

Ατομική Διπλωματική Εργασία

**ΑΠΕΙΚΟΝΙΣΗ ΒΑΣΙΚΩΝ ΑΛΓΟΡΙΘΜΩΝ ΜΕ ΤΗ ΧΡΗΣΗ ΤΗΣ ΜΗΧΑΝΗΣ
ΗΛΕΚΤΡΟΝΙΚΩΝ ΠΑΙΧΝΙΔΙΩΝ UNITY3D**

Νικόλας Τζιωρτζής

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2019

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Απεικόνιση Βασικών Αλγορίθμων με τη Χρήση της Μηχανής Ηλεκτρονικών Παιχνιδιών Unity3D

Νικόλας Τζιωρτζής

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 2019

Ευχαριστίες

Κατά την διάρκεια της ανάπτυξης και υλοποίησης της πιο κάτω πτυχιακής συνέβαλαν αρκετά πρόσωπα. Πρώτο και κύριο θα ήθελα να ευχαριστήσω τον επιβλέπον μου καθηγητή Δρ. Γεωργίου Χρύση για την ευκαιρία που μου έχει δώσει να συνεργαστώ μαζί του σε αυτή την πρωτότυπη εργασία αλλά και για την συμβολή του και την μεγάλη στήριξη και βοήθεια που προσέφερε στην υλοποίηση της οπτικής αναπαράστασης των αλγορίθμων. Επίσης ευχαριστώ όλους τους φοιτητές του τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου που προσέφεραν τις αντιδράσεις τους και τα σχόλιά τους για βελτίωση και καλύτερη ανάπτυξη των πιο κάτω αλγορίθμων και επιπλέον στου κοντινότερούς μου φίλους-συμφοιτητές που έχουν δώσει τις πιο ειλικρινείς αναδράσεις. Επιπρόσθετα, θα ήθελα να δώσω ένα μεγάλο ευχαριστώ στον Δρ. Παναγιώτη Χαραλάμπους και φοιτητές του Εργαστηρίου Γραφικών του Τμήματος Πληροφορικής για την συμβολή τους στην απόκτηση γνώσεων περί γραφικά και μαθηματικών.

Επιπρόσθετά θα ήθελα να ευχαριστήσω το διαδίκτυο και συγκεκριμένα, το Stack Overflow, τους διδάσκοντες των tutorials της σελίδας του Unity, τον Brackeys και άλλες πολλές διαδικτυακές πηγές τις οποίες θα αναφέρω στην βιβλιογραφία, που με βοήθησαν στην αντιμετώπιση και επίλυση διαφόρων προβλημάτων κατά την υλοποίηση καθώς και στην υλοποίηση ορισμένων δυνατοτήτων του προγράμματός μας.

Τέλος ένα μεγάλο ευχαριστώ στους φίλους και στην οικογένειά μου που με στηρίξαν στη μεγάλη αυτή διαδρομή!

Περίληψη

Σε αυτή την Διπλωματική Εργασία αναλύονται και υλοποιούνται αλγόριθμοι σε οπτική αναπαράσταση με την χρήση του εργαλείου Unity3D. Γενικά οι αλγόριθμοι που θα παρουσιαστούν είναι περισσότερο αλγόριθμοι που είναι δύσκολοι στην εκμάθησή τους. Συνεπώς με μια οπτική τρισδιάστατη αναπαράσταση με χρήση γραφικών σε συνδυασμό με κίνηση θα κάνουν πιο εύκολη την κατανόηση από οποιοδήποτε με υπόβαθρο αλγορίθμων και πληροφορικής. Έτσι θα έχουμε και καλύτερη εκμάθηση κατά την διδασκαλία τους.

Καταρχάς περιγράφεται το εργαλείο που χρησιμοποιήσαμε και βασικές έννοιες που θα πρέπει να γνωρίζουμε για να προχωρήσουμε στις περιγραφές όλων των υλοποιήσεων. Ως επακόλουθο, γινόταν μελέτη εξακρίβωσης λεπτομερειών σε κάθε αλγόριθμο. Μετά από περιγραφή κάθε αρχικού προσχεδίου και της σκέψης μας προτού υλοποιήσουμε κάθε αλγόριθμο, παρουσιάζεται η επεξήγηση και ο τρόπος υλοποίησης τους με το εργαλείο Unity. Αυτό περιλαμβάνει την περιγραφή κάθε 3D αντικειμένου που έχει δημιουργηθεί που συνεισφέρει στην παρουσίαση και εκτέλεση καθώς και κάθε σημαντικής γραμμής κώδικα σε όλα τα χρήσιμα script που έχουν γραφτεί στη C#.

Επιπρόσθετα ο κάθε αλγόριθμος ξεχωριστά έχει υποστεί μια μετατροπή για την λειτουργία και οπτική εμφάνιση προς το χρήστη. Με την περιγραφή αυτής της μετατροπής και των τρόπων εφαρμογής τους με μεθόδους προγραμματισμού και άλλα εργαλεία, καταλήγουμε σε μια τελική οπτική αναπαράσταση με εικόνα και κίνηση που καθιστά πιο ενδιαφέρον την εκμάθησή του σε ένα επιθυμητό χρήστη. Τέλος μετά από τακτές δοκιμές των αλγορίθμων, έχουν παρουσιαστεί προς φοιτητές και γνώστες πληροφορικής για να παρθούν γνώμες για αλλαγές αλλά και για μελλοντικές επεκτάσεις και βελτιώσεις.

Περιεχόμενα

Κεφάλαιο 1 Εισαγωγή.....	1
1.1 Κίνητρο και Στόχος ΑΔΕ	1
1.2 Μεθοδολογία	2
1.3 Οργάνωση Κειμένου	3
Κεφάλαιο 2 Προηγούμενη Γνώση - Υπόβαθρο	5
2.1 Προηγούμενες Γνώσεις	5
2.2 Εργαλείο Unity3D	6
2.2.1 Περιγραφή Unity3D	6
2.2.2 Χρήση Πλατφόρμας Unity3D	7
2.3 Αναγκαία Επιπλέον Μελέτη	9
2.4 Ο Πρώτος Αλγόριθμος – Selection Sort	10
2.4.1 Περιγραφή Αλγορίθμου	10
2.4.2 Υλοποίηση Εμφάνισης Αλγορίθμου	11
2.4.3 Υλοποίηση Λειτουργίας Αλγορίθμου και Διεπαφής	15
2.4.4 Δυσκολίες και Διευκρινήσεις Υλοποίησης	22
Κεφάλαιο 3 Κύρια Διεπαφή Προγράμματος (UI)	23
3.1 Περιγραφή Τρόπου Υλοποίησης	23
3.2 Σχεδιασμός και Υλοποίηση	24
3.3 Προβλήματα και Διευκρινήσεις	27
Κεφάλαιο 4 Αλγόριθμος Ταχύ Μετασχηματισμού Fourier	29
4.1 Εισαγωγή	29
4.2 Περιγραφή Αλγορίθμου FFT	30
4.2.1 Αναδρομικός FFT	30
4.2.2 Μετατροπή Αναδρομικού FFT σε Επαναληπτικού	34
4.3 Λεπτομέρειες Υλοποίησης	35
4.3.1 Σχεδιασμός Πρωτοτύπου	35

4.3.2 Υλοποίηση Διεπαφής	38
4.3.3 Αναπαράσταση και Εκτέλεση Αλγορίθμου	39
4.4 Δυσκολίες/Προβλήματα και Τρόπος επίλυσης	45
Κεφάλαιο 5 Αλγόριθμος Επιλογής	47
5.1 Εισαγωγή	47
5.2 Περιγραφή Αλγορίθμου	48
5.3 Λεπτομέρειες Υλοποίησης	50
5.3.1 Μετατροπή σε Επαναληπτικό και Αρχική Σκέψη	50
5.3.2 Σχεδιασμός Πρωτοτύπου	51
5.3.3 Υλοποίηση Διεπαφής	53
5.3.4 Αναπαράσταση και Εκτέλεση Αλγορίθμου	55
5.4 Δυσκολίες, Τρόπος επίλυσης και Διευκρινήσεις	60
Κεφάλαιο 6 Δυναμικός Προγραμματισμός – Αλγόριθμος Υπολογισμού	
Σειράς Αριθμών Fibonacci	62
6.1 Εισαγωγή	62
6.2 Περιγραφή Αλγορίθμου	63
6.3 Λεπτομέρειες Υλοποίησης	68
6.3.1 Σχεδιασμός Πρωτοτύπου	68
6.3.2 Υλοποίηση 3D Δυαδικού Δένδρου	69
6.3.3 Υλοποίηση Διεπαφής	73
6.3.4 Αναπαράσταση και Εκτέλεση Αλγορίθμου	75
6.4 Δυσκολίες, Τρόπος Επίλυσής τους και Διευκρινήσεις	77
Κεφάλαιο 7 Τυχαιοποιημένος Αλγόριθμος Συρρίκνωσης Karger	80
7.1 Εισαγωγή	80
7.2 Περιγραφή Αλγορίθμου	81
7.3 Λεπτομέρειες Υλοποίησης	83
7.3.1 Σχεδιασμός Πρωτοτύπου	83
7.3.1.1 Δημιουργία 3D Συνεκτικού Πολυγράφου	83

7.3.1.2 Αποθήκευση και Ανάκτηση Γράφων	88
7.3.2 Υλοποίηση Διεπαφής	89
7.3.3 Αναπαράσταση και Εκτέλεση Αλγορίθμου	91
7.4 Δυσκολίες, Προβλήματα και Τρόπος Αντιμετώπισής τους	96
Κεφάλαιο 8 Επίλογος - Συμπεράσματα	98
8.1 Επίλογος	98
8.2 Κυριότερα Προβλήματα και Τρόπος Αντιμετώπισης	100
8.3 Μελλοντική Εργασία και Επεκτάσεις	101
Βιβλιογραφία	104
Παράρτημα Α	A-1
Παράρτημα Β	B-1
Παράρτημα Γ	Γ-1
Παράρτημα Δ	Δ-1
Παράρτημα Ε	Ε-1
Παράρτημα Ζ	Z-1

Κεφάλαιο 1.

Εισαγωγή

1.1 Κίνητρο και Στόχος ΑΔΕ	1
1.2 Μεθοδολογία	2
1.3 Οργάνωση Κειμένου	3

1.1 Κίνητρο και Στόχος ΑΔΕ

Κεντρικός στόχος αυτής της Ατομικής Διπλωματικής Εργασίας είναι η υλοποίηση ενός πλαισίου με τη χρήση του εργαλείου Unity, το οποίο προσφέρει οπτικές γραφικές αναπαραστάσεις αλγορίθμων. Αυτό το πρόγραμμα θα προσφέρει καλύτερη και πιο διασκεδαστική κατανόηση συγκεκριμένων αλγορίθμων οι οποίοι είναι δύσκολοι στην κατανόηση χωρίς την εκτέλεσή τους και την παρακολούθηση ενός παραδείγματός τους με πραγματικά δεδομένα. Απευθύνεται προς τους φοιτητές της Πληροφορικής, αλλά και σε οποιοδήποτε άλλο έχει το αντίστοιχο μαθησιακό υπόβαθρο και κίνητρο να μάθει τους αλγορίθμους αυτούς.

Το πρόγραμμα αυτό δημιουργήθηκε για την στήριξη φοιτητών στο μάθημα ΕΠΛ 336 - ΕΠΛ 236: Αλγόριθμοι και Πολυπλοκότητα [4]. Οι αλγόριθμοι που έχουν επιλεγεί να εφαρμοστούν στην εργασία αυτή είναι συγκεκριμένοι αλγόριθμοι που έχουν δυσκολία στην κατανόηση από τους μαθητευόμενους. Σταθήκαν επίσης ιδανικοί να παρουσιαστούν σε μια γραφική αναπαράσταση με κινούμενα τρισδιάστατα αντικείμενα.

Σε τελικό στάδιο το πρόγραμμά αυτό έχει υλοποιημένους τους εξής αλγορίθμους (οι οποίοι διδάσκονται στο μάθημα αυτό) [4,5]:

- Αλγόριθμος Ταξινόμησης με επιλογή (Selection Sort) (Αυτός ο αλγόριθμος υλοποιήθηκε κυρίως για την εκμάθηση του εργαλείου Unity)

- Αλγόριθμος Ταχύ Μετασχηματισμού Φουριέρ (Fast Fourier Transform - FFT)
- Αλγόριθμος Επιλογής (Selection Algorithm)
- Δυναμικός Προγραμματισμός – Αλγόριθμος Fibonacci
- Τυχαιοποιημένος Αλγόριθμος – Αλγόριθμος Karger – Καθολικής Αποκοπής

Αξίζει να σημειωθεί ότι το μάθημα αυτό το είχα παρακολουθήσει το Εαρινό εξάμηνο του 2017 και συνεπώς κατείχα ένα βασικό μαθησιακό υπόβαθρο για ευκολότερη υλοποίηση των αλγορίθμων αυτών.

1.2 Μεθοδολογία

Για την γραφική υλοποίηση κάθε αλγορίθμου χρησιμοποιήθηκε ο ίδιος τρόπος προγραμματισμού και σχεδιασμού πριν εφαρμοστεί η υλοποίηση στον υπολογιστή. Αρχικά γινόταν περεταίρω μελέτη και λεπτομερής ανάλυση του κάθε αλγορίθμου σε σημείο που θα υπήρχαν οι επαρκές γνώσεις για μια πλήρη γραφική τους υλοποίηση χωρίς κενά. Στην συνέχεια γινόταν ένας αρχικός σχεδιασμός της πρωτότυπης εμφάνισης του υφιστάμενου αλγορίθμου, στον οποίο είμαστε επικεντρωμένοι, πάνω στο χαρτί για εξακρίβωση της καλύτερης αναπαράστασης που θα τείνει να είναι πιο ευνόητη εικονικά. Τέλος εφαρμοζόταν η υλοποίηση του πάνω στον υπολογιστή με τη χρήση της ηλεκτρονικής πλατφόρμας Unity3D [1]. Η πλατφόρμα αυτή προσφέρει ισχυρές επιλογές και πολλαπλά εργαλεία για δημιουργία τόσο 3D όσο και 2D αναπαραστάσεων γραφικών και άλλων εικονικών παραστάσεων κυρίως υλοποίησης βιντεοπαιχνιδιών.

Επιπρόσθετα, οι αλγόριθμοι, οι διεπαφές τους και ο προγραμματισμός της αναπαράστασης τους υλοποιήθηκαν με τη χρήση της C# γλώσσας προγραμματισμού που υποστηρίζει το Unity3D. Προτού εφαρμοστεί η υλοποίηση, προϋπήρχαν εκπαιδεύσεις από εκπαιδευτικό υλικό στο διαδίκτυο για το Unity3D και συγκεκριμένους τρόπους υλοποίησης για κάθε συγκεκριμένο στάδιο στο οποίο βρισκόμαστε καθώς η πλατφόρμα αυτή ήταν πρωτόγνωρη για μας.

Σημαντική, κατά την διάρκεια της υλοποίησης του κάθε αλγορίθμου, ήταν και η συμβολή του επιβλέποντα καθηγητή, φοιτητών και φίλων για προτάσεις για βελτιώσεις και σχόλια

για αλλαγές στην εμφάνιση, ευχρηστία και λειτουργία του προγράμματος. Είχαμε επιλέξει ένα συγκεκριμένο αριθμό ατόμων τα οποία πρόσδιδαν ανάδραση κατά την υλοποίηση (Peer Evaluation). Τα ίδια άτομα έδιναν σχόλια τόσο στην αρχή και στη μέση αλλά όσο και στο τελικό πρόγραμμα.

Να σημειωθεί εδώ ότι για την παρουσίαση των scripts που φαίνονται ως σχήματα στην συνέχεια της εργασίας μας έχει γίνει με screenshots από τον Text Editor – “**Sublime Text**” [22]. Επίσης ο σχεδιασμός των προσχεδίων των διεπαφών μας έχει γίνει με χρήση της ιστοσελίδας “**Draw.io**” [23].

1.3 Οργάνωση Κειμένου

Η υφιστάμενη διπλωματική εργασία αποτελείται από ακόμη επτά κεφάλαια:

Στο Κεφάλαιο 2 συνεχίζουμε με την περιγραφή της μεθοδολογίας περαιτέρω συνοψίζοντας την προηγούμενη γνώση και το μαθησιακό υπόβαθρο το οποίο χρειάστηκε για την υλοποίηση. Επίσης περιγράφεται η υλοποίηση του αλγορίθμου Selection Sort, η οποία είναι βασική καθώς εδώ θα παρουσιάσουμε έννοιες που θα χρησιμοποιηθούν αργότερα στους υπόλοιπους αλγορίθμους.

Στο Κεφάλαιο 3 υπάρχει μια ανασκόπηση της κύριας διεπαφής – UI (User Interface) που υλοποιήθηκε και αποτελεί εργαλείο περιήγησης μεταξύ των αλγορίθμων αλλά και ελέγχου. Να σημειώσουμε ότι σε κάθε επακόλουθο κεφάλαιο θα περιλαμβάνονται και η περιγραφή προβλημάτων, διευκρινήσεων και τρόπων επίλυσης για την κάθε μας υλοποίηση.

Στο Κεφάλαιο 4 ξεκινάμε την περιγραφή υλοποίησης με τον πρώτο μας αλγόριθμο, τον Fast Fourier Transform (Αλγόριθμος Μετασχηματισμού Fourier). Αναπτύσσοντας αρχικά την περιγραφή της υλοποίησης της διεπαφής χρήστη, προχωράμε στα προσχέδια και στην υλοποίησή του με το εργαλείο Unity.

Ο αλγόριθμος Επιλογής παρουσιάζεται στο Κεφάλαιο 5. Η ανάλυση του αλγορίθμου, κάθε αρχική σκέψη και υλοποίηση διάφορων τρισδιάστατων αντικειμένων περιγράφεται πλήρως.

Το Κεφάλαιο 6 περιγράφει την υλοποίηση του αλγορίθμου Fibonacci για παρουσίαση της τεχνικής υλοποίησης αλγορίθμων του Δυναμικού προγραμματισμού. Συγκεκριμένα η υλοποίηση περιλαμβάνει ένα τρισδιάστατο δυαδικό δένδρο και παρουσίαση των χρόνων εκτέλεσης σε σύγκριση με άλλους αλγορίθμους.

Στο 7^ο και προτελευταίο κεφάλαιο θα ασχοληθούμε με την κατηγορία των Τυχαιοποιημένων αλγορίθμων. Θα αναλυθεί πλήρως η υλοποίηση του αλγορίθμου συρρίκνωσης Karger. Η υλοποίηση απαρτίζεται από τη δημιουργία ενός μη συνεκτικού πολυγράφου, εφαρμογή αποθήκευσης και ανάκτησης από αρχεία και υλοποίησης animation του αλγορίθμου.

Τέλος στο 8^ο και τελευταίο κεφάλαιο θα ακολουθήσει ένας επίλογος μαζί με τα συμπεράσματά μας, παρουσίαση διαφόρων γενικών προβλημάτων και τρόπων επίλυσής τους και ορισμένες μελλοντικές ενέργειες που θα καθιστούσαν πιο καλή και βελτιωμένη την εργασία αυτή.

Κεφάλαιο 2.

Προηγούμενη Γνώση - Υπόβαθρο

2.1 Προηγούμενες Γνώσεις	5
2.2 Εργαλείο Unity3D	6
2.2.1 Περιγραφή Unity3D	6
2.2.2 Χρήση Πλατφόρμας Unity3D	7
2.3 Αναγκαία Επιπλέον Μελέτη	9
2.4 Ο Πρώτος Αλγόριθμος – Selection Sort	10
2.4.1 Περιγραφή Αλγορίθμου	10
2.4.2 Υλοποίηση Εμφάνισης Αλγορίθμου	11
2.4.3 Υλοποίηση Λειτουργίας Αλγορίθμου και Διεπαφής	15
2.4.4 Δυσκολίες και Διευκρινήσεις Υλοποίησης	22

2.1 Προηγούμενες Γνώσεις

Προτού ξεκινήσουμε με την λεπτομερή υλοποίηση των αλγορίθμων, προϋπήρχε μια βάση γνώσης η οποία χρειαζόταν για την ομαλή διεκπεραίωση του προγράμματος αυτού. Συγκεκριμένες βοηθητικές γνώσεις:

- Γνώσεις προγραμματισμού με αντικειμενοστρέφεια (Java) (από τα βασικά μαθήματα αρχών προγραμματισμού και αντικειμενοστρέφειας του Πανεπιστημίου Κύπρου) που θα βοηθούσαν σημαντικά στην εκμάθηση της γλώσσας C# καθώς έχει παρόμοιες αρχές με την Java.
- Γενική γνώση για Αλγορίθμους, δομές δεδομένων και Πολυπλοκότητα (από τα μαθήματα ΕΠΛ336 - Αλγόριθμοι και Πολυπλοκότητα [4,5] και ΕΠΛ231 – Δομές Δεδομένων και Αλγόριθμοι)[8] και συγκεκριμένη γνώση για τρόπο λειτουργίας των αλγορίθμων που επιλέχθηκαν να υλοποιηθούν.

Συμπληρωματικά ερευνήθηκαν και άλλες διπλωματικές εργασίες για να βοηθήσουν στην δομή της υφιστάμενης. Συγκεκριμένα περισσότερη επιρροή ως προς τη βοήθεια για τον τρόπο δομής παρείχε η διπλωματική του φοιτητή Παντελή Στυλιανίδη με την διπλωματική του εργασία «Οπτικοποίηση Παράλληλων Αλγορίθμων Με Τη Μηχανή Unity» [6]

2.2 Εργαλείο Unity3D

2.2.1 Περιγραφή Unity3D



Λογότυπο Unity - Σχήμα 2.1

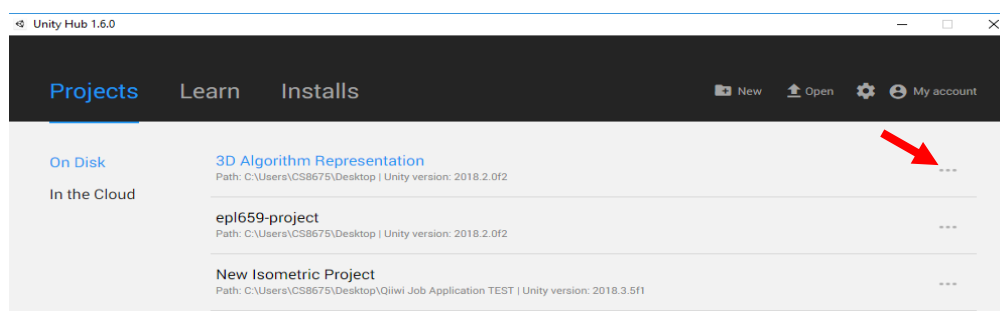
Το Unity3D (Σχήμα 2.1) [1] αποτελεί μια από τις πιο διαδεδομένες και πιο χρήσιμες πλατφόρμες που υπάρχουν στην αγορά για την δημιουργία βιντεοπαιχνιδιών και όχι μόνο. Προσφέρει εργαλεία και υπηρεσίες που σε εξυπηρετούν σε πραγματικό χρόνο χωρίς καθυστερήσεις και επιτρέπει τεράστια προσβασιμότητα σε μεγάλο αριθμό από πλατφόρμες όπως Windows PC, Mac, Linux, Android, IOS, Xbox One, PS4 κλπ. Το πλούσιο και επεκτάσιμο εργαλείο επεξεργασίας των 3D/2D μοντέλων, ήχων, animation και διεπαφών χρήστη αποτελεί ένα από τα πιο χρήσιμα εργαλεία για τους προγραμματιστές.

Επίσης εκτός από τις υψηλές υπολογιστικές επιδόσεις που προσφέρει με προχωρημένα εργαλεία ελέγχου και βελτιστοποίησης, προσφέρει και την δυνατότητα να χρησιμοποιήσεις το δικό της ενσωματωμένο “Cloud” εργαλείο. Το Team Collaboration (**Collab**) χρησιμοποιείται για αποθήκευση, δημοσιοποίηση και συγχρονισμό της εργασίας σου με έλεγχο εκδόσεων. Τέλος με το Unity Asset Store μπορείς να έχεις πρόσβαση σε έτοιμα εργαλεία και άλλο υλικό τόσο δωρεάν όσο και επί πληρωμής για εύκολη επίλυση για χρονοβόρα ή δύσκολα προβλήματα.

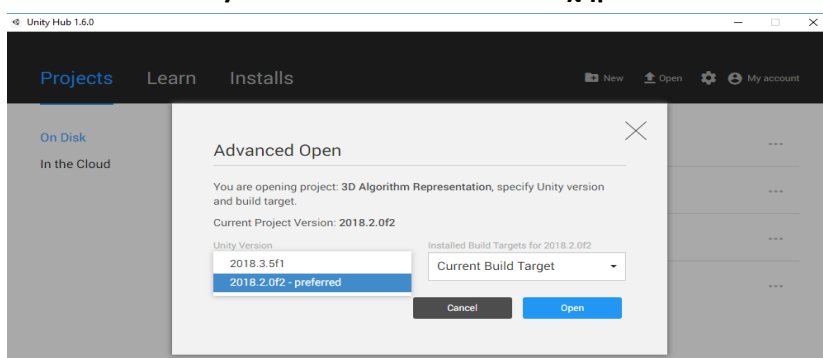
Η πλατφόρμα αυτή είναι χρήσιμη και για αρχάριους χρήστες και προγραμματιστές που μπαίνουν στην βιομηχανία του Game Development. Προσδίδει μεγάλη ευχρηστία με την δυναμική σύνδεσή της μηχανής της μαζί με Scripts με τις γλώσσες προγραμματισμού C# και UnityScript ή αλλιώς JavaScript και Boo.

2.2.2 Χρήση Πλατφόρμας Unity3D

Σε αυτό το υποκεφάλαιο θα αναφέρουμε λίγες βασικές λειτουργίες του Unity3D που πρέπει να γνωρίζουμε για την κατανόηση της διπλωματικής [1,2,3]. Αρχικά, ο ιδανικότερος τρόπος με τον οποίο χειριζόμαστε την πλατφόρμα αυτή είναι δια μέσου του Unity Hub, το οποίο επιτρέπει να ελέγχεις με ποια έκδοση να ανοίξεις μια οποιοδήποτε εργασία φτιαγμένη μέσω του Unity. Αυτό είναι σημαντικό καθώς αν χειρίζεσαι μια εργασία σου σε πολλαπλούς υπολογιστές, θα πρέπει να την επεξεργάζεσαι με την ίδια έκδοση Unity από παντού. Αν χρησιμοποιούσαμε πιο παλιά ή πιο καινούρια έκδοση για επεξεργασία μιας υφιστάμενης εργασίας, ίσως αλλοιώνονταν τα αρχεία και καταστρέφονταν σημαντικά μας δεδομένα.



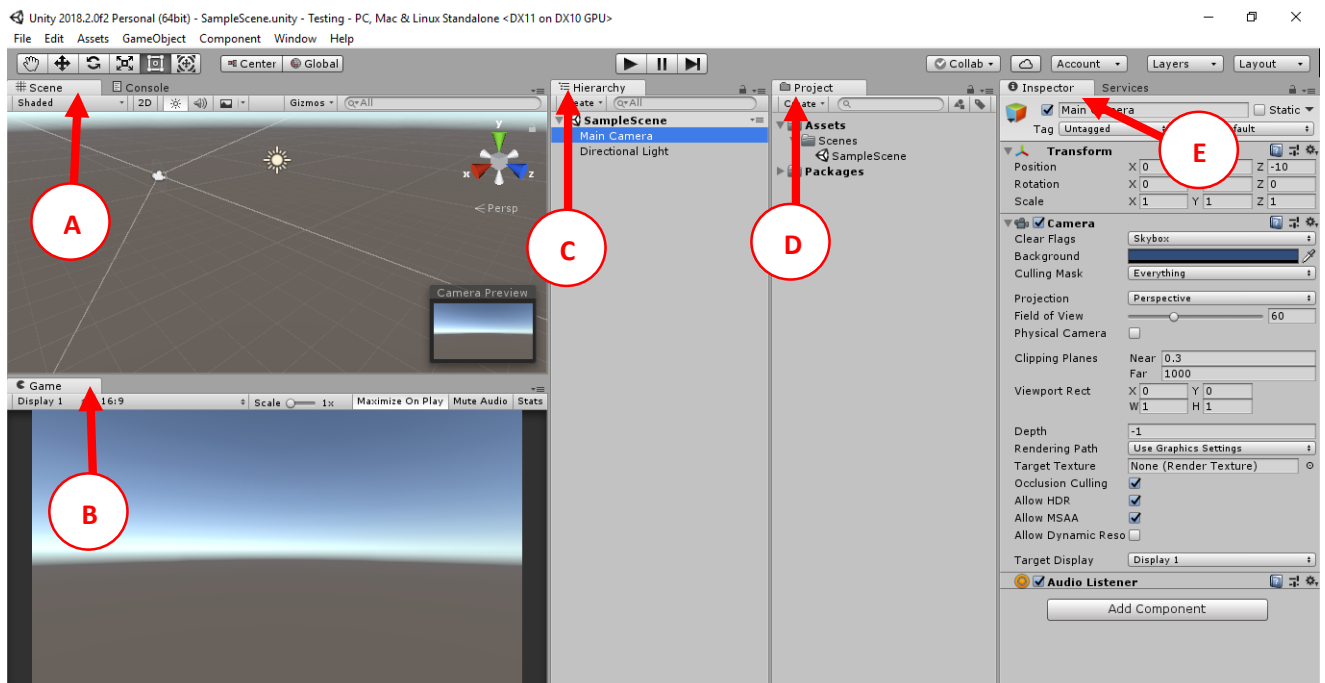
Unity Hub 1.6.0 –Main Screen - Σχήμα 2.2



Unity Hub – Open By - Σχήμα 2.3

Όπως φαίνεται στο Σχήμα 2.2, επιλέγοντας μια εργασία με αυτό το εργαλείο, εμφανίζονται οι πιο κάτω επιλογές όπως είναι στο Σχήμα 2.3. Εκεί μπορούμε να επιλέξουμε πιο συγκεκριμένα με ποια έκδοση να ανοίξουμε το Unity. Στην περίπτωση μας, η έκδοση που χρησιμοποιούμε είναι η **2018.2.0f2**. Αυτή η έκδοση θα παραμείνει όπως και πρέπει καθ' όλη την υλοποίηση της εργασίας, σε όποιο υπολογιστή και να εργαστούμε.

Η εργασία μας είναι αρχικά δημιουργημένη ως μια εργασία 3D. (εμφανίζεται σαν επιλογή κατά την δημιουργία μιας καινούριας εργασίας/project). Σημαντική έννοια που πρέπει να ξέρουμε είναι αυτή των **GameObjects**. Αυτά είναι τα θεμελιώδη αντικείμενα του Unity που μπορούν να αντιπροσωπεύουν χαρακτήρες, μοντέλα και σκηνικά. Δρουν ως δοχεία για τα **Components** τα οποία αυτά υλοποιούν την πραγματική λειτουργικότητα. Τα components μπορεί να είναι scripts, components φωτισμού, φυσικής, μετασχηματισμών και άλλα πολλά.



Κύρια διεπαφή Unity3D για μια 3D εργασία - Σχήμα 2.4

Με την δημιουργία μιας νέας εργασίας εμφανίζονται τα πιο πάνω βασικά παράθυρα και εργαλεία (Σχήμα 2.4):

- A. **Scene View** → το παράθυρο στο οποίο δουλεύουμε με GameObjects όπως μοντέλα γραφικών, φωτισμοί, κάμερες και colliders για την δημιουργία μιας σκηνής (**Scene**)
- B. **Game View** → το παράθυρο όπου μπορούμε να κάνουμε προεπισκόπηση της εργασίας μας και την υφιστάμενη κατάσταση στην οποία βρίσκεται κατά την υλοποίηση.
- C. **Hierarchy Window** → παρουσιάζει σε μια λίστα τα GameObjects στην υφιστάμενη ανοικτή σκηνή σε μια ιεραρχική σειρά.

D. **Project Window** → εδώ μπορούμε να κάνουμε εισαγωγή, αποθήκευση και επεξεργασία των αρχείων **Assets**.

E. **Inspector Window** → αυτό το παράθυρο παρουσιάζει όλες τις επιλογές και ιδιότητες οποιουδήποτε αντικειμένου GameObject, Asset ή Setting έχουμε επιλεγμένο εκείνη τη στιγμή. Είναι σημαντικό παράθυρο καθώς μας δίνει την δυνατότητα να προσθέσουμε διάφορα Components πάνω στα αντικείμενά μας.

Με την διαχείριση και επέκταση των πιο πάνω εργαλείων είτε από δική μας υλοποίηση είτε με έτοιμα εργαλεία από το **Asset Store** του Unity (που έχουμε πρόσβαση από ένα συγκεκριμένο παράθυρο μέσω του Unity), μπορούμε να υλοποιήσουμε οτιδήποτε λογισμικό φανταστούμε και είναι εφικτό με 3D ή 2D γραφικά.

Περισσότερες λεπτομερείς λειτουργίες και επιπλέον βασικές έννοιες του Unity και της C# θα αναλυθούν κατά τη διάρκεια της διπλωματικής αυτής.

2.3 Αναγκαία Επιπλέον Μελέτη

Για την χρήση του Unity3D για τον συγκεκριμένο μας σκοπό πρέπει να παρακολουθήσουμε ορισμένα εκπαιδευτικά φροντιστήρια που προσφέρονταν από την ιστοσελίδα του Unity και άλλα φροντιστήρια και βοήθειες διαδικτυακά.

- Unity3D Tutorials [3]:
 - Basic Tutorials (βασικά φροντιστήρια για τις βασικές αρχές του Unity)
 - Roll-a-ball (εκμάθηση στη δημιουργία ενός απλού παιχνιδιού)
 - Interface & Essentials (επεξήγηση διεπαφής και βασικών εργαλείων)
 - Scripting: Beginner, Intermediate (για εκμάθηση της γλώσσας C#)
 - Animation (κίνηση τρισδιάστατων αντικειμένων)
- Unity Manual [2] (περιέχει πολλές έννοιες για όλα τα εργαλεία του Unity)

Αυτά ήταν άκρως αναγκαία και χρήσιμα για την σωστή και πλήρη υλοποίηση της γραφικής απεικόνισης των αλγορίθμων. Επιπλέον έχουν γίνει περαιτέρω μελέτες κατά τη διάρκεια της υλοποίησης για εξακρίβωση και διόρθωση προβλημάτων που εμφανίζονταν αναπάντεχα. Αυτά τα προβλήματα θα αναλυθούν αργότερα στην διπλωματική.

2.4 Ο Πρώτος Αλγόριθμος – Selection Sort

2.4.1 Περιγραφή Αλγορίθμου

Για αρχή προτού ξεκινήσουμε με την υλοποίηση των βασικών αλγορίθμων, έχει γίνει μια αρχική υλοποίηση του εύκολου στην αναπαράσταση αλγορίθμου ταξινόμησης με επιλογή (Selection Sort). Η υλοποίηση αυτού του αλγορίθμου, αποτελεί βάση για λειτουργίες που θα εφαρμοστούν στους υπόλοιπους. Μετά από μια αρχική ανάλυση του κώδικα και του τρόπου λειτουργίας του αλγορίθμου, σχεδιάστηκε πάνω στο χαρτί μια αρχική εικόνα για το πώς θα φαίνεται στην ψηφιακή του απεικόνιση και στη συνέχεια έχει υλοποιηθεί μέσω του Unity3D.

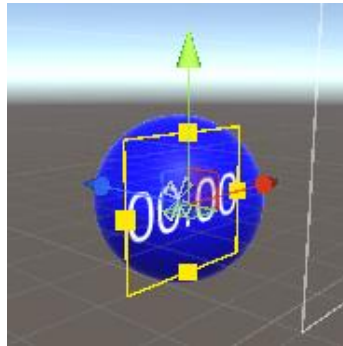
```
1  /*Selection Sort function*/
2  void SelectionSort(float[] arr)
3  {
4      // The length of the array
5      int n = arr.Length;
6
7      // One by one move boundary of unsorted subarray
8      for (int i = 0; i < n - 1; i++)
9      {
10         // Find the minimum element in unsorted array
11         // beginning from the next element to i and forward
12         int minIndex = i;
13         for (int j = i + 1; j < n; j++)
14         {
15             if (arr[j] < arr[minIndex])
16                 minIndex = j;
17         }
18
19         // Swap the found minimum element
20         // with the first element
21         float tmp = arr[minIndex];
22         arr[minIndex] = arr[i];
23         arr[i] = tmp;
24     }
25 }
```

Αλγόριθμος Selection Sort - Σχήμα 2.5

Πιο πάνω φαίνεται μια απλή υλοποίηση του αλγορίθμου ταξινόμησης με επιλογή στην C# (Σχήμα 2.5). Όπως βλέπουμε είναι ένας σχετικά απλός αλγόριθμος ταξινόμησης ενός πίνακα με πραγματικούς-float αριθμούς. Η χρονική του πολυπλοκότητα είναι $O(n^2)$ καθώς έχουμε τα 2 φωλιασμένα for-loops και η χωρική του $O(1)$ καθώς έχουμε μόνο ένα μονοδιάστατό πίνακα με όλους τους αριθμούς. Επίσης ένα κάλο αυτού του αλγορίθμου είναι ότι δεν κάνει ποτέ περισσότερες από n ανταλλαγές τιμών (swaps).

2.4.2 Υλοποίηση Εμφάνισης Αλγορίθμου

Ως αρχική σκέψη, οι αριθμοί αυτοί θα μπορούσαν να αναπαρασταθούν μέσα σε 3D σφαίρες οι οποίες θα μπορούν με ευκολία να χρωματίζονται (με χρήση του `Renderer Component -> material -> color`) και να μετακινούνται σε όποιο σημείο του χώρου θέλουμε. Έτσι δημιουργήσαμε την πιο κάτω σφαίρα (Σχήμα 2.6) η οποία έχει αποθηκευτεί ως **Prefab** για επαναχρησιμοποίηση.



BallTextMesh - Σχήμα 2.6

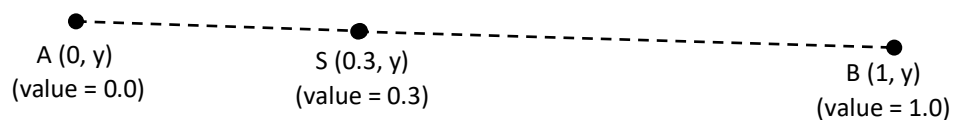
Τα Prefabs είναι αντικείμενα τύπου **GameObject** που μας επιτρέπουν να αποθηκεύσουμε ένα στοιχείο με όλες του τις ιδιότητες στο εσωτερικό του prefab και το prefab να λειτουργεί βασικά ως πρότυπο που χρησιμοποιείτε για να δημιουργήσουμε νέες εμφανίσεις του ίδιου αντικειμένου στη σκηνή. Θα χρησιμοποιούνται πολύ συχνά κατά την πορεία υλοποίησης της εργασίας μας.

Εδώ να σημειωθεί ότι έχει χρησιμοποιηθεί το εργαλείο **TextMeshPro** το οποίο έχουμε κατεβάσει από το Asset Store του Unity. Αυτό το εργαλείο προσφέρει πολύ περισσότερες επιλογές διαμόρφωσης και μια πιο ωραία εμφάνιση σε γραπτό κείμενο τόσο σε 3D μορφή όσο και σε διεπαφές (UI). Αυτό θα μας είναι χρήσιμο σε όλη τη διπλωματική. Επιπρόσθετα για να μπορεί να μετακινηθεί η σφαίρα σε όποιο σημείο του χώρου θέλουμε έχουμε δημιουργήσει ένα Script που είναι πάντοτε πάνω σε αυτή τη σφαίρα, το **BallMover.cs** το οποίο επιτρέπει στη σφαίρα να κινηθεί με μία παραβολική καμπύλη προς ένα επιθυμητό σημείο που ορίζεις εσύ χρησιμοποιώντας με τη σειρά του τη βοηθητική κλάση **MathParabola.cs** [7] (όπως φαίνονται στο Παράρτημα Α).

Για την υλοποίηση έχει ληφθεί υπόψη και πόσο μικρά/μεγάλα θα ήταν τα δεδομένα για να μπορεί ο χρήστης να τα δει σε ικανοποιητικό επίπεδο οπτικά και να είναι ευανάγνωστα

καθώς εξυπηρετούν ταυτόχρονα την σωστή επεξήγηση του αλγορίθμου. Έτσι έχουμε ορίσει όριο δύο δεκαδικών ψηφίων και δύο ακεραίων.

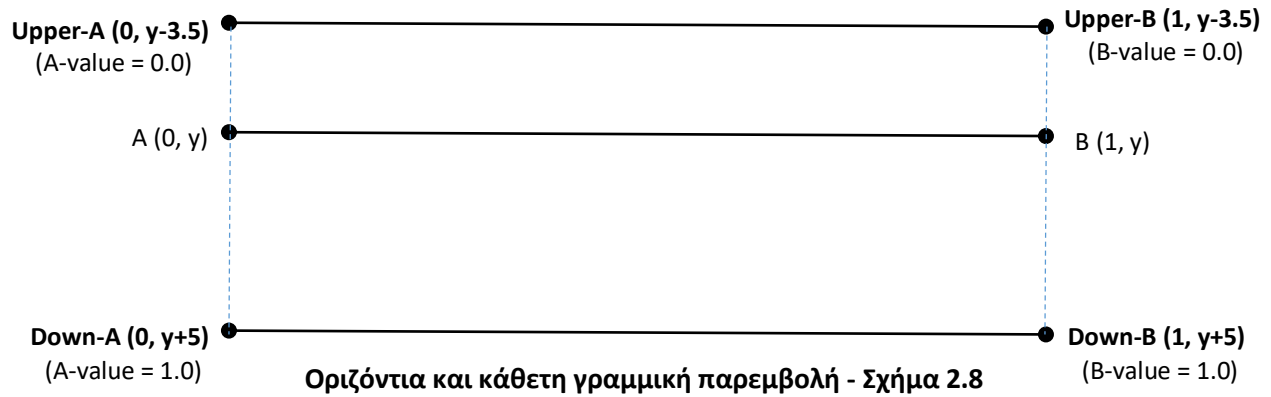
Για να υπάρχει μια δομημένη εμφάνιση του πίνακα, επιλέξαμε να εμφανίσουμε κάθε στοιχείο του σε αυτές τις σφαίρες, οι οποίες θα είναι ίδιου μεγέθους, με ισότιμη απόσταση μεταξύ τους και με σωστή σειρά όπως εμφανίζονται στον πίνακα από την 0 μέχρι και την N θέση (όπου N ο αριθμός των στοιχείων). Ο τρόπος που έχουμε πετύχει αυτή την υλοποίηση είναι με χρήση γραμμικής παρεμβολής (Linear Interpolation - **Lerp**) [2]. Η γραμμική παρεμβολή επιτρέπει την σταθερή εμφάνιση αντικειμένων από μια αρχική (A) θέση στο χώρο σε μία τελική (B) (όπου 0 – Αρχική θέση, 1 – Τελική θέση) σε όποιο σημείο (S) θέλεις επιλέγοντας ένα πραγματικό αριθμό μεταξύ του 0.0 και 1.0 συμπεριλαμβανομένων όπως φαίνεται στο πιο κάτω παράδειγμα (Σχήμα 2.7).



Γραμμική Παρεμβολή 2 σημείων - Σχήμα 2.7

Όπως βλέπουμε επιλέγοντας το 0.3 ως το σημείο S, το σημείο αυτό θα βρίσκεται στο 30% της απόστασης της ευθείας που ενώνει το A(0,y) με το B(1,y) δηλαδή θα είναι το σημείο (0.3,y) (όπου (X,Y) σημείο σε δισδιάστατο χώρο). Έτσι αν διαλέξουμε να τοποθετήσουμε μια σφαίρα στο σημείο 0.3 από το A στο B, θα τοποθετηθεί στο (0.3,y). Συνεπώς έχουμε δημιουργήσει μια συνάρτηση η οποία τοποθετεί κάθε στοιχείο του πίνακα σε μια σφαίρα και στην συνέχεια τα κατανέμει ισότιμα στο χώρο σε μια γραμμή.

Αυτό όμως δεν επαρκεί για την παρουσίαση πολλών σφαιρών στην οθόνη που να είναι ευδιάκριτες προς τον χρήστη. Μετά από πολλούς ελέγχους παρουσίασης των σφαιρών έχουμε παρατηρήσει ότι ο ιδανικός μέγιστος αριθμός σφαιρών που μπορεί να έχει μια γραμμή είναι 30. Έτσι κάθε 30 σφαίρες θα αλλάζουμε γραμμή και ταυτόχρονα θα διατηρούμε ισάξιες αποστάσεις μεταξύ αυτών των γραμμών. Αυτό θα το πετύχουμε εφαρμόζοντας την ίδια τεχνική γραμμικής παρεμβολής αλλά αυτή την φορά στον κάθετο άξονα στην αρχική και τελική θέση που έχουμε ορίσει προηγουμένως.

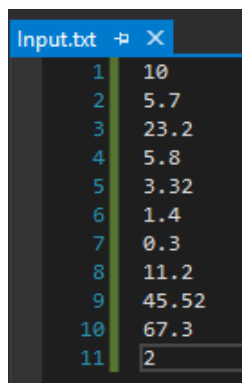


Όπως βλέπουμε στο Σχήμα 2.8 για το αρχικό (A) και τελικό (B) σημείο της κεντρικής γραμμής έχουμε ορίσει δύο άλλα κάθετα προς τα πάνω και προς τα κάτω. Οι αριθμοί που χρησιμοποιούμε έχουν υπολογιστεί από ελέγχους μετά από διάφορες εκτελέσεις του αλγορίθμου στο Unity. (-3.5 στον Y άξονα για το πάνω όριο και +5 για το κάτω όριο (Συνολική απόσταση = 8.5)). Συνεπώς μετά από κάθε 30 αριθμούς θα δημιουργούμε καινούρια γραμμή μεταξύ αυτών των ορίων. Για παράδειγμα: αν έχουμε 65 αριθμούς ξέρουμε ότι θα δημιουργήσουμε 3 γραμμές ($65 = 2 \cdot 30 + 5$), 2 γραμμές θα έχουν από 30 σφαίρες και μία επιπλέον τις υπόλοιπες 5. Η 1^η γραμμή θα είναι στο $1/3$ ($8.5 \cdot 1/3 = 2.833$) η 2^η στα $2/3$ (5.666) και η 3^η στα $3/3$ (8.5) της απόστασης μεταξύ Upper-A με Down-A και Upper-B με Down-B στον άξονα Y.

Τέλος θα πρέπει να λάβουμε υπόψη και τα μεγέθη των σφαιρών καθώς τοποθετούνται στις γραμμές. Αν έχουμε λίγες σφαίρες θα πρέπει να εμφανίζονται πιο μεγάλες ενώ αν έχουμε πιο πολλές θα είναι πιο μικρές. Αυτό θα συμβαίνει όταν έχουμε μονάχα 1 γραμμή για καλύτερη παρουσίαση των σφαιρών. Σε πολλές γραμμές τα μεγέθη των σφαιρών θα είναι τα ίδια για να μην υπάρχει συνωστισμός μεταξύ τους.

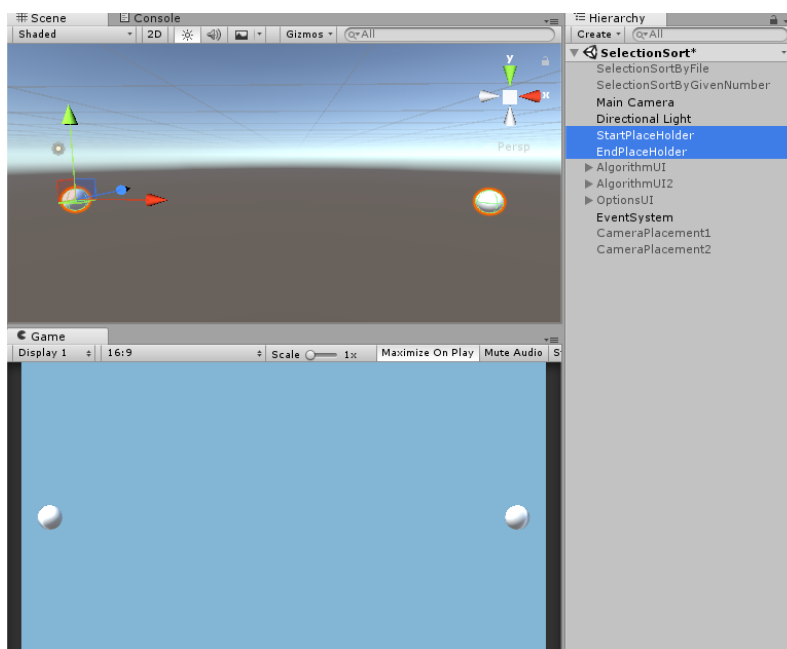
Για την εισαγωγή των αριθμών έχουμε χρησιμοποιήσει ένα αρχείο κειμένου το οποίο βρίσκεται στο φάκελο “Resources” της εργασίας μας. Το διάβασμα έχει υλοποιηθεί με χρήση του Script **ReadFile.cs** που βρίσκεται στο Παράρτημα Α. Το αρχείο “Input.txt” πρέπει να έχει την ακόλουθη μορφή: στην 1^η γραμμή να βρίσκεται ο αριθμός των στοιχείων που θα διαβαστούν και στις επόμενες γραμμές, όλα τα στοιχεία τα οποία θα χρησιμοποιηθούν σε δεκαδική μορφή.

Παράδειγμα το πιο κάτω αρχείο με 10 στοιχεία (Σχήμα 2.9).



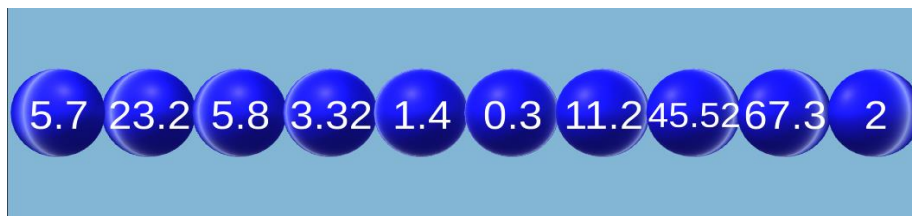
```
Input.txt
1 10
2 5.7
3 23.2
4 5.8
5 3.32
6 1.4
7 0.3
8 11.2
9 45.52
10 67.3
11 2
```

Αρχείο "Input.txt" - Σχήμα 2.9



Lerp between 2 points in Unity3D - Σχήμα 2.10

Στη συνέχεια όπως βλέπουμε στο Σχήμα 2.10, έχουμε τοποθετήσει δύο σφαιρικά placeholders (σημεία στο χώρο) Start και End σε σημείο ώστε να φαίνονται ικανοποιητικά στο χρήστη από το Game View. Επειδή όμως εμείς θέλουμε μόνο τις θέσεις των σημείων, απενεργοποιούμε το Mesh Renderer component και από τις 2 σφαίρες για να μην φαίνονται.



Παράδειγμα Lerp 10 σφαιρών - Σχήμα 2.11

Και έτσι καταλήγουμε με την πιο πάνω εμφάνιση (Σχήμα 2.11) του πίνακα δεκαδικών με βάση το αρχείο του Σχήματος 2.9. Εάν αυξήσουμε τον αριθμό των στοιχείων στα 30 θα έχουμε το μέγιστο αριθμό στοιχείων σε μια γραμμή όπως φαίνεται πιο κάτω στο Σχήμα 2.12. Επίσης παρατηρούμε ότι οι σφαίρες έχουν μικρύνει.

Παράδειγμα Lerp 30 σφαιρών - Σχήμα 2.12

Τώρα αν έχουμε περισσότερα από 30 στοιχεία, θα εμφανίζονται περισσότερες γραμμές με τον τρόπο που έχουμε αναφέρει πιο πριν. Αυτό φαίνεται στο πιο κάτω παράδειγμα (Σχήμα 2.13) όπου έχουμε σύνολο 200 στοιχεία και άρα έχουμε 6 γραμμές με 30 στοιχεία και ακόμα 1 με τα υπόλοιπα 20.

Παράδειγμα Lerp 200 σφαιρών - Σχήμα 2.13

Τώρα που έχουμε ολοκληρώσει την παρουσίαση των στοιχείων του πίνακα πάμε στην υλοποίηση του αλγορίθμου με χρήση χρωματικών ενδείξεων και κινήσεων καθώς επίσης και της διεπαφής.

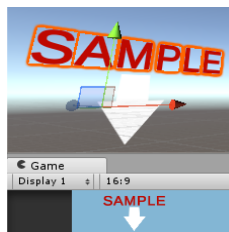
2.4.3 Υλοποίηση Λειτουργίας Αλγορίθμου και Διεπαφής:

Ο βασικός τρόπος με τον οποίο θα μετατρέψουμε τόσο τον αλγόριθμο Selection Sort όσο και τους υπόλοιπους αλγόριθμους σε οπτικούς με γραφικά και κινήσεις είναι με την χρήση **Coroutines**[2]. Ουσιαστικά τα coroutines είναι συναρτήσεις που επιστρέφουν πίσω μόνο όταν ολοκληρωθεί η λειτουργία τους. Αυτό σημαίνει ότι κάθε δράση που συμβαίνει μέσα

σε αυτή τη συνάρτηση, πρέπει να συμβεί σε διάστημα μιας ενημέρωσης ενός frame (συνήθως 50ms). Επίσης αυτή η συνάρτηση έχει την δυνατότητα παύσης της εκτέλεσης και επιστροφής του ελέγχου στο Unity και μετά με το ξανακάλεσμά της, να συνεχίσει από εκεί που έχει μείνει. Με τη χρήση των δύο πιο κάτω τρόπων μπορούμε να έχουμε πλήρη έλεγχο οποιουδήποτε αλγορίθμου για τον χρόνο στον οποίο εκτελείτε και παρουσιάζεται στην οθόνη κάθε γραμμής κώδικά του:

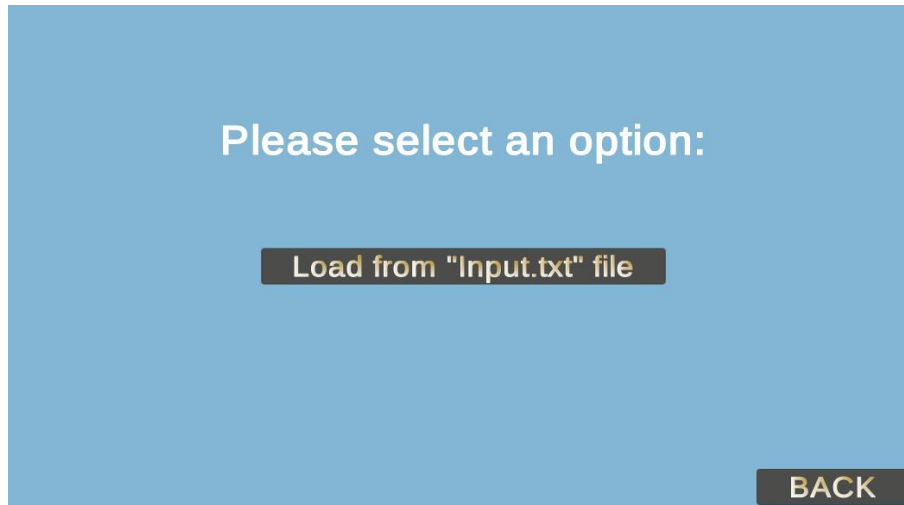
- **yield return new WaitUntil(() => *Boolean*);** => Αναμονή σε αυτό το σημείο του κώδικα μέχρι να ενεργοποιηθεί η έκφραση *Boolean* (=true). (Μπορούμε να χρησιμοποιήσουμε κουμπιά ή οτιδήποτε άλλο χρησιμοποιεί εκφράσεις Boolean για έλεγχο της λειτουργίας ενός αλγορίθμου)
- **yield return new WaitForSeconds(*Float*);** => Αναμονή μέχρι να περάσει το χρονικό διάστημα *Float* σε δευτερόλεπτα. (εδώ μπορεί να καθοριστεί η ταχύτητα που τρέχει ο αλγόριθμος με χρήση μεταβλητής)
- **yield return StartCoroutine(*Coroutine*);** => Αναμονή μέχρι να ολοκληρωθεί μια Coroutine η οποία θα καλεστεί ταυτόχρονα.
- Περισσότερες λεπτομερείς λειτουργίες θα αναλυθούν κατά τη διάρκεια της διπλωματικής αυτής.

Για την εξακρίβωση της σφαίρας την οποία ελέγχουμε εκείνη τη συγκεκριμένη στιγμή και του ελάχιστου στοιχείου του πίνακα με το οποίο συγκρίνετε, έχουμε χρησιμοποιήσει ένα animated τόξο όπως φαίνεται στο πιο κάτω Σχήμα 2.14.



Pointing Arrow prefab - Σχήμα 2.14

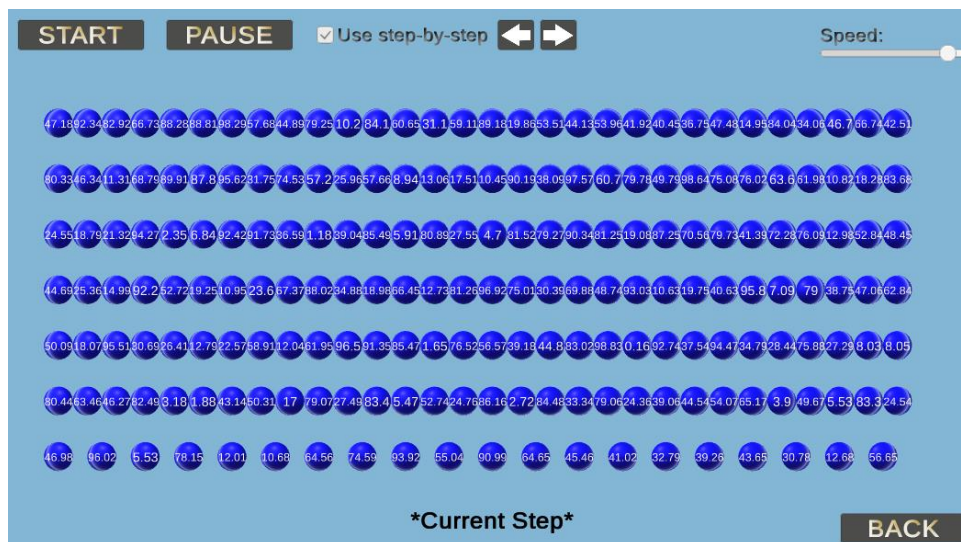
Χρησιμοποιώντας το Animation σύστημα του Unity, έχουμε εφαρμόσει μια ελαφρώς πάνω-κάτω κίνηση στο τόξο αυτό μετακινώντας τη θέση του. Η τοποθεσία του τόξου καθορίζεται μέσω κώδικα κατά την εκτέλεση του αλγορίθμου.



Αρχική διεπαφή Selection Sort - Σχήμα 2.15

Η αρχική διεπαφή του αλγορίθμου μας (Σχήμα 2.15) έχει υλοποιηθεί με μια μονάχα επιλογή λόγω του ότι αποτελεί απλή ενδεικτική δοκιμή για ευκολότερη υλοποίηση των διεπαφών στην συνέχεια των υπόλοιπων αλγορίθμων.

- Με το κουμπί “BACK” (UI->**Button GameObject**) μπορούμε να πάμε πίσω στην αρχική οθόνη η οποία θα αναλυθεί στο επόμενο κεφάλαιο (Κεφάλαιο 3).
- Με το κουμπί “Load from “Input.txt” file” μεταφερόμαστε σε άλλη διεπαφή όπου μπορούμε να δούμε τον αλγόριθμο σε δράση (Σχήμα 2.16 – παράδειγμα με 200 στοιχεία).

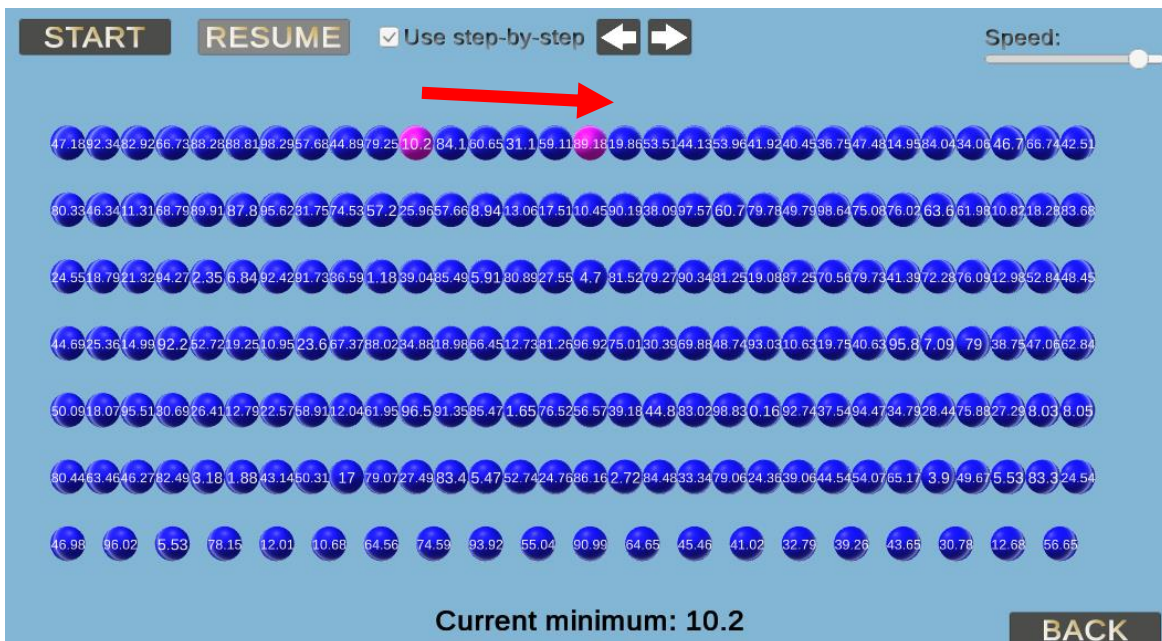


Διεπαφή ελέγχου Selection Sort - Σχήμα 2.16

Αλλάζοντάς διεπαφή, φορτώνονται τα στοιχεία από το αρχείο “Input.txt” όπως έχουμε περιγράψει πιο πριν. Εμφανίζονται επιπλέον οι πιο κάτω επιλογές:

- Κουμπί “START”: Επιλέγοντάς το, αρχίζει να εκτελείται ο αλγόριθμος οπτικά.
- Κουμπί “PAUSE/RESUME”: Παύση του αλγορίθμου σε οποιοδήποτε σημείο εκτέλεσης του και συνέχεια.
- Κουτί επιλογής (UI->Toggle) “Use step-by-step”: Όταν είναι επιλεγμένο, εμφανίζονται τα πιο κάτω κουμπιά:
 - Κουμπί Step back (←): Προχωρά πίσω ένα βήμα (παίρνει το προηγούμενο στοιχείο που έχει ταξινομηθεί στην προηγούμενή του θέση)
 - Κουμπί Step forward (→): Προχωρούμε κανονικά μπροστά ένα βήμα (ταξινομείται το επόμενο στοιχείο κανονικά)
- “Speed” slider (UI->Slider): Καθορίζει την ταχύτητα εκτέλεσης του αλγορίθμου. (Όσο πιο δεξιά βρίσκεται, τόσο πιο γρήγορα τρέχει)
- Κουμπί “BACK”: Μας παίρνει πίσω στην αρχική οθόνη των αλγορίθμων.
- Κείμενο “*Current Step*” (TextMeshProUGUI): Παρουσιάζει σε πιο βήμα βρίσκεται ο αλγόριθμος την συγκεκριμένη χρονική στιγμή που τρέχει.

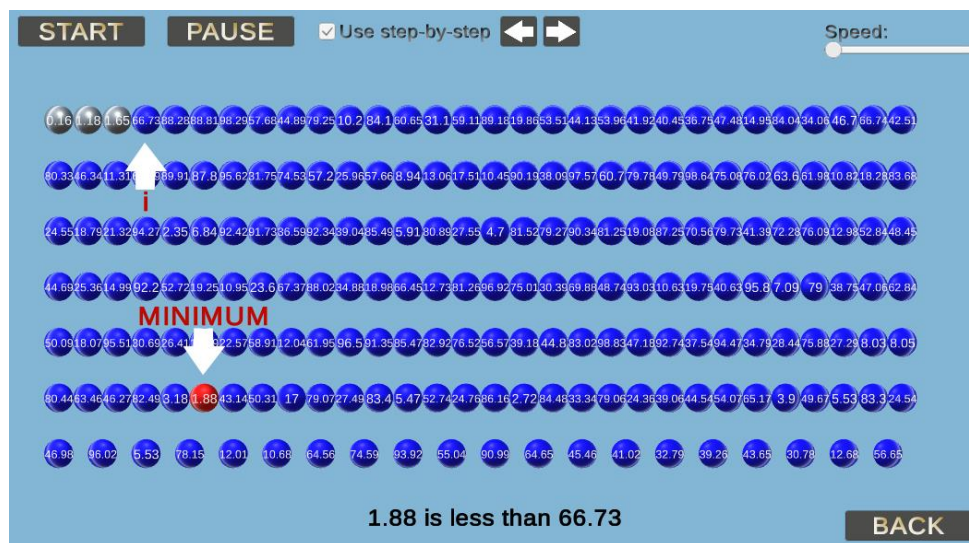
Ξεκινώντας τον αλγόριθμο επιλέγοντας το “START”, βλέπουμε το πιο κάτω (Σχήμα 2.17):



Visual Selection Sort – Βήμα 1 - Σχήμα 2.17

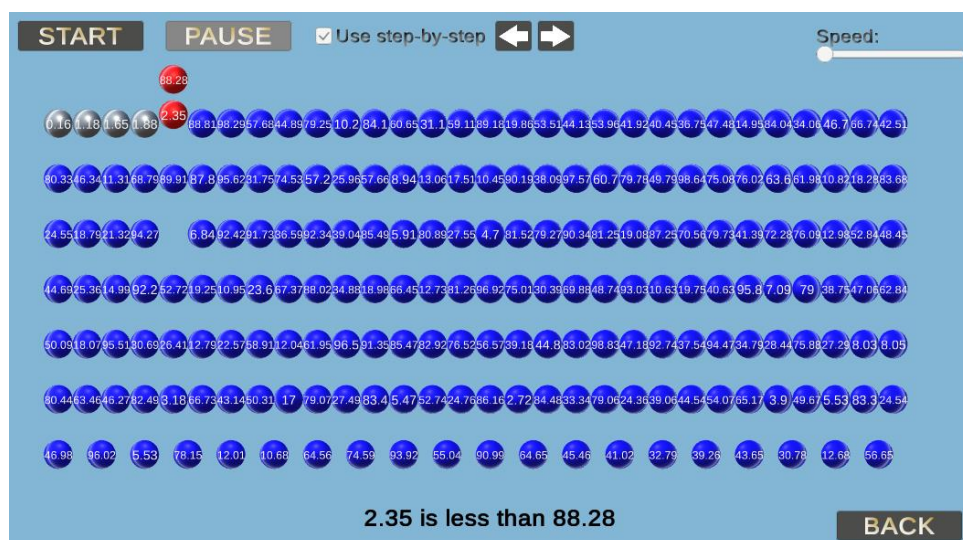
Βλέπουμε ότι αρχίζει να γίνεται προσπέλαση όλων των σφαιρών καθώς χρωματίζονται με χρώμα ροζ-μωβ. Η προσπέλαση γίνεται από τα αριστερά προς τα δεξιά και από την πάνω σειρά προς την κάτω. Όταν βρεθεί πιο μικρό στοιχείο σε σχέση με αυτό της πρώτης σφαίρας που δεν έχει χρωματιστεί γκριζα ακόμη (γκρίζες χρωματίζονται οι σφαίρες οι οποίες έχουν ταξινομηθεί), τότε το “Current minimum” θα είναι ίσο με αυτό το στοιχείο. Το στοιχείο παραμένει ροζ, και παρουσιάζεται στο κάτω μέρος της οθόνης.

Μόλις ολοκληρωθεί η σάρωση όλων των στοιχείων και η εξακρίβωση του ελάχιστου, εμφανίζονται δύο τόξα (που έχουμε δείξει πιο πριν) με την **SetActive(true)** συνάρτηση για ένα GameObject αντικείμενο, τα οποία παρουσιάζουν με ποια στοιχεία γίνεται η σύγκριση (Σχήμα 2.18). Όπου “i” το στοιχείο στο οποίο βρισκόμαστε και θα γίνει η σύγκριση με το “MINIMUM” που έχει βρεθεί πριν. Το ελάχιστο στοιχείο παίρνει κόκκινο χρώμα. Επίσης βλέπουμε ότι «1.88 is less than 66.73» όπως φαίνεται στο παράδειγμα αυτό και έτσι τα στοιχεία δεν θα αλλάξουν θέση. Τα στοιχεία τα οποία έχουν ταξινομηθεί προηγουμένως όπως βλέπουμε έχουν γίνει γκριζα.



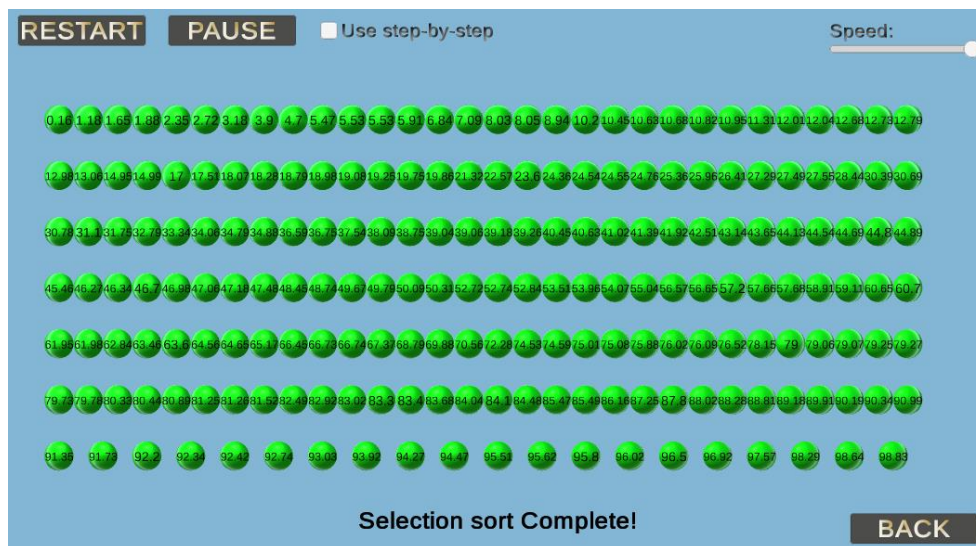
Visual Selection Sort – Βήμα 2 - Σχήμα 2.18

Στην συνέχεια αν το “i” στοιχείο είναι μεγαλύτερο από το ελάχιστο το οποίο έχει βρεθεί τότε θα εκτελεστεί μια ανταλλαγή (swap) των στοιχείων αυτών με χρήση του BallMover το οποίο έχουμε προαναφέρει (Σχήμα 2.19). Τα τόξα αφαιρούνται με την **SetActive(false)**.



Visual Selection Sort – Βήμα 3 - Σχήμα 2.19

Η εκτέλεση αυτή επαναλαμβάνεται συνεχώς μέχρι να γίνουν γκριζα όλα τα στοιχεία του πίνακα και αποτελεσματικά να έχει ταξινομηθεί. Μόλις ολοκληρωθεί η εκτέλεση, όλες οι σφαίρες παίρνουν χρώμα πράσινο με μαύρα γράμματα και παρουσιάζεται το μήνυμα “Selection Sort Complete!” (Σχήμα 2.20)



Visual Selection Sort – Βήμα 4 - Σχήμα 2.20

Με τη χρήση του Step back (←) κουμπιού μπορούμε πάντοτε να πάμε ένα βήμα πίσω. Αυτό έχει επιτευχθεί με χρήση απλής στοιβας στην οποία αποθηκεύουμε τις τοποθεσίες (Indices) των δύο στοιχείων που έχουν ανταλλαχθεί κάθε φορά. (χρήση script-αντικειμένου **Point.cs** που δημιουργήθηκε για να αποθηκεύει 2 ακαιρέους (x,y)). Όλη η πιο πάνω υλοποίηση έχει γίνει στο **SelectionSortByFile.cs** script του παραρτήματος Α.

2.4.4 Δυσκολίες και Διευκρινήσεις Υλοποίησης

Στην ενότητα αυτή θα περιγράψουμε ορισμένα προβλήματα και δυσκολίες οι οποίες έχουν εμφανιστεί κατά την υλοποίησή μας. Για μερικά από αυτά τα προβλήματα αναφέρεται η λύση του. Τέλος θα αναφέρουμε και ορισμένες διευκρινήσεις.

Η φόρτωση από το αρχείο "Input.txt" δεν είναι εφικτή όταν η εργασία είναι στην **Build**-εκτελέσιμή της μορφή. Ο τρόπος που επιλύσαμε αυτό το πρόβλημα είναι με την χρήση κοινού φακέλου που προσφέρει το Unity με όνομα "Streaming Assets" [2]. Η λύση φαίνεται πιο κάτω (Σχήμα 2.21) από το ReadFile.cs του Παραρτήματος Α.

```
string filePath = "";  
// If we are running the project on the editor  
// then load the default file  
if (Application.isEditor)  
{  
    filePath = "Assets/Resources/" + file;  
}  
// else load the file on the mutual StreamingAssets folder  
// on Windows or MAC.  
else  
{  
    filePath = Application.dataPath + "/StreamingAssets/" + file;  
}  
Debug.Log(Application.dataPath + "/StreamingAssets/" + file);  
StreamReader reader = new StreamReader(filePath);
```

Κώδικας φόρτωσης αρχείου από ένα path - Σχήμα 2.21

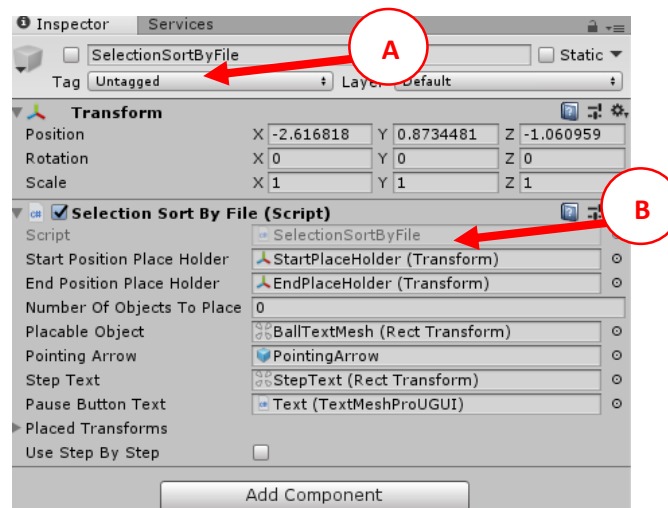
Δοκιμάστηκε αρχικά να γίνονται οι μετακινήσεις των σφαιρών και των αντικειμένων στον χώρο με χρήση Animations που προσφέρει το Unity αλλά στην πορεία δεν λειτουργούσαν σωστά. Τα Animations δεν προσφέρουν δυναμικές μετακινήσεις των αντικειμένων στο χώρο ούτε αλλαγές στις ιδιότητές τους. Έτσι στη θέση τους χρησιμοποιήθηκε κώδικας από scripts ο οποίος ελέγχει τις παραμέτρους ενός αντικειμένου (GameObject) που θέλουμε να κάνουμε animate. Η μόνη εξαίρεση στην διπλωματική είναι το animated τόξο που έχει γίνει στον αλγόριθμο αυτό.

Το animated τόξο δεν μπορεί να μετακινηθεί σε οποιοδήποτε σημείο του χώρου θέλουμε καθώς περιέχει **Animator** με ένα συγκεκριμένο Animation ως component το οποίο κινεί το τόξο σε ένα συγκεκριμένο σημείο του χώρου. Για να μπορούμε να το μετακινούμε σε όποιο σημείο θέλουμε, το έχουμε ορίσει ως παιδί (Child) ενός άδειου GameObject. Έτσι μετακινώντας τον πατέρα (Parent), μπορούμε να μετακινήσουμε το τόξο σε όποιο σημείο θέλουμε με το animation να εκτελείται κανονικά.

Η ευθυγράμμιση μιας σφαίρας στην άκρη ενός animated τόξου, αν το τόξο αλλάξει σε μέγεθος δυστυχώς δεν μπορεί να γίνει.

Τρόπος εμφάνισης έτοιμων αντικειμένων πολλαπλές φορές με χρήση κώδικα σε συγκεκριμένο σημείο στο χώρο: Χρησιμοποιώντας την εντολή **Instantiate** που παρέχει το Unity, μπορούμε να εμφανίσουμε οποιοδήποτε αντικείμενο ορίσουμε μέσω κώδικα σε GameObject ή Transform μορφή σε με μια οποιαδήποτε θέση θέλουμε με οποιαδήποτε περιστροφή επιθυμούμε. Για παράδειγμα, για την εμφάνιση του τόξου πάνω από τις σφαίρες έχουμε τον πιο κάτω κώδικα (Όπου pos είναι το position του τόξου (πάνω ή κάτω από μια σφαίρα) και rot το rotation που είναι το ίδιο με το αρχικό τόξο: `GameObject arrow = Instantiate(PointingArrow, pos, rot);`

Δυσκολία εύρεσης αντικειμένων – GameObject μέσα από script. Μετά από έρευνα, έχουμε ανακαλύψει ότι μπορούμε να βρούμε αντικείμενα που βρίσκονται στο Scene μας με χρήση Tags τα οποία μπορούμε να προσθέσουμε πάνω σε ένα οποιοδήποτε αντικείμενο/prefab που έχουμε στη σκηνή μας (A – Σχήμα 2.22). Χρησιμοποιώντας την εντολή “`GameObject.FindGameObjectsWithTag(*TAG*)`” [2] μπορούμε να βρούμε ό,τι αντικείμενο θέλουμε. Επίσης με την απλή δημιουργία public μεταβλητών τύπου GameObject ή οτιδήποτε άλλου τύπου, θα εμφανίζονται δυναμικά στον Inspector και θα μπορούμε πολύ απλά να τις ορίσουμε στο αντικείμενο που βρίσκετε το Script μας (B – Σχήμα 2.22).



Σχήμα 2.22

Κεφάλαιο 3.

Κύρια Διεπαφή Προγράμματος

3.1 Περιγραφή Τρόπου Υλοποίησης	23
3.2 Σχεδιασμός και Υλοποίηση	24
3.3 Προβλήματα και Διευκρινήσεις	27

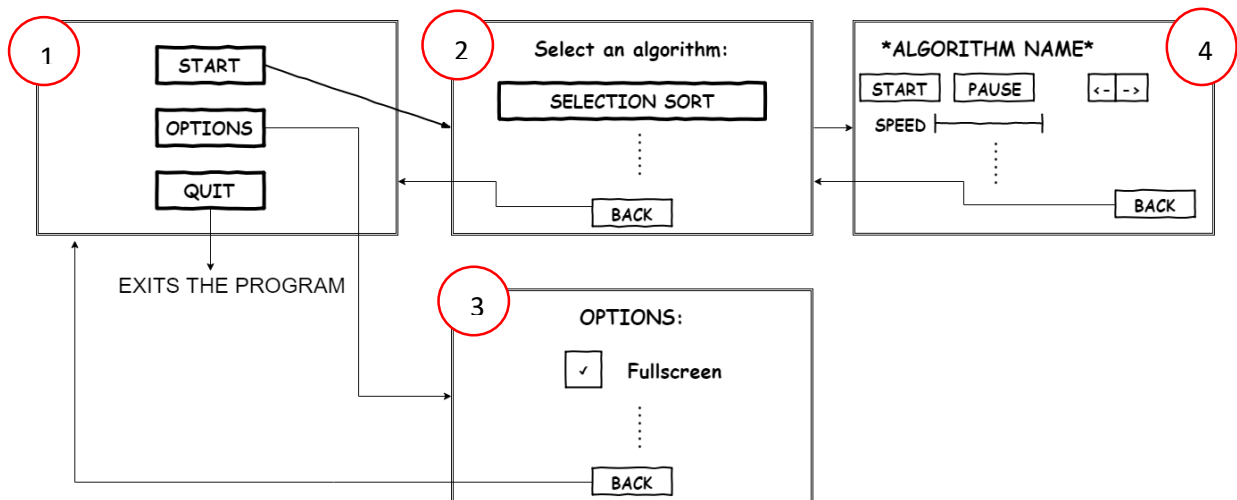
3.1 Περιγραφή Τρόπου Υλοποίησης

Για την δημιουργία των διεπαφών του προγράμματός μας, το εργαλείο Unity προσφέρει πολλά σύνθετα εργαλεία που μας δίνουν την δυνατότητα εύκολης και γρήγορης υλοποίησης. Αρχικά, η πρώτη μορφή των διεπαφών σχεδιαζόταν στο χαρτί και δέχονταν πολλές αλλαγές μέχρι να βρεθεί η βέλτιστη. Όταν βρισκόμαστε σε ικανοποιητικό σημείο και νιώθουμε ευχαριστημένοι με την εικόνα της διεπαφής και το ότι θα έχει σωστή και εύχρηστη λειτουργικότητα, προχωρούμε στην σχεδίαση στο Unity. Εκεί με την χρήση έτοιμων αντικειμένων διεπαφής (UI), σχεδιάζουμε ότι έχουμε σκεφτεί με την κατάλληλη εμφάνιση. Τέλος εφαρμόζουμε λειτουργικότητα πάνω στα αντικείμενα διεπαφής με χρήση των έτοιμων Component ή μέσω κώδικα.

Μετά την πρώτη υλοποίηση, οι διεπαφές περνούσαν από κριτική συγκεκριμένων ατόμων όπου και παίρναμε σχόλια για αλλαγές και βελτιώσεις. Οι αλλαγές εφαρμόζονταν εκείνη την ώρα ή μέσα στην ίδια ημέρα.

3.2 Σχεδιασμός και Υλοποίηση

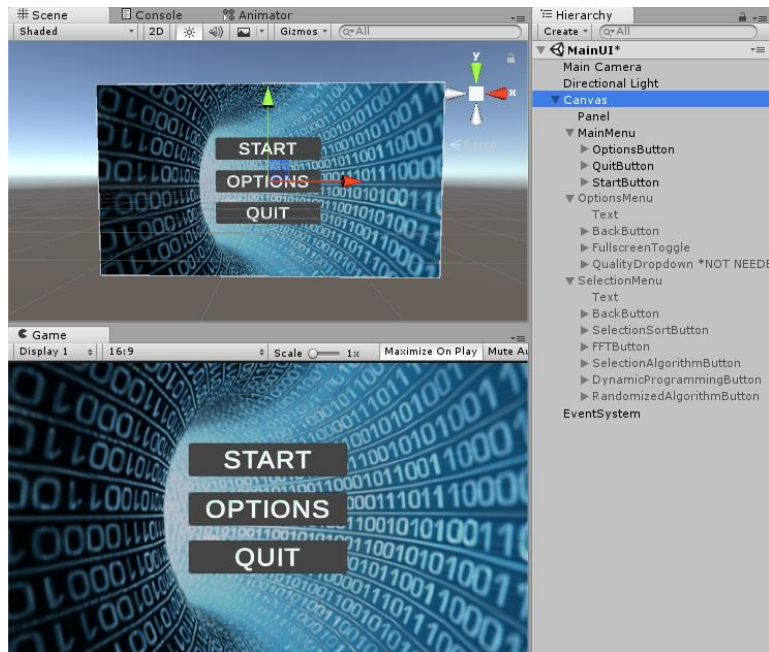
Ξεκινούμε αρχικά σχεδιάζοντας ένα προσχέδιο της διεπαφής και πώς θα συνδέετε γενικά με ολόκληρο το πρόγραμμα. Πιο κάτω στο Σχήμα 3.1 φαίνεται τη αρχική μας σκέψη για την υλοποίηση αυτής της διεπαφής.



Αρχικό προσχέδιο Κύριας Διεπαφής - Σχήμα 3.1

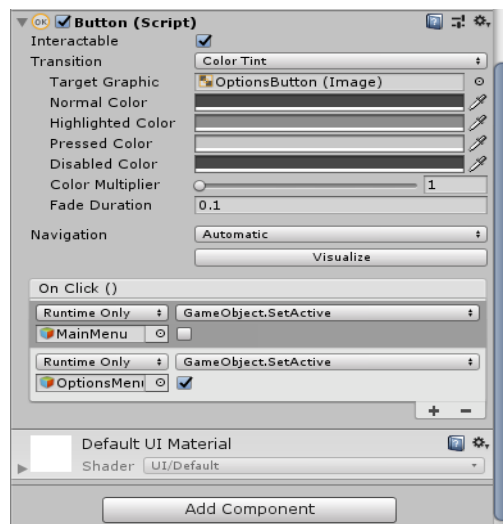
Τρέχοντας το πρόγραμμα, ο χρήστης θα καλωσορίζεται από την αρχική οθόνη (1). Εκεί θα έχει την δυνατότητα να επιλέξει το “START” κουμπί που θα τον παίρνει στην διεπαφή όπου και θα φαίνονται οι διάφοροι αλγόριθμοι που θα μπορεί να τρέξει (2). Οποιοδήποτε επιλέξει θα μεταφέρεται ευθέως στην διεπαφή του αντίστοιχου αλγορίθμου όπου και θα προσφέρονται διάφορες επιλογές για την λειτουργία και έλεγχο κάθε βήματός του (4). Τα κουμπιά “BACK” θα σε κατευθύνουν πάντα στην προηγούμενη οθόνη που βρισκόσουν. Επιπρόσθετα ο χρήστης θα έχει την δυνατότητα να επιλέξει το “OPTIONS” κουμπί από την 1^η διεπαφή το οποίο θα τον παίρνει σε νέα διεπαφή όπου και θα μπορεί να επιλέξει και να τροποποιήσει διάφορες επιλογές για τον τρόπο λειτουργίας του προγράμματος όπως το Fullscreen κλπ (3).

Ακολουθώντας το παραπάνω πρότυπο προχωρούμε στην υλοποίησή του στο εργαλείο Unity. Κατά την πορεία υλοποίησης, πάντα υπάρχει πιθανότητα να υπάρξουν αλλαγές, προσθήκες και ίσως και αφαιρέσεις αντικειμένων από την αρχική ιδέα. Έτσι οι λειτουργίες προστίθενται δυναμικά καθώς υλοποιούμε το πρόγραμμα μέχρι και το τελευταίο λεπτό.

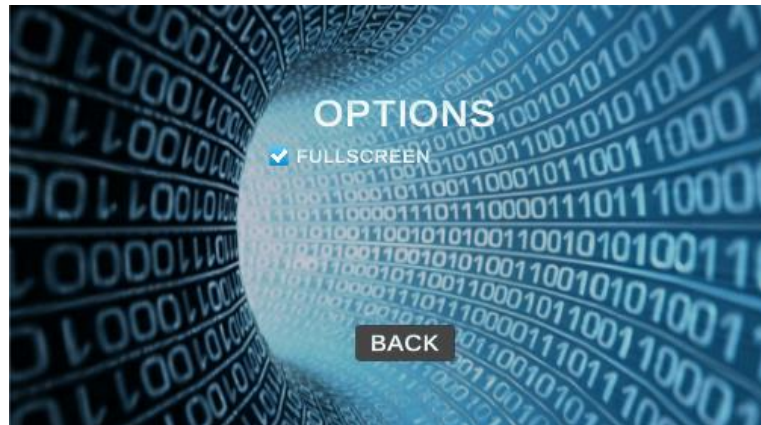


Αρχική Κύρια Διεπαφή στο Unity - Σχήμα 3.2

Για τη διατήρηση της απλότητας και ευχρηστίας, η αρχική διεπαφή έχει υλοποιηθεί όπως έχει σχεδιαστεί στο χαρτί (Σχήμα 3.2). Δεν έχει παρουσιάσει καμία αλλαγή στην πορεία. Αποτελείται από τα τρία κουμπιά που έχουν επεξηγηθεί προηγουμένως. Η εικόνα που βρίσκεται στο φόντο σε παραπέμπει αμέσως στους αλγορίθμους [21]. Τα κουμπιά (**UI**→ **Button**) προσφέρουν διάφορες, εύκολες στη χρήση, επιλογές που τα καθιστούν με λειτουργία με τα components που περιέχουν. Στο καθένα από αυτά μπορούμε εύκολα να ορίσουμε τα χρώματα τους και τα περιεχόμενά τους καθώς επίσης και τις ενέργειες που θα εκτελούν με την επιλογή τους. Στο πιο κάτω παράδειγμα απενεργοποιούμε το Κυρίως μενού (που είναι ένα απλό **Canvas**) και εμφανίζουμε το μενού με τις επιλογές στην ίδια σκηνή (Σχήμα 3.3).

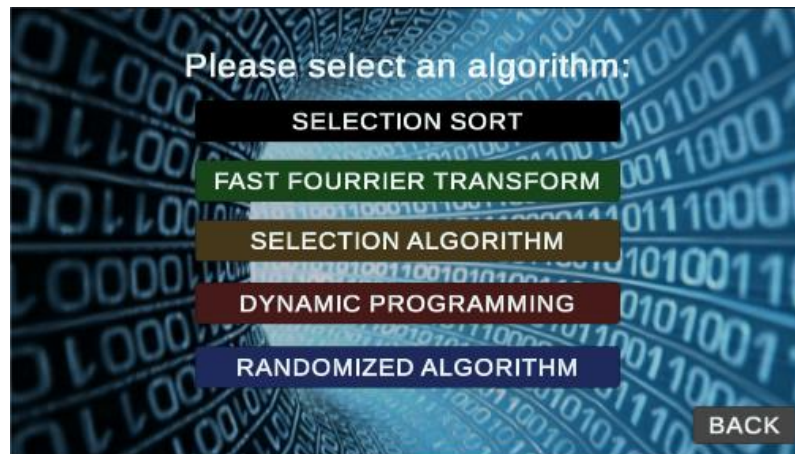


Παράδειγμα χρήσης ενός UI->Button - Σχήμα 3.3



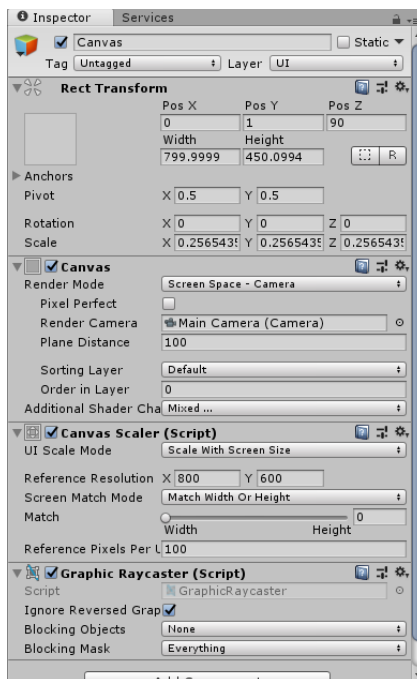
Options Canvas of Main UI - Σχήμα 3.4

Με την επιλογή του “OPTIONS” κουμπιού από την αρχική διεπαφή (Σχήμα 3.2), εμφανίζεται η οθόνη με τις επιλογές (Σχήμα 3.4). Εδώ ο χρήστης μπορεί προς το παρόν να ενεργοποιήσει ή να απενεργοποιήσει την επιλογή της πλήρης οθόνης (Fullscreen). Μελλοντικά μπορούν να προστεθούν πολύ περισσότερες επιλογές όπως επιλογή θέματος, ρύθμιση ήχων (αν ποτέ εφαρμοστούν ήχοι και ηχητικά εφέ στους αλγορίθμους) και άλλα.



Algotihm Selection canvas of Main UI - Σχήμα 3.5

Αν ο χρήστης επιλέξει την επιλογή “START” (Σχήμα 3.2), όπως έχουμε προαναφέρει, εμφανίζονται οι επιλογές των αλγορίθμων. Στο Σχήμα 3.5 φαίνεται πλήρως υλοποιημένη η διεπαφή αυτή με όλους τους αλγορίθμους οι οποίοι έχουν υλοποιηθεί. Με την επιλογή ενός από τους αλγορίθμους αυτούς, ο χρήστης κατευθύνεται προς την **διαφορετική σκηνή** που έχει υλοποιηθεί ο αντίστοιχος αλγόριθμος. Αυτό επιτυγχάνεται με χρήση απλής συνάρτησης σε script που βρίσκεται ως Component του καμβά που βρίσκεται το πιο πάνω μενού.



UI→Canvas GameObject – Inspector View - Σχήμα 3.6

Τέλος αξίζει να σημειωθεί ότι κάθε μενού βρίσκεται σε ένα καμβά (Canvas) το οποίο μας προσφέρει διάφορες επιλογές (Σχήμα). Στην διπλωματική αυτή επιλέχθηκε να εφαρμοστεί Render mode με χρήση της κάμερας που βλέπει την σκηνή μας. Ως αποτέλεσμα δεν θα μετακινείται καμία διεπαφή εκτός των ορίων της κάμερας που παρουσιάζει την σκηνή στο χρήστη (Game View). Επίσης σημαντικό είναι η επιλογή του Scale mode να βρίσκεται στο “Scale with screen size” για να μην αλλάζουν τα μεγέθη των αντικειμένων καθώς μικραίνουμε ή μεγαλώνουμε το παράθυρο που τρέχουμε το πρόγραμμά μας. Αυτές οι επιλογές εφαρμόζονται παντού σε όλες τις διεπαφές.

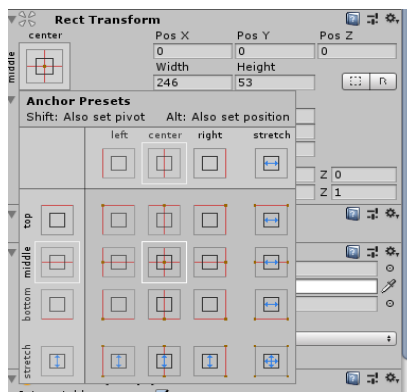
3.3 Προβλήματα και διευκρινήσεις

Παρουσιάστηκε ένα μικρό πρόβλημα κατά την φόρτωση διαφορετικών σκηνών με την επιλογή των κουμπιών και εκτέλεση κώδικα. Παρατηρούσαμε αργοπορία στην φόρτωση και λειτουργία των αλγορίθμων όταν πηγαίναμε στις σκηνές τους. Αυτό οφείλονταν στο ότι υπήρχαν ήδη ανοικτές και φορτωμένες σκηνές από πριν οι οποίες βαρυφορτώνανε τους πόρους μας. Έτσι με τη χρήση ενός απλού mode που προσφέρει το SceneManager του Unity, το Single Load Scene mode, έχει επιλυθεί το πρόβλημά μας. Το mode αυτό κλείνει όλες τις ήδη φορτωμένες σκηνές και ανοίγει αυτή που θέλουμε. Για παράδειγμα, για τον

αλγόριθμο Selection Sort, ο κώδικας φόρτωσης της σκηνής του είναι ο εξής:
`SceneManager.LoadScene("SelectionSort", LoadSceneMode.Single); [2]`

Κατά την υλοποίηση παρουσιάστηκε πρόβλημα στην προσαρμογή των αντικειμένων κατά την αλλαγή μεγέθους του παραθύρου εκτέλεσης του προγράμματος. Αυτό έχει επιλυθεί με τον τρόπο που έχουμε πει στην περιγραφή του σχήματος 3.6.

Τα αντικείμενα ενός καμβά μετακινούνταν σε μη ευδιάκριτα σημεία της οθόνης. Με απλή χρήση του **Rect Transform** component που υπάρχει σε κάθε αντικείμενο που δημιουργείτε μέσα σε ένα καμβά έχει επιλυθεί το πρόβλημα αυτό (Σχήμα 3.7).



Rect Transform Component in Canvas childrens - Σχήμα 3.7

Η διπλωματική αυτή εργασία αφού περιέχει διεπαφές χρήστη οι οποίες έχουν διάφορες μορφές και μεγέθη θα πρέπει να έχουν μια κοινή σύμβαση. Κάθε διεπαφή έχει διαφορετικά χαρακτηριστικά στις εμφανίσεις αντικειμένων για τους διάφορους αλγορίθμους. Συνεπώς οφείλουν να χρησιμοποιούν κοινό Resolution και Aspect Ratio για την εμφάνιση τους στο τελικό εκτελέσιμο. Ορισμένα μέρη αυτών των διεπαφών μπορούν να προσαρμοστούν σε διάφορα Resolutions αλλά το ιδανικότερο είναι να έχουμε ένα συγκεκριμένο. Συγκεκριμένα χρησιμοποιούμε τα Resolution **1080x1920p** και **720x1280p** δηλαδή το aspect ratio **16:9**. Σαφώς χρησιμοποιούμε το πιο διαδεδομένο aspect ratio για να έχουμε πρόσβαση στις περισσότερες οθόνες.

Όλα τα οποία έχουμε αναφέρει σε αυτό το κεφάλαιο, θα είναι εφαρμοσμένα σε ολόκληρο το πρόγραμμα. Όλες οι διεπαφές σε όλους του αλγορίθμους θα λειτουργούν με ομοιότυπο τρόπο με επαναχρησιμοποίηση κουμπιών και άλλων αντικειμένων που έχουν ήδη υλοποιηθεί.

Κεφάλαιο 4.

Αλγόριθμος Ταχύ Μετασχηματισμού Fourier

4.1 Εισαγωγή	29
4.2 Περιγραφή Αλγορίθμου FFT	30
4.2.1 Αναδρομικός FFT	30
4.2.2 Μετατροπή Αναδρομικού FFT σε Επαναληπτικού	34
4.3 Λεπτομέρειες Υλοποίησης	35
4.3.1 Σχεδιασμός Πρωτοτύπου	35
4.3.2 Υλοποίηση Διεπαφής	38
4.3.3 Αναπαράσταση Αλγορίθμου	39
4.4 Δυσκολίες/Προβλήματα και Τρόπος Επίλυσης	45

4.1 Εισαγωγή

Ο αλγόριθμος ταχύ μετασχηματισμού Fourier (Fast Fourier Transform) (**FFT**) είναι ένας αλγόριθμος τύπου «Διαίρει και Βασίλευε» [4,5]. Έχει αρκετές εφαρμογές σε διάφορους κλάδους όπως στην οπτική, ακουστική, κβαντική φυσική, επεξεργασία εικόνας, συμπίεση δεδομένων κλπ [4]. Αποτελεί μία από τις καλύτερες υπολογιστικές υλοποιήσεις του 20^{ου} αιώνα και συνεπώς είναι ένα από τους πιο χρήσιμους αλγορίθμους.

Βασίζεται πάνω στον Διακριτό του αντίστοιχο (DFT – Discrete Fourier Transform). Ο DFT υλοποιεί την αποτίμηση ενός πολυωνύμου P βαθμού β με τις τιμές των $(\beta+1)$ -ιοστών ρίζων της μονάδας. Επεκτατικά ο FFT υπολογίζει πιο γρήγορα το αποτέλεσμα του DFT καθώς είναι τύπου «Διαίρει και Βασίλευε». Έτσι πετυχαίνει χρονική πολυπλοκότητα $T(n) = \Theta(n \log n)$, για ένα πολυώνυμο βαθμού $n-1$ για όλες τις $(2n)$ -ιοστές ρίζες της μονάδας.

Δομικά ο FFT λαμβάνει ως είσοδο ένα πολυώνυμο n-οστού βαθμού το οποίο βρίσκεται σε παράσταση με συντελεστές και επιστρέφει πίσω την αναπαράσταση του πολυωνύμου αυτού με τιμές σε σημεία. Δηλαδή για παράδειγμα έστω η αναπαράσταση σε συντελεστές του A(x) πολυωνύμου: $A(x) = a_0 + a_1 * x + a_2 * x^2 + \dots + a_{n-1} * x^{n-1}$. Η παράστασή του με τιμές σε σημεία θα είναι η εξής: $A(x) : (x_0, y_0), \dots, (x_{n-1}, y_{n-1})$. Αυτό βοηθά στον πολλαπλασιασμό πολυωνύμων (πχ: $A(x) * B(x)$) καθώς οι πράξεις μειώνονται από $O(n^2)$ που απαιτούνται αν τα πολυώνυμα αναπαριστώνται με συντελεστές σε $O(n)$ όταν βρίσκονται στη μορφή τιμές σε σημεία.

Αποτελεσματικά αποτελεί ένα από τους λίγο πιο δύσκολους στην κατανόηση, αναδρομικούς αλγορίθμους τόσο στο μάθημα ΕΠΛ236 όσο και για αυτούς που τον μαθαίνουν για πρώτη φορά. Οι φοιτητές δυσκολεύονται λόγω του ότι στην υλοποίηση του περιέχει μαθηματικούς υπολογισμούς και πολλαπλές αναδρομές. Απαιτεί ένα βασικό μαθηματικό υπόβαθρο στις μιγαδικές ρίζες και πολλαπλασιασμούς τους καθώς επίσης και σε άλλα βασικά μαθηματικά.

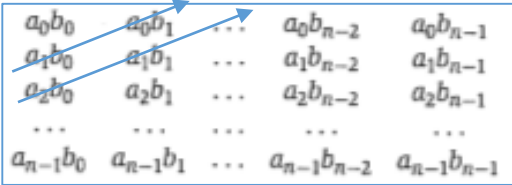
4.2 Περιγραφή Αλγορίθμου FFT

4.2.1 Αναδρομικός FFT

Το βασικό πρόβλημα το οποίο βοηθά στην επίλυσή του ο FFT είναι το πρόβλημα συνέλιξης (Convolution) διανυσμάτων. Έστω ότι έχουμε δύο διανύσματα $a = \langle a_0, a_1, \dots, a_{n-1} \rangle$ και $b = \langle b_0, b_1, \dots, b_{n-1} \rangle$ ίσου μήκους (n), η συνέλιξή τους είναι ένα διάνυσμα $c = a * b = \langle c_0, c_1, \dots, c_{2n-2} \rangle$ (μήκους $2n-1$). Γενικά καταλήγουμε στη πιο κάτω εξίσωση (Σχήμα 4.1)[5].

$$\sum_{\substack{0 \leq i+j=k \\ i, j < n}} a_i b_j$$

In other words,

$$a * b = (a_0 b_0, a_0 b_1 + a_1 b_0, a_0 b_2 + a_1 b_1 + a_2 b_0, \dots, a_{n-2} b_{n-1} + a_{n-1} b_{n-2}, a_{n-1} b_{n-1}).$$


$a_0 b_0$	$a_0 b_1$	...	$a_0 b_{n-2}$	$a_0 b_{n-1}$
$a_1 b_0$	$a_1 b_1$	...	$a_1 b_{n-2}$	$a_1 b_{n-1}$
$a_2 b_0$	$a_2 b_1$	...	$a_2 b_{n-2}$	$a_2 b_{n-1}$
...	...	...	...	...
$a_{n-1} b_0$	$a_{n-1} b_1$	...	$a_{n-1} b_{n-2}$	$a_{n-1} b_{n-1}$

Σχήμα 4.1

Ένας άλλος τρόπος να δούμε την συνέλιξη αυτή είναι με τη χρήση του δεξιά πίνακα, προσθέτοντας όλους τους πολλαπλασιασμούς των διαγωνίων όπως φαίνονται με τα βέλη. Δεν είναι μόνο ο ορισμός της συνέλιξης που είναι δύσκολος στη κατανόηση αρχικά, αλλά και η δυσκολία ανάπτυξης κινήτρου για μάθηση. Συνεπώς θα προσπαθήσουμε να το απλοποιήσουμε όσο πιο πολύ γίνεται αργότερα στην υλοποίηση μας.

Μια χρήση της συνέλιξης βρίσκεται στον πολλαπλασιασμό πολυωνύμων όπως έχουμε μάθει στο γυμνάσιο. Έστω ένα πολυώνυμο $A(x) = a_0 + a_1 * x + a_2 * x^2 + \dots + a_{n-1} * x^{n-1}$ (βαθμού n) μπορεί να αναπαρασταθεί και από διάνυσμα συντελεστών (coefficients) $\langle a_0, a_1, \dots, a_{n-1} \rangle$. Τώρα έστω ότι υπάρχουν δύο πολυώνυμα $A(x)$ και $B(x)$ βαθμού n . Ο υπολογισμός του γινομένου τους έχει ως αποτέλεσμα το $C(x) = A * B = c_0 + c_1 * x + \dots + c_{2n-2} * x^{2n-2}$. Έχουμε την πιο κάτω γενική εξίσωση για τους συντελεστές του C :

$$C_k = \sum_{(i,j): i+j=k} a_i b_j$$

Συνεπώς το διάνυσμα των συντελεστών του γινομένου $C(x)$ είναι η συνέλιξη των διανυσμάτων των συντελεστών του A και B . Παρατηρούμε ότι ο υπολογισμός του πολλαπλασιασμού δύο πολυωνύμων n βαθμού απαιτεί $\Theta(n^2)$ αριθμητικές πράξεις. Μια μέθοδος που μειώνει σε μεγάλο βαθμό τον αριθμό των πράξεων αυτών σε $O(n \log n)$ περιλαμβάνει την τεχνική του FFT.

Για τον υπολογισμό της συνέλιξης δύο πολυωνύμων A και B όπως έχουμε προαναφέρει εφαρμόζουμε τον πιο κάτω αλγόριθμο:

1. Επιλέγουμε $2n$ σημεία, έστω $\{x_0, x_1, \dots, x_{2n-1}\}$ και υπολογίζουμε τις τιμές $A(x_j)$ και $B(x_j)$ για όλα τα $j = 0, 1, 2, \dots, 2n-1$.
2. Υπολογίζουμε εύκολα το $C(x_j)$ για κάθε j ως $C(x_j) = A(x_j) * B(x_j)$
3. Τέλος ανακτούμε όλους τους συντελεστές του $C \{x_0, x_1, \dots, x_{2n-1}\}$ από τις τιμές που έχουμε υπολογίσει στο βήμα 2. Εδώ χρησιμοποιούμε πολυωνυμική παρεμβολή (polynomial interpolation) όπου για κάθε πολυώνυμο βαθμού d , μπορεί να ξανακατασκευαστεί από ένα σύνολο με $d+1$ και παραπάνω σημεία.

Συνεπώς έχουμε να κάνουμε $2n$ αποτιμήσεις τόσο του A όσο και του B . Το βήμα 2. Περιλαμβάνει $O(n)$ υπολογισμούς επειδή έχουμε απλό πολλαπλασιασμό αντίστοιχων στοιχείων. Το βήμα 1. και 3. όμως για την αποτίμηση μονάχα μιας τιμής χρειάζονται $O(n)$ υπολογισμούς. Αφού έχουμε $2n$ αποτιμήσεις για το καθένα από τα δύο πολυώνυμα, θα έχουμε σύνολο $\geq 2n * 2 * O(n) = O(n^2)$ υπολογισμούς το οποίο δεν είναι επιθυμητό.

Η κύρια ιδέα εδώ είναι να βρούμε ένα τρόπο που να συνδέονται αυτές οι $2n$ τιμές ούτως ώστε αποτιμώντας τα A και B , να διαμοιράζονται αυτές οι αποτιμήσεις σε διάφορες άλλες.

Σε αυτό το σημείο θα πρέπει να αναφερθούμε για τις πραγματικές ρίζες και τις πολικές συντεταγμένες ενός χώρου. Αυτοί οι αριθμοί $\{(\alpha, \beta) \text{ ή } (\alpha + \beta i) \text{ όπου } i^2 = -1\}$ περιέχουν ένα πραγματικό (α) και ένα φανταστικό μέρος (β). Για ένα θετικό ακέραιο k η πολωνυμική εξίσωση $x^k = 1$ έχει k διαφορετικές μιγαδικές ρίζες/λύσεις που είναι εύκολο να βρεθούν.

$$W_{j,k} = e^{2\pi i j/k}, j = 0, 1, 2 \dots, k-1$$

$$e^{\pi i} = -1 \Rightarrow e^{2\pi i} = 1$$

$$e^{vi} = \cos(v) + i * \sin(v)$$

Στην υλοποίηση μας εμείς θα παρουσιάσουμε μόνο το Βήμα 1. του αλγορίθμου συνέλιξης που αναφέραμε πιο πριν, δηλαδή την αποτίμηση ενός πολωνύμου. Για τις $2n$ αποτιμήσεις $\{x_0, x_1, \dots, x_{2n-1}\}$ επιλέγουμε τις $2n$ -ισστές μιγαδικές ρίζες τις μονάδας.

Έτσι έχουμε τον πιο κάτω αλγόριθμο τύπου «διαίρει και βασίλευε» (**FFT**):

1. Ορίζουμε 2 πολυώνυμα βαθμού $n/2 - 1$ {**ΦΑΣΗ ΔΙΑΙΡΕΣΕ**}
 $A_{\text{even}}(x) = a_0 + a_2 * x + a_4 * x^2 + \dots + a_{n-2} * x^{(n-2)/2}$ (ζυγοί συντελεστές του A)
 $A_{\text{odd}}(x) = a_1 + a_3 * x + a_5 * x^2 + \dots + a_{n-1} * x^{(n-2)/2}$ (περιττοί συντελεστές του A)
2. Βλέπουμε ότι: $A(x) = A_{\text{even}}(x^2) + x * A_{\text{odd}}(x^2)$
3. Έτσι υπολογίζουμε αναδρομικά τις τιμές του A_{even} και A_{odd} για τις n -οστές ρίζες της μονάδας (χρόνος: $2 * T(n/2)$). {**ΦΑΣΗ ΚΑΤΕΚΤΗΣΕ**} Απομένει να υπολογίσουμε τις τιμές $A(W_{j,2n}), 0 \leq j \leq 2n-1$ από τις τιμές $A_{\text{even}}(W_{j,n})$ και $A_{\text{odd}}(W_{j,n}), 0 \leq j \leq n-1$
4. Για οποιοδήποτε $j, 0 \leq j \leq 2n-1$: {**ΦΑΣΗ ΣΥΝΔΥΑΣΕ**}

$$A(W_{j,2n}) = A_{\text{even}}(W_{j,2n}^2) + W_{j,2n} * A_{\text{odd}}(W_{j,2n}^2)$$

Επειδή χρησιμοποιούμε τις ιδιότητες των μιγαδικών αριθμών, αυτοί οι υπολογισμοί γίνονται σε γραμμικό χρόνο ($\Theta(n)$).

$$W_{j,2n}^2 = (e^{2\pi i j / 2n})^2 = e^{2\pi i j / n} = W_{j,n}$$

Συνεπώς τα $A_{\text{even}}(W_{j,2n}^2)$ και $A_{\text{odd}}(W_{j,2n}^2)$ έχουν ήδη υπολογιστεί. Συνολικός χρόνος εκτέλεσης: $\Theta(n \log n)$

Η βασική απλή εμφάνιση του αλγορίθμου σε ψευδοκώδικα είναι στην πιο κάτω εικόνα (Σχήμα 4.2) και η τελική υλοποιημένη μορφή στη C# του αναδρομικού στο Σχήμα 4.3.

```
// Simple FFT pseudocode
FFT(A, n){
    if n==1 return (A[0], A[0]); // Terminal Case
    Split A to Aeven, Aodd;
    FFT(Aeven, n/2);
    FFT(Aodd, n/2);
    for every j from j=0 to j=2*n-1 Do{
        A(Wj,n) = Aeven(Wj,n/2) + Wj,n * Aodd(Wj,n/2);
    }
    return A(Wj,n) of every j;
}
```

Ψευδοκώδικας FFT - Σχήμα 4.2

```
1 // The recursive version of FFT
2 public Complex[] recursiveFFT(Complex[] P, int n)
3 {
4     if (n == 1) return new Complex[] { P[0] };
5     if (n % 2 != 0) Debug.Log("Error! Length of the array is not in the power of two!");
6
7     // Split P to Peven and Podd
8     Complex[] Peven = new Complex[n / 2];
9     Complex[] Podd = new Complex[n / 2];
10    int even = 0, odd = 0;
11    for (int i = 0; i < n; i++)
12    {
13        if (i % 2 == 0)
14        {
15            Peven[even] = P[i];
16            even++;
17        }
18        else
19        {
20            Podd[odd] = P[i];
21            odd++;
22        }
23    }
24
25    // Find the results of Peven and Podd
26    Complex[] p1 = recursiveFFT(Peven, n / 2);
27    Complex[] p2 = recursiveFFT(Podd, n / 2);
28
29    // Combine even + odd to produce the final result
30    Complex[] Pnew = new Complex[n];
31    for (int k = 0; k < n / 2; k++)
32    {
33        float kth = 2 * k * Mathf.PI / n;
34        Complex wk = new Complex(Mathf.Cos(kth), Mathf.Sin(kth));
35        Pnew[k] = p1[k] + (wk * (p2[k]));
36        Pnew[k + n / 2] = p1[k] - (wk * (p2[k]));
37    }
38    return Pnew;
39 }
```

Αναδρομικός FFT στην C# - Σχήμα 4.3

4.2.2 Μετατροπή Αναδρομικού FFT σε Επαναληπτικού

Προτού ξεκινήσουμε την υλοποίηση πρέπει να σημειωθεί ότι όποτε έχουμε να κάνουμε με αλγορίθμους που περιέχουν αναδρομή, για να αναπαρασταθούν οπτικά, θα πρέπει να μετατραπούν στην επαναληπτική τους μορφή. Αυτό γίνεται επειδή μετατρέποντας τους από αναδρομικούς σε επαναληπτικούς, διατηρούμε δομές δεδομένων όπως στοίβες με τις οποίες γνωρίζουμε ανά πάσα στιγμή τις αλλαγές των δεδομένων και τα δεδομένα που βγαίνουν ως αποτέλεσμα. Έτσι, με το να τα έχουμε αποθηκευμένα, μπορούμε να τα παρουσιάσουμε οπτικά όποτε επιθυμούμε.

```
1  /* IterativeFFT function */
2  public Complex[] IterativeFFT(Complex[] P)
3  {
4      Stack<Complex[]> polynomials = new Stack<Complex[]>();
5      Stack<Complex[]> results = new Stack<Complex[]>();
6
7      // Insert initial array into stack
8      polynomials.Push(P);
9
10     while(polynomials.Count != 0)
11     {
12         // Pop the next polynomial
13         Complex[] currentP = polynomials.Pop();
14
15         // Terminal Case
16         if(currentP.Length == 1)
17         {
18             // Return {P[0],P[0]}
19             results.Push(new Complex[] {currentP[0],currentP[0]});
20
21             // Remove the same size pairs (2) of polynomial results
22             while (results.Count > 1)
23             {
24                 // Even
25                 Complex[] r1 = results.Pop();
26
27                 // if the next element in the result stack is not
28                 // the same size as the one popped
29                 // (Peven.Length == Podd.Length) then wait until
30                 // its partner gets pushed in the results
31                 if(results.Peek().Length != r1.Length)
32                 {
33                     results.Push(r1);
34                     break;
35                 }
36
37                 // Odd
38                 Complex[] r2 = results.Pop();
39
40                 // calculate the new array for the 2*N roots
41                 // using the popped results of the previous
42                 // fft(Peven) and fft(Podd)
43                 int newN = r1.Length * 2;
44                 Complex[] newP = new Complex[newN];
45                 for (int i = 0; i < newN / 2; i++)
46                 {
47                     float kth = 2 * i * Mathf.PI / newN;
48                     Complex wk = new Complex(Mathf.Cos(kth), Mathf.Sin(kth));
49                     newP[i] = r1[i] + (wk * (r2[i]));
50                     newP[i + newN / 2] = r1[i] - (wk * (r2[i]));
51                 }
52                 results.Push(newP);
53             }
54
55             // Skip the split calculations
56             continue;
57         }
58
59         // Split current array into its Even and Odd positioned numbers
60         Complex[] Peven = new Complex[currentP.Length / 2];
61         Complex[] Podd = new Complex[currentP.Length / 2];
62         int even = 0, odd = 0;
63         for(int i=0; i<currentP.Length; i++)
64         {
65             if (i % 2 == 0)
66                 Peven[even++] = currentP[i];
67             else
68                 Podd[odd++] = currentP[i];
69         }
70
71         // Add Podd and Peven to stack in this order (since stack is FIFO)
72         polynomials.Push(Podd);
73         polynomials.Push(Peven);
74     }
75
76     // The final result in the results stack is
77     // the result of the FFT
78     return results.Pop();
79 }
```

Επαναληπτικός FFT στη C# - Σχήμα 4.4

Χρησιμοποιώντας δύο στοίβες “polynomials” και “results” έχουμε πετύχει την μετατροπή αυτή (Σχήμα 4.4). Μετά από πολλές δοκιμές και λάθη στην μετατροπή του αλγορίθμου, έχουμε καταλήξει εδώ. Η περιγραφή των μετατροπών που έχουν γίνει σε σχέση με τον κώδικα της αναδρομής φαίνονται στο σχήμα. Επιπλέον ανάλυσή του θα γίνει στην περιγραφή υλοποίησης αργότερα.

Το αντικείμενο τύπου “Complex” αποτελείται από δύο float αριθμούς (real και imaginary) και περιγράφει ένα μιγαδικό αριθμό. Κάνοντας μαθηματικές πράξεις (+,-,/,*) με αυτά τα αντικείμενα, εφαρμόζονται αυτόματα και οι συμβάσεις των μαθηματικών πράξεων για τους μιγαδικούς (βλέπε παράρτημα Γ – **Complex.cs**).

4.3 Λεπτομέρειες Υλοποίησης

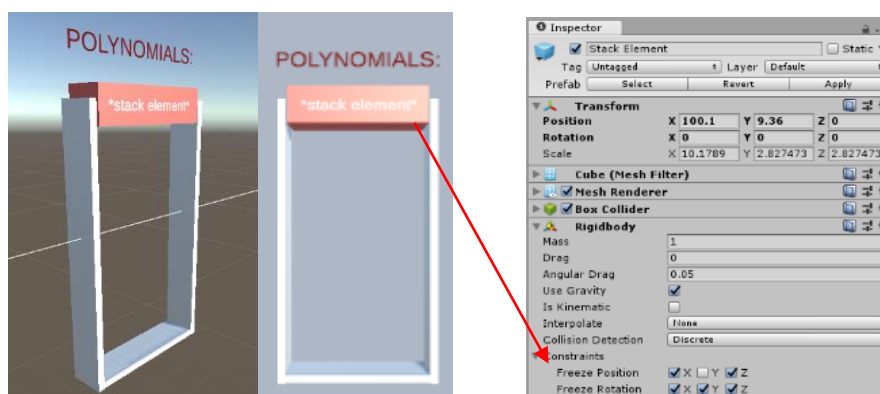
4.3.1 Σχεδιασμός Πρωτοτύπου

Μελετώντας τον επαναληπτικό κώδικα του σχήματος 4.4, έχουν λάβει μέρος διάφορες ιδέες και σχέδια για το πώς θα παρουσιαστεί με την χρήση του Unity.

Αρχικά έχουν γίνει ορισμένα προσχέδια όπως φαίνονται στην αρχή του **παραρτήματος Γ**. Αυτά τα προσχέδια τα έχουμε επεκτείνει στο Unity σταδιακά δημιουργώντας ένα-ένα τα αντικείμενα. Προτού υλοποιηθεί ο αλγόριθμος, έχουμε σκεφτεί πως θα είναι η εμφάνιση των αντικειμένων, τι 3D αντικείμενα θα χρησιμοποιήσουμε και το πώς θα αλληλοεπιδρούν μεταξύ τους.

Για τις δύο στοίβες (“polynomials” και “results”) θεωρήσαμε πιο ιδανικό να τις αναπαραστήσουμε σε τρισδιάστατη μορφή. Έτσι έχουμε υλοποιήσει ένα καινούριο 3D GameObject prefab “3DStack” που έχει το σχήμα μιας στοίβας και ταυτόχρονα εκτελεί όλες τις ιδιότητές της όπως Push και Pop. Καθώς εμφανίζονται τα αποτελέσματα από την συγχώνευση των Even και Odd πινάκων, θα εμφανίζονται και θα μπαίνουν στην στοίβα “Results” με μια κάθετη κίνηση.

Το ίδιο θα συμβαίνει και στην στοίβα “Polynomials” όπου θα μπαίνουν τα πολυώνυμα που θα χρησιμοποιηθούν στην επόμενη αναδρομική εκτέλεση του FFT. Η στοίβα αυτή έχει την τελική εμφάνιση του Σχήματος 4.5.

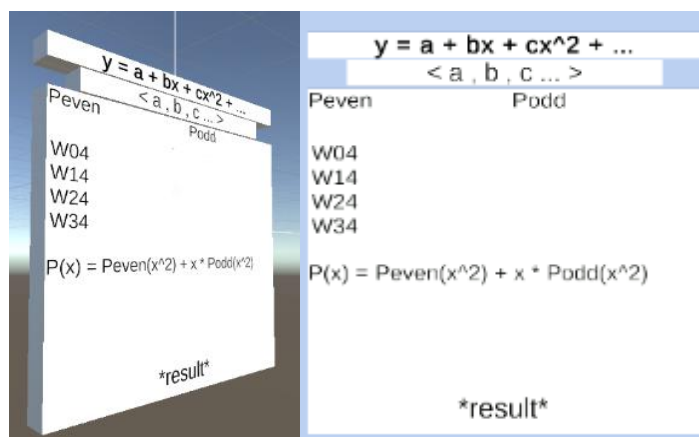


3D Stack και Stack Element GameObjects - Σχήμα 4.5

Στο παράδειγμα του σχήματος βλέπουμε το GameObject το οποίο έχουμε δημιουργήσει και ελέγχεται από το συνημμένο πάνω του script **StackObjectManager.cs** (Παράρτημα Γ). Με την συνάρτηση Push, το αντικείμενο αυτό εμφανίζει (Instantiate) ένα αντικείμενο τύπου Stack Element (prefab) όπως φαίνεται στο σχήμα του οποίου η μόνη κίνηση είναι η πτώση προς τον κάθετο άξονα Y. Αυτό το έχουμε πετύχει παγώνοντας κάθε δυνατότητα κίνησης του αντικειμένου σε κάθε άξονα εκτός του Y σε θέμα θέσης με τη χρήση του “**Rigidbody**” component.

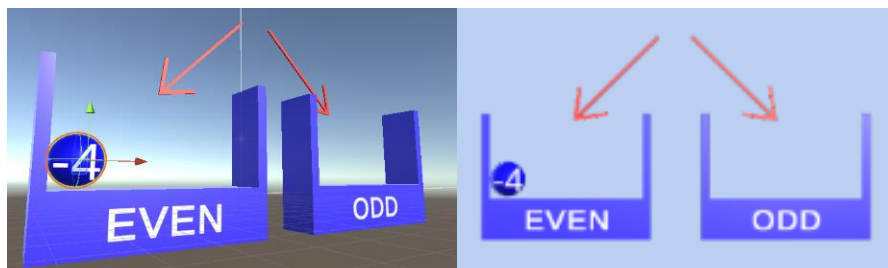
Επίσης για να εφαρμόζεται κίνηση της πτώσης του αντικειμένου, το αντικείμενο εμφανίζεται κάθε φορά στο πάνω μέρος της στοίβας και το Rigidbody με τη “Use gravity” επιλογή του, το επιτρέπει να έχει ελεύθερη πτώση προς τα κάτω μέχρι να αγγίξει τον πάτο της στοίβας ή άλλο αντικείμενο τύπου Stack Element. Όλα τα αντικείμενα έχουν πάνω τους το **Box Collider** component το οποίο επιτρέπει την αυτόματη ανίχνευση σύγκρουσης και εφαρμογής φυσικής πάνω στα αντικείμενα. Με την Pop συνάρτηση, το πρώτο αντικείμενο της στοίβας απλώς διαγράφεται (**Destroy**).

Εκτός από την παρουσίαση με τη χρήση στοιβών, θα πρέπει ταυτόχρονα να παρουσιάζουμε και τις ενδιάμεσες ενέργειες του αλγορίθμου. Επομένως έχουμε υλοποιήσει ένα κεντρικό αντικείμενο στο οποίο θα παρουσιάζονται όλα αυτά τα αποτελέσματα (Σχήμα 4.6).



Results Gameobjects - Σχήμα 4.6

Στο πάνω μέρος θα φαίνεται το πολυώνυμο με το οποίο ασχολείται ο FFT εκείνη την στιγμή τόσο στην πολυωνυμική του μορφή όσο και στην μορφή του σε διάνυσμα συντελεστών. Όταν διαμοιράζεται το πολυώνυμο, θα εμφανίζονται βέλη τα οποία θα παραπέμπουν σε δοχεία (containers) τα οποία παρουσιάζουν τους ζυγούς και περιττούς συντελεστές καθώς διαχωρίζονται (Σχήμα 4.7). Οι συντελεστές αυτοί θα παρουσιάζονται μέσα σε σφαίρες όπως και στον αρχικό μας αλγόριθμο (Κεφάλαιο 2 – Selection Sort). Επιπρόσθετα οι σφαίρες καθώς τοποθετούνται μέσα στο δοχείο, εμφανίζονται με χρήση παρεμβολής από την αρχή του δοχείου μέχρι το τέλος όπως και στις εμφανίσεις των σφαιρών του Selection Sort. Μπορούν να εμφανιστούν μέχρι 4 σφαίρες ανά γραμμή και έχουμε 2 γραμμές. Άρα κάθε δοχείο χωράει 8 συντελεστές. Συνεπώς μπορούμε να χωρίσουμε πολυώνυμο μέχρι 16^{ου} βαθμού (8 ζυγούς και 8 μονούς συντελεστές). Είναι υπέρ αρκετό καθώς αν επιλέξουμε να εφαρμόσουμε FFT σε πολυώνυμο μεγαλύτερου βαθμού θα έχουμε περιπλοκές χώρου και θα ήταν αχρείαστο. Ο μαθητευόμενος μπορεί να μάθει πως δουλεύει ο αλγόριθμος και με την χρήση απλών παραδειγμάτων με χαμηλή πολυπλοκότητα.



3D Containers – Even/Odd division - Σχήμα 4.7

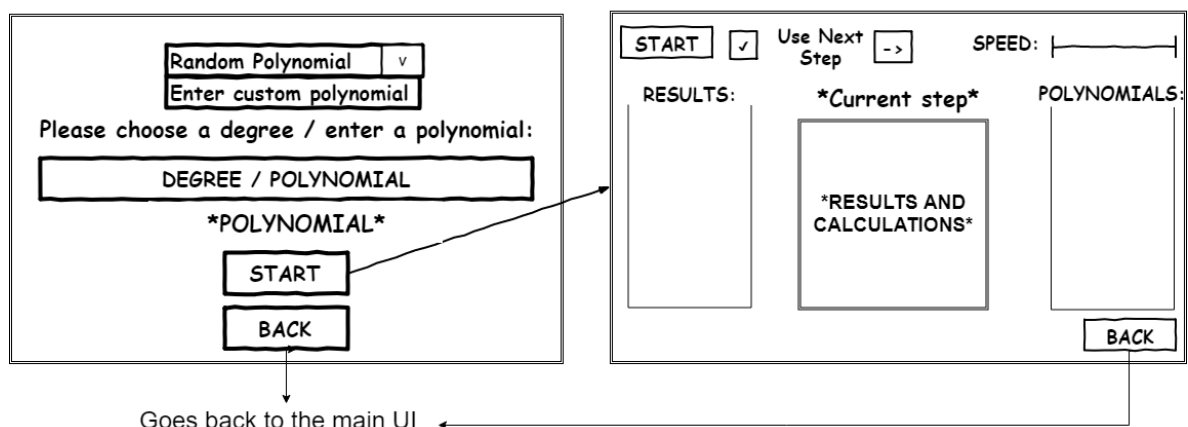
Ο έλεγχος των πιο πάνω δοχείων “Container” γίνεται με το **ContainerDisplay.cs** (Παράρτημα Γ) script component. Επιπλέον θα εμφανίζουμε και θα εξαφανίζουμε τα πιο

πάνω κόκκινα βέλη καθώς μπαίνουν οι συντελεστές με τη σειρά μέσα στο αντίστοιχο δοχείο βάση του πολυωνύμου που βρίσκεται πιο πάνω.

Εκτός από αυτούς τους υπολογισμούς, μέσα στο ίδιο το κεντρικό αντικείμενο θα παρουσιάζονται οι πράξεις που γίνονται με τους μιγαδικούς αριθμούς που έχουμε προαναφέρει. Κάτω από τα δοχεία που γίνεται ο διαμοιρασμός του πολυωνύμου (Σχήμα 4.6 – τρίτο τρισδιάστατο ορθογώνιο σχήμα) εμφανίζεται σε text μορφή (με τη χρήση του TextMeshPro-Text) ο υπολογισμός των ριζών όταν φθάνουμε σε τελική περίπτωση και πιο κάτω το αποτέλεσμα του τελικού πολυωνύμου όπως φαίνονται στις γραμμές 45-52 του Σχήματος 4.4.

4.3.2 Υλοποίηση Διεπαφής

Προτού υλοποιήσουμε την διεπαφή για το χειρισμό του FFT, θα πρέπει να λάβουμε υπόψη τί επιλογές θα έχει ο χρήστης για τον τρόπο εισαγωγής ενός πολυωνύμου. Μετά από μελέτη καταλήξαμε στο ότι ο ιδανικότερος τρόπος εισαγωγής ενός πολυωνύμου είναι με την χρήση ενός τυχαιοποιημένου παράγοντα ή με την χειρωνακτική εισαγωγή των συντελεστών. Το πολυώνυμο όμως που θα λαμβάνεται ως είσοδο θα πρέπει οπωσδήποτε να βρίσκεται σε βαθμό δύναμης του δύο (2/4/8/16/32) για να μπορεί να εφαρμοστεί σωστά η διαδικασία του διαιρεί και βασίλευε καθώς το πολυώνυμο κάθε φορά μοιράζεται στα 2. Όπως έχουμε αναφέρει προηγουμένως, ο μέγιστος βαθμός πολυωνύμου που θα δεχόμαστε θα είναι 16^{ου} βαθμού για σκοπούς απλότητας. Άρα θα μπορούμε να χρησιμοποιήσουμε πολυώνυμα βαθμού 2/4/8/16 μόνο.



Προσχέδιο Διεπαφής FFT - Σχήμα 4.8

Στο σχήμα 4.8 βλέπουμε ένα προσχέδιο της διεπαφής για τον αλγόριθμό μας. Μπαίνοντας στην σκηνή του FFT ο χρήστης έχει την επιλογή να διαλέξει μεταξύ παραγωγής τυχαίου πολυωνύμου ή χειροκίνητα καταχωρημένου. Ανάλογα με την επιλογή του, επιλέγοντας το κουμπί “START” ξεκινά η εκτέλεσή του. Ο επόμενος καμβάς παρουσιάζει βασικές ενέργειες ελέγχου όπως έχουμε υλοποιήσει και στον Selection Sort με ευνόητες λειτουργίες. Τα κουμπιά “BACK” σε παραπέμπουν στην προηγούμενη κύρια διεπαφή του προγράμματός μας. Πιο κάτω φαίνεται η τελική εμφάνιση του πρώτου μας καμβά (Σχήμα 4.9).

Polynomial Input Interface - Σχήμα 4.9

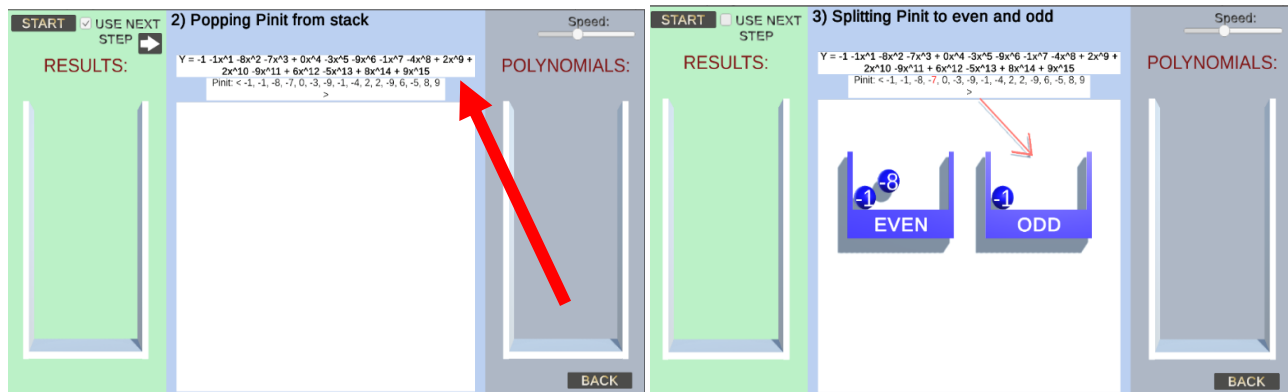
Στη επιλογή “Random Polynomial” μπορούμε να επιλέξουμε ένα τυχαίο πολυώνυμο 2/4/8 ή 16^{ου} βαθμού που παράγεται καθώς επιλέγουμε την ανάλογη επιλογή από το dropdown box. Η άλλη μας επιλογή “Enter Custom Polynomial” επιτρέπει στο χρήστη να εισάγει δικό του πολυώνυμο με απλή εισαγωγή αριθμών διαχωρισμένων με διάστημα μεταξύ τους. Αν ο χρήστης εισάγει λάθος πολυώνυμο, θα χρησιμοποιηθεί το $Y = -2 + 3x$.

4.3.3 Αναπαράσταση και Εκτέλεση Αλγορίθμου

Μετά από την επιλογή ενός επιθυμητού πολυωνύμου, καθώς επιλέγουμε το κουμπί “START” (Σχήμα 4.9), επιλέγοντας ξανά το κουμπί “START”, ο αλγόριθμος FFT ξεκινά.

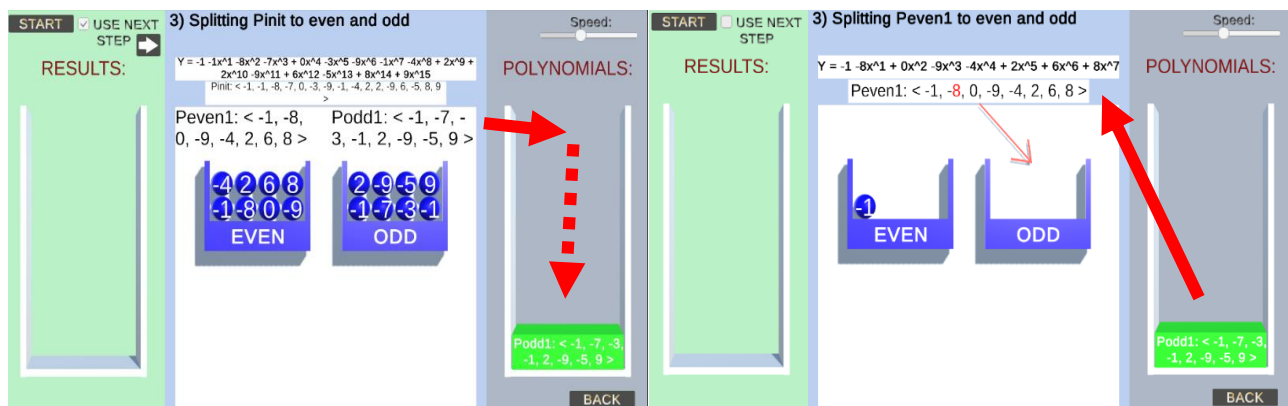
Visual FFT – Βήμα 1 - Σχήμα 4.10

Το πολυώνυμο εμφανίζεται στο πάνω κεντρικό κουτί με χρήση της εντολής Instantiate και ταυτόχρονα μπαίνει στην στοίβα “Polynomials” τόσο προγραμματιστικά όσο και εικονικά (Σχήμα 4.10) Ακολούθως εκτελείται το 2^ο βήμα.



Visual FFT – Βήμα 2 - Σχήμα 4.11

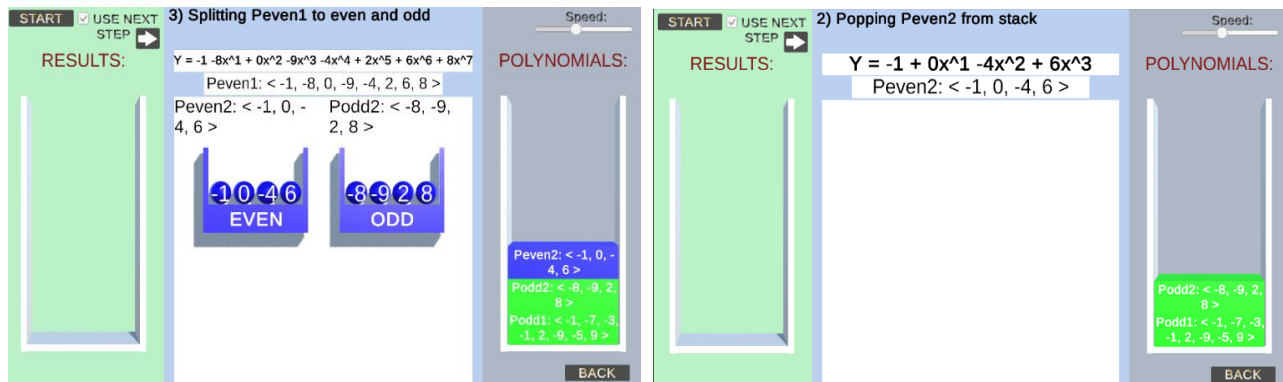
Στο βήμα αυτό ο αλγόριθμος κοιτάζει αν υπάρχει πολυώνυμο στην στοίβα “Polynomials”, αν ναι, τότε το κάνει Pop από την στοίβα του Script που είναι συνημμένο σε αυτό το αντικείμενο και ταυτόχρονα κάνει Destroy το αντίστοιχο αντικείμενο εικονικά. Συνεπώς αρχίζει και μοιράζει τους συντελεστές του στα δοχεία που έχουμε αναφέρει με κίνηση χρησιμοποιώντας Instantiations (Σχήμα 4.11).



Visual FFT – Βήμα 3 - Σχήμα 4.12

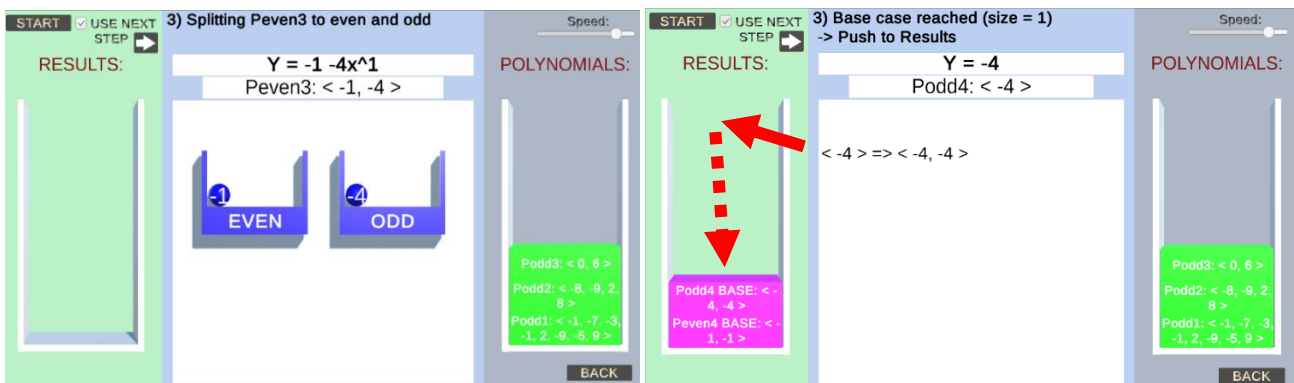
Όταν ολοκληρωθεί ο διαμοιρασμός (Βήμα 3)), τα δύο νέα πολυώνυμα μπαίνουν στη στοίβα “Polynomials” (Σχήμα 4.12). Οι Odd συντελεστές μπαίνουν με πράσινο χρώμα ενώ τα Even με μπλε με αυτή τη σειρά. Επίσης δίπλα από το όνομα του πολυωνύμου, καταγράφεται και το επίπεδο στο οποίο βρισκόμαστε (όπως φαίνεται στο σχήμα μας πχ: Peven1 : το πολυώνυμο με τους ζυγούς συντελεστές βάθους 1). Ακολούθως αφού η στοίβα

“Results” είναι ακόμη κενή και η “Polynomials” περιέχει έστω και ένα στοιχείο μέσα, ο αλγόριθμος βγάζει το επόμενο πολυώνυμο από τη στοίβα και εκτελεί την ίδια διαδικασία διαμοιρασμού συντελεστών (Βήμα 3)).



Visual FFT – Επανάληψη βήματος 2,3 - Σχήμα 4.13

Η διαδικασία αυτή της διάσπασης πολυωνύμων επαναλαμβάνεται συνεχώς (Σχήμα 4.13) μέχρι να φτάσουμε στην τελική περίπτωση (Σχήμα 4.14) όπου θα έχουμε πολυώνυμο βαθμού 0 (με ένα μόνο συντελεστή) στην στοίβα μας.

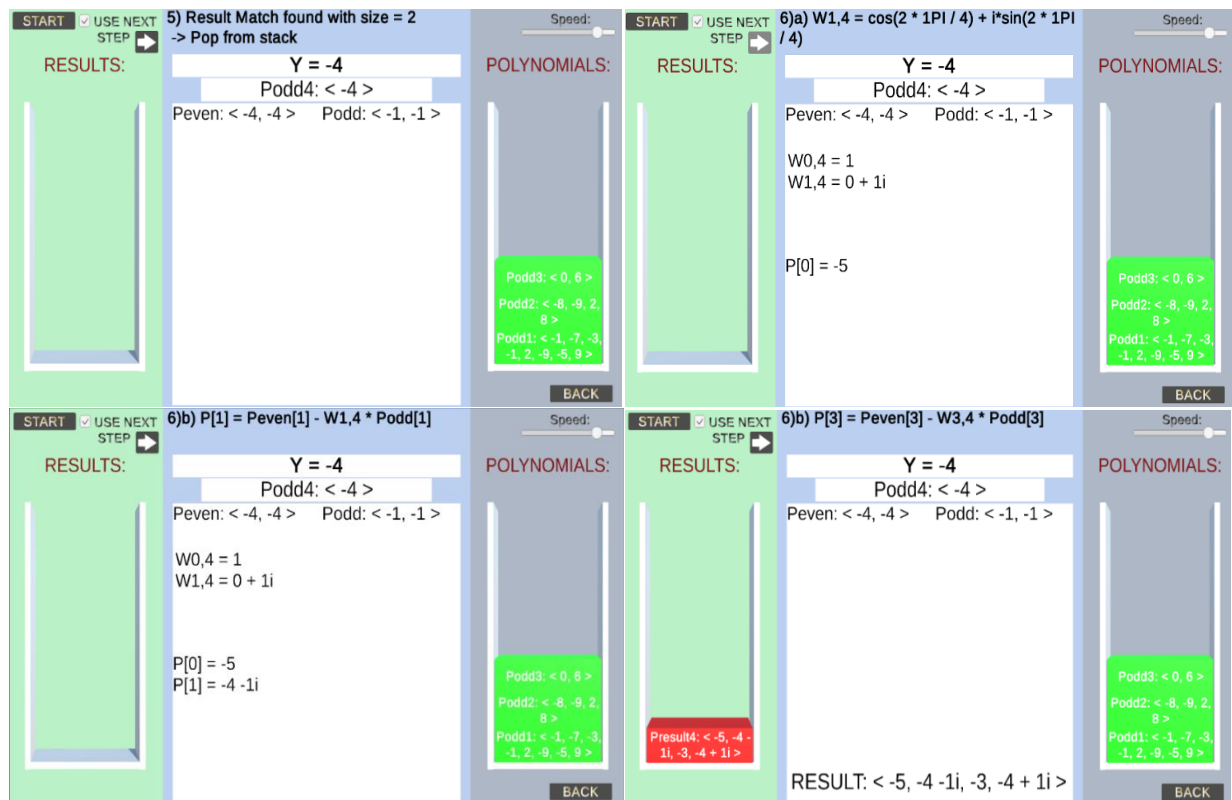


Visual FFT – Βήμα 3 – Τελική Περίπτωση - Σχήμα 4.14

Σε αυτό το σημείο ο αλγόριθμος όταν κάνει Pop από την “Polynomials” θα ανιχνεύσει την τελική περίπτωση και θα δημιουργήσει ως αποτέλεσμα ένα νέο διάνυσμα συντελεστών μεγέθους 2 το οποίο θα περιέχει 2 φορές τον ίδιο συντελεστή. Ακολούθως το αποτέλεσμα αυτό θα γίνει Pushed στη στοίβα “Results” με χρώμα ροζ το οποίο συμβολίζει τις τελικές περιπτώσεις. Επίσης στο όνομα του αποτελέσματος καταγράφεται αν είναι από τους ζυγούς ή περιττούς συντελεστές, το επίπεδο στο οποίο έχει γίνει ο διαμοιρασμός αυτός και η λέξη “BASE” αν είμαστε σε βασική περίπτωση. (Σχήμα 4.14 - πχ: Podd4 BASE <4,-4>)

Όταν συμπληρωθούν δύο συνεχόμενα αποτελέσματα με ίδιο μέγεθος στη στοίβα “Results”, τότε ο αλγόριθμος θα μπορεί να υπολογίσει το αποτέλεσμα του συνδυασμού τους.

Σε κάθε επανάληψη πριν την διάσπαση των πολυωνύμων, ελέγχεται με ένα Pop και ένα Peek από τη στοίβα “Results”, αν έχουμε δύο αποτελέσματα ίσου μεγέθους. Αν ναι τότε γίνεται Popred και το δεύτερο αποτέλεσμα που βρίσκεται στη στοίβα και αρχίζει η διαδικασία συγχώνευσης αυτών των δύο διανυσμάτων όπως έχουμε περιγράψει (Σχήμα 4.15), αλλιώς τοποθετείται πίσω στην στοίβα το αποτέλεσμα που έγινε Popred.



Visual FFT – Βήμα 5,6 - Σχήμα 4.15

Εδώ παρατηρούμε ότι έχει ανιχνευθεί ταίρι με μέγεθος 2 (Βήμα 5)). Συνεπώς θα γίνουν Popred τα δύο αποτελέσματα από την "Results" και θα αρχίσει ο υπολογισμός των μιγαδικών ριζών τους. Επειδή έχουμε δύο πολυώνυμα μεγέθους 2, ο συνδυασμός τους θα έχει ως αποτέλεσμα πολυώνυμο $4^{ου}$ βαθμού. Έτσι υπολογίζονται οι 4 ρίζες με την διαδικασία που έχουμε περιγράψει. Θα εκτελεστούν 4 φορές τα βήματα 6a) και 6b). (όπως

έχουμε πει στην επεξήγηση στο υποκεφάλαιο 4.2.1) Πχ: για το 2^ο στοιχείο του αποτελέσματος μας υπολογίζεται πρώτα ο ακόλουθος μιγαδικός

$$W_{1,4} = \cos\left(2 * \frac{1 * \pi}{4}\right) + 1 * \sin\left(2 * \frac{1 * \pi}{4}\right)$$

Και βάση αυτού του μιγαδικού αριθμού υπολογίζεται το αντίστοιχο αποτέλεσμα του νέου διανύσματος.

$$\begin{aligned} P(W_{j,2n}) &= \text{Peven}(W_{j,2n}^2) + W_{j,2n} * \text{Podd}(W_{j,2n}^2) \\ \Rightarrow P(W_{1,4}) &= \text{Peven}(W_{1,4}^2) + W_{1,4} * \text{Podd}(W_{1,4}^2) \end{aligned}$$

Το αποτέλεσμά μας όπως βλέπουμε στο σχήμα 4.15 για αυτό το παράδειγμα είναι το νέο διάνυσμα <-5, -4-1i, -3, -4+1i>. Αυτό με τη σειρά του μπαίνει στη στοίβα “Results”.

Για τους υπολογισμούς των αποτελεσμάτων αυτών έχει υλοποιηθεί μια μικρή διευκόλυνση στον κώδικά μας (Σχήμα 4.16). Παρατηρήσαμε ότι ο τύπος μας μπορεί να έχει απλοποίηση προγραμματιστικά. Για κάθε $i < n/2$ έχουμε ότι $W_{i,n} = -W_{i+\frac{n}{2},n}$.

πχ:

$$W_{0,4} = -W_{2,4}$$

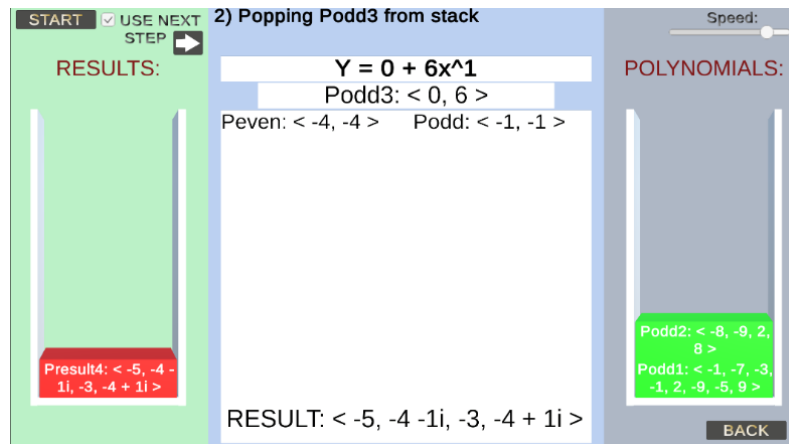
$$W_{1,4} = -W_{3,4}$$

Έτσι κάνοντας μονάχα $n/2$ προσπελάσεις κάθε φορά, μπορούμε να υπολογίσουμε ταυτόχρονα το $P(i)$ και $P(i+n/2)$.

$$\begin{aligned} \Rightarrow P(W_{i,n}) &= \text{Peven}(W_{i,n}) + W_{i,n} * \text{Podd}(W_{i,n}) \quad \text{και} \\ P(W_{i+\frac{n}{2},n}) &= \text{Peven}(W_{i,n}) - W_{i,n} * \text{Podd}(W_{i,n}) \end{aligned}$$

```
// Calculate the new array for the 2*N roots
// using the popped results of the previous
// fft(Peven) and fft(Podd)
int newN = r1.Length * 2;
Complex[] newP = new Complex[newN];
for (int i = 0; i < newN / 2; i++)
{
    float kth = 2 * i * Mathf.PI / newN;
    Complex wk = new Complex(Mathf.Cos(kth), Mathf.Sin(kth));
    newP[i] = r1[i] + (wk * (r2[i]));
    newP[i + newN / 2] = r1[i] - (wk * (r2[i]));
}
results.Push(newP);
```

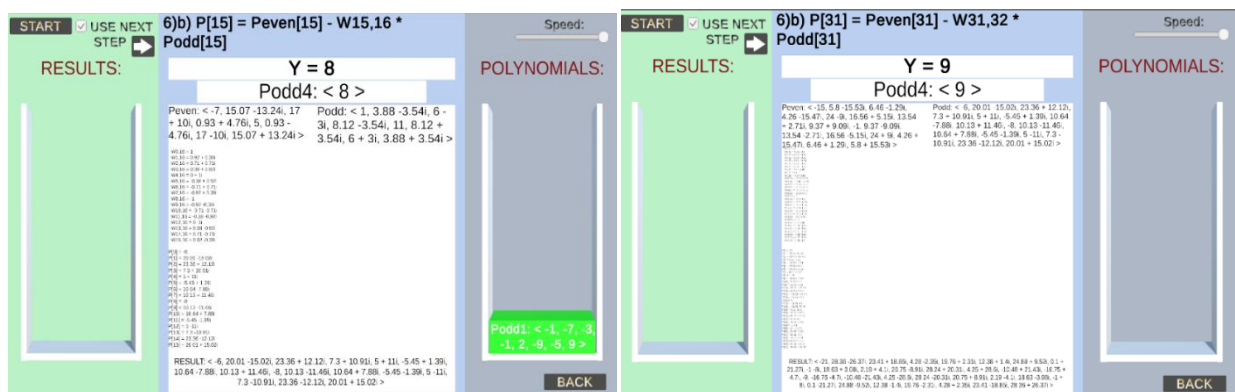
Βελτιστοποίηση υπολογισμού μιγαδικών ριζών FFT - Σχήμα 4.16



Visual FFT – Επανάληψη βήματος 2 - Σχήμα 4.17

Στην συνέχεια (Σχήμα 4.17) ο αλγόριθμός μας αφού βλέπει ότι δεν υπάρχει ταίρι στη στοίβα “Results”, προχωράει κανονικά με την διαδικασία διάσπασης των πολυωνύμων της στοίβας “Polynomials”.

Όσο προχωράει ο αλγόριθμος, δεν μας νοιάζουν οι μαθηματικοί υπολογισμοί που κάνει και έτσι μπορούμε να τους παρουσιάσουμε σε μη διακριτό σημείο. Ο μαθητευόμενος θέλει απλώς να μάθει την βασική διαδικασία.

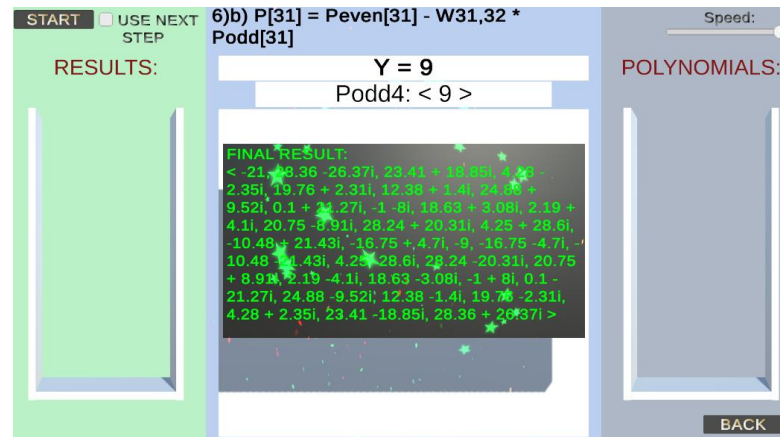


Visual FFT – Βήμα 6b) για 16/32 συντελεστές - Σχήμα 4.18

Έτσι όπως βλέπουμε στο πιο πάνω σχήμα (4.18), γίνονται υπολογισμοί αποτελεσμάτων για διανύσματα μεγέθους 16 και 32 συντελεστών οι οποίοι δεν είναι και τόσο ευδιάκριτοι. Στην χειρίστη περίπτωση μας θα έχουμε διανύσματα μεγέθους 32.

Τέλος όταν ολοκληρωθούν όλες αυτές οι επαναληπτικές εκτελέσεις και καταλήξουμε με άδεια στοίβα “Polynomials” και με στοίβα “Results” η οποία έχει μόνο ένα αποτέλεσμα, θα έχουμε καταλήξει στο τελικό μας αποτέλεσμα (Σχήμα 4.19)! Το αποτέλεσμα αυτό γίνεται Popped από την στοίβα και εμφανίζεται στην οθόνη σε ένα άλλο αντικείμενο το

οποίο περιλαμβάνει TextMeshPro με πράσινο χρώμα και πυροτεχνήματα [24] τα οποία δίνουν έμφαση στο τελικό μας αποτέλεσμα.



Visual FFT – Τελικό Αποτέλεσμα - Σχήμα 4.19

Να σημειωθεί ότι επιλέγοντας το “USE NEXT STEP” κουτί, ο αλγόριθμος κάνει μια παύση της εκτέλεσής του και με την επιλογή του βέλους → προχωρούμε σταδιακά βήμα-βήμα. Έτσι δεν χρειάστηκε να υλοποιήσουμε κουμπί παύσης. Ακόμη Μπορούμε να αλλάξουμε την ταχύτητα εκτέλεσης με το “Speed” slider που βρίσκεται πάνω δεξιά όπως και με τους άλλους αλγορίθμους.

Τα αποτελέσματά μας τα έχουμε επαληθεύσει με χρήση ενός ιστότοπου που υπολογίζει αποτελέσματα δεδομένου ενός πολυωνύμου για τον FFT [8].

4.4 Δυσκολίες/Προβλήματα και Τρόπος Επίλυσης

Κατά τη διάρκεια της υλοποίησής μας έχουν προκύψει τα πιο κάτω:

Προσαρμογή ευκρίνειας της οθόνης (Resolution) για την παρουσίαση όλων των αντικειμένων με τη σωστή οπτική γωνία. Το διορθώσαμε με τοποθέτηση τριών διαφορετικών καμερών κεντραρισμένων πάνω στα αντικείμενα. Μία για την στοίβα “Results”, μία για τους υπολογισμούς και παρουσίαση των αποτελεσμάτων και μία για την στοίβα “Polynomials”

Παρουσιάστηκε δυσκολία στην υλοποίηση και μετατροπή του αναδρομικού αλγορίθμου FFT στον επαναληπτικό του αντίστοιχο. Χρησιμοποιήσαμε συμβάσεις μετατροπής με την χρήση στοιβών αλλά επειδή ο αλγόριθμος κάνει χρήση δύο αναδρομικών καλεσμάτων, εμφανίστηκε δυσκολία. Μετά από πολλές προσπάθειες και πειράματα (trial and error), στη

τελική περίπτωση επιπλέων έπρεπε να γίνεται έλεγχος στη στοίβα των “Results” αν υπάρχουν δύο αποτελέσματα ίσου μεγέθους για να γίνουν popped από την στοίβα και να εφαρμοστεί πάνω τους η φάση «συνδύασε».

Δεν υπήρχε υλοποιημένο αντικείμενο για τους μιγαδικούς αριθμούς με τις πράξεις τους. Έτσι δημιουργήσαμε από την αρχή το αντικείμενο “Complex” (παράρτημα Γ) το οποίο είναι βασισμένο στην υπάρχον Java υλοποίηση του.

Στην αρχή έχει γίνει προσπάθεια υλοποίησης του αλγορίθμου με χρήση δυαδικού δένδρου για κάθε πολυώνυμο που δημιουργόταν. Αυτό έχει καταλήξει να μην έχει ευχάριστη εμφάνιση στο χρήστη καθώς επίσης και δυσκολία στην εκμάθησή του. Μετά από συζήτηση με τον επιβλέπον καθηγητή και από απόψεις από άλλα άτομα, έχουμε καταλήξει στην παρούσα κατάσταση του αλγορίθμου.

Κεφάλαιο 5.

Αλγόριθμος Επιλογής

5.1 Εισαγωγή	47
5.2 Περιγραφή Αλγορίθμου	48
5.3 Λεπτομέρειες Υλοποίησης	50
5.3.1 Μετατροπή σε Επαναληπτικό και Αρχική Σκέψη	50
5.3.2 Σχεδιασμός Πρωτοτύπου	51
5.3.3 Υλοποίηση Διεπαφής	53
5.3.4 Αναπαράσταση και Εκτέλεση Αλγορίθμου	55
5.4 Δυσκολίες, Τρόπος Επίλυσης και Διευκρινήσεις	60

5.1 Εισαγωγή

Ο επόμενος αλγόριθμος με τον οποίο θα ασχοληθούμε είναι ο αλγόριθμος Επιλογής (Selection Algorithm) [4,5]. Ανήκει επίσης στην κατηγορία αλγορίθμων τύπου «Διαίρει και Βασίλευε» όπως και ο FFT. Το πρόβλημα που λύνει αυτός ο αλγόριθμος είναι το πρόβλημα της επιλογής. Έστω ότι έχουμε ένα σύνολο από **n στοιχεία**. Η επιλογή του **i-οστού** στατιστικά διατεταγμένου στοιχείου, είναι εκείνο που είναι μεγαλύτερο ακριβώς από $i-1$ στοιχεία. Για παράδειγμα το 1-οστό είναι το ελάχιστο στοιχείο, το n -οστό είναι το μέγιστο και το $n/2$ -οστό η διάμεσος, δηλαδή το μεσαίο στοιχείο.

Ο αριθμός των συγκρίσεων που πρέπει να γίνουν για να βρεθεί το μεγαλύτερο ή το μικρότερο στοιχείο είναι μονάχα **$O(n)$** καθώς περιλαμβάνει μια απλή σάρωση του συνόλου μας. Τώρα για ένα στοιχείο που βρίσκεται ενδιάμεσα, η πολυπλοκότητά μας ανεβαίνει στις **$O(n \log n)$** συγκρίσεις.

Αυτό οφείλεται δεδομένου ότι εφαρμόζουμε μια απλή ταξινόμηση στο σύνολό μας όπως τη δυαδική εισαγωγή (binary insertion sort).

Ο αλγόριθμος Επιλογής από την άλλη πλευρά όμως πετυχαίνει την εύρεση οποιουδήποτε i -οστού στοιχείου σε γραμμικό χρόνο $O(n)$. Ως αποτέλεσμα είναι ένας χρήσιμος αλγόριθμος και ο τρόπος λειτουργίας του είναι άξιος να φαίνεται σε γραφική μορφή.

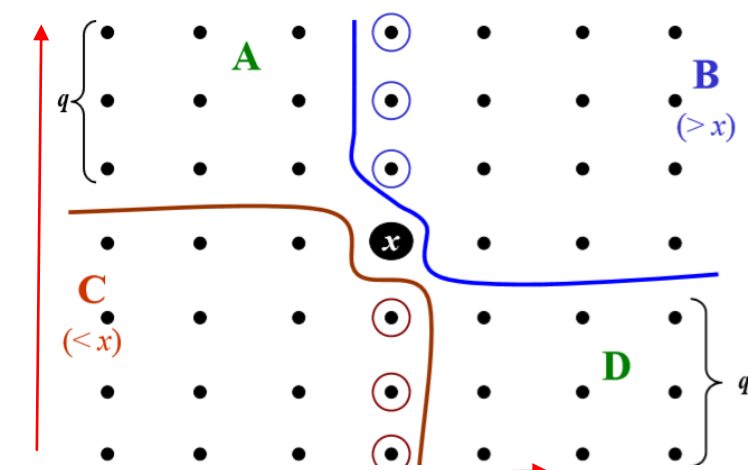
5.2 Περιγραφή Αλγορίθμου

Ο αλγόριθμος ΕΠΙΛΟΓΗ έχει την εξής μορφή (για είσοδο ένα διάνυσμα n στοιχείων και i αριθμού για εύρεση του i -οστού) (ΕΠΙΛΟΓΗ($P(n)$, i)) (Μπορεί να υλοποιηθεί και με 7άδες αντί 5άδες):

1. Διαίρεση των n στοιχείων σε $\left\lceil \frac{n}{5} \right\rceil$ ομάδες των 5 στοιχείων. (και τα υπόλοιπα $n \bmod 5$ στοιχεία σε άλλη ομάδα)
2. Εύρεση διαμέσου κάθε ομάδας με ταξινόμηση με χρήση δυαδικής εισαγωγής και μετά επιλογή του κεντρικού στοιχείου κάθε 5άδας (3^ο).
3. Τρέχουμε αναδρομικά την ΕΠΙΛΟΓΗ και βρίσκουμε τη διάμεσο X όλων των διαμέσων. (Αν έχουμε ζυγό αριθμό διαμέσων, τότε ο X θα είναι η κάτω διάμεσος)
4. Βάση του X , διαμερίζουμε τα στοιχεία γύρω του με τη χρήση της τροποποιημένης διαδικασίας ΔΙΑΜΕΡΙΣΗ.
Ορίζουμε r ίσο με το πλήθος των στοιχείων μικρότερα του X .
5. Αν $i = r+1$ (είναι η θέση του X) τότε επιστρέφουμε το X .
Αν $i < r+1$ τότε τρέχουμε την ΕΠΙΛΟΓΗ αναδρομικά για να βρούμε το i -οστό στοιχείο για τα στοιχεία $< X$.
Αλλιώς τρέχουμε την ΕΠΙΛΟΓΗ αναδρομικά για να βρούμε το $(i-(r+1))$ -οστό στοιχείο για τα στοιχεία $> X$.

Ο αλγόριθμος αυτός όπως και ο FFT έχει μετατραπεί από αναδρομικός σε επαναληπτικό αργότερα για να μπορεί να αναπαρασταθεί με τρισδιάστατα γραφικά και αντικείμενα.

Στο βήμα 4., η διαμέριση των στοιχείων γίνεται σε 4 ομάδες (Σχήμα 5.1)[4].



Διαμέριση ομάδων A-D – Αλγόριθμος επιλογής - Σχήμα 5.1

Όταν έχουμε ταξινομημένες τις ομάδες όσο οριζόντια όσο και κάθετα με βάση τους διαμέσους της κάθε ομάδας, μπορούμε να κάνουμε γρήγορους υπολογισμούς. Τα βέλη στο σχήμα δείχνουν από το μικρότερο στο μεγαλύτερο. Για την ομάδα C μπορούμε να συμπεραίνουμε αμέσως ότι όλα τα στοιχεία της είναι $< x$ και της B $> x$ για προφανές λόγους. Τώρα παραμένει να γίνει έλεγχος για τις ομάδες A και D. Ο έλεγχος αυτός προσφέρει ελάχιστη επίπτωση στη πολυπλοκότητα του αλγορίθμου και έτσι διατηρεί την γραμμική του πολυπλοκότητα.

Ο διαχωρισμός αυτός δεν χρειάζεται να υλοποιηθεί οπτικά. Ο χρήστης θα έχει επίγνωση στο πως λειτουργεί ο αλγόριθμος και στον τρόπο που διαχωρίζονται οι ομάδες.

5.3 Λεπτομέρειες Υλοποίησης

5.3.1 Μετατροπή σε Επαναληπτικό και Αρχική Σκέψη

Για να υλοποιήσουμε οπτικά αυτό τον αλγόριθμο θα πρέπει να σκεφτούμε την διαδικασία του βήμα προς βήμα και τον τρόπο που εκτελείται. Ως είσοδο λαμβάνουμε μια λίστα με ακέραιους αριθμούς και τον αριθμό i που θα αντιπροσωπεύει τον i -οστό αριθμό που γυρεύουμε.

1. Αρχικά για να κάνουμε πιο κατανοητό το τελικό αποτέλεσμα προς το χρήστη, θα παρουσιάσουμε τη λίστα μας ταξινομημένη και θα εμφανίζουμε το i -οστό στοιχείο το οποίο θα φαίνεται ακριβώς σε ποια θέση βρίσκεται.
2. Ακολούθως θα εκτελούμε τον αλγόριθμο επιλογή επαναληπτικά (βήμα 3-8) μέχρι να βρεθεί το τελικό μας αποτέλεσμα για την υφιστάμενη λίστα και i αριθμό.
3. Κάθε φορά θα διαιρούμε τα στοιχεία της υφιστάμενης λίστα μας σε 5άδες δημιουργώντας νέα λίστα για την κάθε ομάδα σε μία λίστα από λίστες με ακέραιους. Τα υπόλοιπα στοιχεία που περισσεύουν από τον διαμοιρασμό των ομάδων μπαίνουν και εκείνα σε δική τους λίστα μεγέθους < 5 .
4. Μετά θα ταξινομείται μία-μία η κάθε ομάδα χρησιμοποιώντας την έτοιμη επιλογή ταξινόμησης λίστας "Sort" για τα αντικείμενα τύπου List στη C#. Εδώ υποθέτουμε ότι η ταξινόμηση γίνεται με δυαδική εισαγωγή. Ο χρήστης απλώς θα βλέπει τα στοιχεία να ταξινομούνται.
5. Βρίσκουμε τις διαμέσους της κάθε ομάδας οι οποίοι βρίσκονται πάντα στη θέση $\text{int}(n/2 + 0.5)$. πχ: **5άδα** $\rightarrow \text{int}(5/2 + 0.5) = \text{int}(2.5+0.5) = \text{int}(3) = 3$, **4άδα** $\rightarrow \text{int}(4/2+0.5) = \text{int}(2.5) = 2$
6. Τοποθετούμε τις διαμέσους αυτές σε δική τους λίστα και ταξινομούμε αυτή τη λίστα. Βρίσκουμε την διάμεσο των διαμέσων (X) χρησιμοποιώντας τον ίδιο τρόπο όπως και στις ομάδες.
7. Χρησιμοποιώντας το X, βρίσκουμε όλα τα στοιχεία τα οποία είναι μικρότερα του X και όλα τα οποία είναι μεγαλύτερα και τα τοποθετούμε σε δικές τους με τη σειρά

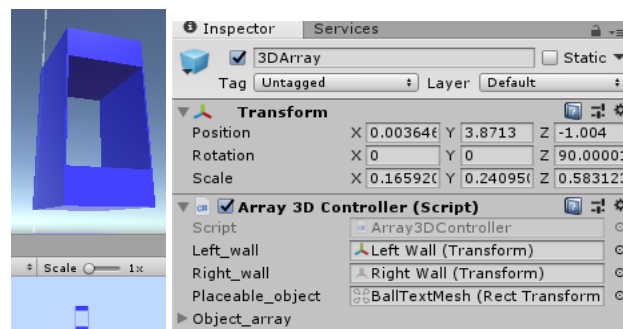
λίστες. Υποθέτουμε ότι έχουν γίνει οι ανάλογες συγκρίσεις για την διαμέριση στις 4 ομάδες που έχουμε προαναφέρει.

8. Έχουμε ότι ο αριθμός των στοιχείων $< X$ είναι r (μέγεθος λίστας με τα στοιχεία $< X$).
 - a. Αν το υφιστάμενο μας i (index) ισούται με το $r+1$ τότε το X είναι ο αριθμός που γυρεύουμε, τον εμφανίζουμε και βγαίνουμε από την επανάληψη.
 - b. Αλλιώς αν το i είναι μικρότερο από το $r+1$ τότε ορίζουμε την λίστα την οποία θα χρησιμοποιήσουμε, την λίστα με τα στοιχεία που είναι μικρότερα του X και το i παραμένει ως έχει. Επιστρέφουμε στην αρχή της επανάληψης.
 - c. Αλλιώς ($i > r+1$) χρησιμοποιούμε την λίστα με τα στοιχεία που είναι μεγαλύτερα του X και το i το αλλάζουμε σε $i-r-1$. Επιστρέφουμε στην αρχή της επανάληψης.

5.3.2 Σχεδιασμός Πρωτοτύπου

Πριν υλοποιήσουμε οποιοδήποτε αντικείμενο μέσα στο Unity, έχει γίνει ένας αρχικός σχεδιασμός της εμφάνισης του αλγορίθμου στο χαρτί. (Παράρτημα Δ)

Για την εμφάνιση κάθε στοιχείου θα χρησιμοποιήσουμε τις σφαίρες που έχουμε δημιουργήσει για τον αλγόριθμο Selection Sort (BallTextMesh - Σχήμα 2.6). Για την παρουσίαση της λίστας με τα στοιχεία έχουμε δημιουργήσει ένα 3D prefab με όνομα “3DArray” (Σχήμα 5.2). Το αντικείμενο αυτό ελέγχεται από το **Array3DController.cs** script (Παράρτημα Δ).

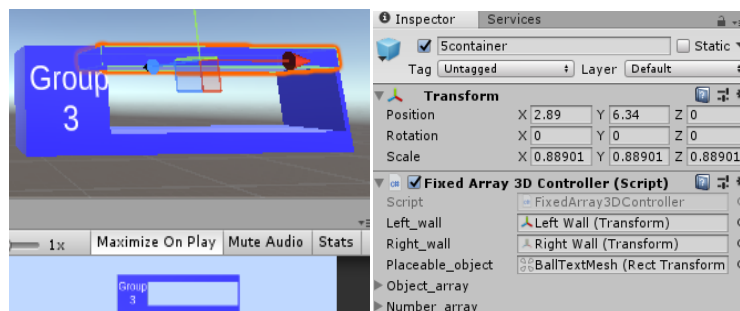


3DArray GameObject - Σχήμα 5.2

Ανάλογα με τον αριθμό των αντικειμένων που θα περιέχει η λίστα την οποία θα παρουσιάσει, αλλάζει δυναμικά το μέγεθός του, αλλάζοντας τις παραμέτρους του **localScale**. Το αρχικό του μέγεθος χωράει μόνο ένα αντικείμενο. Έτσι με n αντικείμενα, αν

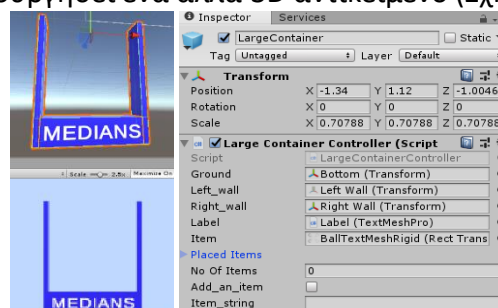
πολλαπλασιάσουμε το μέγεθος του πίνακα αυτού στο Y άξονα με το η, το νέο του μέγεθος θα χωράει η αντικείμενα. Η εμφάνιση των σφαιρών γίνεται με τον ίδιο τρόπο της γραμμικής παρεμβολής όπως και στον Selection Sort. Η μόνη διαφορά εδώ είναι ότι το αρχικό και τελικό μας σημείο μεταβάλλεται ανάλογα με την τοποθεσία του αριστερού και δεξιού τοίχου του 3DArray. Ταυτόχρονα ανάλογα με το μέγεθος του αντικειμένου μας, αλλάζουν και τα μεγέθη των σφαιρών που θα εμφανιστούν ούτως ώστε να χωράνε μέσα στο αντικείμενο. Μετά από πολλούς ελέγχους καταλήξαμε στο ότι μπορούμε να έχουμε μέχρι και 50 στοιχεία σε ένα πίνακα ώστε να είναι ευδιάκριτα προς το χρήστη. Τέλος, το ενσωματωμένο script προσφέρει την δυνατότητα αλλαγής χρώματος συγκεκριμένης σφαίρας ανά πάσα στιγμή και δυνατότητα καθαρισμού του πίνακα.

Για την παρουσίαση των ομάδων έχει δημιουργηθεί ένα ξεχωριστό αντικείμενο το οποίο έχει τις ίδιες ιδιότητες με το “3DArray” αλλά είναι σταθερού μεγέθους και περιέχει κείμενο για την ονομασία του. Το “5container” (Σχήμα 5.3) έχει την δυνατότητα παρουσίασης μέχρι και 5 στοιχείων με χρήση γραμμικής παρεμβολής. Μπορεί να ελεγχθεί από το script **Fixed3DArrayController.cs** (Παράρτημα Δ).



5container GameObject - Σχήμα 5.3

Εκτός από τις ομάδες, θα πρέπει να παρουσιάσουμε και την εύρεση των διαμέσων, της διάμεσου των διαμέσων (X) αλλά και τα μικρότερα και μεγαλύτερα στοιχεία του X. Συνεπώς έχουμε δημιουργήσει ένα άλλο 3D αντικείμενο (Σχήμα 5.4) – “LargeContainer”.



LargeContainer GameObject - Σχήμα 5.4

Το αντικείμενο αυτό όπως και το μικρότερό του αντίστοιχο (FFT - Σχήμα 4.7) έχει τις ίδιες ιδιότητες στην εμφάνιση σφαιρών. Εμφανίζει σε γραμμική παρεμβολή σφαίρες τύπου “BallTextMeshRigid” οι οποίες είναι όμοιες με τις “BallTextMesh” με την διαφορά ότι περιέχουν το component “RigidBody” που τις επιτρέπει να έχουν φυσική και βαρύτητα. Αυτές οι σφαίρες έχουν μόνο κάθετη πτώση (στον άξονα Y) και οι υπόλοιποι άξονες για κίνηση και περιστροφή είναι παγωμένοι. Στο “LargeContainer” μπορούμε να έχουμε σύνολο 6 σειρές από 5 σφαίρες, άρα σύνολο 30 σφαίρες σε κάθε δοχείο. Μας καλύπτει πλήρως καθώς μπορούμε να έχουμε το πολύ 10 5άδες άρα 10 διαμέσους για την εμφάνιση των διαμέσων και 30 στοιχεία το πολύ για τον διαχωρισμό των μεγαλύτερων και μικρότερων στοιχείων από το X.

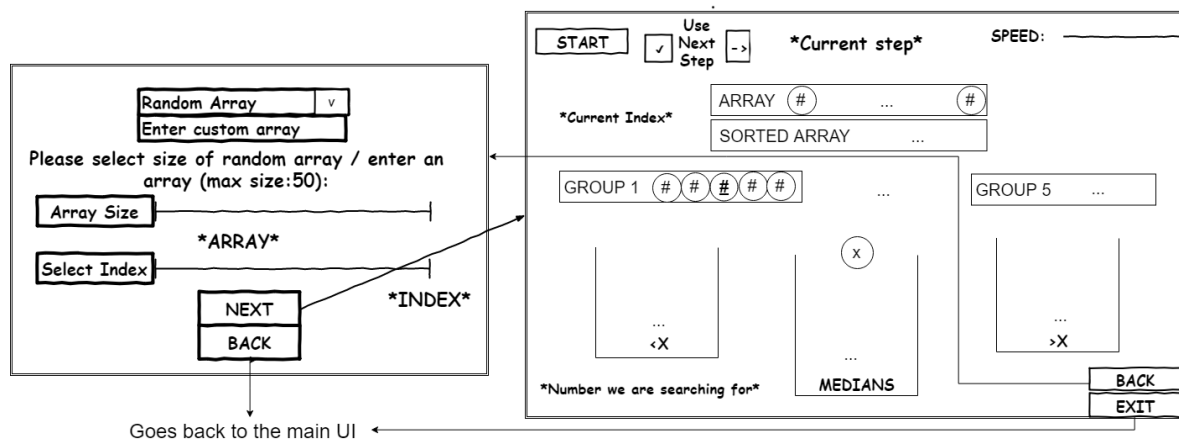
5.3.3 Υλοποίηση Διεπαφής

Μετά από σκέψη και έρευνα έχουμε καταλήξει στους εξής τρόπους με τους οποίους μπορεί να εισάγει ο χρήστης τις εισόδους του αλγορίθμου επιλογής:

- Επιλογή τυχαίου πίνακα ακέραιων στοιχείων από εισαγωγή μεγέθους 2 με 50.
- Δυνατότητα εισαγωγής πίνακα χειρωνακτικά στοιχείο-στοιχείο μεγέθους 2 με 50.

και εισαγωγή i-οστού (Index) στοιχείου που γυρεύουμε από το 1 μέχρι το 50 αναλόγως με το μέγεθος του πίνακα. Για ευκολία εμφάνισης των αριθμών και κατανόησης, οι ακέραιοι αριθμοί θα πρέπει να είναι από το 0-99.

Πριν ξεκινήσει η υλοποίηση στο Unity, έχουν γίνει τα ακόλουθα προσχέδια (Σχήμα 5.5) για τον τρόπο εισαγωγής της εισόδου από το χρήστη και την εμφάνιση του αλγορίθμου.



Προσχέδια διεπαφής αλγορίθμου Επιλογής - Σχήμα 5.5

Αν ο χρήστης θέλει να επιλέξει τυχαίο πίνακα, μπορεί να χρησιμοποιήσει Sliders για την επιλογή του. Αλλιώς αν επιλέξει την χειρωνακτική εισαγωγή πίνακα θα μπορεί να δακτυλογραφήσει ένα πίνακα της επιθυμίας του. Επίσης μπορεί να χρησιμοποιήσει Slider και για την επιλογή του index (i). Επιλέγοντας την επιλογή “NEXT”, προχωράμε στον καμβά που γίνεται η εκτέλεση του αλγορίθμου επιλογής. Όπως βλέπουμε στο σχήμα μας, φαίνονται πως θα φαίνονται περίπου τα βήματα εκτέλεσης μας όπως έχουμε περιγράψει προηγουμένως. Θα αναλυθούν λεπτομερώς στην συνέχεια. Τέλος τα κουμπιά μας εκτελούν τις ίδιες ενέργειες όπως και στους άλλους μας αλγόριθμους.

Μετά από την σχεδιάσή μας καταλήξαμε στη πιο κάτω τελική διεπαφή (Σχήμα 5.6).

The image shows two side-by-side panels of a Unity interface for array and index input.

Left Panel (Random Array):

- Dropdown menu: "Random Array"
- Text: "Please select a size for a random array (with random numbers between 0-99):"
- Slider: A horizontal slider with a value of 12.
- Text: "Array size: 12"
- Text: "Array: < 41, 60, 66, 15, 93, 28, 39, 53, 16, 79, 24, 57 >"
- Text: "Select Index:" followed by a slider with a value of 10.
- Text: "Index: 10"
- Buttons: "NEXT" and "BACK"

Right Panel (Enter Custom Array):

- Dropdown menu: "Enter Custom Array"
- Text: "Please enter your own array with size ranging from 2 to 50 (each value between 0-99 and seperated by space " "):"
- Input field: "9 84 7 2 4 92 75 23 6"
- Text: "Array size: 9"
- Text: "Array: < 9, 84, 7, 2, 4, 92, 75, 23, 6 >"
- Text: "Select Index:" followed by a slider with a value of 3.
- Text: "Index: 3"
- Buttons: "NEXT" and "BACK"

Array and Index Input Interface in Unity - Σχήμα 5.6

Πάνω στο dropdown κουτί που επιλέγουμε τον τρόπο εισαγωγής της εισόδου μας, έχουμε ως component το **DropdownArrayChoice.cs** script (Παράρτημα Δ). Αυτό το script είναι υπεύθυνο στην εμφάνιση και εξαφάνιση των αντίστοιχων στοιχείων στον καμβά μας. Επίσης για να έχουμε δυναμική εμφάνιση του πίνακα καθώς αλλάζουν οι τιμές έχουμε προσθέσει το **onValueChanged listener[2]** το οποίο εκτελεί μια συγκεκριμένη συνάρτηση “ValueChangeCheck” όταν αλλάζει η τιμή του αντίστοιχου αντικειμένου οποιαδήποτε στιγμή. Αυτή την δυνατότητα την προσθέσαμε στα δύο sliders της πρώτης μας οθόνης και στο input text box της δεύτερης όπως φαίνεται πιο πάνω χρησιμοποιώντας την εξής εντολή: `onValueChanged.AddListener(delegate { ValueChangeCheck(); });` Delegate είναι ένας όρος της C# και είναι παρόμοια όπως τους δείκτες σε συναρτήσεις στη C και C++. Η αναφορά αυτή μπορεί να αλλάξει κατά τη διάρκεια εκτέλεσης. Τα script που χρησιμοποιούν αυτή τη λειτουργία και είναι component στο αντίστοιχό τους slider είναι τα **SliderArrayText.cs** και **SliderIndexText.cs** (Παράρτημα Δ).

5.3.4 Αναπαράσταση και Εκτέλεση Αλγορίθμου

Σε αυτό το σημείο θα παρουσιαστεί η εκτέλεση του αλγορίθμου. Ο έλεγχος της εκτέλεσης του γίνεται από το script **SelectionAlgorithm.cs** (Παράρτημα Δ). Ξεκινώντας ως παράδειγμα θα χρησιμοποιήσουμε τον πιο κάτω τυχαιοποιημένο πίνακα με ακεραίους και τον τυχαία επιλεγόμενο αριθμό 39 ως το i (index) μας (Σχήμα 5.7).

Random Array

Please select a size for a random array (with random numbers between 0-99):

Array size: 50

Array: < 61, 34, 96, 88, 19, 12, 38, 63, 42, 47, 6, 46, 55, 30, 37, 72, 93, 65, 83, 62, 85, 66, 73, 16, 39, 9, 41, 23, 86, 49, 57, 28, 0, 20, 74, 78, 75, 45, 95, 69, 52, 53, 1, 97, 48, 18, 29, 17, 14, 15 >

Select Index: Index: 39

NEXT

BACK

Visual Selection Algorithm – Input Example - Σχήμα 5.7

Επιλέγοντας το κουμπί “NEXT” προχωράμε στην οθόνη εκτέλεσης του αλγορίθμου (Σχήμα 5.8). Αρχικά η οθόνη αυτή θα είναι άδεια και θα φαίνονται απλώς τα ακόλουθα ενδεικτικά αντικείμενα. Επιλέγοντας το κουμπί “START”, ο αλγόριθμός μας ξεκινά την λειτουργία του.

START

NEXT STEP

Current Step

Speed:

INDEX

CURRENT INDEX

BACK

EXIT

EXPECTED SOLUTION:

Visual Selection Algorithm – Initial Run Screen - Σχήμα 5.8

Έχουμε προεπιλέξει την επιλογή του “NEXT STEP” για να βρίσκεται ο αλγόριθμος μας σε συνεχή παύση και θα προχωράμε βήμα-βήμα με το “→”. Στην αρχή εμφανίζεται ο πίνακας που έχουμε εισάγει ως είσοδο και στην συνέχεια παρουσιάζεται το Index που γυρεύουμε (Σχήμα 5.9).

START

NEXT STEP

Displaying the given array...

Speed:

INDEX

CURRENT INDEX

START

NEXT STEP

Index we are searching for: 39

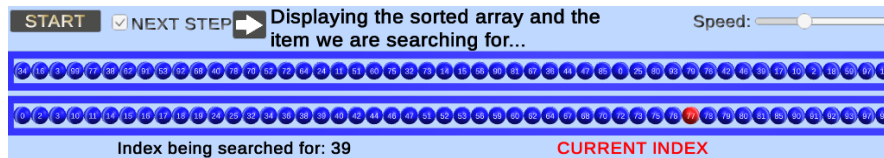
Speed:

Index being searched for: 39

CURRENT INDEX

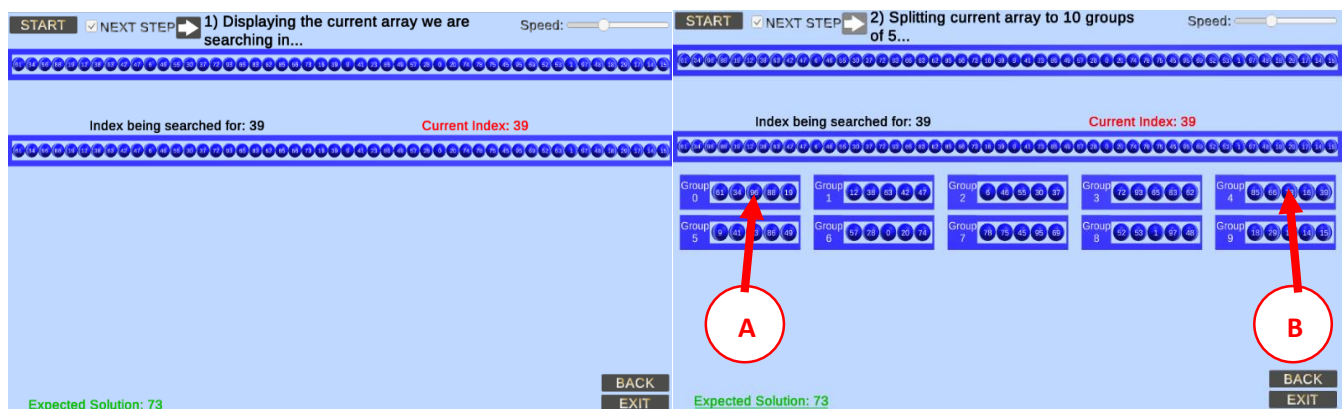
Visual Selection Algorithm – Array display - Σχήμα 5.9

Επίσης πριν ξεκινήσει ο αλγόριθμος, εμφανίζεται ταξινομημένος ο πίνακας με χρωματισμένο το στοιχείο που γυρεύουμε σε κόκκινο χρώμα για να δείξουμε ότι όντως θα είναι σωστό το τελικό μας αποτέλεσμα (Σχήμα 5.10).



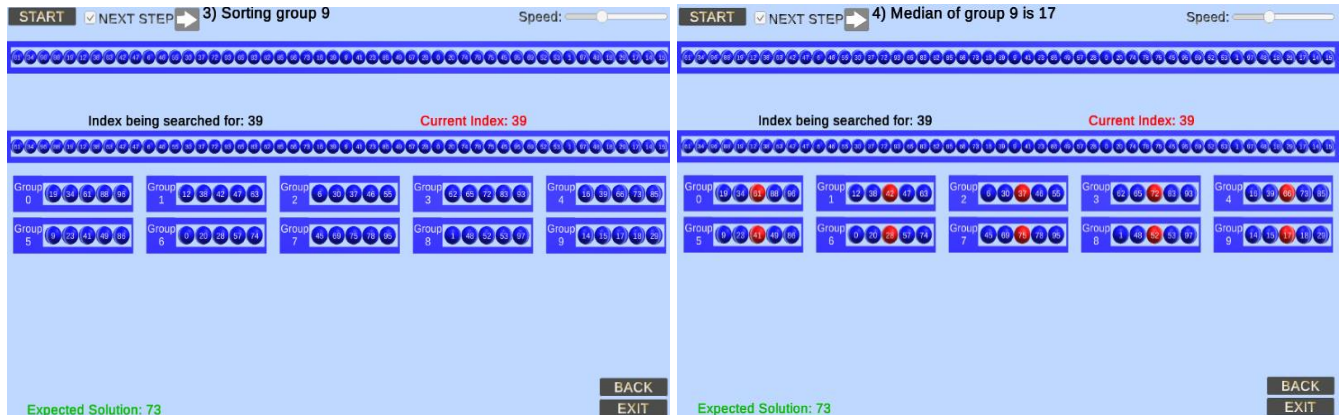
Visual Selection Algorithm – Sorted Array and element - Σχήμα 5.10

Ο αριθμός που γυρεύουμε θα βρίσκεται συνεχώς κάτω στην αριστερή γωνία σε πράσινα γράμματα στο “Expected Solution:” (Σχήμα 5.11), στην περίπτωσή μας είναι το 73. Ακολουθως εκτελείται το βήμα 1) όπου παρουσιάζεται το υφιστάμενο Index και πίνακα που θα χρησιμοποιήσει ο αλγόριθμος για την εκτέλεση του (σε κόκκινα γράμματα το “Current Index” και κάτω του ακριβώς ο πίνακας). Στο βήμα 2) γίνεται ο διαμοιρασμός των 5άδων όπως έχουμε περιγράψει εμφανίζοντας μία-μία κάθε ομάδα από τα αριστερά στα δεξιά και από πάνω προς τα κάτω. Για την εμφάνισή τους έχει γίνει η ίδια υλοποίηση γραμμικής παρεμβολής όπως και στο Selection Sort με αρχικό σημείο το A και τελικό το B όπως φαίνεται στο σχήμα μας. Αφού γνωρίζουμε ότι ο μέγιστος αριθμός στοιχείων που μπορούμε να έχουμε είναι 50 θα έχουμε μόνο 2 γραμμές με 5 ομάδες η κάθε μία στη χειρίστη περίπτωση. Για την εμφάνιση της 2^{ης} γραμμής, πολύ απλά αν τα αντικείμενά μας είναι περισσότερα από 5 τότε αφαιρούμε -1 στον άξονα Y στο αρχικό και τελικό σημείο και εφαρμόζουμε ξανά γραμμική παρεμβολή.



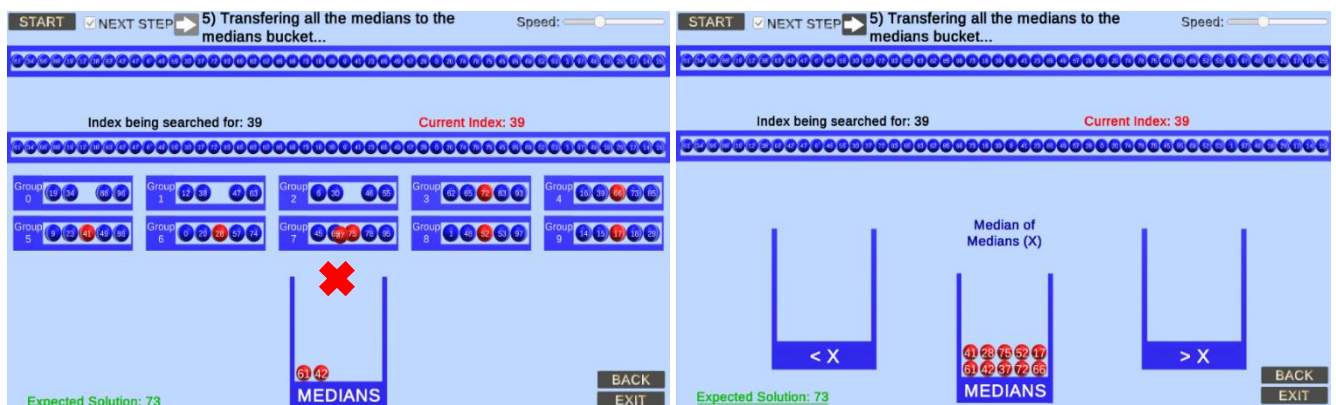
Visual Selection Algorithm – Βήμα 1,2 - Εμφάνιση 5άδων με χρήση lerp - Σχήμα 5.11

Μόλις εμφανιστούν όλες οι ομάδες, εκτελείται το βήμα 3) για την κάθε ομάδα ξεχωριστά (Σχήμα 5.12). Σε αυτό το βήμα ταξινομούνται οι ομάδες μας άμεσα χωρίς εμφάνιση κίνησης κάνοντας Destroy κάθε σφαίρα και επανεμφανίζοντάς τις σε ταξινομημένη μορφή. Όταν ταξινομηθεί και η τελευταία μας ομάδα, αρχίζει να εκτελείται το βήμα 4) για την κάθε μια με τη σειρά. Εδώ αρχίζει να φαίνεται με κόκκινο χρώμα η διάμεσος κάθε ομάδας ταυτόχρονα εμφανίζεται στο υφιστάμενο βήμα πάνω κεντρικά.



Visual Selection Algorithm – Βήμα 3,4 – Sort και εύρεση διαμέσων - Σχήμα 5.12

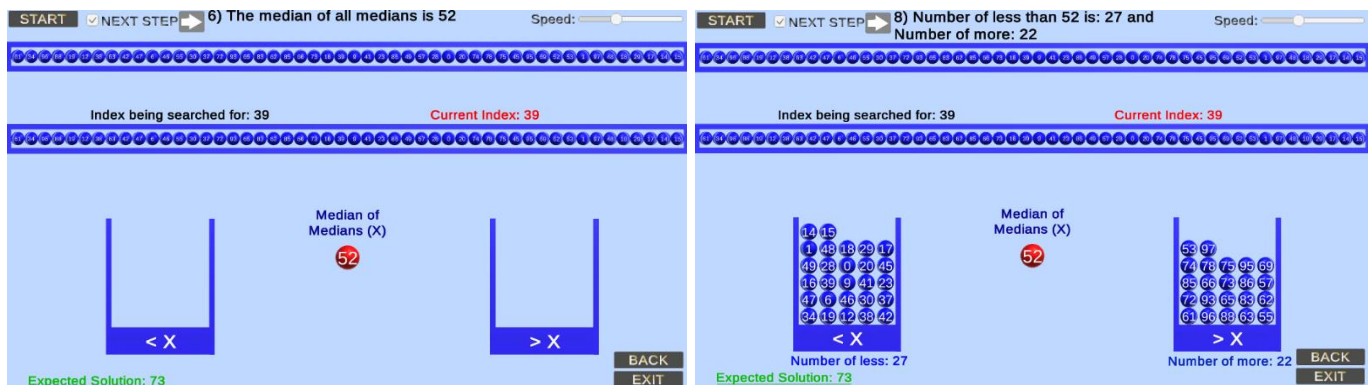
Στο βήμα 5), γίνεται μεταφορά κάθε διαμέσου στο νεοεμφανιζόμενό μας “MEDIANs” LargeContainer (Σχήμα 5.13). Ο τρόπος που υλοποιήθηκε αυτή η κίνηση κάθε διαμέσου είναι με τη χρήση του ενσωματωμένου script που έχει κάθε σφαίρα μας “BallMover” που έχουμε χρησιμοποιήσει στο Selection Sort κατά τα swaps του αλγορίθμου. Ορίζοντας ως πορεία το σημείο X που φαίνεται στο σχήμα μας (η θέση που βρίσκεται το “MEDIANs” δοχείο), ενεργοποιώντας το “BallMover” της επιθυμητής μας διαμέσου, η ανάλογη σφαίρα παίρνει την ανάλογη πορεία προς εκείνο το σημείο. Μόλις φτάσει στο σημείο X, την κάνουμε Destroy και αμέσως εμφανίζουμε την διάμεσο στο δοχείο μας.



Visual Selection Algorithm – Βήμα 5 – Συγκέντρωση των διαμέσων - Σχήμα 5.13

Μόλις ολοκληρωθεί αυτή η μεταφορά των διαμέσων, όλα τα αντικείμενα τα οποία χρησιμοποιηθήκαν για τις ομάδες διαγράφονται από την οθόνη με την Destroy και προγραμματιστικά καθώς δεν θα χρειαστούν άλλο στον αλγόριθμό μας. Επιπρόσθετα, εμφανίζονται τα δοχεία “< X” και “> X” αλλά και το text “Median of Medians (X)”.

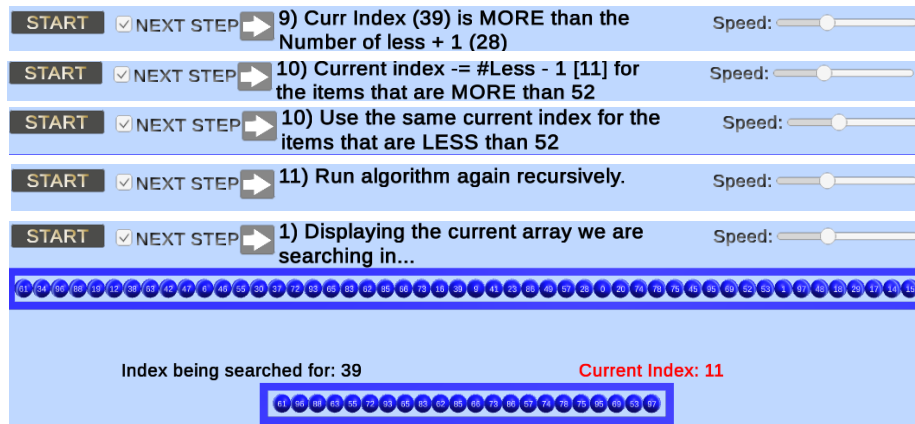
Στο 6^ο βήμα (Σχήμα 5.14), αφαιρείται η σφαίρα που περιέχει την διάμεσο όλων των διαμέσων μας από το κεντρικό μας δοχείο (52). Πρώτα απενεργοποιούμε το “RigidBody” component που χρησιμοποιείται για τη φυσική της και εφαρμόζουμε την “BallMover” προς το σημείο που βλέπουμε κάτω από το “Median of Medians (X)” text που έχουμε ενεργοποιήσει. Μόλις ολοκληρωθεί η κίνηση αυτή, διαγράφουμε το δοχείο “MEDIAN” και όλα τα περιεχόμενά του. Τα περιεχόμενά του περιέχουν και τη σφαίρα που έχουμε μόλις μετακινήσει. Έτσι αφού θα διαγραφτεί και εκείνη, εμφανίζουμε μια νέα κόκκινη σφαίρα “BallTextMesh” με την διάμεσό μας (X) στο ίδιο σημείο με μεγαλύτερο μέγεθος. Ακολούθως εμφανίζονται όλα τα στοιχεία που είναι < X και > X 5-5 σε σταθερή ταχύτητα στα αντίστοιχα δοχεία που έχουμε εμφανίσει. Μόλις ολοκληρωθεί η εμφάνισή τους, εμφανίζουμε σε text τον αριθμό των στοιχείων κάθε δοχείου κάτω από το ανάλογο δοχείο.



Visual Selection Algorithm – Βήμα 6-8 – Εύρεση διάμεσου των διαμέσων και διαμοίραση - Σχήμα 5.14

Ως επακόλουθο έχουμε τα βήματα 9-11 (Σχήμα 5.15). Παρατηρούμε ότι το υφιστάμενο Index που χρησιμοποιεί ο αλγόριθμος (39) είναι μεγαλύτερος από τον αριθμό των μικρότερων του $X + 1$ ($r+1 = 28$). Συνεπώς θα χρησιμοποιήσουμε ως νέο υφιστάμενο Index το προηγούμενό μας πλην των αριθμών των μικρότερων πλην 1 ($39 - 27 - 1 = 11$) και ως νέο πίνακα τα στοιχεία που είναι μεγαλύτερα της διάμεσου των διαμέσων μας (52). Έτσι ο

αλγόριθμος μας επαναλαμβάνεται «αναδρομικά» και πάμε πίσω στο βήμα 1) με νέο Index το 11 και νέο πίνακα αυτό που έχουμε προαναφέρει.



Visual Selection Algorithm – Βήμα 6-11 - Σχήμα 5.15

Ο αλγόριθμος συνεχίζει να επαναλαμβάνεται μέχρι να βρεθεί το αποτέλεσμα μας όπως έχουμε περιγράψει στην υλοποίησή του. Μόλις βρεθεί το αποτέλεσμα, εμφανίζεται μπροστά στο χρήστη όπως φαίνεται στο πιο κάτω σχήμα μαζί με πυροτεχνήματα ("Fireworks.cs" + [24]) που έχουν μικρή χρονική διάρκεια (Σχήμα 5.16).



Visual Selection Algorithm – Παρουσίαση αποτελέσματος- Σχήμα 5.16

5.4 Δυσκολίες, Τρόπος Επίλυσης και Διευκρινήσεις

Σε αυτό τον αλγόριθμο ανακαλύψαμε ένα πρόβλημα που επικρατούσε στην γραμμική παρεμβολή μας (Lerp) όταν εμφανίζουμε τις σφαίρες. Όταν έχουμε μόνο 1 σφαίρα, δεν εμφανίζεται σωστά στα δοχεία και γενικά οπουδήποτε χρησιμοποιούμε γραμμική παρεμβολή. Έτσι δημιουργήσαμε ένα κομμάτι κώδικα που ελέγχει αυτή τη συγκεκριμένη περίπτωση. Όταν έχουμε 1 σφαίρα, την βάζουμε απλώς στο κέντρο μεταξύ της αρχικής και τελικής θέσης ($(\text{αρχική θέση} + \text{τελική θέση}) / 2$) (Σχήμα 5.17).

```
if (number_array.Count == 1)
{
    // Set the text inside the ball.
    placeable_object.GetChild(0).GetComponent<TextMeshPro>().text = number_array[0] + "";

    // Instantiate the new transform 3d object
    Transform instantiatedObject =
        Instantiate(
            placeable_object,
            new Vector3((start_pos.x + end_pos.x) / 2, start_pos.y, start_pos.z),
            Quaternion.LookRotation(Vector3.forward));

    // Add item to array
    object_array.Add(instantiatedObject);
}
```

Γραμμική παρεμβολή 1 αντικειμένου - Σχήμα 5.17

Η περιστροφή των αντικειμένων πρέπει να είναι η ίδια για όλα τα αντικείμενα που τοποθετούνται σε γραμμική παρεμβολή. Μερικά 3D αντικείμενα μπορεί να έχουν 90 μοίρες περιστροφή στο Z άξονα ή οπουδήποτε αλλού. Έτσι θα πρέπει να τα περιστρέφουμε σωστά να βλέπουν προς την κάμερά μας. Έτσι ορίζουμε την περιστροφή κάθε αντικειμένου που εμφανίζουμε να είναι «Quaternion.LookRotation(Vector3.forward)» που το ορίζει να βλέπει προς την κάμερά μας όπως φαίνεται πιο πάνω.

Δημιουργήσαμε δύο είδη τρισδιάστατων πινάκων. Ένα με σταθερό μέγεθος και ένα με μεταβλητό μέγεθος. Αυτό έγινε επειδή το μεταβλητό μέγεθος μπορεί να έχει μόνο ένα συγκεκριμένο ύψος για να φυλάξει τα αντικείμενα μέσα. Έτσι δεν μπορούμε να αλλάξουμε το μέγεθος του μεταβλητού πίνακα. Αντιθέτως, με το σταθερό μέγεθος 3D πίνακα, αν γνωρίζουμε τον αριθμό των αντικειμένων που θα τοποθετηθούν, μπορούμε να αλλάξουμε το μέγεθος του πίνακα όπως θέλουμε.

Παρατηρήσαμε ένα πρόβλημα καλώντας Coroutine μέσα σε Coroutine. Δεν μπορούμε να σταματήσουμε την Coroutine που καλεί την άλλη και έτσι δεν μπορούμε να σταματήσουμε τη ροή του προγράμματος καλώντας την άλλη. Ως αποτέλεσμα ο κώδικας ελέγχου για την παύση εκτέλεσης και το “WaitForSeconds” λειτουργούν μόνο στην υφιστάμενη Coroutine

και έτσι έχουμε κώδικα που επαναλαμβάνεται συνεχώς. Αυτό μετά από έρευνα έχει επιλυθεί. Μπορούμε να σταματήσουμε τη ροή της ρουτίνας μας και να «παραδώσουμε τη σκυτάλη» στην επόμενη εκτελώντας την ακόλουθη εντολή: **«yield return StartCoroutine(waitForCoroutine());»** όπου η «waitForCoroutine» μπορεί να είναι οποιαδήποτε Coroutine θέλουμε. Στη περίπτωση μας, γλυτώνουμε επανάληψη του πιο κάτω κώδικα για έλεγχο ροής με το ‘USE NEXT STEP’ toggle, “Speed” και “Step Forward” (Σχήμα 5.18).

```
/**
 * IMPORTANT **
 * ABILITY TO CALL A COROUTINE WITHIN A COROUTINE
 * yield return StartCoroutine(waitForCoroutine());
 * Hence avoid usage of the same code multiple times.
 */
public IEnumerator waitForCoroutine()
{
    if (use_step_toggle)
    {
        yield return new WaitUntil(() => step_forward || !use_step_toggle);
        step_forward = false;
    }
    else
        yield return new WaitForSeconds(speed);
}
```

waitForCoutine – Used in another coroutine- Σχήμα 5.18

Κεφάλαιο 6.

Δυναμικός Προγραμματισμός - Αλγόριθμος Υπολογισμού Σειράς Αριθμών Fibonacci

6.1 Εισαγωγή	62
6.2 Περιγραφή Αλγορίθμου	63
6.3 Λεπτομέρειες Υλοποίησης	68
6.3.1 Σχεδιασμός Πρωτοτύπου	68
6.3.2 Υλοποίηση 3D Δυαδικού Δένδρου	69
6.3.3 Υλοποίηση Διεπαφής	73
6.3.4 Αναπαράσταση και Εκτέλεση Αλγορίθμου	75
6.4 Δυσκολίες, Τρόπος Επίλυσής τους και Διευκρινήσεις	77

6.1 Εισαγωγή

Μια άλλη τεχνική προγραμματισμού εκτός από το “Διαίρει και Βασίλευε” που έχει χρησιμοποιηθεί στους προηγούμενους μας αλγόριθμους, είναι και η πιο ισχυρή και ευφυής τεχνική του Δυναμικού Προγραμματισμού [4,5]. Η διαφορά του με τις υπόλοιπες τεχνικές είναι ότι εδώ εξερευνούμε το χώρο όλων των πιθανών λύσεων του προβλήματός μας αποσυνθέτοντας προσεκτικά τα πράγματα σε μια σειρά από **υποπροβλήματα** και μετά δομούμε σωστές λύσεις για όλο και μεγαλύτερα υποπροβλήματα. Τυπικά αυτή η τεχνική δεν μπορεί κάποιος να την συνηθίσει εύκολα. Χρειάζεται αρκετή εξάσκηση μέχρι να νιώσει κάποιος εξοικειωμένος μαζί του.

Για να παρουσιάσουμε την βελτίωση που επιβάλλει ο δυναμικός προγραμματισμός στην πολυπλοκότητα και υλοποίηση ενός προβλήματος, θα παρουσιάσουμε ένα συγκεκριμένο

πρόβλημα με τρεις διαφορετικούς τρόπους υλοποίησης. Έχουμε επιλέξει μετά από συζήτηση με τον επιβλέποντα καθηγητή, τον αλγόριθμο Fibonacci ως παράδειγμα. Αρχικά θα τον παρουσιάσουμε υλοποιημένο στην κλασική του αναδρομική μορφή, στην συνέχεια στην μορφή που περιλαμβάνει απομνημόνευση και τέλος χρησιμοποιώντας δυναμικό προγραμματισμό.

Βασικός μας σκοπός είναι να παρουσιάσουμε τον χρόνο εκτέλεσης που χρειάζεται κάθε αλγόριθμος για δείξουμε την πραγματική χρονική πολυπλοκότητα κάθε υλοποίησης σε πράξη. Αρχικά μαζί με τον επιβλέπον καθηγητή έχουμε αποφασίσει να παρουσιάσουμε τις αναδρομικές κλήσεις που εκτελεί ο αναδρομικός αλγόριθμος δια μέσου ενός τρισδιάστατου δυαδικού δένδρου. Στην συνέχεια ο χρήστης θα μπορεί να παρακολουθήσει κάθε αλγόριθμο εν ώρα εκτέλεσης με κάθε υλοποίηση σε ξεχωριστές οθόνες. Μετά το τέλος της εκτέλεσης του κάθε αλγορίθμου, θα φαίνεται ο ακριβής συνολικός χρόνος εκτέλεσης που χρειάστηκε.

Η αναδρομική εκτέλεση του αλγορίθμου είναι και η πιο χρονοβόρα και μεγαλύτερη σε πολυπλοκότητα. Έτσι θα αναμένουμε μια εμφανισιακά αργή εκτέλεση του κώδικα με την τεχνική αυτή. Τώρα για τον αλγόριθμο με απομνημόνευση και με χρήση δυναμικού προγραμματισμού θα αναμένουμε το τι θα μας δείξουν τα τελικά αποτελέσματα μετά την ολοκλήρωση της υλοποίησης μας. Λογικά ο αλγόριθμος απομνημόνευσης θα είναι πιο αργός από αυτό του δυναμικού προγραμματισμού καθώς περιλαμβάνει ακόμη αναδρομικές κλήσεις.

6.2 Περιγραφή Αλγορίθμου

[10,13] Οι αριθμοί Fibonacci χρησιμοποιούν την πιο κάτω αναδρομική εξίσωση-φόρμουλα για να υπολογιστούν:

$$F(n) = F(n-1) + F(n-2), \text{ (για } n \geq 2)$$

$$F(0) = 0, \text{ (για } n=0)$$

$$F(1) = 1, \text{ (για } n=1)$$

Η πιο πάνω αναδρομική εξίσωση μπορεί εύκολα να μετατραπεί σε αναδρομική συνάρτηση στην C#. Έτσι έχουμε δημιουργήσει την πιο κάτω συνάρτηση (Σχήμα 6.1). Ο λόγος που χρησιμοποιούμε αριθμούς τύπου “Double” θα επεξηγηθεί αργότερα στο υποκεφάλαιο με τις δυσκολίες του αλγορίθμου μας.

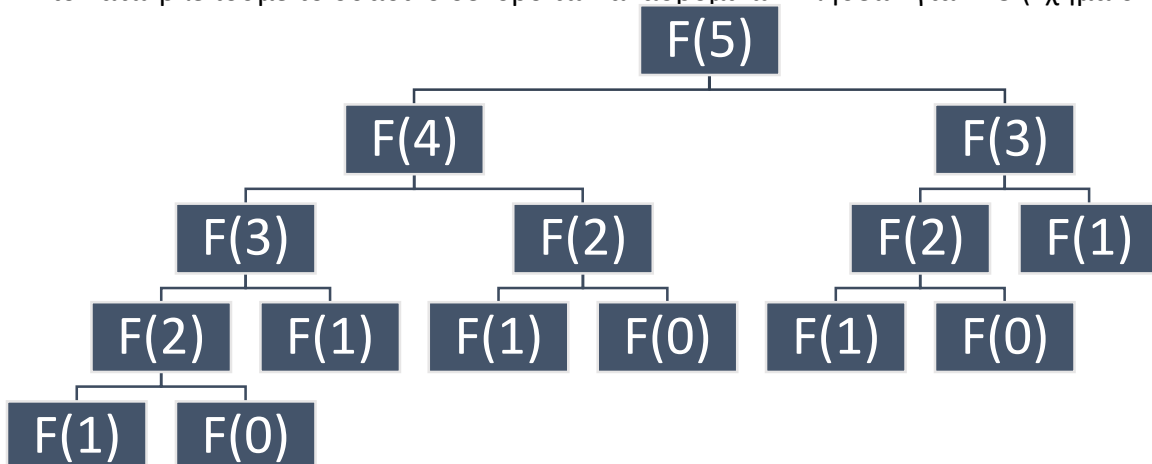
```
1  /*Recursive Fibonacci function*/
2  public double recursiveFib(double n)
3  {
4      // For the base cases: F(0) = 0, F(1) = 1
5      if (n <= 1)
6      {
7          return n;
8      }
9      else
10     {
11         // F(n) = F(n-1) + F(n-2)
12         double n1 = recursiveFib(n - 1);
13         double n2 = recursiveFib(n - 2);
14         return n1 + n2;
15     }
16 }
```

Recursive Fibonacci - Σχήμα 6.1

Ο συνολικός αριθμός κλήσεων θα μεγαλώνει εκθετικά και έτσι δεν θα έχουμε πολωνιμικό χρόνο εκτέλεσης. Συγκεκριμένα έχουμε ότι η χρονική μας πολυπλοκότητα είναι

$$T(n) = T(n-1) + T(n-2) + c \geq 2 \cdot T(n-2) + c \dots \geq 2^k T(n-2 \cdot k) + c(2^{k-1} + 2^{k-2} + \dots + 2 + 1) = \Omega(c2^{n/2})$$

Πιο κάτω βλέπουμε το δυαδικό δένδρο των αναδρομικών κλήσεων για $n=5$ (Σχήμα 6.2).



Recursive Fibonacci Tree for n=5 - Σχήμα 6.2

Παρατηρούμε ότι πολλές περιπτώσεις ξαναυπολογίζονται στον αναδρομικό και έτσι έχουμε μεγάλη επίπτωση στην χρονική και χωρική μας πολυπλοκότητα. Για παράδειγμα το υπόδενδρο $F(3)$ υπολογίζεται δύο φορές και το $F(2)$ τρεις φορές.

Για να βελτιώσουμε την αναδρομική υλοποίηση και για να μειώσουμε την πολυπλοκότητα θα εφαρμόσουμε αρχικά μια άλλη τεχνική. Δεν θα ξαναυπολογίζουμε τα υποπροβλήματα $F(k)$, όπου $k \leq n$ αν έχουν υπολογιστεί ξανά. Η τεχνική της αποθήκευσης τιμών που έχουν ήδη υπολογιστεί ονομάζεται **απομνημόνευση (Memoization)**. Η διαδικασία αυτή, χρησιμοποιώντας έναν πίνακα $MEM[0...1]$ που είναι αρχικά κενός, μετά από κάθε υπολογισμό του $F(J)$, αποθηκεύεται στη θέση $MEM[J]$ η τιμή που υπολογίζεται. Έτσι θα ελαττώσουμε σημαντικά τον χρόνο εκτέλεσης ανακτώντας τιμές που έχουν ήδη υπολογιστεί. Ως αποτέλεσμα μετατρέψαμε τον πιο πάνω αλγόριθμο σε αλγόριθμο με χρήση απομνημόνευσης αναδρομής (Σχήμα 6.3).

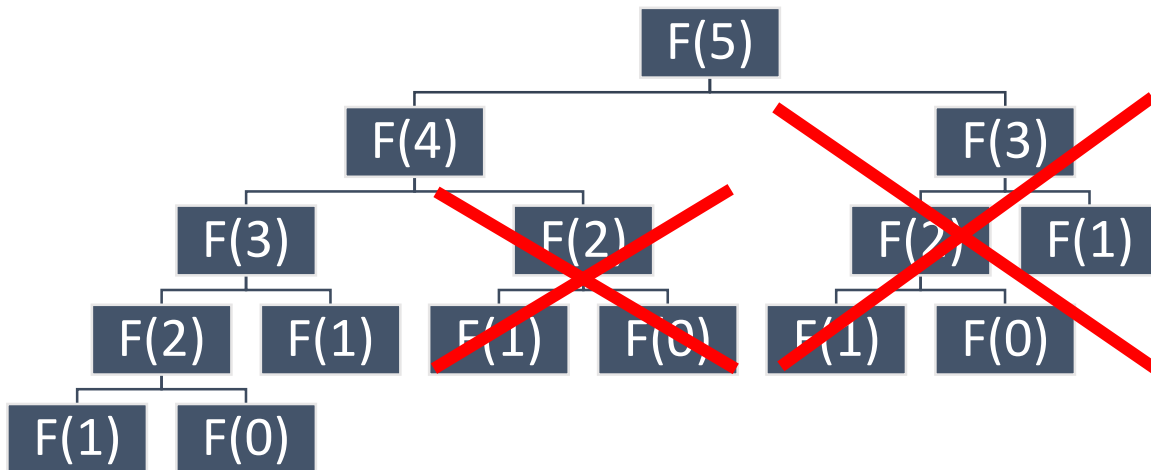
```
1 // Memoization array used to store already
2 // prepared results (Global access)
3 public double[] mem_fib;
4 // Memoization Fibonacci function. Call this to use
5 public double memoizationFibonacci(double n)
6 {
7     // Initialize the memoization table with -1 values.
8     mem_fib = new double[(int)n + 1];
9     for (int i = 0; i <= n; i++)
10     {
11         mem_fib[i] = -1;
12     }
13     return memoizationFib(n);
14 }
15 // Recursive function with memoization
16 private double memoizationFib(double n)
17 {
18     // If the current recursive call is stored in the memoization
19     // array then retrieve it and return the stored answer.
20     if (mem_fib[(int)n] != -1)
21     {
22         return mem_fib[(int)n];
23     }
24     // else return the base case if  $F(0) = 0$ ,  $F(1) = 1$ 
25     else if (n <= 1)
26     {
27         return n;
28     }
29     else
30     {
31         // Calculate new result and store into memoization array
32         double fib_res = memoizationFib(n - 1) + memoizationFib(n - 2);
33         mem_fib[(int)n] = fib_res;
34         return fib_res;
35     }
36 }
```

Memoization Fibonacci - Σχήμα 6.3

Καλώντας την συνάρτηση $memoizationFibonacci(n)$ για το επιθυμητό n , εκτελείται η διαδικασία με απομνημόνευση. Αρχικά όλες οι τιμές του πίνακα που θα αποθηκεύσουμε όλα τα αποτελέσματά μας (mem_fib) ορίζονται ως -1. Στην συνέχεια καλούμε την ίδια αναδρομική συνάρτηση όπως και πριν με την διαφορά ότι αν έχουμε τιμή η οποία έχει ξαναυπολογιστεί (έστω K) τότε θα επιστρέφουμε το $mem_fib[K]$, αλλιώς καλούμε κανονικά αναδρομικά όπως και στον προηγούμενό μας αλγόριθμο αλλά με την επιστροφή του

αποτελέσματός μας, το φυλάγουμε στη θέση `mem_fib[K]`. Για τις βασικές μας περιπτώσεις δεν χρειάζεται να φυλάξουμε τίποτα (για $n=0$ ή $n=1$).

Για να το δούμε τη διαφορά με προηγουμένως έχουμε το πιο κάτω δυαδικό δένδρο, αυτή τη φορά με την χρήση απομνημόνευσης (Σχήμα 6.4).



Memoization Fibonacci Tree for $n=5$ - Σχήμα 6.4

Από τις υλοποιήσεις μας έχουμε ότι εκτελείται πάντα πρώτα το $F(n-1)$ και την συνέχεια το $F(n-2)$ έτσι έχουμε το δένδρο μας να δημιουργείται από το αριστερότερο παιδί πρώτα και ύστερα προς τα δεξιά. Συνεπώς οι τιμές του $F(2)$ και $F(3)$ έχουν ήδη υπολογιστεί από το αριστερότερο υπόδενδρο. Έτσι όταν φτάνει η στιγμή να ξαναυπολογιστούν, ο αλγόριθμος απλώς ανακτά το αποτέλεσμα που έχει αποθηκεύσει στον πίνακα απομνημόνευσης και γι' αυτό το λόγο δεν υπολογίζονται τα πιο πάνω διαγραμμένα υπόδενδρα αργότερα στην εκτέλεση.

Νέος χρόνος εκτέλεσης με χρήση απομνημόνευσης:

$$T(n) = T(n-1) + c = O(cn), T(n) = O(n^2)$$

Τέλος κάνοντας χρήση δυναμικού προγραμματισμού με την μέθοδο “από Κάτω προς τα Πάνω” (Bottom-Up) έχουμε τροποποιήσει ακόμη μια φορά τον αλγόριθμό μας προς το καλύτερο. Για τον σχεδιασμό ενός αλγόριθμου με χρήση δυναμικού προγραμματισμού θα πρέπει αρχικά να χαρακτηρίσουμε τη δομή της βέλτιστης μας λύσης, μετά να ορίσουμε τον

αναδρομικό ορισμό βέλτιστης λύσης και τέλος να υπολογίσουμε τη βέλτιστη λύση και τη τιμή της “από Κάτω προς τα Πάνω”.

Ο αλγόριθμος Fibonacci είναι απλός αλγόριθμος και δεν χρειάζεται να αναπτύξουμε την ιδέα της βέλτιστης λύσης καθώς η λύση του πάντα είναι μία και καθορισμένη.

Έχουμε n υποπροβλήματα: $F(0), F(1) \dots F(n)$. Έχουμε την πιο κάτω αναδρομική εξίσωση για τον υπολογισμό των τιμών κάθε υποπροβλήματος.

$$F[i] = \begin{cases} 0, & \text{αν } i = 0 \\ 1, & \text{αν } i = 1 \\ F[i - 1] + F[i - 2], & \text{αν } i > 1 \text{ και } i \leq n \end{cases}$$

Έτσι έχουμε καταλήξει στον πιο κάτω επαναληπτικό αλγόριθμο ο οποίος υπολογίζει άμεσα τα στοιχεία μας αντί να χρησιμοποιούμε απομνημόνευση (Σχήμα 6.5).

```
1  /*Fibonacci using Dynamic Programming (iterative)*/
2  public double iterativeFib(double n)
3  {
4      // Create a new array containing all the
5      // calculated results
6      double[] F = new double[(int)n + 1];
7      // Store the base case results F(0) = 0, F(1) = 1
8      F[0] = 0;
9      F[1] = 1;
10     // Using a bottom-up strategy, calculate
11     // the results from F(2) to F(N)
12     for (int i = 2; i <= n; i++)
13     {
14         F[i] = F[i - 1] + F[i - 2];
15     }
16     // Return the final result
17     // calculated at the nth position of the array
18     return F[(int)n];
19 }
```

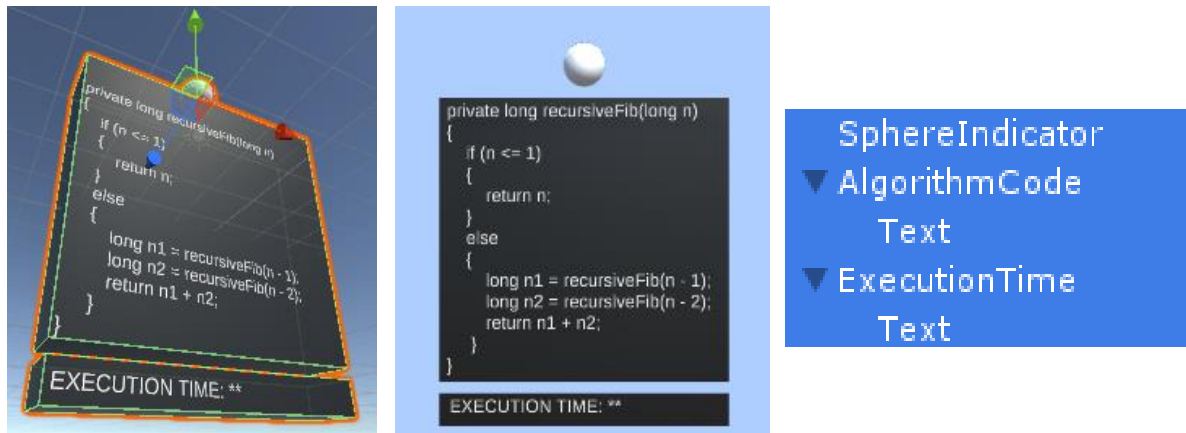
Iterative Fibonacci (Dynamic Programming) - Σχήμα 6.5

Εμείς ψάχνουμε για τη λύση μας για το $F(n)$. Η τιμή αυτή πολύ απλά θα βρίσκεται στο $F[n]$ μετά την ολοκλήρωση εκτέλεσης του νέου μας αλγορίθμου. Με ένα απλό επαναληπτικό βρόγχο παρατηρούμε ότι η πολυπλοκότητά μας τόσο χρονικά όσο και χωρικά έχει μειωθεί σε **$O(n)$** καθώς εκτελεί ρητά n επαναλήψεις οι οποίες η κάθε μια δαπανά σταθερό χρόνο.

6.3 Λεπτομέρειες Υλοποίησης

6.3.1 Σχεδιασμός Πρωτοτύπου

Στόχος μας σε αυτό το σημείο είναι να παρουσιάσουμε απλά μια ζωντανή εκτέλεση των αλγορίθμων που έχουμε περιγράψει και επιπλέον την εμφάνιση ενός δυαδικού δένδρου για τον αναδρομικό. Έχουμε υλοποιήσει τα πιο κάτω βασικά 3D αντικείμενα τα οποία θα χρησιμοποιούμε κάθε φορά για την εμφάνιση του αντίστοιχου αλγόριθμου και των αποτελεσμάτων τους (Σχήμα 6.6).



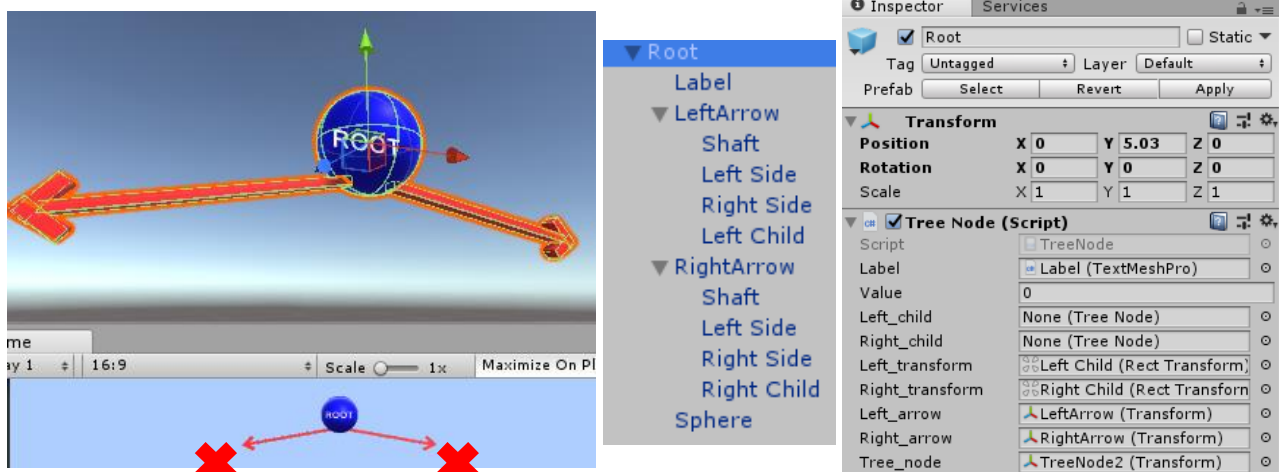
Basic Algorithm 3D view - Σχήμα 6.6

Όπως βλέπουμε στα πιο πάνω σχήματά μας, η εμφάνισή μας αποτελείται από δύο 3D κουτιά με μαύρο χρώμα και μια απλή σφαίρα από πάνω τους. Στο πρώτο κουτί (AlgorithmCode) θα εμφανίζεται ο ανάλογος κώδικας που τρέχει εκείνη τη στιγμή όπως φαίνεται στο παράδειγμά μας. Πιο κάτω θα εμφανίζεται ο τελικός χρόνος εκτέλεσης μόλις ολοκληρωθεί ο ανάλογος αλγόριθμος. Η εμφάνιση του κειμένου θα γίνεται όπως γίνεται και σε προηγούμενές μας υλοποιήσεις, με χρήση TextMeshPro. Στην σφαίρα “SphereIndicator” θα εφαρμόζεται μια απλή εναλλαγή χρώματος από κόκκινο σε άσπρο εφόσον εκτελείται ο κώδικας και με την ολοκλήρωση θα πρασινίζει.

Οι κώδικες που θα εμφανίζονται φαίνονται στην περιγραφή του αλγορίθμου που έχουμε προαναφέρει. Στην συνέχεια θα περιγράψουμε την υλοποίηση του τρισδιάστατου δυαδικού δένδρου που θα εφαρμόσουμε στον αναδρομικό Fibonacci.

6.3.2 Υλοποίηση 3D Δυαδικού Δένδρου

Για την παρουσίαση όλων των αναδρομικών κλήσεων που εκτελεί ο αλγόριθμος Fibonacci έπρεπε να παρουσιάσουμε ένα νέο δυναμικά δημιουργημένο δυαδικό δένδρο [12]. Έχουμε κατασκευάσει μια νέα δομή δεδομένων που έχει τις ίδιες ιδιότητες με ένα δυαδικό δένδρο με την εξαίρεση ότι καθώς δημιουργείται και καθώς προστίθενται ή διαγράφονται νέα παιδιά/φύλλα, εμφανίζονται ταυτόχρονα σε τρισδιάστατη αναπαράσταση στο χρήστη.



Root GameObject for 3D Binary Tree - Σχήμα 6.7

Όπως βλέπουμε στο Σχήμα 6.7 έχουμε δημιουργήσει ένα καινούριο 3D GameObject prefab με όνομα “Root” το οποίο περιλαμβάνει τα ακόλουθα:

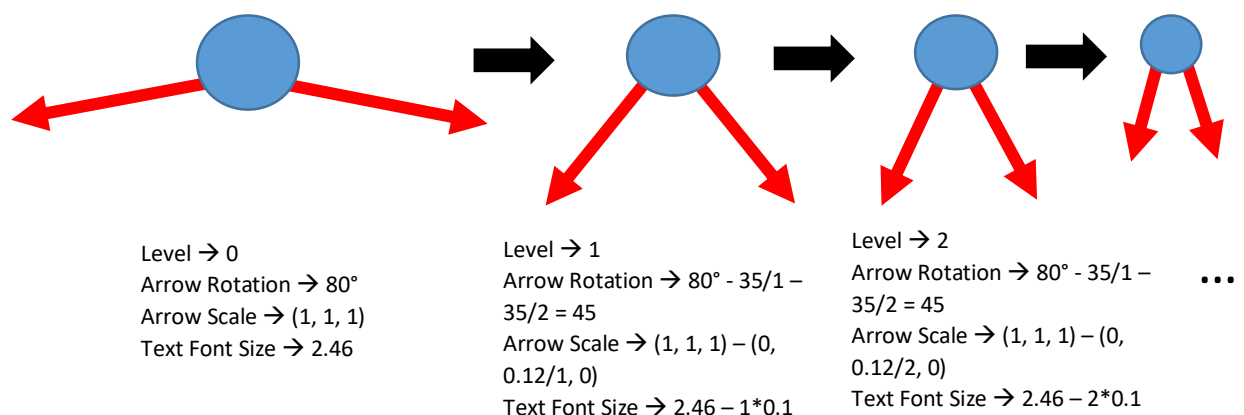
- 3D Σφαίρα (**Sphere**): Περιλαμβάνει ένα **Label** στο οποίο μπορούμε να καθορίσουμε ότι όνομα θέλουμε και επίσης μπορούμε να αλλάξουμε το χρώμα της όπως επιθυμούμε.
- Αριστερό και Δεξί βέλος (**LeftArrow, RightArrow**): Με συνδυασμό 3D κουτιών έχουμε δημιουργήσει το σχήμα ενός τόξου τα οποία αρχικά έχουν κλίση 80 μοίρες αριστερά και δεξιά αντίστοιχα (Θα επεξηγηθεί αργότερα ο λόγος). Επίσης και τα δύο βέλη έχουν κόκκινο χρώμα και μπορείς να αλλάξεις τα χρώματά τους.
- Στην άκρη κάθε βέλους βρίσκεται ένα νοητό GameObject (κοκκινιά «X» στο σχήμα μας) το οποίο περιλαμβάνει μόνο μια τοποθεσία (**LeftChild** και **RightChild** για τα αντίστοιχα βέλη). Καθώς μετακινούνται και αλλάζουν μέγεθος τα βέλη, μετακινείται

μαζί τους και αυτή η τοποθεσία. Αργότερα οι δύο αυτές τοποθεσίες θα χρησιμοποιηθούν για την εμφάνιση των παιδιών του κάθε κόμβου του δένδρου μας.

- Script **“TreeNode.cs”** (Παράρτημα Ε): Ελέγχει κάθε λειτουργία του κόμβου μας όπως δημιουργία και πρόσθεση αριστερού και δεξιού παιδιού, αλλαγή χρωμάτων και αλλαγή μεγέθους και κλίσης των βελών.

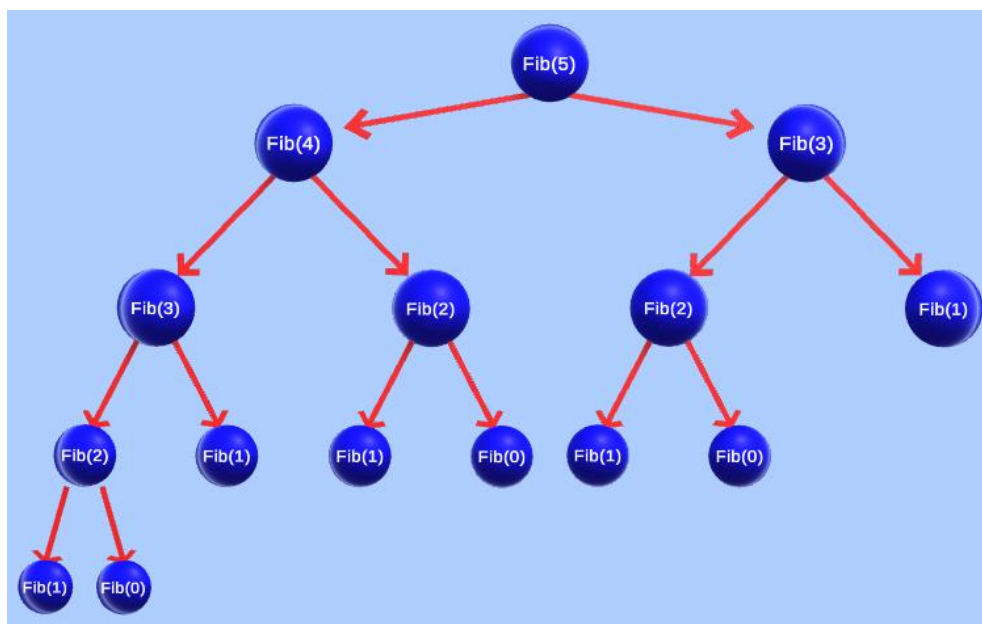
Η εμφάνιση νέων κόμβων γίνεται αναδρομικά. Δηλαδή κάθε φορά που δημιουργούμε νέο παιδί, το παιδί αυτό θα είναι τύπου “Root” και θα έχει τις ίδιες ιδιότητες όπως έχουμε περιγράψει στο Σχήμα 6.7. Για να μην επηρεάζονται όμως οι υπόλοιποι κόμβοι κατά τη δημιουργία ενός νέου, έχουμε δημιουργήσει ένα πιστό αντίγραφο prefab του “Root” με όνομα “TreeNode2” το οποίο με τη σειρά του χρησιμοποιεί το “Root” ως prefab. Έτσι κάθε παιδί που δημιουργείται δεν επηρεάζεται από τις ιδιότητες του πατέρα του.

Το δένδρο που θα εμφανίζεται θα πρέπει να έχει μικρότερα μεγέθη κόμβων καθώς οδεύει προς τα κάτω και ταυτόχρονα μικρότερα μεγέθη τόξων με κλίση προς τα μέσα (Σχήμα 6.8). Το έχουμε πετύχει με χρήση μιας επιπλέον τιμής που ορίζει το βάθος στο οποίο βρισκόμαστε σε κάθε κόμβο του δένδρου μας. Με τη χρήση της συνάρτησης “SetLevel” του “TreeNode.cs”, καθώς ορίζουμε το βάθος ενός κόμβου, αλλάζουν ταυτόχρονα τα πιο κάτω: Για $Level=N \rightarrow$ Κλίση τόξου $= (35/1 + \dots + 35/N)$, $\rightarrow$ Μέγεθος τόξου $= 0.12/N$, $\rightarrow$ Μέγεθος γραμμάτων $= N*0.1$. Οι συγκεκριμένοι αριθμοί που χρησιμοποιούνται είναι βασισμένοι σε πειράματα πολλών δοκιμών. Αυτός είναι και ο λόγος που ορίσαμε εξ αρχής την κλίση των βελών ως 80 μοίρες. Λεπτομερείς υπολογισμοί φαίνονται στο σχήμα μας.



Node and Arrow change based on Tree Level - Σχήμα 6.8

Όταν προστίθεται νέο παιδί, αυξάνεται το βάθος του καινούριου κόμβου κατά 1, εφαρμόζονται οι αλλαγές που έχουμε περιγράψει και επιπλέον έχουμε ορισμένες επιπρόσθετες ενέργειες. Εμφανίζεται το αντίστοιχο τόξο στο οποίο ανήκει το παιδί και κάνουμε Instantiate μια νέα σφαίρα με την τιμή του παιδιού στην αντίστοιχη θέση όπως βλέπουμε στα «X» στο Σχήμα 6.7. Όταν γίνεται Instantiate ο νέος κόμβος, το μέγεθός του μικραίνει ανάλογα με το βάθος στο οποίο βρίσκεται (Νέο μέγεθος $= (0.12/i, 0.12/i, 0.12/i)$ για κάθε $i \geq 1$ και $i \leq \text{level}$).



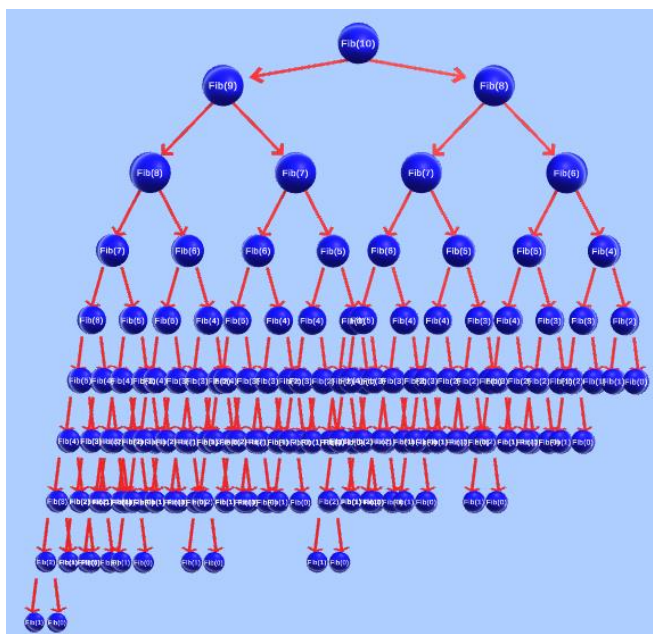
3D Binary Fibonacci Tree for n=5 - Σχήμα 6.9

Όπως βλέπουμε στο παράδειγμά μας στο πιο πάνω σχήμα μας (Σχήμα 6.9) έχουμε καταλήξει σε αυτή τη τελική μορφή για $n=5$.

Οι αλλαγές αυτές των τιμών συμβαίνουν μέχρι ένα συγκεκριμένο επίπεδο όπου και αρχίζουν και παραμένουν σταθερές. Από ένα σημείο και μετά ο χρήστης δεν θα ενδιαφέρεται να παρακολουθήσει τους υπολογισμούς. Για παράδειγμα ο Fibonacci με είσοδο 200 θα έχει τόσες πολλές αναδρομές που δεν θα είναι και τόσο ευδιάκριτες στο μάτι με την παρουσίαση ενός δένδρου και θα μας ενδιαφέρει μόνο το μέγεθος του αριθμού των εκτελέσεων. Έτσι το δένδρο μας θα εμφανίζεται αλλά σε πιο μικρό μέγεθος με όσο πιο μεγάλο n έχουμε.

Ο τρόπος που το έχουμε πετύχει αυτό είναι με τις αλλαγές παραμέτρων της κάμεράς μας. Πρώτα απ' όλα έχουμε ορίσει τον τύπο της κάμερας ως ορθογραφικό. Συνεπώς όλα τα αντικείμενα που βρίσκονται στην οθόνη φαίνονται να έχουν το ίδιο μέγεθος όταν βρίσκονται στην ίδια απόσταση μακριά από τη κάμερα. Αυτό έχει ως αποτέλεσμα την σωστή και ευδιάκριτη παρουσίαση του δένδρου μας. Στην συνέχεια αλλάζοντας τη θέση και το μέγεθος της κάμερας, πετύχαμε την πλήρης εμφάνιση οποιουδήποτε δένδρου στο χρήστη. Συγκεκριμένα, αν έχουμε $n > 6$ τότε αφαιρούμε $-0.7 \cdot (n-6)$ από τον άξονα Y της θέσης της κάμερας για να την μετακινήσουμε προς τα κάτω και να φαίνεται το δένδρο πιο ψηλά στην οθόνη. Ταυτόχρονα αυξάνουμε το μέγεθος της ορθογραφικής κάμερας κατά $0.8 \cdot (n-6)$. Οι αριθμοί που χρησιμοποιήσαμε βασίζονται σε πολλά πειράματα που κάναμε. Ο κώδικάς μας βρίσκεται στο **DynamicProgramming.cs** (Παράρτημα Ε).

Έτσι μπορούμε να δούμε την εμφάνιση ενός μεγαλύτερου δένδρου με βάση όλα αυτά που έχουμε αναφέρει. (Σχήμα 6.10)



3D Binary Fibonacci Tree for n=10 - Σχήμα 6.10

Όπως παρατηρούμε, η κάμερα έχει μετακινηθεί και αλλάξει σε σημείο όπου φαίνεται ακριβώς ολόκληρο το δένδρο. Επιπρόσθετα, από το επίπεδο 5 και μετά βλέπουμε ότι τα βέλη και οι σφαίρες έχουν σταματήσει να αλλάζουν και παραμένουν με τα ίδια χαρακτηριστικά. Ο χρήστης θα είναι ικανοποιημένος.

Η υλοποίηση της εμφάνισης ενός Fibonacci δένδρου έχει υλοποιηθεί στο script “**FibonacciBinary3DTreeController.cs**” (Παράρτημα Ε). Κάνοντας απλή χρήση στοίβας, έχουμε μετατρέψει την αναδρομική μας συνάρτηση Fibonacci σε επαναληπτική (createFibonacciTree coroutine) με τέτοιο τρόπο ώστε να φαίνονται ακριβώς κάθε φορά που εμφανίζεται νέα κλήση. Η τελική μας υλοποίηση αυτής της coroutine φαίνεται στο Σχήμα 6.11.

```
/**
 * createFibonacciTree
 * The following function creates a 3D binary fibonacci tree
 * using the number n as the base number.
 */
public IEnumerator createFibonacciTree(int n)
{
    root.setLevel(0);
    root.setValue(n);
    root.setText("Fib(" + n + ")");
    yield return new WaitForSeconds(1f);

    Stack<TreeNode> fibStack = new Stack<TreeNode>();
    fibStack.Push(root);

    while (fibStack.Count != 0)
    {
        // current Number to create it's children
        TreeNode currNode = fibStack.Pop();
        float currValue = currNode.getValue();
        if (currValue <= 1f)
        {
            //if the current number is one of the base cases (0 or 1) then that's a leaf.
            continue;
        }

        // Left and right child values
        float n1 = currValue - 1;
        float n2 = currValue - 2;

        // This is the left child of currN
        currNode.addLeftChild("Fib(" + n1 + ")", n1);
        yield return new WaitForSeconds(speed);

        // This is the right child of currN
        currNode.addRightChild("Fib(" + n2 + ")", n2);
        yield return new WaitForSeconds(speed);

        // Add the right child first to the stack
        fibStack.Push(currNode.right_child);

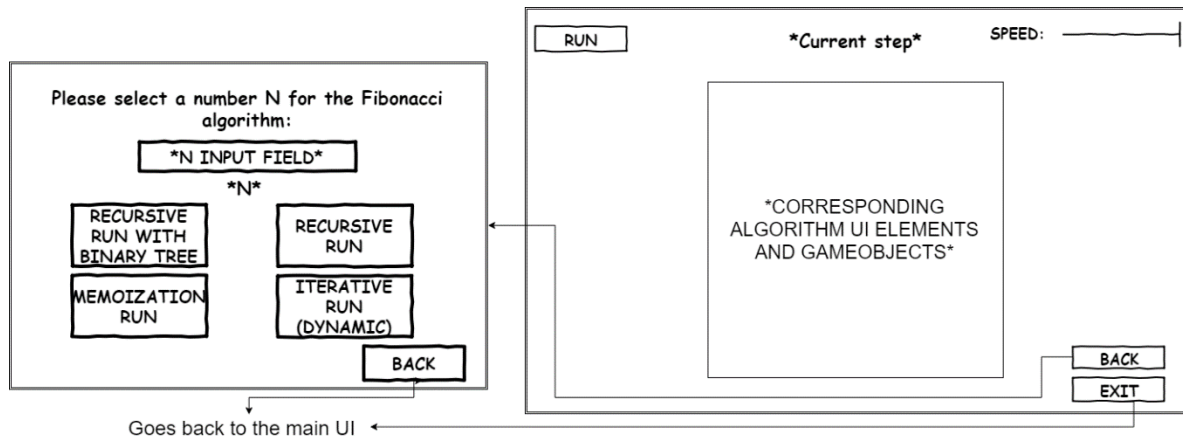
        // Then add the left and also remember all the left
        // childs by adding to the other stack
        fibStack.Push(currNode.left_child);
    }
}
```

createFibonacciTree Coroutine - Σχήμα 6.11

6.3.3 Υλοποίηση Διεπαφής

Όπως και με κάθε υλοποίηση μας μέχρι τώρα, τα αρχικά προσχέδια της διεπαφής μας γίνονται αρχικά στο χαρτί. Ακολουθώντας τα σχεδιάζουμε λίγο πιο λεπτομερώς ηλεκτρονικά με για να δούμε μια πρώτη τους εμφάνιση πριν υλοποιηθούν στο Unity. Στο Σχήμα 6.12, φαίνεται ο τρόπος που δεχόμαστε μια είσοδο ενός αριθμού n για τον Fibonacci αλγόριθμό μας και ο τρόπος που συνδέονται οι διεπαφές μεταξύ τους. Ο χρήστης αρχικά καταχωρεί

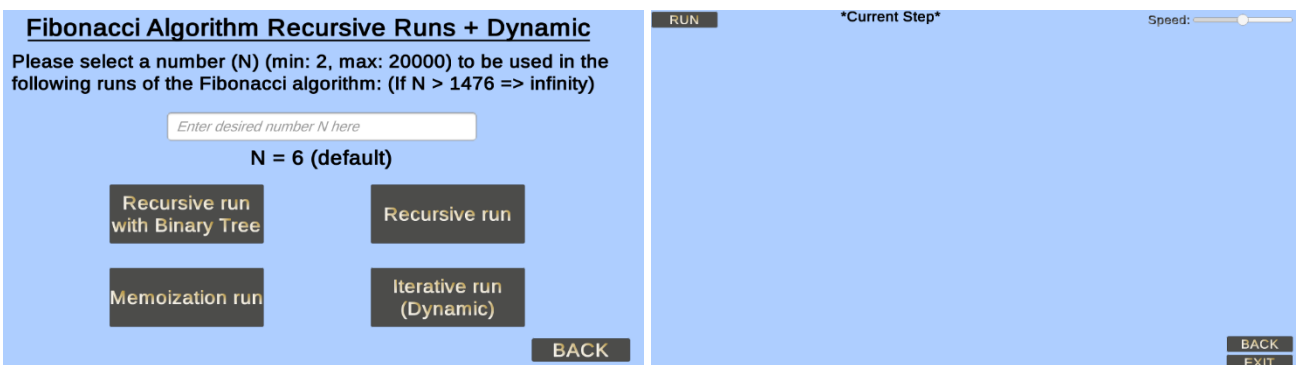
τον επιθυμητό αριθμό N στο πεδίο εισόδου που φαίνεται στον πρώτο καμβά μας. Η προεπιλεγμένη τιμή είναι το 6. Επίσης έχουμε ελάχιστη τιμή το 2 και μέγιστη το 20000.



Dynamic Programming Main UI Design - Σχήμα 6.12

Όταν καταχωρήσει την επιθυμητή του τιμή ο χρήστης επιλέγοντας το ανάλογο κουμπί για τον αλγόριθμο που επιθυμεί να τρέξει, το πρόγραμμά μας προχωρεί στον επόμενο καμβά. Εκεί έχουμε τις επιλογές να τρέξουμε τον αλγόριθμο με το κουμπί “RUN”, να καθορίσουμε την ταχύτητα εκτέλεσής του αν χρειάζεται με το “SPEED” slider και τέλος να πάει πίσω στον προηγούμενο καμβά ή πίσω στην κεντρική μας διεπαφή με το κουμπί “BACK” και “EXIT” αντίστοιχα.

Η τελική εμφάνιση της διεπαφής μας είναι η ακόλουθη (Σχήμα 6.13):



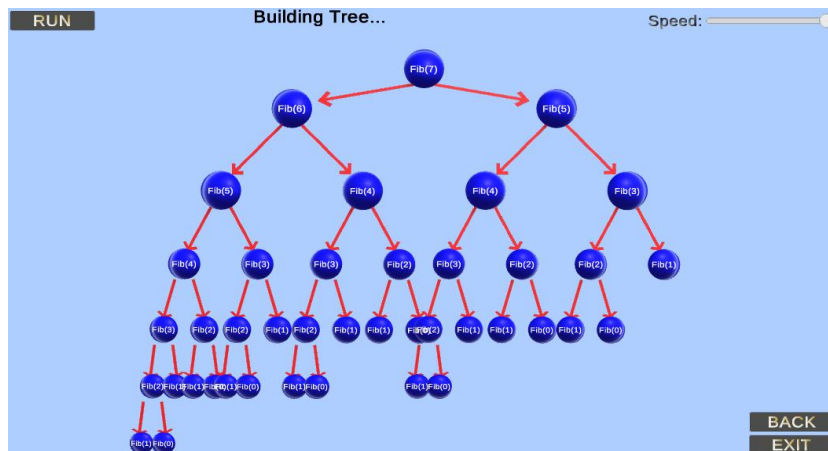
Dynamic Programming Main UI (Final) - Σχήμα 6.13

Εκτός από την ελάχιστη τιμή και μέγιστη τιμή που είναι το 2 και το 20000 αντίστοιχα, υπάρχει και μια ενδιάμεση τιμή. Η τιμή αυτή είναι το 1476. Μετά από έλεγχο εκτελέσεων όλων των αλγορίθμων Fibonacci, παρατηρήσαμε ότι ο αλγόριθμος παρουσιάζει αποτέλεσμα μέχρι εκείνη την τιμή. Για τιμές μεγαλύτερες του 1476, παρουσιάζεται

αποτέλεσμα “Infinity” δηλαδή άπειρο. Αυτό οφείλεται στο ότι έχουμε φτάσει σε υπερχείλιση του τύπου δεδομένων “Double” μετά από μεγάλες πράξεις Fibonacci. Η χρήση του τύπου “Double” θα επεξηγηθεί στις δυσκολίες.

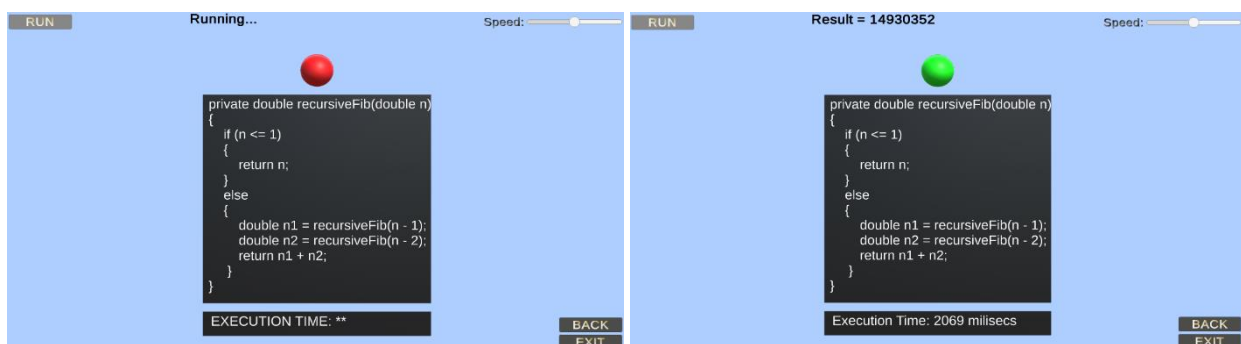
6.3.4 Αναπαράσταση και Εκτέλεση Αλγορίθμου

Προχωράμε τώρα στην εκτέλεση του αλγορίθμου μας. Για είσοδο $N=7$ και επιλογή του “Recursive run with Binary Tree” παραπεμπόμαστε στην οθόνη εκτέλεσης με εμφάνιση δένδρου. Επιλέγοντας το κουμπί “RUN” αρχίζει να εμφανίζεται το δένδρο για το $F(7)$ (Σχήμα 6.14) με τον τρόπο που έχουμε περιγράψει προηγουμένως στο Σχήμα 6.11 με σειρά από το αριστερότερο κλαδί στο δεξιότερο. Επίσης ο χρήστης μπορεί να αλλάξει την ταχύτητα εμφάνισης του δένδρου μετακινώντας το “Speed” slider.



Εμφάνιση Δυαδικού Δένδρου Fibonacci(7) - Σχήμα 6.14

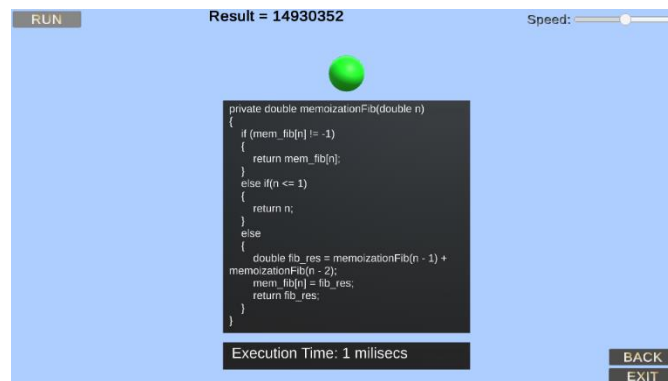
Στην συνέχεια πάμε σε άλλη εκτέλεση του αλγορίθμου. Αν καταχωρήσουμε $N=36$ και επιλέξουμε την επιλογή “Recursive Run”, θα εκτελέσουμε τον αναδρομικό αλγόριθμο Fibonacci(36) (Σχήμα 6.15). Επιλέγοντας το “RUN” εμφανίζονται όλα τα αντικείμενα που έχουμε προαναφέρει και ξεκινά η εκτέλεσή του.



Εκτέλεση Recursive Fibonacci (n=36) - Σχήμα 6.15

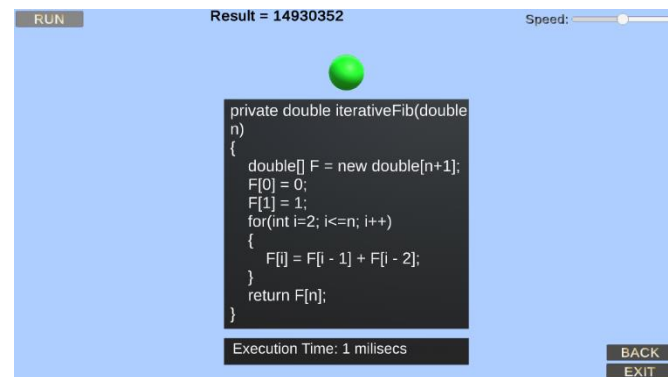
Η πάνω σφαίρα αρχίζει να αναβοσβήνει με κόκκινο και άσπρο χρώμα καθώς εκτελείται ο αναδρομικός αλγόριθμος που φαίνεται στο κάτω κουτί. Μετά την ολοκλήρωση της εκτέλεσής του, εμφανίζεται ο χρόνος εκτέλεσης στο κάτω κουτί που στην περίπτωση μας είναι στα 2069ms. Επίσης εμφανίζεται το αποτέλεσμα του αλγορίθμου στο πάνω κεντρικό κείμενο TextMeshPro που παρουσιάζει το βήμα που εκτελείται κάθε φορά. (Result = 14930352) Βλέπουμε ότι το αποτέλεσμά μας αυξάνεται κάθε φορά εκθετικά καθώς θεωρητικά ακόμη χρησιμοποιούμε μικρό N.

Χρησιμοποιώντας το ίδιο N (36) επιλέγουμε αυτή τη φορά την επιλογή εκτέλεσης με χρήση Memoization. Επιλέγοντας για ακόμη μια φορά το “Run” παρατηρούμε ότι ο αλγόριθμός μας έχει ολοκληρωθεί σε άμεσο χρόνο (σε μονάχα 1ms) (Σχήμα 6.16).



Εκτέλεση Memoization Fibonacci (n=36) - Σχήμα 6.16

Το ίδιο παρατηρούμε και στην εκτέλεση του αλγορίθμου με την χρήση δυναμικού προγραμματισμού (1ms) (Σχήμα 6.17). Το αποτέλεσμα εμφανίζεται το ίδιο για επαλήθευση ότι οι αλγόριθμοί μας εκτελούνται σωστά. Αυτά τα αποτελέσματα είναι αναμενόμενα καθώς ο χρόνος εκτέλεσης πέφτει από τον εκθετικό του αναδρομικού αλγορίθμου σε πολωνιμικό σε αυτό της απομνημόνευσης και γραμμικό με τον δυναμικό προγραμματισμό.



Εκτέλεση Iterative (Dynamic) Fibonacci (n=36) - Σχήμα 6.17

Για να καταφέρουμε να εφαρμόσουμε την αναβόσβηση της σφαίρας μας έπρεπε οι αλγόριθμοι που εκτελούσαμε εκείνη τη στιγμή να εκτελούνται στο παρασκήνιο. Έτσι κάθε φορά που εκτελούσαμε ένα αλγόριθμο, την εκτέλεσή του την βάζαμε σε ένα νέο Νήμα (Thread) [11]. Συνεπώς έως ότου ολοκληρωθεί το νήμα μας (**While(thread.IsAlive)**), η σφαίρα θα αλλάζει χρώμα κάθε 0.5 δευτερόλεπτα με τη χρήση της “WaitForSeconds” μέσα σε μια coroutine.

Επίσης για να καταγράψουμε τον χρόνο που χρειάστηκε να εκτελεστεί η ανάλογη συνάρτηση, χρησιμοποιήσαμε την ενσωματωμένη λειτουργία της C# “StopWatch”. Συγκεκριμένα έχουμε εφαρμόσει τα πιο κάτω:

```
var stopWatch = System.Diagnostics.Stopwatch.StartNew();
```

```
stopWatch.Start();
```

```
...*Εκτέλεση αλγορίθμου*...
```

```
stopWatch.Stop();
```

```
execution_time = stopWatch.ElapsedMilliseconds;
```

Τέλος με την ανάλογη επιλογή του αλγορίθμου αλλάζουμε το κείμενο που φαίνεται στο κεντρικό κουτί. Για την μετατροπή των συναρτήσεων σε string για την εμφάνισή μας έχουμε χρησιμοποιήσει το εργαλείο “String Builder for C#” [14].

6.4 Δυσκολίες, Τρόπος Επίλυσής τους και Διευκρινήσεις

Όσο μεγαλύτερο N έχουμε τόσο πιο πολύ αργοπορούνε γενικά όλοι οι αλγόριθμοι να τρέξουνε. Για να λειτουργούν σωστά οι κινήσεις, τα animations και άλλες λειτουργίες του Unity καθώς τρέχουνε οι αλγόριθμοι αυτοί και για να φαίνεται ταυτόχρονα η καταμέτρηση χρόνου, θα πρέπει με κάποιο τρόπο να τρέχουνε στο παρασκήνιο. Για να λύσουμε αυτό το πρόβλημα, μετά από έρευνά μας, έχουμε καταλήξει στο Job και Thread System του Unity [11].

Τελικά όμως παρατηρήσαμε ότι το πρόβλημά μας επιλύεται μονάχα με την δημιουργία ενός απλού νέου νήματος (Thread) χωρίς να κάνουμε πιο περίπλοκα τα πράγματα με το Job System. Αναλόγως με ποιο αλγόριθμο θα τρέξει ο χρήστης, η αντίστοιχη συνάρτηση θα καθιερωθεί ως νέο νήμα και θα εκτελείται στο παρασκήνιο (Thread run_thread = new Thread(*function name*); run_thread.Start();). Επιπλέον επειδή στα νήματα δεν επιτρέπεται έλεγχος και αλλαγή εξωτερικών αντικειμένων όπως GameObjects, οι αλλαγές στη διεπαφή του χρήστη θα πρέπει να εκτελούνται εξωτερικά του εκτελούμενού μας νήματος δια μέσου του Main Thread (δηλαδή στον κώδικα που βρίσκεται εκτός του νήματος που μόλις δημιουργήθηκε.)

Η καταγραφή του χρόνου εκτέλεσης μέσα σε ένα νήμα (thread) το οποίο μόλις δημιουργήσαμε δεν επιτρέπεται. Έτσι θα πρέπει ο χρόνος αυτός να μετρηθεί σαν εκτελείται το νήμα εξωτερικά, από το κεντρικό μας (main thread).

Δεν μπορούμε να εμφανίσουμε άμεσα το δυαδικό δένδρο Fibonacci. Χωρίς τη χρήση Coroutine και χρησιμοποιώντας κανονική συνάρτηση, το δένδρο δεν μπορεί να εμφανίσει τα βέλη του και αν χρησιμοποιούμε ένα μεγάλο αριθμό N τότε χρειάζεται ακόμη περισσότερο χρόνο για να εμφανιστεί το τελικό δένδρο. Το πρόβλημα αυτό παραμένει ακόμη και αν μετατρέψουμε την συνάρτησή μας σε νέο νήμα. Έτσι δεν μπορεί να επιλυθεί το πρόβλημα αυτό.

Το μέγεθος του ακέραιου τύπου που δέχεται το Fibonacci (Integer) δεν είναι επαρκές για να παρουσιαστούν αποτελέσματα που έχουν μεγάλο μέγεθος. Μετά από πολλές έρευνες και δοκιμασίες διαφόρων άλλων τύπων όπως long int , long long int, ulong (Επειδή έχουμε πάντα μόνο θετικούς αριθμούς), έχουμε καταλήξει στο ότι ο μεγαλύτερος αριθμός που μας επιτρέπει η C# να έχουμε στον κώδικά μας είναι τύπου **Double (64-bit floating point)**. Συνεπώς μετατρέψαμε όλες τις συναρτήσεις μας ούτως ώστε να δέχονται αριθμούς double και όλες οι πράξεις να γίνονται μόνο με double.

Η καταμέτρηση των χρόνων μας βασίζεται σε μεγάλο βαθμό στις εναλλαγές του frame rate (1ms είναι 1 frame rate). Επειδή αυτό επηρεάζεται από την συσκευή που τρέχει το

πρόγραμμά μας, έχουμε ορίσει ότι αν έχουμε 0-900ms στην καταμέτρησή μας, να θεωρούμε ότι είναι 1 ms. Δυστυχώς όμως δεν καταφέραμε να επιλύσουμε το πρόβλημα ασάφειας των χρόνων εκτέλεσης. Αυτό οφείλεται στο ότι ο αλγόριθμος της απομνημόνευσης εκτελείται σε χρόνο $O(n^2)$ και ο δυναμικός σε χρόνο $O(n)$. Επειδή το οι υπολογισμοί προγραμμάτων με τέτοιες πολυπλοκότητες στη C# και στο Unity γίνονται πολύ γρήγορα που η διαφορά μεταξύ πολυωνιμικού χρόνου με γραμμικού είναι σχεδόν μηδαμινή. Θα έπρεπε να είχαμε τεράστια νούμερα και μεγαλύτερη πολυωνιμική πολυπλοκότητα για να εμφανιστεί αργοπορία στον αλγόριθμο απομνημόνευσης.

Κεφάλαιο 7.

Τυχαιοποιημένος Αλγόριθμος Συρρίκνωσης Karger

7.1 Εισαγωγή	80
7.2 Περιγραφή Αλγορίθμου	81
7.3 Λεπτομέρειες Υλοποίησης	83
7.3.1 Σχεδιασμός Πρωτοτύπου	83
7.3.1.1 Δημιουργία 3D Συνεκτικού Πολυγράφου	83
7.3.1.2 Αποθήκευση και Ανάκτηση Γράφων	88
7.3.2 Υλοποίηση Διεπαφής	89
7.3.3 Αναπαράσταση και Εκτέλεση Αλγορίθμου	91
7.4 Δυσκολίες/Προβλήματα και Τρόπος Αντιμετώπισής τους	96

7.1 Εισαγωγή

Ο τελευταίος τύπος αλγορίθμων με τον οποίο θα αφοσιωθούμε στην υφιστάμενη διπλωματική εργασία είναι οι «Τυχαιοποιημένοι» (Randomized). [4,5]

Συγκεκριμένα θα παρουσιάσουμε το πρόβλημα της εύρεσης της καθολικής ελάχιστης αποκοπής ενός μη κατευθυνόμενου γραφήματος. Θα περιγράψουμε ορισμένους όρους που θα πρέπει να ξέρουμε πριν την υλοποίησή μας. Έστω το γράφημά μας $G=(V,E)$. Αποκοπή του G είναι ο διαχωρισμός του V σε δύο μη-κενά σύνολα A και B . Το **σύνολο των πιθανών μας αποκοπών** είναι 2^n-2 αν έχουμε ότι ο **αριθμός των κόμβων μας V είναι n** . Το **μέγεθος αποκοπής** (A,B) είναι ο αριθμός των ακμών που ενώνουν το A με το B σύνολο. **Ελάχιστη αποκοπή** είναι η αποκοπή η οποία έχει ελάχιστο μέγεθος. **Καθολική ελάχιστη**

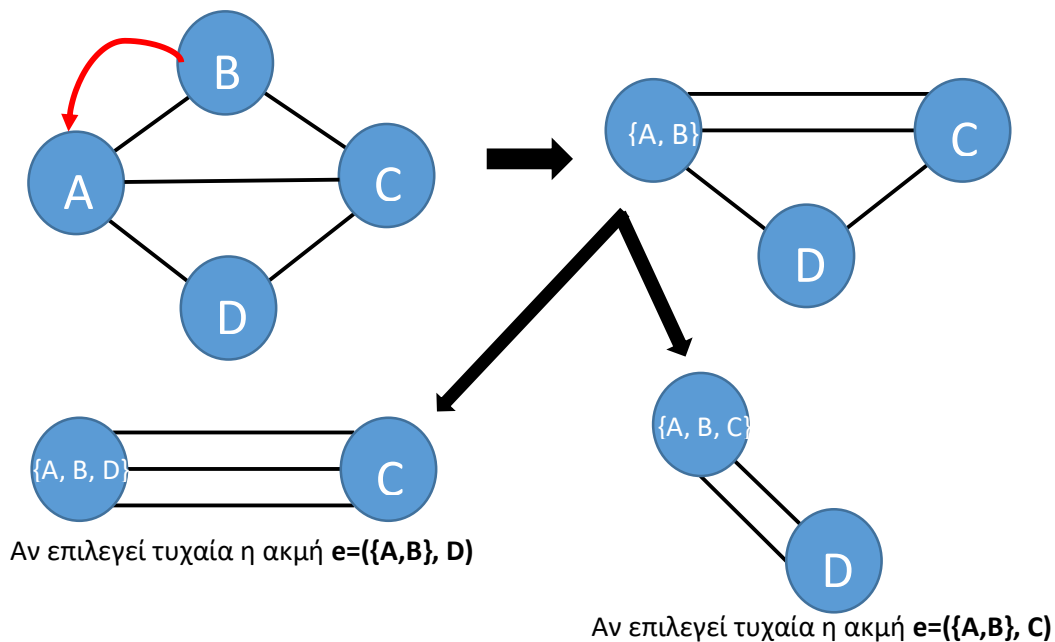
αποκοπή είναι η ελάχιστη αποκοπή όπου επιτρέπεται οποιαδήποτε αποκοπή του γραφήματος δεν υπάρχει προέλευση ή απόληξη.

Με την ανακάλυψη του **τυχαιοποιημένου Αλγορίθμου Συρρίκνωσης** από τον **David Karger** [17] το 1992 φτάσαμε σε μια ποιοτικά απλούστερη μέθοδο σε σχέση με προηγούμενους αλγόριθμους εύρεσης καθολικής ελάχιστης αποκοπής. Ο αλγόριθμος αυτός είναι τόσο απλός που, με πρώτη ματιά, είναι δύσκολο να πιστέψουμε ότι πραγματικά λειτουργεί. Αποτελεί ένα από τους πιο αποδοτικούς αλγόριθμους για καθολικές ελάχιστες αποκοπές αν και εκτελείται σε πολυωνυμικό χρόνο.

7.2 Περιγραφή Αλγορίθμου

Πριν ξεκινήσουμε με την περιγραφή του αλγορίθμου θα πρέπει να περιγράψουμε την έννοια του **Συνεκτικού Πολυγραφήματος (Multigraph)**. Έστω $G=(V,E)$ ένας γράφος τέτοιου τύπου. Ο γράφος είναι μη-κατευθυνόμενος στον οποίο επιτρέπεται να υπάρχουν «παράλληλες» πολλαπλές ακμές του ίδιου ζεύγους κόμβων.

Ο Αλγόριθμος Συρρίκνωσης (Karger) ξεκινά επιλέγοντας με ομοιόμορφα τυχαίο τρόπο μια ακμή $e=(u,v)$ του G και εφαρμόζει «συρρίκνωση» (πχ: Σχήμα 7.1).



Συρρίκνωση για γράφημα 4 κόμβων - Σχήμα 7.1

Στη συρρίκνωση, παράγουμε ένα νέο γράφημα G' στο οποίο οι κόμβοι u και v της ακμής που έχει επιλεχθεί, έχουν αποτελέσει ένα νέο μοναδικό κόμβο w και οι υπόλοιποι κόμβοι διατηρούν τη ταυτότητά τους. Οι ακμές που είχαν τα δύο άκρα τους στο u και v , διαγράφονται στο G' . Οι υπόλοιπες ακμές διατηρούνται αλλά αν ένα από τα άκρα τους ισούται είτε με u είτε με v , τότε το άκρο αυτό ενημερώνεται ώστε να είναι ίσο με το νέο κόμβο w . Βλέπουμε ότι στο G' μπορούν να εμφανιστούν πολλαπλές παράλληλες γραμμές.

Ακολούθως ο αλγόριθμος αυτός συνεχίζει αναδρομικά για το G' επιλέγοντας μια ακμή με ομοιόμορφα τυχαίο τρόπο και συρρικνώνοντάς την. Καθώς προχωράμε, οι συστατικές κορυφές του G' τις θεωρούμε **υπερκόμβους (supernodes)**. Κάθε υπέρκομβος w αντιστοιχεί στο υποσύνολο $S(w)$ του συνόλου V που έχουν συγχωνευθεί από προηγούμενες συρρικνώσεις. Ο αλγόριθμός μας τερματίζει όταν φθάσουμε σε ένα γράφημα G' που περιέχει μόνο δύο υπέρκομβους v_1 και v_2 . Οι δύο αυτοί κόμβοι έχουν τα αντίστοιχα υποσύνολα $S(v_1)$, $S(v_2)$ του συνόλου V που αποτελούνται από τους κόμβους που έχουν συρρικνωθεί. Αυτά τα δύο σύνολα σχηματίζουν τη διαμέριση του V .

Πιο κάτω έχουμε τον ψευδοκώδικα του αλγορίθμου μας (Σχήμα 7.2) με είσοδο ένα πολυγράφημα $G=(V,E)$ (Χρόνος εκτέλεσης: $\Theta(n^2)$):

```

1. Συρρίκνωση( $G$ ){
2.     Για κάθε κόμβο  $v$  {
3.         Καταγραφή συνόλου  $S(v)$  κόμβων που έχουν συρρικνωθεί στον  $v$ 
4.         Αρχικά  $S(v) = \{v\}$  για κάθε  $v$ 
5.     }
6.     Αν το  $G$  έχει δύο κόμβους  $v_1, v_2 \rightarrow$  επέστρεψε αποκοπή ( $S(v_1), S(v_2)$ )
7.     Αλλιώς{
8.         Διάλεξε μια ακμή  $e=(u, v)$  του  $G$  με ομοιόμορφα τυχαίο τρόπο
9.         Έστω  $G'$  το γράφημα που προκύπτει μετά τη συρρίκνωση της  $e$ , με ένα νέο κόμβο  $z$ 
10.        που αντικαθιστά τους  $u$  και  $v$ .
11.        Όρισε  $S(zuv) = S(u) \cup S(v)$ 
12.        Εφάρμοσε τον αλγόριθμο συρρίκνωσης αναδρομικά στο  $G'$  (Συρρίκνωση( $G'$ ))
13.    }
14. }
```

Ψευδοκώδικας Αλγόριθμου Συρρίκνωσης - Σχήμα 7.2

7.3 Λεπτομέρειες Υλοποίησης

7.3.1 Σχεδιασμός Προτύπου

Για την υλοποίηση του αλγόριθμου Karger στο Unity θα πρέπει να σκεφτούμε αρχικά ένα τρόπο με τον οποίο θα εισάγουμε ένα επιθυμητό πολυγράφο. Ο τρόπος που έχουμε καταλήξει είναι διαδραστικός. Ο χρήστης θα έχει την δυνατότητα να δημιουργήσει ο ίδιος τον γράφο χρησιμοποιώντας το έτοιμο τελικό μας εργαλείο με τη χρήση ποντικιού. Εκτός από την δημιουργία ενός γράφου, ο χρήστης θα έχει την δυνατότητα να τον αποθηκεύσει με ένα επιθυμητό όνομα. Αργότερα θα μπορεί να τον φορτώσει από μια λίστα με όλους τους αποθηκευμένους γράφους. Ακολούθως φορτώνοντας τον ανάλογο γράφο ή μετά τη δημιουργία ενός καινούριου, θα μπορεί να τρέξει τον αλγόριθμο Συρρίκνωσης.

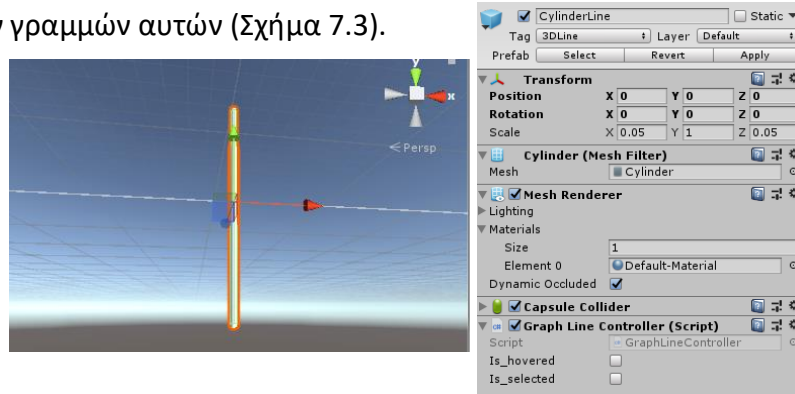
7.3.1.1 Δημιουργία 3D Συνεκτικού Πολυγράφου

Συνεπώς για την οπτική παρουσίαση του γράφου έχουν ακολουθήσει διάφορες σκέψεις. Μετά από διάφορα προσχέδια, έχουμε καταλήξει στην εμφάνιση των κόμβων του γράφου με χρήση των τρισδιάστατων μας σφαιρών που έχουμε ήδη υλοποιημένες και περιέχουν το ενσωματωμένο script που επιτρέπει την κινούμενη οπτικά μετακίνηση τους (**BallTextMesh** Σχήμα 2.6 από Κεφάλαιο 2). Στην σφαίρα αυτή προσθέσαμε ακόμη ένα script, το **“GraphNodeController.cs”** (Παράρτημα Ζ). Το script αυτό επιτρέπει διάφορες ιδιότητες με χρήση του ποντικιού πάνω στη σφαίρα. Έχουν υλοποιηθεί τα πιο κάτω με την χρήση των έτοιμων συναρτήσεων που προσφέρουν οι **“MonoBehaviour”** τύποι κλάσεων:

- **onMouseEnter** (όταν εισέλθει ο δείκτης του ποντικιού στη σφαίρα) ➔ Αλλαγή χρώματος σφαίρας σε κόκκινο χρώμα και ενεργοποίηση μεταβλητής **“isHovered”**
- **onMouseDown** (όταν πατήσουμε το ποντίκι πάνω στη σφαίρα) ➔ Αν είναι η πρώτη φορά θα θεωρηθεί μόνο πάτημα του ποντικιού αλλιώς μετά από ένα πολύ μικρό χρονικό διάστημα, αν ξαναπατηθεί το ποντίκι πάνω από τη σφαίρα, θα θεωρηθεί ότι έχει επιλεγθεί. Αλλαγή χρώματος σε πράσινο και ενεργοποίηση του **“isSelected”**
- **onMouseExit** (όταν εξέλθει ο δείκτης του ποντικιού στη σφαίρα) ➔ Αλλαγή χρώματος στο προεπιλεγμένο χρώμα (μπλε) και **“isHovered” = false**.

Με αυτό τον τρόπο υλοποιήσαμε την επιλογή μιας σφαίρας και ανίχνευση ποντικού πάνω στη σφαίρα. Επίσης το script αυτό μας επιτρέπει να αλλάξουμε/αντλήσουμε τις τιμές του κόμβου τόσο προγραμματιστικά όσο και με το κείμενο που θα φαίνεται στην 3D παρουσίασή της.

Θα πρέπει επιπλέον να προστεθεί η δυνατότητα εμφάνισης γραμμών που θα ενώνουν κάθε κόμβο. Καταλήξαμε στη χρήση τρισδιάστατου κυλίνδρου (3D Cylinder) για την εμφάνιση των γραμμών αυτών (Σχήμα 7.3).



CylinderLine prefab GameObject - Σχήμα 7.3

Για το αντικείμενο αυτό έχουμε καθορίσει ένα μέγεθος συγκεκριμένο (Scale = (0.05, 1, 0.05)) ούτως ώστε να φαίνεται ως μια γραμμή που θα ενώνει κόμβους σε ένα γράφο. Όπως και με τους κόμβους, έτσι και εδώ έχουμε δημιουργήσει και ορίσει ως component ένα script, “**GraphLineController.cs**” (Παράρτημα Ζ) το οποίο παρέχει τις ίδιες λειτουργίες όπως με αυτό των σφαιρών εκτός από τον ορισμό τιμών και της επιλογής [15].

Για την σύνδεση δύο κόμβων (δύο σημείων δηλαδή) με χρήση της πιο πάνω γραμμής εκτελούμε τα ακόλουθα:

- Βρίσκουμε το διάνυσμα $\bar{V} = \overline{P2} - \overline{P1} = (X, Y)$ που δείχνει από το P1 στο P2 σημείο του χώρου (κέντρα δύο κόμβων).
- Ορίζουμε τη θέση της ακμής μας να βρίσκεται στο κέντρο των δύο σημείων:

$$\bar{C} = \overline{P1} - \frac{\bar{V}}{2}$$

- Ορίζουμε το μέγεθος της ακμής μας στον άξονα Y να είναι **το μήκος του $\frac{\bar{V}}{2}$** . Ο τρόπος που έχουμε καθορίσει το αντικείμενό μας αρχικά, με την αλλαγή του μεγέθους του σε αυτό, έχουμε το ακριβές μέγεθος ακμής που ενώνει τα δύο σημεία.
- Τέλος ορίζουμε την περιστροφή της ακμής να είναι η ίδια με το $\bar{V}$.

Με τον ίδιο τρόπο εμφανίζουμε ακμές αργότερα στην υλοποίησή μας από το σημείο που είναι το ποντίκι στην οθόνη.

Μετά τον σχεδιασμό των πιο πάνω αντικειμένων έχουμε φθάσει σε σημείο όπου μπορούμε να υλοποιήσουμε τον γράφο μας προγραμματιστικά με ταυτόχρονη χρήση των αντικειμένων αυτών. Το κύριο script που περιγράφει τον γράφο είναι το **“Undirected3DGraph.cs”** το οποίο μπορεί να χρησιμοποιηθεί ως αντικείμενο. Η υλοποίηση του γράφου περιλαμβάνει δύο λίστες. Η μία περιέχει τους κόμβους του γράφου (τύπου **“GraphNode.cs”**) και οι άλλη τις ακμές μας (τύπου **“Edge.cs”**).

Επιγραμματικά και με συντομία το script του γράφου περιλαμβάνει τις ακόλουθες λειτουργίες με τις συναρτήσεις που προσφέρει:

- void Destroy() → Διαγράφει όλο τον γράφο
- void **RebuildFromDataFile**(Und3DGraphData graphData) → Ξανακτίζει όλο το γράφο βασισμένο από ένα συγκεκριμένο αρχείο (θα περιγράψουμε αργότερα πώς γίνεται η αποθήκευση/ανάκτηση)
- List<GraphNode> getNodeList(), void SetNodeList(List<GraphNode> nl), List<Edge> getEdgeList(), void SetEdgeList(List<Edge> el) → Ορισμός και ανάκτηση λίστας Ακμών και Κόμβων
- Void addNode(GraphNode n), void removeNode(GraphNode node) → Πρόσθεση και αφαίρεση κόμβου από γράφο. Στην αφαίρεση, διαγράφονται και όλες οι ακμές που ενώνουν τον κόμβο αυτό.
- Void addEdge(GraphNode src, GraphNode dst)/(GraphNode src, GraphNode dst,int noOfEdges) → Προσθήκη ακμής στον γράφο από κόμβο src προς dst. Επιπλέον επιλογή για ορισμό noOfEdges αριθμών ακμών που θα φαίνονται.

- `Bool deleteEdge(GraphNode src, GraphNode dst)` → Διαγραφή ακμής από `src` προς `dst` και επιστροφή `true/false` αν ήταν επιτυχής
- `Int getNodeIndex(List<int> vals)` → Βρίσκει τον αριθμό σειράς του κόμβου μέσα στη λίστα με τους άλλους κόμβους βάση τις `vals` τιμές
- `Void ResetAllEdgePositions()` → Επανατοποθετεί όλες τις θέσεις των ακμών στις σωστές θέσεις μετά από πιθανές μετακινήσεις κόμβων.
- `Edge edgeExists(GraphNode src, GraphNode dst)` → Ελέγχει αν υπάρχει ακμή που συνδέει τον κόμβο `src` με `dst` και την επιστρέφει
- `Void updateNode(GraphNode node, List<int> new_vals)` → Ενημερώνει ένα κόμβο με νέες επιθυμητές τιμές.
- `Vector3 FindCentrePointFromEdge(int edge_no)` → Βρίσκει το κεντρικό σημείο μιας ακμής (μεταξύ κόμβου `src` και `dst`)
- `Void moveEdges(GraphNode from, GraphNode to)` → Μετακινεί όλες τις ακμές που βρίσκονται στον `from` κόμβος προς τον `to`.
- `Void removeAllEdges(GraphNode node)` → αφαιρεί όλες τις ακμές που βρίσκονται στον επιθυμητό κόμβο.
- `Bool mergeEdge(int edge_no)` → Η κύρια συνάρτηση που χρησιμοποιεί ο αλγόριθμός μας. Χρησιμοποιείται για την συγχώνευση μιας ακμής δημιουργώντας ένα νέο κόμβο με τις τιμές του `src` και `dst` κόμβου, αφαιρώντας αυτούς τους κόμβους και ακμή και στη συνέχεια παρουσιάζοντας τις νέες ακμές που θα εμφανιστούν στο πολυγράφο μας.

Για σύγκριση των στοιχείων που περιέχουν 2 λίστες εφαρμόσαμε την μέθοδο [16].

Θα πρέπει ο γράφος μας να αλλάζει δυναμικά ανάλογα με την δομή του προγραμματιστικά καθώς εφαρμόζεται ο αλγόριθμος συρρίκνωσης όπου συγχωνεύονται κόμβοι και εμφανίζονται πολλαπλές γραμμές. Αυτή την αναγκαία λειτουργία που παρουσιάζει ένας πολυγράφος, την υλοποιήσαμε με χρήση ορισμένων βασικών μαθηματικών εννοιών που χρησιμοποιούνται στα μαθήματα γραφικών.

Συγκεκριμένα έχουμε την πιο κάτω πιθανή περίπτωση (Σχήμα 7.4):



Εμφάνιση πολλαπλών ακμών μεταξύ κόμβων - Σχήμα 7.4

Έστω ότι έχουμε δύο κόμβους (**Source Node** και **Destination Node**) που συνδέονται με μια ακμή. (Δεν μας νοιάζει ο Z άξονας καθώς οι κόμβοι μας θα βρίσκονται στο ίδιο σημείο Z. Έτσι τα σημεία μας υποθέτουμε πως τα λαμβάνουμε ως 2D σε δύο διαστάσεις (X,Y)) Οι κόμβοι αυτοί μπορούν να βρίσκονται σε οποιαδήποτε θέση στο χώρο. Έχουμε το κεντρικό σημείο κάθε κόμβου να είναι το **C1** και **C2** αντίστοιχα. Αν έχουμε μονάχα μία ακμή είναι εύκολο να υπολογιστεί ενώνοντας απλώς τα δύο αυτά κέντρα. Τώρα για πολλαπλές γραμμές (>2) κάνουμε το εξής:

- Έστω **Radius** η ακτίνα μας (όλοι οι κόμβοι έχουν την ίδια ακτίνα)
- Θα πρέπει να ορίσουμε δύο νέα σημεία στις άκρες του κάθε κόμβου που θα είναι παράλληλα και απέναντι μεταξύ τους. Έστω **A** και **B** τα σημεία του Source και **C**, **D** τα σημεία του Destination. Το A είναι απέναντι από το C και το B έναντι του D.
- Για να πετύχουμε την εύρεση αυτών των σημείων εκτελούμε τα ακόλουθα:
 - Βρίσκουμε το διάνυσμα $\bar{V} = \bar{C2} - \bar{C1} = (X, Y)$ που δείχνει από το C1 στο C2 σημείο.
 - Υπολογίζουμε το κάθετο διάνυσμα πάνω σε αυτό $\bar{T} = \bar{V}_T = (-Y, X)$
 - Κανονικοποιούμε το αποτέλεσμα μας (σε αριθμούς από 0 ως 1) $\hat{T} = \frac{\bar{T}}{|\bar{T}|}$ για να μπορούμε να εφαρμόσουμε απλούς υπολογισμούς στην συνέχεια.
 - Υπολογίζουμε τα σημεία μας:
 - $A = \bar{C1} + \hat{T} * Radius$
 - $B = \bar{C1} - \hat{T} * Radius$

- $C = \overline{C2} + \hat{T} * Radius$
- $D = \overline{C2} - \hat{T} * Radius$

Τώρα, χρησιμοποιώντας αυτά τα σημεία μπορούμε πολύ εύκολα χρησιμοποιώντας γραμμική παρεμβολή (Linear Interpolation) να εμφανίσουμε ένα επιθυμητό αριθμό ακμών μεταξύ των δύο κόμβων. Δηλαδή από εφαρμόζουμε γραμμική παρεμβολή για κάθε ακμή που δημιουργούμε (P1,P2) ούτως ώστε: P1 → lerp μεταξύ A και B και για το P2 → lerp μεταξύ C και D.

7.3.1.2 Αποθήκευση και Ανάκτηση Γράφων

Για την αποθήκευση και ανάκτηση γράφων έχουμε δημιουργήσει ένα σύστημα αποθήκευσης και ανάκτησης δεδομένων από αρχεία, το “**SaveSystem.cs**” (Παράρτημα Z). Για να πετύχουμε αυτή την αποθήκευση και ανάκτηση δεδομένων χρησιμοποιήσαμε την γνωστή σύμβαση “**Serializable**” που χρησιμοποιείται ευρέως για την αποθήκευση αντικειμένων και δομών δεδομένων [25,26,27,29].

Χρησιμοποιώντας το πρότυπο “**Singleton**” έχουμε πετύχει χρήση ενός script το οποίο διατηρεί τις ιδιότητές του εφαρμόζοντάς το πάνω στο “**SaveSystem.cs**”. Ένα Singleton είναι ένα μοτίβο σχεδιασμού λογισμικού που περιορίζει την παράσταση μιας κλάσης σε μία “μοναδική” παρουσία. Αυτό είναι χρήσιμο όταν απαιτείται ένα ακριβώς αντικείμενο για τον συντονισμό των ενεργειών στο σύνολο ενός συστήματος. Ο όρος προέρχεται από τη μαθηματική έννοια ενός singleton. [30] Έτσι απλώς καλώντας το script αυτό από όπου και να βρισκόμαστε, μπορούμε να αποθηκεύσουμε και να ανακτήσουμε ένα γράφο από ένα επιθυμητό αρχείο.

Αρχικά για να αποθηκεύσουμε ένα γράφο πρέπει να ορίσουμε τι δεδομένα θα χρειαστούν να αποθηκευθούν. Μετά από σκέψη καταλήξαμε στα ακόλουθα (τα οποία είναι μαζεμένα ως “**SerializeFields**” στο “**Serializable**” script “**Und3DGraphData.cs**”):

- **Nodes**: Λίστα με του κόμβους και τις τιμές τους (φωλιασμένη λίστα int: **List<List<int>>** → Επειδή όμως οι συμβάσεις του Unity δεν επιτρέπουν αποθήκευση φωλιασμένων λιστών, έχουμε ορίσει ένα νέο αντικείμενο

“**ListWrapper.cs**” που περιέχει απλώς μια λίστα `List<int>` και είναι “`Serializable`”.
Συνεπώς έχουμε `List<ListWrapper>`) [28]

- **Node Positions:** Λίστα με τις θέσεις του κάθε κόμβου (`List<Vector3>` → Οι συμβάσεις του Unity δεν μας επιτρέπουν την αποθήκευση `Vector3` και `Quaternions`. Συνεπώς χρησιμοποιήσαμε ένα νέο αντικείμενο “**SerializableVector3.cs**” το οποίο είναι “`Serializable`” και περιέχει τις τιμές `x,y,z` σε μεταβλητές τύπου `float`)
- **srcNodes, dstNodes:** Λίστες με τους δείκτες των `Source` και `Destination Nodes` αντίστοιχα για κάθε ακμή. Πχ. (`srcNodes(0),dstNodes(0)`) → η ακμή 0.
- **noOfEdges:** Λίστα με τον αριθμό γραμμών που έχει η κάθε ακμή.

Με αυτά τα δεδομένα μπορούμε εύκολα να ξανακτίσουμε ένα γράφο ανακτώντας τα από ένα αρχείο με τη χρήση της συνάρτησης “`RebuildFromDataFile()`” που βρίσκεται στο script του γράφου μας όπως έχουμε προαναφέρει.

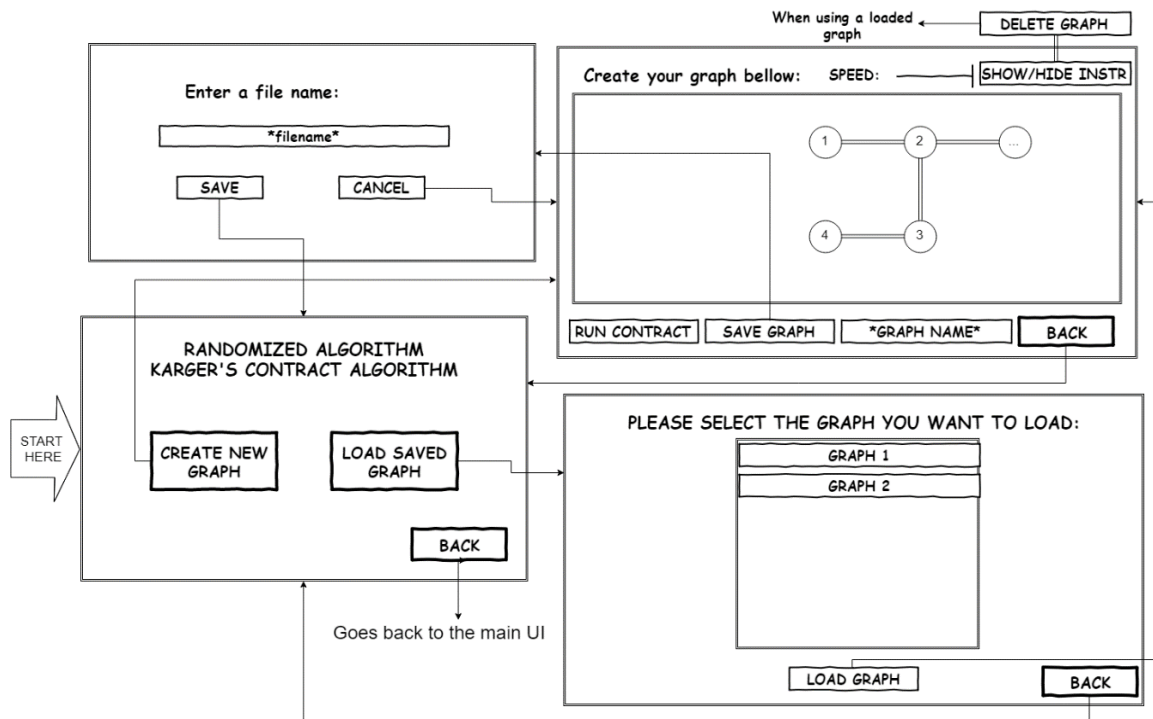
Τέλος να αναφέρουμε ότι η αποθήκευση και ανάκτηση αρχείων γίνεται με χρήση “**FileStream**” σε ένα όνομα αρχείου που επιθυμεί ο χρήστης τύπου “.dat” και “**BinaryFormatter**” για το `Serialization` και `Deserialization` των δεδομένων μας. Περισσότερα μπορείτε να δείτε στα scripts του παραρτήματος [31].

7.3.2 Υλοποίηση Διεπαφής

Αρχικά σχεδιάστηκε ένα πρωτότυπο στο χαρτί όπως κάνουμε πάντα για να διευκρινιστεί ο τρόπος με τον οποίο θα χειρίζεται ο αλγόριθμος από το χρήστη. Μετά από πολλές απορρίψεις σχεδίων έχουμε καταλήξει στα πιο κάτω αρχικά προσχέδια μιας απλής διεπαφής (Σχήμα 7.5). Ο χρήστης έχει τις ακόλουθες δυνατότητες:

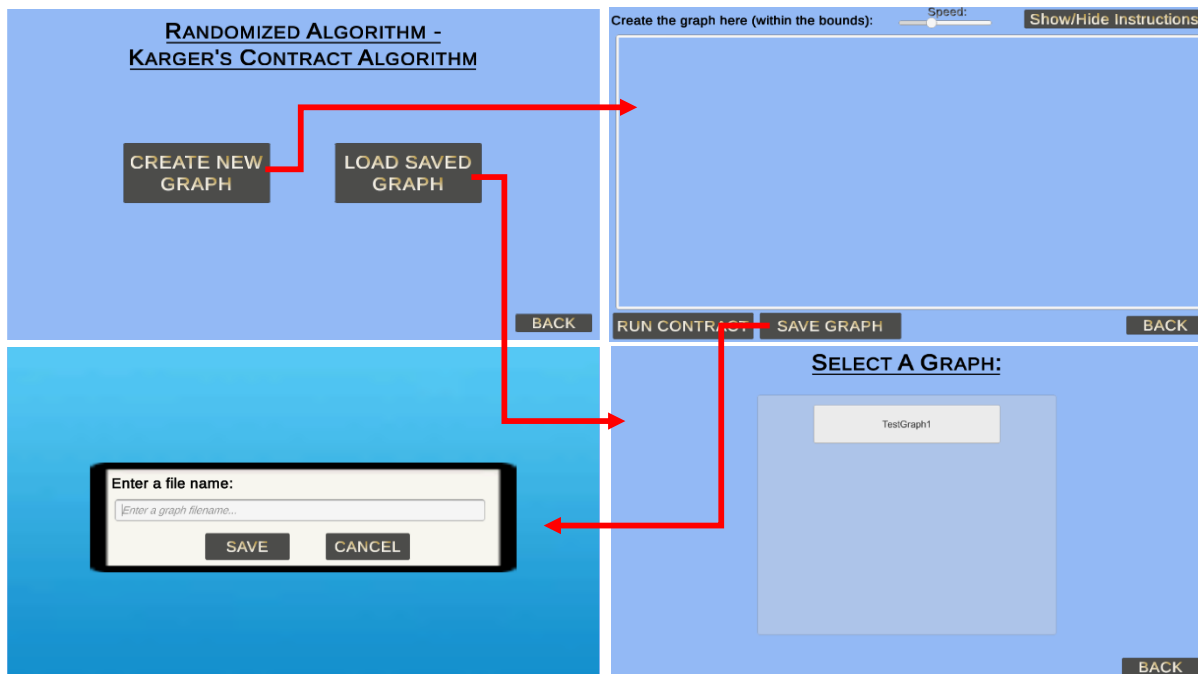
- Από την αρχική διεπαφή με το κουμπί “`CREATE NEW GRAPH`” θα οδηγείτε ευθέως στον καμβά από όπου θα μπορεί να δημιουργήσει τον γράφο (Θα επεξηγηθεί αργότερα ο τρόπος δημιουργίας).
- Μόλις ολοκληρώσει το σχεδιασμό του επιθυμητού γράφου ο χρήστης θα μπορεί αμέσως να τρέξει τον αλγόριθμο συρρίκνωσης επιλέγοντας το “`RUN CONTRACT`” όπου ταυτόχρονα μπορεί να ορίσει την ταχύτητά του με το “`Speed`” slider.

- Με το “SHOW/HIDE INSTRUCTIONS” μπορεί να δει οδηγίες για δημιουργία ενός γράφου.
- Επιλέγοντας το “SAVE GRAPH” παραπεμπόμαστε σε άλλο καμβά όπου μπορούμε να επιλέξουμε ένα επιθυμητό όνομα για το γράφο μας και να τον αποθηκεύσουμε.
- Από τον αρχικό μας καμβά μπορούμε να ανακτήσουμε ένα γράφο που αποθηκεύσαμε από μια λίστα με τους αποθηκευμένους μας γράφους επιλέγοντας το “LOAD SAVED GRAPH”.
- Στο νέο καμβά που εμφανίζεται ο χρήστης μπορεί πολύ απλά να επιλέξει ένα γράφο που επιθυμεί και να τον φορτώσει. Με την φόρτωση του γράφου, το κουμπί για τις οδηγίες αλλάζει σε “DELETE GRAPH”. Επιλέγοντάς το, διαγράφεται ο γράφος και επιστρέφουμε στην αρχική διεπαφή.
- Τα κουμπιά “CANCEL” και “BACK” εκτελούν την ευνόητη λειτουργία τους.



Αρχικό προσχέδιο διεπαφής αλγόριθμου Karger - Σχήμα 7.5

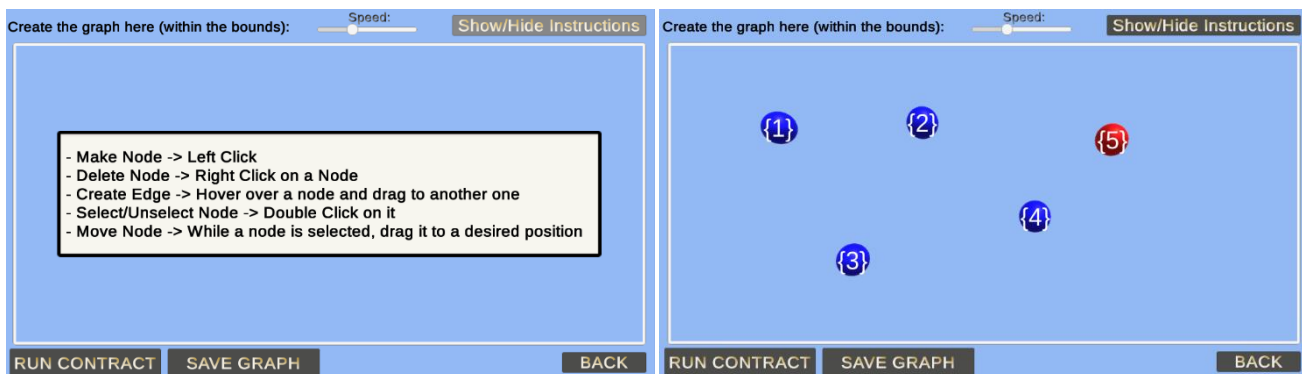
Με το προσχέδιο αυτό και με την υλοποίησή μας μέσα από το Unity καταλήξαμε στην ακόλουθη διεπαφή του αλγορίθμου μας με αυτές τις οθόνες (Σχήμα 7.6).



Τελική εμφάνιση διεπαφών αλγορίθμου Karger - Σχήμα 7.6

7.3.3 Αναπαράσταση και Εκτέλεση Αλγορίθμου

Προχωράμε τώρα σε ένα παράδειγμα εκτέλεσης της εφαρμογής μας με τον αλγόριθμο Karger. Ταυτόχρονα θα περιγραφούν και ορισμένες λειτουργίες και υλοποιήσεις που έχουν γίνει στη πορεία. Όλες οι πιο κάτω λειτουργίες είναι υλοποιημένες στο script "CreateGraphUIController.cs".



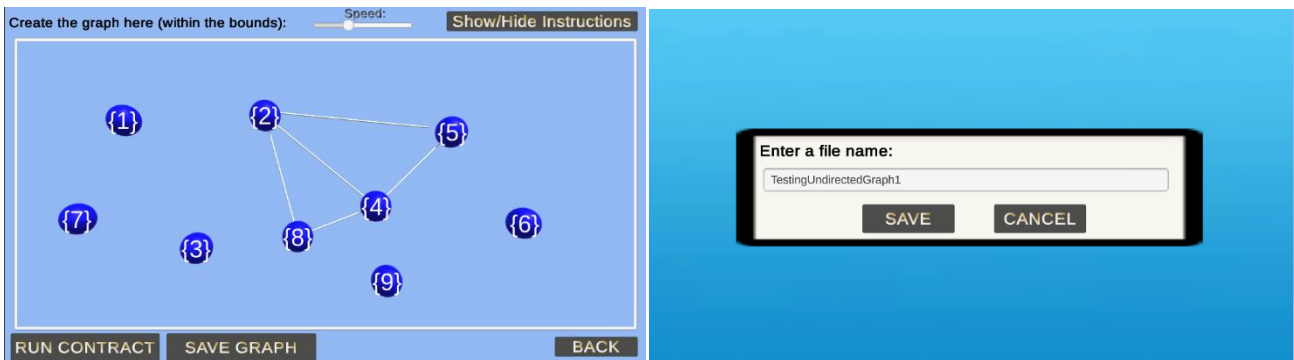
Δημιουργία γράφου 1 – Αλγόριθμος Karger - Σχήμα 7.7

Με την επιλογή του χρήστη στο κουμπί "CREATE NEW GRAPH" εμφανίζεται η πιο πάνω οθόνη (Σχήμα 7.7). Επιλέγοντας το "SHOW/HIDE INSTRUCTIONS" φαίνονται οι οδηγίες χρήσης για δημιουργία ενός γράφου. Κάνοντας αριστερό κλικ σε οποιοδήποτε σημείο

βρίσκεται μέσα στο πλαίσιο, εμφανίζεται νέος κόμβος. Την εμφάνιση των αντικειμένων μόνο μέσα στο πλαίσιο το πετύχαμε δημιουργώντας ένα πεδίο “bounds”, τύπου “Rectangle”, όπου μπορεί να λειτουργήσει το ποντίκι στη δημιουργία γράφου [18]. Την ανίχνευση του ποντικιού σε αυτό το πεδίο την πετύχαμε με χρήση της απλής εντολής “bounds.Contains(Input.mousePosition)”. Να σημειωθεί ότι ο χρήστης έχει όριο μέχρι και 30 κόμβους να δημιουργήσει. Κάνοντας “hover” πάνω από ένα κόμβο, ο κόμβος αυτός παίρνει κόκκινο χρώμα όπως φαίνεται στη δεξιά εικόνα.

Με ένα δεξί κλικ, διαγράφεται ο αντίστοιχος κόμβος και όλες οι ακμές που είναι ενωμένες με αυτόν. Εδώ αξίζει να σημειωθεί ότι υπάρχει απομνημόνευση των κόμβων που διαγράφονται και που φαίνονται με ένα πίνακα Boolean. Αν διαγραφεί ένας κόμβος, την επόμενη φορά που θα δημιουργήσει ένα κόμβο ο χρήστης, θα είναι με τη σωστή σειρά ή θα είναι ένας από τους προηγούμενα διαγραμμένους. Οι κόμβοι απαριθμούνται μόνο από το {1} μέχρι το {30}.

Με διπλό αριστερό κλικ πάνω σε ένα κόμβο, ο κόμβος αυτός πρασινίζει και θεωρείται επιλεγμένος. Αν τώρα έχει πατημένο το ποντίκι ο χρήστης πάνω από ένα επιλεγμένο κόμβο, τότε μπορεί να το μετακινήσει. Για την απενεργοποίηση της επιλογής, απλά πατάμε ξανά διπλό αριστερό κλικ στον επιλεγόμενο κόμβο. Εδώ επίσης αν ο χρήστης επιλέξει άλλο κόμβο ενώσω είναι επιλεγμένος ένας άλλος, τότε ο άλλος κόμβος σταματάει να είναι επιλεγμένος και επιλέγεται ο υφιστάμενος.

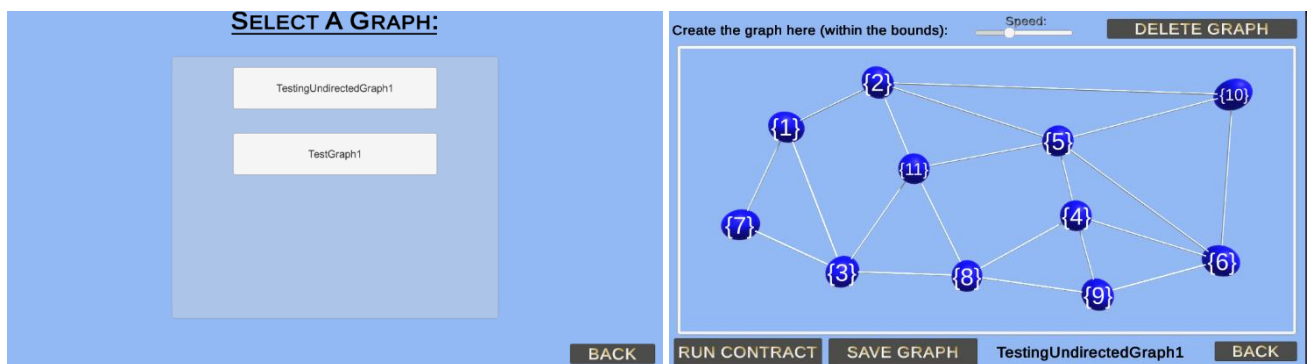


Δημιουργία γράφου 2 + Αποθήκευση – Αλγόριθμος Karger - Σχήμα 7.8

Έχοντας το ποντίκι πάνω από ένα κόμβο (hovered), ο χρήστης μπορεί να έχει πατημένο το ποντίκι του και να δημιουργήσει μια νέα ακμή (Σχήμα 7.8). Αν το αφήσει το ποντίκι σε κενό

χώρο ή πάνω από κόμβο που έχει ήδη σύνδεση με ακμή ή πάνω από τον ίδιο κόμβο, τότε η γραμμή αυτή θα διαγραφεί. Καινούρια ακμή δημιουργείται μόνο αν τραβήξει από ένα κόμβο σε ένα άλλο που δεν έχει ένωση. Ομοίως, όπως και στους κόμβους μας, ο χρήστης μπορεί να διαγράψει ακμές αλλά δεν μπορεί να επιλέξει/ξανα-μετακινήσει.

Επιπρόσθετα όπως βλέπουμε στην δεξιά εικόνα, ο χρήστης επιλέγοντας το κουμπί “SAVE GRAPH”, εμφανίζεται η οθόνη όπου μπορεί να δώσει ένα όνομα, στη περίπτωση μας “TestingUndirectedGraph1”, και επιλέγοντας το “SAVE” αποθηκεύεται ο γράφος μας και επιστρέφουμε ευθέως πίσω στην κεντρική διεπαφή του αλγορίθμου.



Ανάκτηση γράφου από λίστα – Αλγόριθμος Karger - Σχήμα 7.9

Σε αυτό το σημείο, έχουμε την δυνατότητα ανάκτησης ενός αποθηκευμένου γράφου. Επιλέγοντας το κουμπί “LOAD SAVED GRAPH”, εμφανίζεται ο καμβάς ο οποίος περιέχει μια λίστα τύπου “UI→Scroll View” με το script “**ScrollGraphList.cs**” το οποίο περιέχει κανονικά κουμπιά με script “**GraphItemButton.cs**” το οποίο χρησιμοποιείται για τη φόρτωση του αντίστοιχου γράφου που έχει στο όνομά τους. Τα κουμπιά φορτώνονται με χρήση pool [20]. Επιλέγοντας τον γράφο που έχουμε αποθηκεύσει προηγουμένως, παρατηρούμε ότι εμφανίζεται αμέσως. Στην οθόνη μας αυτή τη φορά όπως βλέπουμε στα δεξιά, εμφανίζεται το “DELETE GRAPH” το οποίο, επιλέγοντάς το, διαγράφεται ο υφιστάμενός μας γράφος.

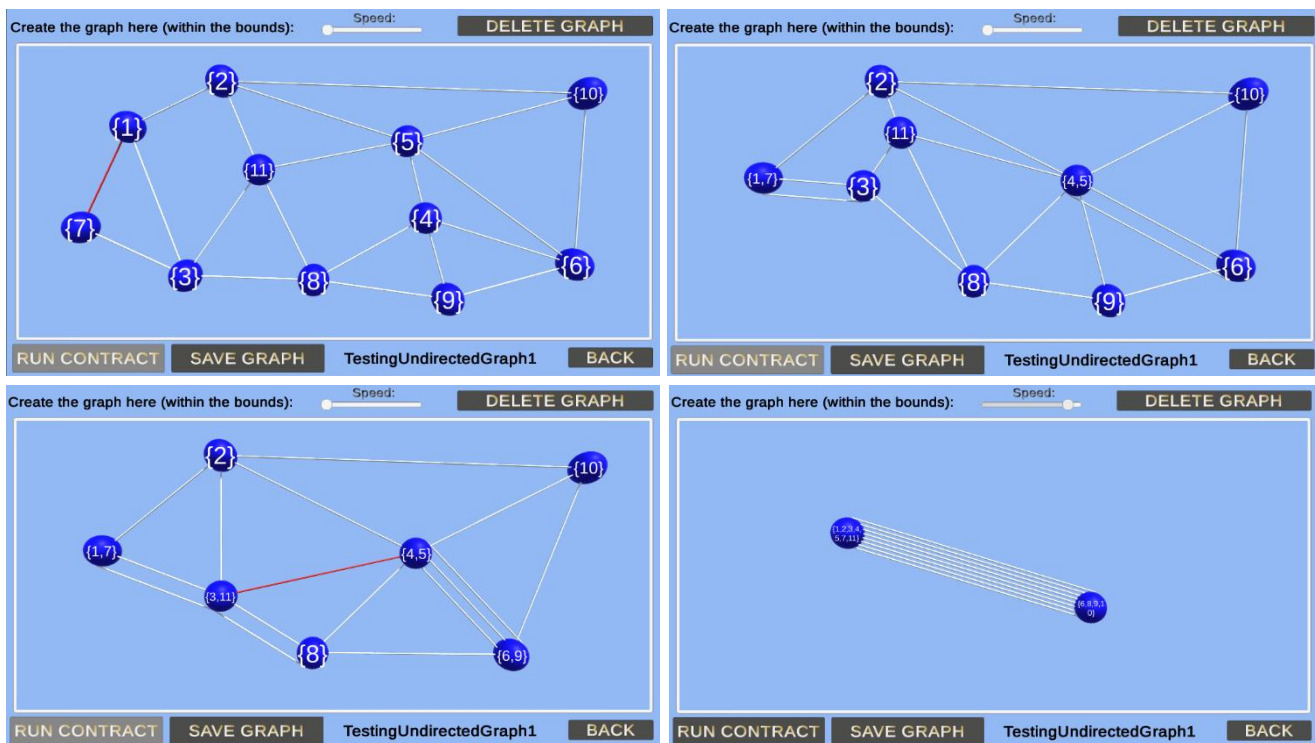
Έφτασε η στιγμή να τρέξουμε τον αλγόριθμό μας. Θα χρησιμοποιήσουμε το παράδειγμα γράφου που έχουμε φτιάξει πιο πριν. Ο αλγόριθμος ξεκινά να τρέχει επιλέγοντας το κουμπί “RUN CONTRACT”. Καθώς τρέχει μπορούμε με το “Speed” slider να αυξομειώσουμε την ταχύτητα εκτέλεσής του. (Σχήμα 7.10) Αρχικά ο αλγόριθμος επιλέγει ομοιόμορφα τυχαία

μια ακμή από το γράφο μας χρησιμοποιώντας τυχαίο Seed βασισμένο στο ρολόι του υπολογιστή χρησιμοποιώντας το Random της C# όπως έχουμε πιο κάτω:

```
Random.seed = (int)System.DateTime.Now.Ticks;
```

```
int random_edge_no = (int)(Random.value * und3dGraph.getEdgeList().Count - 1);
```

Στη περίπτωση μας όπως βλέπουμε στην 1^η οθόνη, έχει επιλεγεί η ακμή $(\{1\}, \{7\})$. Ακολουθώντας την συρρικνώνει όπως βλέπουμε στην πάνω δεξιά οθόνη μετακινώντας τους δύο κόμβους ("BallMover.cs") τον ένα πάνω στον άλλο στο κέντρο της ακμής και εμφάνιση καινούριου.



Εκτέλεση Αλγορίθμου Karger για ένα παράδειγμα - Σχήμα 7.10

Η συρρίκνωση της ακμής (συνάρτηση mergeEdge() – Σχήμα 7.11 του Undirected3DGraph.cs) επιτυγχάνεται με τον ακόλουθο τρόπο για μια ακμή $E(A,B)$:

- Δημιουργία νέου κόμβου **N** με όλες τις τιμές του A και του B μαζί, στο κέντρο της ακμής.
- Αν υπάρχει ακμή μεταξύ του A και ενός άλλου κόμβου **X** και από τον X προς τον κόμβο B, τότε

→ Πρόσθεσε το άθροισμα των ακμών τους (αφού έχουμε πολυγάφο) του $A \rightarrow X$ και $X \rightarrow B$ έστω **Sum**.

→ Όρισε μια νέα ακμή από το X στο N με αριθμό ακμών = Sum.

- Διαγράφουμε τους κόμβους A και B και όλες τις ακμές με τις οποίες συνδέονταν, τις μετακινούμε πάνω στον καινούριο κόμβο N .

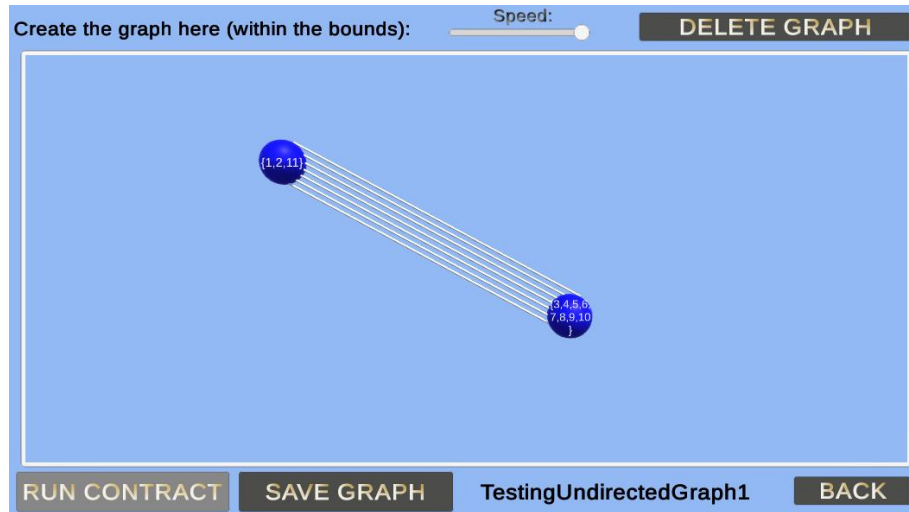
```
1 public bool mergeEdge(int edge_no)
2 {
3     // Get the src and dst nodes of that edge
4     GraphNode srcNode = edgeList[edge_no].GetSourceNode();
5     GraphNode dstNode = edgeList[edge_no].GetDestinationNode();
6
7     // Get the centre point of an edge
8     Vector3 centrePoint = FindCentrePointFromEdge(edge_no);
9
10    // Delete the current selected edge
11    deleteEdge(srcNode, dstNode);
12
13    // Merge the values of the two nodes into a new node!
14    List<int> new_node_values = srcNode.getValues().ToList();
15    for (int i = 0; i < dstNode.getValues().Count; i++)
16        new_node_values.Add(dstNode.getValues()[i]);
17    new_node_values.Sort();
18
19    // Create a new node with the new values in the middle
20    GraphNode newNode = new GraphNode(new_node_values, centrePoint);
21
22    // Add it to the graph
23    addNode(newNode);
24
25    // Add an edge to all of the edges connected with the DST node from
26    // all the dst nodes of all the edges from the SRC node
27    // of the edge we want to merge
28    for (int i = 0; i < nodeList.Count; i++)
29    {
30        // if edge existst SRC->X
31        if (edgeExists(srcNode, nodeList[i]) != null)
32            // and edge exists X->DST
33            if (edgeExists(nodeList[i], dstNode) != null)
34            {
35                // Add all the edges of SRC->X AND X->DST together = SUM
36                // delete those edges and create a new one X->NEW_NODE with SUM edges
37                int newNumberOfEdges = edgeExists(srcNode, nodeList[i]).numberOfEdges + edgeExists(nodeList[i], dstNode).numberOfEdges;
38                deleteEdge(nodeList[i], dstNode);
39                deleteEdge(srcNode, nodeList[i]);
40                addEdge(nodeList[i], newNode, newNumberOfEdges);
41            }
42    }
43
44    // Move all edges from dst and src node to the New Node!
45    moveEdges(dstNode, newNode);
46    moveEdges(srcNode, newNode);
47    return true;
48 }
```

Εκτέλεση Αλγορίθμου Karger – Συγχώνευση ακμής - Σχήμα 7.11

Ο αλγόριθμός μας εκτελείται επαναληπτικά (Σχήμα 7.10) μέχρι να καταλήξει σε ένα πολυγράφο με μόνο 2 κόμβους. Όπως βλέπουμε το αποτέλεσμα μας στη περίπτωση μας είναι ένας γράφος με 2 κόμβους: $A \rightarrow \{1,2,3,4,5,7,11\}$, $B \rightarrow \{6,8,9,10\}$ και ακμή μεταξύ τους με 8 γραμμές. Συνεπώς αν διαχωρίζαμε τον αρχικό μας γράφο σε αυτά τα δύο μέρη A και B , θα είχαμε 8 ακμές που θα δρούσαν ως γέφυρες σε αυτό τον διαχωρισμό.

Ο αλγόριθμός μας όμως είναι τυχαιοποιημένος. Έτσι με μία δεύτερη εκτέλεσή του μπορεί να μας δώσει ένα διαφορετικό αποτέλεσμα. Για παράδειγμα χρησιμοποιώντας και πάλι

τον ίδιο γράφο, έχουμε το πιο κάτω αποτέλεσμα μετά από μια άλλη εκτέλεση του αλγορίθμου (Σχήμα 7.12).



Εκτέλεση Αλγορίθμου Karger – δεύτερο αποτέλεσμα - Σχήμα 7.12

Παρατηρούμε ότι ο αλγόριθμος μας έχει παράξει αυτή τη φορά ένα γράφο με 2 κόμβους: $A \rightarrow \{1,2,11\}$, $B \rightarrow \{3,4,5,6,7,8,9,10\}$ και ακμή μεταξύ τους με 7 γραμμές. Έτσι φαίνεται το τυχαιοποιημένο αντίκτυπο του αλγορίθμου Karger.

7.4 Δυσκολίες/Προβλήματα και Τρόπος Αντιμετώπισής τους

Γενικά μετά από έρευνα που έχει γίνει, δεν βρέθηκε έτοιμος πολυγράφος υλοποιημένος στην C# για ευκολότερή μας υλοποίηση. Συνεπώς όλα τα οποία έχουμε περιγράψει πιο πάνω είναι υλοποιημένα από την αρχή και ο γράφος μας αποτελεί μια καινούρια δομή δεδομένων που μπορεί να χρησιμοποιηθεί επεκτατικά για εμφάνιση 3D γράφου.

Για την υλοποίηση πολλαπλών ακμών μεταξύ κόμβων χρειαστήκαν ορισμένα μαθηματικά για υπολογισμούς σημείων κλπ που χρησιμοποιούνται συνήθως στα γραφικά. Αυτό ήταν λίγο εκτός εμβέλειας της γνώσης μας. Έτσι αποταθήκαμε στον Δρ. Παναγιώτη Χαραλάμπους για την εξακρίβωση ορισμένων πράξεων και τύπων για την εφαρμογή στην υλοποίησή μας και βοήθεια από μάθημα γραφικών.

Προβλήματα στη υλοποίηση της συγχώνευσης. Παρατηρήσαμε ότι ήταν δύσκολο στην εύρεση τρόπου για να συγχωνεύσουμε τους κόμβους μιας ακμής και να εμφανίσουμε στη

συνέχεια πολλαπλές άλλες. Μετά από αρκετή σκέψη όμως και πολλές δοκιμές με λάθη έχουμε καταλήξει στη πρωτότυπη λύση που περιγράψαμε προηγουμένως όπως φαίνεται στο σχήμα 7.11.

Για εμφάνιση λίστας με τους αποθηκευμένους γράφους χρειάστηκε υλοποίηση από την αρχή καθώς το Unity δεν προσφέρει έτοιμη λίστα Scroll List όπως προσφέρουν άλλα προγράμματα δημιουργίας διεπαφών [19].

Στο Save System μας, η αποθήκευση των αρχείων με χρήση του έτοιμου εργαλείου του Unity το οποίο παρουσιάζει και παράθυρο για επιλογή ονόματος και τοποθεσίας αποθήκευσης δεν ήταν συμβατό στο εκτελέσιμο αρχείο του προγράμματός μας καθώς είναι ένα εργαλείο που δουλεύει μονάχα στο Unity Editor. Έτσι έπρεπε να υλοποιήσουμε εξ αρχής μια απλή διεπαφή αποθήκευσης και άλλη για ανάκτηση αρχείων για τους γράφους μας.

Κεφάλαιο 8.

Επίλογος - Συμπεράσματα

8.1 Επίλογος	98
8.2 Κυριότερα Προβλήματα και Τρόπος Αντιμετώπισης	100
8.3 Μελλοντική Εργασία και Επεκτάσεις	101

8.1 Επίλογος

Συνοψίζοντας, με την υφιστάμενη διπλωματική εργασία έχουμε υλοποιήσει ένα μεγάλο χρήσιμο εργαλείο το οποίο παρουσιάζει οπτικά με τη χρήση τρισδιάστατων γραφικών διάφορους αλγορίθμους. Οι αλγόριθμοι που έχουν επιλεγεί για την υλοποίησή τους ήταν επαρκείς για μια όμορφη εμφάνιση και καλύτερη κατανόησή τους μετά την ολοκλήρωσή τους.

Αρχικά μετά την περιγραφή του εργαλείου Unity3D και ορισμένων σημαντικών όρων που χρησιμοποιούνται, υλοποιήσαμε ως εξάσκηση τον αλγόριθμο “Selection Sort”. Η υλοποίηση αυτού του αλγορίθμου ήταν σημαντική καθώς έχουμε υλοποιήσει σημαντικά εργαλεία για εμφάνιση και κίνηση 3D αντικειμένων. Υλοποιήσαμε γραμμική παρεμβολή 3D σφαιρών, τόσο κάθετα όσο και οριζόντια, αλλά και κίνηση σφαιρών με χρήση ενός script. Αυτές οι υλοποιήσεις έχουν χρησιμοποιηθεί και αργότερα σε υλοποιήσεις μερικών από τους υπόλοιπους μας αλγορίθμους.

Στην συνέχεια προχωρήσαμε με την υλοποίηση του αλγορίθμου “Fast Fourier Transform” ο οποίος δυσκόλευε σε μεγάλο βαθμό τους φοιτητές στην κατανόησή του. Η υλοποίηση ήταν επιτυχής προσφέροντας διαδραστικές κινήσεις αντικειμένων. Υλοποιήθηκε τρισδιάστατη στοίβα, δοχεία που μπορείς να τοποθετήσεις σφαίρες μέσα και τρισδιάστατα κουτιά για παρουσίαση αποτελεσμάτων.

Προχωρήσαμε στην υλοποίηση του αλγορίθμου επιλογής. Η κατανομή των στοιχείων του πίνακα σε κάθε βήμα του αλγορίθμου με χρήση animation βοηθά σε μεγάλο βαθμό στην κατανόησή του. Έχει εφαρμοστεί οπτική αναπαράσταση του πίνακα, των ομάδων και δοχείων διαμοιρασμού στοιχείων πάλι με την χρήση σφαιρών ως αναπαράσταση των στοιχείων μας.

Η τεχνική του δυναμικού προγραμματισμού είναι μια δύσκολη αρχικά στην κατανόηση τεχνική. Με την παρουσίαση ενός απλού αλγορίθμου, στην περίπτωσή μας έχουμε επιλέξει τον αλγόριθμο “Fibonacci”, έχουμε πετύχει καλύτερη παρουσίαση της μεγάλης βελτίωσης που προσφέρει εφαρμόζοντάς τον πάνω σε ένα δύσκολο πρόβλημα. Έχουμε υλοποιήσει ένα 3D δυαδικό δένδρο παρέχοντας όλες τις λειτουργίες και ιδιότητές του προγραμματιστικά, για παρουσίαση των αναδρομικών εκτελέσεων του αναδρομικού Fibonacci. Στην συνέχεια παρουσιάσαμε τον χρόνο εκτέλεσής του με χρήση μεθόδου απομνημόνευσης και με τον δυναμικό προγραμματισμό. Συγκρίνοντας κάθε εκτέλεση παρατηρούμε τα θετικά που προσφέρει ο δυναμικός στην βελτίωση πολυπλοκότητας.

Τέλος έχουμε δημιουργήσει εξ 'αρχής ένα 3D μη-συνεκτικό πολυγράφο για την παρουσίαση του τυχαιοποιημένου αλγορίθμου συρρίκνωσης Karger. Ακόμη μια επιτυχής υλοποίηση, προσφέροντας ένα εργαλείο με το οποίο μπορείς να φτιάξεις τον δικό σου γράφο, να τον αποθηκεύσεις και να τον ανακτήσεις και συνεπώς να τρέξεις τον αλγόριθμο συρρίκνωσης. Μετά από πολλές εκτελέσεις του αλγορίθμου για τον ίδιο γράφο βλέπουμε ότι έχουμε πολυωνμικά μικρή πιθανότητα επιτυχής εύρεσης μιας καθολικά ελάχιστης αποκοπής.

Μετά από πολλές παρουσιάσεις υλοποιήσεων σε ένα κοινό από φοιτητές, φίλους και γνώστες της πληροφορικής έχουμε βελτιώσει σε μεγάλο βαθμό όλα τα οποία έχουν υλοποιηθεί. Εκτός από αυτό, η ανάδραση που έχει δώσει αυτή η ομάδα ανθρώπων μας δείχνει ότι ο σκοπός μας έχει επιτευχθεί σε μεγάλο βαθμό και η αναπαράσταση των αλγορίθμων, τους βοηθά στην επιπλέον κατανόηση. Η παρουσίαση του προγράμματος σε “Beta” μορφή στο μάθημα «Αλγόριθμοι και πολυπλοκότητα» έχει πάρει καλές κριτικές από τους φοιτητές και έχει κάνει πιο ευχάριστη την διδασκαλία των αλγορίθμων αυτών.

8.2 Κυριότερα προβλήματα και τρόπος αντιμετώπισης

Σε αυτό το σημείο θα αναφερθούμε στα κυριότερα και γενικότερα προβλήματα που έχουν εμφανιστεί στη διπλωματική αυτή εργασία. Ως επακόλουθο θα περιγράψουμε υπαρκτούς ή πιθανούς τρόπους επίλυσης για το κάθε μας πρόβλημα.

- Δυσκολία στην εφαρμογή animations πάνω σε 3D αντικείμενα. Αντί να χρησιμοποιήσουμε το εργαλείο με τα animations έχει γίνει συγγραφή scripts τα οποία εφαρμόζουν κινήσεις με χρήση γραμμικών και μη-γραμμικών εξισώσεων.
- Προσαρμογή ευκρίνειας και μεγέθους οθόνης → Έχουμε μελετήσει περισσότερο τις κάμερες που προσφέρει το Unity, οι οποίες χρησιμοποιούνται για την εμφάνιση της σκηνής μας προς το χρήστη. Μελλοντικά μπορεί να βελτιωθεί.
- Μετατροπή επαναληπτικών αλγορίθμων από αναδρομικούς σε επαναληπτικούς για οπτικοποίηση. Εδώ έχει παρουσιαστεί μεγάλη δυσκολία καθώς είχαμε δυσκολία στην κατανόηση της έτοιμης μεθόδου μετατροπής από αναδρομικό σε επαναληπτικό κώδικα. Έτσι μετά από πολλές δοκιμές και λάθη (trial and error) έχουμε καταλήξει στους υφιστάμενους αλγορίθμους οι οποίοι έχουν υλοποιηθεί με δικό μας τρόπο για καλύτερη εμφάνισή τους οπτικά.
- Γενικές δυσκολίες στην εφαρμογή μαθηματικών που διδάσκονται σε μαθήματα γραφικών για μετατοπίσεις και μετατροπές αντικειμένων σε διάφορα μέρη. Έπρεπε να γίνει εκμάθηση αυτών των μαθηματικών όρων και εξισώσεων για να μπορέσουμε να τα εφαρμόσουμε σε κώδικα και πάνω στα αντικείμενα μας.
- Μείωση επανάληψης κομματιού κώδικα στην ίδια Coroutine. Μαθαίνοντας για την δυνατότητα χρήσης Coroutine μέσα σε Coroutine έχουμε μειώσει την αχρείαστη επανεμφάνιση κώδικα.
- Αποθήκευση δεδομένων σε αρχείο. Το εργαλείο Unity προσέφερε ορισμένες κλάσεις και αντικείμενα τα οποία δεν μπορούσαν να αποθηκευθούν με χρήση ενός απλού Serializable. Μετά από πολλές έρευνες και προσπάθειες υλοποίησης έχουμε δημιουργήσει κλάσεις και μεθόδους αποφυγής αυτών των συγκρούσεων και έχουμε καταφέρει επιτυχής αποθήκευση ενός τρισδιάστατου πολυγράφου.

8.3 Μελλοντική Εργασία και Επεκτάσεις

Το πρόγραμμά μας έχει μεγάλο φάσμα μελλοντικών βελτιώσεων και επεκτάσεων. Ορισμένες από τις μελλοντικές εργασίες αφορούν γενικά την εμφάνιση και υλοποίηση του προγράμματος και άλλες αφορούν βελτιώσεις που θα μπορούσε να έχει ο κάθε αλγόριθμος ξεχωριστά. Αρχικά θα επικεντρωθούμε στις γενικές πιθανές μελλοντικές εργασίες που θα μπορούσαν να εφαρμοστούν και στην συνέχεια θα αναφερθούμε σε κάθε αλγόριθμο ξεχωριστά.

Γενικά πάνω σε αυτή την εργασία μπορούν να εφαρμοστούν διάφορες αλλαγές και βελτιώσεις. Συγκεκριμένα τα πιο κάτω είναι ορισμένοι στόχοι που θα μπορούσαν βελτιώσουν τόσο εμφανισιακά όσο και στην υλοποίηση:

1. Χρήση πιο λεπτομερών με καλύτερη ποιότητα 3D μοντέλων που θα κάνουν πιο ευχάριστη την θέαση των αλγορίθμων
2. Προσθήκη ήχων και ηχητικών εφέ πάνω στις κινήσεις αντικειμένων και γενικά για την ατμόσφαιρα του προγράμματος
3. Περισσότερες επιλογές στα “Options” της κύριας διεπαφής. Για παράδειγμα, αλλαγή θέματος, αυξομείωση έντασης ήχου, ευκρίνεια οθόνης κλπ.
4. Βελτίωση μεγεθών κουμπιών και εργαλείων χειρισμού καθώς και χρωμάτων όλων των διεπαφών μας.
5. Βελτίωση και επέκταση επιλογών ευκρίνειας και μεγέθους οθόνης για ένα πιο δυναμικό πρόγραμμα που θα προσαρμόζεται σε διάφορες συσκευές και οθόνες.
6. Επέκταση υλοποίησής και σε κινητά τηλέφωνα Android και IOS καθώς η διεπαφή μας μπορεί να δουλεύει και με touch controls στη θέση του ποντικιού.
7. Βελτίωση αργοπορίας εκτέλεσης και εξόδου. Ίσως φταίει ο τρόπος που έχουμε υλοποιήσει γενικά το πρόγραμμά μας.

Ο κάθε αλγόριθμος είναι υλοποιημένος ξεχωριστά. Οι πλείστες ιδέες για βελτίωση και επέκταση έχουν εμφανιστεί στην πορεία της υλοποίησής μας όπου και ήταν πολύ αργά για

εφαρμογή αλλαγών. Πιο κάτω περιγράφονται ορισμένες πιθανές μελλοντικές αλλαγές και προσθήκες για τον κάθε αλγόριθμο:

Αλγόριθμος FFT:

- Χρήση υλοποιημένου δυαδικού δένδρου από τον Δυναμικό προγραμματισμό για παρουσίαση κάθε αλλαγής βάθους και παρουσίασης κάθε ομάδας περιττών και άρτιων αριθμών. Με αυτό τον τρόπο θα μπορεί ο χρήστης να κατανοήσει το πώς διασπάται και εκτελείται ο αλγόριθμος.
- Παρουσίαση γραφικής απεικόνισης υπολογισμού μιγαδικών ριζών με χρήση ενός κύκλου που δουλεύει όπως το ρολόι όταν διαμοιράζεται και εμφανίζονται οι ρίζες που χειρισόμαστε μια συγκεκριμένη στιγμή στον αλγόριθμο.

Αλγόριθμος Επιλογής:

- Με ένα restart ξαναξεκινά ο αλγόριθμος με τα ίδια δεδομένα εισόδου
- Βελτίωση της μεταφοράς των σφαιρών όλων των διαμέσων στο κεντρικό δοχείο με όλες τις διαμέσους. Γενικά μπορεί να χρησιμοποιηθεί πιο εξειδικευμένο animation.

Δυναμικός Προγραμματισμός – Fibonacci:

- Εξαγωγή αποτελεσμάτων σε εικόνα ή PDF με το δένδρο που υπολογίζεται

Τυχαιοποιημένος αλγόριθμος Karger:

- Δυνατότητα τροποποίησης ανακτημένου γράφου από αρχείο “Edit graph”.
- Βελτίωση των animations.

Αν υλοποιούσαμε το πρόγραμμά αυτό ξανά από την αρχή θα ήταν εντελώς διαφορετικό. Πρώτο και κύριο έχουμε αποκτήσει άφθονες γνώσεις ως προς το εργαλείο Unity και όλων των επιλογών που το απαρτίζουν. Έτσι θα είχαμε υλοποιήσει πιο όμορφα και ευδιάκριτα, με περισσότερες λειτουργίες τρισδιάστατα αντικείμενα με ευέλικτο animation. Επίσης αντί

να είχαμε γνώση μονάχα της γλώσσας Java και επιφανειακών όρων αντικειμενοστρέφειας, με την πλήρη μάθηση της γλώσσας C# και επιπλέον μελέτη χρήσης αντικειμένων, κληρονομικότητας και άλλων συμβάσεων της αντικειμενοστρέφειας, θα είχαμε ένα πολύ πιο καλύτερο αποτέλεσμα. Ο κώδικάς μας θα ήτανε και σε βελτιστοποιημένη-Optimized μορφή αν ξεκινούσαμε ξανά από την αρχή, ακολουθώντας όλες τις αναγκαίες συμβάσεις γραφής και σύνταξης της C#. Τέλος θα είχαμε απορρίψει επιπλέον αχρείαστες υλοποιήσεις και η ταχύτητα υλοποίησης θα ήταν πολύ πιο μεγάλη με πιο συγυρισμένο πρόγραμμα. Αν είχαμε περισσότερες γνώσεις περί γραφικών και εργαλείου Unity (δηλαδή ταυτόχρονα και Gaming) θα είχαμε ένα πολύ καλύτερο αποτέλεσμα.

Βιβλιογραφία

- [1] Κύριες κεντρικές σελίδες εργαλείου Unity: <https://unity.com/>, <https://unity3d.com/unity>, Last accessed 10/05/2019
- [2] Οδηγός χρήσης εργαλείου Unity: <https://docs.unity3d.com/Manual/index.html>, Last accessed 10/05/2019
- [3] Βασικά μαθήματα εκμάθησης Unity: <https://unity3d.com/learn/tutorials>, Last accessed 10/05/2019
- [4] Σημειώσεις και διαλέξεις ΕΠΛ236 Αλγόριθμοι και πολυπλοκότητα, Διδάσκων Καθηγητής: Χρύσης Γεωργίου, <https://www.cs.ucy.ac.cy/~chryssis/EPL236/>, Last accessed 10/05/2019
- [5] Kleinberg, J., & Tardos, E. (2006). *Algorithm design*. Pearson Education India και η μετάφραση του στα ελληνικά: *Σχεδιασμός Αλγορίθμων*, ελληνική έκδοση. Εκδόσεις Κλειδάριθμος.
- [6] Παντελής Στυλιανίδης, Οπτικοποίηση Παράλληλων αλγορίθμων με τη μηχανή Unity, Διπλωματική εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, 2016
- [7] Για παραβολική κίνηση σφαιρών:
Moving Object Along A Parabola | Unity Tutorial – DitzelGames:
<https://www.youtube.com/watch?v=ddakS7BgHRI>, Last accessed 10/05/2019
ditzel/MathParabola.cs – GitHub:
<https://gist.github.com/ditzel/68be36987d8e7c83d48f497294c66e08>, Last accessed 10/05/2019
- [8] Έλεγχος αποτελεσμάτων FFT – FFT Calculator:
<http://scistatcalc.blogspot.com/2013/12/fft-calculator.html>, Last accessed 10/05/2019
- [9] Διαλέξεις μαθήματος «EPL231 Algorithms and data structures» -
<http://www.cs.ucy.ac.cy/courses/EPL231/lectures.html>, Last accessed 10/05/2019
- [10] GeeksForGeeks (A computer science portal for geeks) (Χρήση για επεξήγηση ορισμένων αλγορίθμων) <https://www.geeksforgeeks.org/>, Last accessed 10/05/2019
- [11] Multithreading for Dynamic Programming – Threading in C#:
<http://www.albahari.com/threading/>, Last accessed 10/05/2019
- [12] Ιδέες για δημιουργία 3D δυαδικού δένδρου:
Binary Tree data structure: <https://www.geeksforgeeks.org/binary-tree-data-structure/>, Last accessed 10/05/2019

Implementing a Binary tree in Java: <https://www.baeldung.com/java-binary-tree>, Last accessed 10/05/2019

[13] Fibonacci: Recursive programming, Memoization, Dynamic Programming: <https://perlmaven.com/memoization-caching-of-return-values>, Last accessed 10/05/2019
http://courses.csail.mit.edu/6.006/fall09/lecture_notes/lecture18.pdf, Last accessed 10/05/2019
<https://www.cs.cmu.edu/~adamchik/15-121/lectures/Recursions/recursions.html>, Last accessed 10/05/2019

[14] Μετατροπή κειμένου σε String - String Builder for C# (Dynamic Programming - Algorithm text box) <https://www.buildmystring.com>, Last accessed 10/05/2019

[15] Γραμμές και καμπύλες: <https://catlikecoding.com/unity/tutorials/curves-and-splines/>, Last accessed 10/05/2019

[16] Checking if two lists (eg. List<int>) have the same numbers: <https://stackoverflow.com/questions/22173762/check-if-two-lists-are-equal>, Last accessed 10/05/2019

[17] Karger's Contract algorithm for Minimum Cut: https://en.wikipedia.org/wiki/Karger%27s_algorithm, Last accessed 10/05/2019

[18] Allow and detect mouse clicks on a specific area: <https://stackoverflow.com/questions/10111821/unity3d-how-to-detect-click-within-an-area>, Last accessed 10/05/2019

Convert rect transform rect to screen space (to draw a box): <https://answers.unity.com/questions/1013011/convert-recttransform-rect-to-screen-space.html>, Last accessed 10/05/2019

[19] Scroll View – Load Menu for Randomization Algorithm: <https://unity3d.com/learn/tutorials/topics/user-interface-ui/scroll-view>, Last accessed 10/05/2019
<https://www.youtube.com/watch?v=Zl6DwJtjIBA&v=en>, Last accessed 10/05/2019

[20] GameObject Pooling: <https://unity3d.com/learn/tutorials/topics/scripting/object-pooling?playlist=17117>, Last accessed 10/05/2019

[21] Main UI Background Image: <http://www.bris.ac.uk/engineering/research/theory-algorithms/>, Last accessed 10/05/2019

[22] Sublime Text – Text Editor: <https://www.sublimetext.com/>, Last accessed 10/05/2019

[23] Draw.io – Free online diagram software: <https://www.draw.io/>, Last accessed 10/05/2019

- [24] FullTiltBoogie – Starter Particle Pack (used for fireworks):
<https://assetstore.unity.com/packages/vfx/particles/starter-particle-pack-83179>, Last accessed 10/05/2019
- [25] Saving and Loading via Serialization (Randomization algorithm):
Script Serialization: <https://docs.unity3d.com/Manual/script-Serialization.html>, Last accessed 10/05/2019
- [26] Serialization of a GameObject: <https://stackoverflow.com/questions/36852213/how-to-serialize-and-save-a-gameobject-in-unity>, Last accessed 10/05/2019
- [27] Serialization of quaternions and Vector 3:
<https://answers.unity.com/questions/956047/serialize-quaternion-or-vector3.html>, Last accessed 10/05/2019
- [28] Saving a List of a class: <https://answers.unity.com/questions/977881/save-a-list-of-a-class.html>, Last accessed 10/05/2019
- [29] Save and Load System in Unity: https://www.youtube.com/watch?v=XOjd_qU2ldo, Last accessed 10/05/2019
- [30] Persistence – Saving and Loading Data:
<https://unity3d.com/learn/tutorials/topics/scripting/persistence-saving-and-loading-data>, Last accessed 10/05/2019
- [31] Saving, Writing and Loading from a File:
<https://answers.unity.com/questions/1300019/how-do-you-save-write-and-load-from-a-file.html>, Last accessed 10/05/2019

Παράρτημα Α

Σε αυτό το παράρτημα υπάρχουν κομμάτια κώδικα σε C# τα οποία χρησιμοποιήθηκαν κατά την υλοποίηση του πρώτου αλγορίθμου (Selection Sort). Ορισμένα από αυτά τα scripts είναι σημαντικά για αργότερη χρήση στην υλοποίηση άλλων αλγορίθμων.

BallMover.cs :

```
using UnityEngine;

public class BallMover : MonoBehaviour {

    public Vector3 startPos;          /*The starting/initial position of the sphere*/
    public Vector3 target;            /*The target position where the sphere is going to move to*/

    public bool active = false;       /* If active then the sphere is moving*/
    public float height = 2;          /* The height of the parabola curve*/
    public static float speed = 0.3f; /* The speed the sphere moves from start pos to target */

    private float time; /*The time the sphere needs to go from startPos to target*/

    // Initialization of variables
    void Start () {
        height = 2;
        time = 0;
        startPos = transform.position;
    }

    /**
     * updatePosition
     * Used to update the starting position with the current pos.
     */
    public void updatePosition()
    {
        startPos = transform.position;
    }

    /**
     * SetTarget
     * Used to set a desired target
     */
    public void SetTarget(Vector3 t)
    {
        this.target = t;
    }

    /**
     * StartMoving
     * Activates the script
     */
    public void StartMoving()
    {
        active = true;
    }

    // FixedUpdate is called once per frame
    void FixedUpdate () {

        // When ths script is active, move the current GameObject this
        // script is attached from current Start Pos to target position
        // (We use the parabola method of MathParabola where you can set
```

```

        // the height of the parabola curve and the time as an interpolation
        // value)
        if (active)
        {
            time += Time.deltaTime;

            MathParabola mp = new MathParabola();

            transform.position = mp.Parabola(startPos, target, height, time);
        }

        // When the position of the sphere reaches the target, stop moving
        if (transform.position == target)
        {
            time = 0;
            startPos = transform.position;
            active = false;
        }
    }
}

```

MathParabola.cs :

```

using System;
using UnityEngine;

/*MathParabola script: Contains interpolation functions for 2D/3D parabolic curves*/
public class MathParabola{

    public Vector3 Parabola(Vector3 start, Vector3 end, float height, float t)
    {
        Func<float, float> f = x => -4 * height * x * x + 4 * height * x;
        var mid = Vector3.Lerp(start, end, t);
        return new Vector3(mid.x, f(t) + Mathf.Lerp(start.y, end.y, t), mid.z);
    }

    public Vector3 inverseParabola(Vector3 start, Vector3 end, float height, float t)
    {
        Func<float, float> f = x => -4 * height * x * x + 4 * height * x;
        var mid = Vector3.Lerp(start, end, t);
        return new Vector3(mid.x, f(t) + Mathf.Lerp(start.y, end.y, t), mid.z);
    }

    public Vector2 Parabola(Vector2 start, Vector2 end, float height, float t)
    {
        Func<float, float> f = x => -4 * height * x * x + 4 * height * x;
        var mid = Vector2.Lerp(start, end, t);
        return new Vector2(mid.x, f(t) + Mathf.Lerp(start.y, end.y, t));
    }
}

```

ReadFile.cs :

```

using System;
using System.IO;
using UnityEngine;

/**
 * ReadFile script
 * provides tools to read a file containing float numbers
 * used in the Selection Sort algorithm.
 */

```

```

public class ReadFile{

    // Size of the array with the numbers
    private int size;

    /**
     * readFile function
     * Used to read float numbers from a given file
     */
    public float[] readFile(string file)
    {
        float[] arr;

        //"Assets/Resources/Input.txt"
        string filePath = file;

        StreamReader reader = new StreamReader(filePath);
        String line;

        //in the beginning of the line is an integer (size of the file)
        size = int.Parse(reader.ReadLine());

        arr = new float[size];
        int count = 0;
        while ((line = reader.ReadLine()) != null)
        {
            float num;
            if (!float.TryParse(line, out num))
            {
                Debug.LogError("Error!\n" + line + "\n can't be converted into float!");
            }

            arr[count] = num;
            count++;

            // Check if the file correctly has floats and the right
            // amount of numbers
            if (count > size)
            {
                Debug.LogError("Error! Size of the array not matching size given in file!");
            }
        }
        return arr;
    }
}

```

Point.cs :

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/**
 * Point class
 * Used to create a point object containing an x and y integer value
 */
public class Point {

    private int x;
    private int y;

    public Point()
    {

    }

}

```

```

public Point(int x, int y)
{
    this.x = x;
    this.y = y;
}

public int getX()
{
    return x;
}

public int getY()
{
    return y;
}

public void setX(int x)
{
    this.x = x;
}

public void setY(int y)
{
    this.y = y;
}
}

```

SelectionSortByFile.cs :

```

using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

/**
 * Visual Selection Sort using values from a file
 */
public class SelectionSortByFile : MonoBehaviour {

    // Variables and objects provided and being able to
    // get from the inspector of where this script is
    // attached onto.
    public Transform StartPositionPlaceholder;
    public Transform EndPositionPlaceholder;
    public int NumberOfObjectsToPlace;
    public Transform PlacableObject;
    public GameObject PointingArrow;
    public Transform StepText;
    public TextMeshProUGUI pauseButtonText;

    /*The placed spheres containing the numbers of the array*/
    public List<Transform> PlacedTransforms;

    /*Private variables*/
    private Vector3 startPos;
    private Vector3 endPos;
    private float[] numArray;
    private IEnumerator swap;
    private Color defaultCol;
    private GameObject backBtn;
    private GameObject forwardBtn;

    // Used for the buttons controlling the algorithm

```

```

private bool StartBtn;
private bool PauseBtn;
public bool useStepByStep;
private bool stepForward;
private bool stepBack;
private float speed;

// Use this for initialization
void Start()
{
    //Back button to be disabled/enabled
    backBtn = GameObject.FindGameObjectsWithTag("stepBackBtn")[0];
    backBtn.SetActive(false);

    //Forward button to be disabled/enabled
    forwardBtn = GameObject.FindGameObjectsWithTag("stepForwardBtn")[0];
    forwardBtn.SetActive(false);

    startPos = StartPositionPlaceholder.localPosition;
    endPos = EndPositionPlaceholder.localPosition;

    GameObject.Destroy(StartPositionPlaceholder.gameObject);
    GameObject.Destroy(EndPositionPlaceholder.gameObject);

    PlaceObjectsApart();

    defaultCol = PlacedTransforms[0].GetComponent<Renderer>().material.color;

    StartBtn = false;
    PauseBtn = false;
    speed = 2f;
    StartCoroutine("SelectionSort");
}

/*UI FUNCTIONS (used by the UI buttons and other elements)*/
public void SetStart(bool b)
{
    StartBtn = b;
}
public void SetPause()
{
    PauseBtn = !PauseBtn;
    if (!PauseBtn)
    {
        pauseButtonText.text = "PAUSE";
    }
    else
    {
        pauseButtonText.text = "RESUME";
    }
}
public void SetStepByStep(bool b)
{
    useStepByStep = b;
}
public void SetStepForward(bool b)
{
    stepForward = b;
}
public void SetStepBack(bool b)
{
    stepBack = b;
}
public void SetSpeed(float value)
{
    speed = value;
}

```

```

        BallMover.speed = value;
    }
    public void BackButton()
    {
        SceneManager.LoadScene("MainUI");
    }
    /*END OF UI FUNCTIONS*/

    /**
     * PlaceObjectsAppart function
     * Used to linearly interpolate between two positions
     * and place objects apart between them. (Vertically and horizontally)
     */
    private void PlaceObjectsAppart()
    {
        // Read the input file containing the numbers we need
        // and transfer them to the array
        ReadFile rf = new ReadFile();
        numArray = rf.readFile("Input.txt");
        NumberOfObjectsToPlace = numArray.Length;

        // We need a maximum of 30 interpolated spheres per line
        float no_of_interpolated_items = 30;

        // Number of rows is defined by the total number of objects
        int noOfRows = Mathf.CeilToInt((float)NumberOfObjectsToPlace / no_of_interpolated_items);
        int row = noOfRows;

        // The current element
        int element = 0;

        // The final remaining number of items based on the maximum number of objects
        // we need to place in each line
        int remainder = NumberOfObjectsToPlace - ((noOfRows-1) * (int)no_of_interpolated_items);

        // Place the objects
        for (int i = 0; i < NumberOfObjectsToPlace; i++)
        {
            // For more than one rows
            if (noOfRows > 1)
            {
                // Set the upper and down Start and End pinpoint edges.
                Vector3 upperStartEdge = new Vector3(startPos.x, startPos.y - 3.5f, startPos.z);
                Vector3 downStartEdge = new Vector3(startPos.x, startPos.y + 5f, startPos.z);
                Vector3 upperEndEdge = new Vector3(endPos.x, endPos.y - 3.5f, endPos.z);
                Vector3 downEndEdge = new Vector3(endPos.x, endPos.y + 5f, endPos.z);

                // Set the scale of the sphere to correctly fit the line
                float siz = (float)(19.0f / (float)no_of_interpolated_items);
                PlacableObject.transform.localScale = new Vector3(siz, siz, siz);

                // Every 30 items we change row
                if (i% no_of_interpolated_items == 0)
                {
                    row--;
                    element = 0;
                }

                // Set the text value of the object to display the current array value
                PlacableObject.GetChild(0).GetComponent<TextMeshPro>().text = numArray[i] + "";

                // If we are on row 0 we instantiate the remaining objects
                Transform instantiatedObject;
                if (row == 0)
                {
                    instantiatedObject =

```

```

        Instantiate(
            PlacableObject,
            Vector3.Lerp(Vector3.Lerp(upperStartEdge, downStartEdge, (float)row /
(float)noOfRows),
                Vector3.Lerp(upperEndEdge, downEndEdge, (float)row /
(float)noOfRows),
                (float)element / (float)(remainder)),
            Quaternion.LookRotation(Vector3.forward));
    }

    // else on all the remaining rows we instantiate 30 objects at a time
    else
    {
        instantiatedObject =
            Instantiate(
                PlacableObject,
                Vector3.Lerp(Vector3.Lerp(upperStartEdge, downStartEdge, (float)row /
(float)noOfRows),
                    Vector3.Lerp(upperEndEdge, downEndEdge, (float)row /
(float)noOfRows),
                    (float)element / (float)(no_of_interpolated_items)),
                Quaternion.LookRotation(Vector3.forward));
    }
    element++;
    PlacedTransforms.Add(instantiatedObject);
}

// If we only have one row only then we can change the size
// of the spheres too!
else
{
    // Set the scale of the sphere
    float siz = (float)(19.0f / (float)no_of_interpolated_items);
    //float siz = 8.0f / NumberOfObjectsToPlace;

    // Set the text inside the ball.
    PlacableObject.GetChild(0).GetComponent<TextMeshPro>().text = numArray[i] + "";

    Transform instantiatedObject =
        Instantiate(
            PlacableObject,
            Vector3.Lerp(startPos, endPos, (float)i / (float)(NumberOfObjectsToPlace -
1)),
            Quaternion.LookRotation(Vector3.forward));
    instantiatedObject.localScale = new Vector3(siz, siz, siz);
    PlacedTransforms.Add(instantiatedObject);
}
}

}

/**
 * SelectionSort coroutine
 * the selection sort algorithm using visualization
 */
public IEnumerator SelectionSort()
{
    yield return new WaitUntil(() => StartBtn);

    // Stack used to store the previous steps of the algorithm (with Point object (i,min))
    Stack<Point> prevSteps = new Stack<Point>();

    for (int i = 0; i < numArray.Length; i++)
    {
        int min = i;
        for (int j = i + 1; j < numArray.Length; j++)
        {

```

```

        //StepText.GetComponent<TextMeshProUGUI>().text = numArray[j]+" <
"+numArray[min]+"?";
        //yield return new WaitForSeconds(1f);
        bool isMin = numArray[j] < numArray[min];
        if (isMin)
        {
            //StepText.GetComponent<TextMeshProUGUI>().text = "Yes!";
            PlacedTransforms[min].GetComponent<Renderer>().material.color = defaultCol;
            min = j;
            StepText.GetComponent<TextMeshProUGUI>().text = "Current minimum: " +
numArray[min];
        }
        else
        {
            //StepText.GetComponent<TextMeshProUGUI>().text = "No!";
            PlacedTransforms[j].GetComponent<Renderer>().material.color = Color.magenta;
            // yield return new WaitForSeconds(0.03f);
            if (PauseBtn)
            {
                yield return new WaitUntil(() => !PauseBtn);
            }
            yield return new WaitForSeconds(speed* 0.01f);

            if (!isMin)
            {
                PlacedTransforms[j].GetComponent<Renderer>().material.color = defaultCol;
            }
        }
        // Indicate the minimum sphere as red colored
        PlacedTransforms[min].GetComponent<Renderer>().material.color = Color.red;

        // Display the two animated arrows over
        // the current objects we may swap.
        ShowArrow(min, "MINIMUM");
        ShowReverseArrow(i, "i");

        StepText.GetComponent<TextMeshProUGUI>().text = numArray[min] + " is less than " +
numArray[i];

        float tmp = numArray[i];
        numArray[i] = numArray[min];
        numArray[min] = tmp;

        if (PauseBtn)
        {
            yield return new WaitUntil(() => !PauseBtn);
        }
        yield return new WaitForSeconds(speed*0.7f);

        Debug.Log("i=" + i + ", min=" + min);
        swap = SwapBalls(i, min);
        if (PauseBtn)
        {
            yield return new WaitUntil(() => !PauseBtn);
        }
        yield return StartCoroutine(swap);

        // Push the previous step into the stack
        prevSteps.Push(new Point(i, min));

        PlacedTransforms[i].GetComponent<Renderer>().material.color = Color.grey;

        // If the user has selected to use the step-by-step option
        if (useStepByStep)
        {

```

```

        // Activate the step back and forward buttons
        backBtn.SetActive(true);
        forwardBtn.SetActive(true);

        // Wait until any of those buttons is pressed or the step-by-step check box gets
disabled
        if (PauseBtn)
        {
            yield return new WaitUntil(() => !PauseBtn);
        }
        yield return new WaitUntil(() => stepBack || stepForward || !useStepByStep);

        // While we have steps that have been executed, we can load them back from
        // the stack using the Back button.
        while (stepBack && prevSteps.Count!=0)
        {
            stepBack = false;
            stepForward = false;
            Point prev = prevSteps.Pop();
            int previ = prev.getX(), prevmin = prev.getY();

            if (prevSteps.Count == 0)
            {
                backBtn.SetActive(false);
            }

            // Restore previous i position
            i = previ - 1;

            StepText.GetComponent<TextMeshProUGUI>().text = numArray[previ] + " < " +
numArray[prevmin];

            // Show the arrows and swap the elements
            ShowArrow(previ, "MINIMUM");
            ShowReverseArrow(prevmin, "i");
            tmp = numArray[previ];
            numArray[previ] = numArray[prevmin];
            numArray[prevmin] = tmp;

            if (PauseBtn)
            {
                yield return new WaitUntil(() => !PauseBtn);
            }
            yield return new WaitForSeconds(speed);

            Debug.Log("i=" + previ + ", min=" + prevmin);
            swap = SwapBalls(previ, prevmin);
            PlacedTransforms[previ].GetComponent<Renderer>().material.color = Color.grey;
            if (PauseBtn)
            {
                yield return new WaitUntil(() => !PauseBtn);
            }
            yield return StartCoroutine(swap);
            yield return new WaitUntil(() => stepBack || stepForward || !useStepByStep);
        }
    }
    stepBack = false;
    stepForward = false;
}
yield return null;

// STOP COROUTINE AND MAKE ALL BALLS GREEN AND DISPLAY FINISH MESSAGE AS WELL AS START-
>RESTART BUTTON
StepText.GetComponent<TextMeshProUGUI>().text = "Selection sort Complete!";
for (int i=0; i<numArray.Length; i++)
{
    PlacedTransforms[i].GetComponent<Renderer>().material.color = Color.green;
}

```

```

        PlacedTransforms[i].GetChild(0).GetComponent<TextMeshPro>().color = Color.black;
    }

    StartBtn = false;
    GameObject startButton = GameObject.FindGameObjectsWithTag("startBtn")[0];
    startButton.GetComponentInChildren<TextMeshProUGUI>().text = "RESTART";
    StartBtn = false;
    yield return new WaitUntil(() => StartBtn);
    if (StartBtn)
    {
        SceneManager.LoadScene("SelectionSort");
    }
}

/**
 * Displays an animated arrow over the i placed transform pointing down with text
 */
private void ShowArrow(int i, string text)
{
    Vector3 pos = new Vector3(PlacedTransforms[i].position.x, PlacedTransforms[i].position.y +
1f, PlacedTransforms[i].position.z);
    Quaternion rot = PlacedTransforms[i].rotation;
    GameObject arrow = (GameObject)Instantiate(PointingArrow, pos, rot);
    arrow.transform.GetChild(1).GetComponent<TextMeshPro>().text = text;
    arrow.transform.GetChild(0).GetComponent<Renderer>().sortingOrder = 32766; // for the arrow
    arrow.transform.GetChild(1).GetComponent<Renderer>().sortingOrder = 32767; //for the text
    //Destroy(arrow, 2f);
    Destroy(arrow, speed* 0.7f);
}

/**
 * Displays an animated arrow down the i placed transform pointing up with text
 */
private void ShowReverseArrow(int i, string text)
{
    // Inverse the position of the arrow on the y axis
    Vector3 pos = new Vector3(PlacedTransforms[i].position.x, PlacedTransforms[i].position.y -
1f, PlacedTransforms[i].position.z);

    // Rotate the animated arrow 180 degrees on the x-axis.
    Quaternion rot = Quaternion.Euler(new Vector3(PlacedTransforms[i].rotation.x + 180,
PlacedTransforms[i].rotation.y, PlacedTransforms[i].rotation.z));
    GameObject arrow = (GameObject)Instantiate(PointingArrow, pos, rot);
    arrow.transform.GetChild(1).GetComponent<TextMeshPro>().text = text;
    arrow.transform.GetChild(1).transform.Rotate(180, 0, 0);

    //The value must be between -32768 and 32767 (how far behind or closer to the camera is the
object)
    // (the arrow is behind the text and they should all be shown closest to the camera)
    arrow.transform.GetChild(0).GetComponent<Renderer>().sortingOrder = 32766; // for the arrow
    arrow.transform.GetChild(1).GetComponent<Renderer>().sortingOrder = 32767; //for the text
    //Destroy(arrow, 2f);
    Destroy(arrow, speed* 0.7f);
}

/**
 * Swap spheres from->to using the BallMover script component attached on each sphere
 */
private IEnumerator SwapBalls(int from, int to)
{
    // Set the color of the spheres to red
    PlacedTransforms[from].GetComponent<Renderer>().material.color = Color.red;
    PlacedTransforms[to].GetComponent<Renderer>().material.color = Color.red;

    // Swap the transforms from -> to, to -> from

```

```

PlacedTransforms[from].GetComponent<BallMover>().target = PlacedTransforms[to].position;
PlacedTransforms[from].GetComponent<BallMover>().active = true;
PlacedTransforms[to].GetComponent<BallMover>().target = PlacedTransforms[from].position;
PlacedTransforms[to].GetComponent<BallMover>().active = true;

// Swap the locations of the balls in the array
Transform tmp = PlacedTransforms[from];
PlacedTransforms[from] = PlacedTransforms[to];
PlacedTransforms[to] = tmp;

//Wait until both swaps are finished (they become inactive)
if (PauseBtn)
{
    yield return new WaitUntil(() => !PauseBtn);
}
yield return new WaitUntil(()=> !PlacedTransforms[from].GetComponent<BallMover>().active &&
!PlacedTransforms[to].GetComponent<BallMover>().active);
//yield return new WaitForSeconds(1f);

// Restore the color of the spheres to blue
PlacedTransforms[from].GetComponent<Renderer>().material.color = Color.blue;
PlacedTransforms[to].GetComponent<Renderer>().material.color = Color.blue;
if (PauseBtn)
{
    yield return new WaitUntil(() => !PauseBtn);
}
yield return null;
}
}

```

Παράρτημα Β

Πιο κάτω βρίσκονται τα scripts που χρησιμοποιηθήκαν στον έλεγχο των διεπαφών χρήστη σε όλη αυτή την εργασία. Αναφέρονται κυρίως για το Κεφάλαιο 3.

MainMenu.cs :

```
using UnityEngine;

/**
 * Main menu script
 * Attached on the main menu canvas
 */
public class MainMenu : MonoBehaviour {

    /**
     * Used for the Quit Button to quit the Application
     */
    public void QuitApp()
    {
        Debug.Log("Quitting Application...");
        Application.Quit();
    }
}
```

OptionsMenu.cs :

```
using UnityEngine;
using UnityEngine.Audio;

/**
 * OptionsMenu script
 * Attached over the options menu canvas
 */
public class OptionsMenu : MonoBehaviour {

    // Additional future feature (Set volume)
    public AudioMixer audioMixer;
    public void SetVolume(float volume)
    {
        audioMixer.SetFloat("volume",volume);
    }

    /*Used to set the quality of the application's graphics*/
    public void SetQuality(int qualityIndex)
    {
        QualitySettings.SetQualityLevel(qualityIndex);
    }

    /*Used to set the window of the app to fullscreen or not*/
    public void SetFullscreen(bool isFullscreen)
    {
        Debug.Log("Fullscreen toggled!");
        Screen.fullScreen = isFullscreen;
    }
}
```

SelectionMenu.cs :

```
using UnityEngine;
using UnityEngine.SceneManagement;

/**
 * SelectionMenu script
 * Used to load all the algorithm scenes
 * Attached to Selection menu canvas
 */
public class SelectionMenu : MonoBehaviour {

    /*The following functions are used to load their corresponding algorithm*/
    public void StartSelectionSort()
    {
        SceneManager.LoadScene("SelectionSort", LoadSceneMode.Single);
    }

    // Extra algorithm
    public void StartBubbleSort()
    {
        SceneManager.LoadScene("BubbleSort", LoadSceneMode.Single);
    }

    public void StartFFT()
    {
        SceneManager.LoadScene("FFT", LoadSceneMode.Single);
    }

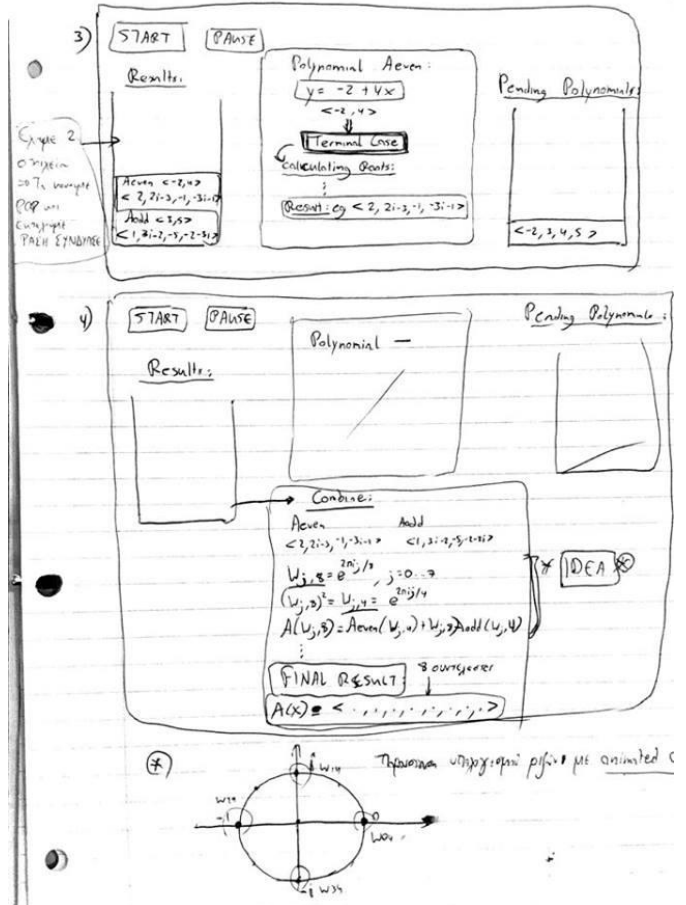
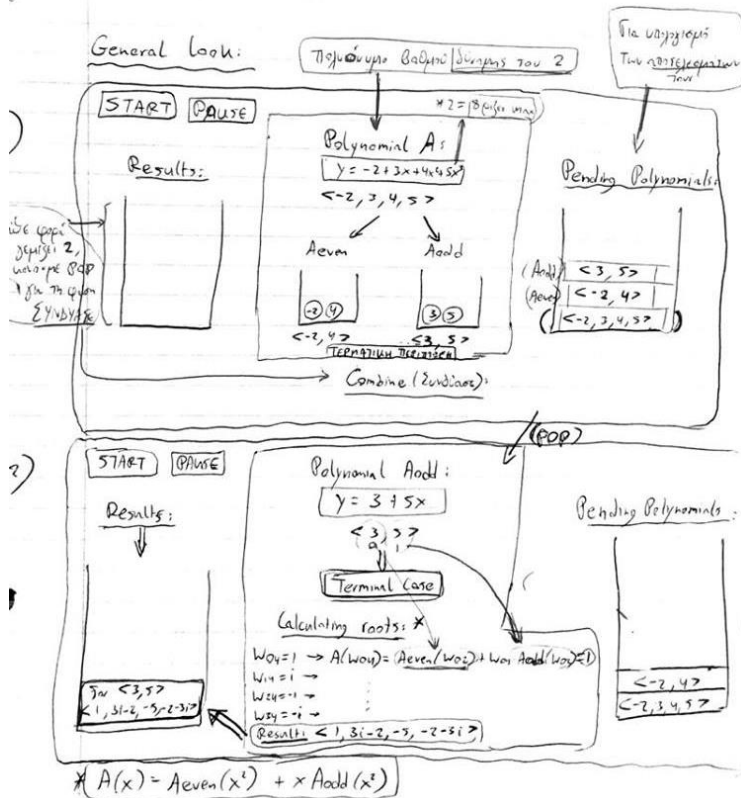
    public void StartSelectionAlgorithm()
    {
        SceneManager.LoadScene("SelectionAlgorithm", LoadSceneMode.Single);
    }

    public void StartDynamicProgramming()
    {
        SceneManager.LoadScene("DynamicProgramming", LoadSceneMode.Single);
    }

    public void StartRandomizedAlgorithm()
    {
        SceneManager.LoadScene("RandomizedAlgorithm", LoadSceneMode.Single);
    }
}
```

Παράρτημα Γ

Πιο κάτω βρίσκονται τα scripts και σχεδιαστικό υλικό που χρησιμοποιηθήκαν κατά την υλοποίηση του αλγορίθμου FFT. Αναφέρονται κυρίως για το Κεφάλαιο 4.



Complex.cs

```
using UnityEngine;
```

```
/**
 * Complex Numbers
 * (Help from Complex.java)
 * (Used in the Fast Fourier Transform algorithm)
 */
public class Complex{

    private float real;
    private float imaginary;

    public float Real    {
        get              { return real; }

        set              { real = value; }
    }
}
```

```

    public float Imaginary
    {
        get { return imaginary; }
        set { imaginary = value; }
    }

    Complex()
    {
        this.real = 0;
        this.imaginary = 0;
    }

    public Complex(float real, float imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex operator +(Complex c1, Complex c2)
    {
        return new Complex(c1.real + c2.real, c1.imaginary + c2.imaginary);
    }

    public static Complex operator -(Complex c1, Complex c2)
    {
        return new Complex(c1.real - c2.real, c1.imaginary - c2.imaginary);
    }

    public static Complex operator *(Complex c1, Complex c2)
    {
        return new Complex((c1.real * c2.real) - (c1.imaginary * c2.imaginary),
                           (c1.real * c2.imaginary) + (c1.imaginary * c2.real));
    }

    public static Complex operator *(int i, Complex c2)
    {
        return new Complex((i * c2.real) - (i * c2.imaginary),
                           (i * c2.imaginary) + (i * c2.real));
    }

    public static Complex operator /(Complex c1, Complex c2)
    {
        return c1*c2.reciprocal();
    }

    public Complex reciprocal()
    {
        float scale = real * real + imaginary * imaginary;
        return new Complex(real / scale, -imaginary / scale);
    }

    public Complex exp()
    {
        return new Complex(Mathf.Exp(real) * Mathf.Cos(imaginary), Mathf.Exp(real) *
        Mathf.Cos(imaginary));
    }

    public override string ToString()
    {
        if (imaginary == 0) return real.ToString("0.##") + "";
        if (real == 0) return imaginary.ToString("0.##") + "i";
        if (imaginary < 0) return real.ToString("0.##") + " " + imaginary.ToString("0.##") + "i";
        return real.ToString("0.##") + " + " + imaginary.ToString("0.##") + "i";
    }
}

```

StackObjectManager.cs

```

using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

/**
 * StackObjectManager
 * Used for controlling the stack 3D object for Pushing, Popping and
 * Clearing Elements.
 * */
public class StackObjectManager : MonoBehaviour {

    // The stack element to be instantiated each time in the stack
    public Transform stack_element;

    // All the elemnts currently in the stack
    private Stack<Transform> placed_elements;

    // Maximum stack elements
    private const int MAX_ELEMENTS = 8;

    // Use this for initialization
    void Start()
    {
        placed_elements = new Stack<Transform>();
        //StartCoroutine("addElements");
    }

    /**
     * pushElement function
     * Push an element into the stack with s string name and col color
     */
    public void pushElement(string s, Color col)
    {
        // Instantiate a new stack element at the top position of the stack
        if (placed_elements.Count == MAX_ELEMENTS) return;
        Transform instantiatedItem = Instantiate(stack_element);
        instantiatedItem.SetParent(gameObject.transform);
        instantiatedItem.GetComponent<Renderer>().material.color = col;
        instantiatedItem.GetChild(0).GetComponent<TextMeshPro>().text = s;
        instantiatedItem.position = new Vector3(gameObject.transform.position.x,
        gameObject.transform.position.y + 20, gameObject.transform.position.z);
        placed_elements.Push(instantiatedItem);
    }

    /**
     * popElement function
     * Pop an element from the stack and return its string value
     */
    public string popElement()
    {
        if (placed_elements.Count == 0) return "ERROR";
        Transform popped_element = placed_elements.Pop();
        string s = popped_element.GetChild(0).GetComponent<TextMeshPro>().text;
        Destroy(popped_element.gameObject);
        return s;
    }

    // Used for testing
    public IEnumerator addElements()
    {
        for (int i=0; i<8; i++)
        {
            pushElement("[ "+i+" ]", Color.red);
            yield return new WaitForSeconds(2f);
        }
        for (int i = 0; i < 8; i++)

```

```

        {
            gameObject.transform.GetChild(0).GetComponent<TextMeshPro>().text = popElement();
            yield return new WaitForSeconds(2f);
        }
        yield return null;
    }
}

```

ContainerDisplay.cs

```

using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

/**
 * Used for the Container 3D gameobject
 * This object acts like a data structure with 8 items capacity
 * */
public class ContainerDisplay : MonoBehaviour {
    // FOR scriptable object
    //public Container container;

    // The three walls of the container
    public Transform ground;
    public Transform left_wall;
    public Transform right_wall;

    // Container's label
    public TextMeshPro label;

    // The item that is going to be added each time in the container
    public Transform item;

    // Number of items to be interpolated in each row
    public int numberOfInterpolatedItems = 4;

    private Vector3 startPos;      // initial position of where the interpolation of items start
    private Vector3 endPos;        // end position of the interpolation
    private float siz;             // size value

    private List<Transform> PlacedItems;    // current placed items
    private int noOfItems = 0;             // current number of items

    private Stack<string> remainingItemsToAdd; // The remaining items left to add

    public bool add_an_item = false;    // if true then add an item
    public string item_string = "";      // current item string value

    // Use this for initialization
    void Start () {
        // FOR scriptable object
        //label.text = container.name;
        //item.gameObject.AddComponent<Rigidbody>();
        PlacedItems = new List<Transform>();
        remainingItemsToAdd = new Stack<string>();
        noOfItems = 0;
        siz = gameObject.transform.localScale.x/1.2f;
        item.transform.localScale = new Vector3(siz, siz, siz);
        startPos = gameObject.transform.localPosition;
        startPos.x += siz/3.4f;
        startPos.y += siz*1.4f;
        endPos = startPos;
        endPos.x += siz * 3.2f;
        Debug.Log("Container working");
        //StartCoroutine("Containing");
    }
}

```

```

    }

    void FixedUpdate()
    {
        // Add an item while we are able
        if (add_an_item)
        {
            addItem(item_string);
            add_an_item = false;
        }
    }

    /**
     * addItemToContainer function
     * Set the current item to add value and add it to the container
     */
    public void addItemToContainer(string s)
    {
        item_string = s;
        add_an_item = true;
    }

    /**
     * addItem function
     * add an item while we have space in the correct instantiated
     * interpolated position
     */
    public bool addItem(string s)
    {
        if (noOfItems == 8)
        {
            return false;
        }
        item.GetChild(0).GetComponent<TextMeshPro>().text = s;
        item.GetChild(0).GetComponent<TextMeshPro>().fontSize = 20;
        Transform instantiatedItem = Instantiate(
            item,
            Vector3.Lerp(startPos, endPos, (float)noOfItems %
numberOfInterpolatedItems
            / (float)(numberOfInterpolatedItems -
1)),
            Quaternion.LookRotation(Vector3.forward));
        instantiatedItem.SetParent(gameObject.transform);
        PlacedItems.Add(instantiatedItem);
        noOfItems++;
        return true;
    }

    /**
     * addItem function
     * Remove a specific item and move items to fit the right places
     * after the removal
     */
    public bool c(string s)
    {
        bool available = false;
        for(int i=0; i<noOfItems; i++)
            if(PlacedItems[i].GetChild(0).GetComponent<TextMeshPro>().text == s)
            {
                available = true;
                break;
            }
        if (available)
            StartCoroutine("moveItems", s);
        else return false;
        return true;
    }
}

```

```

/**
 * moveItems coroutine
 * Move items to fill the empty space on s item
 * and remove the s item from the list
 */
public IEnumerator moveItems(string s)
{
    Vector3 pos = PlacedItems[noOfItems-1].transform.position;

    // Remove the rigidbody component from all items
    for (int i = 0; i < noOfItems; i++)
    {
        Destroy(PlacedItems[i].gameObject.GetComponent<Rigidbody>());
    }
    for (int i = 0; i < noOfItems; i++)
    {
        if (PlacedItems[i].GetChild(0).GetComponent<TextMeshPro>().text == s)
        {
            GameObject p = PlacedItems[i].gameObject;
            pos = p.transform.position;
            PlacedItems.RemoveAt(i);
            Destroy(p);
            noOfItems--;

            // Use the ball mover component of the spheres
            // to move the spheres one step backs
            for (int j = i; j < noOfItems; j++)
            {
                if (PlacedItems[j] != null)
                {
                    Vector3 tmpPos = PlacedItems[j].transform.position;
                    PlacedItems[j].GetComponent<BallMover>().target = pos;
                    PlacedItems[j].GetComponent<BallMover>().startPos =
PlacedItems[j].transform.position;
                    PlacedItems[j].GetComponent<BallMover>().active = true;
                    yield return new WaitForSeconds(0.3f);
                    pos = tmpPos;
                }
            }
        }
    }
    yield return true;
}

/**
 * removeItem function
 * Removes an item from the container
 */
public bool removeItem()
{
    if (noOfItems == 0) return false;
    GameObject p = PlacedItems[0].gameObject;
    PlacedItems.RemoveAt(0);
    Destroy(p);
    noOfItems--;
    return true;
}

/**
 * makeEmpty
 * returns true if container can be emptied and empties it
 * else false
 */
public bool makeEmpty()
{

```

```

        if (noOfItems == 0) return false;
        for(int i=noOfItems-1; i>=0; i--)
        {
            GameObject p = PlacedItems[i].gameObject;
            PlacedItems.RemoveAt(i);
            Destroy(p);
        }
        noOfItems=0;
        return true;
    }

    /**
     * isEmpty function
     * returns true if container is empty
     */
    public bool isEmpty()
    {
        if (noOfItems == 0)
            return true;
        return false;
    }

    /**Sets the label of the container*/
    public void setLabel(string s)
    {
        label.text = s;
    }

    /**Coroutine used for testing the container*/
    public IEnumerator Containing()
    {
        for (int i = 0; i < 10; i++)
        {
            addItem(""+i);

            Debug.Log("Added item");
            yield return new WaitForSeconds(1f);
        }
        for (int i = 4; i < 6; i++)
        {
            //removeItem(0 + "");
            yield return new WaitForSeconds(2f);
        }

        // Always wait 2 seconds after remove!
        yield return null;
    }
}

```

FFTPolynomialDropdownController.cs

```

using System;
using TMPPro;
using UnityEngine;
using UnityEngine.UI;

/**
 * FFTPolynomialDropdownController script
 * Used to control the main functionalities of the
 * drop down box of FFT main UI
 */
public class FFTPolynomialDropdownController : MonoBehaviour {

    // UI elements that are being controlled
    public TextMeshProUGUI title;
}

```

```

public GameObject randomDropbox;
public GameObject customInputField;
public GameObject FFTAlgorithm;
public TextMeshProUGUI polynomialText;

// Use this for initialization
void Start()
{
    customInputField.GetComponent<TMP_InputField>().onValueChanged.AddListener(delegate {
ValueChangeCheck(); });
}

// Update is called once per frame
void Update()
{
    if (GetComponent<Dropdown>().value == 0)
    {
        title.text = "Please choose a degree for a random polynomial:";
        randomDropbox.SetActive(true);
        customInputField.SetActive(false);
    }
    else if (GetComponent<Dropdown>().value == 1)
    {
        title.text = "Please enter your own polynomial with 2, 4, 8 or 16 coefficients
(seperated by \" \"):";
        randomDropbox.SetActive(false);
        customInputField.SetActive(true);
    }
}

/**
 * ValueChangeCheck
 * Called each time the user changes the value of the
 * input text box. Used to display the final appearance
 * of the polynomial.
 */
public void ValueChangeCheck()
{
    string[] splittedInput = customInputField.GetComponent<TMP_InputField>().text.Split(' ');
    // Check the given polynomial if it is a valid size or
    // use the default one (-2+3x)
    int size = splittedInput.Length;
    if (splittedInput.Length > 16)
    {
        size = 16;
    }
    else if (size > 8)
    {
        size = 8;
    }
    else if (size > 4)
    {
        size = 4;
    }
    else if (size > 2)
    {
        size = 2;
    }
    else
    {
        splittedInput = new string[] { "-2", "3" };
        size = 2;
    }

    int[] coeff = new int[size];
    Complex[] polComplex = new Complex[size];
    string pol = "P: ";

```

```

        for (int i = 0; i < size; i++)
        {
            int num = 0;
            if (Int32.TryParse(splittedInput[i], out num))
            {
                Console.WriteLine(num);
            }
            else
            {
                Console.WriteLine("String could not be parsed. ERROR!");
            }
            coeff[i] = num;
            polComplex[i] = new Complex(num, 0);
        }
        FFTAlgorithm.GetComponent<FastFourierTransformAlgorithm>().P = polComplex;
        pol =
        FFTAlgorithm.GetComponent<FastFourierTransformAlgorithm>().calculatePolynomialText(polComplex);
        polynomialText.text = pol;
    }
}

```

FastFourierTransformAlgorithm.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using Random = System.Random;

/**
 * FastFourierTransformAlgorithm script
 *
 * (Help from FFT.java)
 */
public class FastFourierTransformAlgorithm : MonoBehaviour {

    // The polynomial used:
    public Complex[] P = { new Complex(-2, 0), new Complex(3, 0) };
    public static Complex[] inputFieldP;

    // Used to set the speed of the algorithm
    public float speed = 2f;

    // If start button is pressed then it turns true
    private bool start_btn;

    // Main objects used to be controlled during the
    // execution of the algorithm
    public Transform container;
    public Transform polynomial;
    public Transform coefficients;
    public Transform results_panel;
    public Transform polynomials_3dstack;
    public Transform results_3dstack;
    public Transform even_container;
    public Transform odd_container;
    public Transform left_arrow;
    public Transform right_arrow;
    public Transform current_step;
    public Transform final_result;

    // from the Choice UI

```

```

public Dropdown degree_dropdown;
public TextMeshProUGUI polynomial_choice;
public Transform fireworks;

// Used for the step next button
public bool useStepByStep;
private bool stepForward;

// All the text Ui elements used to show every
// calculation and result during the execution
private TextMeshPro coefficients_text;
private TextMeshPro peven_text;
private TextMeshPro podd_text;
private TextMeshPro result_calculations_text;
private TextMeshPro w_calculations_text;
private TextMeshPro result_text;
private TextMeshProUGUI current_step_text;

/*Functions enabled by the UI buttons for controlling the flow of the program*/
public void setSpeed(float s)
{
    this.speed = s;
}
public void setStart(bool s)
{
    this.start_btn = s;
}
public void SetStepForward(bool b)
{
    stepForward = b;
}
public void SetStepByStep(bool b)
{
    useStepByStep = b;
}
public void BackButton()
{
    SceneManager.LoadScene("MainUI");
}
public void dropdownPolynomial()
{
    // Based on the user's choice, we set a polynomial degree
    // value and calculate a random polynomial
    int choice = degree_dropdown.value;
    switch (choice)
    {
        case 0:
            this.P = calculateRandomPolynomial(2);
            break;

        case 1:
            this.P = calculateRandomPolynomial(4);
            break;

        case 2:
            this.P = calculateRandomPolynomial(8);
            break;

        case 3:
            this.P = calculateRandomPolynomial(16);
            break;
    }
    polynomial_choice.text = calculatePolynomialText(this.P);
}
/*Control flow functions END*/

/**

```

```

* complexArrayToString
* Based on a complex array, it returns a string
* displaying the analogous coefficient array
**/
public String complexArrayToString(Complex[] P)
{
    String s = "< ";
    for (int i = 0; i < P.Length-1; i++)
    {
        s += P[i] + ", ";
    }
    s += P[P.Length - 1] + ">";
    return s;
}

/**
* calculatePolynomialText function
* Using a complex number array P, it displays it
* in string form "Y = r0 + r1^1 + r2^2... rn^n" using
* the real part of the complex numbers.
**/
public String calculatePolynomialText(Complex[] p)
{
    string s = "Y = ";
    for(int i=0; i<p.Length; i++)
    {
        if (i == 0)
            s += p[i].Real + " ";
        else
        {
            if (p[i].Real >= 0)
                s += "+ ";
            s += p[i].Real + "x^" + i + " ";
        }
    }
    return s;
}

/**
* calculateRandomPolynomial function
* Based on a given degree, it calculates random coefficients
* for the polynomial with that degree. If we want to display
* a polynomial, we just see the values of the real part of a
* Complex number and 0 to the imaginary.
**/
private Complex[] calculateRandomPolynomial(int degree)
{
    Complex[] p = new Complex[degree];
    Random r = new Random();
    for (int i = 0; i < degree; i++)
    {
        // Include 0 except the final value of the polynomial
        if(i==degree-1)
            p[i] = new Complex(r.Next(1,10), 0f);
        else
            p[i] = new Complex(r.Next(-9,10), 0f);
    }
    return p;
}

// Clear and initialize all UI and Object elements
private void Start()
{
    if (inputFieldP != null)
    {
        this.P = inputFieldP;
    }
}

```

```

// Clear and initialize text boxes:
setCurrPol("");
peven_text = results_panel.GetChild(0).GetComponent<TextMeshPro>();
podd_text = results_panel.GetChild(1).GetComponent<TextMeshPro>();
w_calculations_text = results_panel.GetChild(2).GetComponent<TextMeshPro>();
result_calculations_text = results_panel.GetChild(3).GetComponent<TextMeshPro>();
result_text = results_panel.GetChild(4).GetComponent<TextMeshPro>();
current_step_text = current_step.GetComponent<TextMeshProUGUI>();
coefficients_text = coefficients.GetChild(0).GetComponent<TextMeshPro>();

peven_text.text = "";
podd_text.text = "";
result_calculations_text.text = "";
w_calculations_text.text = "";
result_text.text = "";
current_step_text.text = "";
coefficients_text.text = "";
polynomial_choice.text = calculatePolynomialText(this.P);
// NORMAL ITERATIVE FFT RUN
//Complex[] result = IterativeFFT(P);
//Debug.Log("RESULT:\t"+complexArrayToString(result));
// RUN THE ITERATIVE VISUAL FFT :
// MAX: 16th degree polynomial
//Complex[] P = { new Complex(3, 0), new Complex(5, 0), new Complex(2, 0), new Complex(1,
0), new Complex(4, 0), new Complex(5, 0), new Complex(1, 0), new Complex(-1, 0),
//
new Complex(3, 0), new Complex(5, 0), new Complex(2, 0), new Complex(1,
0), new Complex(4, 0), new Complex(5, 0), new Complex(1, 0), new Complex(-1, 0) };
// 8th degree polynomial
//Complex[] P = { new Complex(3, 0), new Complex(5, 0), new Complex(2, 0), new Complex(1,
0), new Complex(4, 0), new Complex(5, 0), new Complex(1, 0), new Complex(-1, 0) };
// 2nd degree polynomial
//Complex[] P = { new Complex(-2, 0), new Complex(3, 0) };

// Begin the algorithm
StartCoroutine("iterativeVisualFFT");
}

/**
 * normalFFT function
 * A normal recursive version of FFT
 * Used for testing.
 */
public Complex[] normalFFT(Complex[] P, int n)
{
    if (n == 1) return new Complex[] { P[0] };
    if (n % 2 != 0) Debug.Log("Error! Length of the array is not in the power of two!");

    Complex[] Peven = new Complex[n / 2];
    Complex[] Podd = new Complex[n / 2];
    int even = 0, odd = 0;

    for (int i = 0; i < n; i++)
    {
        if (i % 2 == 0)
        {
            Peven[even] = P[i];
            even++;
        }
        else
        {
            Podd[odd] = P[i];
            odd++;
        }
    }
}

```

```

Complex[] p1 = normalFFT(Peven, n / 2);
Complex[] p2 = normalFFT(Podd, n / 2);
/*for(int j=0; j<2*n; j++)
{
    // W2πij/n
    float curr = ((2f * Mathf.PI * (float)j) / (float)n);
    Complex W = new Complex(Mathf.Cos(curr), Mathf.Sin(curr));
    Pnew[j] = p1[j % n] + (W * p2[j % n]);
}*/

// Combine even + odd
Complex[] Pnew = new Complex[n];
for (int k = 0; k < n / 2; k++)
{
    float kth = 2 * k * Mathf.PI / n;
    Complex wk = new Complex(Mathf.Cos(kth), Mathf.Sin(kth));
    Pnew[k] = p1[k] + (wk * (p2[k]));
    Pnew[k + n / 2] = p1[k] - (wk * (p2[k]));
}
return Pnew;
}

/**
 * IterativeFFT function
 * Used for testin the results
 */
public Complex[] IterativeFFT(Complex[] P)
{
    Stack<Complex[]> polynomials = new Stack<Complex[]>();
    Stack<Complex[]> results = new Stack<Complex[]>();

    // Insert initial array into stack
    polynomials.Push(P);

    while(polynomials.Count != 0)
    {
        // Pop the next polynomial
        Complex[] currentP = polynomials.Pop();

        // terminal option
        if(currentP.Length == 1)
        {
            // return {P[0],P[0]}
            results.Push(new Complex[] {currentP[0],currentP[0]});

            while (results.Count > 1)
            {
                // Even
                Complex[] r1 = results.Pop();

                // if the next element in the result stack is not
                // the same size as the one popped
                // (Peven.Length == Podd.Length) then wait until
                // its partner gets pushed in the results
                if(results.Peek().Length != r1.Length)
                {
                    results.Push(r1);
                    break;
                }

                // Odd
                Complex[] r2 = results.Pop();

                /*String s1 = "", s2 = "";
                for (int i = 0; i < r1.Length; i++)
                {

```

```

        s1 += r1[i] + ", ";
        s2 += r2[i] + ", ";
    }*/
    Debug.Log(complexArrayToString(r1));
    Debug.Log(complexArrayToString(r2));

    // Calculate the new array for the 2*N roots
    // using the popped results of the previous
    // fft(Peven) and fft(Podd)
    int newN = r1.Length * 2;
    Complex[] newP = new Complex[newN];
    for (int i = 0; i < newN / 2; i++)
    {
        float kth = 2 * i * Mathf.PI / newN;
        Complex wk = new Complex(Mathf.Cos(kth), Mathf.Sin(kth));
        newP[i] = r1[i] + (wk * (r2[i]));
        newP[i + newN / 2] = r1[i] - (wk * (r2[i]));
    }
    results.Push(newP);
}
continue;
}

// Split current array into its Even and Odd positioned numbers
Complex[] Peven = new Complex[currentP.Length / 2];
Complex[] Podd = new Complex[currentP.Length / 2];
int even = 0, odd = 0;
for(int i=0; i<currentP.Length; i++)
{
    if (i % 2 == 0)
        Peven[even++] = currentP[i];
    else
        Podd[odd++] = currentP[i];
}

// Add Podd and Peven to stack in this order (since stack is FIFO)
polynomials.Push(Podd);
polynomials.Push(Peven);
}

// The final result in the results stack is
// the result of the FFT
return results.Pop();
}

/*****
*****

/*Used to push a string value of s with color col to the stack st*/
private void pushTo3DStack(Transform st, string s, Color col)
{
    st.GetComponent<StackObjectManager>().pushElement(s, col);
}
/*Pops and returns a stack element from stack st.*/
private string popFrom3DStack(Transform st)
{
    return st.GetComponent<StackObjectManager>().popElement();
}
/*Sets the current polynomial value text box to s*/
private void setCurrPol(string s)
{
    polynomial.GetChild(0).GetComponent<TextMeshPro>().text = s;
}
/*Changes the color of a specific character to color on a string t at position pos*/
private string changeColorChar(string t, int pos, string color)
{

```

```

        string text = t;
        string sub1 = text.Substring(0,pos);
        string sub2 = text.Substring(pos+1, text.Length-sub1.Length-1);
        // We use tags
        text = sub1 + "<color=" + color + ">" + text.ToCharArray()[pos] + "</color>" + sub2;
        return text;
    }

    /**
     * iterativeVisualFFT function
     * Our main iterative visual FFT algorithm
     */
    public IEnumerator iterativeVisualFFT()
    {
        // Wait until the start button gets pressed
        yield return new WaitUntil(() => start_btn);

        setCurrPol(calculatePolynomialText(P));

        // Polynomials stack used to store the pending polynomials to be used
        // to be splitted into its even and odd polynomials
        Stack<Complex[]> polynomials = new Stack<Complex[]>();
        Stack<Complex[]> results = new Stack<Complex[]>();

        // Current level of deapth of the recursive split (tree)
        // eg: Peven1, Podd1 : peven and podd of level 1
        int level = 1;

        string curr_pol_name = "";

        // The final result in the results stack is
        // the result of the FFT
        Complex[] result = null;

        current_step_text.text = "1) Pushing initial polynomial to Polynomial stack";

        // Insert initial array into stack
        polynomials.Push(P);
        pushTo3DStack(polynomials_3dstack, "Pinit: " + complexArrayToString(P), Color.red);

        // If we are using step-by-step, we wait until the step forward button gets pressed
        // or if the Use step forward toggle gets disabled. Else we use the default
        // wait for the desired run speed.
        if (useStepByStep)
        {
            yield return new WaitUntil(() => stepForward || !useStepByStep);
            stepForward = false;
        }
        else
            yield return new WaitForSeconds(speed);

        while (polynomials.Count != 0)
        {
            // Pop the next polynomial
            Complex[] currentP = polynomials.Pop();

            // Get the name of the polynomial popped
            curr_pol_name = popFrom3DStack(polynomials_3dstack);
            curr_pol_name = curr_pol_name.Substring(0, curr_pol_name.IndexOf(':'));

            current_step_text.text = "2) Popping " + curr_pol_name + " from stack";

            setCurrPol(calculatePolynomialText(currentP));
            coefficients_text.text = curr_pol_name + ": " + complexArrayToString(currentP);

            if (useStepByStep)
            {

```

```

        yield return new WaitUntil(() => stepForward || !useStepByStep);
        stepForward = false;
    }
    else
        yield return new WaitForSeconds(speed);

    // Base Case
    if (currentP.Length == 1)
    {
        current_step_text.text = "3) Base case reached (size = 1)\n -> Push to Results";
        // return {P[0],P[0]}
        Complex[] Pfinal = new Complex[] { currentP[0], currentP[0] };
        results.Push(Pfinal);
        pushTo3DStack(results_3dstack, curr_pol_name + " BASE: " +
complexArrayToString(Pfinal), Color.magenta);
        w_calculations_text.text = complexArrayToString(currentP)+ " => " +
complexArrayToString(Pfinal);

        if (useStepByStep)
        {
            yield return new WaitUntil(() => stepForward || !useStepByStep);
            stepForward = false;
        }
        else
            yield return new WaitForSeconds(speed);

        // Clear results panel.
        clearResults();

        current_step_text.text = "4) Checking the Results stack for a Pair Match...";
        while (results.Count > 1)
        {
            // Even
            Complex[] r1 = results.Pop();

            // if the next element in the result stack is not
            // the same size as the one popped
            // (Peven.Length == Podd.Length) then wait until
            // its partner gets pushed in the results
            if (results.Peek().Length != r1.Length)
            {
                current_step_text.text = "4) Result match NOT found!";

                results.Push(r1);

                if (useStepByStep)
                {
                    yield return new WaitUntil(() => stepForward || !useStepByStep);
                    stepForward = false;
                }
                else
                    yield return new WaitForSeconds(speed);

                break;
            }

            // Odd
            Complex[] r2 = results.Pop();

            // Pop the 3D stacks too
            current_step_text.text = "5) Result Match found with size = "+r1.Length+"\n ->
Pop from stack";
            level--;
            string s1 = popFrom3DStack(results_3dstack);
            peven_text.text = "Peven: " + complexArrayToString(r1);
            yield return new WaitForSeconds(speed);
            string s2 = popFrom3DStack(results_3dstack);

```

```

podd_text.text = "Podd: " + complexArrayToString(r2);
yield return new WaitForSeconds(speed);

if (useStepByStep)
{
    yield return new WaitUntil(() => stepForward || !useStepByStep);
    stepForward = false;
}

Debug.Log(complexArrayToString(r1));
Debug.Log(complexArrayToString(r2));

// Calculate the new array for the 2*N roots
// using the popped results of the previous
// fft(Peven) and fft(Podd)
int newN = r1.Length * 2;
Complex[] newP = new Complex[newN];

// Use an array to store the results of the calculations
// to display them in order later.
// List1: for  $W_{i,2n}$ , List2: for  $W_{i+n,2n}$ 
Complex[] wi2ns = new Complex[newN];

for (int i = 0; i < newN / 2; i++)
{
    float kth = 2 * i * Mathf.PI / newN;
    Complex wk = new Complex(Mathf.Cos(kth), Mathf.Sin(kth));

    // Store current roots  $W_{i,2n}$  and  $W_{i+n,2n}$ 
    wi2ns[i] = wk;
    wi2ns[i + newN / 2] = wk - (wk + wk);

    newP[i] = r1[i] + (wk * (r2[i]));
    newP[i + newN / 2] = r1[i] - (wk * (r2[i]));
}

current_step_text.text = "6) Calculating the roots along with the results";

if (useStepByStep)
{
    yield return new WaitUntil(() => stepForward || !useStepByStep);
    stepForward = false;
}
else
    yield return new WaitForSeconds(speed);

// Display results
for (int i=0; i<newN; i++)
{
    string keth = "2 * " + i + "PI / " + newN;
    current_step_text.text = "6)a)  $W_{i+1,2n} = \cos(keth) + i\sin(keth)$ ";
    w_calculations_text.text += "W" + i + ", " + newN + " = " + wi2ns[i] + "\n";
    yield return new WaitForSeconds(speed);
    current_step_text.text = "6)b)  $P_{i+1} = P_{even[i+1]} + ((i\%2==0)? "+" : "-") + " W_{i+1,2n} * P_{odd[i+1]}$ ";
    result_calculations_text.text += "P[" + i + "] = " + newP[i] + "\n";
    yield return new WaitForSeconds(speed);

    if (useStepByStep)
    {
        yield return new WaitUntil(() => stepForward || !useStepByStep);
        stepForward = false;
    }
}

```

```

        result_text.text = "RESULT: " + complexArrayToString(newP);

        if (useStepByStep)
        {
            yield return new WaitUntil(() => stepForward || !useStepByStep);
            stepForward = false;
        }
        else
            yield return new WaitForSeconds(speed);

        w_calculations_text.text = "";
        result_calculations_text.text = "";

        // If polynomials is empty then we reached
        // the FINAL RESULT
        if (polynomials.Count == 0 && results.Count == 0) {
            result = newP;
            break;
        }

        results.Push(newP);
        pushTo3DStack(results_3dstack, "Presult" + level + ": " +
complexArrayToString(newP), Color.red);

        if (useStepByStep)
        {
            yield return new WaitUntil(() => stepForward || !useStepByStep);
            stepForward = false;
        }
        else
            yield return new WaitForSeconds(speed);
    }
    continue;
}

// Clear results panel
clearResults();

current_step_text.text = "3) Splitting "+curr_pol_name+" to even and odd";

// Split current array into its Even and Odd positioned numbers
Complex[] Peven = new Complex[currentP.Length / 2];
Complex[] Podd = new Complex[currentP.Length / 2];

// Activate containers
even_container.gameObject.SetActive(true);
odd_container.gameObject.SetActive(true);

int even = 0, odd = 0;
string curr_coeff = coefficients_text.text;
int char_pos = curr_coeff.IndexOf(',')-1;
for (int i = 0; i < currentP.Length; i++)
{
    // Highlight current letter
    coefficients_text.text = changeColorChar(curr_coeff, char_pos, "red");
    if (i % 2 == 0)
    {
        // Add even positioned numbers to EVEN container
        Peven[even++] = currentP[i];
        left_arrow.gameObject.SetActive(true);
        yield return new WaitForSeconds(speed);
        even_container.GetComponent<ContainerDisplay>().addItem(currentP[i]+"");
        left_arrow.gameObject.SetActive(false);
    }
    else

```

```

        {
            // Add odd positioned numbers to ODD container
            Podd[odd++] = currentP[i];
            right_arrow.gameObject.SetActive(true);
            yield return new WaitForSeconds(speed);
            odd_container.GetComponent<ContainerDisplay>().addItem(currentP[i] + "");
            right_arrow.gameObject.SetActive(false);
        }
        // Change letter back to black
        coefficients_text.text = changeColorChar(curr_coeff, char_pos, "black");

        // Get the position of the character before the ','
        if(i >= currentP.Length-2)
            char_pos = curr_coeff.IndexOf('>') - 2;
        else
            char_pos += curr_coeff.Substring(char_pos + 2, curr_coeff.Length - char_pos -
3).IndexOf(',') + 1;

        if (useStepByStep)
        {
            yield return new WaitUntil(() => stepForward || !useStepByStep);
            stepForward = false;
        }
    }

    string podd_coeff = "Podd" + level + ": " + complexArrayToString(Podd);
    string peven_coeff = "Peven" + level + ": " + complexArrayToString(Peven);

    podd_text.text = podd_coeff;
    peven_text.text = peven_coeff;

    // Add Podd and Peven to stack in this order (since stack is FIFO)
    polynomials.Push(Podd);
    pushTo3DStack(polynomials_3dstack, podd_coeff, Color.green);

    if (useStepByStep)
    {
        yield return new WaitUntil(() => stepForward || !useStepByStep);
        stepForward = false;
    }
    else
        yield return new WaitForSeconds(speed);

    polynomials.Push(Peven);
    pushTo3DStack(polynomials_3dstack, peven_coeff, Color.blue);

    if (useStepByStep)
    {
        yield return new WaitUntil(() => stepForward || !useStepByStep);
        stepForward = false;
    }
    else
        yield return new WaitForSeconds(speed);

    level++; // INCREMENT DEPTH

    // Clear results panel
    clearResults();

}

// Clear results panel
clearResults();

// Display the final result
final_result.gameObject.SetActive(true);
final_result.GetChild(0).GetComponent<TextMeshPro>().text = "FINAL RESULT: \n" +
complexArrayToString(result);

```

```

        fireworks.gameObject.SetActive(true);
        //w_calculations_text.text = "<color=green> RESULT: " + complexArrayToString(result) +
"</color>";

        Debug.Log("RESULT: " + complexArrayToString(result));
        yield return null;
    }

    /**
     * clearResults function
     * Clear all the text labels containing the calculations and results
     * and initialize the containers
     */
    private void clearResults()
    {
        // Clear text boxes of results panel
        podd_text.text = "";
        peven_text.text = "";
        result_calculations_text.text = "";
        w_calculations_text.text = "";
        result_text.text = "";

        // Deactivate and clear containers and text boxes
        even_container.GetComponent<ContainerDisplay>().makeEmpty();
        odd_container.GetComponent<ContainerDisplay>().makeEmpty();
        even_container.gameObject.SetActive(false);
        odd_container.gameObject.SetActive(false);
    }
}

```

Παράρτημα Δ

Πιο κάτω βρίσκονται τα scripts και σχεδιαστικό υλικό που χρησιμοποιήθηκαν κατά την υλοποίηση του αλγορίθμου Επιλογής. Αναφέρονται κυρίως για το Κεφάλαιο 5.

Array3DController.cs

```
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using Random = System.Random;

/**
 * Array3DController script
 * Attatched onto a 3DArray object
 */
public class Array3DController : MonoBehaviour {

    // Left and right walls of the array
    public Transform left_wall;
    public Transform right_wall;

    public Transform placeable_object;    // The object we want to interpolate inside the array
    public List<Transform> object_array;    // All the objects placed inside the array

    // Start and end placeholder for the object placement
    private Vector3 start_pos;
    private Vector3 end_pos;

    // Local scale to scale the array depending on the number of objects (siz)
    private Vector3 currLocalScale;
    private float siz;

    // Use this for initialization
    void Start () {
        currLocalScale = transform.localScale;
        // For testing purposes
        /*
        List<int> numArray = new List<int>();
        Random r = new Random(DateTime.Now.Ticks.GetHashCode());
        while (numArray.Count != 5)
        {
            int num = r.Next(0, 100);
            if (!numArray.Contains(num))
                numArray.Add(num);
        }
        numArray.Sort();
        spawnSpheres(numArray);*/
    }

    /**
     * setColor function
     * given an index i of the spawned objects,
     * it's color is set to c
     */
    public bool setColor(int i, Color c)
    {
        if (i > object_array.Count - 1)
            return false;
        object_array[i].GetComponent<Renderer>().material.color = c;
        return true;
    }
}
```

```

/**
 * clearArray function
 * Used to clear the array and remove all the spheres
 * from the game view.
 */
public void clearArray()
{
    for (int i = 0; i < object_array.Count; i++)
    {
        Destroy(object_array[i].gameObject);
    }
    object_array.Clear();
}

/**
 * spawnSpheres
 * Given an array with integers, it spawns them into spheres
 * into an auto adjusted array
 */
public void spawnSpheres(List<int> numArray)
{
    int number_of_objects = numArray.Count;

    // resize the size of the array based on the number of objects we place
    if(number_of_objects <= 10)
        transform.localScale = new Vector3(currLocalScale.x, currLocalScale.y *
(number_of_objects+5), currLocalScale.z);
    else
        transform.localScale = new Vector3(currLocalScale.x, currLocalScale.y *
number_of_objects, currLocalScale.z);

    // Resize the sphere so that it fits the inside of the array based on the size of the
    // array box.
    siz = currLocalScale.x * 2.4f;
    placeable_object.transform.localScale = new Vector3(siz, siz, siz);

    // Set the start and end position of where the balls will spawn a few points after and
before
    // the left and right wall consecutively of the array box
    float padding = siz; //left_wall.position.y / 20;
    start_pos = new Vector3(left_wall.position.x + padding, left_wall.position.y,
left_wall.position.z);
    end_pos = new Vector3(right_wall.position.x - padding, right_wall.position.y,
right_wall.position.z);

    if (numArray.Count == 1)
    {
        // Set the text inside the ball.
        placeable_object.GetChild(0).GetComponent<TextMeshPro>().text = numArray[0] + "";

        Transform instantiatedObject =
            Instantiate(
                placeable_object,
                new Vector3((start_pos.x + end_pos.x) / 2, start_pos.y, start_pos.z),
                Quaternion.LookRotation(Vector3.forward));

        object_array.Add(instantiatedObject);
    }
    else
    {
        for (int i = 0; i < number_of_objects; i++)
        {
            // Set the scale of the sphere
            //float siz = (float)(22.0 / (float)number_of_objects);
            //placeable_object.transform.localScale = new Vector3(siz, siz, siz);

```

```

        // Set the text inside the ball.
        placeable_object.GetChild(0).GetComponent<TextMeshPro>().text = numArray[i] + "";

        Transform instantiatedObject =
            Instantiate(
                placeable_object,
                Vector3.Lerp(start_pos, end_pos, (float)i / (float)(number_of_objects - 1)),
                Quaternion.LookRotation(Vector3.forward));

        object_array.Add(instantiatedObject);
    }
}
}
}

```

FixedArray3DController.cs

```

using System.Collections.Generic;
using TMPro;
using UnityEngine;
using Random = System.Random;

/**
 * FixedArray3DController script
 * Attached onto the 5 container prefab
 */
public class FixedArray3DController : MonoBehaviour {

    public Transform left_wall;
    public Transform right_wall;

    public Transform placeable_object;
    public List<Transform> object_array;
    public List<int> number_array;

    private Vector3 start_pos;
    private Vector3 end_pos;
    private float siz;

    // Use this for initialization
    void Start () {
        /*
        List<int> numArray = new List<int>();
        Random r = new Random(DateTime.Now.Ticks.GetHashCode());
        while (numArray.Count != 5)
        {
            int num = r.Next(0, 100);
            if (!numArray.Contains(num))
                numArray.Add(num);
        }
        numArray.Sort();
        spawnSpheres(numArray);
        */
    }

    public void spawnSpheres(List<int> numArray)
    {
        number_array = numArray;
        int number_of_objects = numArray.Count;
        //Vector3 currLocalScale = transform.localScale;
        //transform.localScale = new Vector3(currLocalScale.x, currLocalScale.y * numArray.Count,
        currLocalScale.z);

        // Resize the sphere so that it fits the inside of the array based on the size of the
        // array box.
    }
}

```

```

        siz = gameObject.transform.localScale.x / 2f ;
        placeable_object.transform.localScale = new Vector3(siz, siz, siz);

        // Set the start and end position of where the balls will spawn a few points after and
before
        // the left and right wall consecutively of the array box
        float padding = siz/1.5f; //left_wall.position.y / 20;
        start_pos = new Vector3(left_wall.position.x + padding, left_wall.position.y,
left_wall.position.z);
        end_pos = new Vector3(right_wall.position.x - padding, right_wall.position.y,
right_wall.position.z);

        if (number_of_objects == 1)
        {
            // Set the text inside the ball.
            placeable_object.GetChild(0).GetComponent<TextMeshPro>().text = numArray[0] + "";

            Transform instantiatedObject =
                Instantiate(
                    placeable_object,
                    new Vector3((start_pos.x + end_pos.x) / 2, start_pos.y, start_pos.z),
                    Quaternion.LookRotation(Vector3.forward));

            object_array.Add(instantiatedObject);
        }
        else
        {
            for (int i = 0; i < number_of_objects; i++)
            {
                // Set the scale of the sphere
                //float siz = (float)(22.0 / (float)number_of_objects);
                //placeable_object.transform.localScale = new Vector3(siz, siz, siz);

                // Set the text inside the ball.
                placeable_object.GetChild(0).GetComponent<TextMeshPro>().text = numArray[i] + "";

                Transform instantiatedObject =
                    Instantiate(
                        placeable_object,
                        Vector3.Lerp(start_pos, end_pos, (float)i / (float)(number_of_objects - 1)),
                        Quaternion.LookRotation(Vector3.forward));

                object_array.Add(instantiatedObject);
            }
        }
    }

    /**
     * sortArray function
     * Used to sort the current array visually
     */
    public void sortArray()
    {
        number_array.Sort();

        // Destroy all previous objects in the array
        for(int i= object_array.Count-1; i>=0 ; i--)
        {
            Destroy(object_array[i].gameObject);
        }
        object_array.Clear();

        if (number_array.Count == 1)
        {
            // Set the text inside the ball.
            placeable_object.GetChild(0).GetComponent<TextMeshPro>().text = number_array[0] + "";

```

```

        Transform instantiatedObject =
            Instantiate(
                placeable_object,
                new Vector3((start_pos.x + end_pos.x) / 2, start_pos.y, start_pos.z),
                Quaternion.LookRotation(Vector3.forward));

        object_array.Add(instantiatedObject);
    }
    else
    {
        // Add the new sorted list in the array
        for (int i = 0; i < number_array.Count; i++)
        {
            // Set the text inside the ball.
            placeable_object.GetChild(0).GetComponent<TextMeshPro>().text = number_array[i] +
"";

            Transform instantiatedObject =
                Instantiate(
                    placeable_object,
                    Vector3.Lerp(start_pos, end_pos, (float)i / (float)(number_array.Count -
1)),
                    Quaternion.LookRotation(Vector3.forward));

            object_array.Add(instantiatedObject);
        }
    }
}

/**
 * clearArray function
 * Used to clear the array and remove all the spheres
 * from the game view.
 */
public void clearArray()
{
    for(int i=0; i< object_array.Count; i++)
    {
        Destroy(object_array[i].gameObject);
    }
    object_array.Clear();
}
}

```

DropDownArrayChoice.cs

```

using System;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

/**
 * DropdownArrayChoice
 * Used in the selection algorithm to select a way
 * run the algorithm (randomly generated array or given by the user)
 */
public class DropdownArrayChoice : MonoBehaviour {

    // The label containing the current description
    // of the choice of the user
    public TextMeshProUGUI title;

    // UI elements being controlled

```

```

public TextMeshProUGUI ArraySizeText;
public TextMeshProUGUI ArrayText;
public GameObject Slider;
public GameObject ArrayInputField;
public Slider IndexSlider;

// The selection algorithm object containing the main scrip
// (used to change the main variables)
public GameObject SelectionAlgorithm;

/* GameObjects to enable/disable after the dropdown changes */

// Use this for initialization
void Start () {
    ArrayInputField.GetComponent<TMP_InputField>().onValueChanged.AddListener(delegate {
ValueChangeCheck(); });
}

// Update is called once per frame
void Update () {
    if (GetComponent<Dropdown>().value == 0)
    {
        title.text = "Please select a size for a random array (with random numbers between 0-
99):";
        Slider.SetActive(true);
        //ArraySizeText.SetActive(true);
        ArrayInputField.SetActive(false);
    }
    else if (GetComponent<Dropdown>().value == 1)
    {
        title.text = "Please enter your own array with size ranging from 2 to 50 (each value
must be different between 0-99 and seperated by space \" \"):";
        Slider.SetActive(false);
        //ArraySizeText.SetActive(false);
        ArrayInputField.SetActive(true);
    }
}

/**
 * ValueChangeCheck
 * Called when the value of the input field is changed
 * (provides a dynamically changing polynomial in the text box
 * based on the user's choice)
 */
public void ValueChangeCheck()
{
    // Parse the integer array from string
    string[] splittedInput = ArrayInputField.GetComponent<TMP_InputField>().text.Split(' ');

    string array_string = "Array: < ";
    List<int> array = new List<int>();
    bool error_flag = false;
    for (int i = 0; i < splittedInput.Length; i++)
    {
        int num = 0;
        if (Int32.TryParse(splittedInput[i], out num))
        {
            // Check if each number is within bounds, correct length and
            // each element is different.
            if(num<0 || num > 99 || splittedInput.Length>50 || splittedInput.Length < 2 ||
array.Contains(num))
            {
                error_flag = true;
            }
            else
            {

```

```

        array.Add(num);
        if (i < splittedInput.Length - 1)
            array_string += num + ", ";
        else
            array_string += num;
    }
    Console.WriteLine(num);
}
else
{
    Console.WriteLine("String could not be parsed. ERROR!");
}
}

// If error then use the default array
if (error_flag)
{
    ArraySizeText.text = "";
    ArrayText.text = "ERROR! PLEASE ENTER A VALID ARRAY!";
    // Default array
    SelectionAlgorithm.GetComponent<SelectionAlgorithm>().initial_array = new List<int> { 1,
2, 3, 4, 5};
}
else
{
    ArraySizeText.text = "Array size: " + splittedInput.Length;
    ArrayText.text = array_string + ">";
    SelectionAlgorithm.GetComponent<SelectionAlgorithm>().initial_array = array;

    // Change the maximum value of the index slider
    IndexSlider.maxValue = splittedInput.Length;
}
}
}

```

SliderArrayText.cs

```

using System;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using Random = System.Random;

/**
 * SliderArrayText script
 * Used to control the functionality of the slider
 * used to enter a random array (Selection Algorithm)
 */
public class SliderArrayText : MonoBehaviour {

    // UI elements being controlled
    public TextMeshProUGUI ArraySizeText;
    public TextMeshProUGUI ArrayText;
    public GameObject SelectionAlgorithm;
    public Slider IndexSlider;

    void Start()
    {
        GetComponent<Slider>().onValueChanged.AddListener(delegate { ValueChangeCheck(); });
    }

    /**
     * ValueChangeCheck
     * Called when the value of the input field is changed
     * (provides a dynamically changing size of the array in the text box

```

```

    * based on the user's choice)
    */
    public void ValueChangeCheck()
    {
        int size = (int)(GetComponent<Slider>().value);

        // Change the maximum value of the index slider
        IndexSlider.maxValue = size;

        List<int> initial_array = new List<int>();
        string array_string = "Array: < ";
        Random r = new Random(DateTime.Now.Ticks.GetHashCode());
        while (initial_array.Count != size)
        {
            int num = r.Next(0, 100);
            if (!initial_array.Contains(num))
                initial_array.Add(num);
            else
                continue;

            if (initial_array.Count < size)
                array_string += num + ", ";
            else
                array_string += num;
        }
        ArraySizeText.text = "Array size: " + GetComponent<Slider>().value.ToString();
        ArrayText.text = array_string + ">";
        SelectionAlgorithm.GetComponent<SelectionAlgorithm>().initial_array = initial_array;
    }
}

```

SliderIndexText.cs

```

using TMPro;
using UnityEngine;
using UnityEngine.UI;
using Random = System.Random;

/**
 * SliderIndexText script
 * Used to control the functionality of the index slider
 * (Selection Algorithm)
 */
public class SliderIndexText : MonoBehaviour {

    // UI elements being controlled
    public TextMeshProUGUI IndexText;
    public GameObject SelectionAlgorithm;

    void Start()
    {
        GetComponent<Slider>().onValueChanged.AddListener(delegate { ValueChangeCheck(); });
    }

    /**
     * ValueChangeCheck
     * Called when the value of the input field is changed
     * (provides a dynamically changing index in the text box
     * based on the user's choice)
     */
    public void ValueChangeCheck()
    {
        IndexText.text = "Index: " + GetComponent<Slider>().value.ToString();
    }
}

```

```

        SelectionAlgorithm.GetComponent<SelectionAlgorithm>().index = (int)
GetComponent<Slider>().value;
    }
}

```

FireworksController.cs

```

using System.Collections;
using UnityEngine;

/**
 * FireworksController script
 * Controls the time the fireworks are being displayed
 */
public class FireworksController : MonoBehaviour {

    public float waitSeconds = 5f;

    // Use this for initialization
    void Start () {
        StartCoroutine("waitToDisable");
    }

    IEnumerator waitToDisable()
    {
        yield return new WaitForSeconds(waitSeconds);
        gameObject.SetActive(false);
    }
}

```

SelectionAlgorithm.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;
using Random = System.Random;

/**
 * SelectionAlgorithm script
 * The main script used to display and control
 * our main Selection Algorithm UI elements and functionality.
 */
public class SelectionAlgorithm : MonoBehaviour {

    /* Variables used in the main visual 3d selection algorithm function */
    public List<int> initial_array = new List<int> { 1, 2, 3, 4, 5 }; //Default array
    public int index = 1; //Default index

    /* UI Elements being controlled */
    public Transform main_3darray;
    public Transform sorted_main_3darray;
    public Transform current_3darray;
    public Transform index_text;
    public Transform current_index_text;
    public Transform expected_solution_text;
    public Transform medians_container;

    public Transform less_than_container;
    public Transform number_of_less_text;
}

```

```

public Transform number_of_more_text;
public Transform more_than_container;

public Transform x_position;

public Transform five_container;
public Transform start_placeholder;
public Transform end_placeholder;

public Transform result_object;
public GameObject fireworks;
/*END Of UI elements*/

// The list with all the 3D group containers
private List<Transform> groups_3dcontainers;

/* Variables used for the interface (UI) */
private float speed;
private bool use_step_toggle;
private bool step_forward;
public TextMeshProUGUI current_step_text;

/*Start of UI Functions*/
/*Functions enabled by the UI buttons for controlling the flow of the program*/
public void setSpeed(float s)
{
    this.speed = s;
}
/**
 * When the back button is pressed on the UI, it goes back to
 * the SelectionAlgorithm start screen
 */
public void BackButton()
{
    SceneManager.LoadScene("SelectionAlgorithm");
}
/**
 * When the exit button is pressed on the UI, it goes back to
 * the MainUI
 */
public void ExitButton()
{
    SceneManager.LoadScene("MainUI");
}
/**
 * setStepToggle function
 * When the step toggle is enabled, then use_step_toggle = true else false.
 */
public void setStepToggle(bool b)
{
    this.use_step_toggle = b;
}
/**
 * setStepForward function
 * When the step forward button is pressed, then step_forward = true else false.
 */
public void setStepForward(bool b)
{
    this.step_forward = b;
}
/*End of UI Functions*/

/*Start() is used for initialization*/
void Start () {
    speed = 2f;
    main_3darray.gameObject.SetActive(true);
}

```

```

        sorted_main_3darray.gameObject.SetActive(true);
        StartCoroutine("visual3DSelectionAlgorithm");
    }

    /**
     * IMPORTANT **
     * ABILITY TO CALL A COROUTINE WITHIN A COROUTINE
     * yield return StartCoroutine(waitForCoroutine());
     * Hence avoid usage of the same code multiple times.
     */
    public IEnumerator waitForCoroutine()
    {
        if (use_step_toggle)
        {
            yield return new WaitUntil(() => step_forward || !use_step_toggle);
            step_forward = false;
        }
        else
            yield return new WaitForSeconds(speed);
    }

    /**
     * visual3DSelectionAlgorithm function
     * Used to run the visual representation of the Selection algorithm.
     */
    public IEnumerator visual3DSelectionAlgorithm()
    {
        // UNCOMMENT THE FOLLOWING FOR TESTING
        // The number's index we want to find:
        //int index = 3;
        // Create an array containing a desired number of random integer
        // numbers up to 50 in size max. (from 0-100)
        /*List<int> initial_array = new List<int>();
        Random r = new Random(DateTime.Now.Ticks.GetHashCode());
        while (initial_array.Count != 50)
        {
            int num = r.Next(0, 100);
            if (!initial_array.Contains(num))
                initial_array.Add(num);
        }*/

        // Create a sorted array with all the elements of array being sorted!
        List<int> sorted_array = new List<int>();
        for (int i = 0; i < initial_array.Count; i++)
            sorted_array.Add(initial_array[i]);
        sorted_array.Sort();

        // Display the main array (unsorted)
        current_step_text.SetText("Displaying the given array...");
        main_3darray.GetComponent<Array3DController>().spawnSpheres(initial_array);
        yield return StartCoroutine(waitForCoroutine());

        // Display the index for which you are searching the element
        current_step_text.SetText("Index we are searching for: "+index);
        index_text.GetComponent<TextMeshPro>().text = "Index being searched for: " + index;
        yield return StartCoroutine(waitForCoroutine());

        // Display the sorted array in the 3d sorted array and highlight the
        // element we are searching with red color
        current_step_text.SetText("Displaying the sorted array and the item we are searching
for...");
        sorted_main_3darray.GetComponent<Array3DController>().spawnSpheres(sorted_array);
        sorted_main_3darray.GetComponent<Array3DController>().setColor(index-1, Color.red);
        yield return StartCoroutine(waitForCoroutine());

        // Display the expected element and remove the sorted array

```

```

        expected_solution_text.GetComponent<TextMeshPro>().text = "Expected Solution: " +
sorted_array[index-1];
sorted_main_3darray.GetComponent<Array3DController>().clearArray();
sorted_main_3darray.gameObject.SetActive(false);
yield return StartCoroutine(waitForCoroutine());

//Stack<List<int>> array_stack = new Stack<List<int>>();
//array_stack.Push(array);

List<int> curr_array = initial_array;
int curr_index = index;
int result = 0;
while (true)
{
    current_index_text.GetComponent<TextMeshPro>().text = "Current Index: " + curr_index;

    int n = curr_array.Count;

    // Display the current array in the 3d array
    current_step_text.SetText("1) Displaying the current array we are searching in...");
    current_3darray.GetComponent<Array3DController>().spawnSpheres(curr_array);
    yield return StartCoroutine(waitForCoroutine());

    // Divide the array into groups of 5 making ceil(n/5) groups
    int num_of_groups = (int)Math.Ceiling(n / 5.0);
    Debug.Log(num_of_groups);

    // Create the groups of 5 of the main array which has
    // been splitted
    current_step_text.SetText("2) Splitting current array to "+num_of_groups+" groups of
5...");
    yield return StartCoroutine(waitForCoroutine());
    List<List<int>> five_groups = new List<List<int>>();
    List<int> curr_group = new List<int>();
    for (int i = 0; i < n; i++)
    {
        if (i % 5 == 0 && i != 0)
        {
            five_groups.Add(curr_group);
            curr_group = new List<int>();
        }
        curr_group.Add(curr_array[i]);
    }
    if (curr_group != null) five_groups.Add(curr_group);

    // Linearly interpolate and place the 3d array groups within a range of
    // start and end placeholder positions.
    groups_3dcontainers = new List<Transform>();
    int interp_groups = five_groups.Count;
    if (five_groups.Count > 5) interp_groups = 5;
    // ***** Fixed the error when interpolating only 1 object!
    if (five_groups.Count == 1)
    {
        five_container.GetChild(0).GetComponent<TextMeshPro>().text = "Group " + 0;
        Transform instantiatedObject =
            Instantiate(
                five_container,
                new Vector3((start_placeholder.position.x + end_placeholder.position.x) / 2,
                    start_placeholder.position.y, start_placeholder.position.z),
                five_container.rotation);
        groups_3dcontainers.Add(instantiatedObject);
    }
    groups_3dcontainers[0].GetComponent<FixedArray3DController>().spawnSpheres(five_groups[0]);
    }
    else
    {
        for (int i = 0; i < interp_groups; i++)

```

```

        {
            five_container.GetChild(0).GetComponent<TextMeshPro>().text = "Group " + i;
            Transform instantiatedObject =
                Instantiate(
                    five_container,
                    Vector3.Lerp(start_placeholder.position, end_placeholder.position,
(float)i / (float)(interp_groups - 1)),
                    five_container.rotation);
            groups_3dcontainers.Add(instantiatedObject);

groups_3dcontainers[i].GetComponent<FixedArray3DController>().spawnSpheres(five_groups[i]);
            yield return StartCoroutine(waitForCoroutine());
        }
    }

    // If we have more than 5 groups, create a new
    // line to place the rest of the containers below
    // the first one.
    if (five_groups.Count > 5)
    {
        Vector3 new_start = new Vector3(start_placeholder.position.x,
start_placeholder.position.y - 1, start_placeholder.position.z);
        Vector3 new_end = new Vector3(end_placeholder.position.x, end_placeholder.position.y
- 1, end_placeholder.position.z);
        if (five_groups.Count == 6)
        {
            five_container.GetChild(0).GetComponent<TextMeshPro>().text = "Group " + 5;
            Transform instantiatedObject =
                Instantiate(
                    five_container,
                    new Vector3((new_start.x + new_end.x)/2, new_start.y, new_start.z),
                    five_container.rotation);
            groups_3dcontainers.Add(instantiatedObject);

groups_3dcontainers[5].GetComponent<FixedArray3DController>().spawnSpheres(five_groups[5]);
        }
        else
        {
            for (int i = 5; i < five_groups.Count; i++)
            {
                five_container.GetChild(0).GetComponent<TextMeshPro>().text = "Group " + i;
                Transform instantiatedObject =
                    Instantiate(
                        five_container,
                        Vector3.Lerp(new_start, new_end, (float)(i - 5) /
(float)(five_groups.Count - 6f)),
                        five_container.rotation);
                groups_3dcontainers.Add(instantiatedObject);

groups_3dcontainers[i].GetComponent<FixedArray3DController>().spawnSpheres(five_groups[i]);
                yield return StartCoroutine(waitForCoroutine());
            }
        }
    }

    yield return StartCoroutine(waitForCoroutine());

    // Sort the groups of five of each container
    // (without animation)
    for (int i = 0; i < groups_3dcontainers.Count; i++)
    {
        current_step_text.SetText("3) Sorting group "+i);
        groups_3dcontainers[i].GetComponent<FixedArray3DController>().sortArray();
        yield return StartCoroutine(waitForCoroutine());
    }
}

```

```

// Find the medians in each array and highlight them using red color
List<int> medians_list = new List<int>();
List<int> median_positions_list = new List<int>();
for (int i = 0; i < groups_3dcontainers.Count; i++)
{
    // Used a trick from EPL445 for rounding up to ceiling x = int(n/2 + 0.5)
    int median_position =
(int)((groups_3dcontainers[i].GetComponent<FixedArray3DController>().object_array.Count / 2) +
0.5f);

groups_3dcontainers[i].GetComponent<FixedArray3DController>().object_array[median_position].GetCompo
nent<Renderer>().material.color = Color.red;
    int median =
groups_3dcontainers[i].GetComponent<FixedArray3DController>().number_array[median_position];
    medians_list.Add(median);
    median_positions_list.Add(median_position);
    current_step_text.SetText("4) Median of group "+i+" is "+ median);
    yield return StartCoroutine(waitForCoroutine());
}

// Animate the spheres to go over the large median container
// and then Place all the medians into the large median container
current_step_text.SetText("5) Transferring all the medians to the medians bucket...");
medians_container.gameObject.SetActive(true); // Enable the medians container
for (int i = 0; i < medians_list.Count; i++)
{
    // Animate and destroy the median spheres towards the medians container
    Transform from_sphere =
groups_3dcontainers[i].GetComponent<FixedArray3DController>().object_array[median_positions_list[i]]
;

groups_3dcontainers[i].GetComponent<FixedArray3DController>().object_array.Remove(from_sphere);
    from_sphere.GetComponent<BallMover>().target = medians_container.position;
    from_sphere.GetComponent<BallMover>().active = true;
    yield return new WaitUntil(() => !from_sphere.GetComponent<BallMover>().active);
    Destroy(from_sphere.gameObject);

    // Add the sphere to the container
    medians_container.GetComponent<LargeContainerController>().addItem(medians_list[i] +
""");
    yield return new WaitForSeconds(0.5f);
}

// Sort the medians list and find the median of medians:
medians_list.Sort();
int x = medians_list[(int)((medians_list.Count / 2) + 0.5f)];

// Remove the group of five containers
for (int i = groups_3dcontainers.Count - 1; i >= 0; i--)
{
    groups_3dcontainers[i].GetComponent<FixedArray3DController>().clearArray();
    Transform rem_group = groups_3dcontainers[i];
    groups_3dcontainers.Remove(rem_group);
    Destroy(rem_group.gameObject);
}
groups_3dcontainers.Clear();

// Enable the less than and more than containers to compare the X value.
less_than_container.gameObject.SetActive(true);
more_than_container.gameObject.SetActive(true);
x_position.gameObject.SetActive(true);

yield return StartCoroutine(waitForCoroutine());

```

```

        // Remove the median of medians and show it to the user using animation
        current_step_text.SetText("6) The median of all medians is "+x);
        Transform x_transform =
medians_container.GetComponent<LargeContainerController>().getItem(x+"");
        // Remove physics from X element sphere
        Destroy(x_transform.GetComponent<Rigidbody>());

medians_container.GetComponent<LargeContainerController>().PlacedItems.Remove(x_transform);
medians_container.GetComponent<LargeContainerController>().noOfItems--;
x_transform.GetComponent<BallMover>().updatePosition();
x_transform.GetComponent<BallMover>().target = x_position.position;
x_transform.GetComponent<BallMover>().active = true;
yield return new WaitUntil(() => !x_transform.GetComponent<BallMover>().active);

// The current X median of medians element gets instantiated in the center of game view
Transform curr_x = Instantiate(x_transform, x_position.position,
Quaternion.LookRotation(Vector3.forward));
Destroy(x_transform.gameObject);

// Empty the medians container and hide from game view
medians_container.GetComponent<LargeContainerController>().makeEmpty();
medians_container.gameObject.SetActive(false);
yield return StartCoroutine(waitForCoroutine());

// Place the spheres less than x into the less than x container
// and those who are more than x into the more than x container
// TODO place animations on the current array which sends spheres here
current_step_text.SetText("7) Finding all the numbers which are less from " + x+ " and
all which are bigger...");
List<int> less_than_x_array = new List<int>();
List<int> more_than_x_array = new List<int>();
List<int> lessx_five = new List<int>();
List<int> morex_five = new List<int>();
for (int i = 0; i < curr_array.Count; i++)
{
    if (curr_array[i] < x)
    {
        less_than_x_array.Add(curr_array[i]);
        lessx_five.Add(curr_array[i]);
        if (lessx_five.Count == 5)
        {
            for(int j=0; j<5; j++)

less_than_container.GetComponent<LargeContainerController>().addItem(lessx_five[j] + "",
Color.blue);

            lessx_five = new List<int>();
            yield return new WaitForSeconds(0.3f);
        }
    }
    else if(curr_array[i] > x)
    {
        more_than_x_array.Add(curr_array[i]);
        morex_five.Add(curr_array[i]);
        if(morex_five.Count == 5)
        {
            for (int j = 0; j < 5; j++)

more_than_container.GetComponent<LargeContainerController>().addItem(morex_five[j] + "",
Color.blue);

            morex_five = new List<int>();
            yield return new WaitForSeconds(0.3f);
        }
    }
}
// Add the remaining splitted less and more numbers at the same time in their
corresponding containers
for (int j = 0; j < lessx_five.Count; j++)

```

```

        less_than_container.GetComponent<LargeContainerController>().addItem(lessx_five[j] +
"", Color.blue);
        yield return new WaitForSeconds(0.3f);
        for (int j = 0; j < morex_five.Count; j++)
            more_than_container.GetComponent<LargeContainerController>().addItem(morex_five[j] +
"", Color.blue);
        yield return new WaitForSeconds(0.3f);

        // the number of items less than x
        int num_of_less = less_than_x_array.Count;
        int num_of_more = (n - (num_of_less + 1));
        // Enable number of less and more texts in game view
        number_of_less_text.gameObject.SetActive(true);
        number_of_more_text.gameObject.SetActive(true);
        number_of_less_text.GetComponent<TextMeshPro>().text = "Number of less: " + num_of_less;
        number_of_more_text.GetComponent<TextMeshPro>().text = "Number of more: " + num_of_more;
        current_step_text.SetText("8) Number of less than " + x + " is: " + num_of_less + " and
Number of more: "+num_of_more);
        yield return StartCoroutine(waitForCoroutine());

        if(curr_index == num_of_less + 1)
        {
            current_step_text.SetText("9) Current Index (" + curr_index + ") is EQUAL with the
Number of less + 1 (" + (num_of_less + 1) + ")");
            yield return StartCoroutine(waitForCoroutine());

            current_step_text.SetText("10) Result has been found!!");
            yield return StartCoroutine(waitForCoroutine());

            Debug.Log("Found! " + x);
            result = x;
        }
        else if(curr_index < num_of_less + 1)
        {
            current_step_text.SetText("9) Current Index (" + curr_index + ") is LESS than the
Number of less + 1 (" + (num_of_less + 1) + ")");
            yield return StartCoroutine(waitForCoroutine());
            current_step_text.SetText("10) Use the same current index for the items that are
LESS than "+x);
            yield return StartCoroutine(waitForCoroutine());
            current_3darray.GetComponent<Array3DController>().clearArray();
            curr_array = less_than_x_array;
            current_step_text.SetText("11) Run algorithm again recursively.");
        }
        else
        {
            current_step_text.SetText("9) Curr Index (" + curr_index + ") is MORE than the
Number of less + 1 (" + (num_of_less + 1) + ")");
            curr_index = curr_index - num_of_less - 1;
            current_3darray.GetComponent<Array3DController>().clearArray();
            curr_array = more_than_x_array;
            yield return StartCoroutine(waitForCoroutine());
            current_step_text.SetText("10) Current index -= #Less - 1 [" + curr_index + "] for
the items that are MORE than " + x);
            yield return StartCoroutine(waitForCoroutine());
            current_step_text.SetText("10) Use the same current index for the items that are
LESS than " + x);
            yield return StartCoroutine(waitForCoroutine());
            current_step_text.SetText("11) Run algorithm again recursively.");
        }
        // Clear and disable the less and more than x containers
        less_than_container.GetComponent<LargeContainerController>().makeEmpty();
        less_than_container.gameObject.SetActive(false);
        more_than_container.GetComponent<LargeContainerController>().makeEmpty();
        more_than_container.gameObject.SetActive(false);
        number_of_less_text.gameObject.SetActive(false);

```

```

        number_of_more_text.gameObject.SetActive(false);
        x_position.gameObject.SetActive(false);
        Destroy(curr_x.gameObject);
        if (curr_index == num_of_less + 1)
            break;
        yield return StartCoroutine(waitForCoroutine());
    }

    // Show result
    current_step_text.SetText("Number found!");
    result_object.gameObject.SetActive(true);
    result_object.GetChild(0).GetComponent<TextMeshPro>().SetText("Number found on index " +
index + " is: " + result);

    //... with some fireworks babyyy!
    fireworks.SetActive(true);

    yield return null;
}
}

```

Παράρτημα Ε

Πιο κάτω βρίσκονται τα scripts και σχεδιαστικό υλικό που χρησιμοποιηθήκαν κατά την υλοποίηση των αλγορίθμων Fibonacci, του τρισδιάστατου δυαδικού δένδρου και του Δυναμικού προγραμματισμού. Αναφέρονται κυρίως για το Κεφάλαιο 6.

FibonacciBinary3DTreeController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/**
 * FibonacciBinary3DTreeController script
 * Used to controll the functionality of the 3D
 * binary tree used for displaying each recursive
 * Fibonacci Call
 */
public class FibonacciBinary3DTreeController : MonoBehaviour {

    public Transform root_transform;    /*The root of the tree (3D GameObject - sphere)*/
    public TreeNode root;               /*The script attached to the 3D root node*/
    public int N = 6;                   /*Fibonacci number used to calculate the tree (Default is
6)*/
    public bool begin = false;          /*If true, then start the instantiation of the tree*/
    private float speed;                /*Speed of instantiation*/

    // Use this for initialization
    void Start () {
        //StartCoroutine("testingTree");
        // Get N from DynamicProgramming class
        speed = 1f;
    }

    void Update()
    {
        if (begin)
        {
            StartCoroutine("createFibonacciTree", N);

            //PROBLEM: CANT INSTANTLY CREATE THE BINARY TREE!
            //createFibonacciTreeInstantly(N);
            begin = false;
        }
    }

    // Sets the speed of the tree instantiation
    public void setSpeed(float s)
    {
        this.speed = s;
    }

    /*The normal fibonacci recursive function*/
    private int normFib(int n)
    {
        if (n <= 1)
        {
            return n;
        }
        else
        {
            int n1 = normFib(n - 1);

```

```

        int n2 = normFib(n - 2);
        return n1 + n2;
    }
}

/**
 * createFibonacciTree
 * The following function creates a 3D binary fibonacci tree
 * using the number n as the base number.
 */
public IEnumerator createFibonacciTree(int n)
{
    root.setLevel(0);
    root.setValue(n);
    root.setText("Fib("+n+")");
    yield return new WaitForSeconds(1f);

    Stack<TreeNode> fibStack = new Stack<TreeNode>();
    fibStack.Push(root);

    while (fibStack.Count != 0)
    {
        // Current Number to create it's children
        TreeNode currNode = fibStack.Pop();
        float currValue = currNode.getValue();
        if (currValue <= 1f)
        {
            //if the current number is one of the base cases (0 or 1) then that's a leaf.
            continue;
        }

        // Left and right child values
        float n1 = currValue - 1;
        float n2 = currValue - 2;

        // This is the left child of currN
        currNode.addLeftChild("Fib(" + n1 + ")", n1);
        yield return new WaitForSeconds(speed);

        // This is the right child of currN
        currNode.addRightChild("Fib(" + n2 + ")", n2);
        yield return new WaitForSeconds(speed);

        // Add the right child first to the stack
        fibStack.Push(currNode.right_child);

        // Then add the left and also remember all the left
        // childs by adding to the other stack
        fibStack.Push(currNode.left_child);
    }
}

/*Used for testing*/
public IEnumerator testingTree()
{
    root.setLevel(0);
    root.setValue(11);
    root.setText(11 + "");
    yield return new WaitForSeconds(1f);
    add(6);
    yield return new WaitForSeconds(1f);
    add(19);
}

/**
 * add function
 * Used to add recursively a value to our tree
 */

```

```

public void add(float value)
{
    addRecursive(root, value);
}

/**
 * addRecursive function
 * Adds a value to the current TreeNode (Binary Tree)
 */
private void addRecursive(TreeNode current, float value)
{
    /*if (current == null)
    {
        return new TreeNode(value, value + "", 0);
    }*/

    // Add to left child if < than current node
    if (value < current.value)
    {
        if (current.left_child == null)
            current.addLeftChild(value + "", value);
        else
            addRecursive(current.left_child, value);
    }
    // Add to right child if > than current node
    else if (value > current.value)
    {
        if (current.right_child == null)
            current.addRightChild(value + "", value);
        else
            addRecursive(current.right_child, value);
    }
    else
    {
        // value already exists
        return;
    }
}
}

```

TreeNode.cs

```

using TMPro;
using UnityEngine;

/**
 * TreeNode script
 * It is attached to a 3D node used to be instantiated
 * when the binary tree is being created
 */
public class TreeNode : MonoBehaviour{

    public TextMeshPro label;           /*Label of the node on the 3D sphere*/
    public float value;                 /*node's value*/
    private int level;                 /*Depth Level*/
    public TreeNode left_child;         /*Left child node*/
    public TreeNode right_child;        /*Right child node*/

    public Transform left_transform;    /*Left child GameObject*/
    public Transform right_transform;   /*Right child GameObject*/
    public Transform left_arrow;        /*Left 3D arrow*/
    public Transform right_arrow;       /*Right 3D arrow*/

    // Is used to be instantiated as children.
    public Transform tree_node;
}

```

```

// Use this for initialization
void Start()
{
    left_child = null;
    right_child = null;
    left_arrow.gameObject.SetActive(false);
    right_arrow.gameObject.SetActive(false);
}

void Update()
{
    //transform.Rotate(Vector3.forward, 1);
    //transform.localScale += new Vector3(0.003f,0.003f,0.003f);
}

/**
 * TreeNode constructor
 */
public TreeNode(float val, string s, int l)
{
    this.value = val;
    setText(s);
    setLevel(l);
    left_child = null;
    right_child = null;
    left_arrow.gameObject.SetActive(false);
    right_arrow.gameObject.SetActive(false);
}

/*Getters and setters for all our variables*/
public void setText(string s)
{
    label.text = s;
}
public string getText()
{
    return label.text;
}
public void setValue(float val)
{
    this.value = val;
}
public float getValue()
{
    return this.value;
}
public int getLevel()
{
    return this.level;
}
/*End of Getters and setters*/

/**
 * setLevel function
 * Sets the level of the node and based on that level,
 * the function adjusts the arrows and position of the children of the
 * new node to fit the level of the tree. As a result there will not
 * be any collisions between the nodes of the 3D tree up to an acceptable level.
 */
public void setLevel(int l)
{
    this.level = l;
    float sum_rot_factor = 0;
    float sum_scale_factor = 0.1f;
    if (l == 1)
    {

```

```

        float rot_factor = 36f;
        sum_rot_factor += rot_factor;
        left_arrow.Rotate(Vector3.forward, rot_factor);
        left_arrow.localScale -= new Vector3(0, 0.1f, 0);
        right_arrow.Rotate(Vector3.back, rot_factor);
        right_arrow.localScale -= new Vector3(0, 0.1f, 0);
    }
    else if(l > 1)
    {
        for (int i = 1; i <= l; i++)
        {
            float rot_factor = 35f / i;
            float scale_factor = 0.12f / i;
            sum_rot_factor += rot_factor;
            sum_scale_factor += scale_factor;
            if (sum_rot_factor <= 79f)
            {
                left_arrow.Rotate(Vector3.forward, rot_factor);
                right_arrow.Rotate(Vector3.back, rot_factor);
            }
            if (sum_scale_factor <= 0.37)
            {
                left_arrow.localScale -= new Vector3(0, scale_factor, 0);
                right_arrow.localScale -= new Vector3(0, scale_factor, 0);
            }
            label.fontSize -= 0.1f;
        }
    }
}

/*Left and Right 3D arrow color change functions*/
public void setLeftArrowColor(Color col)
{
    left_arrow.GetChild(0).GetComponent<Renderer>().material.color = col;
    left_arrow.GetChild(1).GetComponent<Renderer>().material.color = col;
    left_arrow.GetChild(2).GetComponent<Renderer>().material.color = col;
}
public void setRightArrowColor(Color col)
{
    right_arrow.GetChild(0).GetComponent<Renderer>().material.color = col;
    right_arrow.GetChild(1).GetComponent<Renderer>().material.color = col;
    right_arrow.GetChild(2).GetComponent<Renderer>().material.color = col;
}

/**
 * addLeftChild and addRightChild functions
 * These functions are used to both add the values of the
 * corresponding child and the display text to the 3D visual
 * tree
 */
public void addLeftChild(string s, float value)
{
    // Make the left arrow visible and instantiate a new node
    // at left child position
    left_arrow.gameObject.SetActive(true);
    left_transform = Instantiate(tree_node, left_transform.position,
Quaternion.LookRotation(Vector3.forward));

    // Based on our current level, set the size of the node
    for (int i=1; i<level; i++)
        left_transform.GetChild(3)..localScale -= new Vector3(0.2f / i, 0.2f / i, 0.2f / i);

    // Set the text, value and level of the new child
    left_child = left_transform.GetComponent<TreeNode>();
    left_child.setText(s);
    left_child.setValue(value);
    left_child.setLevel(level+1);
}

```

```

    }
    public void addRightChild(string s, float value)
    {
        right_arrow.gameObject.SetActive(true);
        right_transform = Instantiate(tree_node, right_transform.position,
Quaternion.LookRotation(Vector3.forward));
        for (int i = 1; i < level; i++)
            right_transform.GetChild(3).localScale -= new Vector3(0.2f / i, 0.2f / i, 0.2f / i);
        right_child = right_transform.GetComponent<TreeNode>();
        right_child.setText(s);
        right_child.setValue(value);
        right_child.setLevel(level+1);
    }

    /*FUTURE DEVELOPMENT: ADD REMOVE CHILDS, MORE FUNCTIONALLITY ETC*/
}

```

DynamicProgramming.cs

```

using System.Collections;
using System.Threading;
using TMPPro;
using UnityEngine;
using UnityEngine.SceneManagement;

/**
 * DynamicProgramming script
 * The main script used to control all the elements
 * of the dynamic programming section of our program.
 */
public class DynamicProgramming : MonoBehaviour {

    public double N = 6; // Current N used for every fibonacci algorithm
    public double[] mem_fib; // The memoization table
    private float execution_time = -1; // Current execution time of the respective algorithm
    private double result; // Result of current fibonacci

    /* Objects being displayed based on each algorithm */
    public GameObject sphere_indicator;
    public GameObject algorithm_text_object;
    private TextMeshPro algorithm_text;
    public GameObject execution_time_text_object;
    private TextMeshPro execution_time_text;
    public FibonacciBinary3DTreeController fibonacci_3d_tree;
    public GameObject main_camera;

    /* Variables used for the interface (UI) */
    private int run_choice;
    private float speed;
    private bool use_step_toggle;
    private bool step_forward;
    public TextMeshProUGUI current_step_text;
    public TMP_InputField n_input_field;

    // Use this for initialization
    void Start () {
        algorithm_text = algorithm_text_object.GetComponentInChildren<TextMeshPro>();
        execution_time_text = execution_time_text_object.GetComponentInChildren<TextMeshPro>();
        StartCoroutine("mainControlFibonacci");
    }

    /**START of UI Functions***/
    public void setN(int n)
    {
        this.N = n;
    }
}

```

```

}
public void setRunChoice(int c)
{
    this.run_choice = c;
}
/*Functions enabled by the UI buttons for controlling the flow of the program*/
public void setSpeed(float s)
{
    this.speed = s;
}
/**
 * When the back button is pressed on the UI, it goes back to
 * the DynamicProgramming start screen
 * */
public void BackButton()
{
    SceneManager.LoadScene("DynamicProgramming");
}
/**
 * When the exit button is pressed on the UI, it goes back to
 * the MainUI
 * */
public void ExitButton()
{
    SceneManager.LoadScene("MainUI");
}
/**END of UI Functions***/

// MAIN CONTROL COROUTINE
public IEnumerator mainControlFibonacci()
{
    // The thread is empty at the beginning
    Thread run_thread = null;

    // Start time using built in stopwatch
    //float time_start = Environment.TickCount * 0.001f;
    var stopWatch = System.Diagnostics.Stopwatch.StartNew();
    stopWatch.Start();

    switch (run_choice)
    {
        case 0:
            // Recursive with tree
            current_step_text.text = "Building Tree...";
            recursiveFibonacciWithBinaryTree();
            //StartCoroutine("recursiveFibonacciWithBinaryTree");
            break;
        case 1:
            // Recursive
            algorithm_text.text =
                "private double recursiveFib(double n)\n" +
                "{\n" +
                "    if (n <= 1)\n" +
                "    {\n" +
                "        return n;\n" +
                "    }\n" +
                "    else\n" +
                "    {\n" +
                "        double n1 = recursiveFib(n - 1);\n" +
                "        double n2 = recursiveFib(n - 2);\n" +
                "        return n1 + n2;\n" +
                "    }\n" +
                "};";
            run_thread = new Thread(recursiveFibonacci);
            run_thread.Start();
            current_step_text.text = "Running...";
            //recursiveFibonacci();
    }
}

```

```

        break;
    case 2:
        // Memoization
        algorithm_text.text =
            "private double memoizationFib(double n)\n" +
            "{\n" +
            "    if (mem_fib[n] != -1)\n" +
            "    {\n" +
            "        return mem_fib[n];\n" +
            "    }\n" +
            "    else if(n <= 1)\n" +
            "    {\n" +
            "        return n;\n" +
            "    }\n" +
            "    else\n" +
            "    {\n" +
            "        double fib_res = memoizationFib(n - 1) + memoizationFib(n - 2);\n" +
            "        mem_fib[n] = fib_res;\n" +
            "        return fib_res;\n" +
            "    }\n" +
            "}\n";
        run_thread = new Thread(memoizationFibonacci);
        run_thread.Start();
        current_step_text.text = "Running...";
        //memoizationFibonacci();
        //StartCoroutine("memoizationFibonacci");
        break;
    case 3:
        // Iterative
        algorithm_text.text =
            "private double iterativeFib(double n)\n" +
            "{\n" +
            "    double[] F = new double[n+1];\n" +
            "    F[0] = 0;\n" +
            "    F[1] = 1;\n" +
            "    for(int i=2; i<=n; i++)\n" +
            "    {\n" +
            "        F[i] = F[i - 1] + F[i - 2];\n" +
            "    }\n" +
            "    return F[n];\n" +
            "}\n";
        run_thread = new Thread(iterativeFibonacci);
        run_thread.Start();
        current_step_text.text = "Running...";
        //iterativeFibonacci();
        //StartCoroutine("iterativeFibonacci");
        break;
}

if (run_choice != 0)
{
    // Enable the algorithm gameobjects
    sphere_indicator.SetActive(true);
    algorithm_text_object.SetActive(true);
    execution_time_text_object.SetActive(true);

    // Blinking of the sphere until the algorithm finishes its run
    while (run_thread.IsAlive)
    {
        //execution_time_text.text = "Execution Time: " + stopwatch.ElapsedMilliseconds + "
milisecs";

        sphere_indicator.GetComponent<Renderer>().material.color = Color.white;
        //yield return new WaitUntil(()=> !run_thread.IsAlive || new WaitForSeconds(0.2f)) ;
        yield return new WaitForSeconds(0.5f);
        if (!run_thread.IsAlive)
            break;
        sphere_indicator.GetComponent<Renderer>().material.color = Color.red;
    }
}

```

```

        yield return new WaitForSeconds(0.5f);
    }
    sphere_indicator.GetComponent<Renderer>().material.color = Color.green;

    // When the thread is finished, display the execution time
    // Execution time = End time - Start time
    //execution_time = (Environment.TickCount * 0.001f) - time_start;
    stopwatch.Stop();
    execution_time = stopwatch.ElapsedMilliseconds;
    // Frame rate overhead - SET TIME TO 1ms
    if (execution_time > 0 && execution_time <= 600)
    {
        execution_time = 1;
    }
    execution_time_text.text = "Execution Time: " + execution_time + " milisecs";

    // Display the result
    current_step_text.text = "Result = " + result;
}
yield return null;
}

// FIBONACCI ALGORITHMS
public void recursiveFibonacciWithBinaryTree()
{
    // Change camera position and size based on N for fitting the
    // binary tree correctly on the screen.
    // Decrease Y position and increase camera size with a bigger than 6 N.
    if (N > 6)
    {
        main_camera.gameObject.transform.position = new
Vector3(main_camera.transform.position.x, main_camera.transform.position.y - (0.7f * ((int)N - 6)),
main_camera.transform.position.z);
        main_camera.GetComponent<Camera>().orthographicSize += 0.8f * ((int)N - 6);
    }
    fibonacci_3d_tree.N = (int)N;
    fibonacci_3d_tree.begin = true;
    //yield return null;
}

public void recursiveFibonacci()
{
    //float start_time = Time.time;
    result = recursiveFib(N);
    //execution_time = Time.time - start_time;

    Debug.Log("Execution time: " + execution_time.ToString());
    Debug.Log("Result: " + result);
    //yield return null;
}

private double recursiveFib(double n)
{
    //Debug.Log("Fibonacci(" + n + ")");
    if (n <= 1)
    {
        return n;
    }
    else
    {
        double n1 = recursiveFib(n - 1);
        double n2 = recursiveFib(n - 2);
        return n1 + n2;
    }
}

public void memoizationFibonacci()
{

```

```

        // Initialize the memoization table.
        mem_fib = new double[(int)N + 1];
        for (int i = 0; i <= N; i++)
        {
            mem_fib[i] = -1;
        }
        result = memoizationFib(N);
        //execution_time = Time.time - start_time;
        Debug.Log("memoizationFibonacci - Execution time: " + execution_time.ToString());
        Debug.Log("Result: " + result);
        //yield return null;
    }
    private double memoizationFib(double n)
    {
        // If the current recursive call is stored in the memoization
        // array then retrieve it and return the stored answer.
        //Debug.Log("Fibonacci("+n+")");
        if (mem_fib[(int)n] != -1)
        {
            return mem_fib[(int)n];
        }
        else if (n <= 1)
        {
            return n;
        }
        else
        {
            double fib_res = memoizationFib(n - 1) + memoizationFib(n - 2);
            mem_fib[(int)n] = fib_res;
            return fib_res;
        }
    }
}

public void iterativeFibonacci()
{
    //float start_time = Time.time;
    result = iterativeFib(N);
    //execution_time = Time.time - start_time;
    Debug.Log("iterativeFibonacci - Execution time: " + execution_time.ToString());
    Debug.Log("Result: " + result);
    //yield return null;
}

private double iterativeFib(double n)
{
    double[] F = new double[(int)n + 1];
    F[0] = 0;
    F[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        F[i] = F[i - 1] + F[i - 2];
    }
    return F[(int)n];
}
}

```

Παράρτημα Z

Πιο κάτω βρίσκονται τα scripts και σχεδιαστικό υλικό που χρησιμοποιήθηκαν κατά την υλοποίηση του τυχαιοποιημένου αλγορίθμου Karger, του τρισδιάστατου συνεκτικού πολυγράφου και της αποθήκευσής του. Αναφέρονται κυρίως για το Κεφάλαιο 7.

GraphLineController.cs

```
using UnityEngine;

/**
 * GraphLineController script
 * Attached onto a 3D cylinder line
 * Provides onMouse features for drawing
 * lines on a multigraph.
 */
public class GraphLineController : MonoBehaviour {

    /*If is hovered or is selected*/
    public bool is_hovered;
    public bool is_selected;

    // Initial color of the line
    private Color start_color;

    // Use this for initialization
    void Start () {
        is_hovered = false;
        is_selected = false;
        start_color = GetComponent<Renderer>().material.color;
    }

    /*When a mouse is hovered over the line, turn it red*/
    private void OnMouseEnter()
    {
        is_hovered = true;
        GetComponent<Renderer>().material.color = Color.red;
    }

    /*When the mouse gets away from the line, return to default color*/
    private void OnMouseExit()
    {
        is_hovered = false;
        GetComponent<Renderer>().material.color = start_color;
    }
}
```

GraphNodeController.cs

```
using System.Collections.Generic;
using System.Linq;
using TMPro;
using UnityEngine;

/**
 * GraphNodeController script
 * Attached onto a graph node (3d sphere) which allows it
 * to have graph node features and mouse events.
```

```

/**/
public class GraphNodeController : MonoBehaviour {

    private List<int> valueList;           // Values contained inside the node

    public bool is_hovered;               // True if mouse is hovering over the node
    public bool is_selected;             // True if mouse is selected (via Double Click)

    private bool one_click = false;      // True if one click
    private bool timer_running;          // True if timer is running
    private float timer_for_double_click = 0f; // A timer for detecting double clicks

    //this is how long in seconds to allow for a double click
    private float delay = 0.5f;

    private Color start_color;           // Initial color of the sphere

    // Use this for initialization
    void Start () {
        //this.valueList = new List<int>();
        is_hovered = false;
        is_selected = false;
        start_color = GetComponent<Renderer>().material.color;
    }

    private void Update()
    {
        if (one_click)
        {
            // if the time now is delay seconds more than when the first click started.
            if ((Time.time - timer_for_double_click) > delay)
            {
                // basically if thats true its been too long
                // and we want to reset so the next click is
                // simply a single click and not a double click.
                one_click = false;
            }
        }
    }

    /**
     * addValue function
     * Used to add a value to the node (supernode)
     */
    public void addValue(int val)
    {
        this.valueList.Add(val);
        GetComponentInChildren<TextMeshPro>().text = "" + val;
    }

    /**
     * setValues/getValues functions
     * Used to set/get the values of the node (supernode)
     */
    public void setValues(List<int> vals)
    {
        this.valueList = vals;

        // Set the textbox inside the node's sphere to show all the values
        string valuesString = string.Join(",", this.valueList.Select(x => x.ToString()).ToArray());
        GetComponentInChildren<TextMeshPro>().text = "{" + valuesString + "}";
    }

    public List<int> getValues()
    {
        return this.valueList;
    }
}

```

```

/**
 * resetNode function
 * Reset the mouse values of the node
 */
public void resetNode()
{
    is_hovered = false;
    is_selected = false;
    GetComponent<Renderer>().material.color = start_color;
}

public void OnMouseDown()
{
    if (!one_click) // first click no previous clicks
    {
        one_click = true;
        timer_for_double_click = Time.time; // save the current time
                                           // do one click things;
    }
    else
    {
        one_click = false; // found a double click, now reset

        // DOUBLE CLICK -> set color to green
        is_selected = !is_selected;
        GetComponent<Renderer>().material.color = Color.green;
    }
}

/*Used to detect if the mouse hovered over the node*/
private void OnMouseEnter()
{
    is_hovered = true;
    //Set color to red
    GetComponent<Renderer>().material.color = Color.red;
}

/*Used to detect if the mouse has moved away from the node*/
private void OnMouseExit()
{
    is_hovered = false;
    // If the node is not selected then set to default color
    // else set to the selected color (Green)
    if (!is_selected)
    {
        GetComponent<Renderer>().material.color = start_color;
    }
    else
    {
        GetComponent<Renderer>().material.color = Color.green;
    }
}
}

```

RndAlgoGraphCreatorUI.cs

```

using UnityEngine;

/**
 * RndAlgoGraphCreatorUI script
 * Used to show/hide graph creation instructions (Rand. Algorithm)
 */
public class RndAlgoGraphCreatorUI : MonoBehaviour {

    public GameObject instructionsGameObject;
}

```

```

    public void ShowHideInstructions()
    {
        bool isActive = instructionsGameObject.active;
        instructionsGameObject.SetActive(!isActive);
    }
}

```

GraphItem class and ScrollGraphList.cs

```

using UnityEngine;
using System.Collections.Generic;
using System.IO;

// A mini-class containing the data of a graph item.
[System.Serializable]
public class GraphItem
{
    /*The file name*/
    public string itemName;

    public GraphItem()
    {
    }

    public GraphItem(string itemN)
    {
        this.itemName = itemN;
    }

    /*A screenshot of the graph (OPTIONAL)*/
    //public Sprite icon;
}

public class ScrollGraphList : MonoBehaviour {

    public GameObject loadingUIObject;
    public GameObject algorithmUIObject;
    public GameObject createGraphObject;
    public List<GraphItem> itemList;
    public Transform contentPanel;
    //public ScrollGraphList otherShop;
    //public Text myGoldDisplay;
    public SimpleObjectPool buttonObjectPool;

    //public float gold = 20f;

    // Use this for initialization
    void Start()
    {
        RefreshDisplay();
    }

    public void RefreshDisplay()
    {
        //myGoldDisplay.text = "Gold: " + gold.ToString();
        RemoveButtons();
        AddButtonsFromFile();
        //AddButtons();
    }

    private void RemoveButtons()
    {
        while (contentPanel.childCount > 0)
        {

```

```

        GameObject toRemove = transform.GetChild(0).gameObject;
        buttonObjectPool.ReturnObject(toRemove);
    }
}

/**
 * AddButtonsFromFile function
 * Used to add buttons to the scroll list based on the
 * graph files that are in the default directory of the SaveSystem.
 */
private void AddButtonsFromFile()
{
    int fileCount = 0;
    itemList = new List<GraphItem>();
    string currFilePath = SaveSystem.defaultPath;
    //Application.persistentDataPath + "/Graph-" + fileCount + ".dat";
    string[] files = Directory.GetFiles(currFilePath);
    int i = 0;
    while (i < files.Length)
    {
        if (File.Exists(files[i]))
        {
            //GraphItem item = itemList[i];
            GameObject newButton = buttonObjectPool.GetObject();
            newButton.transform.SetParent(contentPanel);
            newButton.transform.SetAsFirstSibling();
            string fileName = Path.GetFileNameWithoutExtension(files[i]);
            GraphItem gI = new GraphItem(fileName);

            GraphItemButton graphButton = newButton.GetComponent<GraphItemButton>();
            graphButton.Setup(gI, this);
            fileCount++;

            newButton = null;
            graphButton = null;
            gI = null;
        }
        i++;
    }
}
}

```

GraphItemButton.cs

```

using UnityEngine;
using UnityEngine.UI;

/**
 * GraphItemButton script
 * Used to control the functionality of a button of a graph scroll list
 */
public class GraphItemButton : MonoBehaviour
{
    public Button buttonComponent;
    public Text nameLabel;

    private GraphItem item;
    private ScrollGraphList scrollList;

    // Use this for initialization
    void Start()
    {
        buttonComponent.onClick.AddListener(HandleClick);
    }
}

```

```

    public void Setup(GraphItem currentItem, ScrollGraphList currentScrollList)
    {
        item = currentItem;
        nameLabel.text = currentItem.itemName;
        //    iconImage.sprite = item.icon;
        scrollList = currentScrollList;
    }

    public void HandleClick()
    {
        scrollList.RefreshDisplay();
        // Load Graph
        scrollList.loadingUIObject.SetActive(false);
        scrollList.algorithmUIObject.SetActive(true);
        scrollList.createGraphObject.SetActive(true);

        scrollList.createGraphObject.GetComponent<CreateGraphUIController>().LoadGraph(item.itemName);
        //scrollList.AddItem(item, scrollList);
    }
}

```

GraphNode.cs

```

using System.Collections.Generic;
using System.Linq;
using UnityEngine;

/**
 * GraphNode script
 * The graph node object used to describe the nodes for
 * the 3D undirected graph
 */
public class GraphNode{

    public List<int> valueList;           // The list with all the values of the node
    private GameObject sphereObject;    // The 3D sphere object

    /*GraphNode Constructors*/
    public GraphNode()
    {
        this.valueList = new List<int>();
    }
    public GraphNode(List<int> vals, GameObject s)
    {
        vals.Sort();
        this.valueList = vals;
        this.sphereObject = s;
        this.sphereObject.GetComponent<GraphNodeController>().setValues(vals);
    }
    public GraphNode(List<int> vals, Vector3 pos)
    {
        vals.Sort();
        this.valueList = vals;
        this.sphereObject = GameObject.Instantiate(GameObject.FindGameObjectWithTag("3DNode"), pos,
Quaternion.LookRotation(Vector3.forward));
        this.sphereObject.SetActive(true);
        this.sphereObject.GetComponent<GraphNodeController>().setValues(vals);
    }

    /**
     * Delete function
     * Deletes the current graphNode
     */
    public void Delete()
    {

```

```

        valueList = new List<int>();
        GameObject.Destroy(sphereObject);
    }

    /**
     * addValue function
     * Used to add and show a value to the current node
     */
    public void addValue(int v)
    {
        this.valueList.Add(v);
        this.valueList.Sort();
        this.sphereObject.GetComponent<GraphNodeController>().setValues(new List<int> {v});
    }

    /**Getters and setters for the values and 3d object*/
    public void setValues(List<int> vals)
    {
        vals.Sort();
        this.valueList = new List<int>();
        this.valueList = vals;
        this.sphereObject.GetComponent<GraphNodeController>().setValues(vals);
    }
    public List<int> getValues()
    {
        this.valueList.Sort();
        return this.valueList;
    }
    public void set3DSphereObject(GameObject s)
    {
        this.sphereObject = s;
    }
    public GameObject get3DSphereObject()
    {
        return this.sphereObject;
    }

    /**
     * isEqual functions
     * Used to check if two value integer lists have equal amount of values
     * and the same ones.
     */
    public bool isEqual(List<int> valList2)
    {
        //ints1.All(ints2.Contains) && ints1.Count == ints2.Count
        if (this.valueList.All(valList2.Contains) && (this.valueList.Count == valList2.Count))
        {
            return true;
        }
        return false;
    }
    public bool isEqual(GraphNode graphNode2)
    {
        List<int> valList2 = graphNode2.getValues();
        //ints1.All(ints2.Contains) && ints1.Count == ints2.Count
        if (this.valueList.All(valList2.Contains) && (this.valueList.Count == valList2.Count))
        {
            return true;
        }
        return false;
    }
}

```

Edge.cs

```

using System.Collections.Generic;
using System.Linq;
using UnityEngine;

/**
 * Edge script object
 * The edge object is used to describe an edge connecting
 * two nodes on an undirected 3D graph.
 */
public class Edge {

    private GraphNode src;           // Source node
    private GraphNode dst;           // Destination node
    private Vector3 pos1;             // position of the CENTRE of the source node
    private Vector3 pos2;             // position of the CENTRE of the destination node
    public List<GameObject> edgeObjectList; // A list containing all the edges connecting the two
nodes together

    public int numberOfEdges; //Number of edges connecting node src to dst (DEFAULT IS 1)*/
    public float radius;      // Radius of the sphere of a node

    /*Edge constructor*/
    public Edge(GraphNode s, GraphNode d, int noOfEdges = 1)
    {
        this.src = s;
        this.dst = d;
        this.pos1 = s.get3DSphereObject().transform.position;
        this.pos2 = d.get3DSphereObject().transform.position;

        // Radius of the node is half its diameter
        this.radius = src.get3DSphereObject().transform.lossyScale.y / 2f;
        this.edgeObjectList = new List<GameObject>();
        this.numberOfEdges = noOfEdges;

        // Reset the edge positions
        ResetEdgePositions();
    }

    /*Used to delete the current edge object*/
    public void Delete()
    {
        src = null;
        dst = null;
        numberOfEdges = 0;
        DeleteAllEdges();
    }

    /*Used to delete all the 3D edges currently connecting the two nodes*/
    public void DeleteAllEdges()
    {
        // Remove all previous edges
        while(edgeObjectList.Count != 0)
        {
            GameObject.Destroy(edgeObjectList[0]);
            edgeObjectList[0] = null;
            edgeObjectList.RemoveAt(0);
        }
        //edgeObjectList.Clear();
        edgeObjectList = new List<GameObject>();
    }

    /*Sets the color of each edge to c*/
    public void SetColor(Color c)
    {
        for (int i=0; i<edgeObjectList.Count; i++)
        {
            edgeObjectList[i].GetComponent<Renderer>().material.color = c;
        }
    }
}

```

```

    }
}

/*Reset node positions based on their new position*/
public void ResetNodePositions()
{
    this.pos1 = src.get3DSphereObject().transform.position;
    this.pos2 = dst.get3DSphereObject().transform.position;
}

/**
 * ResetEdgeCount function
 * Reset the edge count between the nodes
 */
public void ResetEdgeCount()
{
    // Delete all current edges
    DeleteAllEdges();

    // Interpolate the edges between nodes
    InterpolateMultipleEdges();
}

/**
 * InterpolateMultipleEdges function
 * An important function. It is used to linearly instantiate
 * interpolated 3D lines between two nodes.
 */
public void InterpolateMultipleEdges()
{
    // Find the vector pointing from source(pos1) --> destination(pos2) node
    Vector3 v3 = pos2 - pos1;

    // By finding the tangent of that vector (by multiplying -1 the y-axis)
    // we are able to determining the tangent direction of the edges
    Vector3 tangent = new Vector3(-v3.y, v3.x, v3.z);

    // Normalize that direction
    tangent.Normalize();

    // Find the start and end point of the source node
    // based on the radius and direction of the tangent
    Vector3 startPos1 = pos1 + tangent * radius;
    Vector3 endPos1 = pos1 - tangent * radius;

    // Find the start and end point of the destination node
    Vector3 startPos2 = pos2 + tangent * radius;
    Vector3 endPos2 = pos2 - tangent * radius;

    // If we have only one edge, create only one
    if (numberOfEdges == 1)
    {
        CreateEdge(pos1, pos2);
        return;
    }
    else
    {
        // Else interpolate linearly between start and end positions
        // of the two nodes, in between edges with equal distance.
        for (int i = 0; i < numberOfEdges; i++)
        {
            Vector3 currP1 = Vector3.Lerp(startPos1, endPos1, (float)i / (float)numberOfEdges);
            Vector3 currP2 = Vector3.Lerp(startPos2, endPos2, (float)i / (float)numberOfEdges);
            CreateEdge(currP1, currP2);
        }
    }
}
}

```

```

public void ResetEdgePositions()
{
    ResetNodePositions();
    DeleteAllEdges();
    InterpolateMultipleEdges();
}

/**
 * AddEdge function
 * Used to add a 3D edge to the current edge
 */
public void AddEdge()
{
    this.numberOfEdges++;
    ResetEdgePositions();
    //ResetNumberOfEdges(1+numberOfEdges);
}

/**
 * CreateEdge function
 * A function that creates an edge between to specific points.
 */
public void CreateEdge(Vector3 p1, Vector3 p2)
{
    // Use the 3DLine already placed on our scene
    GameObject edge3DObject = GameObject.FindGameObjectWithTag("3DLine");

    // Instantiate the edge on the centre of the two points
    // connecting the two points together
    Vector3 v3 = p2 - p1;
    GameObject newEdge = GameObject.Instantiate(edge3DObject);
    newEdge.transform.position = p1 + (v3) / 2.0f;
    newEdge.transform.localScale = new Vector3(newEdge.transform.localScale.x, v3.magnitude /
2.0f, newEdge.transform.localScale.z);
    newEdge.transform.rotation = Quaternion.FromToRotation(Vector3.up, v3);
    newEdge.SetActive(true);
    edgeObjectList.Add(newEdge);

    //edge3DObject = null;
    newEdge = null;
}

/*Getters and setters for source and dest nodes*/
public void addSrcNode(int s)
{
    src.addValue(s);
}
public void addDstNode(int d)
{
    dst.addValue(d);
}
public void setSrcValues(List<int> s)
{
    s.Sort();
    this.src.setValues(s);
}
public void setDstValues(List<int> d)
{
    d.Sort();
    this.dst.setValues(d);
}
public List<int> getSrcValues()
{
    return this.src.getValues();
}
public List<int> getDstValues()

```

```

{
    return this.dst.getValues();
}
public GraphNode GetSourceNode()
{
    return this.src;
}
public GraphNode GetDestinationNode()
{
    return this.dst;
}
/*End of getters/setters*/

/**
 * isEqual function
 * Returns true if this edge is the same with another one
 * based on the node values.
 */
public bool isEqual(GraphNode from, GraphNode to)
{
    if(from == null || to == null || from.isEqual(to))
    {
        return false;
    }
    /*if ((this.src.isEqual(from.getValues()) && this.dst.isEqual(to.getValues())) ||
        (this.src.isEqual(to.getValues()) && this.dst.isEqual(from.getValues()))*/

    // Check if the two nodes are equal if src==from and dst==to OR src==to and dst==from
    if ( (this.src.isEqual(from) && this.dst.isEqual(to)) ||
        (this.src.isEqual(to) && this.dst.isEqual(from)) )
    {
        return true;
    }
    else
    {
        return false;
    }
}

/**
 * ChangeEdge function
 * Used to change the nodes of the current edge
 */
public Edge ChangeEdge(GraphNode newSrc, GraphNode newDst)
{
    if (newSrc == null || newDst == null || newSrc.isEqual(newDst))
    {
        return null;
    }
    this.src = newSrc;
    this.dst = newDst;
    ResetEdgePositions();
    return this;
}

/**
 * ToString override method
 * Used to print the edge in string form.
 */
public override string ToString()
{
    string srcString = "{" + string.Join(",", this.getSrcValues().Select(x =>
x.ToString()).ToArray()) + "}";
    string dstString = "{" + string.Join(",", this.getDstValues().Select(x =>
x.ToString()).ToArray()) + "}";
    return "Src->" + srcString + " Dst->" + dstString + " Links: " + numberOfEdges;
}

```

```
}
```

Undirected3DGraph.cs

```
using System.Collections.Generic;
using System.Linq; /*For SequenceEqual (to check equality of lists)*/
using System.Runtime.Serialization;
using UnityEngine;

/**
 * Undirected3DGraph script
 * The main undirected 3D graph object used for Karger's Contract
 * randomization algorithm
 */
[DataContract]
public class Undirected3DGraph{

    [DataMember]
    public List<GraphNode> nodeList; /*The list with all the nodes of the graph*/

    [DataMember]
    public List<Edge> edgeList; /*The list with all the edges*/

    /*Undirected 3D Graph constructors*/
    public Undirected3DGraph()
    {
        this.nodeList = new List<GraphNode>();
        this.edgeList = new List<Edge>();
    }
    public Undirected3DGraph(List<GraphNode> nl, List<Edge> el)
    {
        this.nodeList = nl;
        this.edgeList = el;
    }

    /**
     * Destroy function
     * Used to destroy the whole graph
     */
    public void Destroy()
    {
        foreach(GraphNode n in nodeList)
        {
            n.Delete();
        }
        nodeList = null;

        foreach(Edge e in edgeList)
        {
            e.Delete();
        }
        edgeList = null;
    }

    /**
     * RebuildFromDataFile function
     * Rebuilds the undirected 3d multigraph based on the
     * data from the graphData restored from a file.
     */
    public void RebuildFromDataFile(Und3DGraphData graphData)
    {
        // Destroy the current graph.
        Destroy();

        // Find the already placed 3D objects with their tags
    }
}
```

```

GameObject node_3d_sphere = GameObject.FindGameObjectWithTag("3DNode");
GameObject edge_3d_line = GameObject.FindGameObjectWithTag("3DLine");

this.nodeList = new List<GraphNode>();
this.edgeList = new List<Edge>();

// Rebuild nodes
for(int i=0; i<graphData.nodes.Count; i++)
{
    Vector3 currNodePos = graphData.nodePositions[i];
    List<int> currNodeVals = graphData.nodes[i].nestedList;
    GameObject new_node = GameObject.Instantiate(node_3d_sphere, currNodePos,
Quaternion.LookRotation(Vector3.forward));
    //new_node.GetComponent<GraphNodeController>().setValues(currNodeVals);
    new_node.SetActive(true);
    //addNode(new_node.GetComponent<GraphNodeController>().getValues(), new_node);
    GraphNode n = new GraphNode(currNodeVals, new_node);
    addNode(n);
    new_node = null; // Remove pointer
}

// Rebuild the edges
for(int i=0; i<graphData.srcNodes.Count; i++)
{
    int srcNodeIndex = graphData.srcNodes[i];
    int dstNodeIndex = graphData.dstNodes[i];
    int currEdgeNoOfEdges = graphData.noOfEdges[i];
    addEdge(this.nodeList[srcNodeIndex], this.nodeList[dstNodeIndex], currEdgeNoOfEdges);
}

// Remove pointers
node_3d_sphere = null;
edge_3d_line = null;
}

/*Getters and setters for Node and Edge List*/
public List<GraphNode> getNodeList()
{
    return this.nodeList;
}
public void SetNodeList(List<GraphNode> nl)
{
    this.nodeList = nl;
}
public List<Edge> getEdgeList()
{
    return this.edgeList;
}
public void SetEdgeList(List<Edge> el)
{
    this.edgeList = el;
}
/*End of getters and setters*/

/*Adding and removing a node*/
public void addNode(GraphNode n)
{
    nodeList.Add(n);
}
public void removeNode(GraphNode node)
{
    // First we remove all the edges connected with the node
    removeAllEdges(node);

    // Remove the node by removing it from the list
    // Deleting it (Destroying it) and settin it to null
    nodeList.Remove(node);
}

```

```

        node.Delete();
        node = null;
    }

    /**
     * addEdge functions
     * Add an edge to our graph
     */
    public void addEdge(GraphNode src, GraphNode dst)
    {
        this.edgeList.Add(new Edge(src, dst));
    }
    public void addEdge(GraphNode src, GraphNode dst, int noOfEdges)
    {
        this.edgeList.Add(new Edge(src, dst, noOfEdges));
    }

    /**
     * deleteEdge function
     * Deletes an edge src->dst
     */
    public bool deleteEdge(GraphNode src, GraphNode dst)
    {
        // Check all the edges in the list
        for (int i = 0; i < this.edgeList.Count; i++)
        {
            // Check if src -> dst or dst -> src (because we have an undirected graph)
            // if we have a match then destroy that edge and remove it from the list.
            if (edgeList[i].isEqual(src, dst))
            {
                edgeList[i].Delete();
                edgeList[i] = null;
                edgeList.Remove(edgeList[i]);
                //edgeList.RemoveAt(i);
                return true;
            }
        }
        return false;
    }

    /**
     * getNodeIndex function
     * USED WHEN SAVING to get the index of a specific node based on its values
     */
    public int getNodeIndex(List<int> vals)
    {
        vals.Sort();
        for (int i = 0; i < nodeList.Count; i++)
        {
            GraphNode curr_node = nodeList[i];
            if (curr_node.isEqual(vals))
            {
                return i;
            }
        }
        return -1;
    }

    /**
     * ResetAllEdgePositions function
     * Used to reset all the edge positions based on the
     * movement of nodes (Used for animation mainly)
     */
    public void ResetAllEdgePositions()
    {
        // Check all the edges in the list
        for (int i = 0; i < this.edgeList.Count; i++)
        {

```

```

        this.edgeList[i].ResetEdgePositions();
    }
}

/**
 * edgeExists function
 * Check if an edge exists based on a specific src and dst node
 */
public Edge edgeExists(GraphNode src, GraphNode dst)
{
    if (src == null || dst == null || src.isEqual(dst))
    {
        return null;
    }
    // Check all the edges in the list
    for (int i = 0; i < this.edgeList.Count; i++)
    {
        // Check if src -> dst or dst -> src (because we have an undirected graph)
        // if we have a match then destroy that edge and remove it from the list.
        if (edgeList[i].isEqual(src, dst))
        {
            return edgeList[i];
        }
    }
    return null;
}

/**
 * updateNode
 * Used to change the node values of a node to the new node values
 */
public void updateNode(GraphNode node, List<int> new_vals)
{
    //vals.Sort();
    new_vals.Sort();
    node.setValues(new_vals);
}

/**
 * FindCentrePointFromEdge function
 * Used to find the centre point between the source and dest
 * nodes of an edge.
 */
public Vector3 FindCentrePointFromEdge(int edge_no)
{
    // Find the centre point between the two nodes on the edge
    Vector3 posSrcNode =
edgeList[edge_no].GetSourceNode().get3DSphereObject().transform.position;
    Vector3 posDstNode =
edgeList[edge_no].GetDestinationNode().get3DSphereObject().transform.position;
    Vector3 centrePoint = (posSrcNode + posDstNode) / 2f;
    return centrePoint;
}

/**
 * moveEdges function
 * Used to move all the edges from the "from" node to any other
 * node to the "to" node
 */
public void moveEdges(GraphNode from, GraphNode to)
{
    if (from == null || to == null || from.isEqual(to))
    {
        return;
    }
    for (int i = 0; i < nodeList.Count; i++)
    {

```

```

        // If an edge exists from the from node to any other node
        // then add an edge from the from node to that one and delete its connection
        // with the to node
        if (edgeExists(from, nodeList[i]) != null)
        {
            addEdge(to, nodeList[i], edgeExists(from, nodeList[i]).numberOfEdges);
            deleteEdge(from, nodeList[i]);
        }
    }
    removeNode(from);
}

/**
 * removeAllEdges function
 * Used to remove all edges connected to a specific node
 */
public void removeAllEdges(GraphNode node)
{
    for (int i = 0; i < nodeList.Count; i++)
    {
        if (edgeExists(node, nodeList[i]) != null)
        {
            deleteEdge(node, nodeList[i]);
        }
    }
}

/**
 * Merge Edge function
 * This function is used to merge an edge by sending all the
 * values of the dst node to the src node and then removing the
 * src node thus merging the two connected nodes together!
 */
public bool mergeEdge(int edge_no)
{
    // Get the src and dst nodes of that edge
    GraphNode srcNode = edgeList[edge_no].GetSourceNode();
    GraphNode dstNode = edgeList[edge_no].GetDestinationNode();

    // Get the centre point of an edge
    Vector3 centrePoint = FindCentrePointFromEdge(edge_no);

    // Delete the current selected edge
    deleteEdge(srcNode, dstNode);

    // Merge the values of the two nodes into a new node!
    List<int> new_node_values = srcNode.getValues().ToList();
    for (int i = 0; i < dstNode.getValues().Count; i++)
    {
        new_node_values.Add(dstNode.getValues()[i]);
    }
    new_node_values.Sort();

    // Create a new node with the new values in the middle
    GraphNode newNode = new GraphNode(new_node_values, centrePoint);

    // Add it to the graph
    addNode(newNode);

    // Add an edge to all of the edges connected with the DST node from
    // all the dst nodes of all the edges from the SRC node
    // of the edge we want to merge
    for (int i = 0; i < nodeList.Count; i++)
    {
        // if edge existst SRC->X
        if (edgeExists(srcNode, nodeList[i]) != null)
        {

```

```

        // and edge exists X->DST
        if (edgeExists(nodeList[i], dstNode) != null)
        {
            // Add all the edges of SRC->X AND X->DST together = SUM
            // delete those edges and create a new one X->NEW_NODE with SUM edges
            int newNumberOfEdges = edgeExists(srcNode, nodeList[i]).numberOfEdges +
edgeExists(nodeList[i], dstNode).numberOfEdges;
            deleteEdge(nodeList[i], dstNode);
            deleteEdge(srcNode, nodeList[i]);
            addEdge(nodeList[i], newNode, newNumberOfEdges);
        }
    }

    // Move all edges from dst and src node to the New Node!
    moveEdges(dstNode, newNode);
    moveEdges(srcNode, newNode);

    // Remove possible mutual edges
    deleteEdge(dstNode, dstNode);
    deleteEdge(newNode, newNode);
    deleteEdge(srcNode, srcNode);
    srcNode = null;
    dstNode = null;
    newNode = null;
    return true;
}
}

```

ListWrapper.cs

```

using System.Collections.Generic;

/**
 * ListWrapper script object
 * Used to store a nested list (Serializable)
 */
[System.Serializable]
public class ListWrapper {
    public List<int> nestedList;
}

```

SaveSystem.cs

```

using System.IO;
using UnityEngine;
using System.Runtime.Serialization.Formatters.Binary;

/**
 * SaveSystem singleton script
 * Used to provide saving/loading features for the Undirected3DGraph
 * used in Karger's algorithm
 */
public static class SaveSystem {

    // The default file path where the graphs will be saved and loaded.
    public static string defaultPath = Application.persistentDataPath + "/Graphs/";

    /**
     * SaveUnd3DGraph
     * Saves the graph data to a file
     */
}

```

```

public static string SaveUnd3DGraph(Undirected3DGraph graph, string fileName)
{
    // If default directory does not exist, create it.
    if (!Directory.Exists(defaultPath))
    {
        Directory.CreateDirectory(defaultPath);
    }

    // ONLY WORKS IN THE UNITY EDITOR AND NOT DURING THE BUILD!!
    // HAD TO CREATE A NEW UI INTERFACE FOR SAVING.
    //string path = UnityEditor.EditorUtility.SaveFilePanel("Save Your Graph", defaultPath,
    "Graph-Name", "dat");

    if (!string.IsNullOrEmpty(fileName))
    {
        // Actually save the graph!
        // Display the path we are going to store the
        // save file on. (persistent directory) (it does not change)
        //fileName = Path.GetFileNameWithoutExtension(fileName);
        string filePath = defaultPath + "/" + fileName + ".dat";
        Debug.Log("Saving at... ." + filePath);

        // Stream the file with a File Stream. (Note that File.Create() 'Creates' or
        'Overwrites' a file.)
        FileStream file = File.Create(filePath);

        BinaryFormatter bf = new BinaryFormatter();

        // Create a new Und3DGraphData and save all the graph's data.
        Und3DGraphData data = new Und3DGraphData(graph);

        bf.Serialize(file, data);

        file.Close();
    }
    return fileName;
}

/**
 * Delete3DGraph
 * Deletes a specific graph file if it exists
 */
public static void Delete3DGraph(string fileName)
{
    string filePath = defaultPath + fileName + ".dat";
    if (File.Exists(filePath))
    {
        File.Delete(filePath);
    }
    else
    {
        Debug.LogError("File " + filePath + " does not exist!");
    }
}

/**
 * LoadUnd3DGraph
 * Loads all the graph data of a specific file
 */
public static Und3DGraphData LoadUnd3DGraph(string fileName)
{
    string filePath = defaultPath + fileName + ".dat";
    if (File.Exists(filePath))
    {
        // Stream the file with a File Stream. (Note that File.Create() 'Creates' or
        'Overwrites' a file.)
        FileStream file = new FileStream(filePath, FileMode.Open);

```

```

        BinaryFormatter bf = new BinaryFormatter();

        // Deserialize file to the data object.
        Und3DGraphData readData = bf.Deserialize(file) as Und3DGraphData;

        file.Close();

        Debug.Log("Nodes: " + readData.nodes.Count);
        Debug.Log("Edges: " + readData.srcNodes.Count);

        return readData;
    }
    else
    {
        Debug.LogError("File "+filePath+" does not exist!");
        return null;
    }
}
}

```

Und3DGraphData.cs

```

using System.Collections.Generic;
using UnityEngine;

/**
 * Und3DGraphData script
 * It contains a single Undirected3DGraph's data to be stored in a
 * file using the Serializable function.
 */
[System.Serializable]
public class Und3DGraphData{

    [SerializeField]
    public List<ListWrapper> nodes;
    [SerializeField]
    public List<SerializableVector3> nodePositions;
    [SerializeField]
    public List<int> srcNodes;
    [SerializeField]
    public List<int> dstNodes;
    [SerializeField]
    public List<int> noOfEdges;

    public Und3DGraphData(Undirected3DGraph graph)
    {
        nodes = new List<ListWrapper>();
        nodePositions = new List<SerializableVector3>();
        srcNodes = new List<int>();
        dstNodes = new List<int>();
        noOfEdges = new List<int>();

        List<Edge> edgeList = graph.getEdgeList();
        List<GraphNode> nodeList = graph.getNodeList();

        // Add each node's values and positions to the nodes list
        // to save their state.
        for(int i=0; i<nodeList.Count; i++)
        {
            ListWrapper lw = new ListWrapper();
            lw.nestedList = nodeList[i].getValues();
            nodes.Add(lw);

            Vector3 nodePos = nodeList[i].get3DSphereObject().transform.position;

```

```

        SerializableVector3 serVector = new SerializableVector3(nodePos.x, nodePos.y,
nodePos.z);

        nodePositions.Add(serVector);
    }

    // Save the state of each edge
    for(int i=0; i<edgeList.Count; i++)
    {
        Edge currEdge = edgeList[i];
        int srcIndex = graph.getNodeIndex(currEdge.getSrcValues());
        int dstIndex = graph.getNodeIndex(currEdge.getDstValues());

        srcNodes.Add(srcIndex);
        dstNodes.Add(dstIndex);

        noOfEdges.Add(currEdge.numberOfEdges);

        currEdge = null;
    }

    // Remove the pointers from the 3d graph
    edgeList = null;
    nodeList = null;
}
}

```

CreateGraphUIController.cs

```

using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

/**
 * CreateGraphUIController script
 * Used to control all the user mouse abilities for graph creation
 * and visualizes the Karger's contract algorithm (Randomization Algorithm)
 */
public class CreateGraphUIController : MonoBehaviour
{
    // The initial filename and maximum number of nodes in a graph
    public const string FILENAME = "Graph-0";
    public const int MAX_NODES = 30;

    // A boolean array used to remember the nodes that have been instantiated
    // to keep an order of the numbering
    public bool[] hasNodeArray;

    // Variables used when the user creates a new edge
    public GameObject line_3d_edge;
    private Vector3 pos1;
    private Vector3 pos2;
    private bool is_line;
    private GameObject new_3d_line;
    private Transform initial_3d_edge_transform;

    // The current node being hovered over
    private GraphNode current_hovered_node;
    // The current selected node
    private GraphNode current_selected_node;
    private GraphNode previous_selected_node;

    // From and To node when creating a line
    private GraphNode from_node;

```

```

private GraphNode to_node;
private List<int> from_val;
private List<int> to_val;

// Our graph
private Undirected3DGraph und3dGraph = new Undirected3DGraph();

// The bounds where the user is able to instantiate nodes
private Rect bounds;

// Disabled flag is true when you are not able to
// create and edit the graph
private bool disabled = false;
private bool isMoveEdges = false;

// UI elements
public TextMeshProUGUI fileNameText;
public RectTransform boundOutline;
public TMP_InputField inputFieldSaveFile;
public GameObject instructionsButton;
public GameObject deleteButton;
// Used to set the speed of the algorithm
public float speed = 2f;

// Use this for initialization
void Start()
{
    // Set node array's size to max nodes
    hasNodeArray = new bool[MAX_NODES];

    // Find the two gameobjects needed for instantiating a graph. (node and edge)
    line_3d_edge = GameObject.FindGameObjectWithTag("3DLine");

    // The current 3D line being created
    new_3d_line = Instantiate(line_3d_edge, line_3d_edge.transform.position,
line_3d_edge.transform.rotation);
}

// Update is called once per frame
void Update()
{
    // If disabled and move edges is enabled, dont identify mouse actions
    // just move edges for animations
    if (disabled && isMoveEdges)
    {
        und3dGraph.ResetAllEdgePositions();
    }
    else if (!disabled)
    {
        // Must update the size of the bounds for fullscreen adjustment etc
        Vector2 boundSize = new Vector2(Screen.width, Screen.height);
        Vector2 boundPos = Camera.main.ScreenToWorldPoint(boundSize);
        bounds = new Rect(boundPos.x + boundSize.x / 19f, boundPos.y + boundSize.y / 7f,
boundSize.x / 1.10f, boundSize.y / 1.35f);

        // DRAW THE RECT ON SCENE VIEW FOR DEBUG PURPOSES
        Debug.DrawLine(new Vector2(bounds.xMin, bounds.yMin), new Vector2(bounds.xMin,
bounds.yMax));
        Debug.DrawLine(new Vector2(bounds.xMin, bounds.yMin), new Vector2(bounds.xMax,
bounds.yMin));
        Debug.DrawLine(new Vector2(bounds.xMax, bounds.yMax), new Vector2(bounds.xMin,
bounds.yMax));
        Debug.DrawLine(new Vector2(bounds.xMax, bounds.yMax), new Vector2(bounds.xMax,
bounds.yMin));

        // Find the current mouse position
        Vector3 mouse_pos = new Vector3(Input.mousePosition.x, Input.mousePosition.y, 10f);

```

```

// The current selected node
// THE USER IS ONLY ABLE TO SELECT ONLY 1 NODE
current_selected_node = getSelectedNode();

// The current node being hovered over
// (only 1 node is hovered over at a time)
current_hovered_node = getHoveredNode();

// Check if mouse gets pressed within the desired bounds!
// Only there, will the user be able to instantiate and create the graph.
if (Input.GetMouseButtonDown(0) && bounds.Contains(Input.mousePosition))
{
    // When the left mouse gets pressed (not held down)
    // ,when no other node is hovered over, when we are
    // not creating a line and when we are within maximum bounds
    // then we can instantiate a new node
    if (current_hovered_node == null && !is_line && und3dGraph.getNodeList().Count <
MAX_NODES)// && !Input.GetMouseButton(0))
    {
        Debug.Log("Pressed left mouse click");

        // Check if the node number has been instantiated again
        // if not instantiate that number
        int nodeValue = 0;
        for (int i = 0; i < hasNodeArray.Length; i++)
        {
            if (!hasNodeArray[i])
            {
                hasNodeArray[i] = true;
                nodeValue = i + 1;
                break;
            }
        }
        GraphNode newNode = new GraphNode(new List<int>() { nodeValue },
Camera.main.ScreenToWorldPoint(mouse_pos));

        und3dGraph.addNode(newNode);
        newNode = null;
    }

    if (current_selected_node != null)
    {
        und3dGraph.ResetAllEdgePositions();

        // Move the selected node if it is selected, is hovered over and dragged!
        if (Input.GetMouseButton(0)
            && current_hovered_node != null
            && current_hovered_node.Equals(current_selected_node))
        {
            Vector3 newPos = new Vector3(Input.mousePosition.x, Input.mousePosition.y, 10f);
            newPos = Camera.main.ScreenToWorldPoint(newPos);
            current_selected_node.get3DSphereObject().transform.position = newPos;
        }
    }
    else
    {
        // THE FOLLOWING CODE IS EXECUTED ON THE INSTANTIATED NODES
        // HENCE WE DON'T NEED TO CHECK IF WE ARE WITHIN BOUNDS
        // INSTANTIATING A LINE BETWEEN NODES
        if (Input.GetMouseButtonDown(0) && current_hovered_node != null)
        {
            // Current hovered node is the start of the edge
            from_node = current_hovered_node;
            from_val =
from_node.get3DSphereObject().GetComponent<GraphNodeController>().getValues();

```

```

        pos1 = new Vector3(Input.mousePosition.x, Input.mousePosition.y, 10f);
        pos1 = Camera.main.ScreenToWorldPoint(pos1);
        pos2 = pos1;
        is_line = true;

        // Instantiate new 3d line
        Destroy(new_3d_line);
        new_3d_line = null;
        new_3d_line = Instantiate(line_3d_edge, line_3d_edge.transform.position,
line_3d_edge.transform.rotation);
    }

    // Translate the mouse position to world point
    // to be able to show the end of the 3D Line on the mouse pointer
    // where the user wants to drag it
    if (Input.GetMouseButton(0) && is_line)
    {
        pos2 = new Vector3(Input.mousePosition.x, Input.mousePosition.y, 10f);
        pos2 = Camera.main.ScreenToWorldPoint(pos2);
    }

    // If the left mouse gets released while it is hovering over a node and
    // it's not the same node from which we started creating the line then
    // we can finish the edge!
    if (Input.GetMouseButtonUp(0) && is_line)
    {
        // Create the edge if it does not exist
        // If we reach a different node and is hovered that's at the end of the edge
        if (current_hovered_node != null && current_hovered_node != from_node)
        {
            to_node = current_hovered_node;
            to_val =
to_node.get3DSphereObject().GetComponent<GraphNodeController>().getValues();
            is_line = false;
            if (und3dGraph.edgeExists(from_node, to_node) == null)
            {
                und3dGraph.addEdge(from_node, to_node, 1);
                Destroy(new_3d_line);
                new_3d_line = null;
            }
            else
            {
                Destroy(new_3d_line);
                new_3d_line = null;
            }
        }
        else
        {
            Destroy(new_3d_line);
            new_3d_line = null;
        }
    }

    // If the user releases the left mouse button and there is no sphere under it
    // then we cancel the line(edge) creation
    if (Input.GetMouseButtonUp(0) && is_line && current_hovered_node == null)
    {
        is_line = false;
    }

    // Calculate the 3D Line gameObject based on the
    // pos1 -> pos2 positions!
    if (pos2 != pos1 && is_line && new_3d_line != null)
    {
        Vector3 v3 = pos2 - pos1;
        new_3d_line.transform.position = pos1 + (v3) / 2.0f;
    }

```

```

        new_3d_line.transform.localScale = new
Vector3(new_3d_line.transform.localScale.x, v3.magnitude / 2.0f,
new_3d_line.transform.localScale.z);
        new_3d_line.transform.rotation = Quaternion.FromToRotation(Vector3.up, v3);
    }
}

// Remove hovered node if Right Clicked
if (Input.GetMouseButtonDown(1) && current_hovered_node != null)
{
    removeHoveredNode();
}

// Remove hovered edge if right clicked
Edge current_hovered_edge = getHoveredEdge();
if (Input.GetMouseButtonDown(1) && current_hovered_edge != null)
{
    und3dGraph.deleteEdge(current_hovered_edge.GetSourceNode(),
current_hovered_edge.GetDestinationNode());
    current_hovered_edge.Delete();
}
current_hovered_edge = null;
}
}

/**
 * removeHoveredNode function
 * Used to delete and remove the current hovered
 * node (using right click)
 */
private bool removeHoveredNode()
{
    for (int i = 0; i < und3dGraph.getNodeList().Count; i++)
    {
        if
(und3dGraph.getNodeList()[i].get3DSphereObject().GetComponent<GraphNodeController>().is_hovered)
        {
            GraphNode n = und3dGraph.getNodeList()[i];
            hasNodeArray[n.getValues()[0]-1] = false;
            und3dGraph.getNodeList().Remove(und3dGraph.getNodeList()[i]);
            und3dGraph.removeAllEdges(n);
            n.Delete();
            n = null;
            return true;
        }
    }
    return false;
}

/**
 * getHoveredNode function
 * Used to get the current hovered node
 * the user's mouse is on.
 */
private GraphNode getHoveredNode()
{
    for (int i = 0; i < und3dGraph.getNodeList().Count; i++)
    {
        if
(und3dGraph.getNodeList()[i].get3DSphereObject().GetComponent<GraphNodeController>().is_hovered)
        {
            return und3dGraph.getNodeList()[i];
        }
    }
    return null;
}
}

```

```

/**
 * getHoveredEdge function
 * Used to get the current hovered edge
 * the user's mouse is on.
 */
private Edge getHoveredEdge()
{
    for(int i=0; i<und3dGraph.getEdgeList().Count; i++)
    {
        for(int j=0; j< und3dGraph.getEdgeList()[i].numberOfEdges; j++)
        {
            if
(und3dGraph.getEdgeList()[i].edgeObjectList[j].GetComponent<GraphLineController>().is_hovered)
            {
                return und3dGraph.getEdgeList()[i];
            }
        }
    }
    return null;
}

/**
 * getSelectedNode function
 * The get selected Node function is used to get the
 * currently selected node and unselect the previous one.
 */
private GraphNode getSelectedNode()
{
    //previous_selected_node = current_selected_node;
    GraphNode sel_node = null;
    int countSel = 0;
    for (int i = 0; i < und3dGraph.getNodeList().Count; i++)
    {
        // If a selected node is found, disable every othe one.
        if
(und3dGraph.getNodeList()[i].get3DSphereObject().GetComponent<GraphNodeController>().is_selected)
        {
            // Get only the first selected node and set the rest of them as unselected.
            sel_node = und3dGraph.getNodeList()[i];
            countSel++;

            if (previous_selected_node != null && countSel == 2)
            {
                previous_selected_node.get3DSphereObject().GetComponent<GraphNodeController>().resetNode();
            }
        }
    }
    if (countSel == 0)
    {
        previous_selected_node = null;
    }
    else
    {
        previous_selected_node = sel_node;
    }
    return sel_node;
}

/**
 * When the back button is pressed on the UI, it goes back to
 * the main start screen
 */
public void BackButton()
{

```

```

        SceneManager.LoadScene("MainUI");
    }
    /**
     * setSpeed function
     * Used to set the speed of the algorithm via a UI slider
     */
    public void setSpeed(float s)
    {
        this.speed = s;
    }
    public void SetDisabled(bool isDis)
    {
        this.disabled = isDis;
    }

    /**
     * RunMinCut
     * Used to run min-cut contract algorithm
     */
    public void RunMinCut()
    {
        // Run CONTRACT ALGORITHM
        StartCoroutine("contractAlgorithm");
        disabled = true;
        //graph.printGraph();
    }

    /**
     * DestroyGraph function
     * Used to remove all the graph that has been created
     * (when the back button is pressed in the graph creator UI)
     */
    public void DestroyGraph()
    {
        if (und3dGraph != null)
        {
            disabled = true;
            und3dGraph.Destroy();
            und3dGraph = new Undirected3DGraph();
            for (int i = 0; i < hasNodeArray.Length; i++)
                hasNodeArray[i] = false;
        }
        else
        {
            disabled = false;
        }
        fileNameText.text = "";
    }

    /**
     * Karger's Contract algorithm (Algorithmos Sirriknosis) coroutine
     *
     */
    public IEnumerator contractAlgorithm()
    {
        int iterationNo = 0;
        disabled = true;
        while (und3dGraph.getNodeList().Count > 2)
        {
            Debug.Log("Iteration " + iterationNo + " -> Nodes: " +
und3dGraph.getNodeList().Count.ToString());

            // *** GENERATE a random seed based on the computer's clock
            // and choose uniformly at random an edge from the graph
            Random.seed = (int)System.DateTime.Now.Ticks;
            int random_edge_no = (int)(Random.value * und3dGraph.getEdgeList().Count - 1);

```

```

        und3dGraph.getEdgeList()[random_edge_no].SetColor(Color.red);
        yield return new WaitForSeconds(speed);

        // Animate two Nodes merging with each other using BallMover Component
        GameObject node1 =
und3dGraph.getEdgeList()[random_edge_no].GetSourceNode().get3DSphereObject();
        GameObject node2 =
und3dGraph.getEdgeList()[random_edge_no].GetDestinationNode().get3DSphereObject();
        Vector3 centrePoint = und3dGraph.FindCentrePointFromEdge(random_edge_no);
        node1.GetComponent<BallMover>().SetTarget(centrePoint);
        node2.GetComponent<BallMover>().SetTarget(centrePoint);
        isMoveEdges = true;
        node1.GetComponent<BallMover>().active = true;
        node2.GetComponent<BallMover>().active = true;
        yield return new WaitUntil(() => !node1.GetComponent<BallMover>().active &&
!node2.GetComponent<BallMover>().active);
        isMoveEdges = false;
        node1 = null;
        node2 = null;

        und3dGraph.mergeEdge(random_edge_no);
        und3dGraph.ResetAllEdgePositions();

        yield return new WaitForSeconds(speed);

        iterationNo++;
    }
    yield return null;
}

/**
 * SaveGraph function
 * Used to save the created graph to a desired file
 */
public void SaveGraph()
{
    // Get file name from input text field
    string fileName = inputFieldSaveFile.text;

    // Save that file
    string savedFileName = SaveSystem.SaveUnd3DGraph(und3dGraph, fileName);
}

/**
 * LoadGraph funct
 * Used to load an already created graph via a filename
 */
public void LoadGraph(string fileName)
{
    disabled = true;
    Und3DGraphData graphData = SaveSystem.LoadUnd3DGraph(fileName);
    fileNameText.text = fileName;
    instructionsButton.SetActive(false);
    deleteButton.SetActive(true);
    und3dGraph = new Undirected3DGraph();
    und3dGraph.RebuildFromDataFile(graphData);
}

/**
 * DeleteGraph function
 * Used to delete the current graph file.
 */
public void DeleteGraph()
{
    string file = fileNameText.text;
    DestroyGraph();
}

```

```
        SaveSystem.Delete3DGraph(file);  
        instructionsButton.SetActive(true);  
        deleteButton.SetActive(false);  
    }  
}
```