O(*log n*). Αυτό συμβαίνει γιατί ο αλγόριθμος σε κάθε βήμα μειώνει τα στοιχεία στα μισά. Ο χρόνος για το διάβασμα του στοιχείου και την σύγκριση με αυτό χρειάζεται σταθερό χρόνο δηλαδή Θ(*1*). Έτσι αφού τα στοιχεία υποδιπλασιάζονται κάθε φορά μέχρι να βρεθεί το στοιχείο που ψάχνουμε θα χρειαστούμε από μία μέχρι logn επαναλήψεις. Έτσι ο χρόνος εκτέλεσης του σειριακού αλγόριθμου άθροισης αποδεικνύεται πως είναι O(*logn*) , όπως και το κόστος του αφού χρησιμοποιούμε μόνο ένα επεξεργαστή .

5.3 Παράλληλος Αλγόριθμος

Ακολούθως γίνεται περιγραφή του αλγόριθμου παράλληλης αναζήτησης και λεπτομέρειες για την γραφική αναπαράσταση που υλοποιήθηκε στην εργασία.

5.3.1 Περιγραφή Αλγορίθμου

Όπως και στον σειριακό μας αλγόριθμο έτσι και στον παράλληλο αλγόριθμο έχουμε τον ίδιο σκοπό. Δηλαδή να βρούμε την θέση του στοιχείου *y* σε μια ταξινομημένη λίστα με *n* αριθμούς. Θα ακολουθηθεί παρόμοια διαδικασία όπως και στη σειριακή λύση με την δυαδική αναζήτηση αλλά στην παράλληλη εκδοχή θα χωρίζεται σε περισσότερες υπό-λίστες ανάλογα με τον αριθμό των επεξεργαστών που χρησιμοποιούνται.

Συγκεκριμένα σε κάθε βήμα η λίστα θα σπάζει σε *p+1* ίσες υπό-λίστες ίσου μεγέθους (ίσως με εξαίρεση την τελευταία η οποία μπορεί να είναι μικρότερη) οι οποίες θα παραμένουν ταξινομημένες . Ο κάθε επεξεργαστής (με προσωπικούς αριθμούς από 0 μέχρι *p-1*) θα αναλαμβάνει με βάση τον προσωπικό του αριθμό μια υπό-λίστα π.χ. ο επεξεργαστής με προσωπικό αριθμό 0 θα αναλάβει την πρώτη υπό-λίστα. Στη συνέχεια ο κάθε επεξεργαστής συγκρίνει το τελευταίο στοιχείο της υπό-λίστας της οποίας έχει αναλάβει με το στοιχείο *y*. Το αποτέλεσμα της σύγκρισης μπορεί να δώσει 3 αποτελέσματα:

1. Το στοιχείο που έλεγξε ο επεξεργαστής είναι μικρότερο από το *y* άρα τα υπόλοιπα στοιχεία της υπό-λίστας αποκλείεται να είναι μεγαλύτερα του *y*, έτσι στα επόμενα βήματα αποκλείονται τα στοιχεία αυτά από τους ελέγχους μας.
2. Το στοιχείο που έλεγξε ο επεξεργαστής είναι μεγαλύτερο από το *y* άρα τα υπόλοιπα στοιχεία της υπό-λίστας αποκλείεται να είναι μικρότερα του *y*, έτσι στα επόμενα βήματα αποκλείονται τα στοιχεία αυτά από τους ελέγχους μας.
3. Το στοιχείο που έλεγξε ο επεξεργαστής είναι ίσο με το *y* άρα έχουμε λύση.

Μετά το τέλος της αναζήτησης αυτής είτε έχουμε βρει το *y* είτε έχουμε καταλήξει σε μια από τις *p+1* υπό-λίστες.
Η διαδικασία αυτή τελειώνει όταν έχουμε βρει το *y* ή το μέγεθος της λίστας μας δεν ξεπερνά τον αριθμό των επεξεργαστών. Στην δεύτερη περίπτωση γίνεται μια παράλληλη αναζήτηση όπου ο κάθε επεξεργαστής αναλαμβάνει μέχρι ένα στοιχείο και έτσι μπορούμε να βρούμε την θέση του τελευταίου μεγαλύτερου στοιχείου της λίστας το οποίο είναι μικρότερο από το *y*.

Για την σωστή λειτουργία και τοποθέτηση των επεξεργαστών μας σε κάθε βήμα θα χρησιμοποιήσουμε δύο κοινόχρηστες μεταβλητές την *left* και *right* και δυο κοινόχρηστους πίνακες τον *c[0…p+1]* και *last[0…p+1]*
Κατά αρχάς ο P1 αρχικοποιεί ως εξής:
-Left=0, right =n+1
*-c[0]=0,*c[*p+1*]*=1*

Στη συνέχεια για να λειτουργήσει ότι αναφέρθηκε πιο πάνω, στην αρχή κάθε επανάληψης έχουμε :[2]
-Ο P1 θέτει *last*[*0*] = *left* και *last*[*p+1*] = *right*
-Ο κάθε επεξεργαστής $1 \leq i \leq p$ κάνει

- *last[i]=left+i[right-left/(p+1)].* (υπολογίζει τη θέση του si)
- Αν *y > si (xlast[i])* τότε *c[i] = 0*
- Αν *y < si* τότε *c[i] = 1*
- Αν *y = si* τότε *solution = last[i]* -- ΤΕΛΟΣ
- Αν *c[i] < c[i+1]* τότε *left = last[i], right = last[i+1]*
- [*P1* μόνο] Αν *c[0] < c[1]* τότε *left = last[0], right = last[1]*

Όλα τα βήματα εκτελούνται συγχρονισμένα και παράλληλα από τους επεξεργαστές για τον λόγο αυτό αναμφισβήτητα χρησιμοποιείται μοντέλο PRAM για την ανάπτυξη του αλγορίθμου. Πιο συγκεκριμένα χρησιμοποιείται το μοντέλο CREW-PRAM . Αυτό γίνεται για τον λόγο ότι σε κανένα σημείο δεν υπάρχει περίπτωση να παρουσιαστεί ταυτόχρονο γράψιμο σε κοινή κυψελίδα μνήμης , όμως θα συμβεί σίγουρα ταυτόχρονο διάβασμα. Ταυτόχρονο διάβασμα γίνεται στο πρώτο βήμα όπου όλοι οι επεξεργαστές ταυτόχρονα διαβάζουν τον αριθμό *y*. Στη συνέχεια ο κάθε επεξεργαστής διαβάζει διαφορετική υπό-λίστα, διαφορετικούς αριθμούς και γράφει σε διαφορετικές θέσεις στην μνήμη έτσι δεν χρειάζεται ξανά ταυτόχρονο διάβασμα ή γράψιμο και το CREW είναι το καταλληλότερο μοντέλο για το πρόβλημα αυτό.

Είμαστε βέβαιοι για την ορθότητα του αλγορίθμου επειδή σε κάθε βήμα η ταξινομημένη λίστα χωρίζεται σε συνεχόμενες υπό-λίστες οι οποίες είναι και αυτές ταξινομημένες. Έτσι εξασφαλίζεται ότι σε κάθε βήμα απορρίπτοντας στοιχεία δεξιά ή αριστερά συγκρίνοντας το *y* μόνο με το τελευταίο στοιχείο της υπό-λίστας, είμαστε βέβαιοι πως τα στοιχεία που απορρίψαμε έχουν απορριφθεί σωστά.

Χρόνος : $T_p(n) = O(\log n / \log p)$

Στο πρώτο βήμα για τον διαχωρισμό και ανάθεση των λιστών στους επεξεργαστές χρειάζεται σταθερό χρόνο $\Theta(1)$.

Αν το μέγεθος της λίστας δεν ξεπερνά τον αριθμό των επεξεργαστών τότε η εύρεση του δείκτη του κ υπολογίζεται σε χρόνο $\Theta(1)$ (Η περίπτωση αυτή μπορεί να είναι στο αρχικό στάδιο παίρνοντας μια μικρή λίστα ή μετά από ένα αριθμό αναδρομών)

Άρα μένει να υπολογίσουμε τον μέγιστο αριθμό επαναλήψεων της αναδρομής αυτής. Σε κάθε επανάληψη το μέγεθος της λίστας μικραίνει κατά $1/(p+1)$. Έτσι στην ι-οστή επανάληψη έχουμε $n/(p+1)^{i-1}$.

Έστω το μέγεθος έφτασε στο p+1 σε λ επαναλήψεις

Τότε έχουμε $n/(p+1)^{\lambda-1}=p+1 \Rightarrow n=(p+1)^{\lambda} \Rightarrow \lambda = O(\log n/ \log p))$

Έτσι έχουμε $O(\log n/\log p)$ επαναλήψεις όπου η κάθε επανάληψη χρειάζεται $\Theta(1)$ χρόνο άρα ο συνολικός χρόνος του αλγόριθμου είναι $O(\log n/\log p)$

Χρησιμοποιούμε *p* επεξεργαστές άρα το κόστος μας είναι ο χρόνος επί το *p*

Κόστος : $C_p(n) = O(p \log n/\log p)$

Όπως ξέρουμε για να είναι ο αλγόριθμος βέλτιστος πρέπει το κόστος του να είναι ίσο με το κόστος του καλύτερου σειριακού που όπως αναφέραμε το κόστος του καλύτερου σειριακού για το συγκεκριμένο πρόβλημα είναι O(*logn*). Έτσι ο αλγόριθμος αυτός είναι βέλτιστος για σταθερό αριθμό επεξεργαστών

Επίσης είναι και χρόνου βέλτιστος γιατί ο χρόνος αυτός είναι το κατώτερο φράγμα αναζήτησης.[2,5]

Αυτό αποδεικνύεται γιατί κάθε αλγόριθμος που χρησιμοποιεί p <= n επεξεργαστές, μπορεί να συγκρίνει ένα στοιχείο y το πολύ με p στοιχεία μιας λίστας Λ ταυτόχρονα. Μετά από αυτές τις συγκρίσεις και την διαγραφή των στοιχείων που σίγουρα δεν ισούνται με το y, η υπο- λίστα που παραμένει πρέπει να έχει μέγεθος τουλάχιστον:

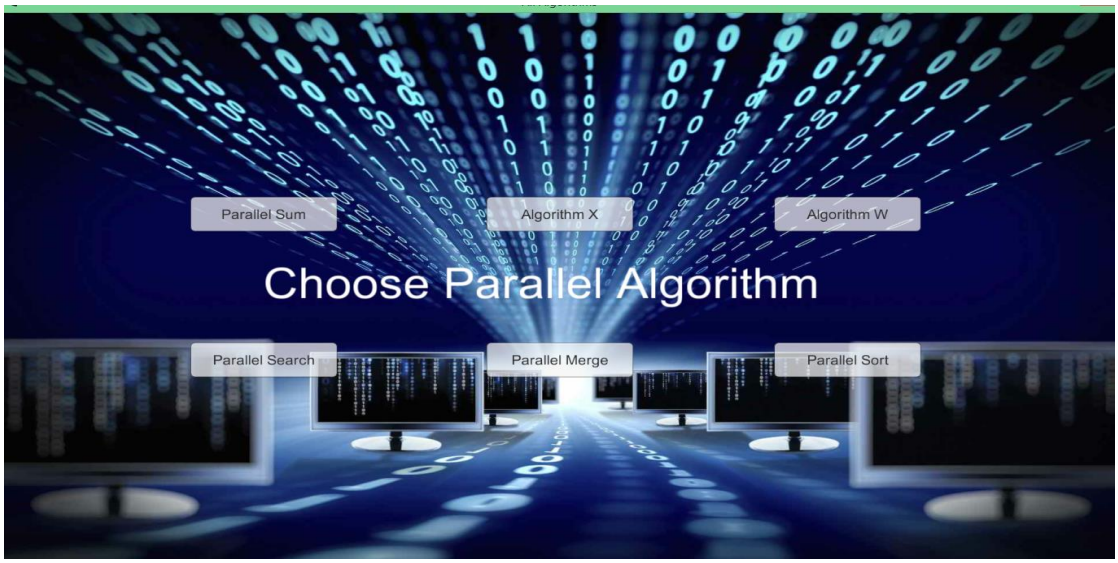
$(n-p)/(p+1) > = (n-p)/(p+1) = [(n+1)/(p+1)] - 1$

Μετά από λ επαναλήψεις, έχουμε μια λίστα μεγέθους $[(n+1)/(p+1)^{\lambda}] - 1$.Έτσι ο αριθμός των επαναλήψεων είναι τέτοιος, ώστε $Q[(n+1)/(p+1)^{\lambda}] - 1 <= 0$

Άρα $\lambda = \Omega(\log n / \log p)$ που είναι το κατώτατο φράγμα.

5.3.2 Γραφική Αναπαράσταση Αλγορίθμου

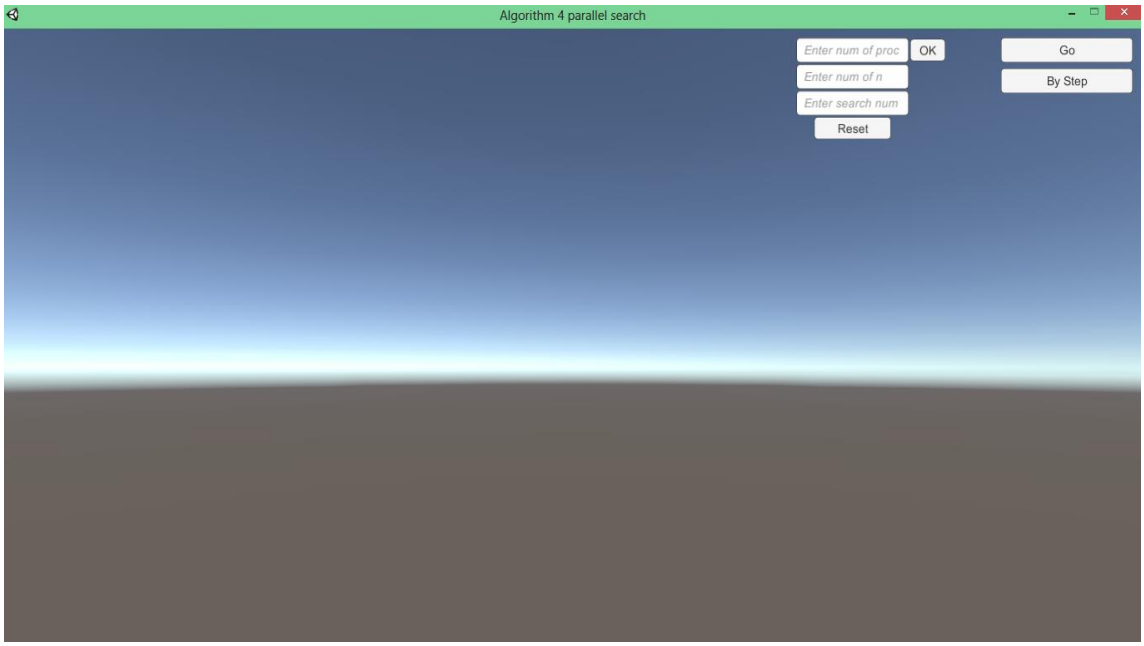
Στο σημείο αυτό θα δοθούν λεπτομέρειες για το πώς υλοποιήθηκε ο αλγόριθμος παράλληλης αναζήτησης στο Unity και βήμα προς βήμα πως τρέχει ο αλγόριθμος. Καταρχάς τρέχουμε το εκτελέσιμο .



Σχήμα 5.1: Πρώτη εικόνα εκτελέσιμου

Στο Σχήμα 5.1 είναι η πρώτη εικόνα που βλέπουμε μόλις τρέξουμε το εκτελέσιμο.Στη συνέχεια επιλέγουμε το κουμπί του Parallel Search όπου είναι ο αλγόριθμος ο οποίος μελετάμε στο συγκεκριμένο κεφάλαιο.

Ακολούθως εμφανίζεται η εικόνα στο Σχήμα 5.2. Σε οποιαδήποτε σημείο που τρέχει το πρόγραμμα πατώντας το κουμπί ESC πηγαίνουμε στην αρχική επιλογή αλγορίθμων ή πατώντας το P γίνεται παύση.



Σχήμα 5.2: Αρχική Εικόνα

Η αρχική εικόνα αποτελείται από:

- 3 input fields τα όποια ο χρήστης περνά τιμές:
  - -Ένα για το πλήθος των επεξεργαστών
  - -Ένα για το πλήθος αριθμών της λίστας
  - -Ένα για τον αριθμό y τον οποίο ψάχνουμε στην λίστα
- Το κουμπί OK με το οποίο καταχωρείτε το πλήθος επεξεργαστών και αριθμών της λίστας σε δυο μεταβλητές αντίστοιχα.
- Ένα κουμπί το κουμπί Reset το οποίο κάνει reset την σκηνή στην αρχική της μορφή για να εκτελέσει νέο παράδειγμα ο χρήστης
- Το κουμπί Go το οποίο τρέχει τον αλγόριθμο ολόκληρο με μια μικρή καθυστέρηση μεταξύ των βημάτων για να διακρίνονται αυτά.
- Το κουμπί By Step με το οποίο ο αλγόριθμος προχωρά στο επόμενο ακριβώς βήμα
- Το canvas που μέσα σε αυτό ήταν όλα τα πιο πάνω.

Τα κουμπιά είναι ανεξάρτητα το ένα με το άλλο δηλαδή αν πρώτα εκτελέσουμε ένα βήμα με το κουμπί By Step πατώντας ακολούθως το Go θα ολοκληρωθεί ο αλγόριθμος από εκεί όπου είχε μείνει.

Η κάμερα είναι σε ορθογραφική προβολή με size 10 ώστε να χωρεί στην οθόνη όσο περισσότερους αριθμούς μπορεί και οι αριθμοί αυτοί να παραμένουν διακριτοί στο μάτι.

Για την δημιουργία του αλγόριθμου αυτού χρησιμοποιήθηκε ένα prefab το οποίο είχε μέσα ένα ακόμα αντικείμενο το οποίο αποτελούταν από ένα text mesh όπου στη συνέχεια σε αυτό το text mesh αποθηκεύεται και εμφανίζεται ο αριθμός.

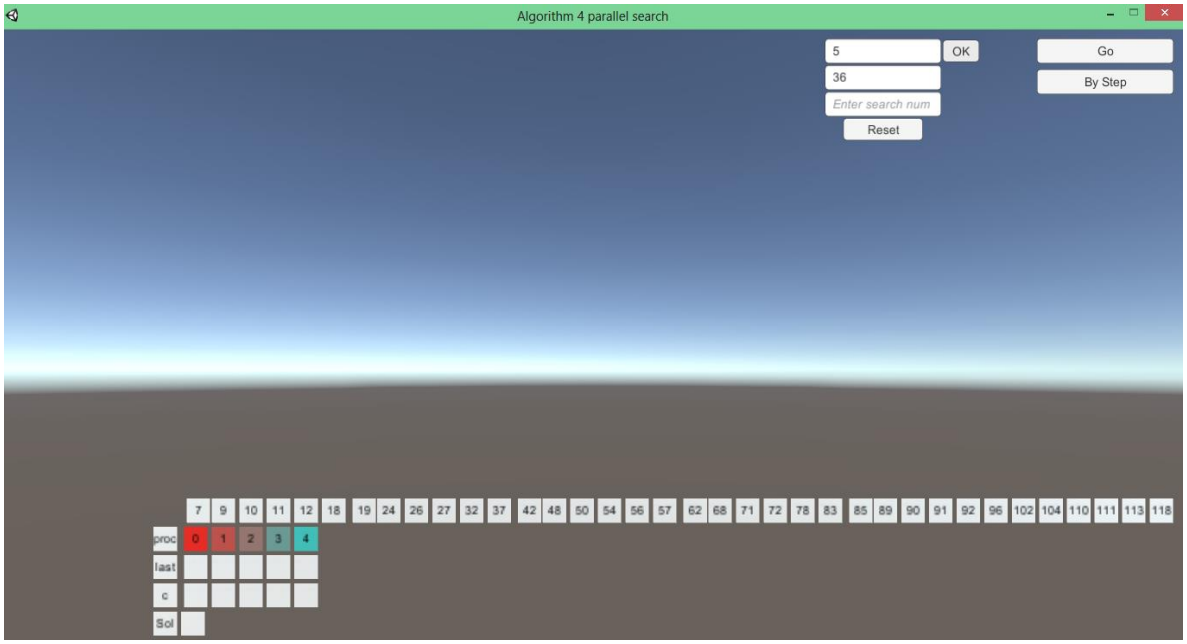
Για την δημιουργία του κώδικα χρησιμοποιήθηκε μόνο ένα script το οποίο ήταν ενσωματωμένο στο canvas και το κάθε κουμπί καλούσε την αντίστοιχη συνάρτηση με αυτή που έπρεπε να εκτελέσει.

Αν ο χρήστης προσπαθήσει να πατήσει το κουμπί OK χωρίς τα 2 πρώτα input fields να έχουν τιμή τότε δεν θα αρχικοποιήσει την σκηνή μας. Αν δεν αρχικοποιηθεί η σκηνή μας κανένα από τα άλλα κουμπιά δεν λειτουργεί. Επίσης τα κουμπιά Go και By Step δεν λειτουργούν μέχρι να υπάρχει μια τιμή στο τρίτο input box. Με τον τρόπο αυτό περιορίζουμε τον χρήστη από το να υποπέσει σε άσκοπα σφάλματα και δημιουργείται μια καλύτερη αλληλεπίδραση με τον πρόγραμμα και το UI .

Ο αλγόριθμος είναι υλοποιημένος με τέτοιο τρόπο ώστε να υποστηρίζει μέχρι πλήθος 36 αριθμών στην λίστα και 36 ενεργών επεξεργαστών.

Στη συνέχεια θα αναλυθεί ένα παράδειγμα με 36 αριθμούς και 5 επεξεργαστές.

Ο χρήστης γράφει τον αριθμό 6 στην περίπτωση μας στο πρώτο input field (Enter num of proc) και τον αριθμό 36 στο δεύτερο input field (Enter num of n) και ακολούθως πατά το Ok button. Στη συνέχεια παρατηρεί να εμφανίζεται στην οθόνη του η ακόλουθη εικόνα.(Σχήμα 5.3)

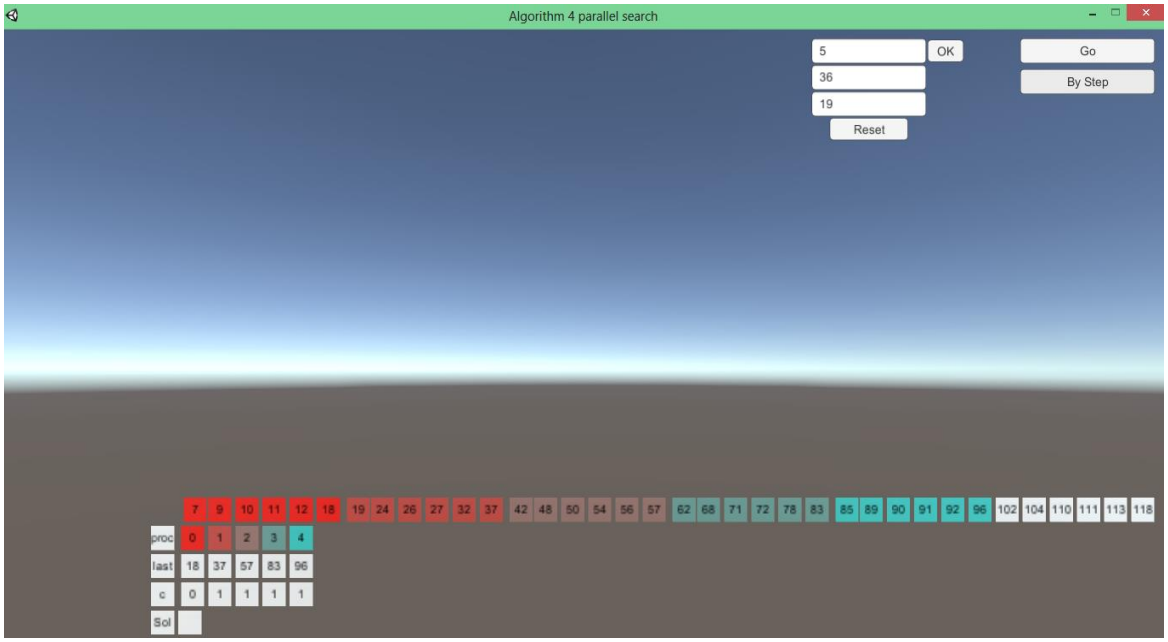


Σχήμα 5.3: Εμφάνιση επεξεργαστών και αριθμών

Όπως βλέπουμε εμφανίζονται 36 κουτιά με αριθμούς όπως έχουμε επιλέξει οι οποίοι είναι ταξινομημένοι. Επίσης βλέπουμε το κουτί proc και δίπλα 5 κουτιά ένα για κάθε επεξεργαστή με το χρώμα που θα χρωματίζει τις ενέργειες που εκτελεί. Πιο κάτω ακριβώς βλέπουμε τον πίνακα c τον οποίο θα χρησιμοποιεί για να βρει σε ποια υπό-λίστα βρίσκεται ο αριθμός και τέλος το Sol όπου θα δοθεί ο αριθμός *k*.
Τα χρώματα για τους επεξεργαστές βρίσκονται με τον ίδιο τρόπο όπως στον αλγόριθμο παράλληλης άθροισης (βλ. σελίδα 28).

Όπως και η δημιουργία και τοποθέτηση των κουτιών έχει γίνει με την εντολή instantiate δημιουργώντας το prefab που έχει επίσης αναφερθεί στον αλγόριθμο παράλληλης άθροισης (βλ. σελίδα 29) . Η διαφορά με την παράλληλη άθροιση είναι ότι στον συγκεκριμένο αλγόριθμο για τον αριθμό που υπάρχει στο κουτί κάθε φορά υπολογιζόταν ένας τυχαίος μικρός αριθμός και γινόταν πρόσθεση με τον προηγούμενο για να διατηρηθεί η ταξινομημένη λίστα ενώ στην παράλληλη ήταν απλά τυχαίος.

Στη συνέχεια πατώντας βάζουμε τον αριθμό που επιθυμούμε στο 3ο input box (Enter search num) π.χ. στο παράδειγμα μας βάζουμε τον αριθμό 19 και πατάμε μια φορά το By Step. Βλέπουμε την ακόλουθη εικόνα να εμφανίζεται που είναι το πρώτο βήμα του αλγόριθμου.



Σχήμα 5.4: Βήμα 1

Οι επεξεργαστές αναλαμβάνουν *n/(p+1)* αριθμούς. Άρα στο παράδειγμα μας αναλαμβάνουν 36/6 = 6 αριθμούς. Όπως φαίνεται στην πιο πάνω εικόνα ο κάθε επεξεργαστής έχει χρωματίσει από 6 αριθμούς και μένουν οι τελευταίοι 6 αριθμοί οι οποίοι δεν έχουν χρωματιστεί (οι τελευταίοι αριθμοί μπορεί να είναι άνισο το πλήθος τους με τις πρώτες υπό-λίστες απλά με τον αριθμό 36 χωρίζεται ακριβώς).

Στη συνέχεια ο κάθε επεξεργαστής έχει τοποθετηθεί στο τελευταίο στοιχείο της υπό-λίστας της οποίας έχει αναλάβει. Αυτό φαίνεται στον πίνακα *last* όπου ο επεξεργαστής 0 έχει τοποθετηθεί στο 18 , ο 1 στο 37 , ο 2 στο 57 , ο 3 στο 83 και ο 4 στο 96.

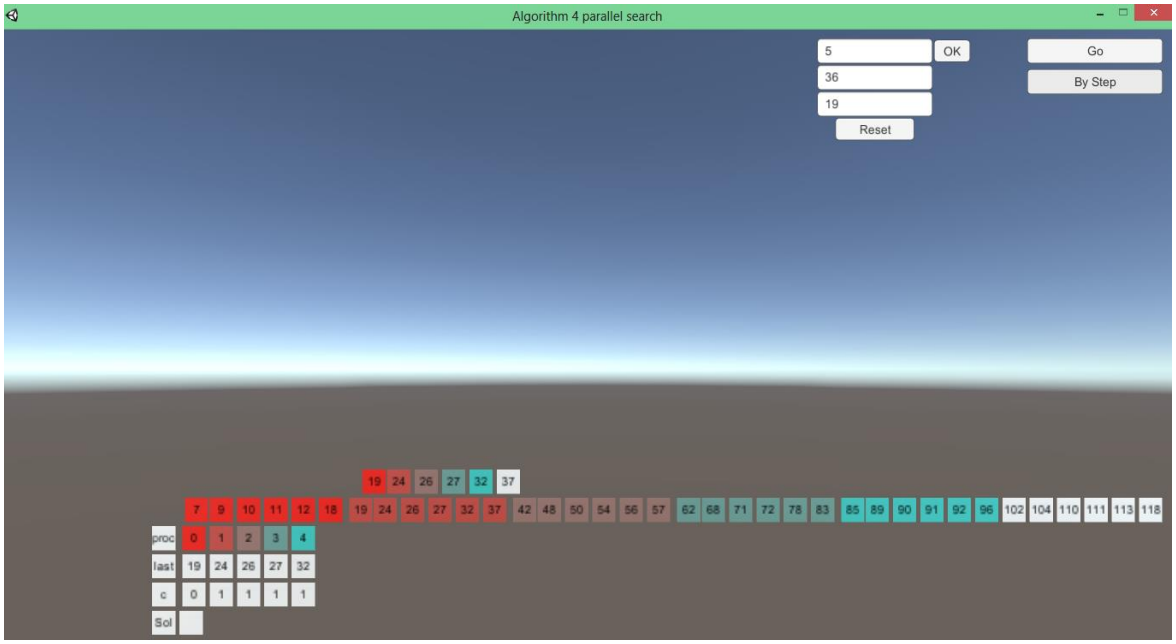
Στη συνέχεια ο κάθε επεξεργαστής συγκρίνει το στοιχείο last του με τον αριθμό που έχουμε περάσει στο input box. Αν ο αριθμός y είναι μεγαλύτερος από το last τότε γράφει 0 στον πίνακα c στην αντίστοιχη θέση αν είναι μικρότερος γράφει 1 αν είναι ίσος τότε έχουμε βρει τον αριθμό τον οποίο ψάχναμε. Στο παράδειγμα μας πήραμε τα ακόλουθα :

- 18<19 -> *c*=0
- 37>19 -> *c*=1
- 57>19 -> *c*=1
- 83>19 -> *c*=1
- 96>19 -> *c*=1

Ακολούθως βλέπουμε το σημείου του κώδικα το οποίο είναι υπεύθυνο για τον υπολογισμό των *last* και *c* σε κάθε βήμα.

```
last [0] = left;
last [proc + 1] = right;
for (int i=0; i<proc; i++) {
        last [i + 1] = left + (i + 1) * ((right - left) / (proc + 1));
        StartCoroutine (waittransf (lasttable [i], table [last [i + 1] - 1], se));
        if (table [last [i + 1] - 1] > search)
                c [i + 1] = 1;
        else if (table [last [i + 1] - 1] < search)
                c [i + 1] = 0;
        else {
                procfound = i;
                solfound = last [i + 1] - 1;
                solved = 1;
                c [i + 1] = 1;
        }
        StartCoroutine (waittransf (ctable [i], c [i + 1], se));
        }
        se = se + 0.6f;
        for (int i=1; i<=proc; i++) {
                if (c [i] < c [i + 1]) {
                        left = last [i];
                        right = last [i + 1];
                }
        }
        if (c [0] < c [1]) {
                left = last [0];
                right = last [1];
        }
        length = right - left;
}
```

Στη συνέχεια πατάμε το κουμπί By Step και παρατηρείται η εικόνα στο Σχήμα 5.5.



Σχήμα 5.5: Βήμα 2

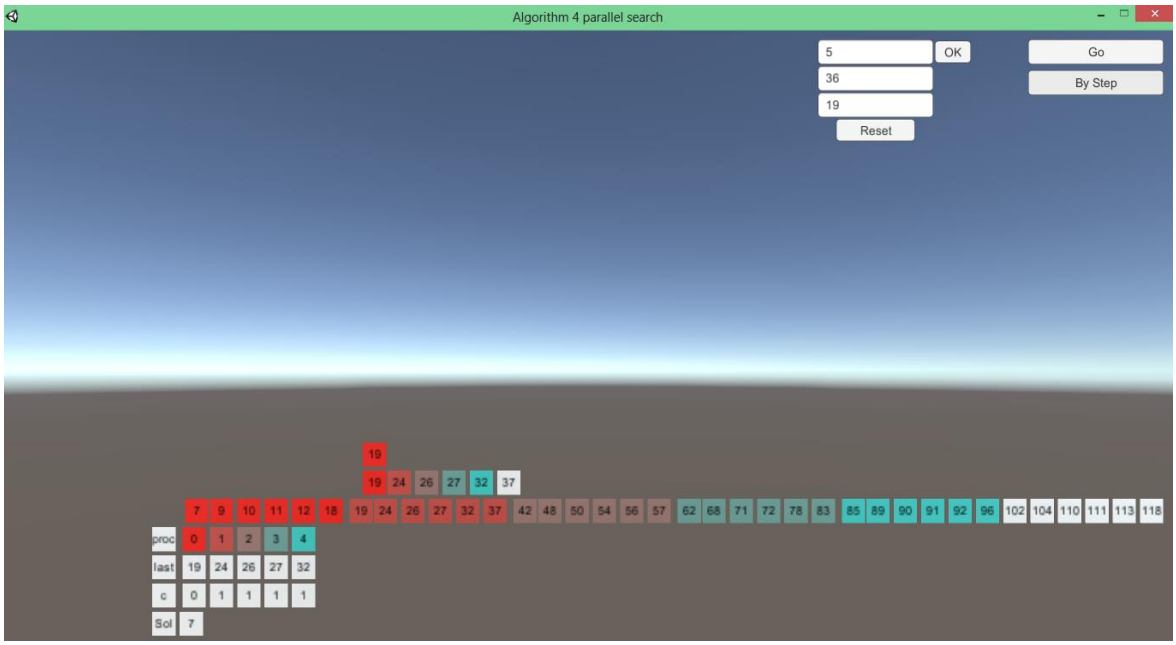
Όπως παρατηρείται στην πιο πάνω εικόνα έχει γίνει επιλογή της δεύτερης υπό-λίστας για να συνεχίσει ο αλγόριθμος. Αυτό γίνεται γιατί σε δύο συνεχόμενες θέσεις του πίνακα c έχουμε 0 και 1 αντίστοιχα. Αυτό σημαίνει ότι ο αριθμός που ψάχνουμε είναι μεγαλύτερος του last στην πρώτη υπό-λίστα και μικρότερος του *last* στην δεύτερη υπό-λίστα. Άρα είμαστε βέβαιοι ότι βρίσκεται ενδιάμεσα από αυτούς και στέλνουμε όλους τους επεξεργαστές να ψάξουν την υπό-λίστα αυτή.

Το κομμάτι κώδικα που κάνει την λειτουργία αυτή φαίνεται εδώ

```
for (int i=1; i<=proc; i++) {
        if (c [i] < c [i + 1]) {
                left = last [i];
                right = last [i + 1];
        }
}
if (c [0] < c [1]) {
left = last [0];
right = last [1];
}
```

Έτσι στο βήμα αυτό η νέα μας λίστα που αποτελείται από 6 αριθμούς χωρίζεται ξανά σε 6 υπό-λίστες μια για κάθε επεξεργαστή και μια που δεν ελέγχεται από κανένα. Η κάθε υπό-λίστα τώρα αποτελείτε από 1 μόνο αριθμό. Ο κάθε επεξεργαστής ανανεώνει το last του με τον αριθμό αυτό όπως φαίνεται πιο πάνω με 19,24,26,27,32 αντίστοιχα όπου οι τελευταίοι 4 βρίσκουν πως ο αριθμός αυτός είναι μεγαλύτερος από αυτόν που ψάχνουμε ενώ ο πρώτος επεξεργαστής μας βρίσκει λύση.

Πατώντας το By Step θα εμφανιστεί η λύση όπως φαίνεται πιο κάτω. (Σχήμα 5.6)



Σχήμα 5.6: Τελικό αποτέλεσμα

Όπως έχουμε αναφέρει η λύση έχει βρεθεί από τον πρώτο επεξεργαστή όπου είναι ο αριθμός 19 και έχει βρει σε πιο σημείο της υπό-λίστας βρίσκεται ο αριθμός αυτός. Έτσι ξέρουμε ο αριθμός y από πόσους αριθμούς είναι μεγαλύτερος ή ίσος και αναγράφεται δίπλα από το κενό κουτί sol όπως φαίνεται στην πιο πάνω εικόνα. Στο παράδειγμα μας η λύση είναι το 7 που είναι ορθή αφού ο αριθμός 19 είναι μεγαλύτερος ή ίσος από 7 αριθμούς στην αρχική μας λίστα τους 7,9,10,11,12,18,19.

Κεφάλαιο 6

Αλγόριθμος Παράλληλης Συγχώνευσης και Αλγόριθμος Παράλληλης Ταξινόμησης

6.1 Αλγόριθμος Παράλληλης Συγχώνευσης

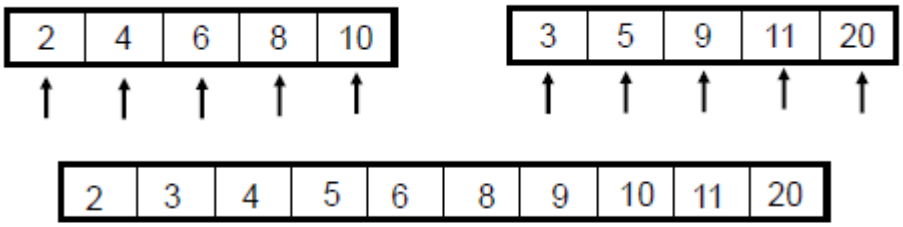
Στο σημείο αυτό γίνεται περιγραφή του προβλήματος συγχώνευσης καθώς και του σειριακού αλγόριθμου για την επίλυση του. Επίσης περιγράφεται και ο αλγόριθμος παράλληλης συγχώνευσης με λεπτομέρειες για την γραφική αναπαράσταση του που υλοποιήθηκε στην εργασία.

6.1.1 Πρόβλημα

Το πρόβλημα της συγχώνευσης είναι το εξής : *Δεδομένο δύο ταξινομημένων λιστών A = (a1, a2, ..., am) και B = (b1, b2, ..., bn). Ζητείται να συγχωνευθούν οι δύο λίστες και να σχηματίσουν την ταξινομημένη λίστα C = (c1, c2, ..., cm+n)* [2,5]

6.1.2 Σειριακός Αλγόριθμος

Στην περίπτωση του σειριακού υπολογισμού, όπου έχουμε δηλαδή μόνο ένα επεξεργαστή αυτός ο επεξεργαστής θα μετακινείται με δύο δείκτες ένα σε κάθε λίστα (Σχήμα 6.1) και θα ελέγχει ποιος αριθμός είναι μικρότερος στις δυο αυτές λίστες στον δείκτη αυτό. Στη συνέχεια θα τον γράφει στην τρίτη λίστα C και θα μετακινεί τον δείκτη από την λίστα που πήρε τον αριθμό.



Σχήμα 6.1[2]

Ο χρόνος εκτέλεσης του σειριακού αλγόριθμου είναι $T(m+n)=\Theta(m+n)$ όπου είναι ο χρόνος που χρειάζεται ο επεξεργαστής για την προσπέλαση των δύο λιστών.
Έτσι και το κόστος του αλγόριθμου είναι $C(m+n)=\Theta(m+n)$

6.1.3 Παράλληλος Αλγόριθμος

Στο υποκεφάλαιο αυτό περιγράφεται ο αλγόριθμος παράλληλης συγχώνευσης με λεπτομέρειες για την γραφική αναπαράσταση του που υλοποιήθηκε στην εργασία.

6.1.3.1 Περιγραφή Αλγορίθμου

Για το πρόβλημα αυτό στον παράλληλο υπολογισμό θα βρίσκουμε παράλληλα που κατατάσσεται το κάθε στοιχείο της μικρότερης λίστας αν το βάλουμε στην πρώτη. Για να το επιτύχουμε αυτό θα χρησιμοποιηθούν τα rank όπως φαίνεται πιο κάτω.

rank(a_i : A) : Είναι ο αριθμός των στοιχείων στην λίστα A που είναι μικρότερα ή ίσα με το a_i να ανήκει στην λίστα A.

rank(b_i : A) : Είναι ο αριθμός των στοιχείων στην λίστα Α που είναι μικρότερα ή ίσα με το b_i να ανήκει στην λίστα Β.
Η θέση του στοιχείου a_i στη λίστα C είναι rank(a_i : C = A+B) =rank(a_i : A) + rank(a_i : B)

Μια λίστα B=(b_1, b_2, …, b_n) κατατάσσεται στην λίστα Α αν υπολογίσουμε την λίστα (r_1,r_2,…,r_n) όπου r_i=rank(b_i:A).
Αυτό συμβολίζεται ως rank(B : A) = (r_1, r_2, …, rn)

Έτσι το τελικό πρόβλημα λύνεται με τον υπολογισμό του rank κάθε στοιχείου της λίστας Α ή Β στη λίστα A+B=C

Παράδειγμα :
A = (2,5,6,8,11)
B = (3,7,13)
Rank (A: B) = (0, 1, 1, 2, 2)
Rank(B:A)=(1,3,5)
C= (2, 3,5,6,7,8,11,13)

Στη συνέχεια θα αναφερθεί πως μπορεί να υπολογιστεί γρήγορα το rank για εξοικονόμηση χρόνου και κόστους.
Έστω ότι έχουμε δύο ταξινομημένες λίστες την Α και Β πλήθους *n* και *m* αντίστοιχα.
Πρώτα χωρίζουμε την λίστα Β σε $m^{1/2}$ υπό-λίστες, $m^{1/2}$ στοιχείων η κάθε μια.
Στη συνέχεια θα υπολογίσουμε την κατάταξη του τελευταίου στοιχείου της υπό-λίστας χρησιμοποιώντας παράλληλα τον Αλγόριθμο παράλληλης αναζήτησης όπως περιγράφηκε στο προηγούμενο κεφάλαιο με $n^{1/2}$ επεξεργαστές.
Ακολούθως για κάθε επεξεργαστή ελέγχεται Αν r(i) = r($i+1$), τότε rank(B_i : A_i) = (0,…,0).
Διαφορετικά υπολογίζεται αναδρομικά το rank(B_i:A_i).
Τέλος αν το 1<=k<=m , δεν είναι πολλαπλάσιο του $m^{1/2}$, και I = floor($k/m^{1/2}$).
Τότε rank(b_k : A) = r(i) + rank(b_k : A_i)

Στη συνέχεια δίνεται ένα παράδειγμα υπολογισμού του rank [2]:

A = (-5, 0, 3, 4, 17, 18, 24, 28) n = 8
B = (1, 2, 15, 21) m = 4

Βήμα 1: Χωρίζεται η B: (1, 2) και (15, 21)

Βήμα 2: r(0) = 0
r(1) = rank(2 : A) = 2
r(2) = rank(21 : A) = 6

rank(B:A) = (2,2,4,6)

Βήμα 3: B_0 = (1) και A_0=(-5,0)
B_1 = (15) και A_1 = (3,4,17,18)

rank(B_0,A_0) = rank(1,A_0) = 2
rank(B_1,A_1) = rank(15,A_1) = 2

Βήμα 4: k =1, 3 δεν είναι πολλαπλάσια του 2
rank(b_1 : A) = r(0) + rank(1 : A_0) = 2
rank(b_3 : A) = r(1) + rank(15 : A1) = 2 + 2 = 4

Όλα τα βήματα εκτελούνται συγχρονισμένα και παράλληλα από τους επεξεργαστές για τον λόγο αυτό αναμφισβήτητα χρησιμοποιείται μοντέλο PRAM για την ανάπτυξη του αλγορίθμου. Πιο συγκεκριμένα χρησιμοποιείται το μοντέλο CREW-PRAM . Αυτό γίνεται για τον λόγο ότι σε κανένα σημείο δεν υπάρχει περίπτωση να παρουσιαστεί ταυτόχρονο γράψιμο σε κοινή κυψελίδα μνήμης , όμως θα συμβεί σίγουρα ταυτόχρονο διάβασμα. Ταυτόχρονο διάβασμα γίνεται στο σημείο όπου οι επεξεργαστές κάνουν παράλληλη αναζήτηση για να βρουν το rank του στοιχείου έτσι πολύ πιθανόν να διαβάσουν την ίδια κυψελίδα την ίδια χρονική στιγμή.

Χρόνος και πολυπλοκότητα αλγορίθμου

Στο βήμα 2 χρησιμοποιείται ο Αλγόριθμος Παράλληλης αναζήτησης για $m^{1/2}$ στοιχεία παράλληλα. Ο χρόνος εκτέλεσης για n στοιχεία του αλγόριθμου χρειάζεται O($logn/logp$). Στην περίπτωση μας όμως παίρνει χρόνο O(1) γιατί για κάθε στοιχείο χρησιμοποιούνται $n^{1/2}$ επεξεργαστές άρα O($log\ (n+1)/log\ (n^{1/2} + 1)$ = O(1) και O($n^{1/2} * m^{1/2}$) = O($n + m$) κόστος

Στο βήμα 3 κάθε αναδρομική εκτέλεση έχει κόστος O($n + m$).Η αναδρομική εκτέλεση του (B_i, A_i) παίρνει χρόνο T($m^{1/2}$).

Έτσι ο συνολικός χρόνος του βήματος 3 είναι T(m) = T($m_{1/2}$) + O(1) = O($loglog\ m$)

Για το Βήμα 4 χρειαζόμαστε σταθερό χρόνο O(1), και το κόστος είναι O($n+m$)

Έτσι καταλήγουμε στα πιο κάτω τελικά αποτελέσματα:

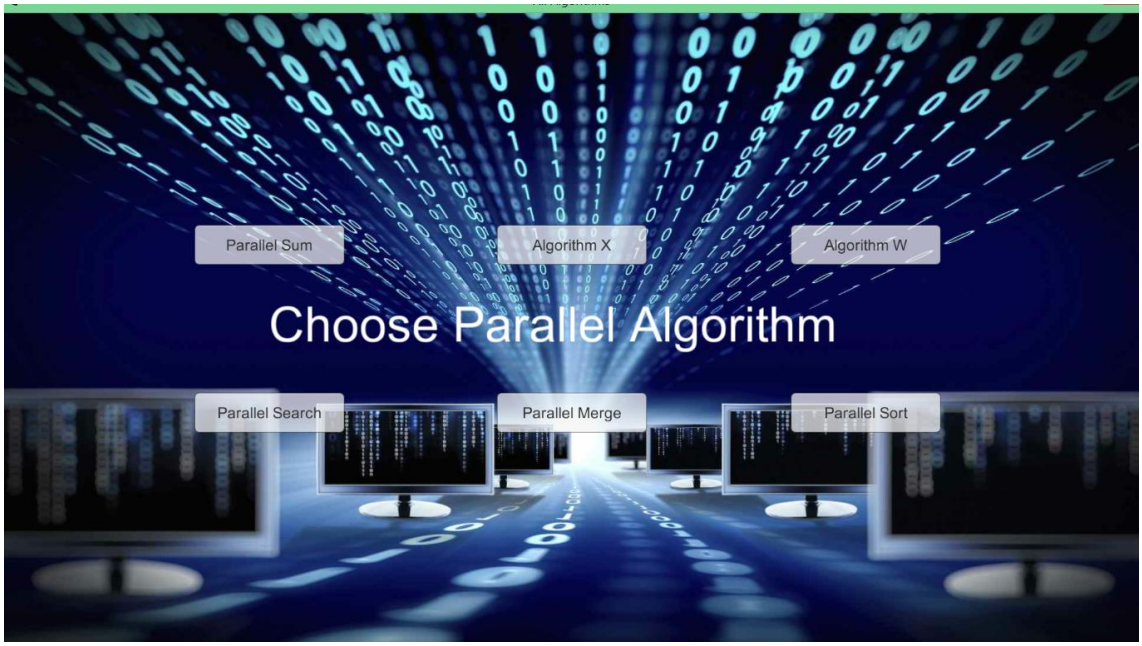
-Συνολικός χρόνος εκτέλεσης: O(loglog m)

-Συνολικό κόστος: O((n+m) loglog m)

Ο αλγόριθμος δηλαδή είναι βέλτιστος προς τον χρόνο αλλά όχι προς το κόστος. Με κάποιες αλλαγές μπορεί να γίνει και βέλτιστος ως προς το κόστος.

6.1.3.2 Γραφική Αναπαράσταση Αλγορίθμου

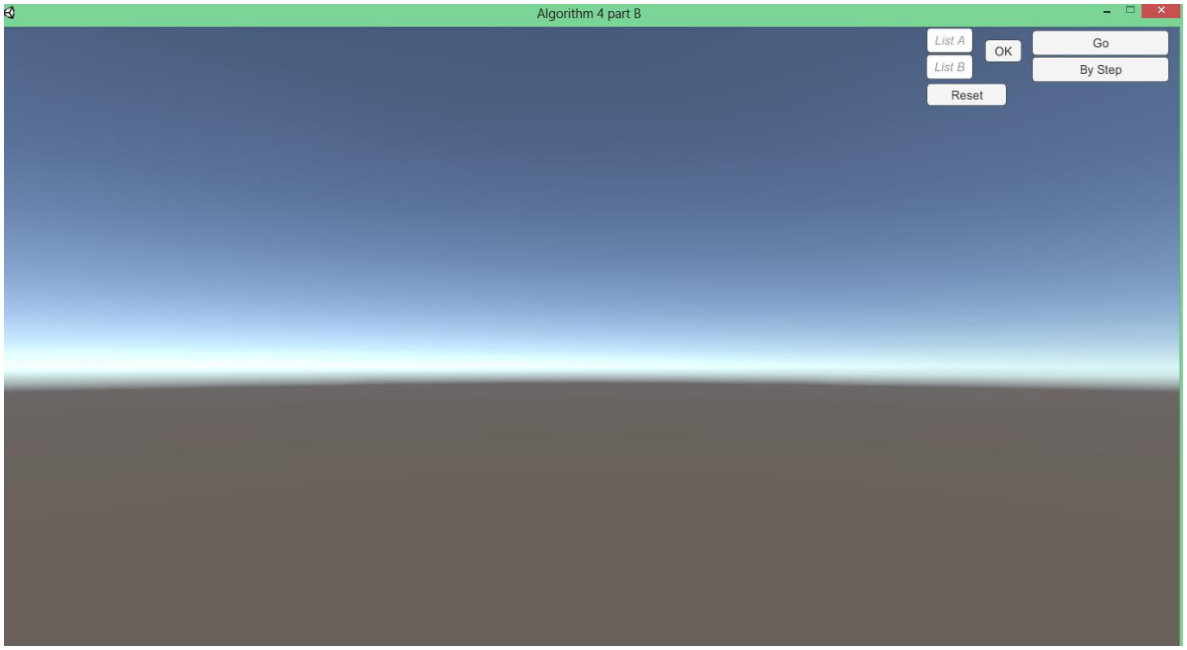
Στη συνέχεια θα δοθούν λεπτομέρειες για το πώς υλοποιήθηκε ο αλγόριθμος παράλληλης συγχώνευσης στο Unity και βήμα προς βήμα πως τρέχει ο αλγόριθμος. Καταρχάς τρέχουμε το εκτελέσιμο . Στο σχήμα 6.2 είναι η πρώτη εικόνα που βλέπουμε μόλις τρέξουμε το εκτελέσιμο



Σχήμα 6.2: Πρώτη εικόνα εκτελέσιμου

Στη συνέχεια επιλέγουμε το κουμπί του Parallel Merge όπου είναι ο αλγόριθμος ο οποίος μελετάμε στο συγκεκριμένο κεφάλαιο.

Ακολούθως εμφανίζεται η πιο κάτω αρχική εικόνα (Σχήμα 6.3). Σε οποιαδήποτε σημείο που τρέχει το πρόγραμμα πατώντας το κουμπί ESC πηγαίνουμε στην αρχική επιλογή αλγορίθμων ή πατώντας το P γίνεται παύση.



Σχήμα 6.3: Αρχική Εικόνα

Ενδιάμεσα σε κάποια βήματα χρησιμοποιείται η συνάρτηση destroyall για να καταστρέφονται κάποια κουτιά τα οποία ομαδοποίησα για να δημιουργείται χώρος στην οθόνη

Η αρχική εικόνα αποτελείται από:

- 3 input fields τα όποια ο χρήστης περνά τιμές:
  -Ένα για το πλήθος αριθμών της λίστας A
  -Ένα για το πλήθος αριθμών της λίστας B
- Το κουμπί OK με το οποίο καταχωρείτε το πλήθος των δύο λιστών σε δυο μεταβλητές αντίστοιχα.
- Ένα κουμπί το κουμπί Reset το οποίο κάνει reset την σκηνή στην αρχική της μορφή για να εκτελέσει νέο παράδειγμα ο χρήστης
- Το κουμπί Go το οποίο τρέχει τον αλγόριθμο ολόκληρο με μια μικρή καθυστέρηση μεταξύ των βημάτων για να διακρίνονται αυτά.
- Το κουμπί By Step με το οποίο ο αλγόριθμος προχωρά στο επόμενο ακριβώς βήμα
- Το canvas που μέσα σε αυτό ήταν όλα τα πιο πάνω.

Τα κουμπιά είναι ανεξάρτητα το ένα με το άλλο δηλαδή αν πρώτα εκτελέσουμε ένα βήμα με το κουμπί By Step πατώντας ακολούθως το Go θα ολοκληρωθεί ο αλγόριθμος από εκεί όπου είχε μείνει.

Η κάμερα είναι σε ορθογραφική προβολή με size 10 ώστε να χωρεί στην οθόνη όσο περισσότερους αριθμούς μπορεί και οι αριθμοί αυτοί να παραμένουν διακριτοί στο μάτι.

Για την δημιουργία του αλγόριθμου αυτού χρησιμοποιήθηκε ένα prefab το οποίο είχε μέσα ένα ακόμα αντικείμενο το οποίο αποτελούταν από ένα text mesh όπου στη συνέχεια σε αυτό το text mesh αποθηκεύεται και εμφανίζεται ο αριθμός.

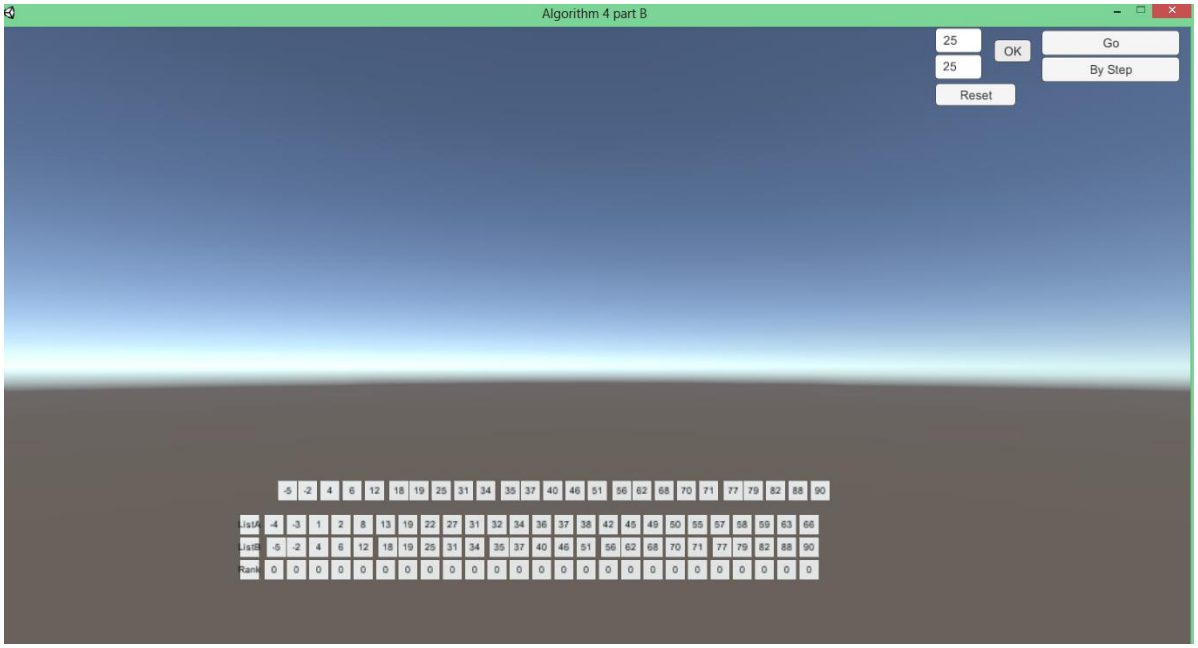
Για την δημιουργία του κώδικα χρησιμοποιήθηκε μόνο ένα script το οποίο ήταν ενσωματωμένο στο canvas και το κάθε κουμπί καλούσε την αντίστοιχη συνάρτηση με αυτή που έπρεπε να εκτελέσει.

Αν ο χρήστης προσπαθήσει να πατήσει το κουμπί OK χωρίς τα 2 input fields να έχουν τιμή τότε δεν θα αρχικοποιήσει την σκηνή μας. Αν δεν αρχικοποιηθεί η σκηνή μας κανένα από τα άλλα κουμπιά δεν λειτουργεί. Με τον τρόπο αυτό περιορίζουμε τον χρήστη από το να υποπέσει σε άσκοπα σφάλματα και δημιουργείται μια καλύτερη αλληλεπίδραση με τον πρόγραμμα και το UI .

Ο αλγόριθμος είναι υλοποιημένος με τέτοιο τρόπο ώστε να υποστηρίζει μέχρι πλήθος 50 στην τελική λίστα.

Στη συνέχεια θα αναλυθεί ένα παράδειγμα με 25 αριθμούς σε κάθε μια από τις δυο λίστες.

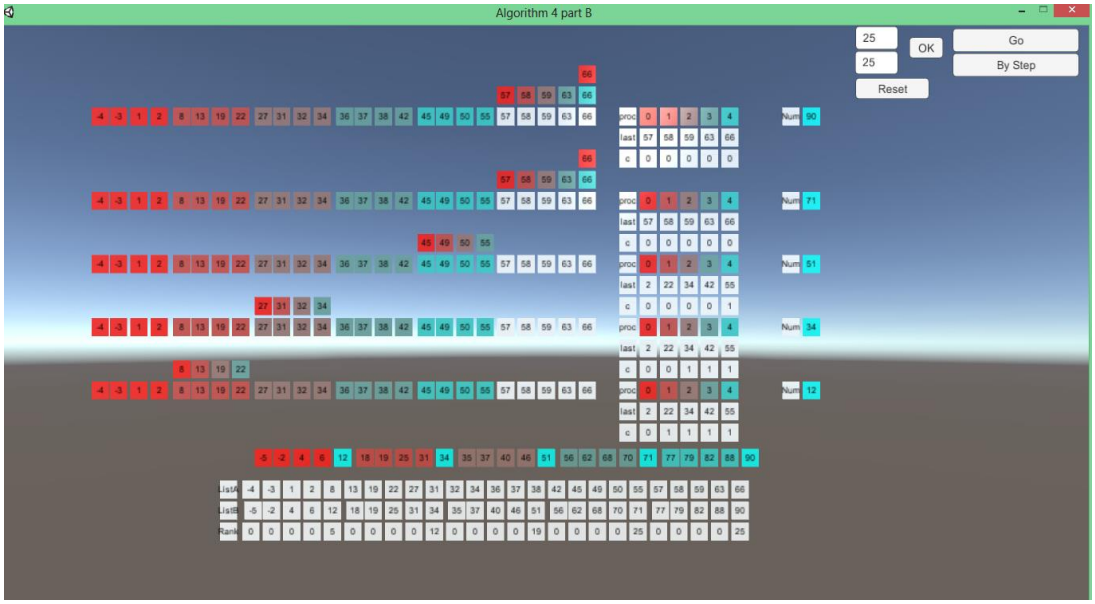
Ο χρήστης γράφει τον αριθμό 25 στην περίπτωση μας στο πρώτο input field (List A) και τον αριθμό 25 επίσης στο δεύτερο input field (List B) και ακολούθως πατά το Ok button. Στη συνέχεια παρατηρεί να εμφανίζεται στην οθόνη του η ακόλουθη εικόνα.(Σχήμα 6.4)



Σχήμα 6.4: Εμφάνιση επεξεργαστών και αριθμών

Όπως βλέπουμε εμφανίζονται 4 σειρές με 25 κουτιά με αριθμούς. Η δεύτερη σειρά είναι η λίστα A και η τρίτη σειρά η λίστα B που αποτελούνται από 25 αριθμούς όπως έχουμε επιλέξει οι οποίοι είναι ταξινομημένοι. Πιο κάτω ακριβώς βλέπουμε τον rank για κάθε αριθμό της λίστας B προς την λίστα A όπου στην αρχή έχουν αρχικοποιηθεί με μηδέν. Τέλος η πρώτη σειρά είναι μια επανάληψη της λίστας B όπου εκεί θα τρέξει ο αλγόριθμος μας.

Στη συνέχεια πατάμε το κουμπί By Step και βλέπουμε το παρακάτω αποτέλεσμα στην οθόνη μας. (Σχήμα 6.5)



Σχήμα 6.5: Βήμα 1

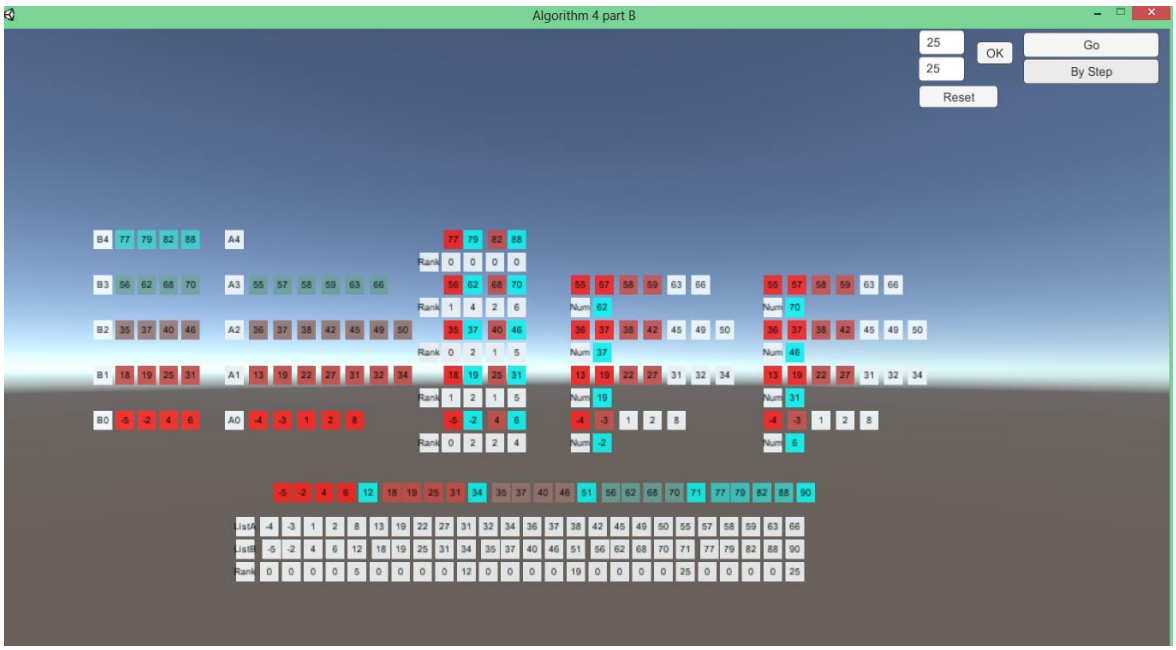
Τα χρώματα για τους επεξεργαστές βρίσκονται με τον ίδιο τρόπο όπως στον αλγόριθμο παράλληλης άθροισης (βλ. σελίδα 28).

Όπως και η δημιουργία και τοποθέτηση των κουτιών έχει γίνει με την εντολή instantiate δημιουργώντας το prefab που έχει επίσης αναφερθεί στον αλγόριθμο παράλληλης άθροισης (βλ. σελίδα 29) .

Οι επεξεργαστές μας έχουν αναλάβει από μια υπό-λίστα την οποία έχουν χρωματίσει και πηγαίνοντας στο τελευταίο στοιχείο της υπό-λίστας (που είναι χρωματισμένο γαλάζιο στο Σχήμα 6.5) κάνουν παράλληλη αναζήτηση όπως έχει εξηγηθεί στο προηγούμενο κεφάλαιο.
Πάνω από την λίστα αυτή φαίνεται αναλυτικά η διαδικασία για κάθε ένα από τους επεξεργαστές που ψάχνουν την θέση του στοιχείου στην λίστα Α. Στα δεξιά με γαλάζιο χρώμα αναγράφεται για πιο στοιχείο γίνεται η αναζήτηση.

Τέλος αφού έχει βρεθεί η θέση του παρατηρείται ότι τα rank στον κάτω πίνακα έχουν αλλάξει ανάλογα με την θέση των στοιχείων στην λίστα Α.

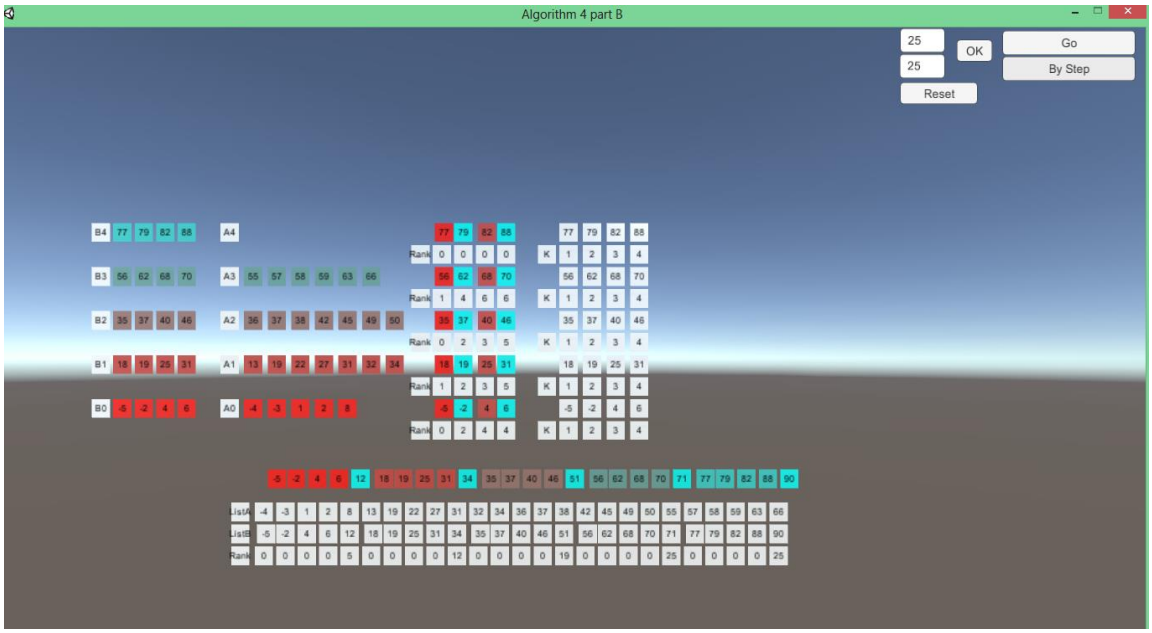
Στη συνέχεια πατώντας το By Step περνάμε στο επόμενο βήμα 2(Σχήμα 6.6)



Σχήμα 6.6: Βήμα 2

Στο Βήμα αυτό δημιουργούνται οι υπό-λίστες B_0 μέχρι το B_{p-1}(αριθμό επεξεργαστών -1) στην περίπτωση μας B_0-B_4 με τα υπόλοιπα στοιχεία της κάθε υπό-λίστας που δεν έχει ακόμη βρεθεί το rank τους. Στη συνέχεια δημιουργούνται και οι αντίστοιχες υπό-λίστες A_0-A_4 με τα στοιχεία της λίστας Α που είναι μικρότερα από τον αριθμό που είχε κάνει αναζήτηση ο κάθε επεξεργαστής στο προηγούμενο βήμα. Ουσιαστικά αποτελείται από τα στοιχεία που είναι ενδιάμεσα των rank των στοιχείων του προηγούμενου βήματος. (π.χ. για την υπό-λίστα Α1 έχουμε rank(1)=5 και rank(2)=12 θα ανήκουν σε αυτή τα στοιχεία στην θέση 6 μέχρι την θέση 12 της λίστας Α.

Ακολούθως η κάθε υπό-λίστας Β σπάζει και οι επεξεργαστές αναλαμβάνουν τα last της και με παράλληλη αναζήτηση βρίσκουν το rank τους στην αντίστοιχη υπό-λίστα Α.
Στη συνέχεια η εικόνα που βλέπουμε αλλάζει μόνη της και παίρνει την ακόλουθη μορφή. (Σχήμα 6.7)



Σχήμα 6.7: Βήμα εύρεσης K

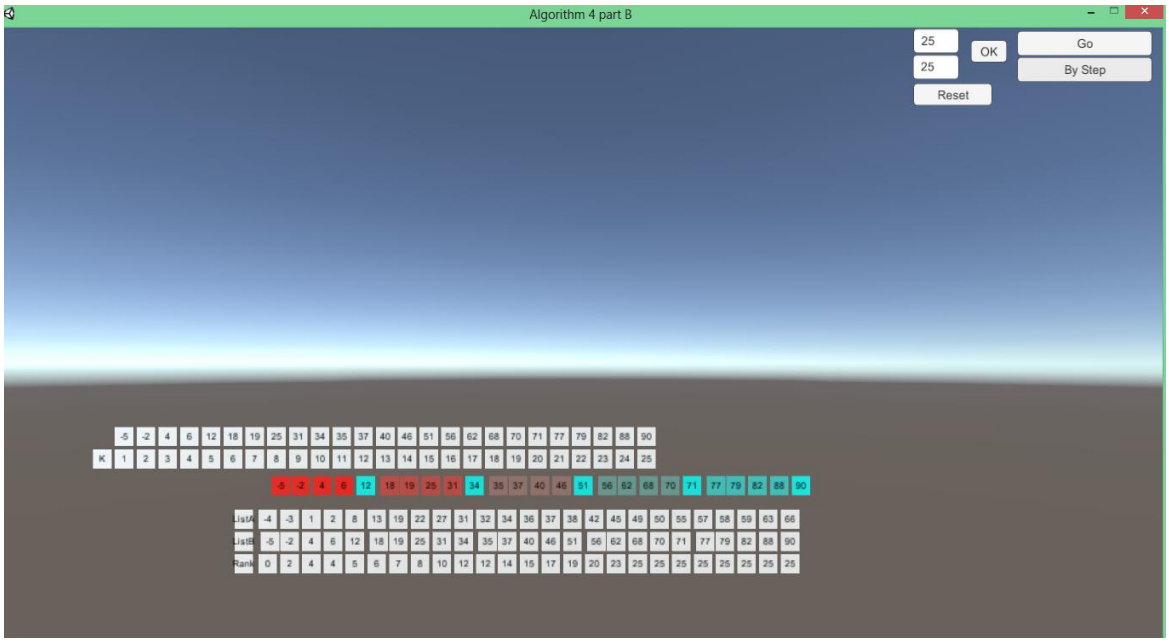
Ακολούθως βρίσκονται τα(προσωρινά) rank των τελευταίων αριθμών. Στο παράδειγμα μας η διαδικασία έχει εκτελεστεί μια φορά αναδρομικά σε μεγαλύτερα παραδείγματα μπορεί να χρειαζόταν περισσότερες μέχρι να βρεθούν τα rank κάθε στοιχείου.

Τέλος εμφανίζονται οι αριθμοί με τον αριθμό Κ αποκάτω τους. Αν ο αριθμός αυτός δεν είναι πολλαπλάσιο του $m^{1/2}$ τότε rank(b_k : A) = r(i) + rank(b_k : A_i) αλλιώς παραμένει το ίδιο.

Κώδικας για υπολογισμό τον αριθμών που είναι πολλαπλάσια του $m^{1/2}$:

```
int sqrt2 = (int)Math.Sqrt (sqrint - 1);

for (int k=1; k<sqrint-1; k++) {
    if (k % sqrt2 == 0) {
        temprank [k] = temprank [k] + temprank [(k + 1) / sqrt2];
        StartCoroutine (waittransfphase2 (k + (times - 1) * sqrint, temprank [k], 11.0f));
        }
}
```

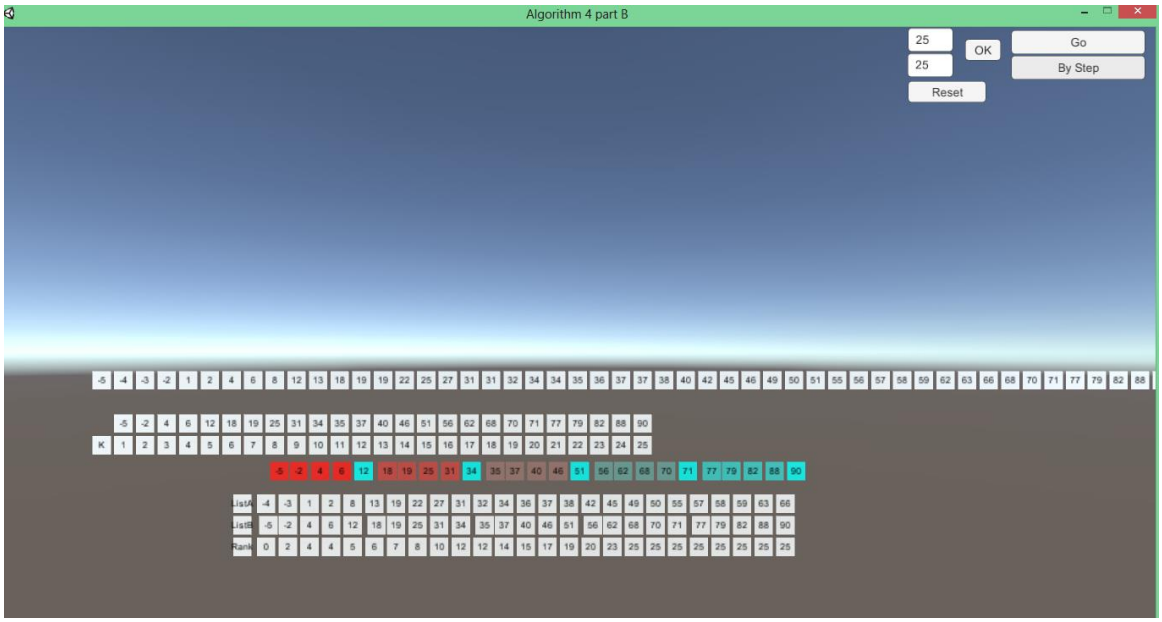
Πατάμε για τελευταία φορά το κουμπί By Step και παίρνουμε την εικόνα στο Σχήμα 6.8



Σχήμα 6.8: Βήμα εύρεσης Κ ολόκληρης λίστας

Αν παρατηρήσουμε γρήγορα την εικόνα που εμφανίζεται τα rank στο κάτω μέρος της εικόνας είναι τα ίδια με αυτά που βρήκαμε πριν. Μετά από μια μικρή καθυστέρηση όμως υπολογίζεται ξανά για τα πολλαπλάσια του Κ με το αρχικό όμως τώρα $m^{1/2}$ το τελικό rank για τον κάθε αριθμό.

Τέλος εμφανίζεται η τελική μας λίστα C όπως φαίνεται στην πιο κάτω εικόνα ταξινομημένη που αυτό ήταν το επιθυμητό αποτέλεσμα. (Σχήμα 6.9)



Σχήμα 6.9: Τελικό αποτέλεσμα

6.2 Αλγόριθμος Παράλληλης Ταξινόμησης

Στο σημείο αυτό γίνεται περιγραφή του προβλήματος ταξινόμησης καθώς και του σειριακού αλγόριθμου για την επίλυση του. Επίσης περιγράφεται και ο αλγόριθμος παράλληλης ταξινόμησης με λεπτομέρειες για την γραφική αναπαράσταση του που υλοποιήθηκε στην εργασία.

6.2.1 Πρόβλημα

Το πρόβλημα της συγχώνευσης είναι το εξής : *Δεδομένου ότι μας δίνεται μια μη-ταξινομημένη λίστα* $A = (a_1, a_2, \ldots, a_n)$ *, ζητείται να ταξινομηθεί κατά αύξουσα* $(a_1 <= a_2 <= \ldots <= a_n)$ *ή φθίνουσα* $(a_1 >= a_2 >= \ldots >= a_n)$ *σειρά.[2,5]*

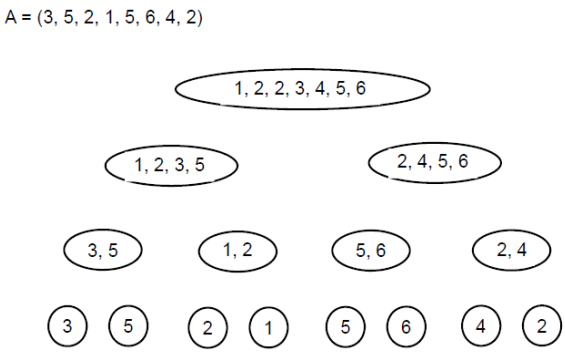
6.2.2 Σειριακός Αλγόριθμος

Για την σειριακή επίλυση του προβλήματος υπάρχουν πολλοί διαδεδομένοι αλγόριθμοι ταξινόμησης:

- Bubble-sort: $\Theta(n^2)$
- Quick-Sort: $O(n^2)$ αλλά κατά μέσο όρο O(nlogn)
- Merge-Sort: Θ(nlogn)

Όπου όπως παρατηρείται η καλύτερη σειριακή λύση χρειάζεται χρόνο O(nlogn).

Παράδειγμα:[2] (Σχήμα 6.10)



Σχήμα 6.10

6.2.3 Παράλληλος Αλγόριθμος

Στο υποκεφάλαιο αυτό περιγράφεται ο αλγόριθμος παράλληλης ταξινόμησης με λεπτομέρειες για την γραφική αναπαράσταση του που υλοποιήθηκε στην εργασία.

6.2.3.1 Περιγραφή Αλγορίθμου

Στον παράλληλο υπολογισμό το πρόβλημα παραμένει το ίδιο δηλαδή η ταξινόμηση μιας μη ταξινομημένης λίστας σε αύξουσα ή φθίνουσα σειρά. Στον αλγόριθμο αυτό θα χρησιμοποιηθεί ο αλγόριθμος παράλληλης συγχώνευσης ο οποίος περιγράφηκε πιο πριν στο κεφάλαιο αυτό. Είναι ουσιαστικά η παράλληλη εκδοχή του αλγόριθμου Merge sort[]

Έστω ότι έχουμε μια μη-ταξινομημένη λίστα με n αριθμούς, θα χρειαστεί να χρησιμοποιήσουμε $\Theta(n)$ επεξεργαστές. Ξεκινώντας από τα φύλλα του νοητού Δυαδικού δένδρου θα ανεβαίνουμε τα επίπεδα συγχωνεύοντας τα ζεύγη υπό-λιστών , όπου σε κάθε βήμα το πλήθος των αριθμών σε κάθε ζεύγος θα διπλασιάζεται μέχρι να φτάσουμε στο πλήθος της αρχικής μας λίστας.

Σε κάθε επίπεδο του δένδρου, έχουμε συνολικά n στοιχεία τα οποία είναι χωρισμένα σε υπό-λίστες ίσου μεγέθους. Για κάθε ζεύγος θα χρησιμοποιείται ο Αλγόριθμος Συγχώνευσης (ή ο Βέλτιστος Αλγόριθμος Συγχώνευσης για καλύτερο κόστος). Έτσι μόλις φτάσουμε στη ρίζα του δένδρου η λίστα μας θα έχει ταξινομηθεί.

Όλα τα βήματα εκτελούνται συγχρονισμένα και παράλληλα από τους επεξεργαστές για τον λόγο αυτό αναμφισβήτητα χρησιμοποιείται μοντέλο PRAM για την ανάπτυξη του αλγορίθμου. Πιο συγκεκριμένα χρησιμοποιείται το μοντέλο CREW-PRAM . Αυτό γίνεται για τον λόγο ότι σε κανένα σημείο δεν υπάρχει περίπτωση να παρουσιαστεί ταυτόχρονο γράψιμο σε κοινή κυψελίδα μνήμης , όμως θα συμβεί σίγουρα ταυτόχρονο διάβασμα. Ταυτόχρονο διάβασμα θα γίνει επειδή ο αλγόριθμος χρησιμοποιεί τον αλγόριθμο παράλληλης συγχώνευσης ο οποίος είναι σε μοντέλο CREW-PRAM άρα αναγκαστικά ο αλγόριθμος αυτός πρέπει να χρησιμοποιήσει το ίδιο περιοριστικό μοντέλο η ακόμη πιο χαλαρό.

Χρόνος :$T_{\Theta(n)}(n) = O(\log n \log\log n)$

Έστω ότι θα χρησιμοποιηθεί ο Βέλτιστος Αλγόριθμος Συγχώνευσης ο οποίος έχει $T=O(\log\log n)$ και $C=O(n)$. Έτσι σε κάθε επίπεδο, χρησιμοποιείται ο Βέλτιστος Αλγόριθμος Συγχώνευσης, παράλληλα για κάθε ζεύγος υπό-λιστών.

- Αν το μέγεθος των υπό-λιστών είναι $x <= n$, τότε $T_{\text{ζεύγους}}= O(\log\log x)$ και $C_{\text{ζεύγους}} = O(x)$
- Σε κάθε επίπεδο υπάρχουν n στοιχεία, οπότε έχουμε n/x υπό-λίστες
- Για κάθε επίπεδο $T_{\text{επιπέδου}}= O(\log\log n)$ και $C_{\text{επιπέδου}}= O(n)$
- Υπάρχουν $\Theta(\log n)$ επίπεδα

Άρα $\log n * \log\log n$ μας δίνει τον τελικό χρόνο αλγορίθμου που είναι $O(\log n \log\log n)$

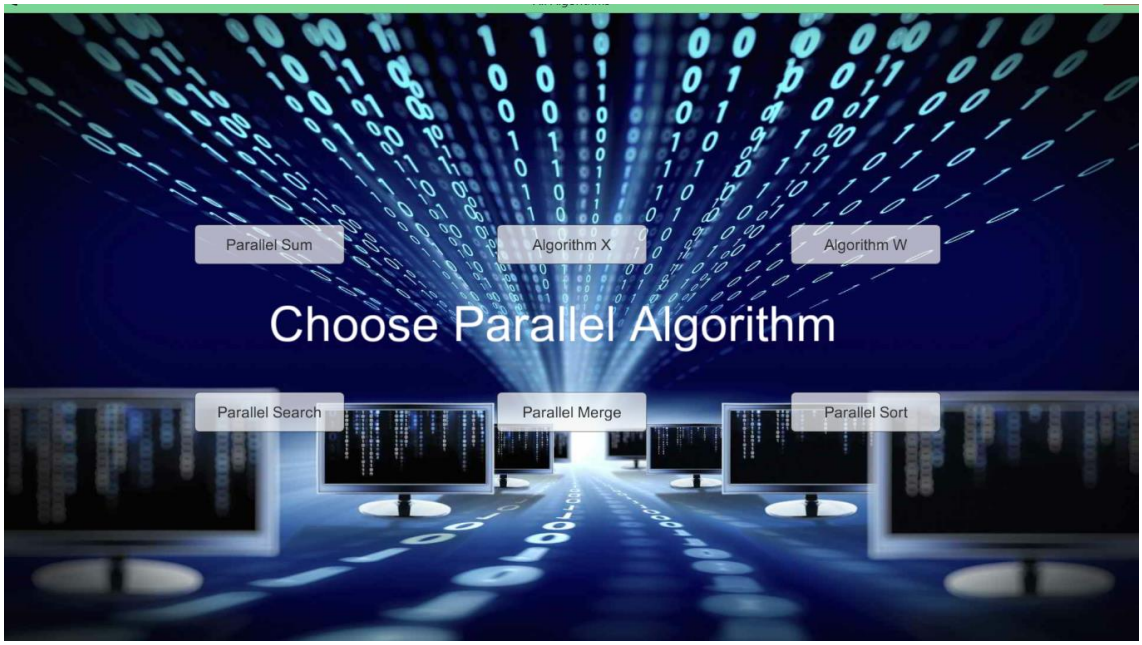
Χρησιμοποιούμε p επεξεργαστές άρα το κόστος μας είναι ο χρόνος επί το p όπου το $\log\log n$ τώρα γίνεται πολύ μικρό και παραλείπεται.

Κόστος :$C_{\Theta(n)}(n)= O(n \log n)$

Ο αλγόριθμος αυτός είναι βέλτιστος και αρκετά γρήγορος. Υπάρχει ο πιο γρήγορος βέλτιστος αλγόριθμος του Cole ο οποίος επιλύει το πρόβλημα με το ίδιο κόστος σε χρόνο $O(\log n)$.

6.2.3.2 Γραφική Αναπαράσταση Αλγορίθμου

Στο σημείο αυτό θα δοθούν λεπτομέρειες για το πώς υλοποιήθηκε ο αλγόριθμος παράλληλου merging στο Unity και βήμα προς βήμα πως τρέχει ο αλγόριθμος. Καταρχάς τρέχουμε το εκτελέσιμο . Η παρακάτω εικόνα (Σχήμα 6.11) είναι η πρώτη εικόνα που βλέπουμε μόλις τρέξουμε το εκτελέσιμο



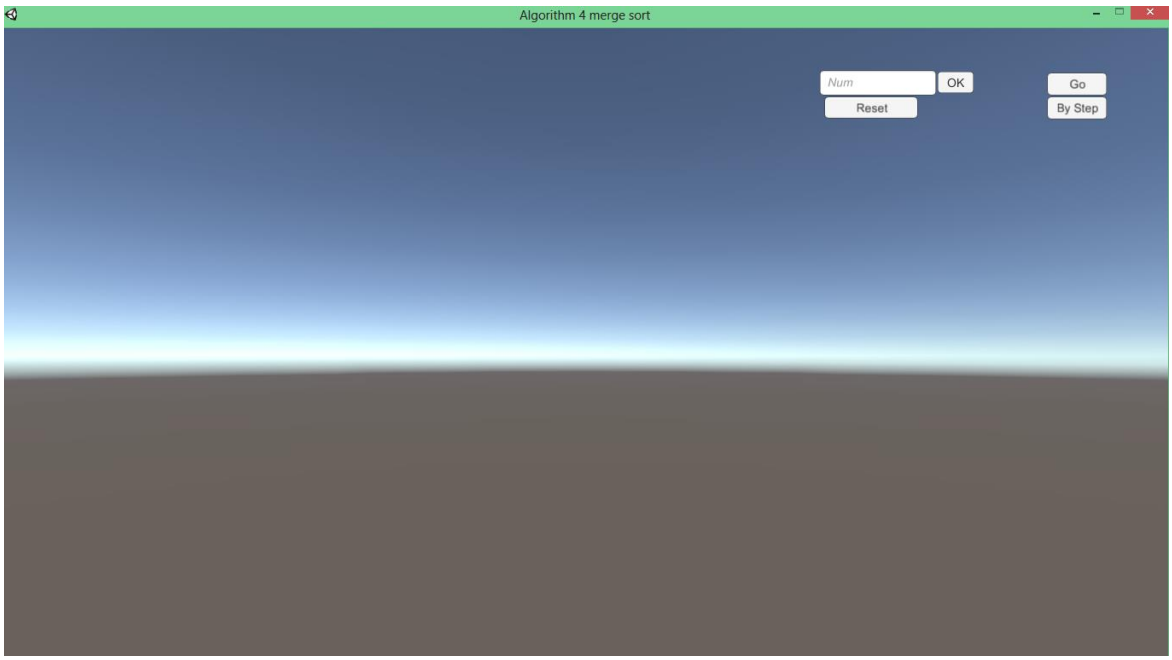
Σχήμα 6.11: Πρώτη εικόνα εκτελέσιμου

Στη συνέχεια επιλέγουμε το κουμπί του Parallel Sort όπου είναι ο αλγόριθμος ο οποίος μελετάμε στο συγκεκριμένο κεφάλαιο.
Ακολούθως εμφανίζεται η πιο κάτω αρχική εικόνα (Σχήμα 6.12)
Σε οποιαδήποτε σημείο που τρέχει το πρόγραμμα πατώντας το κουμπί ESC πηγαίνουμε στην αρχική επιλογή αλγορίθμων ή πατώντας το P γίνεται παύση.

Η αρχική εικόνα αποτελείται από:

- 1 input field το όποιο ο χρήστης περνά τιμές για το πλήθος αριθμών της λίστας
- Το κουμπί OK με το οποίο καταχωρείτε το πλήθος των αριθμών της λίστας στη μεταβλητή και ξεκινά η λειτουργία του αλγόριθμου
- Ένα κουμπί το κουμπί Reset το οποίο κάνει reset την σκηνή στην αρχική της μορφή για να εκτελέσει νέο παράδειγμα ο χρήστης

- Το κουμπί Go το οποίο τρέχει τον αλγόριθμο ολόκληρο με μια μικρή καθυστέρηση μεταξύ των βημάτων για να διακρίνονται αυτά.
- Το κουμπί By Step με το οποίο ο αλγόριθμος προχωρά στο επόμενο ακριβώς βήμα
- Το canvas που μέσα σε αυτό ήταν όλα τα πιο πάνω.



Σχήμα 6.12: Αρχική Εικόνα

Τα κουμπιά είναι ανεξάρτητα το ένα με το άλλο δηλαδή αν πρώτα εκτελέσουμε ένα βήμα με το κουμπί By Step πατώντας ακολούθως το Go θα ολοκληρωθεί ο αλγόριθμος από εκεί όπου είχε μείνει.

Το μέγεθος της κάμερας για μεγαλύτερο αριθμό *n* μικραίνει και μεγαλώνει αντίστοιχα ώστε να υποστηρίζει μέχρι 64 αριθμούς και οι αριθμοί αυτοί να παραμένουν διακριτοί στο μάτι.

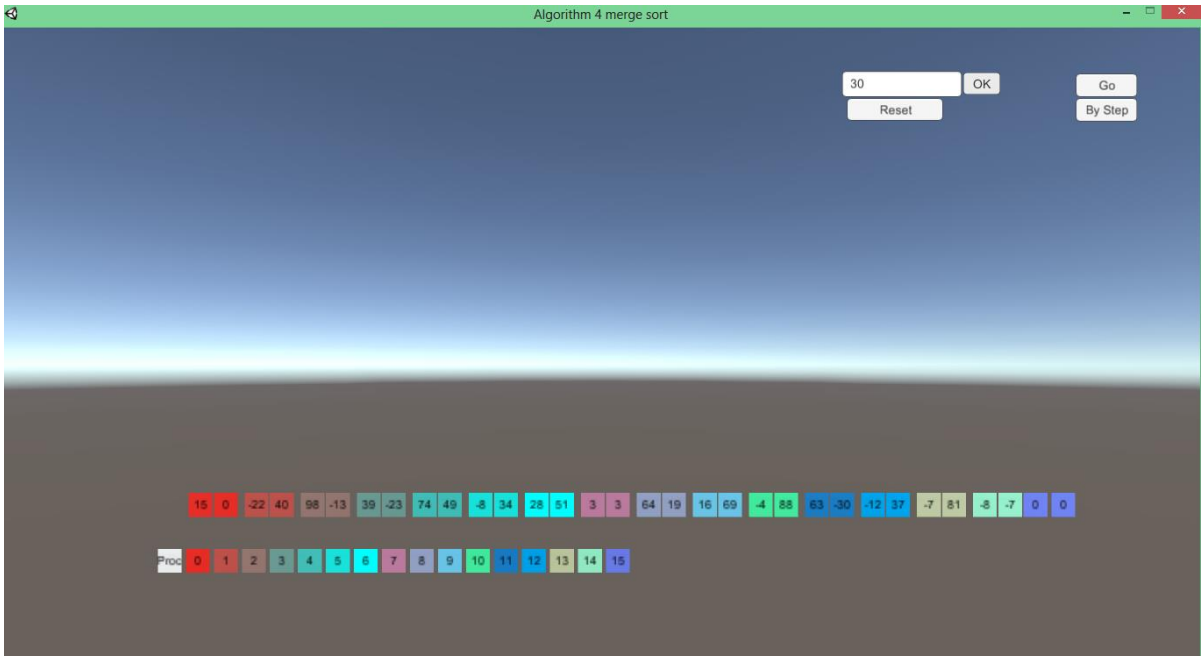
Για την δημιουργία του αλγόριθμου αυτού χρησιμοποιήθηκε ένα prefab το οποίο είχε μέσα ένα ακόμα αντικείμενο το οποίο αποτελούταν από ένα text mesh όπου στη συνέχεια σε αυτό το text mesh αποθηκεύεται και εμφανίζεται ο αριθμός.

Για την δημιουργία του κώδικα χρησιμοποιήθηκε μόνο ένα script το οποίο ήταν ενσωματωμένο στο canvas και το κάθε κουμπί καλούσε την αντίστοιχη συνάρτηση με αυτή που έπρεπε να εκτελέσει.

Αν ο χρήστης προσπαθήσει να πατήσει το κουμπί OK χωρίς το input field να έχει τιμή τότε δεν θα αρχικοποιήσει την σκηνή μας. Αν δεν αρχικοποιηθεί η σκηνή μας κανένα από τα άλλα κουμπιά δεν λειτουργεί. Με τον τρόπο αυτό περιορίζουμε τον χρήστη από το να υποπέσει σε άσκοπα σφάλματα και δημιουργείται μια καλύτερη αλληλεπίδραση με τον πρόγραμμα και το UI .

Στη συνέχεια θα αναλυθεί ένα παράδειγμα με 30 αριθμούς όπου θα δούμε πως υποστηρίζει και patching η λύση μας (βλ. σελίδα 35 patching στον αλγόριθμο παράλληλης άθροισης

Ο χρήστης γράφει τον αριθμό 30 στην περίπτωση μας στο input field και ακολούθως πατά το Ok button. Στη συνέχεια παρατηρεί να εμφανίζεται στην οθόνη του η ακόλουθη εικόνα. (Σχήμα 6.13)



Σχήμα 6.13: Εμφάνιση επεξεργαστών και αριθμών

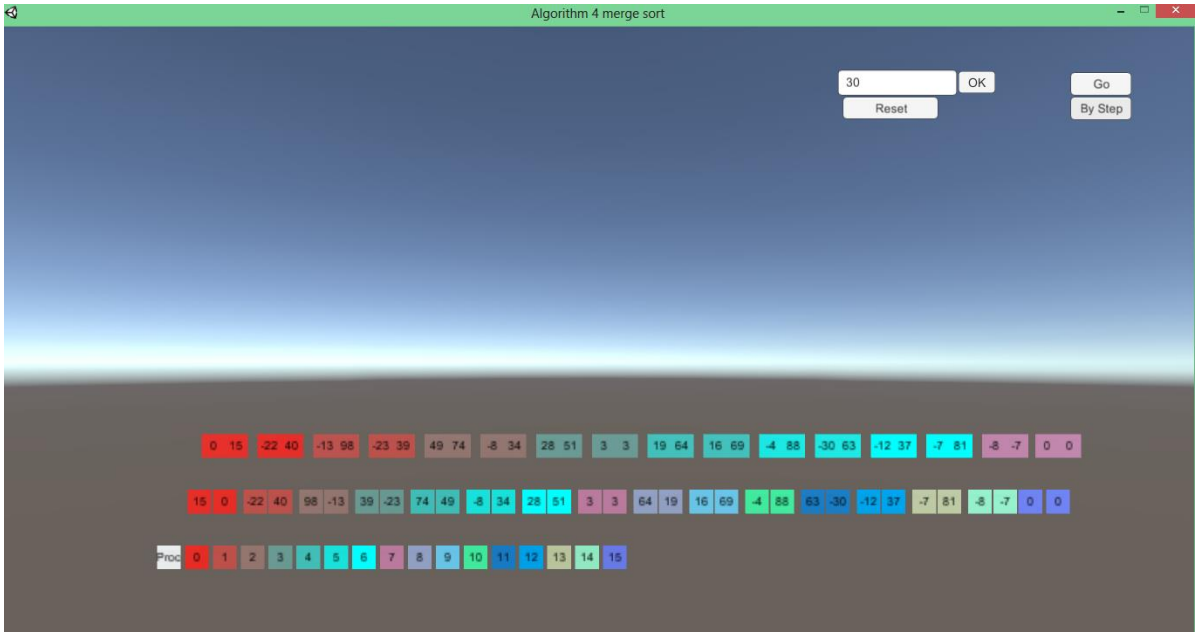
Όπως βλέπουμε εμφανίζονται 30+2 κουτιά με αριθμούς όπως έχουμε επιλέξει οι οποίοι είναι ταξινομημένοι. Τα 30 είναι οι αριθμοί που επιλέξαμε και τα 2 μηδενικά είναι αυτά που προστέθηκαν για την ομαλή λειτουργία του αλγόριθμου(pathing). Επίσης βλέπουμε το κουτί proc και δίπλα 15 κουτιά ένα για κάθε επεξεργαστή με το χρώμα που θα

χρωματίζει τις ενέργειες που εκτελεί. Παρατηρείτε ότι στο πρώτο βήμα οι επεξεργαστές έχουν αναλάβει από ένα ζεύγος αριθμών.

Τα χρώματα για τους επεξεργαστές βρίσκονται με τον ίδιο τρόπο όπως στον αλγόριθμο παράλληλης άθροισης (βλ. σελίδα 28).

Όπως και η δημιουργία και τοποθέτηση των κουτιών έχει γίνει με την εντολή instantiate δημιουργώντας το prefab που έχει επίσης αναφερθεί στον αλγόριθμο παράλληλης άθροισης (βλ. σελίδα 29) .

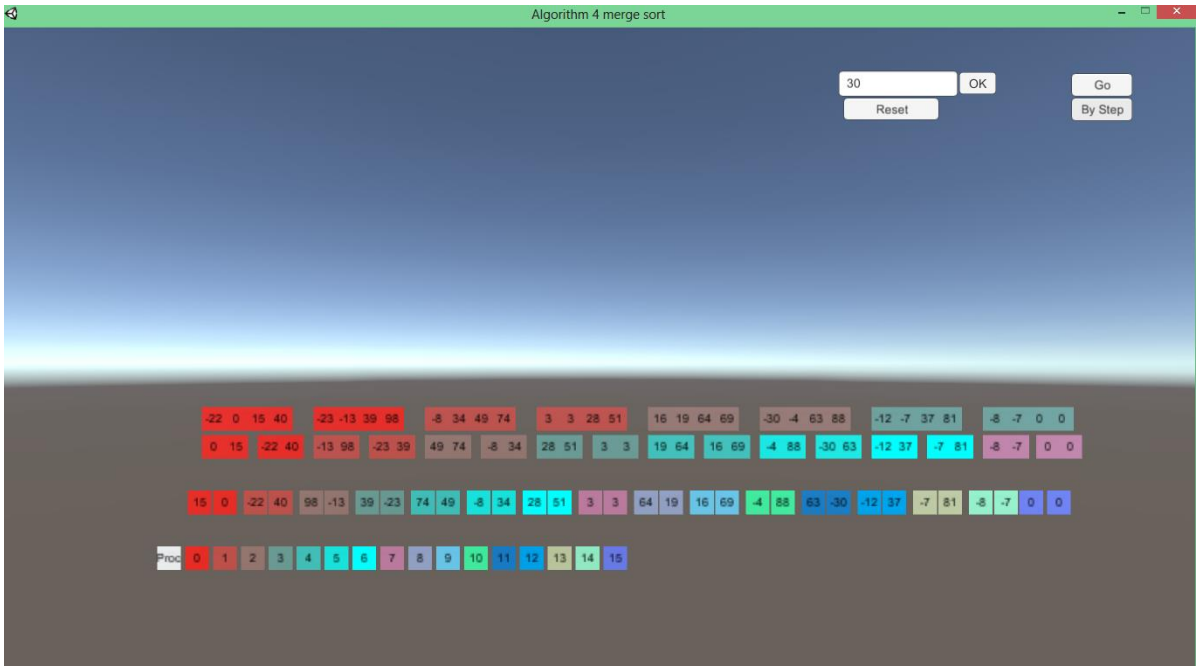
Στη συνέχεια πατάμε μια φορά το κουμπί By Step και βλέπουμε την ακόλουθη εικόνα να εμφανίζεται που είναι το ακόλουθο βήμα του αλγόριθμου. (Σχήμα 6.14)



Σχήμα 6.14: Βήμα 1

Βλέπουμε πώς τα ζεύγη αριθμών έχουν μείνει τα μισά δηλαδή από 16 έχουν μείνει 8. Επίσης οι υπό-λίστες έχουν διπλασιαστεί και έχουν από 2 ταξινομημένα στοιχεία τώρα. Στο βήμα αυτό συνεχίζουν μόνο οι πρώτοι 8 επεξεργαστές και οι άλλοι εκτελούν εντολή no-op.

Στη συνέχεια πατάμε μια φορά το κουμπί By Step και βλέπουμε την ακόλουθη εικόνα να εμφανίζεται που είναι το ακόλουθο βήμα του αλγόριθμου. (Σχήμα 6.15)

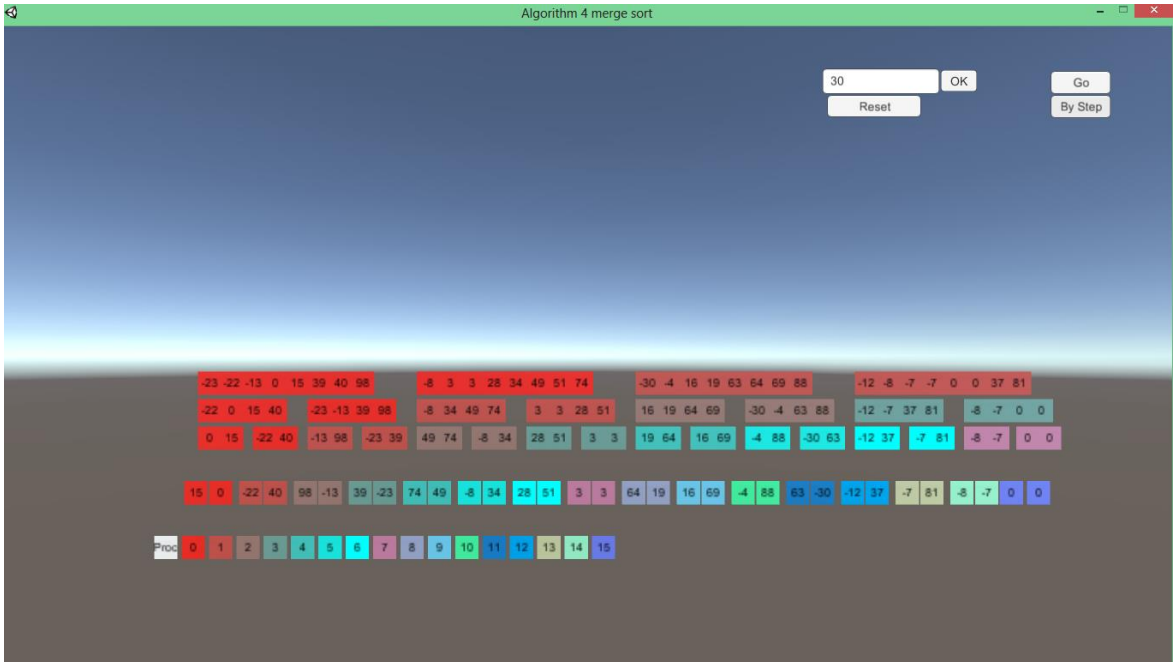


Σχήμα 6.15: Βήμα 2

Η διαδικασία παραμένει η ίδια. Δηλαδή τα ζεύγη αριθμών έχουν μείνει τα μισά, από 8 έχουν μείνει 4. Οι υπό-λίστες έχουν διπλασιαστεί και έχουν από 4 ταξινομημένα στοιχεία τώρα. Στο βήμα αυτό συνεχίζουν οι μισοί επεξεργαστές από πριν δηλαδή οι 4 και οι υπόλοιποι εκτελούν εντολή no-op.

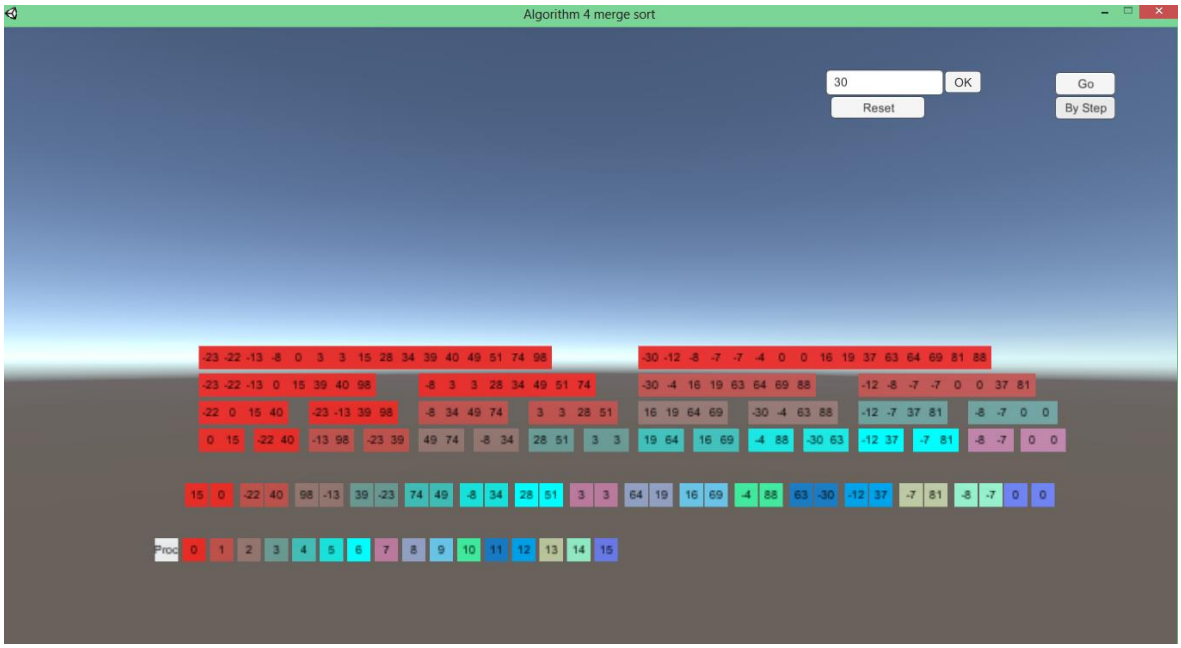
Στη συνέχεια πατάμε για ακόμα μια φορά το κουμπί By Step και βλέπουμε την ακόλουθη εικόνα να εμφανίζεται που είναι το επόμενο του αλγόριθμου. (Σχήμα 6.16)

Για ακόμα μια φορά παρατηρείται η ίδια συμπεριφορά. Δηλαδή τα ζεύγη αριθμών έχουν μείνει τα μισά, από 4 έχουν μείνει 2. Οι υπό-λίστες έχουν διπλασιαστεί και έχουν από 8 ταξινομημένα στοιχεία τώρα. Στο βήμα αυτό συνεχίζουν οι μισοί επεξεργαστές από πριν δηλαδή οι 2 και οι υπόλοιποι εκτελούν εντολή no-op.



Σχήμα 6.16: Βήμα 3

Στη συνέχεια πατάμε για ακόμα μια φορά το κουμπί By Step και βλέπουμε την ακόλουθη εικόνα (Σχήμα 6.17) να εμφανίζεται που είναι το προτελευταίο βήμα του αλγόριθμου.



Σχήμα 6.17: Προτελευταίο βήμα

Τέλος τα ζεύγη αριθμών έχουν μείνει μόνο 2. Οι υπό-λίστες έχουν διπλασιαστεί και έχουν από 16 ταξινομημένα στοιχεία τώρα. Στο βήμα αυτό συνεχίζει μόνο ο πρώτος επεξεργαστής και οι υπόλοιποι εκτελούν εντολή no-op.

Ακολούθως πατάμε για τελευταία φορά το κουμπί By Step και βλέπουμε την ακόλουθη εικόνα (Σχήμα 6.18) να εμφανίζεται που είναι το τελευταίο βήμα του αλγόριθμου και το τελικό αποτέλεσμα.

Σχήμα 6.18:Τελικό βήμα και αποτέλεσμα

Στο Σχήμα 6.18 παρατηρείται η αρχική μας λίστα με την διαφορά ότι τώρα είναι ταξινομημένη και συγχωνευμένη που αυτό ήταν το αποτέλεσμα το οποίο επιθυμούσαμε.

Κεφάλαιο 7

Αλγόριθμος W

7.1 Πρόβλημα Write-All στο F-PRAM

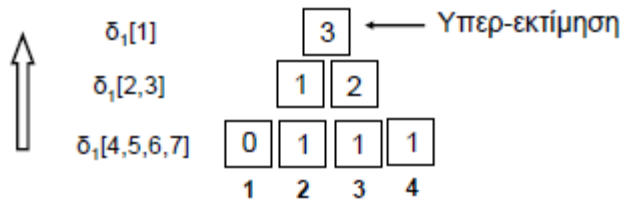
Το πρόβλημα το οποίο έχουμε να αντιμετωπίσουμε είναι το Write-All στο F-PRAM μοντέλο[]. Το F-PRAM όπως έχει αναφερθεί είναι ένα συγχρονισμένο μοντέλο όπου οι επεξεργαστές υπόκεινται σε σφάλματα κατάρρευσης. Το πρόβλημα Write-All αναφέρεται στην περίπτωση όπου δεδομένου μιας λίστας *n* στοιχείων όπου αυτά αρχικά έχουν την τιμή 0, οι τιμές των στοιχείων πρέπει να μετατραπούν σε 1. Θα χρησιμοποιηθούν *p* επεξεργαστές στο μοντέλο F-PRAM όπου δηλαδή μπορούν να καταρρεύσουν μέχρι $f<p$ επεξεργαστές.

7.2 Περιγραφή Αλγορίθμου

Για την επίλυση του προβλήματος Write-All θα χρησιμοποιήσουμε τον αλγόριθμο W[2,5,6]. Ο αλγόριθμος αυτός αποτελείται από 4 φάσεις όπου είναι αριθμημένες από το ένα μέχρι το τέσσερα και εκτελούνται διαδοχικά μέχρι να πάρουμε το αποτέλεσμα το οποίο θέλουμε. Στην πρώτη φορά που εκτελείται ο αλγόριθμος ξεκινά με την φάση 3 στη συνέχεια εκτελεί την φάση 4 και συνεχίζει εκτελώντας τον βρόχο με την φάση 1 μέχρι 4 ώστε όλα τα στοιχεία να έχουν γραμμένα την τιμή 1.

Στην πρώτη φάση του αλγόριθμου την W1, γίνεται εύρεση σφαλμάτων μέσω καταμέτρησης των ενεργών επεξεργαστών . Χρησιμοποιείται ένα διάνυσμα δ_1 το οποίο αναπαρίσταται ως ένα νοητό δυαδικό δένδρο (Σχήμα 7.1). Στο δένδρο αυτό καταμέτρησης μετρούνται οι επεξεργαστές οι οποίοι δεν έχουν καταρρεύσει. Οι επεξεργαστές ξεκινούν από τα φύλλα όπου τοποθετούνται με βάση τον προσωπικό τους αριθμό. Αφού τοποθετηθούν αναγράφουν την τιμή 1 σε αυτά και ανεβαίνουν το δένδρο τρέχοντας μια εκδοχή του αλγόριθμου παράλληλης άθροισης στο PRAM. Στην ρίζα του δένδρου θα έχουμε ένα αριθμό ο οποίος θα είναι ο υπολογιζόμενος αριθμός των ενεργών επεξεργαστών. Αυτός ο αριθμός είναι υπερεκτίμηση του πραγματικού αριθμού για τον λόγο ότι κάποιος επεξεργαστής όπου έγραψε την τιμή 1 σε αρχικό στάδιο μπορεί να κατέρριψε πριν την ολοκλήρωση της φάσης W1

Παράδειγμα



Σχήμα 7.1[2]

Στη δεύτερη φάση του αλγόριθμου την W2, οι επεξεργαστές κατανέμονται στα στοιχεία της λίστας με τέτοιο ισοζυγισμένο τρόπο ώστε να γίνουν οι εργασίες που δεν έχουν εκτελεστεί ακόμα. Αυτό γίνεται χρησιμοποιώντας ένα διάνυσμα δ_2 (Σχήμα 7.2), αναπαριστάμενο ως ένα νοητό δυαδικό δένδρο. Το δένδρο το δένδρο προόδου. Οι επεξεργαστές ξεκινούν από την ρίζα και κατεβαίνουν προς τα κάτω διαλέγοντας αν θα παν αριστερά η δεξιά ανάλογα με τα υπολειπόμενα μηδενικά της λίστας στα φύλλα του δένδρου. Αυτό γίνεται με το pnum το οποίο υπολογίζεται από την φάση W1 και τον αριθμό τον οποίο υπολογίζεται στην φάση W4 που βρίσκεται στην ρίζα και είναι η εκτίμηση των στοιχείων που έχουν τον αριθμό 1. Επίσης χρησιμοποιείται ο παρακάτω τύπος.

size = n

while (size ≠ 1) do

v[c1] = v[p]*(size/2 – δ3[c1]) / (size – δ3[p])

v[c2] = v[p] – v[c1]

size = size / 2

end do

(c1 αριστερό παιδί,

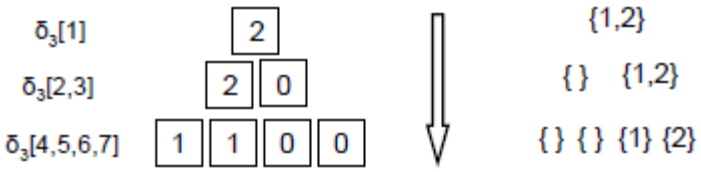
c2 δεξιό παιδί, p πατέρας,

v[i] αριθμός επεξεργαστών που πάνε στο i,

όπου αρχικά p = 1 (ρίζα) και v[1] = δ1[1])

Παράδειγμα

Έστω δύο ενεργοί επεξεργαστές με προσωπικούς αριθμούς 1 και 2 αντίστοιχα.



Σχήμα 7.2[2]

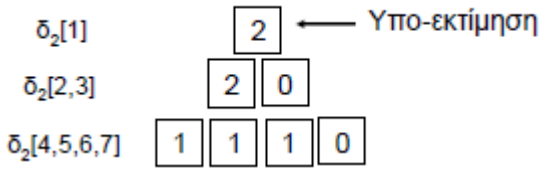
Στην τρίτη φάση του αλγόριθμου την W3, οι επεξεργαστές βρίσκονται στα φύλλα του δένδρου προόδου με βάση το πού τοποθετήθηκαν από την φάση W2 . Ο κάθε ενεργός επεξεργαστής αλλάζει την τιμή του στοιχείου που αντιστοιχεί στο φύλλο που βρίσκεται από 0 σε 1

Στην τέταρτη φάση του αλγόριθμου την W4 γίνεται η λεγόμενη αξιολόγηση προόδου. Δηλαδή οι ενεργοί επεξεργαστές ξεκινούν από τα φύλλα του δένδρου προόδου δ2 (Σχήμα 7.3), που βρέθηκαν στην προηγούμενη φάση W3 και ανεβαίνουν προς τα πάνω εκτελώντας μια εκδοχή του αλγόριθμου παράλληλης άθροισης σε PRAM, για να υπολογίσουν τον αριθμό των στοιχείων που έχουν την τιμή 1. Ο συνολικός αριθμός των στοιχείων που έχουν την τιμή 1 αποθηκεύεται στην ρίζα. Ο αριθμός αυτός είναι μια υποεκτίμηση του πραγματικού αριθμού για τον λόγο ότι μια εργασία η οποία έγινε στην

φάση W3 αν δεν υπάρχει επεξεργαστής να την μεταφέρει προς την ρίζα δεν θα υπολογιστεί.

Ο αλγόριθμος τερματίζει όταν στο τέλος της φάσης W4 η τιμή στην ρίζα είναι ίση με το πλήθος των στοιχείων της λίστας.

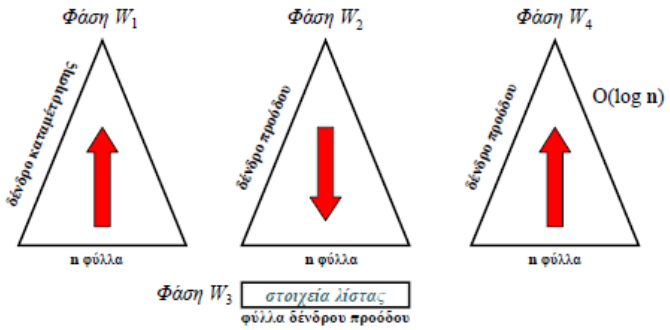
Παράδειγμα



Σχήμα 7.3[2]

Παρακάτω φαίνεται ο τρόπος εκτέλεσης του αλγορίθμου και η σειρά με την οποία εκτελούνται οι φάσεις του (Σχήμα 7.4):

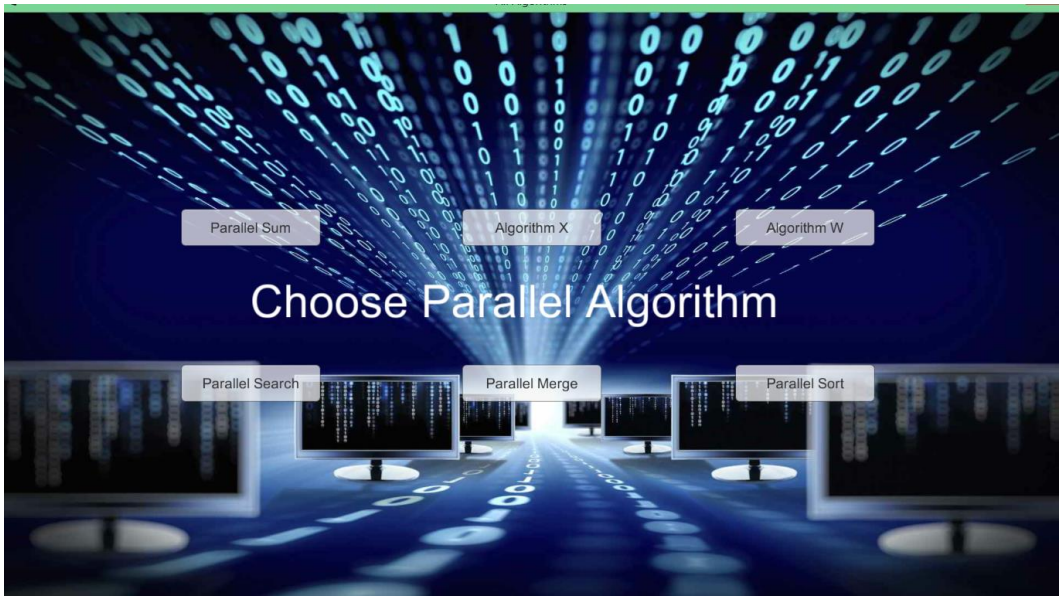
```
Processors i=1,…,n do in parallel
      Phase W3
      Phase W4
      while δ2[1] ≠ n do
            Phase W1
            Phase W2
            Phase W3
            Phase W4
      end while
end do
```



Σχήμα 7.4[2]

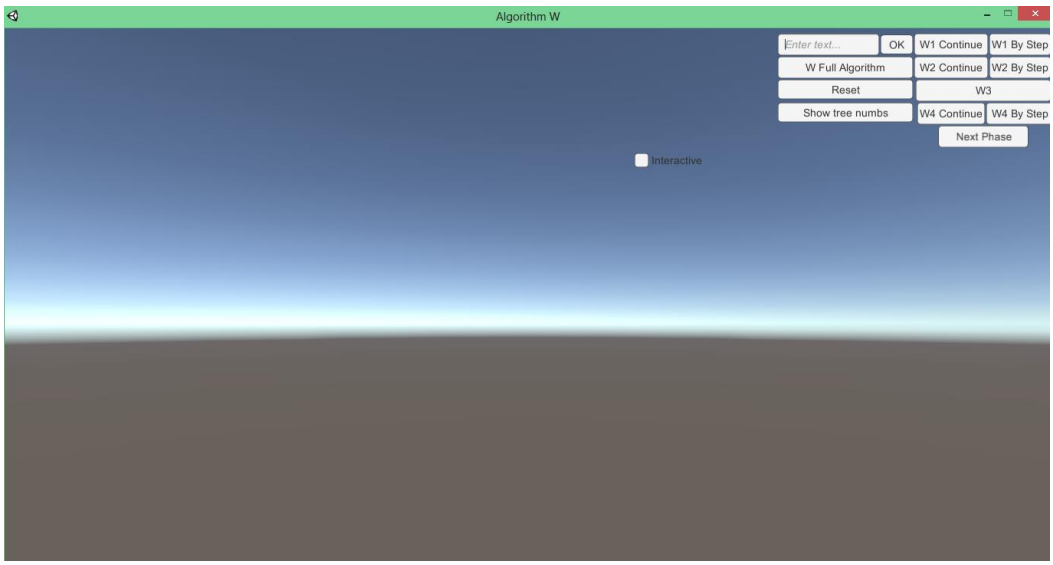
7.3 Γραφική Αναπαράσταση Αλγορίθμου

Τώρα θα δοθούν λεπτομέρειες για την υλοποίηση του αλγόριθμου W στο Unity και βήμα προς βήμα πως αυτός τρέχει. Καταρχάς τρέχουμε το εκτελέσιμο . Στο Σχήμα 7.5 είναι η πρώτη εικόνα που βλέπουμε μόλις τρέξουμε το εκτελέσιμο



Σχήμα 7.5: Πρώτη εικόνα εκτελέσιμου

Στη συνέχεια επιλέγουμε το κουμπί του Algorithm W όπου είναι ο αλγόριθμος ο οποίος μελετάμε στο συγκεκριμένο κεφάλαιο. Ακολούθως εμφανίζεται η πιο κάτω αρχική εικόνα (Σχήμα 7.6) Σε οποιαδήποτε σημείο που τρέχει το πρόγραμμα πατώντας το κουμπί ESC πηγαίνουμε στην αρχική επιλογή αλγορίθμων ή πατώντας το P γίνεται παύση.



Σχήμα 7.6: Αρχική Εικόνα

Η αρχική εικόνα αποτελείται από:

- 1 input field το οποίο ο χρήστης περνά το πλήθος αριθμών της λίστας.
- Το κουμπί OK με το οποίο καταχωρείτε το πλήθος αυτό.
- Ένα κουμπί το κουμπί Reset το οποίο κάνει reset την σκηνή στην αρχική της μορφή για να εκτελέσει νέο παράδειγμα ο χρήστης.
- Το κουμπί W Full Algorithm το οποίο τρέχει τον αλγόριθμο ολόκληρο με μια μικρή καθυστέρηση μεταξύ των βημάτων για να διακρίνονται αυτά.
- Το κουμπί Next phase με το οποίο ο αλγόριθμος προχωρά στην ακριβώς επόμενη φάση.
- Το κουμπί Show tree numbs όπου μετατρέπει το δ_1 (δένδρο της φάσης w1) σε δένδρο που δείχνει πως είναι αριθμημένες οι θέσεις του και με το πάτημα ξανά του κουμπιού επιστρέφει στην αρχική μορφή.
- 7 ακόμη κουμπιά 2 για κάθε μια από τις 4 φάσεις (εκτός της W3 η οποία έχει μόνο 1) τα οποία το ένα εκτελεί ολόκληρη την φάση και το άλλο εκτελεί μόνο το επόμενο της βήμα.
- Ένα toggle box το οποίο ο χρήστης μπορεί να επιλέξει αν οι επεξεργαστές θα καταρρέουν διαδραστικά ή τυχαία.
- Το canvas που μέσα σε αυτό ήταν όλα τα πιο πάνω.

Η κάμερα είναι σε ορθογραφική προβολή με size 10 ώστε να χωρεί στην οθόνη όσο περισσότερους αριθμούς μπορεί και οι αριθμοί αυτοί να παραμένουν διακριτοί στο μάτι.

Για την δημιουργία του αλγόριθμου αυτού χρησιμοποιήθηκε ένα prefab το οποίο είχε μέσα ένα ακόμα αντικείμενο το οποίο αποτελούταν από ένα text mesh όπου στη συνέχεια σε αυτό το text mesh αποθηκεύεται και εμφανίζεται ο αριθμός.

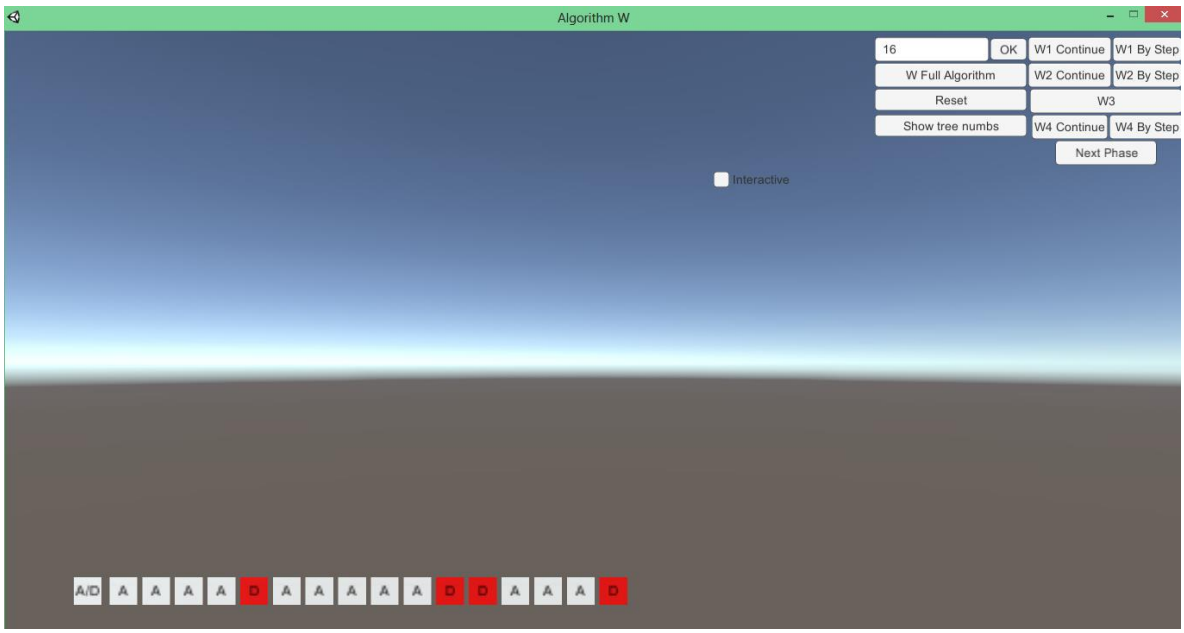
Για την δημιουργία του κώδικα χρησιμοποιήθηκαν δύο script. Το ένα ήταν ενσωματωμένο στο canvas και είχε τις συναρτήσεις για κάθε κουμπί , τα οποία καλούσαν την αντίστοιχη συνάρτηση μέσο αυτού και το άλλο ήταν υπεύθυνο για την κατάρρευση των επεξεργαστών με το αριστερό κλικ του ποντικιού .

Αν ο χρήστης προσπαθήσει να πατήσει το κουμπί OK χωρίς input field να έχει τιμή τότε δεν θα αρχικοποιήσει την σκηνή μας. Αν δεν αρχικοποιηθεί η σκηνή μας κανένα από τα άλλα κουμπιά δεν λειτουργεί. Επίσης τα κουμπιά των φάσεων πρέπει να πατιούνται με την σωστή σειρά αλλιώς δεν βγάζουν αποτέλεσμα. Με τον τρόπο αυτό περιορίζουμε τον χρήστη από το να υποπέσει σε άσκοπα σφάλματα και δημιουργείται μια καλύτερη αλληλεπίδραση με τον πρόγραμμα και το UI .

Ο αλγόριθμος είναι υλοποιημένος με τέτοιο τρόπο ώστε να υποστηρίζει μέχρι πλήθος 16 αριθμών στην λίστα και των αντίστοιχων επεξεργαστών.

Στη συνέχεια θα αναλυθεί ένα παράδειγμα με 16 αριθμούς με την επιλογή interactive να μην είναι επιλεγμένη έτσι τα σφάλματα μας θα είναι τυχαία. Σε οποιοδήποτε σημείο του αλγόριθμου πατώντας στο D ή στο A του επεξεργαστή μπορούμε να ενεργοποιήσουμε ή να καταρρεύσουμε τον αντίστοιχο επεξεργαστή. Θα αναλυθούν τα βήματα με φάσεις και όχι με το By Step κάθε φάσης

Ο χρήστης γράφει τον αριθμό 16 στην περίπτωση μας στο input field και ακολούθως πατά το Ok button. Στη συνέχεια παρατηρεί να εμφανίζεται στην οθόνη του η εικόνα στο Σχήμα 7.7 .



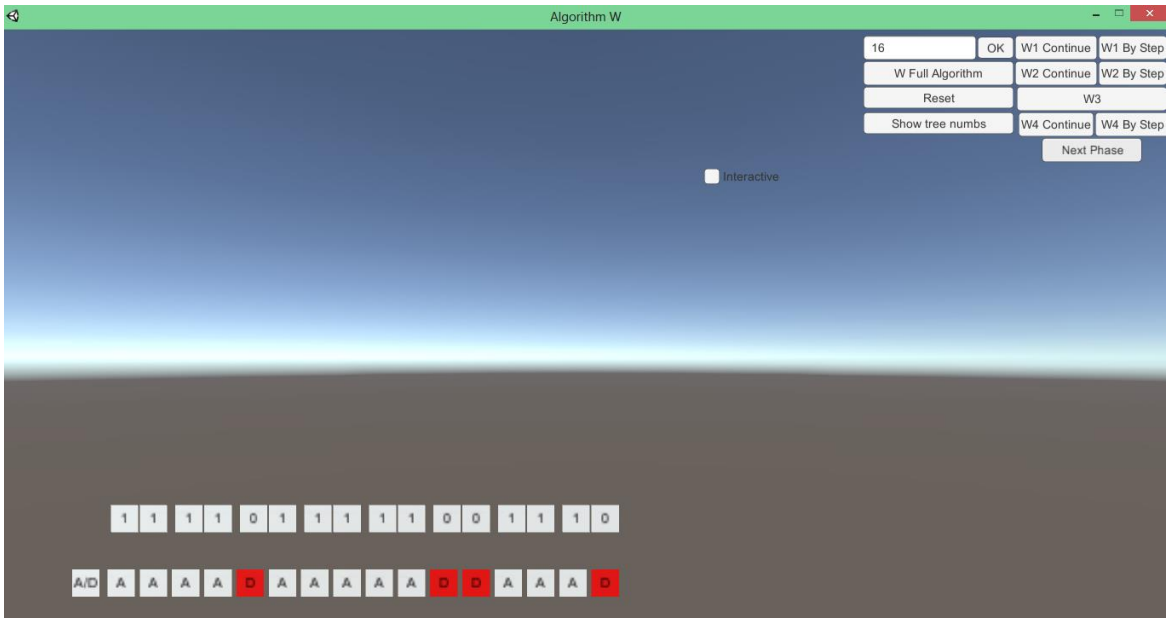
Σχήμα 7.7: Εμφάνιση επεξεργαστών και κατάστασης τους

Όπως βλέπουμε εμφανίζεται ο αριθμός των επεξεργαστών που έχουμε επιλέξει και τυχαία βλέπουμε κάποιους επεξεργαστές οι οποίοι γράφουν την τιμή D και είναι κόκκινοι. Αυτοί είναι οι επεξεργαστές που έχουν καταρρεύσει στο συγκεκριμένο βήμα.

Μόλις ξεκινήσει η εκτέλεση του αλγορίθμου, πριν εισέλθει στο βρόγχο, εκτελούνται πρώτα οι φάσεις W3 και W4. Έτσι η υλοποίηση θα ξεκινήσει με την υλοποίηση της φάσης W3 για να συνεχίσει με τις φάση W4 και ακολούθως θα μπει στην εκτέλεση του βρόγχου μέχρι να πάρουμε το τελικό επιθυμητό αποτέλεσμα. Άρα για την επόμενη φάση θα πατήσουμε τώρα το Next Phase ή το W3 για την εκτέλεση της.

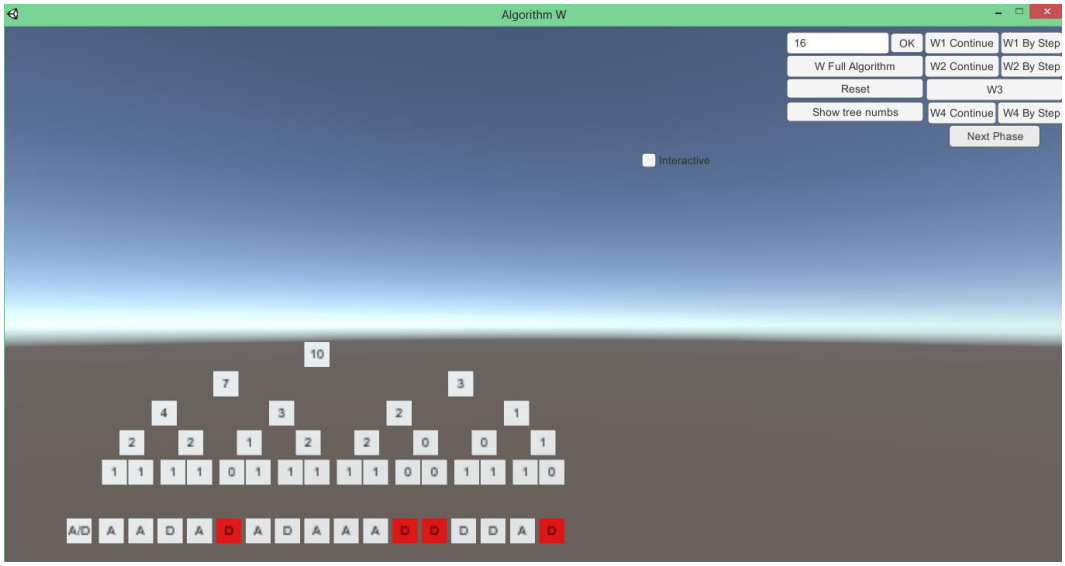
Εδώ εκτελείται η φάση W3(Σχήμα 7.8 όπου ο κάθε ενεργός επεξεργαστής γράφει την τιμή 1 στα φύλλα του νοητό δένδρο το οποίο δημιουργείται ανάλογα με τον προσωπικό του αριθμό.

Για το δένδρο αυτό γίνεται χρήση του διανύσματος δ_2, δηλαδή του δέντρου προόδου. Όπως φαίνεται στο παράδειγμα όλοι οι ενεργοί επεξεργαστές κατάφεραν να γράψουν την τιμή 1 χωρίς να καταρρεύσουν .



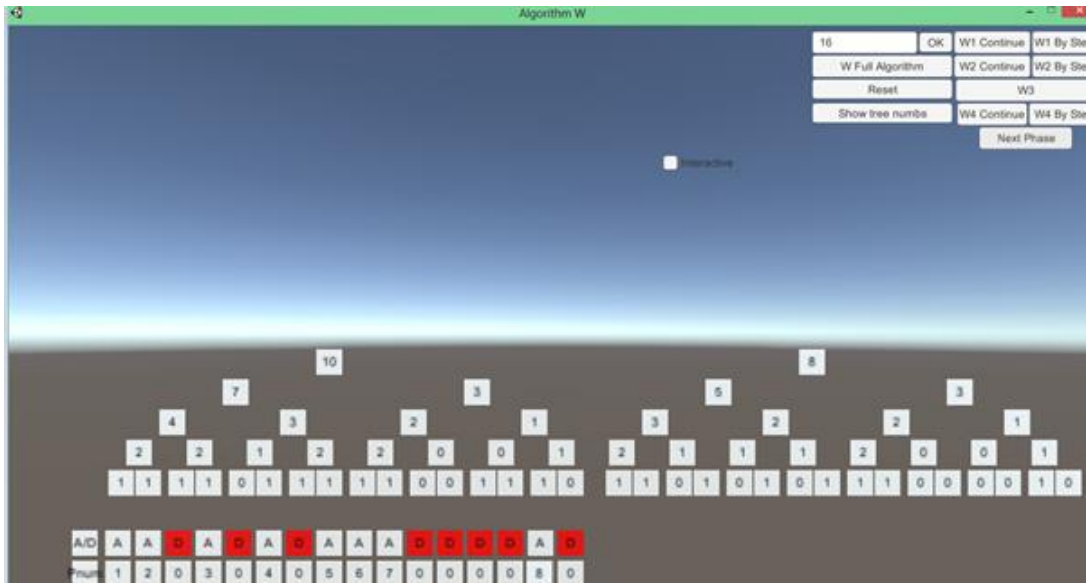
Σχήμα 7.8: Φάση W3

Πατώντας τώρα το Next Phase ή το W4(ή το By Step) και εκτελείται η επόμενη φάση .



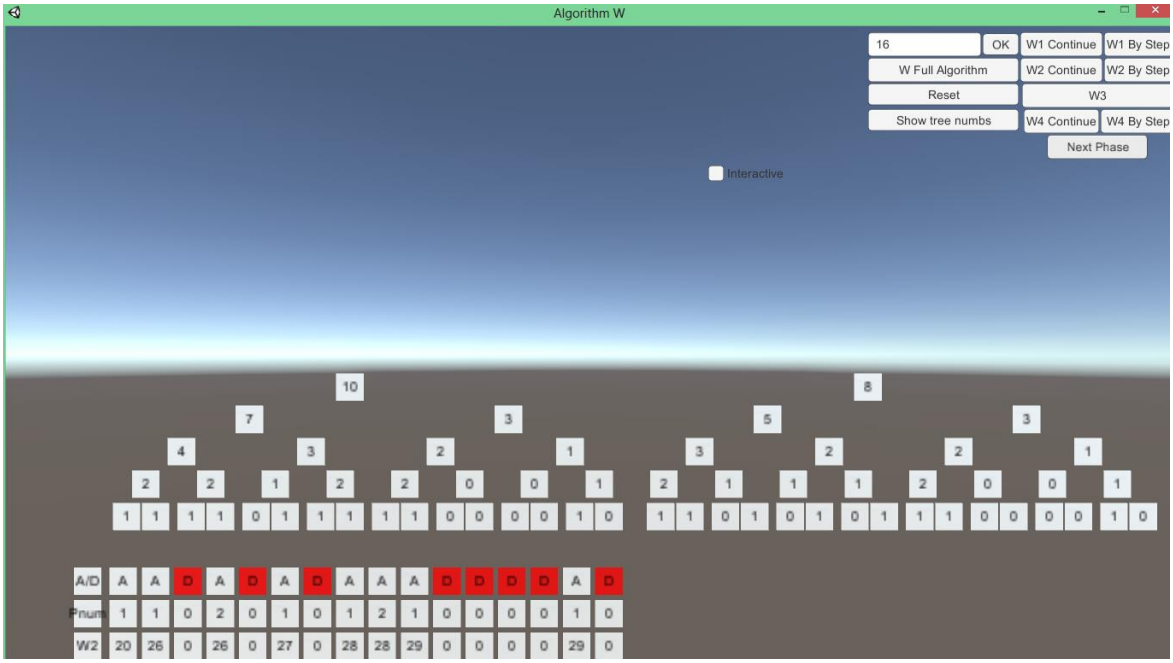
Σχήμα 7.9: Φάση W4

Στην φάση W4 (Σχήμα 7.9) οι επεξεργαστές "κτίζουν" το δένδρο προόδου δ2 προς τα πάνω. Οι επεξεργαστές που έμειναν ενεργοί από την προηγούμενη φάση και έγραψαν την τιμή 1 εκτελούν μια εκδοχή παράλληλης άθροισης μέχρι την ρίζα. Όμως βλέπουμε πώς υπάρχει η υποεκτίμηση αφού έχουν υπολογιστεί μόνο 10 εργασίες στην ρίζα ενώ αν παρατηρήσουμε έχουν ολοκληρωθεί 12 εργασίες με την φάση W3. Αυτό γίνεται γιατί αν προσέξουμε καταρρεύσαν ακόμη 4 επεξεργαστές οι οποίοι φαίνεται ότι κοκκινίζουν στην επόμενη εικόνα. Αν ο αριθμός είναι ίσος με το n ο αλγόριθμος ολοκληρώνεται. Όπως παρατηρείται στην ρίζα δεν είναι ο αριθμός 16 άρα ο αλγόριθμος δεν έχει ολοκληρωθεί.Πατώντας τώρα το Next Phase ή το W1(ή το By Step) και εκτελείται η επόμενη φάση.



Σχήμα 7.10: Φάση W1

Στην φάση W1 (Σχήμα 7.10) οι επεξεργαστές “κτίζουν” το δένδρο καταμέτρησης δ1. Οι αριθμοί τοποθετούνται με στα φύλλα με βάση τον προσωπικό τους αριθμό και δημιουργούν το δένδρο προς τα πάνω χρησιμοποιώντας μια εκδοχή της παράλληλης άθροισης Ο αριθμός που υπάρχει στην ρίζα είναι υπερεκτίμηση των ενεργών επεξεργαστών, αλλά στο παράδειγμα μας δεν έχει καταρρεύσει κάποιος στο βήμα αυτό για τον λόγο αυτό είναι σωστός. Επίσης υπολογίζεται το pnum του κάθε αριθμού όπου και δημιουργήθηκε μια γραμμή κάτω από το A/D με αυτό για κάθε επεξεργαστή, το οποίο θα χρησιμοποιήσει η επόμενη φάση του αλγόριθμου η W2. Πατώντας τώρα το Next Phase ή το W2(ή το By Step) εκτελείται η επόμενη φάση.



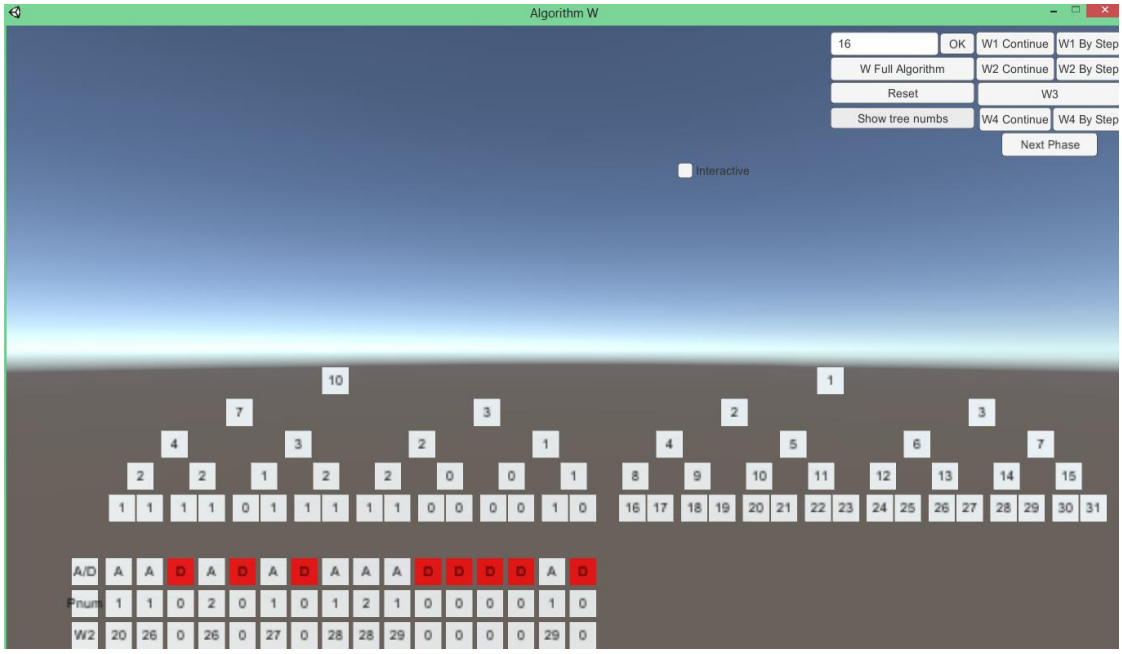
Σχήμα 7.11: Φάση W2

Στην φάση W2 (Σχήμα 7.11) γίνεται η ανάθεση των επεξεργαστών με την χρήση ενός νοητού δυαδικού δένδρου. Το νοητό δυαδικό δένδρο αυτό είναι το ίδιο με το διάνυσμα δ2 το δένδρο προόδου το όποιο δημιουργήθηκε από την φάση W4. Όμως για να μην αλλοιωθούν οι τιμές του διανύσματος αυτού δημιουργείται ένα προσωρινό διάνυσμα δ3 με αρχικές τιμές τις ίδιες με το δ2. Οι επεξεργαστές ξεκινούν από την ρίζα και κατεβαίνουν προς τα κάτω μέχρι να ανατεθούν ισοζυγισμένα στα φύλλα τα οποία δεν έχουν πάρει την τιμή 1 ακόμη. Αυτό γίνεται με την βοήθεια των τιμών που παίρνουμε από το δ3 για τα υπολειπόμενα μηδενικά και το δ1 για την εκτίμηση των ενεργών επεξεργαστών με το pnum τους. Παρατηρείται πως κάτω από το pnum έχει δημιουργηθεί

το W2 το οποίο είναι η τελική θέση στο δένδρο η οποία θα ανατεθούν.Πιο κάτω φαίνεται κομμάτι του κώδικα το οποίο δημιουργήθηκε για τον τρόπο ανάθεσης των επεξεργαστών.

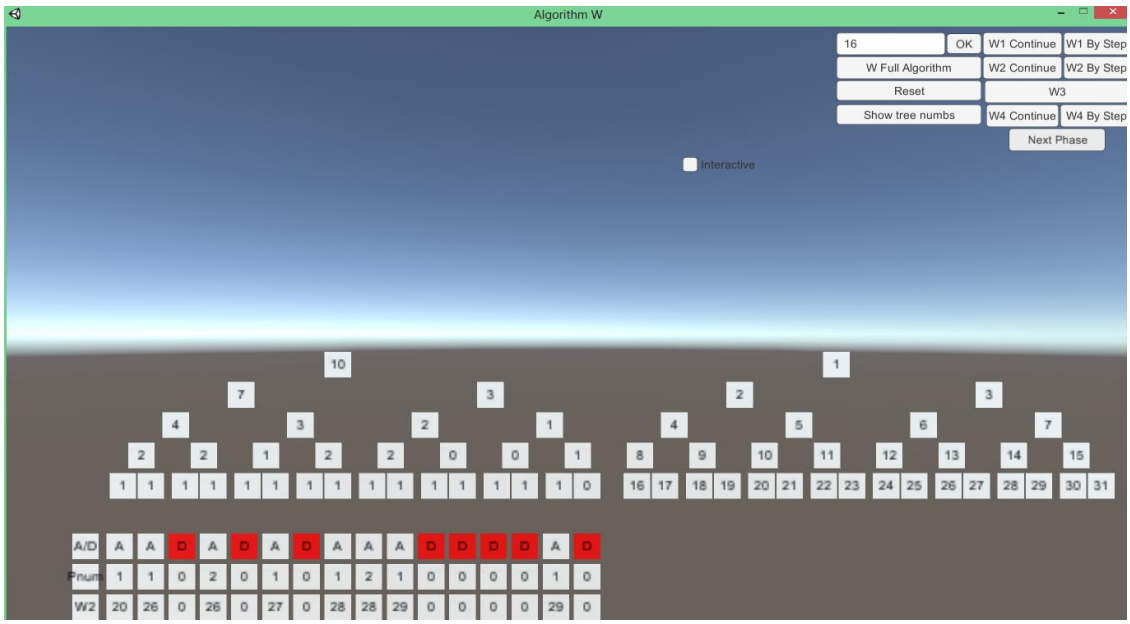
```
child1 = currentv2 [i] * 2;
child2 = child1 + 1;
if (done [child1] + done [child2] == 0)
        auxv2 [i, child1] = 0;
else
        auxv2 [i, child1] = Mathf.RoundToInt ((float)(auxv2 [i, currentv2 [i]] * done [child1]) / (done [child1] + done [child2]));

auxv2 [i, child2] = auxv2 [i, currentv2 [i]] - auxv2 [i, child1];

taskobjects [child1].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + auxv2 [i, child1];
taskobjects [child2].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + auxv2 [i, child2];

if (w2size == auxv2 [i, currentv2 [i]])
        count [child1] = 0;
else
        count [child1] = Mathf.RoundToInt ((float)(count [currentv2 [i]] * (Mathf.RoundToInt ((float)(w2size / 2)) - auxv2 [i, child1])) / (w2size - auxv2 [i, currentv2 [i]]));

count [child2] = count [currentv2 [i]] - count [child1];
if (pnum [i] <= count [child1])
        currentv2 [i] = child1;
else {
        currentv2 [i] = child2;
        pnum [i] = pnum [i] - count [child1];
StartCoroutine (waittransf (pnums [i], pnum [i], se));
        }
StartCoroutine (waittransf (w2tree [i], currentv2 [i], se));
```

Στη συνέχεια πατούμε το κουμπί Show tree numbs για να δούμε πια θέση αντιστοιχεί με αυτές που βρήκε το W2. (Σχήμα 7.12)



Σχήμα 7.12: Show tree numbs

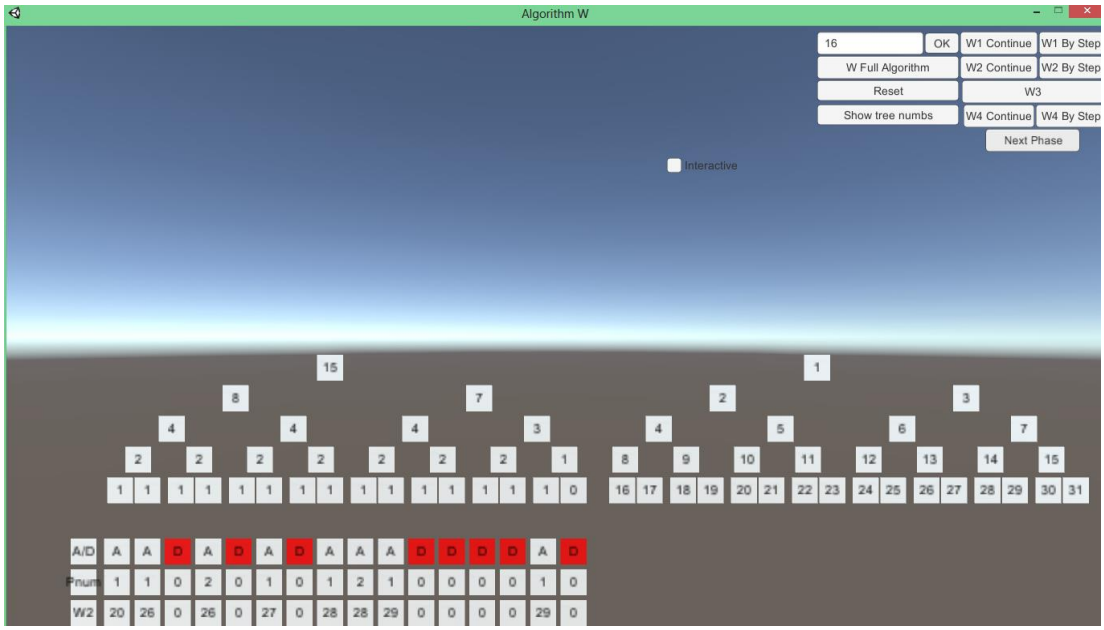
Έτσι εμφανίζονται οι αντίστοιχες αριθμημένες θέσεις αρχίζοντας την αρίθμηση με τον αριθμό 1 από την ρίζα μέχρι το (2*n-1*). Η αρίθμηση γίνεται κατά πλάτος του δένδρου. Με το πάτημα του Next Phase ή του W3 εκτελείται η επόμενη φάση .



Σχήμα 7.13: Φάση W3

Ανάλογα με το πού τοποθετήθηκαν οι επεξεργαστές από την φάση W2 πάνε και γράφουν την τιμή 1 στα αντίστοιχα φύλλα (Σχήμα 7.13) όπως αναφέραμε και στην πρώτη εκτέλεση του W3

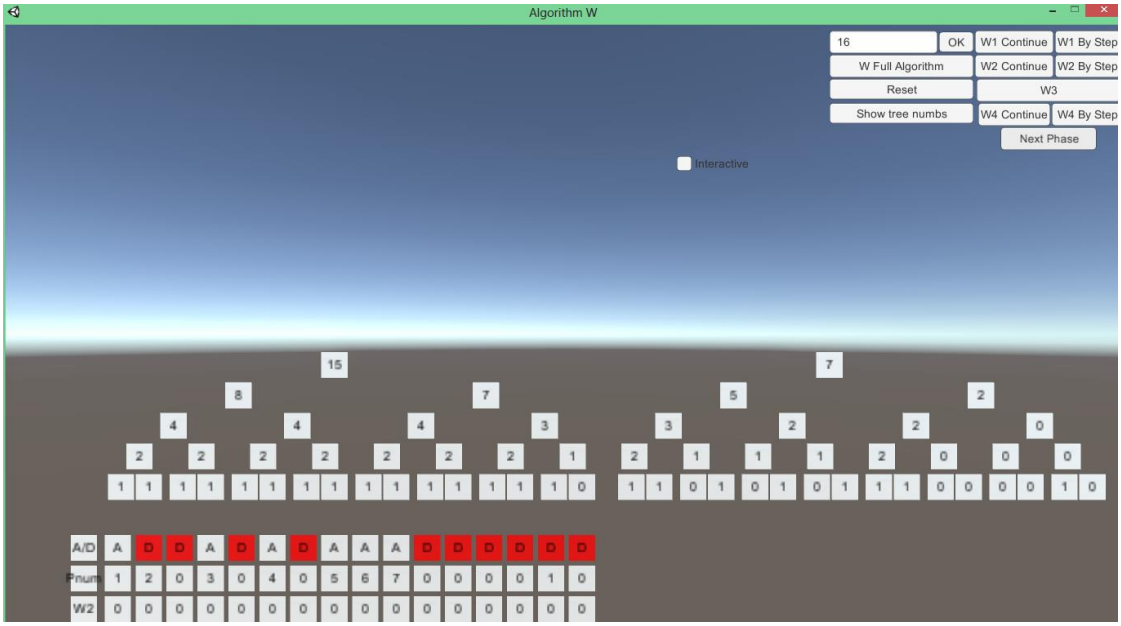
Με το πάτημα του Next Phase ή του W4(By Step) εκτελείται η επόμενη φάση .



Σχήμα 7.14: Φάση W4

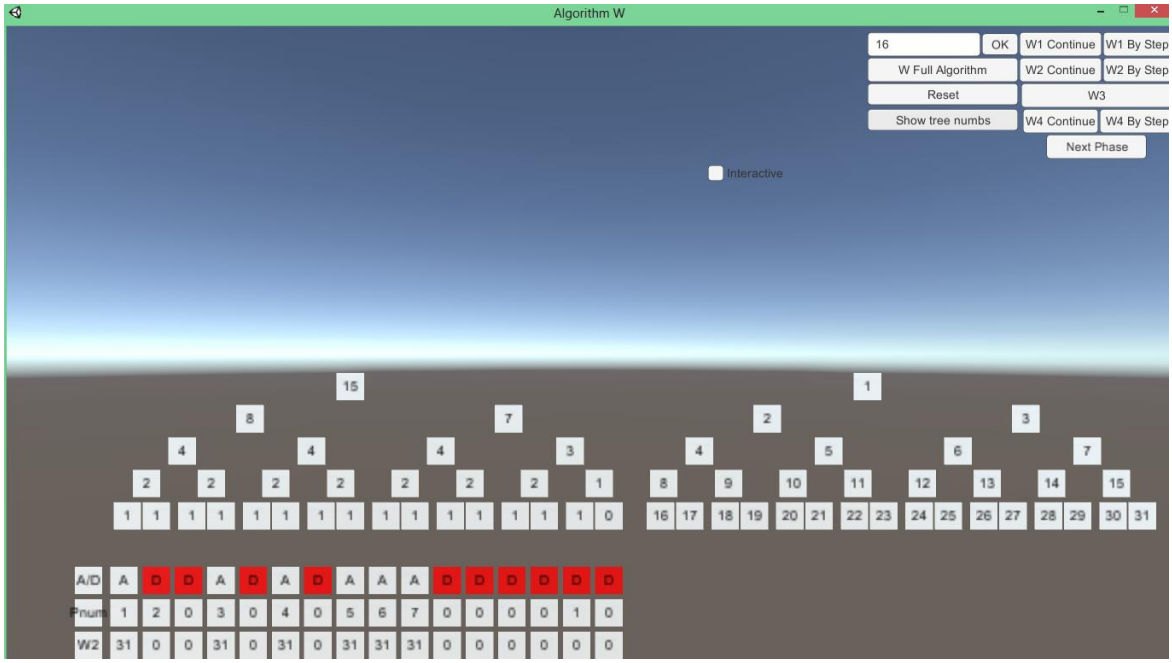
Όπως και πριν η φάση W4 εκτελείται με τον ίδιο τρόπο και τώρα βλέπουμε στην ρίζα του δένδρου τον αριθμό 15(Σχήμα 7.14). Άρα ο αλγόριθμος μας δεν έχει ολοκληρωθεί και πρέπει να εκτελέσει ξανά τον βρόγχο .

Με το πάτημα του Next Phase ή του W1(By Step) εκτελείται η επόμενη φάση .



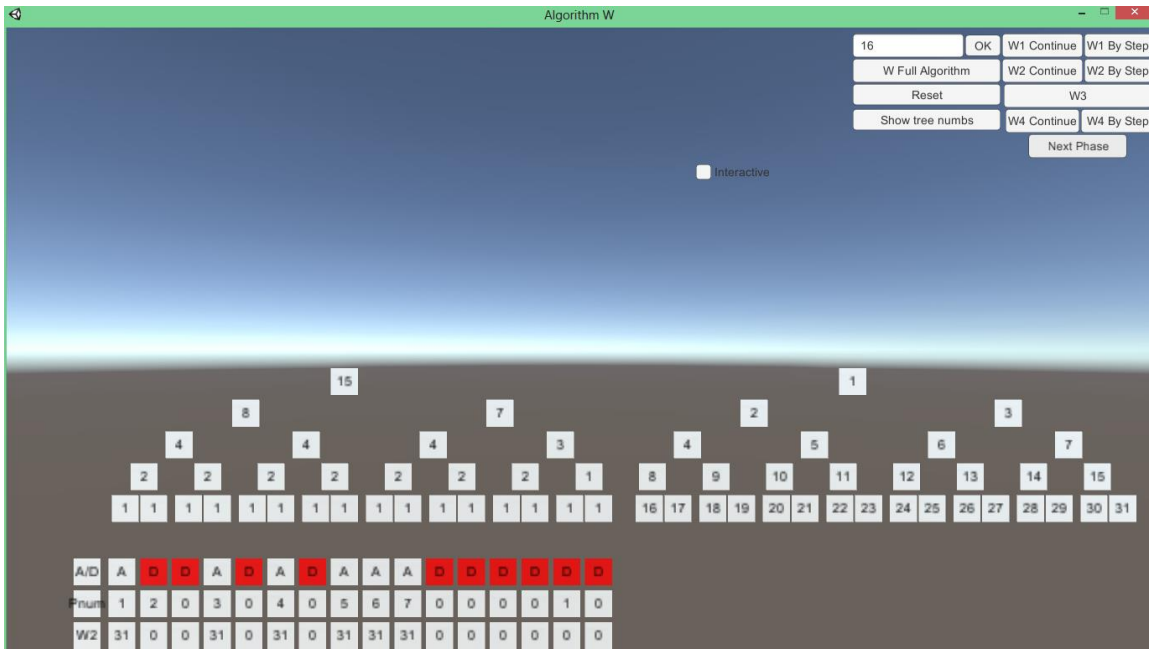
Σχήμα 7.15: Φάση W1

Εκτελείται ξανά η φάση W1 (Σχήμα 7.15) που αυτή την φορά παρατηρείται υπερεκτίμηση αφού η ρίζα γράφει 7 ενώ έχουμε 6 ενεργούς επεξεργαστές και περνάμε στην επομένη φάση την W2.



Σχήμα 7.16: Φάση W2

Στην φάση αυτή (Σχήμα 7.16) παρατηρείται πως όλοι οι επεξεργαστές πάνε να γράψουν στην μόνη τιμή που έχει μείνει 0 άρα στην θέση 31 του δένδρου μας. Που αυτό θα γίνει στην επόμενη φάση όπως φαίνεται στην επόμενη εικόνα



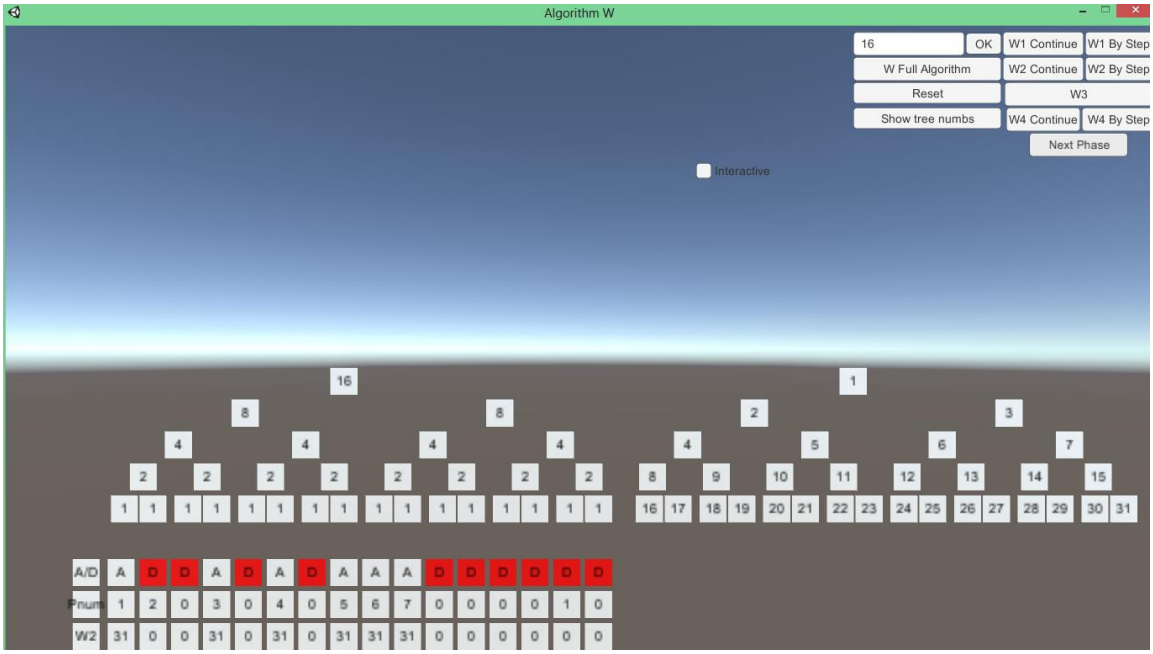
Σχήμα 7.17: Φάση W3

Οι επεξεργαστές έχουν συμπληρώσει και τον τελευταίο αριθμό με 1 (Σχήμα 7.17) στην φάση αυτή και μένει να εκτελεστεί η φάση W4.

Τέλος οι επεξεργαστές εκτελούν την φάση W4 (Σχήμα 7.18) και φτάνουν στην ρίζα με τον αριθμό 16. Ο αριθμός 16 είναι ίσος με τον αριθμό τον οποίο πέρασε ο χρήστης, άρα έχουμε φτάσει στο επιθυμητό τελικό αποτέλεσμα όπου τερματίζει και ο αλγόριθμος.

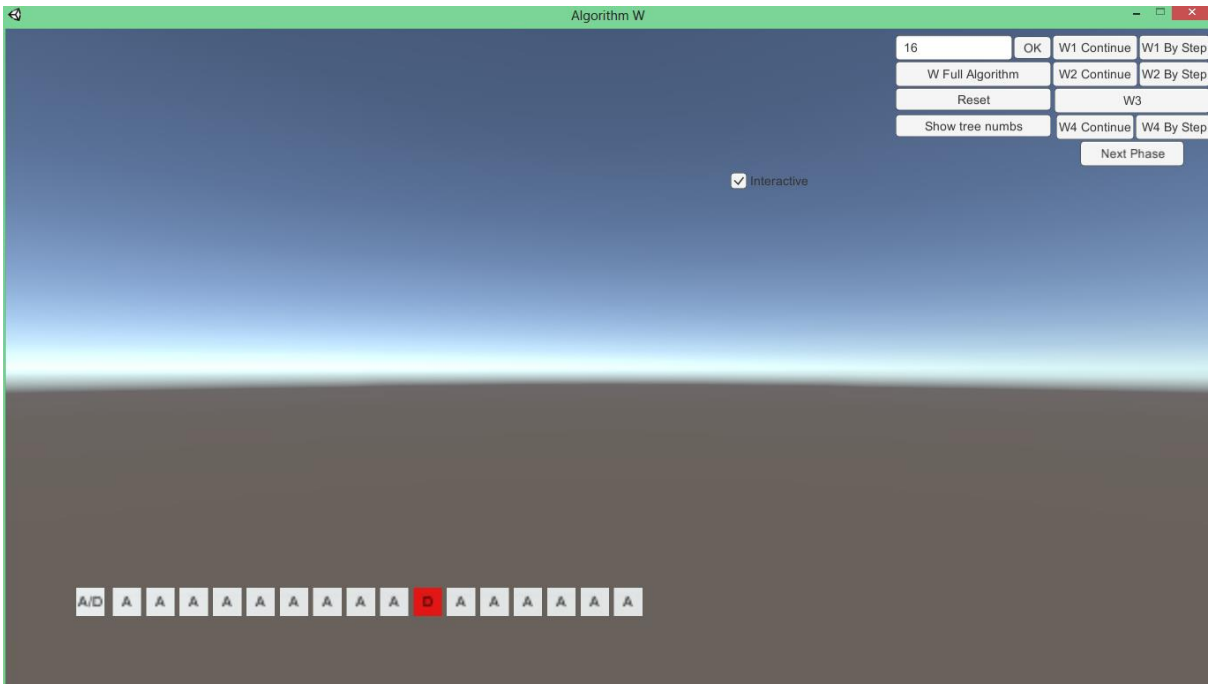
Στο σημείο αυτό παρατάσσεται ο κώδικας αν πατηθεί το κουμπί W Full Algorithm, με τον οποίο λειτουργούν οι φάσεις του αλγορίθμου, όπου έχουν καθυστέρηση μεταξύ τους για να φαίνονται τα βήματα.

```
IEnumerator waitw(float se){//create new gameobject and add it to taskobjects
        yield return new WaitForSeconds(se);
        W3();
        se = se + 1.5f;
        yield return new WaitForSeconds(se);
        W4 ();
        se = se + 1.5f;
        yield return new WaitForSeconds(se);
        while (done[1]!=check) {
                W1ver2 ();
                se = se + 0.8f;
                yield return new WaitForSeconds (se);
                W2v2 ();
                se = se + 0.8f;
                yield return new WaitForSeconds(se);
                W3();
                se = se + 0.8f;
                yield return new WaitForSeconds(se);
                W4 ();
                se = se + 0.8f;
                yield return new WaitForSeconds(se);
        }
}
```



Σχήμα 7.18: Φάση W4 και τελικό αποτέλεσμα

Στο Σχήμα 7.19 όπως φαίνεται έχει επιλεχθεί το interactive box με 16 επεξεργαστές. Σαν πρώτο στάδιο εμφανίστηκαν 16 επεξεργαστές όλοι alive και με το πάτημα του αριστερού κλικ στο κουτί το οποίο φαίνεται έγινε κόκκινο και dead. Σε κανένα σημείο του αλγόριθμου δεν θα καταρρεύσει επεξεργαστής εκτός μόνο αν το επιλέξει ο χρήστης.



Σχήμα 7.19: Interactive

Κεφάλαιο 8

Αλγόριθμος X

8.1Πρόβλημα Write-All στο A-PRAM

Το πρόβλημα το οποίο έχουμε να αντιμετωπίσουμε είναι το Write-All στο A-PRAM μοντέλο[]. Το A-PRAM όπως έχει αναφερθεί είναι ένα ασυγχρόνιστο μοντέλο όπου οι επεξεργαστές ενεργούν εντελώς ασυγχρόνιστα και μπορούν να διαβάσουν ή να γράψουν σε οποιαδήποτε κυψελίδα της μνήμης και η κάθε πρόσβαση μπορεί να πάρει διαφορετικό χρόνο. Επίσης μπορεί να υποπέσουν σε σφάλματα κατάρρευσης. Το πρόβλημα Write-All (όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο) αναφέρεται στην περίπτωση όπου δεδομένου μιας λίστας n στοιχείων όπου αυτά αρχικά έχουν την τιμή 0, οι τιμές των στοιχείων πρέπει να μετατραπούν σε 1. Θα χρησιμοποιηθούν p ασυγχρόνιστοι επεξεργαστές στο μοντέλο A-PRAM όπου δηλαδή μπορούν να καταρρεύσουν μέχρι $f<p$ επεξεργαστές.

8.2 Περιγραφή Αλγορίθμου

Στον αλγόριθμο X[2,5,6] όπως αναφέραμε χρησιμοποιεί n επεξεργαστές για την επίλυση του προβλήματος Write-All n αριθμών, όπου οι επεξεργαστές αυτοί ενεργούν εντελώς ασυγχρόνιστα. Ο αλγόριθμος χρησιμοποιεί ένα διάνυσμα μεγέθους $2n-1$, αναπαριστάμενο ως ένα νοητό δυαδικό δένδρο με n φύλλα. Το διάνυσμα αυτό είναι αποθηκευμένο στην κοινόχρηστη μνήμη των επεξεργαστών σε ένα μονοδιάστατο πίνακα Π του οποίου αρχικά οι τιμές είναι όλες 0. Τα στοιχεία του προβλήματος Write-All στην αρχή βρίσκονται στα φύλλα του δένδρου (δηλαδή στα $\Pi[n],\ldots,\Pi[2n-1]$).

Οι επεξεργαστές στην αρχή τοποθετούνται στα φύλλα του δένδρου ανάλογα με τον προσωπικό τους αριθμό π.χ. ο επεξεργαστής με προσωπικό αριθμό i όπου $0<i<n+1$, τοποθετείται στο $\Pi[n+i-1]$.

Στόχος του αλγόριθμου αυτού είναι οι επεξεργαστές να ανεβοκατεβαίνουν πάνω κάτω γράφοντας την τιμή 1 στις κυψελίδες οι οποίες έχουν την τιμή 0 μέχρι η ρίζα του δένδρου να πάρει και αυτή την τιμή 1.

Η διαδικασία αυτή εκτελείται με τον παρακάτω τρόπο. Ο επεξεργαστής ανάλογα με την ταχύτητα του θα προσπαθήσει να γράψει 1 στην θέση που βρίσκεται. Αν είναι 0 ο επεξεργαστής την αλλάζει σε 1 και ανεβαίνει επίπεδο. Αν όμως ένα άλλος επεξεργαστής έχει προλάβει και γράψει το 1 εκεί τότε ο επεξεργαστής θα ανέβει ένα βήμα προς τα πάνω και θα διαβάσει την τιμή του πατέρα , όπου θα προχωρά προς τα πάνω μέχρι να βρει ένα κόμβο με τιμή 0. Το ανέβασμα στο δένδρο γίνεται διαιρώντας την θέση που βρισκόμαστε δια 2. Μόλις βρεθεί ο κόμβος με το 0 ελέγχουμε πιο από τα δύο παιδιά έχει την τιμή 0 (η θέση των παιδιών είναι η θέση του πατέρα επί 2 για το αριστερά παιδί και η θέση του πατέρα επί δυο συν ένα για το δεξιά παιδί.) όπου υπάρχουν τέσσερις περιπτώσεις.

- Αν και τα δύο παιδιά έχουν την τιμή 1 τότε μετατρέπουμε την τιμή που βρισκόμαστε σε 1 και ο επεξεργαστής ανεβαίνει ένα επίπεδο πάνω.
- Αν όμως το αριστερά παιδί είναι 0 και το δεξιά ένα τότε πηγαίνουμε στο αριστερά , αν είναι το αντίθετο πηγαίνουμε στο δεξιά .
- Υπάρχει όμως και η περίπτωση η οποία και τα δύο παιδιά είναι 0. Σε αυτή την περίπτωση πηγαίνουμε ανάλογα με το διφίο του προσωπικού αριθμού του επεξεργαστή. Συγκεκριμένα ο επεξεργαστής υπολογίζει το δυαδικό αριθμό του προσωπικού του αριθμού και ανάλογα με το επίπεδο ελέγχει το διφίο αυτό. Αν η τιμή του bit αυτού είναι 0 πάει αριστερά αν είναι 1 πάει δεξιά. Για παράδειγμα αν ο επεξεργαστής με προσωπικό αριθμό τρία βρίσκεται στο δεύτερο επίπεδο θα ελέγξει τον δυαδικό του αριθμό στο δεύτερο διφίο. Δηλαδή ο δυαδικός του αριθμός που είναι το 01 θα δει πως το δεύτερο διφίο είναι το 0 και θα προχωρήσει στο αριστερά παιδί.

Οι περιπτώσεις αυτές γίνονται μέχρι να φτάσουμε σε φύλλο του δένδρου είτε σε κάποιο κόμβο ο οποίος τα παιδιά του είναι και τα δύο 0. Μόλις φτάσουμε σε μια από αυτές τις

δύο περιπτώσεις όπως αναφέραμε και πριν αλλάζουμε την τιμή σε 1 και ανεβαίνουμε ένα επίπεδο.

Ο αλγόριθμος αυτός τερματίζει όταν η ρίζα πάρει την τιμή 1. Συγκεκριμένα μόλις οι επεξεργαστές διαβάσουν ότι Π[1]=1 τερματίζουν.

Το κόστος του αλγόριθμου είναι $C^*_n(n) = O(n^{\log(2)3}) = O(n^{1.6})$.

Για τον χρόνο εξαρτάται η ταχύτητα των επεξεργαστών.

Στη συνέχεια δίνεται ο αλγόριθμος Χ σε μορφή ψευδοκώδικα για n αριθμούς με n επεξεργαστές :[2]

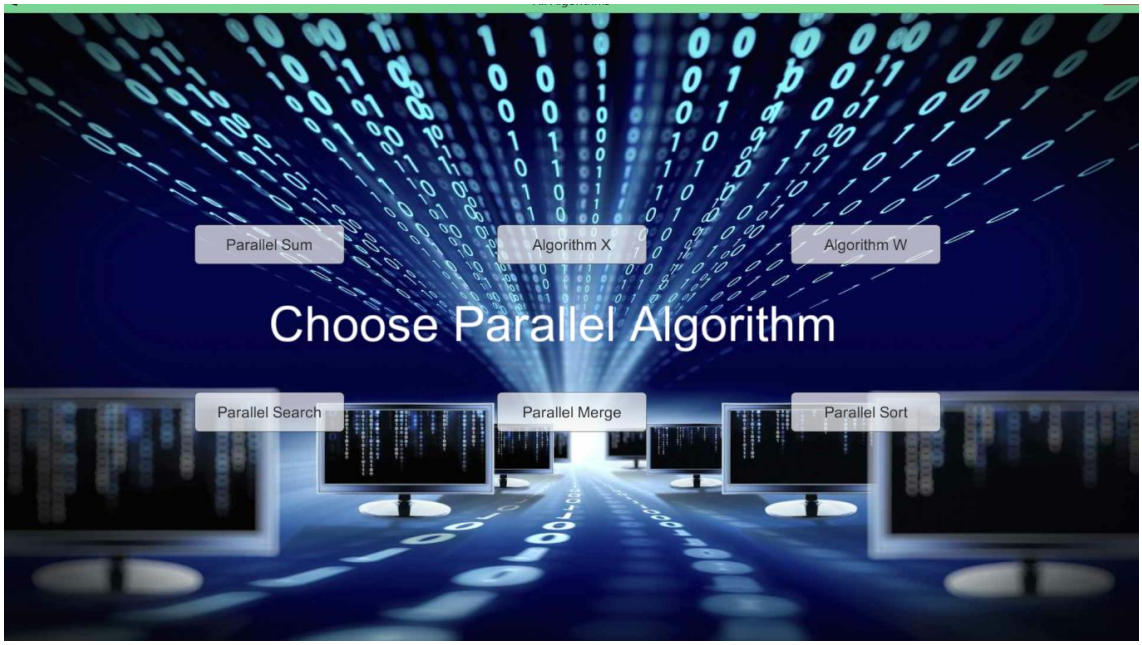
```
Processors P1, ..., Pn do asynchronously
      Shared array WA[1…2n-1]
      Pi: where = n + i - 1
           while WA[1] ≠ 1 do
            if (WA[where] ≠ 1 && where > n-1) then  % σε φύλλο
                  WA[where]=1
                  where = ⌊where/2⌋  %ανεβαίνει επίπεδο
            else if (WA[where] ≠ 1 && where ≤ n-1) then  %όχι σε φύλλο
                  if (WA[2*where] = 1 && WA[2*where+1] = 1) then
                        WA[where]=1  %τελειωμένο υποδέντρο
                        where = ⌊where/2⌋ %ανεβαίνει επίπεδο
                  if (WA[2*where] = 1 && WA[2*where+1] = 0) then
                        where = 2*where+1  %κατεβαίνει επίπεδο
                  if (WA[2*where] = 0 && WA[2*where] = 1) then
                        where = 2*where  %κατεβαίνει επίπεδο
                  if (WA[2*where] = 0 && WA[2*where] = 0) then
                        if (i_log(where)=0) then
                              where = 2*where  %στο αριστερό παιδί
                        else where = 2*where + 1  %στο δεξί παιδί
            end if
      end while
end do
```

(το $i_{log(k)}$ συμβολίζει την τιμή του log(k)^{στου} διφίου της log(n)-δυαδικής αναπαράστασης του i)

8.3 Γραφική Αναπαράσταση Αλγορίθμου

Τώρα θα δοθούν λεπτομέρειες για την υλοποίηση του αλγόριθμου X για την επίλυση του προβλήματος Write-All στο μοντέλο A-PRAM στην κονσόλα Unity και βήμα προς βήμα πως αυτός τρέχει.

Στο σχήμα 8.1 είναι η εικόνα που βλέπουμε μόλις τρέξουμε το εκτελέσιμο.



Σχήμα 8.1: Πρώτη εικόνα εκτελέσιμου

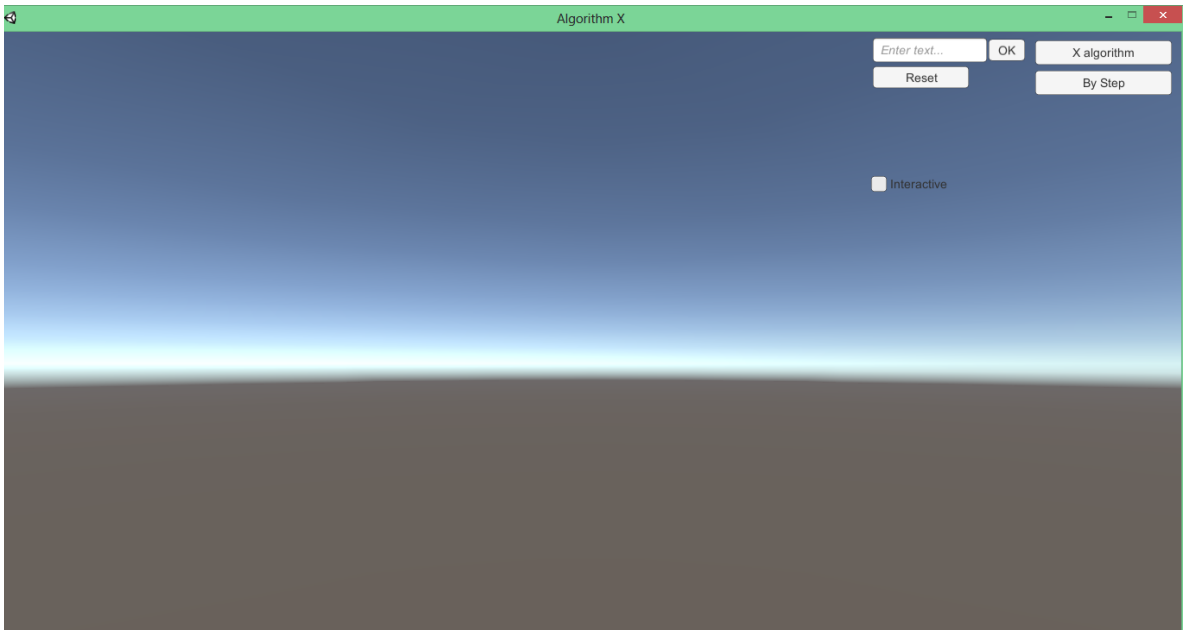
Στη συνέχεια επιλέγουμε το κουμπί του Algorithm X όπου είναι ο αλγόριθμος ο οποίος μελετάμε στο συγκεκριμένο κεφάλαιο. Ακολούθως εμφανίζεται η πιο κάτω αρχική εικόνα (Σχήμα 8.2) Σε οποιαδήποτε σημείο που τρέχει το πρόγραμμα πατώντας το κουμπί ESC πηγαίνουμε στην αρχική επιλογή αλγορίθμων ή πατώντας το P γίνεται παύση.

Για την υλοποίηση του αλγόριθμου αυτού χρησιμοποιήθηκε ένας πίνακας για να κρατά πληροφορίες για την κατάσταση του κάθε επεξεργαστή αν είναι ενεργός ή όχι.

Επίσης χρησιμοποιήθηκαν δύο πίνακες για την βοήθεια υπολογισμού του προσωπικού αριθμού κάθε επεξεργαστή σε δυαδική μορφή. Ένας με τον οποίο κρατούσαμε τον αριθμό των δυφίων και ένα δισδιάστατο που είχε σε κάθε επίπεδο του ένα διφίο του επεξεργαστή π.χ. για τον επεξεργαστή με προσωπικό αριθμό 3 (11 στο δυαδικό) στο πρώτο επίπεδο είχε τον αριθμό 1 και στο δεύτερο 1 , άρα στον πίνακα μας στην θέση Π[2,0]=1 και στην θέση Π[2,1]=1 .

Η αρχική εικόνα αποτελείται από:

- 1 input field το οποίο ο χρήστης περνά το πλήθος αριθμών της λίστας.
- Το κουμπί OK με το οποίο καταχωρείτε το πλήθος αυτό.
- Ένα κουμπί το κουμπί Reset το οποίο κάνει reset την σκηνή στην αρχική της μορφή για να εκτελέσει νέο παράδειγμα ο χρήστης.
- Το κουμπί X Algorithm το οποίο τρέχει τον αλγόριθμο ολόκληρο με μια μικρή καθυστέρηση μεταξύ των βημάτων για να διακρίνονται αυτά.
- Το κουμπί By Step με το οποίο ο αλγόριθμος εκτελεί ένα βήμα.
- Ένα toggle box το οποίο ο χρήστης μπορεί να επιλέξει αν οι επεξεργαστές θα καταρρέουν διαδραστικά ή τυχαία.
- Το canvas που μέσα σε αυτό ήταν όλα τα πιο πάνω.



Σχήμα 8.2: Αρχική Εικόνα

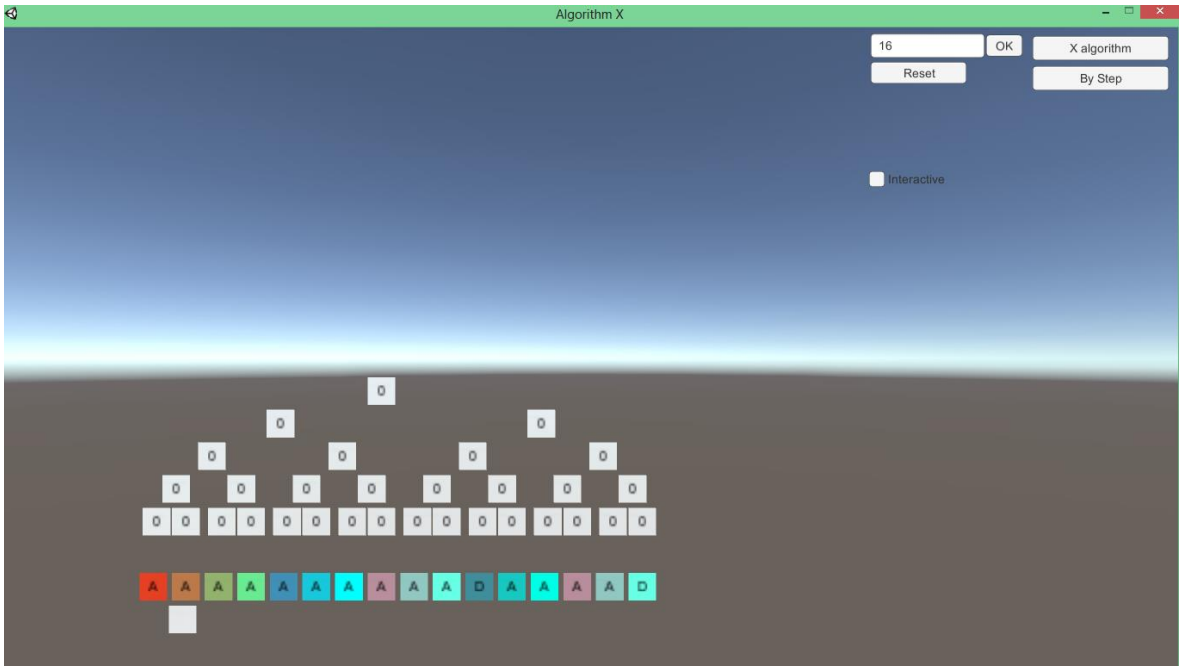
Το μέγεθος της κάμερας για μεγαλύτερο αριθμό *n* μικραίνει και μεγαλώνει αντίστοιχα ώστε να υποστηρίζει μέχρι 64 αριθμούς και οι αριθμοί αυτοί να παραμένουν διακριτοί στο μάτι.

Για την δημιουργία του αλγόριθμου αυτού χρησιμοποιήθηκε ένα prefab το οποίο είχε μέσα ένα ακόμα αντικείμενο το οποίο αποτελούταν από ένα text mesh όπου στη συνέχεια σε αυτό το text mesh αποθηκεύεται και εμφανίζεται ο αριθμός.

Για την δημιουργία του κώδικα χρησιμοποιήθηκαν δύο script. Το ένα ήταν ενσωματωμένο στο canvas και είχε τις συναρτήσεις για κάθε κουμπί , τα οποία καλούσαν την αντίστοιχη συνάρτηση μέσο αυτού και το άλλο ήταν υπεύθυνο για την κατάρρευση των επεξεργαστών με το αριστερό κλικ του ποντικιού .

Αν ο χρήστης προσπαθήσει να πατήσει το κουμπί OK χωρίς input field να έχει τιμή τότε δεν θα αρχικοποιήσει την σκηνή μας. Αν δεν αρχικοποιηθεί η σκηνή μας κανένα από τα άλλα κουμπιά δεν λειτουργεί. Με τον τρόπο αυτό περιορίζουμε τον χρήστη από το να υποπέσει σε άσκοπα σφάλματα και δημιουργείται μια καλύτερη αλληλεπίδραση με τον πρόγραμμα και το UI .

Στη συνέχεια θα αναλυθεί ένα παράδειγμα με 16 αριθμούς με την επιλογή interactive να μην είναι επιλεγμένη έτσι τα σφάλματα μας θα είναι τυχαία. Σε οποιοδήποτε σημείο του αλγόριθμου πατώντας στο D ή στο A του επεξεργαστή μπορούμε να ενεργοποιήσουμε ή να καταρρεύσουμε τον αντίστοιχο επεξεργαστή. Ο χρήστης γράφει τον αριθμό 16 στην περίπτωση μας στο input field και ακολούθως πατά το Ok button. Στη συνέχεια παρατηρεί να εμφανίζεται στην οθόνη του η ακόλουθη εικόνα. (Σχήμα 8.3)



Σχήμα 8.3: Εμφάνιση επεξεργαστών και νοητού δένδρου

Όπως παρατηρείται εμφανίστηκαν στην οθόνη 16 επεξεργαστές με αντίστοιχα χρώματα οι οποίοι μερικοί είναι “alive”(A) και μερικοί “dead”(D) αυτό γίνεται τυχαία επειδή το interactive δεν είναι επιλεγμένο όπως φαίνεται και από τον πιο κάτω κώδικα.

```
int random=UnityEngine.Random.Range (0, 101);
if (randomtog.isOn) {
	random = 100;
}
if (random>=33){ //random Dead/Alive
	table[i]=1;
	alive[i]="A";
}
else{
	table[i]=0;
	alive[i]="D";
}
```

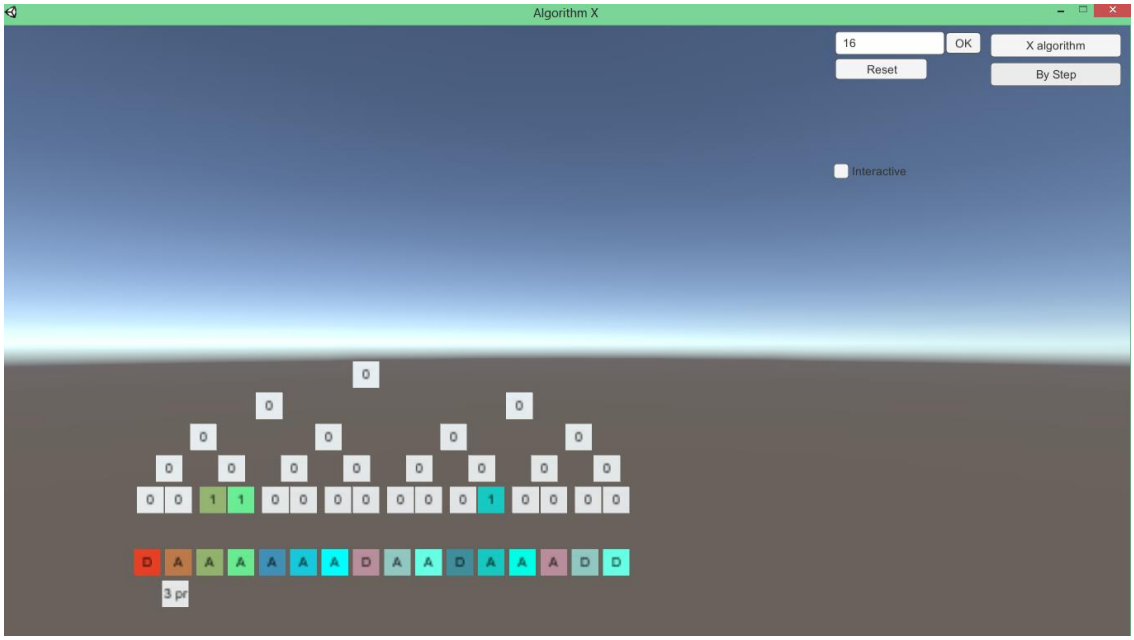
Στο σημείο αυτό παίρνουν τιμές οι δύο πίνακες για τους δυαδικούς αριθμούς οι οποίοι αναφέρθηκα πριν (βλ. σελίδα 96) με τον εξής κώδικα.

```
for (int i=0; i<check; i++) { //convert to bits and initialize lvl to 0
	var result = Convert.ToString(i, 2);
	binstrings[i]=result;
	lvl[i]=0;
}
for (int i=0; i<check; i++) {//cut bits of binary
	for (int j=0; j<binstrings[i].Length; j++) {
		string piece = binstrings [i].Substring (j, 1);
		bincut[i,(log-binstrings[i].Length)+j]=Int32.Parse(piece);
	}
}
```

Επίσης δημιουργείται το αντίστοιχο νοητό δυαδικό δένδρο με επίπεδα log(n) όπου εμφανίζεται στην οθόνη μας με αρχικές τιμές 0 σε όλους τους κόμβους και οι επεξεργαστές είναι τοποθετημένοι στα φύλλα του δένδρου.

Ακόμη εμφανίζεται ένα άσπρο κενό κουτί κάτω από τους επεξεργαστές όπου σε επόμενα βήματα θα δείχνει ποιος επεξεργαστής εκτέλεσε λειτουργία στο συγκεκριμένο βήμα ή το πλήθος τους αν ήταν περισσότεροι από ένας.

Στη συνέχεια πατάμε το By Step και εμφανίζεται η παρακάτω εικόνα (Σχήμα 8.4)



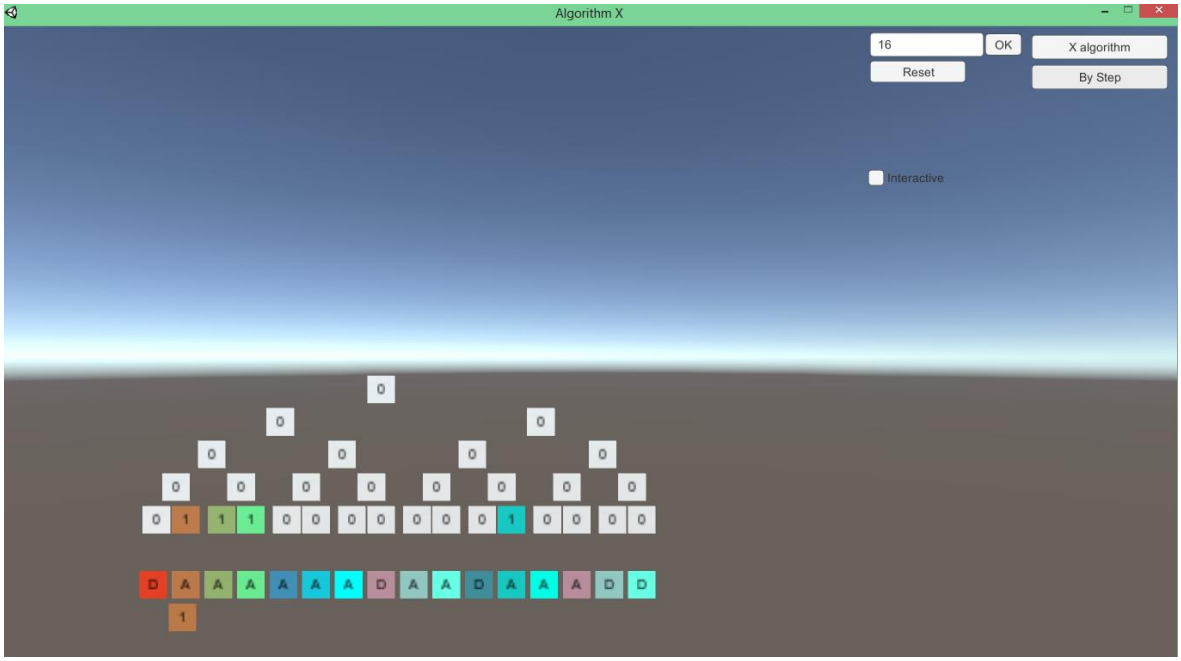
Σχήμα 8.4: Βήμα 1

Στο συγκεκριμένο σημείο παρατηρούμε ότι 3 επεξεργαστές εκτέλεσαν την λειτουργία και έγραψαν τον αριθμό 1 στο φύλλο το οποίο βρίσκονταν. Συγκεκριμένα ο επεξεργαστής 2,3 και 11. Έτσι κάτω από τους επεξεργαστές παρατηρείτε ότι το κουτί έχει πάρει την τιμή 3 pr όπου συμβολίζει πως 3 processors έγραψαν την τιμή 1 στο βήμα αυτό και ανέβηκαν ένα επίπεδο προς τα πάνω.

Ο κώδικας για τα φύλλα είναι ο ακόλουθος:

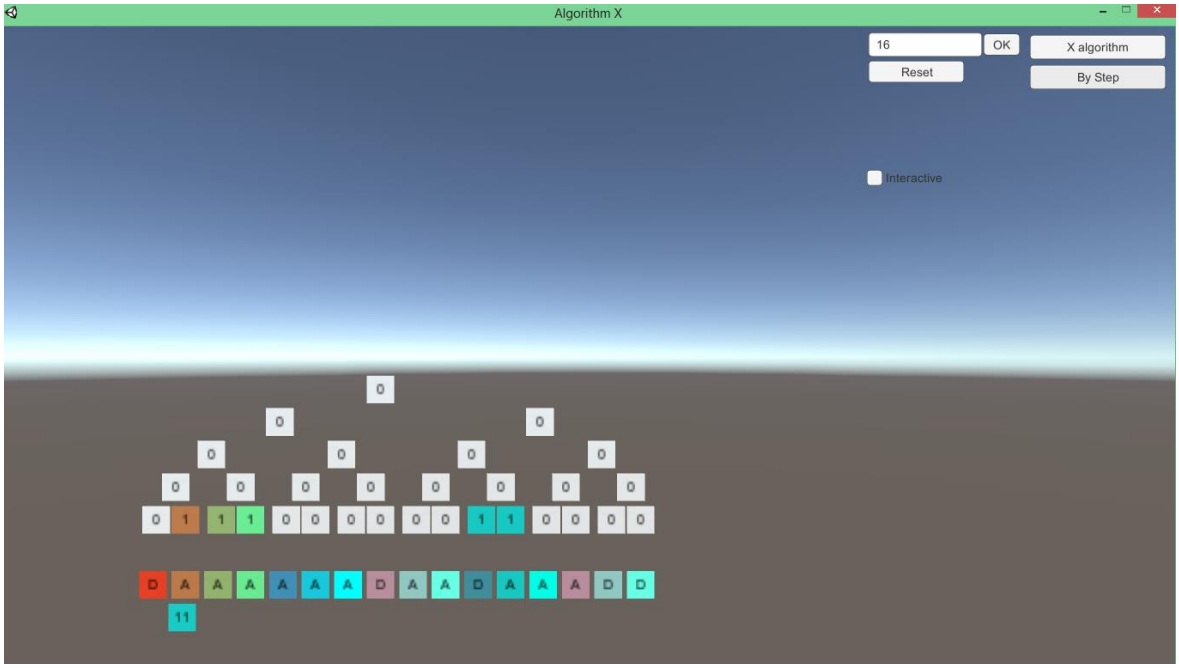
```
if (done [where [faster]] == 0 && where [faster] >= check) { //if you are at leaf make the
                                                              value 0 and move 1 lvl
        done [where [faster]] = 1;
        StartCoroutine (waitcolor (treeobjects [where [faster]], 1, colortable [faster]));
        where [faster] = where [faster] / 2;
        lvl [faster]++;
        execute = 1;
}
```

Στη συνέχεια θα παρατηρήσουμε το επόμενο βήμα(Σχήμα 8.5) όπου μόνο ένας επεξεργαστής έχει εκτελέσει λειτουργία την συγκεκριμένη φορά. Όπως φαίνεται στην ακόλουθη εικόνα ο επεξεργαστής 1 ήταν ο γρηγορότερος και έγραψε την τιμή 1 στο φύλλο το οποίο βρισκόταν. Έτσι στο σημείο το οποίο γράφει ποιος επεξεργαστής ενέργησε γράφει τον προσωπικό αριθμό του επεξεργαστή και το χρώμα του όπου στο παράδειγμα μας είναι 1 με χρώμα καφέ.



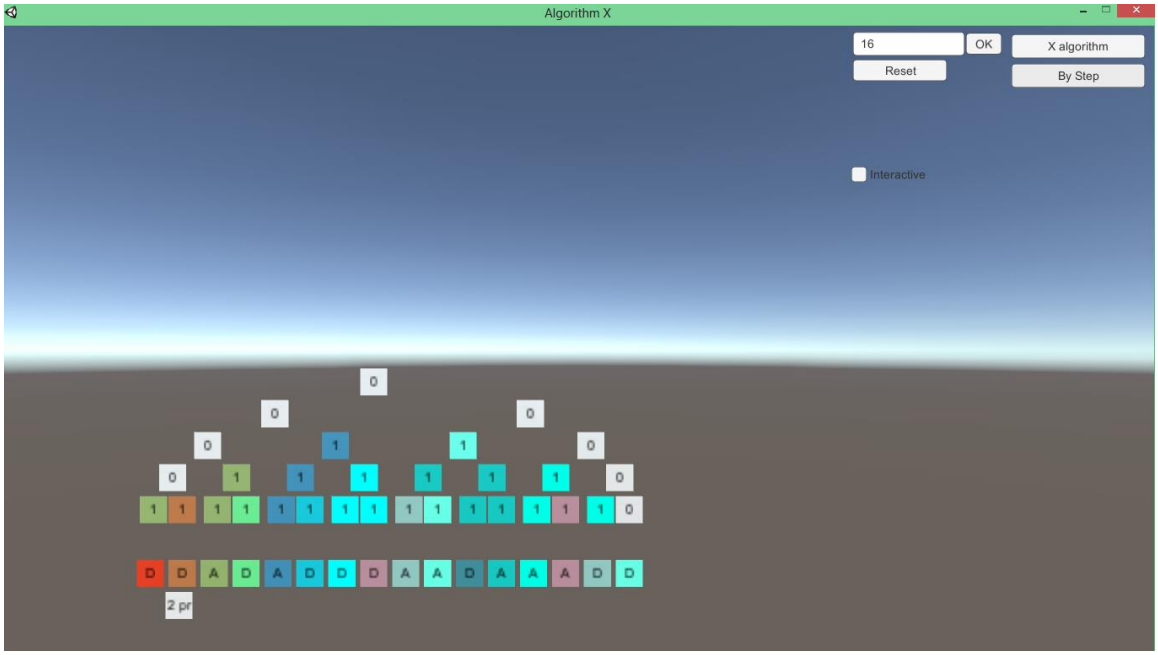
Σχήμα 8.5: Βήμα 2

Ακολούθως γίνεται το ίδιο ακριβώς στο βήμα 3(Σχήμα 8.6) με τον επεξεργαστή 11 όπου γράφει την τιμή 1 στο φύλλο το οποίο βρίσκεται.



Σχήμα 8.6: Βήμα 3

Στη συνέχεια μετά την εκτέλεση αρκετών βημάτων πήραμε ως αποτέλεσμα την ακόλουθη εικόνα(Σχήμα 8.7) που όπως παρατηρούμε κάποιοι επεξεργαστές οι οποίοι ήταν ενεργοί τώρα έχουν καταρρεύσει και έχουν πάρει την τιμή D.



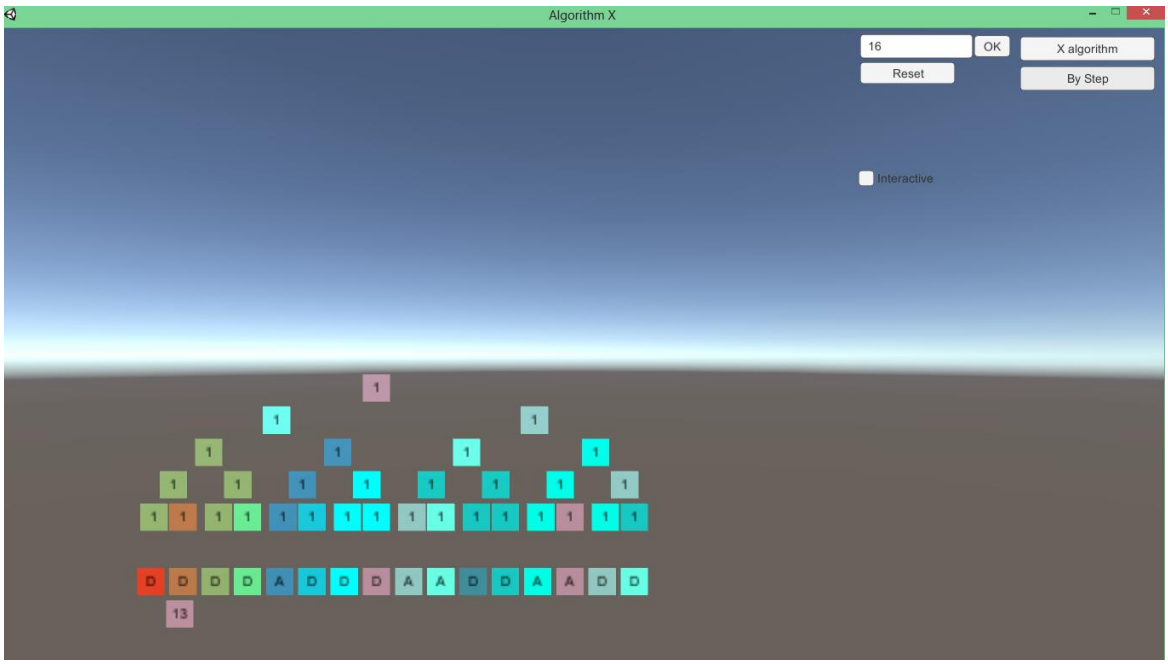
Σχήμα 8.7: Ενδιάμεσο Βήμα

Στη συνέχεια παραθέτετε το κομμάτι κώδικα με τον υπολογισμό τον οποίο εκτελεί ο γρηγορότερος επεξεργαστής σε κάθε βήμα για το πού θα γράψει την τιμή 1 αν δεν βρίσκεται σε φύλλο.

```
if (done [where [faster]] == 0 && where [faster] < check && execute == 0) {
                                                    //if you are not on a leaf
  if (done [where [faster] * 2] == 1 && done [where [faster] * 2 + 1] == 1) {
                                        //if both children are 1 make your value 1 too
        done [where [faster]] = 1;
        StartCoroutine (waitcolor (treeobjects [where [faster]], 1, colortable [faster]));
        where [faster] = where [faster] / 2;
        lvl [faster]++;
        execute = 1;
}
  else if (execute == 0) {
        if (where [faster] * 2 <= check * 2) {//you cant go futher than leaves
            while (done[where[faster]*2]==0 || done[where[faster]*2+1]==0) {
                                                    //while one of the children is 0
                if (done [where [faster] * 2 + 1] == 0 && done [where [faster] *2] == 1)
                                                    //if the right children is 0 make child=2
                    child = 2;
                else if (done [where [faster] * 2] ==0&&done[wher [faster] * 2 + 1] == 1)
                                                    //if left children is 1 make child=1
                    child = 1;
                else {
//if both children are 0 check with your binary and lvl if its 0 go left if its 1 go right
                    if (binstrings [faster].Length < lvl [faster]) {
                        child = 1;
                    }
                    else {
                        int loglv = log - lvl [faster];
                        if (bincut [faster, loglv] == 0)
                            child = 1;
                        else
                            child = 2;
                    }
```

```
                }
                lvl [faster]--;
                if (child == 1)//go left
                        where [faster] = where [faster] * 2;
                if (child == 2)//go right
                        where [faster] = where [faster] * 2 + 1;
                if (where [faster] * 2 >= check * 2 - 1)
                        break;
            }
        }
        done [where [faster]] = 1;
        StartCoroutine (waitcolor (treeobjects [where [faster]], 1, colortable [faster]));
        where [faster] = where [faster] / 2;
        lvl [faster]++;
    }
}
```

Έτσι ο αλγόριθμος συνεχίζει την λειτουργία του μέχρι το σημείο που φαίνεται στην πιο κάτω εικόνα (Σχήμα 8.8)



Σχήμα 8.8: Τελικό αποτέλεσμα

Ο αλγόριθμος έχει ολοκληρωθεί στο σημείο αυτό γιατί η ρίζα έχει πάρει την τιμή 1. Αυτό σημαίνει ότι τα 2 παιδία της έχουν και αυτά την τιμή 1 και αυτό συνεχίζεται μέχρι το τελευταίο επίπεδο που είναι τα φύλλα. Άρα ο αλγόριθμος μας έχει δώσει λύση στο πρόβλημα μας αφού όλοι οι κόμβοι έχουν αλλάξει τιμή από 0 σε 1. Έτσι ο επόμενος επεξεργαστής όπου και να βρίσκεται θα ανέβει μέχρι την ρίζα όπου θα δει την τιμή 1 και θα τερματίσει τον αλγόριθμο.

Στο παράδειγμα αυτό επιλέγηκε η μη διαδραστική λύση του αλγόριθμου όπου οι καταρρεύσεις επεξεργαστών ήταν τυχαίες. Η διαδραστική λύση πατώντας το κουμπί interactive λειτουργεί ακριβώς με τον ίδιο τρόπο όπως και ο αλγόριθμος W (βλ. σελίδα 89).

Κεφάλαιο 9

Επίλογος

9.1 Επίλογος

Κλείνοντας την αναφορά για την διπλωματική αυτή εργασία θα ήταν παράλειψη να μην αναφερθώ στα συμπεράσματα και τα οφέλη της εργασίας αυτής. Στο κεφάλαιο αυτό επίσης θα αναφερθούν προοπτικές που υπάρχουν για μελλοντική εργασία για το θέμα αυτό.

Μέσα από την διπλωματική εργασία αυτή μπορέσαμε να αναπαραστήσουμε γραφικά με την χρήση του Unity διάφορους αλγόριθμους και να εξηγήσουμε τους λόγους τους οποίους ο παράλληλος υπολογισμός είναι τόσο σημαντικός στη σημερινή εποχή και τις δυνατότητες που μας παρέχει.

Στην αρχή έγινε μια αναφορά στο game-machine Unity το οποίο χρησιμοποιήθηκε για την γραφική αναπαράσταση κάθε αλγορίθμου στην διπλωματική αυτή. Συγκεκριμένα αναφερθήκαμε στις δυνατότητες του, στοιχεία γι’ αυτό και ακόμη έγινε μια αναφορά σε βασικές γνώσεις για την δημιουργία ενός έργου με την χρήση του εργαλείου αυτού και επεξηγήσεις ως προς την διεπαφή χρήστη.

Επίσης στην εργασία αυτή έγινε μελέτη ως προς τα μοντέλα παραλληλισμού, τις ιδιαιτερότητες τους και τις υποκατηγορίες τον μοντέλων όπου οι περισσότερες χρησιμοποιήθηκαν για την υλοποίηση των αλγορίθμων οι οποίοι παρουσιάστηκαν.

Στη συνέχεια μελετήθηκαν εις βάθος ο κάθε αλγόριθμος ως προς τι πρόβλημα λύνει , τον σειριακό τρόπο επίλυσης των προβλημάτων αυτών, πως μπορούμε να το λύσουμε πιο γρήγορα με την χρήση παραλληλισμού και αναφορά ως προς χρόνο και κόστος εκτέλεσης του καθενός.

Στο τέλος υλοποιήθηκε η γραφική αναπαράσταση των παράλληλων αλγορίθμων δίνοντας λεπτομέρειες για αυτή. Οι αλγόριθμοι οι οποίοι παρουσιάστηκαν ήταν οι εξής : μη βέλτιστος αλγόριθμος παράλληλης άθροισης, βέλτιστος αλγόριθμος παράλληλης άθροισης, αλγόριθμος παράλληλης αναζήτησης, αλγόριθμος παράλληλης συγχώνευσης, αλγόριθμος παράλληλου ταξινόμησης , αλγόριθμος W και τέλος ο αλγόριθμος X.

7.2 Προβλήματα Υλοποίησης

Στην παρούσα διπλωματική δεν παρουσιάστηκαν μεγάλα προβλήματα στην υλοποίηση, τα περισσότερα ήταν λόγο απειρίας με το πρόγραμμα Unity.

Ένα πρόβλημα το οποίο είχε παρουσιαστεί ήταν το πώς θα τοποθετούνται οι επεξεργαστές και τα κουτιά στην οθόνη επειδή δημιουργούνταν όλα κατά την διάρκεια εκτέλεσης και ήταν δύσκολο να βρεθούν οι συντεταγμένες του καθενός σε κάθε αλγόριθμο.

Επίσης άλλο πρόβλημα το οποίο είχε αντιμετωπισθεί ήταν ο χρωματισμός των επεξεργαστών γιατί σε μεγάλο αριθμό επεξεργαστών οι επεξεργαστές είχαν παρόμοια χρώματα.

Ακόμη υπήρχε ένα πρόβλημα μεταξύ τα βήματα των αλγορίθμων όπου έπρεπε να βάλουμε μια μικρή καθυστέρηση για να διακρίνονται αυτά. Για παράδειγμα ο αλγόριθμος X σε μεγάλο αριθμό n καθυστερεί αρκετά λόγο ότι η καθυστέρηση μεγαλώνει σε κάθε βήμα, το οποίο όμως διορθώθηκε.

Ένα ακόμη μικρό πρόβλημα που αντιμετωπίσθηκε ήταν ο τρόπος με τον οποίο έκανε διαιρέσεις η C# σε δεκαδικούς αριθμούς όπου χρειάζονταν να μπουν πολλές μαθηματικές συναρτήσεις για να πάρουμε σωστά αποτελέσματα.

Τέλος ο μέγιστος αριθμός επεξεργαστών και αριθμών που μπορούσαμε να βάλουμε ήταν περιορισμένος γιατί όσο απομακρυνόταν η κάμερα για να μπορούν να μπουν περισσότερα δεδομένα στην οθόνη δεν ήταν πια διακριτοί οι αριθμοί στα στοιχεία αυτά.

7.3 Οφέλη Διπλωματικής Εργασίας

Με την ολοκλήρωση της παρούσας διπλωματικής εργασίας αποκόμισα πάρα πολλά οφέλη. Καταρχάς είχα την ευκαιρία να μάθω πως λειτουργεί ένα πολύ ωραίο εργαλείο με απίστευτες δυνατότητες όπως το Unity που ενώ δεν ήξερα καν την ύπαρξη του με σκληρή δουλεία κατάφερα να το μάθω και να φέρω εις πέρας την εργασία αυτή. Προγραμματιστικά επίσης με βοήθησε να μάθω την γλώσσα C# , να βελτιώσω τις προγραμματιστικές μου ικανότητες και τον τρόπο σκέψης μου.

Επίσης μου δόθηκε η ευκαιρία να δω πως είναι να δουλεύεις σε μια μεγάλη εργασία, πώς να οργανωθείς γι’ αυτή και πώς να την συγγράψεις.

Ακόμη ενώ ήξερα πως λειτουργούσαν οι αλγόριθμοι αυτοί σε γενικές γραμμές με την υλοποίηση τους έμαθα πολλές περισσότερες λεπτομέρειες γι’ αυτούς. Επίσης αναλύοντας τους αλγόριθμους αυτούς συνειδητοποίησα πόσες δυνατότητες έχουν και μέχρι που μπορεί να φτάσει ο παραλληλισμός, όταν με αυτά τα μικρά παραδείγματα μπορούμε να δούμε τόση διαφορά στην ταχύτητα επίλυσης των προβλημάτων.

7.4 Μελλοντική Εργασία

Η εργασία αυτή όπως αναφέρθηκε και πριν είχε σκοπό να υλοποιήσει διάφορους αλγόριθμους ώστε κάποιος να μπορεί να τους διδάξει ή κάποιος να τους κατανοήσει πιο εύκολα. Με την γραφική αναπαράσταση του κάθε αλγόριθμου και με την υλοποίηση να

είναι τέτοια ώστε κάθε αλγόριθμος να μπορεί να προχωράει βήμα με βήμα πέτυχε τον σκοπό της εργασίας σε μεγάλο βαθμό.

Όμως στην εργασία αυτή υλοποιήθηκαν έξι διαφορετικοί αλγόριθμοι απλοί όπως αυτός της παράλληλης άθροισης αλλά και πιο πολύπλοκοι όπως τον αλγόριθμο W. Ως μελλοντική εργασία θα πρότεινα να δημιουργούνταν και άλλοι παράλληλοι αλγόριθμοι με τον ίδιο τρόπο ώστε να μπορούν να διδαχτούν και αυτοί καλύτερα. Για παράδειγμα θα μπορούσε να υλοποιηθεί ο αλγόριθμος του Euler.

Επίσης θα με την πάροδο του χρόνου και την διδασκαλία τον αλγορίθμων αυτών θα πάρουμε πληροφορίες για το που υστερούν στην εκμάθηση και τη θα μπορούσε να βελτιωθεί ώστε να είναι πιο κατανοητό τι κάνει ο κάθε αλγόριθμος και πώς.

Τέλος στην εργασία αυτή το πλήθος των αριθμών που είχε χρησιμοποιηθεί δεν ήταν μεγάλο για τον λόγο ότι οι αριθμοί δεν διακρίνονταν με μεγαλύτερο πλήθος. Θα μπορούσε να βρεθεί κάποιος τρόπος εκτός από την απομάκρυνση της κάμερας ώστε οι αριθμοί να είναι διακριτοί και σε μεγαλύτερο πλήθος. Για παράδειγμα όταν ο χρήστης περνά το κέρσορα πάνω από ένα αριθμό αυτός να μεγεθυνόταν. Έτσι σε μεγαλύτερο πλήθος οι αριθμοί θα μπορούσαν να παραμείνουν διακριτοί.

Βιβλιογραφία

[1] Unity http://unity3d.com/ ,last accessed 20/4/2016

[2] Χρύσης Γεωργίου, Σημειώσεις Μαθήματος ΕΠΛ431 – Σύνθεση Παράλληλων Αλγορίθμων, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, http://www2.cs.ucy.ac.cy/~chryssis/EPL431/ ,2016

[3] Ελένη Σκιττίδου , Γραφική Αναπαράσταση Παράλληλων Εύρωστων Αλγορίθμων Με Το Εργαλείο Java Swing , Ατομική Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, 2011

[4] Νάταλη Τεμενέ , Implementation and Visual Representation of A Fault-tolerant Algorithm for Gathering Fat Crash-prone robots on a plane, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, 2015

[5] Βιβλίο μαθήματος: An Introduction to Parallel Algorithms, Joseph JaJa,1992

[6] Βιβλίο: Cooperative Task-Oriented Computing : Algorithms & Complexity , Chryssis Georgiou & Alexander A. Shvartsman , Morgan & Claypool Publishers, July 2011

[7] Wikipedia https://en.wikipedia.org/wiki/Unity_(game_engine) ,
last accessed 20/4/2016

Παράρτημα Α

Κώδικας από Αλγορίθμους που Υλοποιήθηκαν

Α.1 Αλγόριθμος Παράλληλης Άθροισης (Βέλτιστος και Μη Βέλτιστος)

```
using UnityEngine;
using System.Collections;
using System;
using UnityEngine.UI;

public class ParallelSum : MonoBehaviour {

	public InputField numOfProcessorsText;  // reference on input field
	public InputField numOfProcessorsOpt;  // reference on input field
	public GameObject processorPrefab;		// reference on game object



	private int steps = 0;					// step of the algorithm
	private float stepSize = 1.0f;			// for the position of object
	int num=0;					//num given by the
user
	public int []table;				//temp table for the tree
	public float [] vectors;			//temp table for positions of vectors
	public int[,] done;				//which processors are done on each step
	public GameObject[] tableobject;		//table with reference on each gameobject of the tree
	public Color[] colortable;			//table with colors

	float col1=1.20f;
	float col2=-0.10f;
	float col3=-0.13f;

	int colorcount=0;
	int glob=0;
	int firsttime=0;
	int log=1;
	int check = 2;
	int best=0;
	public void Reset() {			//reset the algorithm
		Application.LoadLevel("ParallelSum");

	}

	public void Update() {
		if (Input.GetKeyDown(KeyCode.Escape)) {
			Application.LoadLevel ("MainScene");
		}
	}


	IEnumerator waittransf(GameObject ob,int val,float se){//insert value to a gameobject after a delay
		yield return new WaitForSeconds(se);
		ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + val;
	}

	IEnumerator waitonesecond(Vector3 position,int value,float se,Color newc){ //insert color and value to a processor
after a delay
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);

		go.GetComponent <MeshRenderer> ().material.color = newc;
	}
```

```
IEnumerator waitcolor(Vector3 position,int value,Color newc){  //insert color and value to a processor after a delay
	GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
	go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
	yield return new WaitForSeconds(0.6f);

	go.GetComponent <MeshRenderer> ().material.color = newc;
}

IEnumerator waitcolor(GameObject ob,int value,Color newc){//color a gameobject and put an int on it
	ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
	yield return new WaitForSeconds(0.6f);

	ob.GetComponent <MeshRenderer> ().material.color = newc;
}


public void Initproc(){//initialize processors
	if ((numOfProcessorsText.text.Length != 0)&(num==0) ){
			num = Int32.Parse (numOfProcessorsText.text);
		var l = Math.Log (num, 2);
		l = Math.Ceiling (l);
		//PlayerPrefs.SetInt ("num", num);
		//PlayerPrefs.Save(); //gia na filaksis
		//num=PlayerPrefs.GetInt("name"); //gia na piasis


		for (int i=1; i<l; i++) { //find log and processors for patching
			check = check * 2;
			log++;

		}

		colortable = new Color[check];

		table = new int[check];
		vectors = new float[check];
		done = new int[check, log + 1];
		tableobject = new GameObject[check];




		int correct = Math.Abs (check - num); //for patching

		float temp = 0;
		int cont = 0;
		for (int i=0; i<num; i++) {
			if (i % 2 == 0)//every 2 leafs leave some space
				temp = 1.2f;
			else
				temp = 1.1f;

			Vector3 position = new Vector3 ((i * stepSize) - 10.5f + temp, (steps * stepSize), 0);
			vectors [i] = (i * stepSize - 10 + temp);

			if (i % 2 == 0) {//new color to each processor
				if (col1 < 0)
					col1 = 1;
				col1 = col1 - 0.20f;
				if (col2 > 1)
					col2 = 0.30f;
				col2 = col2 + 0.18f;
				if (col3 > 1)
					col3 = 0.47f;
				col3 = col3 + 0.18f;
				Color newColor = new Color (col1, col2, col3, 1);
				colortable [colorcount] = newColor;
				colorcount++;

			}
```

```
                                        int random = UnityEngine.Random.Range (0, 101);
                                        table [i] = random;
                                        GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as
GameObject;
                                        go.GetComponent <MeshRenderer> ().material.color = colortable [colorcount - 1];
                                        go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table [i];

                                        cont = i + 1;



                                }


                                while (correct != 0) {

                                        if (correct % 2 == 0) {
                                                if (col1 < 0)
                                                        col1 = 1;
                                                col1 = col1 - 0.20f;
                                                if (col2 > 1)
                                                        col2 = 0.30f;
                                                col2 = col2 + 0.18f;
                                                if (col3 > 1)
                                                        col3 = 0.47f;
                                                col3 = col3 + 0.18f;
                                                Color newColor = new Color (col1, col2, col3, 1);
                                                colortable [colorcount] = newColor;
                                                colorcount++;
                                        }

                                        Vector3 position = new Vector3 ((cont * stepSize) - 10.6f + temp, (steps * stepSize),
0);
                                        vectors [cont] = (cont * stepSize - 10 + temp);
                                        GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as
GameObject;
                                        table [cont] = 0;
                                        go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table
[cont];
                                        go.GetComponent <MeshRenderer> ().material.color = colortable [colorcount - 1];
                                        cont++;


                                        correct--;
                                }

                                num = num / 2;
                                check = check / 2;
                                steps++;

                                for (int i=0; i<colortable.Length/2; i++) {
                                        Vector3 pos = new Vector3 ((i * stepSize) - 10.5f + 1.1f, -2, 0);
                                        GameObject ob = Instantiate (processorPrefab, pos, Quaternion.identity) as
GameObject;


                                        ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + i;
                                        StartCoroutine (waitcolor (ob, i, colortable [i]));


                                        Vector3 positionobj = new Vector3 ((i * stepSize) - 10.5f + 1.1f, -3, 0);
                                        GameObject obj = Instantiate (processorPrefab, positionobj, Quaternion.identity) as
GameObject;
                                        obj.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + 0;
                                        tableobject [i] = obj;

                                }
                        }

                }

                public void Best(){//Optimum
```

```
float se=1.3f;
if ((numOfProcessorsOpt.text.Length != 0) & (num == 0))
if (firsttime == 0) {//works only if its the begin of algorithm
	num = Int32.Parse (numOfProcessorsOpt.text);

	var l = Math.Log (num, 2);
	l = Math.Ceiling (l);



	for (int i=1; i<l; i++) {
		check = check * 2;
		log++;
	}
	best=1;



	int processors = check / log;
	int perproc = check / processors;
	colortable = new Color[processors + 2];
	table = new int[check];
	vectors = new float[check];
	done = new int[check, log + 1];
	tableobject = new GameObject[check];


	var l2 = Math.Log (perproc, 2);
	l2 = Math.Ceiling (l2);


	int bcheck = 2;
	for (int i=1; i<l2; i++) {
		bcheck = bcheck * 2;
	}


	int bcorrect = Math.Abs (bcheck - processors);

	int correct = Math.Abs (check - num);
	int cont = processors;


	float temp = 0;



	for (int i=0; i<num; i++) {
		if (i % perproc == 0) {
			temp = 1.3f;


			if (col1 < 0)
				col1 = 1;
			col1 = col1 - 0.20f;
			if (col2 > 1)
				col2 = 0.30f;
			col2 = col2 + 0.18f;
			if (col3 > 1)
				col3 = 0.47f;
			col3 = col3 + 0.18f;
			Color newColor = new Color (col1, col2, col3, 1);
			colortable [colorcount] = newColor;
			colorcount++;


		} else if (i % perproc != 1)//group by mod
			temp = 0.9f;
		else
			temp = 1.1f;

		Vector3 position = new Vector3 ((i * stepSize) - 10 + temp, (steps * stepSize), 0);
		vectors [i] = (i * stepSize - 10 + temp);
```

```
                        GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as
GameObject;

                        int random = UnityEngine.Random.Range (0, 101);
                        table [i] = random;
                        go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table [i];
                        go.GetComponent <MeshRenderer> ().material.color = colortable [colorcount - 1];


                }


                int novalue = num % perproc; //extra processors

                cont = processors;

                if (novalue != 0)
                        processors++;

                for (int i=0; i<processors; i++) {
                        Vector3 pos = new Vector3 ((i * stepSize) - 10.5f + 1.1f, -2, 0);
                        GameObject ob = Instantiate (processorPrefab, pos, Quaternion.identity) as
GameObject;


                        ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + i;
                        StartCoroutine (waitcolor (ob, i, colortable [i]));


                        Vector3 positionobj = new Vector3 ((i * stepSize) - 10.5f + 1.1f, -3, 0);
                        GameObject obj = Instantiate (processorPrefab, positionobj, Quaternion.identity) as
GameObject;
                        obj.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + 0;
                        tableobject [i] = obj;

                }

                int t = 1;
                if (t == 2) {
                        while (correct != 0) {

                                if (novalue % perproc == 0) {
                                        temp = 1.3f;


                                        if (col1 < 0)
                                                col1 = 1;
                                        col1 = col1 - 0.20f;
                                        if (col2 > 1)
                                                col2 = 0.30f;
                                        col2 = col2 + 0.18f;
                                        if (col3 > 1)
                                                col3 = 0.47f;
                                        col3 = col3 + 0.18f;
                                        Color newColor = new Color (col1, col2, col3, 1);
                                        colortable [colorcount] = newColor;
                                        colorcount++;
                                } else if (novalue % perproc != 1)
                                        temp = 0.9f;
                                else
                                        temp = 1.1f;

                                Vector3 position = new Vector3 ((cont * stepSize) - 10 + temp, (steps *
stepSize), 0);
                                vectors [cont] = (cont * stepSize - 10 + temp);
                                GameObject go = Instantiate (processorPrefab, position,
Quaternion.identity) as GameObject;

                                table [cont] = 0;
```

```
                go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" +
table [cont];

                go.GetComponent <MeshRenderer> ().material.color = colortable
[colorcount - 1];

                Vector3 positionobj = new Vector3 ((cont * stepSize) - 10.5f + 1.1f, -3, 0);
                GameObject obj = Instantiate (processorPrefab, positionobj,
Quaternion.identity) as GameObject;
                obj.transform.FindChild ("Value").GetComponent<TextMesh> ().text = ""
+ table [cont];
                tableobject [cont] = obj;

                cont++;
                correct--;
                novalue++;
        }


        int remain = check % perproc;

        num = num / 2;
        check = check / 2;
        steps++;

        colorcount = 0;
        int sum = 0;
        if (remain != 0)
                log++;
        if (num == 4)
                log--;

        for (int i=0; i<log; i++) {
                sum = 0;
                for (int j=0; j<perproc; j++)
                        sum = sum + table [(i * perproc) + j];
                table [i] = sum;
        }
        sum = 0;
        if (remain != 0) {
                for (int i=0; i<remain; i++)
                        sum = sum + table [(perproc * cont) + i];
                table [cont] = sum;
        } else
                table [cont] = 0;

        for (int i=0; i<log; i++) {
                vectors [i] = (vectors [perproc * i]) + steps - 0.5f;
                Vector3 position = new Vector3 ((vectors [i]), (steps * stepSize), 0);

                if (i % 2 == 0) {
                        colorcount++;
                }
                StartCoroutine (waitonesecond (position, table [i], se, colortable [colorcount
- 1]));
                StartCoroutine (waittransf (tableobject [i], table [i], se));
        }

        while (bcorrect != 0) {
                if (cont == 7)
                        temp = temp + 1.5f;
                if (cont == 2)
                        temp = temp - 15.0f;

                vectors [cont] = (cont * stepSize + 17 + temp);
                Vector3 position = new Vector3 (vectors [cont], (steps * stepSize), 0);

                if (cont == 2)
                        vectors [cont] = (cont * stepSize + 12 + temp);

                StartCoroutine (waitonesecond (position, table [cont], se, colortable
[colorcount]));
                cont++;
                table [cont] = 0;
```

```
                                        bcorrect--;
                                }

                                steps++;
                                check = bcheck / 2;


                        }
                }
                firsttime++;
        }


        public void ByStepSerial(){//with only one processor
                if ((check != 0)&(num!=0)) {
                        if (steps==5){
                                vectors [glob] = (vectors [2 * glob]) + steps + 2.5f;
                        }
                        else{
                        vectors[glob]=(vectors[2*glob])+steps-0.9f;
                        }
                        if (firsttime==1)
                                vectors[glob]=vectors[glob]+1.0f;
                        if (firsttime==1 && steps==3)
                                vectors[glob]=vectors[glob]+1.5f;
                        if (firsttime==1 && steps==4)
                                vectors[glob]=vectors[glob]+4.5f;
                        Vector3 position = new Vector3 ((vectors[glob]), (steps * stepSize), 0);

                        int a=table[2*glob];
                        int b=table[2*glob+1];
                        table[glob]=a+b;
                        done[glob,steps]=1;

                        GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
                        go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table[glob];
                        tableobject[0].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" +
table[glob];
                        glob++;


                        if (glob==check){//move to next step
                                glob=0;
                                steps++;
                                check=check/2;
                        }
                }
        }

        public void ByStep(){//parallel sum step by step

                if (best == 1) {
                        best++;
                        float se1 = 0.0f;
                        var l = Math.Log (num, 2);
                        l = Math.Ceiling (l);

                        log = 1;
                        check = 2;
                        for (int i=1; i<l; i++) {
                                check = check * 2;
                                log++;
                        }



                        int processors = check / log;
                        int perproc = check / processors;

                        var l2 = Math.Log (perproc, 2);
                        l2 = Math.Ceiling (l2);
```

```
int bcheck = 2;
for (int i=1; i<l2; i++) {
	bcheck = bcheck * 2;
}


int bcorrect = Math.Abs (bcheck - processors);

int correct = Math.Abs (check - num);
int cont = processors;


float temp = 0;


int novalue = num % perproc; //extra processors


if (novalue != 0)
	processors++;

while (correct != 0) {

	if (novalue % perproc == 0) {
		temp = 1.3f;


		if (col1 < 0)
			col1 = 1;
		col1 = col1 - 0.20f;
		if (col2 > 1)
			col2 = 0.30f;
		col2 = col2 + 0.18f;
		if (col3 > 1)
			col3 = 0.47f;
		col3 = col3 + 0.18f;
		Color newColor = new Color (col1, col2, col3, 1);
		colortable [colorcount] = newColor;
		colorcount++;
	} else if (novalue % perproc != 1)
		temp = 0.9f;
	else
		temp = 1.1f;

	Vector3 position = new Vector3 ((cont * stepSize) - 10 + temp, (steps * stepSize), 0);
	vectors [cont] = (cont * stepSize - 10 + temp);
	GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as
GameObject;

	table [cont] = 0;

	go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table
[cont];


	go.GetComponent <MeshRenderer> ().material.color = colortable [colorcount - 1];

	Vector3 positionobj = new Vector3 ((cont * stepSize) - 10.5f + 1.1f, -3, 0);
	GameObject obj = Instantiate (processorPrefab, positionobj, Quaternion.identity) as
GameObject;
	obj.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table

[cont];
	tableobject [cont] = obj;

	cont++;
	correct--;
	novalue++;
}


int remain = check % perproc;

num = num / 2;
check = check / 2;
steps++;
```

```
		colorcount = 0;
		int sum = 0;
		if (remain != 0)
			log++;
		if (num == 4)
			log--;

		for (int i=0; i<log; i++) {
			sum = 0;
			for (int j=0; j<perproc; j++)
				sum = sum + table [(i * perproc) + j];
			table [i] = sum;
		}
		sum = 0;
		if (remain != 0) {
			for (int i=0; i<remain; i++)
				sum = sum + table [(perproc * cont) + i];
			table [cont] = sum;
		} else
			table [cont] = 0;

		for (int i=0; i<log; i++) {
			vectors [i] = (vectors [perproc * i]) + steps - 0.5f;
			Vector3 position = new Vector3 ((vectors [i]), (steps * stepSize), 0);

			if (i % 2 == 0) {
				colorcount++;
			}
			StartCoroutine (waitonesecond (position, table [i], se1, colortable [colorcount - 1]));
			StartCoroutine (waittransf (tableobject [i], table [i], se1));
		}

		while (bcorrect != 0) {
			if (cont == 7)
				temp = temp + 1.5f;
			if (cont == 2)
				temp = temp - 15.0f;

			vectors [cont] = (cont * stepSize + 17 + temp);
			Vector3 position = new Vector3 (vectors [cont], (steps * stepSize), 0);

			if (cont == 2)
				vectors [cont] = (cont * stepSize + 12 + temp);

			StartCoroutine (waitonesecond (position, table [cont], se1, colortable [colorcount]));
			cont++;
			table [cont] = 0;
			bcorrect--;
		}

		steps++;
		check = bcheck / 2;


	} else {
		int count = 0;
		if ((check != 0) & (num != 0)) {
			colorcount = -1;
			for (int i=0; i<check; i++) {
				if (done [i, steps] == 0) {
					if (steps == 5) {
						vectors [i] = (vectors [2 * i]) + steps + 3.3f;
					} else {
						vectors [i] = (vectors [2 * i]) + steps - 0.9f;
					}

					if (firsttime == 1)
						vectors [i] = vectors [i] + 1.0f;
					if (firsttime == 1 && steps == 3)
						vectors [i] = vectors [i] + 1.5f;
					if (firsttime == 1 && steps == 4)
						vectors [i] = vectors [i] + 4.5f;
```

```
                                                                Vector3 position = new Vector3 ((vectors [i]), (steps * stepSize),
0);

                                                                int a = table [2 * i];
                                                                int b = table [2 * i + 1];
                                                                table [i] = a + b;

                                                                if (i % 2 == 0) {
                                                                        colorcount++;
                                                                }
                                                                tableobject [count].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + table [i];

                                                                count++;
                                                                GameObject go = Instantiate (processorPrefab, position,
Quaternion.identity) as GameObject;

                                                                go.transform.FindChild ("Value").GetComponent<TextMesh>
().text = "" + table [i];

                                                                go.GetComponent <MeshRenderer> ().material.color =
colortable [colorcount];

                                                                done [i, steps] = 1;
                                                        }
                                                }
                                                steps++;

                                                check = check / 2;
                                        }
                                }

                        }

                        public void InitializeProcessors () {//parallel sum full version
                                if (best == 1) {
                                        best++;
                                        float se1 = 1.0f;
                                        var l = Math.Log (num, 2);
                                        l = Math.Ceiling (l);

                                        log=1;
                                        check=2;
                                        for (int i=1; i<l; i++) {
                                                check = check * 2;
                                                log++;
                                        }



                                        int processors = check / log;
                                        int perproc = check / processors;

                                        var l2 = Math.Log (perproc, 2);
                                        l2 = Math.Ceiling (l2);


                                        int bcheck = 2;
                                        for (int i=1; i<l2; i++) {
                                                bcheck = bcheck * 2;
                                        }


                                        int bcorrect = Math.Abs (bcheck - processors);

                                        int correct = Math.Abs (check - num);
                                        int cont = processors;


                                        float temp = 0;


                                        int novalue = num % perproc; //extra processors


                                        if (novalue != 0)
                                                processors++;
```

```
while (correct != 0) {

		if (novalue % perproc == 0) {
			temp = 1.3f;


			if (col1 < 0)
				col1 = 1;
			col1 = col1 - 0.20f;
			if (col2 > 1)
				col2 = 0.30f;
			col2 = col2 + 0.18f;
			if (col3 > 1)
				col3 = 0.47f;
			col3 = col3 + 0.18f;
			Color newColor = new Color (col1, col2, col3, 1);
			colortable [colorcount] = newColor;
			colorcount++;
		} else if (novalue % perproc != 1)
			temp = 0.9f;
		else
			temp = 1.1f;

		Vector3 position = new Vector3 ((cont * stepSize) - 10 + temp, (steps * stepSize), 0);
		vectors [cont] = (cont * stepSize - 10 + temp);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as
GameObject;

		table [cont] = 0;

		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table
[cont];

		go.GetComponent <MeshRenderer> ().material.color = colortable [colorcount - 1];

		Vector3 positionobj = new Vector3 ((cont * stepSize) - 10.5f + 1.1f, -3, 0);
		GameObject obj = Instantiate (processorPrefab, positionobj, Quaternion.identity) as
GameObject;
		obj.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table
[cont];
		tableobject [cont] = obj;

		cont++;
		correct--;
		novalue++;
}


int remain = check % perproc;

num = num / 2;
check = check / 2;
steps++;

colorcount = 0;
int sum = 0;
if (remain != 0)
		log++;
if (num == 4)
		log--;

for (int i=0; i<log; i++) {
		sum = 0;
		for (int j=0; j<perproc; j++)
			sum = sum + table [(i * perproc) + j];
		table [i] = sum;
}
sum = 0;
if (remain != 0) {
		for (int i=0; i<remain; i++)
			sum = sum + table [(perproc * cont) + i];
		table [cont] = sum;
} else
		table [cont] = 0;
```

```
                for (int i=0; i<log; i++) {
                        vectors [i] = (vectors [perproc * i]) + steps - 0.5f;
                        Vector3 position = new Vector3 ((vectors [i]), (steps * stepSize), 0);

                        if (i % 2 == 0) {
                                colorcount++;
                        }
                        StartCoroutine (waitonesecond (position, table [i], se1, colortable [colorcount - 1]));
                        StartCoroutine (waittransf (tableobject [i], table [i], se1));
                }

                while (bcorrect != 0) {
                        if (cont == 7)
                                temp = temp + 1.5f;
                        if (cont == 2)
                                temp = temp - 15.0f;

                        vectors [cont] = (cont * stepSize + 17 + temp);
                        Vector3 position = new Vector3 (vectors [cont], (steps * stepSize), 0);

                        if (cont == 2)
                                vectors [cont] = (cont * stepSize + 12 + temp);

                        StartCoroutine (waitonesecond (position, table [cont], se1, colortable [colorcount]));
                        cont++;
                        table [cont] = 0;
                        bcorrect--;
                }

                steps++;
                check = bcheck / 2;


        }

                float se = 2.2f;

        if (best ==0)
                se = 0.8f;

                while ((check != 0)&(num!=0)) {
                        int count = 0;
                        colorcount = 0;
                        for (int i=0; i<check; i++) {
                                if (done [i, steps] == 0) {
                                        if (steps == 5) {
                                                vectors [i] = (vectors [2 * i]) + steps + 2.5f;
                                        } else {
                                                vectors [i] = (vectors [2 * i]) + steps - 0.9f;
                                        }

                                        if (firsttime == 1)
                                                vectors [i] = vectors [i] + 1.0f;
                                        if (firsttime == 1 && steps == 3)
                                                vectors [i] = vectors [i] + 1.5f;
                                        if (firsttime == 1 && steps == 4)
                                                vectors [i] = vectors [i] + 4.5f;
                                        Vector3 position = new Vector3 ((vectors [i]), (steps * stepSize),
0);

                                        int a = table [2 * i];
                                        int b = table [2 * i + 1];
                                        table [i] = a + b;

                                        if (i % 2 == 0)
                                                colorcount++;

                                        StartCoroutine (waitonesecond (position, table [i], se, colortable
[colorcount - 1]));
                                        StartCoroutine (waittransf (tableobject [count], table [i], se));
                                        count++;

                                }
```

```
                                        }
                                        se = se + 1.4f;
                                        check = check / 2;
                                        steps++;
                                }
                        }
                }
```

A.2 Αλγόριθμος Παράλληλης Αναζήτησης

```
using UnityEngine;
using System.Collections;
using System;
using UnityEngine.UI;

public class ParallelSearch : MonoBehaviour {

        public InputField numOfProcessorsText;  // reference on input field
        public InputField numofn;
        public InputField searchnum;
        public GameObject processorPrefab;              // reference on game object



        private int steps = 0;                                  // step of the algorithm
        private float stepSize = 1.0f;                  // for the position of object
        int proc=0;                                                                             //proc given by the
user
        int num=0;
        int search;
        public int []table;                                                             //temp table for the tree
        public float [] vectors;                                //temp table for positions of vectors
        public GameObject[] tableobject;            //table with reference on each gameobject of the tree
        public Color[] colortable;                              //table with colors
        public GameObject[] lasttable;                  //table with reference on each last
        public GameObject[] ctable;
        public GameObject sol;
        int solfound=0;
        int finished=0;
        int divider=0;
        float col1=1.20f;
        float col2=-0.10f;
        float col3=-0.13f;

        int templen=0;
        int length=0;
        int colorcount=0;
        int firsttime=0;
        int not=0;
        public int right;
        public int left;
        public int [] c;
        public int [] last;
        int solved=0;

        public void Reset() {           //reset the algorithm
                Application.LoadLevel("Search");

        }

        public void Update() {
                if (Input.GetKeyDown(KeyCode.Escape)) {
                        Application.LoadLevel ("MainScene");
                }
        }

        IEnumerator waittransf(GameObject ob,int val,float se){//insert value to a gameobject after a delay
                yield return new WaitForSeconds(se);
                ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + val;
        }
```

```
IEnumerator waittransf(GameObject ob,string val,float se){//insert value to a gameobject after a delay
	yield return new WaitForSeconds(se);
	ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + val;
}

IEnumerator waitonesecond(Vector3 position,int value,float se,Color newc){ //insert color and value to a processor after a
delay
	yield return new WaitForSeconds(se);
	GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
	go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
	yield return new WaitForSeconds(0.6f);

	go.GetComponent <MeshRenderer> ().material.color = newc;
}

IEnumerator waitcolor(Vector3 position,int value,Color newc){  //insert color and value to a processor after a delay
	GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
	go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
	yield return new WaitForSeconds(0.6f);

	go.GetComponent <MeshRenderer> ().material.color = newc;
}

IEnumerator waitcolor(GameObject ob,int value,Color newc){//color a gameobject and put an int on it
	ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
	yield return new WaitForSeconds(0.6f);

	ob.GetComponent <MeshRenderer> ().material.color = newc;
}

public void Initalg4(){
	if ((numOfProcessorsText.text.Length != 0) & (num == 0) & (numofn.text.Length != 0) & (proc == 0)) {
		proc = Int32.Parse (numOfProcessorsText.text);
		num = Int32.Parse (numofn.text);

		int random = UnityEngine.Random.Range (0, 8);

		var l = Math.Log (num, 2);
		l = Math.Ceiling (l);



		colortable = new Color[proc + 1];
		table = new int[num];
		vectors = new float[num];
		c = new int[proc + 2];
		last = new int [proc + 2];
		lasttable = new GameObject[proc + 2];
		ctable = new GameObject[proc + 2];
		tableobject = new GameObject[num];

		if (num > proc) {
			divider = num / (proc + 1);
			int div2 = num - divider * proc;
			if ((div2 > proc) & (num / divider != proc + 1))
				divider++;

		}


		float temp = 1.1f;
		for (int i=0; i<num; i++) {
			if (divider != 0) {
				if (i % divider == 0 & (i < (divider * proc))) {
					temp = 1.2f;
					if (col1 < 0)
						col1 = 1;
					col1 = col1 - 0.20f;
					if (col2 > 1)
						col2 = 0.30f;
					col2 = col2 + 0.18f;
					if (col3 > 1)
						col3 = 0.67f;
					col3 = col3 + 0.18f;
```

```
                                        Color newColor = new Color (col1, col2, col3, 1);
                                        colortable [colorcount] = newColor;
                                        colorcount++;
                                } else
                                        temp = 1.1f;
                        }



                        Vector3 position = new Vector3 ((i * stepSize) - 10.5f + temp, (steps * stepSize), 0);
                        vectors [i] = (i * stepSize - 10 + temp);

                        table [i] = random;
                        int random2 = UnityEngine.Random.Range (1, 7);
                        random = random + random2;

                        GameObject ob1 = Instantiate (processorPrefab, position, Quaternion.identity) as
GameObject;
                        StartCoroutine (waittransf (ob1, table [i], 0.6f));
                        tableobject [i] = ob1;


                }


                if (divider == 0) {
                        for (int i=0; i<proc; i++) {
                                if (col1 < 0)
                                        col1 = 1;
                                col1 = col1 - 0.20f;
                                if (col2 > 1)
                                        col2 = 0.30f;
                                col2 = col2 + 0.18f;
                                if (col3 > 1)
                                        col3 = 0.67f;
                                col3 = col3 + 0.18f;
                                Color newColor = new Color (col1, col2, col3, 1);
                                colortable [colorcount] = newColor;
                                colorcount++;
                        }
                }

                steps++;
                colorcount = 0;

                for (int i=0; i<proc; i++) {
                        Vector3 positionobj = new Vector3 ((i * stepSize) - 10.5f + 1.1f, -1, 0);
                        GameObject obj = Instantiate (processorPrefab, positionobj, Quaternion.identity) as
GameObject;
                        StartCoroutine (waitcolor (obj, i, colortable [i]));



                        Vector3 positionobj2 = new Vector3 ((i * stepSize) - 10.5f + 1.1f, -2, 0);
                        GameObject obj2 = Instantiate (processorPrefab, positionobj2, Quaternion.identity) as
GameObject;
                        lasttable [i] = obj2;

                        Vector3 positionobj3 = new Vector3 ((i * stepSize) - 10.5f + 1.1f, -3, 0);
                        GameObject obj3 = Instantiate (processorPrefab, positionobj3, Quaternion.identity) as
GameObject;
                        ctable [i] = obj3;

                }

                Vector3 pos = new Vector3 (-10.5f, -1, 0);
                GameObject ob = Instantiate (processorPrefab, pos, Quaternion.identity) as GameObject;
                ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "proc";

                Vector3 pos2 = new Vector3 (-10.5f, -2, 0);
                GameObject ob2 = Instantiate (processorPrefab, pos2, Quaternion.identity) as GameObject;
                ob2.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "last";
```

```
                Vector3 pos3 = new Vector3 (-10.5f, -3, 0);
                GameObject ob3 = Instantiate (processorPrefab, pos3, Quaternion.identity) as GameObject;
                ob3.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "c";

                Vector3 pos4 = new Vector3 (-10.5f, -4, 0);
                GameObject ob4 = Instantiate (processorPrefab, pos4, Quaternion.identity) as GameObject;
                ob4.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "Sol";

                Vector3 pos5 = new Vector3 (-9.5f, -4, 0);
                GameObject ob5 = Instantiate (processorPrefab, pos5, Quaternion.identity) as GameObject;

                sol = ob5;
        }
}


public void execute () {
        int procfound = 0;
        //int solfound = 0;
        float se = 0.6f;

        if ((searchnum.text.Length != 0) & (search == 0)) {
        if (finished==0)
        if (firsttime == 0) {
                search = Int32.Parse (searchnum.text);


                left = 0;
                right = num + 1;
                c [0] = 0;
                c [proc + 1] = 1;
                length = right - left;



                if ((length-1) < proc){
                        for (int i=0; i<length-1; i++) {
                                StartCoroutine (waitcolor (tableobject [i], table [i], colortable [colorcount]));
                                StartCoroutine (waittransf (lasttable [i], table [i], se));
                                colorcount++;
                        }
                        right--;
                }
                else {
                        divider = length / (proc + 1);
                        int div2 = num - divider * proc;
                        if ((div2 > proc) & (num / divider != proc + 1))
                                divider++;

                        for (int i=1; i<=num; i++) {
                                if (i <= divider * proc)
                                        StartCoroutine (waitcolor (tableobject [i - 1], table [i - 1],
colortable [colorcount]));
                                else
                                        StartCoroutine (waitcolor (tableobject [i - 1], table [i - 1],
Color.white));
                                if ((i % divider == 0) && (colorcount < proc))
                                        colorcount++;
                        }


                        last [0] = left;
                        last [proc + 1] = right;

                        for (int i=0; i<proc; i++) {
                                last [i + 1] = left + (i + 1) * ((right - left) / (proc + 1));
                                StartCoroutine (waittransf (lasttable [i], table [last [i + 1] - 1], se));
                                if (table [last [i + 1] - 1] > search)
                                        c [i + 1] = 1;
                                else if (table [last [i + 1] - 1] < search)
                                        c [i + 1] = 0;
                                else {
                                        procfound=i;
                                        solfound=last [i + 1] - 1;
```

```
                                                solved=1;
                                                c [i + 1] = 10;
                                        }
                                        StartCoroutine (waittransf (ctable [i], c [i + 1], se));
                                }
                                se = se + 0.6f;
                                for (int i=1; i<=proc; i++) {
                                        if (c [i] < c [i + 1]) {
                                                left = last [i];
                                                right = last [i + 1];
                                        }
                                }
                                if (c [0] < c [1]) {
                                        left = last [0];
                                        right = last [1];
                                }
                                length = right - left;
                        }
                }
                firsttime++;
                        if (solved == 0) {
                        while (length>proc) {
                                colorcount = 0;
                                int counter = 0;

                                divider = length / (proc + 1);
                                int div3 = (length-1) - divider * proc;
                                if ((div3 > proc) & (length / (divider) != proc + 1))
                                        divider++;
                                for (int i=left; i<right; i++) {
                                        counter++;
                                        if (i < num) {
                                                Vector3 position = new Vector3 ((vectors [i]), (steps * stepSize),
0);
                                                if (counter <= divider * proc)
                                                        StartCoroutine (waitonesecond (position, table [i], se,
colortable [colorcount]));
                                                else
                                                        StartCoroutine (waitonesecond (position, table [i], se,
Color.white));
                                                if ((counter % divider == 0) && (colorcount < proc))
                                                        colorcount++;
                                        }
                                }



                                last [0] = left;
                                last [proc + 1] = right;

                                for (int i=0; i<proc; i++) {
                                        int temp=0;
                                        temp=Mathf.RoundToInt((float)(right-left)/(proc+1));
                                        last [i + 1] = left + (i + 1) * temp;
                                        StartCoroutine (waittransf (lasttable [i], table [last [i + 1] - 1], 0.6f));
                                        if (table [last [i + 1] - 1] > search)
                                                c [i + 1] = 1;
                                        else if (table [last [i + 1] - 1] < search)
                                                c [i + 1] = 0;
                                        else {
                                                procfound = i;
                                                solfound = last [i + 1] - 1;
                                                StartCoroutine (waittransf (sol, "Sol", se));
                                                solved = 1;
                                        }
                                        StartCoroutine (waittransf (ctable [i], c [i + 1], se));
                                }

                                for (int i=1; i<=proc; i++) {
                                        if (c [i] < c [i + 1]) {
                                                left = last [i];
                                                right = last [i + 1];
                                        }
                                }
```

```
                                        if (c [0] < c [1]) {
                                                left = last [0];
                                                right = last [1];
                                        }
                                        length = right - left;
                                        steps++;
                                        se = se + 0.6f;
                                        if (templen == length)
                                                break;
                                        templen = length;
                                }
                        }

                                colorcount = 0;
                        if (right > num) {
                                right--;
                                left--;
                        }

                        if (solfound != 0) {
                                Vector3 position = new Vector3 ((vectors [solfound]), (steps * stepSize), 0);
                                StartCoroutine (waitonesecond (position, table [solfound], se, colortable [procfound]));
                                        StartCoroutine (waittransf (sol, ""+(solfound+1), se));
                        }
                        else {

                                for (int i=left; i<right; i++) {
                                        Vector3 position = new Vector3 ((vectors [i]), (steps * stepSize), 0);
                                        StartCoroutine (waitonesecond (position, table [i], se, colortable [colorcount]));
                                        colorcount++;
                                }
                                        for (int i=0; i<num; i++)
                                                if (table[i]<=search)
                                                        solved=i+1;
                                        StartCoroutine (waittransf (sol, solved, se));
                        }
                                finished++;
                        }
                }


                public void executebystep () {
                        float se = 0.6f;
                        int procfound = 0;


                        if (searchnum.text.Length != 0) {
                                if (finished == 0)
                                if (firsttime == 0) {
                                        search = Int32.Parse (searchnum.text);


                                        left = 0;
                                        right = num + 1;
                                        c [0] = 0;
                                        c [proc + 1] = 1;
                                        length = right - left;

                                        if ((length - 1) < proc) {
                                                for (int i=0; i<length-1; i++) {
                                                        StartCoroutine (waitcolor (tableobject [i], table [i], colortable
[colorcount]));
                                                        StartCoroutine (waittransf (lasttable [i], table [i], se));
                                                        colorcount++;
                                                }
                                                right--;
                                        } else {
                                                divider = length / (proc + 1);
                                                int div2 = num - divider * proc;
                                                if ((div2 > proc) & (num / divider != proc + 1))
                                                        divider++;

                                                for (int i=1; i<=num; i++) {
                                                        if (i <= divider * proc)
```

```
                                                StartCoroutine (waitcolor (tableobject [i - 1], table [i -
1], colortable [colorcount]));
                                        else
                                                StartCoroutine (waitcolor (tableobject [i - 1], table [i -
1], Color.white));
                                        if ((i % divider == 0) && (colorcount < proc))
                                                colorcount++;
                                }


                                last [0] = left;
                                last [proc + 1] = right;

                                for (int i=0; i<proc; i++) {
                                        last [i + 1] = left + (i + 1) * ((right - left) / (proc + 1));
                                        StartCoroutine (waittransf (lasttable [i], table [last [i + 1] - 1], se));
                                        if (table [last [i + 1] - 1] > search)
                                                c [i + 1] = 1;
                                        else if (table [last [i + 1] - 1] < search)
                                                c [i + 1] = 0;
                                        else {
                                                procfound = i;
                                                solfound = last [i + 1] - 1;
                                                solved = 1;
                                                c [i + 1] = 1;
                                        }
                                        StartCoroutine (waittransf (ctable [i], c [i + 1], se));
                                }
                                se = se + 0.6f;
                                for (int i=1; i<=proc; i++) {
                                        if (c [i] < c [i + 1]) {
                                                left = last [i];
                                                right = last [i + 1];
                                        }
                                }
                                if (c [0] < c [1]) {
                                        left = last [0];
                                        right = last [1];
                                }
                                length = right - left;
                        }
                }
                else {


                        if ((length > proc)&(not==0)) {
                                colorcount = 0;
                                int counter = 0;

                                divider = length / (proc + 1);
                                int div3 = (length - 1) - divider * proc;
                                if ((div3 > proc) & (length / (divider) != proc + 1))
                                        divider++;

                                for (int i=left; i<right; i++) {
                                        counter++;
                                        if (i < num) {
                                                Vector3 position = new Vector3 ((vectors [i]), (steps *
stepSize), 0);
                                                if (counter <= divider * proc)
                                                        StartCoroutine (waitonesecond (position,
table [i], se, colortable [colorcount]));
                                                else
                                                        StartCoroutine (waitonesecond (position,
table [i], se, Color.white));
                                                if ((counter % divider == 0) && (colorcount < proc))
                                                        colorcount++;
                                        }
                                }

                                last [0] = left;
                                last [proc + 1] = right;
```

```
                                        for (int i=0; i<proc; i++) {
                                                int temp=0;
                                                temp=Mathf.RoundToInt((float)(right-left)/(proc+1));
                                                last [i + 1] = left + (i + 1) * temp;
                                                StartCoroutine (waittransf (lasttable [i], table [last [i + 1] - 1],
0.6f));

                                                if (table [last [i + 1] - 1] > search)
                                                        c [i + 1] = 1;
                                                else if (table [last [i + 1] - 1] < search)
                                                        c [i + 1] = 0;
                                                else {
                                                        procfound = i;
                                                        solfound = last [i + 1] - 1;
                                                        solved = 1;
                                                }
                                                StartCoroutine (waittransf (ctable [i], c [i + 1], se));
                                        }

                                        for (int i=1; i<=proc; i++) {
                                                if (c [i] < c [i + 1]) {
                                                        left = last [i];
                                                        right = last [i + 1];
                                                }
                                        }
                                        if (c [0] < c [1]) {
                                                left = last [0];
                                                right = last [1];
                                        }
                                        length = right - left;
                                        steps++;
                                        se = se + 0.6f;
                                        if (templen == length)
                                                not=1;
                                        templen = length;
                                }

                                else {

                                        colorcount = 0;
                                        if (right > num) {
                                                right--;
                                                left--;
                                        }
                                        if (solfound != 0) {
                                                Vector3 position = new Vector3 ((vectors [solfound]), (steps *
stepSize), 0);

                                                StartCoroutine (waitonesecond (position, table [solfound], se,
colortable [procfound]));

                                                StartCoroutine (waittransf (sol, ""+(solfound+1), se));
                                        } else {


                                                for (int i=left; i<right; i++) {
                                                        Vector3 position = new Vector3 ((vectors [i]), (steps *
stepSize), 0);

                                                        StartCoroutine (waitonesecond (position, table [i], se,
colortable [colorcount]));

                                                        colorcount++;
                                                }
                                                for (int i=0; i<num; i++)
                                                        if (table[i]<=search)
                                                                solved=i+1;
                                                StartCoroutine (waittransf (sol, solved, se));

                                        }
                                        finished++;
                                }
                        }
                        firsttime++;
                }
        }
}
```

A.3 Αλγόριθμος Παράλληλης Συγχώνευσης

```
using UnityEngine;
using System.Collections;
using System;
using UnityEngine.UI;

public class Rank : MonoBehaviour {

	public InputField numOfProcessorsText;  // reference on input field
	public InputField numofn;
	public InputField searchnum;
	public InputField Acapacity;
	public InputField Bcapacity;
	public GameObject processorPrefab;			// reference on game object



	private int steps = 0;						// step of the algorithm
	private float stepSize = 1.0f;				// for the position of object
	int proc;									//proc given by the user
	int search;
	public int []table;							//temp table for the tree
	public int []Atable;						//temp table for the tree
	public int []Btable;						//temp table for the tree
	public float [] vectors;					//temp table for positions of vectors
	public float [] Avectors;					//temp table for positions of vectors
	public float [] Bvectors;					//temp table for positions of vectors
	public Color[] colortable;					//table with colors
	double sqrt;
	double sqrtA;
	public GameObject [] rankobj;
	public int [] rank;
	int Alength=0;
	int Blength=0;
	public GameObject[] Bobject;
	public GameObject[] lastobj;				//table with reference on each last
	public GameObject[] cobj;
	public GameObject[] tableobj;				//table with reference on each gameobject of the tree
	public GameObject[] phase2obj;
	int divider=0;
	float col1=1.20f;
	float col2=-0.10f;
	float col3=-0.13f;
	int templen=0;
	int length=0;
	int colorcount=0;
	int firsttime=0;
	public int right;
	public int left;
	public int [] c;
	public int [] last;
	int solved=0;
	public int [] r;
	int [,]Asubl;
	public int clicktimes=0;

	public void Reset() {			//reset the algorithm
		Application.LoadLevel("Rank Scene");

	}

	public void Update() {
		if (Input.GetKeyDown(KeyCode.Escape)) {
			Application.LoadLevel ("MainScene");
		}
	}

	IEnumerator waittransf(GameObject ob,int val,float se){//insert value to a gameobject after a delay
		yield return new WaitForSeconds(se);
		ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + val;
	}
```

```
	IEnumerator waittransf(GameObject ob,string val,float se){//insert value to a gameobject after a delay
		yield return new WaitForSeconds(se);
		ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + val;
	}

	IEnumerator waittransfphase2(int i,int val,float se){//insert value to a gameobject after a delay
		yield return new WaitForSeconds(se);
		phase2obj[i].transform.FindChild ("Value").GetComponent<TextMesh> ().text = ""+val;
	}

	IEnumerator waitonesecond(Vector3 position,int value,float se,Color newc){ //insert color and value to a processor after a
delay
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);

		go.GetComponent <MeshRenderer> ().material.color = newc;
	}

	IEnumerator waitonesecondtag(Vector3 position,int value,float se,Color newc){ //insert color and value to a processor
after a delay
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);

		go.GetComponent <MeshRenderer> ().material.color = newc;
		go.transform.tag = "Finish";
	}


	IEnumerator waitonesecondtag(Vector3 position,string value,float se,Color newc){ //insert color and value to a processor
after a delay
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = value;
		yield return new WaitForSeconds(0.6f);

		go.GetComponent <MeshRenderer> ().material.color = newc;
		go.transform.tag = "Finish";
	}

	IEnumerator waitonesecondtagv2(Vector3 position,int value,float se,Color newc,int i){ //insert color and value to a
processor after a delay
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);

		go.GetComponent <MeshRenderer> ().material.color = newc;
		go.transform.tag = "Finish";
		phase2obj [i] = go;
	}

	IEnumerator waitonesecondtagv3(Vector3 position,string value,float se,Color newc){ //insert color and value to a
processor after a delay
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = value;
		yield return new WaitForSeconds(0.6f);

		go.GetComponent <MeshRenderer> ().material.color = newc;
		go.transform.tag = "Player";
	}

	IEnumerator waitonesecondtagv3(Vector3 position,int value,float se,Color newc){ //insert color and value to a processor
after a delay
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text =""+ value;
		yield return new WaitForSeconds(0.6f);
```

```
		go.GetComponent <MeshRenderer> ().material.color = newc;
		go.transform.tag = "Player";
	}

	IEnumerator waitonesecondv2(Vector3 position,int value,float se,Color newc,GameObject[] list,int i){ //insert color and value to a processor after a delay
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);

		go.GetComponent <MeshRenderer> ().material.color = newc;
		list [i] = go;
	}

	IEnumerator waitcolor(Vector3 position,int value,Color newc){  //insert color and value to a processor after a delay
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);

		go.GetComponent <MeshRenderer> ().material.color = newc;
	}

	IEnumerator destroyall(float se){  //insert color and value to a processor after a delay
		yield return new WaitForSeconds(se);
		GameObject[] list = GameObject.FindGameObjectsWithTag ("Finish");
		for (int i=0; i<list.Length; i++) {
			Destroy(list[i]);
		}
	}

	IEnumerator destroyallv2(float se){  //insert color and value to a processor after a delay
		yield return new WaitForSeconds(se);
		GameObject[] list = GameObject.FindGameObjectsWithTag ("Player");
		for (int i=0; i<list.Length; i++) {
			Destroy(list[i]);
		}
	}

	IEnumerator waitcolor(GameObject ob,int value,Color newc){//color a gameobject and put an int on it
		ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);

		ob.GetComponent <MeshRenderer> ().material.color = newc;
	}

	IEnumerator waitcolorv2(GameObject ob,Color newc,float se){//color a gameobject and put an int on it
		yield return new WaitForSeconds (se);

		ob.GetComponent <MeshRenderer> ().material.color = newc;
	}

	public void Init2(){
		if ((Acapacity.text.Length != 0) & (Bcapacity.text.Length != 0) & (Alength == 0) & (Blength == 0)) {
			Alength = Int32.Parse (Acapacity.text);
			Blength = Int32.Parse (Bcapacity.text);
			int maxlen = 0;
			if (Alength > Blength)
				maxlen = Alength;
			else
				maxlen = Blength;
			int random = UnityEngine.Random.Range (-5, 8);
			Atable = new int[Alength];
			Btable = new int[Blength];
			Avectors = new float[Alength];
			Bvectors = new float[Blength];
			vectors = new float[maxlen];
			sqrt = Math.Sqrt (Blength);
			sqrtA = Math.Sqrt (Alength);
			rank = new int[Blength];
			rankobj = new GameObject[Blength];
			Bobject = new GameObject[Blength];
			c = new int[maxlen];
			last = new int [maxlen];
```

```
lastobj = new GameObject[maxlen];
cobj = new GameObject[maxlen];
tableobj = new GameObject[maxlen];
r = new int[maxlen];
Asubl = new int[6, maxlen];

float temp = 1.1f;

for (int i=0; i<6; i++)
        for (int j=0; j<Blength; j++) {
                Asubl [i, j] = -100;
        }

for (int i=0; i<Alength; i++) {
        Vector3 position = new Vector3 ((i * stepSize) - 10.5f + temp, -2, 0);
        Avectors [i] = (i * stepSize - 10 + temp);

        Atable [i] = random;
        int random2 = UnityEngine.Random.Range (1, 7);
        random = random + random2;

        GameObject ob1 = Instantiate (processorPrefab, position, Quaternion.identity) as
GameObject;
        StartCoroutine (waittransf (ob1, Atable [i], 0.6f));
}

random = UnityEngine.Random.Range (-5, 8);


colortable = new Color[Blength];


colorcount = 0;

for (int i=0; i<Blength; i++) {
        if (i % sqrt == 0) {
                temp = 1.2f;
                if (col1 < 0)
                        col1 = 1;
                col1 = col1 - 0.20f;
                if (col2 > 1)
                        col2 = 0.30f;
                col2 = col2 + 0.18f;
                if (col3 > 1)
                        col3 = 0.67f;
                col3 = col3 + 0.18f;
                Color newColor = new Color (col1, col2, col3, 1);
                colortable [colorcount] = newColor;
                colorcount++;
        } else
                temp = 1.1f;
        Vector3 position = new Vector3 ((i * stepSize) - 10.5f + temp, -3, 0);
        Bvectors [i] = (i * stepSize - 10 + temp);

        Btable [i] = random;
        int random2 = UnityEngine.Random.Range (1, 7);
        random = random + random2;

        GameObject ob1 = Instantiate (processorPrefab, position, Quaternion.identity) as
GameObject;
        StartCoroutine (waittransf (ob1, Btable [i], 0.6f));


        Vector3 position2 = new Vector3 ((i * stepSize) - 10.5f + 1.1f, -4, 0);
        GameObject ob2 = Instantiate (processorPrefab, position2, Quaternion.identity) as
GameObject;
        StartCoroutine (waittransf (ob2, 0, 0.6f));
        rankobj [i] = ob2;
}
if (col1 < 0)
        col1 = 1;
col1 = col1 - 0.20f;
if (col2 > 1)
        col2 = 0.30f;
```

```
                col2 = col2 + 0.18f;
                if (col3 > 1)
                        col3 = 0.67f;
                col3 = col3 + 0.18f;
                Color newColor2 = new Color (col1, col2, col3, 1);
                colortable [colorcount] = newColor2;
                colorcount++;

                Vector3 pos = new Vector3 (-10.5f, -2, 0);
                GameObject ob = Instantiate (processorPrefab, pos, Quaternion.identity) as GameObject;
                ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "ListA";

                Vector3 pos2 = new Vector3 (-10.5f, -3, 0);
                GameObject ob4 = Instantiate (processorPrefab, pos2, Quaternion.identity) as GameObject;
                ob4.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "ListB";

                Vector3 pos3 = new Vector3 (-10.5f, -4, 0);
                GameObject ob3 = Instantiate (processorPrefab, pos3, Quaternion.identity) as GameObject;
                ob3.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "Rank";

                float se = 0.9f;
                //colorcount = 0;
                for (int i=0; i<Blength; i++) {
                        Vector3 position = new Vector3 ((Bvectors [i]), (steps * stepSize) - 0.5f, 0);
                        StartCoroutine (waitonesecondv2 (position, Btable [i], se, Color.white, Bobject, i));
                }
        }
}

public void exec2 () {
        if (Alength != 0 & Blength != 0) {

                int k2 = 0;
                int slength = 0;
                if (Alength > Blength)
                        slength = Blength;
                else
                        slength = Alength;

                float se = 0.6f;
                int count;
                colorcount = 0;
                proc = (int)sqrtA;
                int sqrint = (int)sqrt;

                if (clicktimes == 0) {
                        for (int i=0; i<Blength; i++) {
                                StartCoroutine (waitcolorv2 (Bobject [i], colortable [colorcount], 0.0f));
                                if ((i + 1) % sqrt == 0)
                                        colorcount++;
                        }
                        se = se + 0.9f;
                }


                for (int times=1; times<=sqrint; times++) {
                        if (clicktimes == 0) {
                                if (Alength > proc) {
                                        divider = Alength / (proc + 1);
                                        int div2 = Alength - divider * proc;
                                        if ((div2 > proc) & (Alength / divider != proc + 1))
                                                divider++;
                                }


                                StartCoroutine (waitcolorv2 (Bobject [times * sqrint - 1], colortable [sqrint],
se));


                                steps = steps + 2;
                                colorcount = -1;

                                float temp = 1.1f;
                                for (int i=0; i<slength; i++) {
                                        if (divider != 0) {
```

```
                        if (i % divider == 0 & (i < (divider * proc))) {
                                temp = 1.2f;
                                colorcount++;
                        } else
                                temp = 1.1f;
                }
                Vector3 position = new Vector3 ((i * stepSize) + temp - 18.0f,
(steps * stepSize + 0.7f), 0);

                vectors [i] = (i * stepSize) + temp - 18.0f;

                GameObject ob1 = Instantiate (processorPrefab, position,
Quaternion.identity) as GameObject;

                StartCoroutine (waittransf (ob1, Atable [i], 0.6f));
                if (divider * proc > i)
                        StartCoroutine (waitcolorv2 (ob1, colortable
[colorcount], se));

                ob1.transform.tag = "Finish";
                tableobj [i] = ob1;
}


for (int i=0; i<proc; i++) {
                Vector3 positionobj = new Vector3 ((Alength * stepSize) + i +
1.1f - 16.0f, (steps * stepSize + 0.7f), 0);

                GameObject obj = Instantiate (processorPrefab, positionobj,
Quaternion.identity) as GameObject;

                StartCoroutine (waitcolor (obj, i, colortable [i]));
                obj.transform.tag = "Finish";
                Vector3 positionobj2 = new Vector3 ((Alength * stepSize) + i +
1.1f - 16.0f, (steps * stepSize + 0.7f) - 1, 0);

                GameObject obj2 = Instantiate (processorPrefab, positionobj2,
Quaternion.identity) as GameObject;

                lastobj [i] = obj2;
                obj2.transform.tag = "Finish";
                Vector3 positionobj3 = new Vector3 ((Alength * stepSize) + i +
1.1f - 16.0f, (steps * stepSize + 0.7f) - 2, 0);

                GameObject obj3 = Instantiate (processorPrefab, positionobj3,
Quaternion.identity) as GameObject;

                cobj [i] = obj3;
                obj3.transform.tag = "Finish";
}

Vector3 pos = new Vector3 ((Alength * stepSize) + 1.1f - 17.0f, (steps *
stepSize + 0.7f), 0);

GameObject ob = Instantiate (processorPrefab, pos, Quaternion.identity) as
GameObject;

ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text =
"proc";

ob.transform.tag = "Finish";
Vector3 pos2 = new Vector3 ((Alength * stepSize) + 1.1f - 17.0f, (steps *
stepSize + 0.7f) - 1, 0);

GameObject ob2 = Instantiate (processorPrefab, pos2, Quaternion.identity)
as GameObject;

ob2.transform.FindChild ("Value").GetComponent<TextMesh> ().text =
"last";

ob2.transform.tag = "Finish";
Vector3 pos3 = new Vector3 ((Alength * stepSize) + 1.1f - 17.0f, (steps *
stepSize + 0.7f) - 2, 0);

GameObject ob3 = Instantiate (processorPrefab, pos3, Quaternion.identity)
as GameObject;

ob3.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "c";
ob3.transform.tag = "Finish";
Vector3 pos4 = new Vector3 ((Alength * stepSize) + 1.1f - 9.0f, (steps *
stepSize + 0.7f), -1);

GameObject ob4 = Instantiate (processorPrefab, pos4, Quaternion.identity)
as GameObject;

ob4.transform.FindChild ("Value").GetComponent<TextMesh> ().text =
"Num";

ob4.transform.tag = "Finish";
```

```
count = 0;
solved = 0;
if (firsttime == 0) {
        search = Btable [times * sqrint - 1];

        Vector3 pos5 = new Vector3 ((Alength * stepSize) + 1.1f - 8.0f,
(steps * stepSize + 0.7f), -1);

        GameObject ob5 = Instantiate (processorPrefab, pos5,
Quaternion.identity) as GameObject;

        ob5.transform.FindChild ("Value").GetComponent<TextMesh>
().text = "" + search;

        ob5.GetComponent <MeshRenderer> ().material.color =
colortable [sqrint];

        ob5.transform.tag = "Finish";

        steps++;
        left = 0;
        right = Alength + 1;
        c [0] = 0;
        c [proc + 1] = 1;
        length = right - left;



        divider = length / (proc + 1);
        int div2 = Alength - divider * proc;
        if ((div2 > proc) & (Alength / divider != proc + 1))
                divider++;



        if ((length - 1) < proc) {
                for (int i=0; i<length-1; i++) {
                        StartCoroutine (waitcolor (tableobj [i],
Atable [i], colortable [colorcount]));
                        StartCoroutine (waittransf (lastobj [i],
Atable [i], se));
                        colorcount++;
                        if (Atable [i] <= search)
                                count++;
                }
        } else {
                for (int i=1; i<=slength; i++) {
                        if (i <= divider * proc)
                                StartCoroutine          (waitcolor
(tableobj [i - 1], Atable [i - 1], colortable [colorcount]));
                        else
                                StartCoroutine          (waitcolor
(tableobj [i - 1], Atable [i - 1], Color.white));
                        if ((i % divider == 0) && (colorcount <
proc))
                                colorcount++;
                }



                last [0] = left;
                last [proc + 1] = right;

                for (int i=0; i<proc; i++) {
                        last [i + 1] = left + (i + 1) * ((right - left) /
(proc + 1));
                        StartCoroutine (waittransf (lastobj [i],
Atable [last [i + 1] - 1], se));
                        if (Atable [last [i + 1] - 1] > search)
                                c [i + 1] = 1;
                        else if (Atable [last [i + 1] - 1] < search) {
                                c [i + 1] = 0;
                                if (count < last [i + 1] - 1)
                                        count = last [i + 1] -
1;
                        } else {
                                solved = 1;
```

```
                                                                        //c [i + 1] = 10;
                                                                        if (count < last [i + 1] - 1)
                                                                                count = last [i + 1] -
1;
                                                                }
                                                                StartCoroutine (waittransf (cobj [i], c [i +
1], se));
                                                        }
                                                        for (int i=1; i<=proc; i++) {
                                                                if (c [i] < c [i + 1]) {
                                                                        left = last [i];
                                                                        right = last [i + 1];
                                                                }
                                                        }
                                                        if (c [0] < c [1]) {
                                                                left = last [0];
                                                                right = last [1];
                                                        }
                                                        length = right - left;
                                                }
                                        }
                                        firsttime++;


                                        if (solved == 0) {

                                                while (length>proc) {
                                                        colorcount = 0;
                                                        divider = length / (proc + 1);
                                                        int div3 = (length - 1) - divider * proc;
                                                        if ((div3 > proc) & (length / (divider) != proc + 1))
                                                                divider++;


                                                        int counter = 0;

                                                        for (int i=left; i<right; i++) {
                                                                counter++;
                                                                if (i < slength) {
                                                                        Vector3    position    =    new
Vector3 ((vectors [i]), (steps * stepSize + 0.7f), 0);

                                                                        if (counter <= divider * proc)
                                                                                StartCoroutine
(waitonesecondtag (position, Atable [i], se, colortable [colorcount]));
                                                                        else
                                                                                StartCoroutine
(waitonesecondtag (position, Atable [i], se, Color.white));

                                                                        if ((counter % divider == 0) &&
(colorcount < proc))
                                                                                colorcount++;
                                                                }
                                                        }

                                                        last [0] = left;
                                                        last [proc + 1] = right;

                                                        for (int i=0; i<proc; i++) {
                                                                last [i + 1] = left + (i + 1) * ((right - left) /
(proc + 1));
                                                                StartCoroutine    (waittransf    (lastobj    [i],
Atable [last [i + 1] - 1], se));
                                                                if (Atable [last [i + 1] - 1] > search)
                                                                        c [i + 1] = 1;
                                                                else if (Atable [last [i + 1] - 1] < search) {
                                                                        c [i + 1] = 0;

                                                                        if (count < last [i + 1] - 1)
                                                                                count = last [i + 1] -
1;
                                                                } else {
                                                                        solved = 1;
                                                                        if (count < last [i + 1] - 1)
```

```
                                                                                count = last [i + 1] -
1;
                                                                                        }
                                                                                        StartCoroutine (waittransf (cobj [i], c [i +
1], se));
                                                                                }

                                                                                for (int i=1; i<=proc; i++) {
                                                                                        if (c [i] < c [i + 1]) {
                                                                                                left = last [i];
                                                                                                right = last [i + 1];
                                                                                        }
                                                                                }
                                                                                if (c [0] < c [1]) {
                                                                                        left = last [0];
                                                                                        right = last [1];
                                                                                }
                                                                                length = right - left;
                                                                                steps++;
                                                                                se = se + 0.6f;
                                                                                if (templen == length)
                                                                                        break;
                                                                                templen = length;
                                                                        }
                                                                }
                                                                colorcount = 0;
                                                                if (solved == 0) {
                                                                        if (right > slength) {
                                                                                left--;
                                                                                right--;
                                                                        }
                                                                        for (int i=left; i<right; i++)
                                                                                if (Atable [i] <= search)
                                                                                        count = i;
                                                                }



                                                                for (int i=left; i<right; i++) {
                                                                        if (i < slength) {
                                                                                Vector3 position = new Vector3 ((vectors [i]), (steps *
stepSize + 0.7f), 0);
                                                                                StartCoroutine (waitonesecondtag (position, Atable
[i], se, colortable [colorcount]));
                                                                                colorcount++;
                                                                        }
                                                                }

                                                                for (int i=0; i<Alength; i++) {
                                                                        if (Atable [i] <= search)
                                                                                count = i + 1;
                                                                }
                                                                rank [sqrint * times - 1] = count;
                                                                StartCoroutine (waittransf (rankobj [sqrint * times - 1], count, 0.6f));
                                                                firsttime = 0;

                                                                r [times] = count;


                                                        }
                                                }
                                                if (clicktimes == 0)
                                                        clicktimes++;

                                                if (clicktimes == 1) {
                                                        StartCoroutine (destroyall (5.0f));


                                                        steps = 2;
                                                        int temp2 = 0;
                                                        colorcount = 0;
                                                        for (int i=0; i<sqrint; i++) {
                                                                for (int j=0; j<sqrint-1; j++) {
```

```
                                        Vector3 position = new Vector3 ((j * 1.0f) - 16.0f, (steps *
stepSize + 0.7f), 0);
                                        StartCoroutine (waitonesecondtag (position, Btable [temp2], 5.0f,
colortable [colorcount]));

                                        temp2++;
                                }
                                Vector3 position2 = new Vector3 (-17.0f, (steps * stepSize + 0.7f), 0);
                                StartCoroutine (waitonesecondtag (position2, "B" + i, 5.0f, Color.white));
                                colorcount++;
                                steps = steps + 2;
                                temp2++;
                        }

                        steps = 2;
                        colorcount = 0;
                        int sqrt2 = (int)Math.Sqrt (sqrint - 1);
                        int maxj = 0;
                        for (int i=1; i<=sqrint; i++) {
                                int posi = 1;
                                for (int j=r[i-1]; j<r[i]; j++) {
                                        Vector3 position = new Vector3 ((sqrint * stepSize) + (posi * 1.1f)
- 16.0f, (steps * stepSize + 0.7f), 0);

                                        StartCoroutine (waitonesecondtag (position, Atable [j], 5.0f,
colortable [colorcount]));
                                        Asubl [(i - 1), (posi - 1)] = Atable [j];
                                        posi++;
                                        if (posi > maxj)
                                                maxj = posi;

                                }
                                Vector3 position2 = new Vector3 ((sqrint * stepSize) - 16.0f, (steps * stepSize
+ 0.7f), 0);
                                StartCoroutine (waitonesecondtag (position2, "A" + (i - 1), 5.0f,
Color.white));
                                colorcount++;
                                steps = steps + 2;
                        }




                        steps = 0;
                        int count1 = 0;
                        phase2obj = new GameObject[Blength];
                        int numoflist = 0;
                        int[] search2 = new int[3];
                        for (int times=1; times<=sqrint; times++) {
                                numoflist = 0;
                                steps = steps + 2;
                                colorcount = -1;
                                divider = sqrt2;
                                float temp = 1.1f;
                                for (int i=0; i<sqrint-1; i++) {
                                        if (i % divider == 0) {
                                                temp = 1.2f;
                                                colorcount++;
                                        } else
                                                temp = 1.1f;

                                        Vector3 position = new Vector3 ((sqrint * stepSize) + (maxj *
1.1f) - 16.0f + temp + (i * stepSize), (steps * stepSize + 0.7f), 0);

                                        if ((i + 1) % divider == 0) {
                                                StartCoroutine (waitonesecondtag (position, Btable
[count1], 5.0f, colortable [sqrint]));

                                                search2 [numoflist] = Btable [count1];
                                                numoflist++;
                                        } else
                                                StartCoroutine (waitonesecondtag (position, Btable
[count1], 5.0f, colortable [colorcount]));
                                        count1++;
```

```
}

Vector3 position5 = new Vector3 ((sqrint * stepSize) + (maxj * 1.1f) - 17.0f
+ temp, ((steps - 1) * stepSize + 0.7f), 0);
StartCoroutine (waitonesecondtag (position5, "Rank", 5.0f, Color.white));

length = 0;
for (int k=0; k<Blength; k++) {
        if (Asubl [times - 1, k] != -100)
                length++;
}
proc = sqrt2;

if (length != 0) {
        divider = length / (proc + 1);
        int div4 = (length - 1) - divider * proc;
        if (divider != 0)
        if ((div4 > proc) & (length / (divider) != proc + 1))
                divider++;


        for (int i=0; i<numoflist; i++) {
                colorcount = -1;
                Vector3 position3 = new Vector3 ((sqrint * stepSize)
+ (maxj * 1.1f) - 14.0f + (sqrint * stepSize) + (i * (maxj) * temp), ((steps - 1) * stepSize + 0.7f), 0);
                StartCoroutine (waitonesecondtagv3 (position3,
"Num", 5.0f, Color.white));
                position3 = new Vector3 ((sqrint * stepSize) + (maxj
* 1.1f) - 13.0f + (sqrint * stepSize) + (i * (maxj) * temp), ((steps - 1) * stepSize + 0.7f), 0);
                StartCoroutine (waitonesecondtagv3 (position3,
search2 [i], 5.0f, colortable [sqrint]));
                int j = times - 1;
                for (int k=0; k<Blength; k++) {
                        if (Asubl [j, k] != -100) {
                                if (divider != 0) {
                                        if (k % divider == 0)

        colorcount++;
                                }
                                Vector3 position = new
Vector3 ((sqrint * stepSize) + (maxj * 1.1f) - 14.0f + (k * temp) + (sqrint * stepSize) + (i * (maxj) * temp), (steps * stepSize + 0.7f),
0);

                                if (k < proc * divider)
                                        StartCoroutine
(waitonesecondtagv3 (position, Asubl [j, k], 5.0f, colortable [colorcount]));
                                else if (length <= proc) {
                                        colorcount++;
                                        StartCoroutine
(waitonesecondtagv3 (position, Asubl [j, k], 5.0f, colortable [colorcount]));
                                } else
                                        StartCoroutine
(waitonesecondtagv3 (position, Asubl [j, k], 5.0f, Color.white));

                        }
                }
        }
}
count1++;



int k1 = 0;

int[] temprank = new int[sqrint];

for (int i=0; i<sqrint-1; i++) {

        Vector3 position2 = new Vector3 ((sqrint * stepSize) + (maxj *
1.1f) - 16.0f + temp + (i * stepSize), ((steps - 1) * stepSize + 0.7f), 0);
        if ((i + 1) % sqrt2 == 0) {
                count = 0;
                for (int j=0; j<length; j++)
                        if (Asubl [times - 1, j] <= search2 [k1])
```

```
                                                count++;
                                        StartCoroutine (waitonesecondtagv2 (position2,
count, 5.0f, Color.white, k2));
                                        k1++;
                                        temprank [i] = count;
                                } else
                                        StartCoroutine (waitonesecondtagv2 (position2, 0,
5.0f, Color.white, k2));
                                k2++;
                        }
                        k2++;

                        if (Blength == 25) {
                                for (int j=0; j<sqrint-1; j++) {
                                        if ((j + 1) % sqrt2 != 0) {
                                                count = 0;
                                                int posnum = j + (sqrint * (times - 1));
                                                if (j == 2) {
                                                        for (int k=temprank[1];
k<length; k++)
                                                                if (Asubl [times - 1,
k] <= Btable [posnum])
                                                                        count++;
                                                } else {
                                                        for (int k=0; k<length; k++)
                                                                if (Asubl [times - 1,
k] <= Btable [posnum])
                                                                        count++;
                                                }
                                                StartCoroutine (waittransfphase2 (posnum,
count, 6.8f));
                                                temprank [j] = count;
                                        }
                                }
                        }


                        StartCoroutine (destroyallv2 (8.0f));


                        Vector3 position4 = new Vector3 ((sqrint * stepSize) + (maxj * 1.1f) - 15.0f
+ (sqrint * stepSize), ((steps - 1) * stepSize + 0.7f), 0);
                        StartCoroutine (waitonesecondtagv3 (position4, "K", 8.5f, Color.white));


                        for (int k=0; k<sqrint-1; k++) {
                                int posnum = k + (sqrint * (times - 1));
                                Vector3 position = new Vector3 ((sqrint * stepSize) + (maxj *
1.1f) - 14.0f + (k * temp) + (sqrint * stepSize), (steps * stepSize + 0.7f), 0);
                                StartCoroutine (waitonesecondtagv3 (position, Btable [posnum],
8.5f, Color.white));
                                Vector3 position6 = new Vector3 ((sqrint * stepSize) + (maxj *
1.1f) - 14.0f + (k * temp) + (sqrint * stepSize), ((steps - 1) * stepSize + 0.7f), 0);
                                StartCoroutine (waitonesecondtagv3 (position6, k + 1, 8.5f,
Color.white));
                        }

                        for (int k=1; k<sqrint-1; k++) {
                                if (k % sqrt2 == 0) {
                                        temprank [k] = temprank [k] + temprank [(k + 1) /
sqrt2];
                                        StartCoroutine (waittransfphase2 (k + (times - 1) *
sqrint, temprank [k], 11.0f));
                                }
                        }
                        for (int k=0; k<sqrint-1; k++) {
                                if (temprank [k] != 0)
                                        rank [k + (times - 1) * sqrint] = temprank [k];
                        }

                }
                clicktimes++;
```

```
                                }
                                if (clicktimes == 2) {
                                        StartCoroutine (destroyallv2 (13.0f));
                                        StartCoroutine (destroyall (13.0f));

                                        steps = 1;
                                        Vector3 position7 = new Vector3 (-17.0f, ((steps - 1) * stepSize + 0.7f), 0);
                                        StartCoroutine (waitonesecondtag (position7, "K", 13.5f, Color.white));


                                        for (int k=0; k<Blength; k++) {
                                                Vector3 position = new Vector3 (-16.0f + k * (1.0f), (steps * stepSize + 0.7f),
0);
                                                StartCoroutine (waitonesecondtagv3 (position, Btable [k], 13.5f,
Color.white));
                                                Vector3 position6 = new Vector3 (-16.0f + (k * 1.0f), ((steps - 1) * stepSize
+ 0.7f), 0);
                                                StartCoroutine (waitonesecondtagv3 (position6, k + 1, 13.5f, Color.white));
                                                StartCoroutine (waittransf (rankobj [k], rank [k], 13.5f));
                                        }

                                        for (int k=1; k<=Blength; k++) {
                                                if (k % sqrint != 0) {
                                                        rank [k - 1] = rank [k - 1] + r [k / sqrint];
                                                }
                                                StartCoroutine (waittransf (rankobj [k - 1], rank [k - 1], 15.5f));
                                        }
                                        int posinum = 0;
                                        for (int k=0; k<Blength; k++) {
                                                if (rank [k] == 0) {
                                                        Vector3 position = new Vector3 (-17.0f + posinum * (1.0f),
((steps + 2) * stepSize + 0.7f), 0);
                                                        StartCoroutine (waitonesecondtagv3 (position, Btable [k], 17.0f,
Color.white));
                                                        posinum++;
                                                }
                                        }
                                        for (int k=0; k<Alength; k++) {
                                                Vector3 position = new Vector3 (-17.0f + posinum * (1.0f), ((steps + 2) *
stepSize + 0.7f), 0);
                                                posinum++;
                                                StartCoroutine (waitonesecondtagv3 (position, Atable [k], 17.0f,
Color.white));
                                                for (int j=0; j<Blength; j++) {
                                                        if (rank [j] == k + 1) {
                                                                Vector3 position6 = new Vector3 (-17.0f + (posinum
* 1.0f), ((steps + 2) * stepSize + 0.7f), 0);
                                                                StartCoroutine (waitonesecondtagv3 (position6,
Btable [j], 17.0f, Color.white));
                                                                posinum++;
                                                        }
                                                }
                                        }
                                }
                        }
                }

                public void executebystep ()  {
                        if (Alength != 0 & Blength != 0) {
                                int k2 = 0;
                                int slength = 0;
                                if (Alength > Blength)
                                        slength = Blength;
                                else
                                        slength = Alength;

                                float se = 0.6f;
                                int count;
                                colorcount = 0;
                                proc = (int)sqrtA;
                                int sqrint = (int)sqrt;

                                if (clicktimes == 0) {
                                        for (int i=0; i<Blength; i++) {
```

```
                                        StartCoroutine (waitcolorv2 (Bobject [i], colortable [colorcount], 0.0f));
                                        if ((i + 1) % sqrt == 0)
                                                colorcount++;
                                }
                                se = se + 0.9f;
                        }

                        for (int times=1; times<=sqrint; times++) {

                                if (Alength > proc) {
                                        divider = Alength / (proc + 1);
                                        int div2 = Alength - divider * proc;
                                        if ((div2 > proc) & (Alength / divider != proc + 1))
                                                divider++;
                                }

                                if (clicktimes == 0) {

                                        StartCoroutine (waitcolorv2 (Bobject [times * sqrint - 1], colortable [sqrint],
se));


                                        steps = steps + 2;
                                        colorcount = -1;

                                        float temp = 1.1f;
                                        for (int i=0; i<slength; i++) {
                                                if (divider != 0) {
                                                        if (i % divider == 0 & (i < (divider * proc))) {
                                                                temp = 1.2f;
                                                                colorcount++;
                                                        } else
                                                                temp = 1.1f;
                                                }
                                                Vector3 position = new Vector3 ((i * stepSize) + temp - 18.0f,
(steps * stepSize + 0.7f), 0);
                                                vectors [i] = (i * stepSize) + temp - 18.0f;

                                                GameObject ob1 = Instantiate (processorPrefab, position,
Quaternion.identity) as GameObject;
                                                StartCoroutine (waittransf (ob1, Atable [i], 0.6f));
                                                if (divider * proc > i)
                                                        StartCoroutine (waitcolorv2 (ob1, colortable
[colorcount], se));
                                                ob1.transform.tag = "Finish";
                                                tableobj [i] = ob1;
                                        }



                                        for (int i=0; i<proc; i++) {
                                                Vector3 positionobj = new Vector3 ((Alength * stepSize) + i +
1.1f - 16.0f, (steps * stepSize + 0.7f), 0);
                                                GameObject obj = Instantiate (processorPrefab, positionobj,
Quaternion.identity) as GameObject;
                                                StartCoroutine (waitcolor (obj, i, colortable [i]));
                                                obj.transform.tag = "Finish";
                                                Vector3 positionobj2 = new Vector3 ((Alength * stepSize) + i +
1.1f - 16.0f, (steps * stepSize + 0.7f) - 1, 0);
                                                GameObject obj2 = Instantiate (processorPrefab, positionobj2,
Quaternion.identity) as GameObject;
                                                lastobj [i] = obj2;
                                                obj2.transform.tag = "Finish";
                                                Vector3 positionobj3 = new Vector3 ((Alength * stepSize) + i +
1.1f - 16.0f, (steps * stepSize + 0.7f) - 2, 0);
                                                GameObject obj3 = Instantiate (processorPrefab, positionobj3,
Quaternion.identity) as GameObject;
                                                cobj [i] = obj3;
                                                obj3.transform.tag = "Finish";
                                        }

                                        Vector3 pos = new Vector3 ((Alength * stepSize) + 1.1f - 17.0f, (steps *
stepSize + 0.7f), 0);
                                        GameObject ob = Instantiate (processorPrefab, pos, Quaternion.identity) as
GameObject;
```

```
ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text =
"proc";

ob.transform.tag = "Finish";
Vector3 pos2 = new Vector3 ((Alength * stepSize) + 1.1f - 17.0f, (steps *
stepSize + 0.7f) - 1, 0);
GameObject ob2 = Instantiate (processorPrefab, pos2, Quaternion.identity)
as GameObject;
ob2.transform.FindChild ("Value").GetComponent<TextMesh> ().text =
"last";

ob2.transform.tag = "Finish";
Vector3 pos3 = new Vector3 ((Alength * stepSize) + 1.1f - 17.0f, (steps *
stepSize + 0.7f) - 2, 0);
GameObject ob3 = Instantiate (processorPrefab, pos3, Quaternion.identity)
as GameObject;

ob3.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "c";
ob3.transform.tag = "Finish";
Vector3 pos4 = new Vector3 ((Alength * stepSize) + 1.1f - 9.0f, (steps *
stepSize + 0.7f), -1);
GameObject ob4 = Instantiate (processorPrefab, pos4, Quaternion.identity)
as GameObject;
ob4.transform.FindChild ("Value").GetComponent<TextMesh> ().text =
"Num";

ob4.transform.tag = "Finish";



count = 0;
solved = 0;
if (firsttime == 0) {
        search = Btable [times * sqrint - 1];

        Vector3 pos5 = new Vector3 ((Alength * stepSize) + 1.1f - 8.0f,
(steps * stepSize + 0.7f), -1);
        GameObject ob5 = Instantiate (processorPrefab, pos5,
Quaternion.identity) as GameObject;
        ob5.transform.FindChild ("Value").GetComponent<TextMesh>
().text = "" + search;
        ob5.GetComponent <MeshRenderer> ().material.color =
colortable [sqrint];
        ob5.transform.tag = "Finish";

        steps++;
        left = 0;
        right = Alength + 1;
        c [0] = 0;
        c [proc + 1] = 1;
        length = right - left;



        divider = length / (proc + 1);
        int div2 = Alength - divider * proc;
        if ((div2 > proc) & (Alength / divider != proc + 1))
                divider++;



        if ((length - 1) < proc) {
                for (int i=0; i<length-1; i++) {
                        StartCoroutine (waitcolor (tableobj [i],
Atable [i], colortable [colorcount]));
                        StartCoroutine (waittransf (lastobj [i],
Atable [i], se));
                        colorcount++;
                        if (Atable [i] <= search)
                                count++;
                }
        } else {
                for (int i=1; i<=slength; i++) {
                        if (i <= divider * proc)
                                StartCoroutine (waitcolor
(tableobj [i - 1], Atable [i - 1], colortable [colorcount]));
                        else
```

```
                                        StartCoroutine (waitcolor
(tableobj [i - 1], Atable [i - 1], Color.white));
                                    if ((i % divider == 0) && (colorcount <
proc))
                                        colorcount++;
                                }


                                last [0] = left;
                                last [proc + 1] = right;

                                for (int i=0; i<proc; i++) {
                                    last [i + 1] = left + (i + 1) * ((right - left) /
(proc + 1));
                                    StartCoroutine (waittransf (lastobj [i],
Atable [last [i + 1] - 1], se));
                                    if (Atable [last [i + 1] - 1] > search)
                                        c [i + 1] = 1;
                                    else if (Atable [last [i + 1] - 1] < search) {
                                        c [i + 1] = 0;

                                        if (count < last [i + 1] - 1)
                                            count = last [i + 1] -
1;
                                    } else {
                                        solved = 1;
                                        //c [i + 1] = 10;
                                        if (count < last [i + 1] - 1)
                                            count = last [i + 1] -
1;
                                    }
                                    StartCoroutine (waittransf (cobj [i], c [i +
1], se));
                                }
                                for (int i=1; i<=proc; i++) {
                                    if (c [i] < c [i + 1]) {
                                        left = last [i];
                                        right = last [i + 1];
                                    }
                                }
                                if (c [0] < c [1]) {
                                    left = last [0];
                                    right = last [1];
                                }
                                length = right - left;
                            }
                        }
                        firsttime++;


                        if (solved == 0) {

                            while (length>proc) {
                                colorcount = 0;
                                divider = length / (proc + 1);
                                int div3 = (length - 1) - divider * proc;
                                if ((div3 > proc) & (length / (divider) != proc + 1))
                                    divider++;


                                int counter = 0;

                                for (int i=left; i<right; i++) {
                                    counter++;
                                    if (i < slength) {
                                        Vector3 position = new
Vector3 ((vectors [i]), (steps * stepSize + 0.7f), 0);

                                        if (counter <= divider * proc)
                                            StartCoroutine
(waitonesecondtag (position, Atable [i], se, colortable [colorcount]));
                                        else
```

```
                                                                StartCoroutine
(waitonesecondtag (position, Atable [i], se, Color.white));
                                                                if ((counter % divider == 0) &&
(colorcount < proc))
                                                                colorcount++;
                                                        }
                                                }

                                                last [0] = left;
                                                last [proc + 1] = right;

                                                for (int i=0; i<proc; i++) {
                                                        last [i + 1] = left + (i + 1) * ((right - left) /
(proc + 1));
                                                        StartCoroutine (waittransf (lastobj [i],
Atable [last [i + 1] - 1], se));
                                                        if (Atable [last [i + 1] - 1] > search)
                                                                c [i + 1] = 1;
                                                        else if (Atable [last [i + 1] - 1] < search) {
                                                                c [i + 1] = 0;

                                                                if (count < last [i + 1] - 1)
                                                                        count = last [i + 1] -
1;
                                                        } else {
                                                                solved = 1;
                                                                if (count < last [i + 1] - 1)
                                                                        count = last [i + 1] -
1;
                                                        }
                                                        StartCoroutine (waittransf (cobj [i], c [i +
1], se));
                                                }

                                                for (int i=1; i<=proc; i++) {
                                                        if (c [i] < c [i + 1]) {
                                                                left = last [i];
                                                                right = last [i + 1];
                                                        }
                                                }
                                                if (c [0] < c [1]) {
                                                        left = last [0];
                                                        right = last [1];
                                                }
                                                length = right - left;
                                                steps++;
                                                se = se + 0.6f;
                                                if (templen == length)
                                                        break;
                                                templen = length;
                                        }
                                }
                                colorcount = 0;
                                if (solved == 0) {
                                        if (right > slength) {
                                                left--;
                                                right--;
                                        }
                                        for (int i=left; i<right; i++)
                                                if (Atable [i] <= search)
                                                        count = i;
                                }



                                for (int i=left; i<right; i++) {
                                        if (i < slength) {
                                                Vector3 position = new Vector3 ((vectors [i]), (steps *
stepSize + 0.7f), 0);
                                                StartCoroutine (waitonesecondtag (position, Atable
[i], se, colortable [colorcount]));
                                                colorcount++;
                                        }
                                }
```

```
                                        for (int i=0; i<Alength; i++) {
                                                if (Atable [i] <= search)
                                                        count = i + 1;
                                        }
                                        rank [sqrint * times - 1] = count;
                                        StartCoroutine (waittransf (rankobj [sqrint * times - 1], count, 0.6f));
                                        firsttime = 0;

                                        r [times] = count;

                                }
                        }
                        if (clicktimes == 1) {

                                StartCoroutine (destroyall (0.0f));

                                steps = 2;
                                int temp2 = 0;
                                colorcount = 0;
                                for (int i=0; i<sqrint; i++) {
                                        for (int j=0; j<sqrint-1; j++) {
                                                Vector3 position = new Vector3 ((j * 1.0f) - 16.0f, (steps *
stepSize + 0.7f), 0);
                                                StartCoroutine (waitonesecondtag (position, Btable [temp2], 0.6f,
colortable [colorcount]));

                                                temp2++;
                                        }
                                        Vector3 position2 = new Vector3 (-17.0f, (steps * stepSize + 0.7f), 0);
                                        StartCoroutine (waitonesecondtag (position2, "B" + i, 0.6f, Color.white));
                                        colorcount++;
                                        steps = steps + 2;
                                        temp2++;
                                }

                                steps = 2;
                                colorcount = 0;
                                int sqrt2 = (int)Math.Sqrt (sqrint - 1);
                                int maxj = 0;
                                for (int i=1; i<=sqrint; i++) {
                                        int posi = 1;
                                        for (int j=r[i-1]; j<r[i]; j++) {
                                                Vector3 position = new Vector3 ((sqrint * stepSize) + (posi * 1.1f)
- 16.0f, (steps * stepSize + 0.7f), 0);

                                                StartCoroutine (waitonesecondtag (position, Atable [j], 0.6f,
colortable [colorcount]));
                                                Asubl [(i - 1), (posi - 1)] = Atable [j];
                                                posi++;
                                                if (posi > maxj)
                                                        maxj = posi;

                                        }
                                        Vector3 position2 = new Vector3 ((sqrint * stepSize) - 16.0f, (steps * stepSize
+ 0.7f), 0);
                                        StartCoroutine (waitonesecondtag (position2, "A" + (i - 1), 0.6f,
Color.white));
                                        colorcount++;
                                        steps = steps + 2;
                                }




                                steps = 0;
                                int count1 = 0;
                                phase2obj = new GameObject[Blength];
                                int numoflist = 0;
                                int[] search2 = new int[3];
                                for (int times=1; times<=sqrint; times++) {
                                        numoflist = 0;
                                        steps = steps + 2;
                                        colorcount = -1;
```

```
divider = sqrt2;
float temp = 1.1f;
for (int i=0; i<sqrint-1; i++) {
	if (i % divider == 0) {
		temp = 1.2f;
		colorcount++;
	} else
		temp = 1.1f;

	Vector3 position = new Vector3 ((sqrint * stepSize) + (maxj * 1.1f) - 16.0f + temp + (i * stepSize), (steps * stepSize + 0.7f), 0);

	if ((i + 1) % divider == 0) {
		StartCoroutine (waitonesecondtag (position, Btable [count1], 0.6f, colortable [sqrint]));

		search2 [numoflist] = Btable [count1];
		numoflist++;
	} else
		StartCoroutine (waitonesecondtag (position, Btable [count1], 0.6f, colortable [colorcount]));
	count1++;
}

Vector3 position5 = new Vector3 ((sqrint * stepSize) + (maxj * 1.1f) - 17.0f + temp, ((steps - 1) * stepSize + 0.7f), 0);
StartCoroutine (waitonesecondtag (position5, "Rank", 0.6f, Color.white));

length = 0;
for (int k=0; k<Blength; k++) {
	if (Asubl [times - 1, k] != -100)
		length++;
}
proc = sqrt2;

if (length != 0) {
	divider = length / (proc + 1);
	int div4 = (length - 1) - divider * proc;
	if (divider != 0)
	if ((div4 > proc) & (length / (divider) != proc + 1))
		divider++;


	for (int i=0; i<numoflist; i++) {
		colorcount = -1;
		Vector3 position3 = new Vector3 ((sqrint * stepSize) + (maxj * 1.1f) - 14.0f + (sqrint * stepSize) + (i * (maxj) * temp), ((steps - 1) * stepSize + 0.7f), 0);
		StartCoroutine (waitonesecondtagv3 (position3, "Num", 0.6f, Color.white));
		position3 = new Vector3 ((sqrint * stepSize) + (maxj * 1.1f) - 13.0f + (sqrint * stepSize) + (i * (maxj) * temp), ((steps - 1) * stepSize + 0.7f), 0);
		StartCoroutine (waitonesecondtagv3 (position3, search2 [i], 0.6f, colortable [sqrint]));
		int j = times - 1;
		for (int k=0; k<Blength; k++) {
			if (Asubl [j, k] != -100) {
				if (divider != 0) {
					if (k % divider == 0)

		colorcount++;
				}
				Vector3 position = new Vector3 ((sqrint * stepSize) + (maxj * 1.1f) - 14.0f + (k * temp) + (sqrint * stepSize) + (i * (maxj) * temp), (steps * stepSize + 0.7f), 0);

				if (k < proc * divider)
					StartCoroutine (waitonesecondtagv3 (position, Asubl [j, k], 0.6f, colortable [colorcount]));
				else if (length <= proc) {
					colorcount++;
					StartCoroutine (waitonesecondtagv3 (position, Asubl [j, k], 0.6f, colortable [colorcount]));
				} else
					StartCoroutine (waitonesecondtagv3 (position, Asubl [j, k], 0.6f, Color.white));
```

```
                                                }
                                        }
                                }
                        }
                        count1++;


                        int k1 = 0;

                        int[] temprank = new int[sqrint];

                        for (int i=0; i<sqrint-1; i++) {

                                Vector3 position2 = new Vector3 ((sqrint * stepSize) + (maxj *
1.1f) - 16.0f + temp + (i * stepSize), ((steps - 1) * stepSize + 0.7f), 0);
                                if ((i + 1) % sqrt2 == 0) {
                                        count = 0;
                                        for (int j=0; j<length; j++)
                                                if (Asubl [times - 1, j] <= search2 [k1])
                                                        count++;
                                        StartCoroutine     (waitonesecondtagv2     (position2,
count, 0.6f, Color.white, k2));
                                        k1++;
                                        temprank [i] = count;
                                } else
                                        StartCoroutine   (waitonesecondtagv2   (position2,   0,
0.6f, Color.white, k2));

                                k2++;
                        }
                        k2++;

                        if (Blength == 25) {
                                for (int j=0; j<sqrint-1; j++) {
                                        if ((j + 1) % sqrt2 != 0) {
                                                count = 0;
                                                int posnum = j + (sqrint * (times - 1));
                                                if (j == 2) {
                                                        for      (int      k=temprank[1];
k<length; k++)
                                                                if (Asubl [times - 1,
k] <= Btable [posnum])
                                                                        count++;
                                                } else {
                                                        for (int k=0; k<length; k++)
                                                                if (Asubl [times - 1,
k] <= Btable [posnum])
                                                                        count++;
                                                }
                                                StartCoroutine (waittransfphase2 (posnum,
count, 2.0f));
                                                temprank [j] = count;
                                        }
                                }
                        }


                        StartCoroutine (destroyallv2 (4.0f));


                        Vector3 position4 = new Vector3 ((sqrint * stepSize) + (maxj * 1.1f) - 15.0f
+ (sqrint * stepSize), ((steps - 1) * stepSize + 0.7f), 0);
                        StartCoroutine (waitonesecondtagv3 (position4, "K", 4.5f, Color.white));


                        for (int k=0; k<sqrint-1; k++) {
                                int posnum = k + (sqrint * (times - 1));
                                Vector3 position = new Vector3 ((sqrint * stepSize) + (maxj *
1.1f) - 14.0f + (k * temp) + (sqrint * stepSize), (steps * stepSize + 0.7f), 0);
                                StartCoroutine (waitonesecondtagv3 (position, Btable [posnum],
4.5f, Color.white));
```

```
                                                Vector3 position6 = new Vector3 ((sqrint * stepSize) + (maxj *
1.1f) - 14.0f + (k * temp) + (sqrint * stepSize), ((steps - 1) * stepSize + 0.7f), 0);
                                                StartCoroutine (waitonesecondtagv3 (position6, k + 1, 4.5f,
Color.white));
                                        }

                                        for (int k=1; k<sqrint-1; k++) {
                                                if (k % sqrt2 == 0) {
                                                        temprank [k] = temprank [k] + temprank [(k + 1) /
sqrt2];
                                                        StartCoroutine (waittransfphase2 (k + (times - 1) *
sqrint, temprank [k], 4.5f));
                                                }
                                        }
                                        for (int k=0; k<sqrint-1; k++) {
                                                if (temprank [k] != 0)
                                                        rank [k + (times - 1) * sqrint] = temprank [k];
                                        }


                                }
                        }
                        if (clicktimes == 2) {
                                StartCoroutine (destroyallv2 (0.0f));
                                StartCoroutine (destroyall (0.0f));
                                steps = 1;
                                Vector3 position7 = new Vector3 (-17.0f, ((steps - 1) * stepSize + 0.7f), 0);
                                StartCoroutine (waitonesecondtag (position7, "K", 0.6f, Color.white));


                                for (int k=0; k<Blength; k++) {
                                        Vector3 position = new Vector3 (-16.0f + k * (1.0f), (steps * stepSize + 0.7f),
0);
                                        StartCoroutine (waitonesecondtagv3 (position, Btable [k], 0.6f,
Color.white));
                                        Vector3 position6 = new Vector3 (-16.0f + (k * 1.0f), ((steps - 1) * stepSize
+ 0.7f), 0);
                                        StartCoroutine (waitonesecondtagv3 (position6, k + 1, 0.6f, Color.white));
                                        StartCoroutine (waittransf (rankobj [k], rank [k], 0.6f));
                                }

                                for (int k=1; k<=Blength; k++) {
                                        if (k % sqrint != 0) {
                                                rank [k - 1] = rank [k - 1] + r [k / sqrint];
                                        }
                                        StartCoroutine (waittransf (rankobj [k - 1], rank [k - 1], 1.5f));
                                }


                                int posinum = 0;
                                for (int k=0; k<Blength; k++) {
                                        if (rank [k] == 0) {
                                                Vector3 position = new Vector3 (-17.0f + posinum * (1.0f),
((steps + 2) * stepSize + 0.7f), 0);
                                                StartCoroutine (waitonesecondtagv3 (position, Btable [k], 2.0f,
Color.white));
                                                posinum++;
                                        }
                                }
                                for (int k=0; k<Alength; k++) {
                                        Vector3 position = new Vector3 (-17.0f + posinum * (1.0f), ((steps + 2) *
stepSize + 0.7f), 0);
                                        posinum++;
                                        StartCoroutine (waitonesecondtagv3 (position, Atable [k], 2.0f,
Color.white));
                                        for (int j=0; j<Blength; j++) {
                                                if (rank [j] == k + 1) {
                                                        Vector3 position6 = new Vector3 (-17.0f + (posinum
* 1.0f), ((steps + 2) * stepSize + 0.7f), 0);
                                                        StartCoroutine (waitonesecondtagv3 (position6,
Btable [j], 2.0f, Color.white));
                                                        posinum++;
                                                }
                                        }
```

```
                                        }

                                }
                                clicktimes++;
                        }
                }


}
```

A.4 Αλγόριθμος Παράλληλου Merging

```
using UnityEngine;
using System.Collections;
using System;
using UnityEngine.UI;

public class Merge : MonoBehaviour {

        public InputField numOfProcessorsText;  // reference on input field
        public InputField numOfProcessorsOpt;  // reference on input field
        public GameObject processorPrefab;                      // reference on game object



        private int steps = 0;                                          // step of the algorithm
        private float stepSize = 1.0f;                          // for the position of object
        int num=0;                                                                                      //num given by the
user
        public int []table;                                                                     //temp table for the tree
        public float [] vectors;                                        //temp table for positions of vectors
        public GameObject[] tableobject;                //table with reference on each gameobject of the tree
        public Color[] colortable;                                      //table with colors

        float col1=1.20f;
        float col2=-0.10f;
        float col3=-0.13f;

        int colorcount=0;
        int log=1;
        int check = 2;
        int tempcount=2;
        public void Reset() {                   //reset the algorithm
                Application.LoadLevel("Merge Sort");

        }

        public void Update() {
                if (Input.GetKeyDown(KeyCode.Escape)) {
                        Application.LoadLevel ("MainScene");
                }
        }

        IEnumerator waittransf(GameObject ob,int val,float se){//insert value to a gameobject after a delay
                yield return new WaitForSeconds(se);
                ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + val;
        }

        IEnumerator waitonesecond(Vector3 position,int value,float se,Color newc){ //insert color and value to a processor after a
delay
                yield return new WaitForSeconds(se);
                GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
                go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
                yield return new WaitForSeconds(0.6f);

                go.GetComponent <MeshRenderer> ().material.color = newc;
        }

        IEnumerator waitcolor(Vector3 position,int value,Color newc){  //insert color and value to a processor after a delay
```

```
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);

		go.GetComponent <MeshRenderer> ().material.color = newc;
}

IEnumerator waitcolor(GameObject ob,int value,Color newc){//color a gameobject and put an int on it
		ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);

		ob.GetComponent <MeshRenderer> ().material.color = newc;
}


public void Initproc(){//initialize processors
		if ((numOfProcessorsText.text.Length != 0)&(num==0) ){
				num = Int32.Parse (numOfProcessorsText.text);
			var l = Math.Log (num, 2);
			l = Math.Ceiling (l);


			for (int i=1; i<l; i++) { //find log and processors for patching
				check = check * 2;
				log++;

			}

			colortable = new Color[check];

			table = new int[check];
			vectors = new float[check];
			tableobject = new GameObject[check];



			int correct = Math.Abs (check - num); //for patching

			float temp = 0;
			int cont = 0;
			for (int i=0; i<num; i++) {
				if (i % 2 == 0)//every 2 leafs leave some space
					temp = 1.2f;
				else
					temp = 1.1f;

				Vector3 position = new Vector3 ((i * stepSize) - 10.5f + temp, (steps * stepSize), 0);
				vectors [i] = (i * stepSize - 10 + temp);

				if (i % 2 == 0) {//new color to each processor
					if (col1 < 0)
						col1 = 1;
					col1 = col1 - 0.20f;
					if (col2 > 1)
						col2 = 0.30f;
					col2 = col2 + 0.18f;
					if (col3 > 1)
						col3 = 0.47f;
					col3 = col3 + 0.18f;
					Color newColor = new Color (col1, col2, col3, 1);
					colortable [colorcount] = newColor;
					colorcount++;

				}




				int random = UnityEngine.Random.Range (-30, 101);
				table [i] = random;
				GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as
GameObject;
```

```
                        go.GetComponent <MeshRenderer> ().material.color = colortable [colorcount - 1];
                        go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table [i];
                        cont = i + 1;



                }


                while (correct != 0) {

                        if (correct % 2 == 0) {
                                if (col1 < 0)
                                        col1 = 1;
                                col1 = col1 - 0.20f;
                                if (col2 > 1)
                                        col2 = 0.30f;
                                col2 = col2 + 0.18f;
                                if (col3 > 1)
                                        col3 = 0.47f;
                                col3 = col3 + 0.18f;
                                Color newColor = new Color (col1, col2, col3, 1);
                                colortable [colorcount] = newColor;
                                colorcount++;
                        }

                        Vector3 position = new Vector3 ((cont * stepSize) - 10.6f + temp, (steps * stepSize), 0);
                        vectors [cont] = (cont * stepSize - 10 + temp);
                        GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as
GameObject;
                        table [cont] = 0;
                        go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + table [cont];
                        go.GetComponent <MeshRenderer> ().material.color = colortable [colorcount - 1];
                        cont++;
                        correct--;
                }

                num = num / 2;
                check = check / 2;
                steps=steps+2;

                for (int i=0; i<colortable.Length/2; i++) {
                        Vector3 pos = new Vector3 ((i * stepSize) - 10.5f + 1.1f, -2, 0);
                        GameObject ob = Instantiate (processorPrefab, pos, Quaternion.identity) as
GameObject;


                        ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + i;
                        StartCoroutine (waitcolor (ob, i, colortable [i]));
                }

                Vector3 pos1 = new Vector3 (-10.4f, -2, 0);
                GameObject ob1 = Instantiate (processorPrefab, pos1, Quaternion.identity) as GameObject;


                ob1.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "Proc";
        }

}

public void ByStep(){//parallel sum step by step
        float se = 0.8f;

        if ((check != 0)&(num!=0)) {
                colorcount = 0;
                for (int i=0; i<check; i++) {
                        int [] list = new int[tempcount];
                        for (int j=0; j<tempcount; j++) {
                                list [j] = table [i * tempcount + j];
                        }
                        if (i % 2 == 0)
                                colorcount++;

                        for (int j=0; j<tempcount; j++) {
```

```
                                        int min = list [j];
                                        int minpos = j;
                                        for (int k=j+1; k<tempcount; k++)
                                        if (list [k] < min) {
                                                min = list [k];
                                                minpos = k;
                                        }
                                        list [minpos] = list [j];
                                        list [j] = min;
                                }
                                for (int j=0; j<tempcount; j++) {
                                        vectors [i] = (vectors [2 * i]);

                                        Vector3 position = new Vector3 ((vectors [i] + j * 0.8f), (steps * stepSize),
0);


                                        StartCoroutine (waitonesecond (position, list [j], se, colortable [colorcount -
1]));
                                }

                        }
                        tempcount = tempcount * 2;
                        se = se + 1.4f;
                        check = check / 2;
                        steps++;
                }

        }

        public void InitializeProcessors2 () {//parallel sum full version
                float se = 0.8f;

                while ((check != 0)&(num!=0)) {
                        colorcount = 0;
                        for (int i=0; i<check; i++) {
                                int [] list = new int[tempcount];
                                for (int j=0; j<tempcount; j++) {
                                        list [j] = table [i * tempcount + j];
                                }
                                if (i % 2 == 0)
                                        colorcount++;

                                for (int j=0; j<tempcount; j++) {
                                        int min = list [j];
                                        int minpos = j;
                                        for (int k=j+1; k<tempcount; k++)
                                                if (list [k] < min) {
                                                        min = list [k];
                                                        minpos = k;
                                                }
                                        list [minpos] = list [j];
                                        list [j] = min;
                                }
                                for (int j=0; j<tempcount; j++) {
                                        vectors [i] = (vectors [2 * i]);

                                        Vector3 position = new Vector3 ((vectors [i] + j * 0.8f), (steps * stepSize),
0);


                                        StartCoroutine (waitonesecond (position, list [j], se, colortable [colorcount -
1]));
                                }

                        }
                        tempcount = tempcount * 2;
                        se = se + 1.4f;
                        check = check / 2;
                        steps++;
                }

        }
}
```

A.5 Αλγόριθμος W

```
using UnityEngine;
using System.Collections;
using System;
using UnityEngine.UI;

public class WAlgorithm : MonoBehaviour {

		public InputField numOfProcessorsText;
		public GameObject processorPrefab;


		public Toggle randomtog;
		private int steps = 0;
		private int w3steps = 0;
		private float stepSize = 1.0f;

		int num;
		public int []table; //temp table of values
		public int []numtable; //temp table of values
		public float [] vectors; //temp table of vectors
		public int[] done; // final table of done processors
		public int[] count; //table for w1
		public int[]tempdone; //temp table of done
		public GameObject[] tableobject;//table of alives
		public GameObject[] treeobjects;//table of w1
		public GameObject[] pnums;//gameobject with pnum of each processor
		public GameObject[] taskobjects;//gamebjects at w3
		public GameObject[] w2tree;//gameobjects of w2
		public int[] task;//table with 0 and 1 for task
		public int[,] children;//children for each processor for 21
		public int[,] w3children;//children for each processor for w3
		public int[] pnum;//pnums table
		public float [] w3vectors;//temp vectors for w3
		public int[] k ;
		public int wtime=0;
		public int w2time=0;
		public int w1time=0;
		public int firsttime = 0;//if its the firsttime you call the algorithm
		int shownum=0;
		public int[,] auxv2;//table for w2
		int[] currentv2;

		int can=0;
		int alivecount=0;
		int log=1;
		int check = 2;
		int counter2=0;//for version 2
		int procleft=0;//for w1
		int w3procleft=0;
		public int w2size=0;
		int possibility=10;
		int taskcounter=0;
		int phase=3;
		public void Reset() {
			Application.LoadLevel("Algorithm W");

		}

		public void Update() {
			if (Input.GetKeyDown(KeyCode.Escape)) {
				Application.LoadLevel ("MainScene");
			}
		}

		public void Nextphase(){
```

```
	if (phase == 1)
		W1ver2 ();
	else if (phase == 2)
		W2v2 ();
	else if (phase == 3)
		W3();
	else if (phase == 4)
		W4 ();
}

public void Initprocver2(){//initialize processors
	if (numOfProcessorsText.text.Length != 0) {
		num = Int32.Parse (numOfProcessorsText.text);
		var l = Math.Log (num, 2);
		l = Math.Ceiling (l);


		for (int i=1; i<l; i++) {
			check = check * 2;
			log++;
		}



		pnums = new GameObject[check];//gameobjects of pnums
		children = new int[num, num];//children for w1
		table = new int[check];//value of gameobjects
		numtable = new int[check * 2];
		vectors = new float[check];//position of every gameobject
		tempdone = new int[check];//temporary table for done
		done = new int[check * 2];//how many tasks with believe that are already done
		tableobject = new GameObject[check];//table with alives processors gameobjects
		taskobjects = new GameObject[check * 2];//table with tasks gameobjects
		w2tree = new GameObject[check];//tree of gameobjects for w2
		task = new int[check];//table with tasks
		treeobjects = new GameObject[check * 2];//gameobjects of the tree
		pnum = new int[check];//table with personal number for each processor
		count = new int[check * 2];//values for w1 tree
		k = new int[check];//assign processors at w2
		w2size = check;

		float move = 0.0f;
		if (num > 8)
			move = -6.0f;


		auxv2 = new int[check, check * 2];
		currentv2 = new int[check];
		for (int i=0; i<check; i++)
			currentv2 [i] = 1;

		w3children = new int[num, num];//children for W3

		for (int i=0; i<check; i++)//initialize children all with -1 except the processor you are at
			for (int j=0; j<check; j++)
				if (i == j)
					children [i, j] = i;
				else
					children [i, j] = -1;


		for (int i=0; i<check; i++)//initialize children all with -1 except the processor you are at
			for (int j=0; j<check; j++)
				if (i == j)
					w3children [i, j] = i;
				else
					w3children [i, j] = -1;



		counter2 = check;//to set the correct numbers to the tree
```

```
                        for (int i=0; i<num; i++) {


                                int random = UnityEngine.Random.Range (0, 101);//random kill processors
                                if (randomtog.isOn) {
                                        random = 100;
                                }
                                if (random >= 25) {
                                        table [i] = 1;
                                        alivecount++;
                                } else
                                        table [i] = 0;



                                Vector3 positionobj = new Vector3 ((i * stepSize) - 10.5f + move + 1.1f, -2, 0);
                                GameObject obj = Instantiate (processorPrefab, positionobj, Quaternion.identity) as
GameObject;


                                if (table [i] == 1) {
                                        obj.transform.FindChild ("Value").GetComponent<TextMesh> ().text =
"A";
                                } else {

                                        obj.transform.FindChild ("Value").GetComponent<TextMesh> ().text =
"D";
                                        obj.GetComponent <MeshRenderer> ().material.color = Color.red;
                                }


                                tableobject [i] = obj;




                        }
                        Vector3 positionobj2 = new Vector3 (-10.5f + move, -2, 0);
                        GameObject pnobj1 = Instantiate (processorPrefab, positionobj2, Quaternion.identity) as
GameObject;
                        pnobj1.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "A/D";


                }
        }

        public void AlgorithmW(){//W whole algorithm
                StartCoroutine(waitw(0.8f));
        }

        public void W1ver2 () {//w1 complete
                if (phase == 1) {
                        phase++;
                        can=1;
                        float move = 0.0f;
                        if (num > 8)
                                move = -6.0f;


                        if (done [1] != check ) {//if there are any alive processors or done is not finished
                                if (wtime == 0) {//if its the first time calling the whole algorithm
                                        if (w1time == 0) {//the first time calling W1 to init pnums and gameobjects
                                                float temp = 0;
                                                for (int i=0; i<check; i++) {
                                                        if (i % 2 == 0)
                                                                temp = 1.2f;
                                                        else
                                                                temp = 1.1f;



                                                        Vector3 position = new Vector3 ((i * stepSize) +
temp, 0, 0);
```

```
                                                vectors [i] = (i * stepSize) + temp;


                                                GameObject go = Instantiate (processorPrefab,
position, Quaternion.identity) as GameObject;



                                                int random = UnityEngine.Random.Range (0, 101);
                                                if ((randomtog.isOn)|alivecount==1){
                                                        random = 100;
                                                }
                                                if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                        if (random >= 5) {
                                                                table [i] = 1;
                                                                pnum [i] = 1;
                                                        }
                                                } else {
                                                        table [i] = 0;
                                                        tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "D";
                                                        tableobject [i].GetComponent
<MeshRenderer> ().material.color = Color.red;
                                                        alivecount--;
                                                }


                                                go.transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + table [i];
                                                treeobjects [counter2] = go;
                                                count [counter2] = table [i];
                                                counter2++;



                                                Vector3 positionobj1 = new Vector3 ((i * stepSize) -
10.5f + move + 1.1f, -3, 0);
                                                GameObject pnobj = Instantiate (processorPrefab,
positionobj1, Quaternion.identity) as GameObject;


                                                pnobj.transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + pnum [i];
                                                pnums [i] = pnobj;
                                        }

                                        Vector3 positionobj2 = new Vector3 (-10.5f + move, -3, 0);
                                        GameObject pnobj1 = Instantiate (processorPrefab, positionobj2,
Quaternion.identity) as GameObject;
                                        pnobj1.transform.FindChild
("Value").GetComponent<TextMesh> ().text = "Pnum";


                                        steps++;
                                        procleft = check / 2;
                                        counter2 = check / 2;
                                }
                                w1time++;

                                float se = 0.8f;

                                while (procleft!=0) {

                                        int living;
                                        int rliv;
                                        for (int i=0; i<procleft; i++) {
                                                living = 0;
                                                rliv = 0;

                                                if (steps == 5) {
                                                        vectors [i] = (vectors [2 * i]) + steps +
2.5f;
                                                } else {
                                                        vectors [i] = (vectors [2 * i]) + steps -
0.9f;
                                                }
```

```
Vector3 position = new Vector3 ((vectors [i]), (steps
* stepSize), 0);

int random = UnityEngine.Random.Range (0, 101);
if ((randomtog.isOn)|alivecount==1){
        random = 100;
}

if (tableobject [2 * i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
        if (random <= possibility) {
                tableobject [2 *
i].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
                tableobject [2 *
i].GetComponent <MeshRenderer> ().material.color = Color.red;
                alivecount--;
                StartCoroutine (waittransf
(tableobject [2 * i], tableobject [2 * i].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
        }
}


int random2 = UnityEngine.Random.Range (0, 101);
if ((randomtog.isOn)|alivecount==1){
        random2 = 100;
}
if (tableobject [2 * i + 1].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
        if (random2 <= possibility) {
                tableobject [2 * i +
1].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
                tableobject [2 * i +
1].GetComponent <MeshRenderer> ().material.color = Color.red;
                alivecount--;
                StartCoroutine (waittransf
(tableobject [2 * i + 1], tableobject [2 * i + 1].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
        }
}



for (int j=0; j<check; j++) {//check if there is a living
child on the left
        if (children [2 * i, j] != -1)
        if (tableobject [children [2 * i,
j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A")
                living = 1;
}

for (int j=0; j<check; j++) {//check if there is a living
child on the right
        if (children [2 * i + 1, j] != -1)
        if (tableobject [children [2 * i + 1,
j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A") {
                rliv = 1;
                pnum [children [2 * i + 1, j]] =
table [2 * i] + pnum [children [2 * i + 1, j]];
                StartCoroutine (waittransf
(pnums [children [2 * i + 1, j]], pnum [children [2 * i + 1, j]], se));
        }
}


int a;
int b;
if ((living == 1) || (rliv == 1)) {
        a = table [2 * i];
        b = table [2 * i + 1];
} else {
        a = 0;
        b = 0;
}
table [i] = a + b;
```

```
                                        if (i != 0)
                                                for (int j=0; j<check; j++)
                                                        children [i, j] = -1;


                                        for (int j=0; j<check; j++) {//processors below i are
its children
                                                if (children [2 * i, j] != -1)
                                                        children [i, j] = children [2 * i,
j];
                                                if (children [2 * i + 1, j] != -1)
                                                        children [i, j] = children [2 * i
+ 1, j];
                                        }

                                        StartCoroutine (waitonesecondv2 (position, i, table
[i], se, Color.white, counter2));
                                        count [counter2] = table [i];
                                        counter2++;

                                }

                                se = se + 1.4f;
                                procleft = procleft / 2;
                                counter2 = procleft;
                                steps++;
                        }
                } else {//if its not the first time calling whole algorithm(after at least one loop)
                        if (w1time == 0) {//first time calling w1 on this loop
                                for (int i=0; i<check; i++)
                                        for (int j=0; j<check; j++)
                                                if (i == j)
                                                        children [i, j] = i;
                                                else
                                                        children [i, j] = -1;

                                counter2 = check;
                                for (int i=0; i<check; i++) {


                                        int random = UnityEngine.Random.Range (0, 101);
                                        if ((randomtog.isOn)|alivecount==1){
                                                random = 100;
                                        }
                                        if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                if (random >= 5) {
                                                        table [i] = 1;
                                                        pnum [i] = 1;
                                                }
                                        } else {
                                                table [i] = 0;
                                                pnum [i] = 0;
                                                tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "D";

                                                tableobject [i].GetComponent
<MeshRenderer> ().material.color = Color.red;

                                                alivecount--;
                                        }


                                        treeobjects [counter2].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + table [i];
                                        count [counter2] = table [i];
                                        counter2++;

                                        pnums [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + pnum [i];
                                }

                                procleft = check / 2;
                                counter2 = check / 2;
                                w1time++;
                        }
```

```
float se = 0.8f;
while (procleft!=0) {

	int living;
	int rliv;
	for (int i=0; i<procleft; i++) {
		living = 0;
		rliv = 0;

		int random = UnityEngine.Random.Range (0, 101);
		if ((randomtog.isOn)|alivecount==1){
			random = 100;
		}

		if (tableobject [2 * i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
			if (random <= possibility - 8) {
				tableobject [2 *
i].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
				tableobject [2 *
i].GetComponent <MeshRenderer> ().material.color = Color.red;
				alivecount--;
				StartCoroutine (waittransf
(tableobject [2 * i], tableobject [2 * i].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
			}
		}


		int random2 = UnityEngine.Random.Range (0, 101);
		if ((randomtog.isOn)|alivecount==1){
			random2 = 100;
		}
		if (tableobject [2 * i + 1].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
			if (random2 <= possibility - 8) {
				tableobject [2 * i +
1].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
				tableobject [2 * i +
1].GetComponent <MeshRenderer> ().material.color = Color.red;
				alivecount--;
				StartCoroutine (waittransf
(tableobject [2 * i + 1], tableobject [2 * i + 1].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
			}
		}



		for (int j=0; j<check; j++) {
			if (children [2 * i, j] != -1)
			if (tableobject [children [2 * i,
j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A")
				living = 1;
		}

		for (int j=0; j<check; j++) {
			if (children [2 * i + 1, j] != -1)
			if (tableobject [children [2 * i + 1,
j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A") {
				rliv = 1;
				pnum [children [2 * i + 1, j]] =
table [2 * i] + pnum [children [2 * i + 1, j]];
				StartCoroutine (waittransf
(pnums [children [2 * i + 1, j]], pnum [children [2 * i + 1, j]], se));
			}
		}


		int a;
		int b;
		if ((living == 1) || (rliv == 1)) {
			a = table [2 * i];
			b = table [2 * i + 1];
		} else {
			a = 0;
```

```
                                                                b = 0;
                                                        }
                                                        table [i] = a + b;

                                                        if (i != 0)
                                                                for (int j=0; j<check; j++)
                                                                        children [i, j] = -1;


                                                        for (int j=0; j<check; j++) {
                                                                if (children [2 * i, j] != -1)
                                                                        children [i, j] = children [2 * i,
j];

                                                                if (children [2 * i + 1, j] != -1)
                                                                        children [i, j] = children [2 * i
+ 1, j];
                                                        }


                                                        StartCoroutine (waittransf (treeobjects [counter2],
table [i], se));
                                                        count [counter2] = table [i];
                                                        counter2++;

                                                }

                                                se = se + 1.4f;
                                                procleft = procleft / 2;
                                                counter2 = procleft;
                                        }
                                }
                        }

                        if (w2time != 0) {
                                for (int i=0; i<check; i++) {
                                        auxv2 [i, 1] = done [1];
                                        w2tree [i].transform.FindChild ("Value").GetComponent<TextMesh>
().text = "0";
                                        currentv2 [i] = 1;
                                }
                                w2size = check;
                        }
                }
        }

        public void W1ByStep(){//w1 step by step
                if (phase == 1) {
                        can=1;
                        float move = 0.0f;
                        if (num > 8)
                                move = -6.0f;

                        if (done [1] != check) {
                                if (wtime == 0) {
                                        if (w1time == 0) {
                                                float temp = 0;
                                                for (int i=0; i<num; i++) {
                                                        if (i % 2 == 0)
                                                                temp = 1.2f;
                                                        else
                                                                temp = 1.1f;

                                                        Vector3 position = new Vector3 ((i * stepSize) +
temp, 0, 0);

                                                        vectors [i] = (i * stepSize) + temp;


                                                        GameObject go = Instantiate (processorPrefab,
position, Quaternion.identity) as GameObject;


                                                        int random = UnityEngine.Random.Range (0, 101);
                                                        if ((randomtog.isOn)|alivecount==1){
                                                                random = 100;
```

```
                                                        }
                                                        if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                                if (random >= 5) {
                                                                        table [i] = 1;
                                                                        pnum [i] = 1;
                                                                }
                                                        } else {
                                                                table [i] = 0;
                                                                tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "D";
                                                                tableobject [i].GetComponent
<MeshRenderer> ().material.color = Color.red;
                                                                alivecount--;
                                                        }

                                                        go.transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + table [i];
                                                        treeobjects [counter2] = go;
                                                        count [counter2] = table [i];
                                                        counter2++;

                                                        Vector3 positionobj1 = new Vector3 ((i * stepSize) -
10.5f + 1.1f + move, -3, 0);
                                                        GameObject pnobj = Instantiate (processorPrefab,
positionobj1, Quaternion.identity) as GameObject;
                                                        pnobj.transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + pnum [i];
                                                        pnums [i] = pnobj;
                                                }

                                                Vector3 positionobj2 = new Vector3 (-10.5f + move, -3, 0);
                                                GameObject pnobj1 = Instantiate (processorPrefab, positionobj2,
Quaternion.identity) as GameObject;
                                                pnobj1.transform.FindChild
("Value").GetComponent<TextMesh> ().text = "Pnum";

                                                steps++;
                                                procleft = check / 2;
                                                counter2 = check / 2;
                                                w1time++;
                                        } else {
                                                float se = 0.8f;
                                                if (procleft != 0) {

                                                        int living;
                                                        int rliv;
                                                        for (int i=0; i<procleft; i++) {
                                                                living = 0;
                                                                rliv = 0;

                                                                if (steps == 5) {
                                                                        vectors [i] = (vectors [2 * i]) +
steps + 2.5f;
                                                                } else {
                                                                        vectors [i] = (vectors [2 * i]) +
steps - 0.9f;
                                                                }

                                                                Vector3 position = new Vector3 ((vectors
[i]), (steps * stepSize), 0);

                                                                int random = UnityEngine.Random.Range
(0, 101);

                                                                if ((randomtog.isOn)|alivecount==1){
                                                                        random = 100;
                                                                }
                                                                if (tableobject [2 * i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                                        if (random <= possibility) {
```

```
                                                                                tableobject [2 *
i].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
                                                                                tableobject [2 *
i].GetComponent <MeshRenderer> ().material.color = Color.red;
                                                                                alivecount--;
                                                                                StartCoroutine
(waittransf (tableobject [2 * i], tableobject [2 * i].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
                                                                        }
                                                                }


                                                                int random2 =
UnityEngine.Random.Range (0, 101);
                                                                if ((randomtog.isOn)|alivecount==1){
                                                                        random2 = 100;
                                                                }
                                                                if (tableobject [2 * i +
1].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A") {
                                                                        if (random2 <= possibility) {
                                                                                tableobject [2 * i +
1].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
                                                                                tableobject [2 * i +
1].GetComponent <MeshRenderer> ().material.color = Color.red;
                                                                                alivecount--;
                                                                                StartCoroutine
(waittransf (tableobject [2 * i + 1], tableobject [2 * i + 1].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
                                                                        }
                                                                }




                                                                for (int j=0; j<check; j++) {
                                                                        if (children [2 * i, j] != -1)
                                                                        if (tableobject [children [2 * i,
j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A")
                                                                                living = 1;
                                                                }


                                                                for (int j=0; j<check; j++) {
                                                                        if (children [2 * i + 1, j] != -1)
                                                                        if (tableobject [children [2 * i +
1, j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A") {
                                                                                rliv = 1;
                                                                                pnum [children [2 *
i + 1, j]] = table [2 * i] + pnum [children [2 * i + 1, j]];
                                                                                StartCoroutine
(waittransf (pnums [children [2 * i + 1, j]], pnum [children [2 * i + 1, j]], se));
                                                                        }
                                                                }


                                                                int a;
                                                                int b;
                                                                if ((living == 1) || (rliv == 1)) {
                                                                        a = table [2 * i];
                                                                        b = table [2 * i + 1];
                                                                } else {
                                                                        a = 0;
                                                                        b = 0;
                                                                }
                                                                table [i] = a + b;

                                                                if (i != 0)
                                                                        for (int j=0; j<check; j++)
                                                                                children [i, j] = -1;


                                                                for (int j=0; j<check; j++) {
                                                                        if (children [2 * i, j] != -1)
                                                                                children [i, j] =
children [2 * i, j];
                                                                                if (children [2 * i + 1, j] != -1)
```

```
                                                                                children [i, j] =
children [2 * i + 1, j];
                                                                        }

                                                                        StartCoroutine (waitonesecondv2
(position, i, table [i], se, Color.white, counter2));
                                                                        count [counter2] = table [i];
                                                                        counter2++;
                                                                }
                                                        }
                                                        steps++;
                                                        procleft = procleft / 2;
                                                        counter2 = procleft;
                                                }
                                        } else {
                                                float se = 0.8f;
                                                if (w1time == 0) {
                                                        for (int i=0; i<check; i++)
                                                                for (int j=0; j<check; j++)
                                                                        if (i == j)
                                                                                children [i, j] = i;
                                                                        else
                                                                                children [i, j] = -1;

                                                        counter2 = check;
                                                        for (int i=0; i<check; i++) {


                                                                int random = UnityEngine.Random.Range (0, 101);
                                                                if ((randomtog.isOn)|alivecount==1){
                                                                        random = 100;
                                                                }
                                                                if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                                        if (random >= 5) {
                                                                                table [i] = 1;
                                                                                pnum [i] = 1;
                                                                        }
                                                                } else {
                                                                        table [i] = 0;
                                                                        pnum [i] = 0;
                                                                        tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "D";
                                                                        tableobject [i].GetComponent
<MeshRenderer> ().material.color = Color.red;
                                                                        alivecount--;
                                                                }


                                                                treeobjects [counter2].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + table [i];
                                                                count [counter2] = table [i];
                                                                counter2++;

                                                                pnums [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + pnum [i];
                                                        }

                                                        procleft = check / 2;
                                                        counter2 = check / 2;
                                                        w1time++;
                                                } else {
                                                        if (procleft != 0) {

                                                                int living;
                                                                int rliv;
                                                                for (int i=0; i<procleft; i++) {
                                                                        living = 0;
                                                                        rliv = 0;

                                                                        int random = UnityEngine.Random.Range
(0, 101);

                                                                        if ((randomtog.isOn)|alivecount==1){
```

```
                random = 100;
        }
        if (tableobject [2 * i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                if (random <= possibility - 8) {
                        tableobject [2 *
i].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
                        tableobject [2 *
i].GetComponent <MeshRenderer> ().material.color = Color.red;
                        alivecount--;
                        StartCoroutine
(waittransf (tableobject [2 * i], tableobject [2 * i].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
                }
        }


        int random2 =
UnityEngine.Random.Range (0, 101);
        if ((randomtog.isOn)|alivecount==1){
                random2 = 100;
        }
        if (tableobject [2 * i +
1].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A") {
                if (random2 <= possibility - 8)
{
                        tableobject [2 * i +
1].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
                        tableobject [2 * i +
1].GetComponent <MeshRenderer> ().material.color = Color.red;
                        alivecount--;
                        StartCoroutine
(waittransf (tableobject [2 * i + 1], tableobject [2 * i + 1].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
                }
        }



        for (int j=0; j<check; j++) {
                if (children [2 * i, j] != -1)
                if (tableobject [children [2 * i,
j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A")
                        living = 1;
        }

        for (int j=0; j<check; j++) {
                if (children [2 * i + 1, j] != -1)
                                                        //
        if(alive[children[2*i+1,j]]=="A"){
                if (tableobject [children [2 * i +
1, j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A") {
                        rliv = 1;
                        pnum [children [2 *
i + 1, j]] = table [2 * i] + pnum [children [2 * i + 1, j]];
                        StartCoroutine
(waittransf (pnums [children [2 * i + 1, j]], pnum [children [2 * i + 1, j]], se));
                }
        }


        int a;
        int b;
        if ((living == 1) || (rliv == 1)) {
                a = table [2 * i];
                b = table [2 * i + 1];
        } else {
                a = 0;
                b = 0;
        }
        table [i] = a + b;

        if (i != 0)
                for (int j=0; j<check; j++)
                        children [i, j] = -1;
```

```
                                                                        for (int j=0; j<check; j++) {
                                                                                if (children [2 * i, j] != -1)
                                                                                        children [i, j] =
children [2 * i, j];

                                                                                if (children [2 * i + 1, j] != -1)
                                                                                        children [i, j] =
children [2 * i + 1, j];
                                                                        }

                                                                        StartCoroutine (waittransf (treeobjects
[counter2], table [i], se));
                                                                        count [counter2] = table [i];
                                                                        counter2++;

                                                                }

                                                                se = se + 1.4f;
                                                                procleft = procleft / 2;
                                                                counter2 = procleft;
                                                        }
                                                }
                                        }
                                }
                                if (w2time != 0) {
                                        for (int i=0; i<check; i++) {
                                                auxv2 [i, 1] = done [1];
                                                w2tree [i].transform.FindChild ("Value").GetComponent<TextMesh>
().text = "0";
                                                currentv2 [i] = 1;
                                        }
                                        w2size = check;
                                }
                        }
                }
                public void W2v2(){//w2 complete
                        if (phase == 2) {
                                phase++;
                                float move = 0.0f;
                                if (num > 8) //for >8 processors
                                        move = -6.0f;
                                if (done [1] != check) {//if the algorithm isn't complete and there are still alive processors
                                        float se = 0.8f;
                                        int child1;
                                        int child2;


                                        if (wtime == 0) {//if its the first time create the objects
                                                for (int i=0; i<check; i++) {
                                                        auxv2 [i, 1] = done [1];

                                                        Vector3 position = new Vector3 ((i * stepSize) - 10.5f + move +
1.1f, -4, 0);
                                                        GameObject w2obj = Instantiate (processorPrefab, position,
Quaternion.identity) as GameObject;

                                                        w2obj.transform.FindChild
("Value").GetComponent<TextMesh> ().text = "0";
                                                        w2tree [i] = w2obj;
                                                }
                                                Vector3 positionobj2 = new Vector3 (-10.5f + move, -4, 0);
                                                GameObject pnobj1 = Instantiate (processorPrefab, positionobj2,
Quaternion.identity) as GameObject;
                                                pnobj1.transform.FindChild ("Value").GetComponent<TextMesh> ().text =
"W2";
                                        }
                                        wtime++;

                                        while (w2size!=1) {
                                                for (int i=0; i<check; i++) {
                                                        if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {

                                                                child1 = currentv2 [i] * 2;
```

```
                                                                        child2 = child1 + 1;

                                                                        if (done [child1] + done [child2] == 0)
                                                                                auxv2 [i, child1] = 0;
                                                                        else
                                                                                auxv2 [i, child1] = Mathf.RoundToInt
((float)(auxv2 [i, currentv2 [i]] * done [child1]) / (done [child1] + done [child2]));
                                                                        auxv2 [i, child2] = auxv2 [i, currentv2 [i]] - auxv2 [i,
child1];

                                                                        taskobjects [child1].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + auxv2 [i, child1];
                                                                        taskobjects [child2].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + auxv2 [i, child2];

                                                                        if (w2size == auxv2 [i, currentv2 [i]])
                                                                                count [child1] = 0;
                                                                        else
                                                                                count [child1] = Mathf.RoundToInt
((float)(count [currentv2 [i]] * (Mathf.RoundToInt ((float)(w2size / 2)) - auxv2 [i, child1])) / (w2size - auxv2 [i, currentv2 [i]]));

                                                                        count [child2] = count [currentv2 [i]] - count [child1];

                                                                        if (pnum [i] <= count [child1])
                                                                                currentv2 [i] = child1;
                                                                        else {
                                                                                currentv2 [i] = child2;
                                                                                pnum [i] = pnum [i] - count [child1];
                                                                                StartCoroutine (waittransf (pnums [i],
pnum [i], se));

                                                                        }
                                                                        StartCoroutine (waittransf (w2tree [i], currentv2 [i],
se));
                                                                }

                                                        }
                                                        w2size = w2size / 2;
                                                        se = se + 1.4f;

                                                }
                                                for (int i=0; i<check; i++)
                                                        if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                                k [i] = currentv2 [i] - (check - 1);
                                                                StartCoroutine (waittransf (w2tree [i], currentv2 [i], se));
                                                        }
                                                w2time++;
                                                w1time = 0;
                                        }
                                }
                        }

                        public void shownumb(){
                                if (can != 0) {
                                        if (shownum == 0) {
                                                for (int i=1; i<check*2; i++)
                                                        treeobjects [i].transform.FindChild ("Value").GetComponent<TextMesh>
().text = "" + i;
                                                shownum = 1;
                                        } else {
                                                for (int i=1; i<check*2; i++)
                                                        treeobjects [i].transform.FindChild ("Value").GetComponent<TextMesh>
().text = "" + count [i];
                                                shownum = 0;
                                        }
                                }
                        }
                        public void W2v2bystep(){//w2 step by step
                                if (phase == 2) {
                                        phase++;
                                        float move = 0.0f;
                                        if (num > 8)
                                                move = -6.0f;
```

```
if (done [1] != check ) {
	float se = 0.8f;
	int child1;
	int child2;


	if (wtime == 0) {
		for (int i=0; i<check; i++) {
			auxv2 [i, 1] = done [1];

			Vector3 position = new Vector3 ((i * stepSize) - 10.5f + 1.1f +
move, -4, 0);
			GameObject w2obj = Instantiate (processorPrefab, position,
Quaternion.identity) as GameObject;

			w2obj.transform.FindChild
("Value").GetComponent<TextMesh> ().text = "0";
			w2tree [i] = w2obj;
		}
		Vector3 positionobj2 = new Vector3 (-10.5f + move, -4, 0);
		GameObject pnobj1 = Instantiate (processorPrefab, positionobj2,
Quaternion.identity) as GameObject;
		pnobj1.transform.FindChild ("Value").GetComponent<TextMesh> ().text =
"W2";
	}
	wtime++;


	if (w2size != 1) {
		for (int i=0; i<check; i++) {
			if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {

				child1 = currentv2 [i] * 2;
				child2 = child1 + 1;

				if (done [child1] + done [child2] == 0)
					auxv2 [i, child1] = 0;
				else
					auxv2 [i, child1] = Mathf.RoundToInt
((float)(auxv2 [i, currentv2 [i]] * done [child1]) / (done [child1] + done [child2]));
				auxv2 [i, child2] = auxv2 [i, currentv2 [i]] - auxv2 [i,
child1];

				taskobjects [child1].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + auxv2 [i, child1];
				taskobjects [child2].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "" + auxv2 [i, child2];

				count [child1] = Mathf.RoundToInt ((float)(count
[currentv2 [i]] * (Mathf.RoundToInt ((float)(w2size / 2)) - auxv2 [i, child1])) / (w2size - auxv2 [i, currentv2 [i]]));

				count [child2] = count [currentv2 [i]] - count [child1];

				if (pnum [i] <= count [child1])
					currentv2 [i] = child1;
				else {
					currentv2 [i] = child2;
					pnum [i] = pnum [i] - count [child1];
					StartCoroutine (waittransf (pnums [i],
pnum [i], se));
				}
				StartCoroutine (waittransf (w2tree [i], currentv2 [i],
se));
			}

		}
		w2size = w2size / 2;
		se = se + 1.4f;

	}
	if (w2size == 1) {
		for (int i=0; i<check; i++)
```

```
                                                        if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                                k [i] = currentv2 [i] - (check - 1);
                                                                w2time = 0;
                                                                StartCoroutine (waittransf (w2tree [i], currentv2 [i],
se));
                                                        }
                                        }
                                        w1time = 0;


                                }
                        }
                }

                public void W3(){//w3 complete (change value to 1 of "where" for each alive processor)
                        if ((phase == 3) && (num!=0)) {
                                phase++;
                                float move = 0.0f;
                                if (num > 8)
                                        move = -6.0f;

                                if (done [1] != check ) {
                                        w3vectors = new float[check];//position of every gameobject
                                        taskcounter = check;
                                        float temp;
                                        int random;
                                        if (wtime == 0) {
                                                for (int i=0; i<check; i++) {//create task objects
                                                        random = UnityEngine.Random.Range (0, 101);
                                                        if ((randomtog.isOn)|alivecount==1){
                                                                random = 100;
                                                        }
                                                        if (random <= possibility) {
                                                                tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text = "D";
                                                                tableobject [i].GetComponent <MeshRenderer>
().material.color = Color.red;
                                                                alivecount--;
                                                                StartCoroutine (waittransf (tableobject [i], tableobject
[i].transform.FindChild ("Value").GetComponent<TextMesh> ().text, 0));
                                                        }

                                                        if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A")
                                                                task [i] = 1;
                                                        else
                                                                task [i] = 0;



                                                        if (i % 2 == 0)
                                                                temp = 1.2f;
                                                        else
                                                                temp = 1.1f;

                                                        Vector3 positionobj = new Vector3 ((i * stepSize) - 10.5f + temp
+ move, (steps * stepSize), 0);
                                                        w3vectors [i] = (i * stepSize - 10 + temp + move);

                                                        GameObject obj = Instantiate (processorPrefab, positionobj,
Quaternion.identity) as GameObject;

                                                        obj.transform.FindChild ("Value").GetComponent<TextMesh>
().text = "" + task [i];
                                                        taskobjects [taskcounter] = obj;
                                                        done [taskcounter] = task [i];
                                                        taskcounter++;
                                                        tempdone [i] = task [i];
                                                }
                                        } else {
                                                for (int i=0; i<check; i++) {//change task objects of alive to 1
                                                        if (k [i] != 0) {
```

```
                                                                        if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                                                task [k [i] - 1] = 1;
                                                                                done [k [i] + check - 1] = 1;
                                                                                taskobjects [k [i] + check -
1].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + 1;
                                                                        }
                                                                }
                                                        }
                                                        for (int i=0; i<check; i++) {
                                                                tempdone [i] = task [i];
                                                                if (k [i] != 0)
                                                                        k [i] = k [i] + check - 1;
                                                        }

                                                }
                                                w3procleft = check / 2;
                                                w3steps++;
                                                taskcounter = check / 2;
                                        }
                                }
                        }

                        public void W4(){//w4 complete
                                if (phase == 4) {
                                        phase = 1;
                                        if (done [1] != check) {
                                                float se = 0.8f;

                                                if (wtime == 0) {//first time create the game objects and do the sum
                                                        while (w3procleft!=0) {

                                                                int living;
                                                                int rliv;
                                                                for (int i=0; i<w3procleft; i++) {
                                                                        living = 0;
                                                                        rliv = 0;

                                                                        if (w3steps == 5) {
                                                                                w3vectors [i] = (w3vectors [2 * i]) +
w3steps + 2.5f;
                                                                        } else {
                                                                                w3vectors [i] = (w3vectors [2 * i]) +
w3steps - 0.9f;
                                                                        }

                                                                        Vector3 position = new Vector3 ((w3vectors [i]),
(w3steps * stepSize), 0);


                                                                        int random = UnityEngine.Random.Range (0, 101);
                                                                        if ((randomtog.isOn)|alivecount==1){
                                                                                random = 100;
                                                                        }
                                                                        if (tableobject [2 * i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                                                if (random <= possibility) {
                                                                                        tableobject [2 *
i].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
                                                                                        alivecount--;
                                                                                        StartCoroutine (waittransf
(tableobject [2 * i], tableobject [2 * i].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
                                                                                }
                                                                        }



                                                                        int random2 = UnityEngine.Random.Range (0, 101);
                                                                        if ((randomtog.isOn)|alivecount==1){
                                                                                random2 = 100;
                                                                        }
                                                                        if (tableobject [2 * i + 1].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                                                if (random2 <= possibility) {
```

```
                                                tableobject [2 * i +
1].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
                                                alivecount--;
                                                StartCoroutine (waittransf
(tableobject [2 * i + 1], tableobject [2 * i + 1].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
                                        }
                                }


                                for (int j=0; j<check; j++) {
                                        if (w3children [2 * i, j] != -1)
                                        if (tableobject [w3children [2 * i,
j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A")
                                                living = 1;
                                }

                                for (int j=0; j<check; j++) {
                                        if (w3children [2 * i + 1, j] != -1)
                                        if (tableobject [w3children [2 * i + 1,
j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A")
                                                rliv = 1;
                                }


                                int a;
                                int b;
                                if ((living == 1) || (rliv == 1)) {
                                        a = tempdone [2 * i];
                                        b = tempdone [2 * i + 1];
                                } else {
                                        a = 0;
                                        b = 0;
                                }
                                tempdone [i] = a + b;

                                if (i != 0)
                                        for (int j=0; j<check; j++)
                                                w3children [i, j] = -1;


                                for (int j=0; j<check; j++) {
                                        if (w3children [2 * i, j] != -1)
                                                w3children [i, j] = w3children
[2 * i, j];

                                        if (w3children [2 * i + 1, j] != -1)
                                                w3children [i, j] = w3children
[2 * i + 1, j];
                                }


                                StartCoroutine (waittasks (position, i, tempdone [i],
se, Color.white, taskcounter));
                                done [taskcounter] = tempdone [i];
                                taskcounter++;
                        }

                        se = se + 1.4f;
                        w3procleft = w3procleft / 2;
                        taskcounter = w3procleft;
                        w3steps++;
                }
        } else {

                while (w3procleft!=0) {

                        for (int i=0; i<check; i++) {
                                if (k [i] != 0) {

                                        int random = UnityEngine.Random.Range
(0, 101);

                                        if ((randomtog.isOn)|alivecount==1){
                                                random = 100;
                                        }
                                        if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
```

```
										if (random <= possibility - 8) {
											tableobject
[i].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
											alivecount--;
											StartCoroutine
(waittransf (tableobject [i], tableobject [i].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
										}
									}


									int lchild = -1;
									if (k [i] % 2 == 0)
										lchild = 1;



									int a = 0;
									int b = 0;


									if (lchild == 1) {
										a = done [k [i]];
										b = done [k [i] + 1];
									} else {
										a = done [k [i]];
										b = done [k [i] - 1];
									}

									k [i] = k [i] / 2;
									if (a != 0 && b != 0)
										done [k [i]] = a + b;


									StartCoroutine (waittransf (taskobjects [k
[i]], done [k [i]], se));
								}
							}
							se = se + 1.4f;
							w3procleft = w3procleft / 2;

						}

					}
				}
				w1time = 0;
			}
		}


		public void W4bystep(){//w4 by step
			if (phase == 4) {
				phase = 1;
				if (done [1] != check ) {
					float se = 0.8f;

					if (wtime == 0) {

						if (w3procleft != 0) {

							int living;
							int rliv;
							for (int i=0; i<w3procleft; i++) {
								living = 0;
								rliv = 0;

								if (w3steps == 5) {
									w3vectors [i] = (w3vectors [2 * i]) +
w3steps + 2.5f;
								} else {
									w3vectors [i] = (w3vectors [2 * i]) +
w3steps - 0.9f;
								}
```

```
Vector3 position = new Vector3 ((w3vectors [i]),
(w3steps * stepSize), 0);


int random = UnityEngine.Random.Range (0, 101);
if ((randomtog.isOn)|alivecount==1){
	random = 100;
}
if (tableobject [2 * i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
	if (random <= possibility) {
		tableobject [2 *
i].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
		alivecount--;
		StartCoroutine (waittransf
(tableobject [2 * i], tableobject [2 * i].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
	}
}

int random2 = UnityEngine.Random.Range (0, 101);
if ((randomtog.isOn)|alivecount==1){
	random2 = 100;
}

if (tableobject [2 * i + 1].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
	if (random2 <= possibility) {
		tableobject [2 * i +
1].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
		alivecount--;
		StartCoroutine (waittransf
(tableobject [2 * i + 1], tableobject [2 * i + 1].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
	}
}


for (int j=0; j<check; j++) {
	if (w3children [2 * i, j] != -1)
	if (tableobject [w3children [2 * i,
j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A")
		living = 1;
}


for (int j=0; j<check; j++) {
	if (w3children [2 * i + 1, j] != -1)
	if (tableobject [w3children [2 * i + 1,
j]].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A")
		rliv = 1;
}


int a;
int b;
if ((living == 1) || (rliv == 1)) {
	a = tempdone [2 * i];
	b = tempdone [2 * i + 1];
} else {
	a = 0;
	b = 0;
}
tempdone [i] = a + b;

if (i != 0)
	for (int j=0; j<check; j++)
		w3children [i, j] = -1;


for (int j=0; j<check; j++) {
	if (w3children [2 * i, j] != -1)
		w3children [i, j] = w3children
[2 * i, j];

	if (w3children [2 * i + 1, j] != -1)
```

```
                                                                w3children [i, j] = w3children
[2 * i + 1, j];
                                                        }

                                                        StartCoroutine (waittasks (position, i, tempdone [i],
se, Color.white, taskcounter));
                                                        done [taskcounter] = tempdone [i];
                                                        taskcounter++;
                                                }

                                                se = se + 1.4f;
                                                w3procleft = w3procleft / 2;
                                                taskcounter = w3procleft;
                                                w3steps++;
                                        }
                                } else {
                                        if (w3procleft != 0) {
                                                for (int i=0; i<check; i++) {
                                                        if (k [i] != 0) {

                                                                int random = UnityEngine.Random.Range
(0, 101);
                                                                if ((randomtog.isOn)|alivecount==1){
                                                                        random = 100;
                                                                }
                                                                if (tableobject [i].transform.FindChild
("Value").GetComponent<TextMesh> ().text == "A") {
                                                                        if (random <= possibility - 8) {
                                                                                tableobject
[i].transform.FindChild ("Value").GetComponent<TextMesh> ().text = "D";
                                                                                alivecount--;
                                                                                StartCoroutine
(waittransf (tableobject [i], tableobject [i].transform.FindChild ("Value").GetComponent<TextMesh> ().text, se));
                                                                        }
                                                                }


                                                                int lchild = -1;
                                                                if (k [i] % 2 == 0)
                                                                        lchild = 1;



                                                                int a = 0;
                                                                int b = 0;


                                                                if (lchild == 1) {
                                                                        a = done [k [i]];
                                                                        b = done [k [i] + 1];
                                                                } else {
                                                                        a = done [k [i]];
                                                                        b = done [k [i] - 1];
                                                                }

                                                                k [i] = k [i] / 2;
                                                                if (a != 0 && b != 0)
                                                                        done [k [i]] = a + b;


                                                                StartCoroutine (waittransf (taskobjects [k
[i]], done [k [i]], se));
                                                        }
                                                }
                                                se = se + 1.4f;
                                                w3procleft = w3procleft / 2;

                                        }

                                }
                        }
                        w1time = 0;
                }
        }
```

```
	IEnumerator waittransf(GameObject ob,int val,float se){//change of an int value in a processor
			yield return new WaitForSeconds(se);
		ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + val;
	}

	IEnumerator waittransf(GameObject ob,string val,float se){//change of a string value in a processor
		yield return new WaitForSeconds(se);
		ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + val;
	}


	IEnumerator waitonesecondv2(Vector3 position,int i,int value,float se,Color newc,int count){//create new gameobject
and add it to treeobjects
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);
		go.GetComponent <MeshRenderer> ().material.color = newc;
		treeobjects [count] = go;
	}

	IEnumerator waittasks(Vector3 position,int i,int value,float se,Color newc,int count){//create new gameobject and add
it to taskobjects
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);
		go.GetComponent <MeshRenderer> ().material.color = newc;
		taskobjects [count] = go;
	}

	IEnumerator waitw(float se){//create new gameobject and add it to taskobjects
		yield return new WaitForSeconds(se);
		W3();
		se = se + 1.5f;
		yield return new WaitForSeconds(se);
		W4 ();
		se = se + 1.5f;
		yield return new WaitForSeconds(se);
		while (done[1]!=check) {
			W1ver2 ();
			se = se + 0.8f;
			yield return new WaitForSeconds (se);
			W2v2 ();
			se = se + 0.8f;
			yield return new WaitForSeconds(se);
			W3();
			se = se + 0.8f;
			yield return new WaitForSeconds(se);
			W4 ();
			se = se + 0.8f;
			yield return new WaitForSeconds(se);
		}
	}


}
```

A.6 Αλγόριθμος X

```
using UnityEngine;
using System.Collections;
using System;
using UnityEngine.UI;

public class AlgorithmX : MonoBehaviour {

	public InputField numOfProcessorsText;  // reference on input field
	public GameObject processorPrefab;			// reference on game object
```

```
	public Toggle randomtog;

	private int steps = 0;                                  // step of the algorithm
	private float stepSize = 1.0f;                  // for the position of object

	int num;                                                                        //num given by the
user
	public int []table;                                     //temp table for the tree
	public string[] alive;                                  //temp table for the alive proc
	public float [] vectors;                        //temp table for positions of vectors
	public int[] done;                                      //which processors of the table have value
0|1
	public GameObject[] tableobject;        //table with reference on each alive/dead gameobject
	public GameObject[] treeobjects;        //table with reference on each gameobject of the tree
	public Color[] colortable;                      //table with colors
	public int[] randomtable;                       //table to find faster processor each time
	public int alivecount=1;                        //remaining alive processors
	public string [] binstrings;            //temp table for binary
	public int[,] bincut;                           //table with digit on every position for each processor
	public GameObject fast;                                 //gameobject to show the faster processor
each time
	public int [] lvl;                                      //lvl of processor on tree
	public int [] where;                            //where is the processor
	float col1=1.20f;
	float col2=-0.10f;
	float col3=-0.13f;

	int possibility=5;                                      //possibility to kill a processor
	int	random;
	int faster;
	public int[] fasterv2;
	int times;
	int log=1;
	int check = 2;
	int counter2=0;


	public void Reset() {                                   //reset the algorithm
		Application.LoadLevel("Algorithm X");

	}

	public void Update() {
		if (Input.GetKeyDown(KeyCode.Escape)) {
			Application.LoadLevel ("MainScene");
		}
	}

	IEnumerator waitonesecondv2(Vector3 position,int i,int value,float se,Color newc,int counter2){ //insert color and
value to a processor after a delay
		yield return new WaitForSeconds(se);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;
		go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
		yield return new WaitForSeconds(0.6f);
		go.GetComponent <MeshRenderer> ().material.color = newc;
		treeobjects [counter2] = go;
	}


	public void Initprocver2(){                                             //init processors
		num = Int32.Parse (numOfProcessorsText.text);
		var l = Math.Log(num, 2);
		l = Math.Ceiling(l);



		for (int i=1; i<l; i++) {                               //calculate log and number of processors
			check = check * 2;
			log++;

		}

		table = new int[check];//value of gameobjects
```

```
vectors = new float[check];//position of every gameobject
done = new int[check*2];
tableobject = new GameObject[check];//table with alives processors gameobjects
alive = new string[check];//values of alive
treeobjects = new GameObject[check*2];//gameobjects of the tree
randomtable = new int[check];
colortable = new Color[check];
where = new int[check];
binstrings = new string[check];
bincut = new int[check, log];
lvl = new int [check];
fasterv2 = new int[check];

for (int i=0; i<check; i++) { //convert to bits and initialize lvl to 0
		var result = Convert.ToString(i, 2);
		binstrings[i]=result;
		lvl[i]=0;
}



for (int i=0; i<check; i++) {//cut bits of binary
		for (int j=0; j<binstrings[i].Length; j++) {
				string piece = binstrings [i].Substring (j, 1);
				bincut[i,(log-binstrings[i].Length)+j]=Int32.Parse(piece);
		}
}



float temp = 0;
counter2 = check;//for processors
for (int i=0; i<num; i++) {
		if (i % 2 == 0)
				temp = 1.2f;
		else
				temp = 1.1f;

		Vector3 position = new Vector3 ((i * stepSize) - 15.5f + temp, (steps * stepSize), 0);
		vectors[i]=(i*stepSize-15+temp);
		GameObject go = Instantiate (processorPrefab, position, Quaternion.identity) as GameObject;



		if (col1<0)
				col1=1;
		col1=col1-0.20f;
		if (col2>1)
				col2=0.30f;
		col2=col2+0.28f;
		if (col3>1)
				col3=0.47f;
		col3=col3+0.18f;
		Color newColor=new Color(col1,col2,col3,1);
		colortable[i]=newColor;




		int random=UnityEngine.Random.Range (0, 101);
		if (randomtog.isOn) {
				random = 100;
		}
		if (random>=33){ //random Dead/Alive
				table[i]=1;
				alive[i]="A";
		}
		else{
				table[i]=0;
				alive[i]="D";
		}
```

```
                go.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + 0;
                treeobjects[counter2]=go;
                Vector3 positionobj= new Vector3((i*stepSize)-15.5f+1.1f,-2,0);
                GameObject obj = Instantiate (processorPrefab, positionobj, Quaternion.identity) as GameObject;


                obj.transform.FindChild("Value").GetComponent<TextMesh>().text=""+alive[i];
                tableobject[i]=obj;
                where[i]=counter2;
                counter2++;
                StartCoroutine(waitcolor(obj,alive[i],colortable[i]));
        }

        Vector3 pos= new Vector3(-13.5f,-3,0);
        GameObject ob = Instantiate (processorPrefab, pos, Quaternion.identity) as GameObject;
        ob.transform.FindChild("Value").GetComponent<TextMesh>().text="";
        fast = ob;

        int proc=check;
        proc = proc / 2;
        steps++;
        counter2=check/2;
        int tempcount = counter2;

        while (proc!=0) {//init starting tree

                for (int i=0; i<proc; i++) {

                        if (steps==5){
                                vectors [i] = (vectors [2 * i]) + steps + 2.5f;
                        }
                        else{
                                vectors[i]=(vectors[2*i])+steps-0.9f;
                        }

                        Vector3 position = new Vector3 ((vectors[i]), (steps * stepSize), 0);



                        StartCoroutine(waitonesecondv2(position,i,0,0,Color.white,counter2));
                        counter2++;



                        //              }

                }
                tempcount=tempcount/2;
                counter2=tempcount;
                proc=proc/2;
                steps++;
        }



}



public void XbyStepv2(){
        alivecount = 0;
        fasterv2 = callrandom2 ();

        int count = 0;
        for (int i=0; i<fasterv2.Length; i++) {
                if (fasterv2 [i] != -1)
                        count++;
        }


        for (int i=0; i<count; i++) {
```

```
faster = fasterv2 [i];

int execute = 0; //1 action every time
int child = 0;   //int to show if you are going to left or right child

if (count==1)
StartCoroutine (waitcolor (fast, faster, colortable [faster]));//color the fast
object
else
StartCoroutine (waitcolor (fast, count+" pr", Color.white));//color the fast
object

while (done[where[faster]]==1) {//while above lvls are done go to next lvl
        where [faster] = where [faster] / 2;
        lvl [faster]++;
}

if (done [where [faster]] == 0 && where [faster] >= check) {//if you are at
leaf make the value 0 and move 1 lvl
        done [where [faster]] = 1;
        StartCoroutine (waitcolor (treeobjects [where [faster]], 1,
colortable [faster]));
        where [faster] = where [faster] / 2;
        lvl [faster]++;
        execute = 1;
}


if (done [where [faster]] == 0 && where [faster] < check && execute == 0)
{//if you are not on a leaf
        if (done [where [faster] * 2] == 1 && done [where [faster] * 2 +
1] == 1) {//if both children are 1 make your value 1 too
                done [where [faster]] = 1;
                StartCoroutine (waitcolor (treeobjects [where
[faster]], 1, colortable [faster]));
                where [faster] = where [faster] / 2;
                lvl [faster]++;
                execute = 1;
        } else if (execute == 0) {
                if (where [faster] * 2 <= check * 2) {//you cant go
futher than leaves
                        while (done[where[faster]*2]==0 ||
done[where[faster]*2+1]==0) {//while one of the children is 0
                                if (done [where [faster] * 2 +
1] == 0 && done [where [faster] * 2] == 1)//if the right children is 0 make child=2
                                        child = 2;
                                else if (done [where [faster] *
2] == 0 && done [where [faster] * 2 + 1] == 1)//if left children is 1 make child=1
                                        child = 1;
                                else {//if both children are 0

check with your binary and lvl if its 0 go left if its 1 go right
                                        if (binstrings
[faster].Length < lvl [faster]) {
                                                child =
1;
                                        } else {
                                                int loglv
= log - lvl [faster];
                                                if
(bincut [faster, loglv] == 0)
                                                        child = 1;
                                                else
                                                        child = 2;
                                        }
                                }

                                lvl [faster]--;
                                if (child == 1)//go left
                                        where [faster] =
where [faster] * 2;
                                if (child == 2)//go right
```

```
                                                                                where [faster] =
where [faster] * 2 + 1;

                                                                                    if (where [faster] * 2 >= check
* 2 - 1)
                                                                                        break;
                                                                                }
                                                                        }
                                                                        done [where [faster]] = 1;
                                                                        StartCoroutine (waitcolor (treeobjects [where
[faster]], 1, colortable [faster]));
                                                                        where [faster] = where [faster] / 2;
                                                                        lvl [faster]++;

                                                        }
                                                }
                                        }
                                }




                                public void XAlgorithm(){
                                        float se = 0.2f;
                                        StartCoroutine (waitx (se));

                                }



                                IEnumerator waittransf(GameObject ob,int val,float se){//change a value of a gameobject with int
                                                yield return new WaitForSeconds(se);
                                        ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + val;
                                }



                                IEnumerator waittransf(GameObject ob,string val,float se){//change a value of a gameobject with string
                                        yield return new WaitForSeconds(se);
                                        ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + val;
                                }


                                IEnumerator waitcolor(GameObject ob,int value,Color newc){//color a gameobject and put an int on it
                                        ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
                                        yield return new WaitForSeconds(0.6f);

                                        ob.GetComponent <MeshRenderer> ().material.color = newc;
                                }

                                IEnumerator waitcolor(GameObject ob,string value,Color newc){//color a gameobject and put a string on it
                                        ob.transform.FindChild ("Value").GetComponent<TextMesh> ().text = "" + value;
                                        yield return new WaitForSeconds(0.6f);

                                        ob.GetComponent <MeshRenderer> ().material.color = newc;
                                }


                                IEnumerator waitx(float se){//create new gameobject and add it to taskobjects
                                        while (done[1]!=1) {
                                                alivecount = 0;
                                                fasterv2 = callrandom2 ();

                                                int count = 0;

                                                for (int i=0; i<fasterv2.Length; i++) {
                                                        if (fasterv2 [i] != -1)
                                                                count++;
                                                }
```

```
for (int i=0; i<count; i++) {
	faster = fasterv2 [i];
	int execute = 0;
	int child = 0;

	if (count==1)
		StartCoroutine (waitcolor (fast, faster, colortable [faster]));//color the fast
object
	else
		StartCoroutine (waitcolor (fast, count+" pr", Color.white));//color the fast
object

	while (done[where[faster]]==1){
		where [faster] = where [faster] / 2;
		lvl[faster]++;
	}

	if (done [where [faster]] == 0 && where [faster] >= check) {
		done [where [faster]] = 1;
		StartCoroutine (waitcolor (treeobjects [where [faster]], 1, colortable
[faster]));
		where [faster] = where [faster] / 2;
		lvl[faster]++;
		execute = 1;
	}


	if (done [where [faster]] == 0 && where [faster] < check && execute == 0) {
		if (done [where [faster] * 2] == 1 && done [where [faster] * 2 + 1] == 1) {
			done [where [faster]] = 1;
			StartCoroutine (waitcolor (treeobjects [where [faster]], 1,
colortable [faster]));
			where [faster] = where [faster] / 2;
			lvl[faster]++;
			execute = 1;
		}

		else if (execute == 0) {
			if (where [faster] * 2 <= check * 2) {
				while (done[where[faster]*2]==0 ||
done[where[faster]*2+1]==0) {
					if (done [where [faster] * 2 + 1] == 0 &&
done [where [faster] * 2] == 1)
						child = 2;
					else if (done [where [faster] * 2] == 0 &&
done [where [faster] * 2 +1] == 1)
						child = 1;
					else {
						if
(binstrings[faster].Length<lvl[faster]){
							child=1;
						}
						else{
							int loglv=log-
lvl[faster];
							if
(bincut[faster,loglv]==0)
								child=1;
							else
								child=2;
						}
					}

					lvl[faster]--;
					if (child == 1)
						where [faster] = where [faster]
* 2;
					if (child == 2)
						where [faster] = where [faster]
* 2 + 1;

					if (where [faster] * 2 >= check * 2 - 1)
						break;
				}
```

```
                                                }
                                                done [where [faster]] = 1;
                                                StartCoroutine (waitcolor (treeobjects [where [faster]], 1,
colortable [faster]));
                                                where [faster] = where [faster] / 2;
                                                lvl[faster]++;
                                        }

                                }

                                yield return new WaitForSeconds (se);

                                if (check > 8)
                                        se = se + 0.1f;
                                else if (check > 16 && times < 25)
                                        se = se + 0.01f;
                                else if (check < 8)
                                        se = se + 0.3f;

                                times++;
                                }
                        }
                }



        int[] callrandom2(){//find faster processor on every call
                int speed;
                int[] fasters = new int[check];

                for (int i=0; i<check; i++)
                        fasters [i] = -1;
                for (int i=0; i<check; i++) {//count alive processors
                        if (tableobject [i].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A") {
                                alivecount++;
                        }
                }

                for (int i=0; i<check; i++) {//use random numbers for "speed" of each processor if its alive
                        speed = UnityEngine.Random.Range (0, 10);
                        if (tableobject [i].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A") {
                                randomtable[i]=speed;
                        }
                }

                for (int i=0; i<check; i++) {//random kill processors
                        if (tableobject [i].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A") {
                                int random = UnityEngine.Random.Range (0, 101);
                                if (randomtog.isOn) {
                                        random = 100;
                                }
                                if ((random <= possibility)&(alivecount!=1)) {
                                        tableobject [i].transform.FindChild ("Value").GetComponent<TextMesh>
().text = "D";
                                        StartCoroutine (waittransf (tableobject [i], tableobject
[i].transform.FindChild ("Value").GetComponent<TextMesh> ().text, 0.8f));
                                        randomtable[i]=-1;
                                        alivecount--;
                                }
                        }
                        else
                                randomtable[i]=-1;
                }



                int max = randomtable[0];

                for (int i=1; i<check; i++) {//find max speed
                        if (randomtable[i]>max){
                                max=randomtable[i];
                        }
                }
```

```
                for (int i=0; i<check; i++) {//count alive processors
                        if (tableobject [i].transform.FindChild ("Value").GetComponent<TextMesh> ().text == "A") {
                                alivecount++;
                        }
                }

                int k = 0;
                for (int i=0; i<check; i++) {
                        if (randomtable [i] == max) {
                                fasters [k] = i;
                                k++;
                        }
                }

                return fasters;

        }


}
```

A.7 Main Scene

```
using UnityEngine;
using System.Collections;

public class MainScript : MonoBehaviour {

        public void Update() {
                if (Input.GetKeyDown(KeyCode.Escape)) {
                        Application.Quit();
                }
        }

         public void Sum(){
                Application.LoadLevel ("ParallelSum");
        }

        public void Merge(){
                Application.LoadLevel ("Merge Sort");
        }

        public void Rank(){
                Application.LoadLevel ("Rank Scene");
        }
        public void Search(){
                Application.LoadLevel ("Search");
        }
        public void X(){
                Application.LoadLevel ("Algorithm X");
        }
        public void W(){
                Application.LoadLevel ("Algorithm W");
        }
}
```