Ατομική Διπλωματική Εργασία

ΠΡΟΣΟΜΟΙΩΣΗ ΣΥΜΠΕΡΙΦΟΡΩΝ ΣΜΗΝΩΝ ΑΠΟ ΠΟΥΛΙΑ

Μηνάς Μηνά

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2014

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΣΟΜΟΙΩΣΗ ΣΥΜΠΕΡΙΦΟΡΩΝ ΣΜΗΝΩΝ ΑΠΟ ΠΟΥΛΙΑ

Μηνάς Μηνά

Επιβλέπων Καθηγητής
Δρ. Γιώργος Χρυσάνθου

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

Μάιος 2014

Ευχαριστίες

Θα ήθελα πρώτα απ' όλα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα καθηγητή μου, Δρ. Γιώργο Χρυσάνθου, του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου, ο οποίος με βοήθησε και με καθοδόγησε κατά την ανάπτυξη και ολοκλήρωση της διπλωματικής μου εργασίας.

Θα ήθελα επίσης να ευχαριστήσω τους διδακτορικούς φοιτητές του DMSL οι οποίοι με βοήθησαν προτείνοντας μου διάφορους αλγορίθμους για προβλήματα που προέκυψαν κατά την υλοποίηση.

Τέλος, ευχαριστώ τους συμφοιτητές μου που μου συμπαραστάθηκαν και με βοήθησαν δίνοντας μου ιδέες για το πώς να προχωρήσω.

Περίληψη

Στόχος της παρούσας ατομικής διπλωματικής εργασίας είναι η δημιουργία προσομοίωσης της κίνησης σμηνών από πουλιά και η παρατήρηση της αλλαγής της συμπεριφοράς τους με την αλλαγή των παραμέτρων που την ορίζουν.
Επίσης, η υλοποίηση λειτουργίας αναπαράστασης συγκεκριμένων σχημάτων από πουλιά, καθώς και ένα απλό παιχνίδι που κάνει χρήση των πιο πάνω ιδεών.

Για την υλοποίηση του συστήματος έγινε μελέτη και υλοποίηση διάφορων αλγορίθμων που έχουν να κάνουν με προσομοίωση συμπεριφορών ομάδων από οντότητες (σμήνη πουλιών στη συγκεκριμένη περίπτωση).

Η υλοποίηση έγινε στην πλατφόρμα Unity3D[1]. Η πλατφόρμα αυτή επιλέχθηκε λόγω των ευκολιών που προσφέρει, π.χ συντάκτης σκηνής (3D scene editor), εργαλείο δημιουργίας τρισδιάστατου κόσμου (terrain editor), εύκολη ενσωμάτωση κανόνων φυσικής (physics). Τέλος, έγινε χρήση της συσκευής Leap Motion[2] ως προαιρετικός τρόπος ελέγχου στο σύστημα.

Περιεχόμενα

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Πίνακας Αλγορίθμων

Κεφάλαιο 1

Εισαγωγή

Πίνακας Περιεχομένων

1.1 Γενικά

Οι ηλεκτρονικοί υπολογιστές αποτελούν αναμφισβήτητα πλέον αναπόσπαστο κομμάτι της καθημερινότητας μας. Τις τελευταίες δύο δεκαετίες έχει αναπτυχθεί ραγδαία ο τομέας των γραφικών υπολογιστών, με αποτέλεσματα η κατασκευή κινηματογραφικών ταινιών καθώς και η δημιουργία ηλεκτρονικών παιχνιδιών να στηρίζονται σε μεγάλο βαθμό σε αυτόν.

Ένας τομέας ενδιαφέροντος στο πλαίσιο του Computer Animation είναι ο έλεγχος όλων των ειδών κίνησης. Ένας από τους στόχους του είναι η εφεύρεση νέων μορφών αφηρημένης κίνησης ή η επανάληψη (με ή χωρίς παραλλαγές) κινήσεων που υπάρχουν στον πραγματικό κόσμο όπως η προσομοίωση της κίνησης ομάδων οντοτήτων (π.χ πουλιών, ψαριών).

Οι τεχνικές που χρησιμοποιούνται για την εφεύρεση των εν λόγω μορφών κίνησης έχουν πολλές εφαρμογές στη βιομηχανία του θεάματος (διαφημίσεις, κινηματογράφος), για το λόγο ότι τεχνικές που αφορούν ομάδες από animations έχουν εφαρμοστεί εκεί πρόσφατα

για τη μείωση του κόστους παραγωγής ή για τη δημιουργία σεναρίων που δεν υπάρχουν στον πραγματικό κόσμο. Επίσης, μπορούν να χρησιμοποιηθούν στη βιομηχανία ηλεκτρονικών παιχνιδιών για την αναπαράσταση ομάδων ζώων του κόσμου του παιχνιδιού.

1.2 Σμήνη, κοπάδια, αγέλες

Ένα σμήνος πουλιών μπορεί να παρουσιάσει επίπεδα συντονισμού και συγχρονισμού που συχνά μας αφήνουν έκπληκτους με την κομψότητα τους. Τα σμήνη και οι σχετικές συμπεριφορές συγχρονισμένων ομάδων είναι και ωραίο να τις βλέπεις καθώς και ενδιαφέρον να τις σκέφτεσαι και να τις μελετάς. Ένα σμήνος παρουσιάζει επίσης πολλές αντιθέσεις. Απαρτίζεται από διακριτά πουλιά, αλλά η συνολική κίνηση φαίνεται ομαλή. Είναι απλή έννοια αλλά οπτικά πολύπλοκη – φαίνεται τυχαία κι όμως είναι υπέροχα συγχρονισμένη.

Ίσως το πιο αινιγματικό είναι η ισχυρή εντύπωση του κεντρικού ελέγχου. Ωστόσο, όλα τα στοιχεία δείχνουν ότι η κίνηση του σμήνους πρέπει να είναι απλώς το συνολικό αποτέλεσμα των ενεργειών των μεμονωμένων ζώων. Το καθένα ενεργεί αποκλειστικά βάσει της δικής του τοπικής αντίληψης του κόσμου.

1.3 Στόχος

Στόχος της παρούσας ατομικής διπλωματικής εργασίας ήταν η δημιουργία προσομοίωσης της κίνησης σμηνών από πουλιά και η παρατήρηση της αλλαγής της συμπεριφοράς τους με την αλλαγή των παραμέτρων που την ορίζουν.
Επίσης, υλοποιήθηκε λειτουργία αναπαράστασης συγκεκριμένων σχημάτων από πουλιά, καθώς και ένα απλό παιχνίδι που κάνει χρήση των πιο πάνω ιδεών.

Σχήμα 1.1: Σμήνος Πουλιών

Κεφάλαιο 2

Σχετική Εργασία

Πίνακας Περιεχομένων

2.1 Ορισμοί

Αυτόνομος χαρακτήρας είναι είδος αυτόνομου (agent) πράκτορα που προορίζεται για χρήση σε computer animation και διαδραστικά μέσα, όπως ηλεκτρονικά παιχνίδια και εικονική πραγματικότητα. Οι πράκτορες αυτοί έχουν την ικανότητα να αυτοσχεδιάζουν τις ενέργειες τους.

Αυτό έρχεται σε αντίθεση τόσο σε ένα χαρακτήρα σε μια ταινία κινουμένων σχεδίων, του οποίου οι ενέργειες είναι καθορισμένες εκ των προτέρων, και σε ένα "είδωλο" *(avatar)* σε ένα παιχνίδι ή εφαρμογή εικονικής πραγματικότητας, των οποίων οι δράσεις κατευθύνονται σε πραγματικό χρόνο από ανθρώπινο παράγοντα.

Στα παιχνίδια, οι αυτόνομοι χαρακτήρες μερικές φορές ονομάζονται non-player characters, δηλαδή χαρακτήρες που δεν ελέγχονται από τον χρήστη. Επίσης ονομάζονται και *boids*. Για λόγους συντομογραφίας θα αναφέρονται ως "χαρακτήρες".

2.2 Συμπεριφορές Κατεύθυνσης

Οι *Συμπεριφορές κατεύθυνσης (steering behaviors)* είναι συμπεριφορές που έχουν ως στόχο να βοηθήσουν αυτόνομους χαρακτήρες να κινηθούν με ρεαλιστικό τρόπο, με τη χρήση απλών δυνάμεων που συνδυάζονται για να αποδόσουν ρεαλιστική πλοήγηση μέσα στο περιβάλλον των χαρακτήρων.

Δε βασίζονται σε πολύπλοκες στρατηγικές που αφορούν το σχεδιασμό διαδρομής ή καθολικούς υπολογισμούς (που λαμβάνουν υπόψη όλο τον εικονικό κόσμο/περιβάλλον), αλλά αντ' αυτού χρησιμοποιούν τοπικές πληροφορίες, όπως τις δυνάμεις των γειτόνων.

Αυτό τις καθιστά απλό στην κατανόηση και υπολοποίηση, αλλά εξακολουθούν να είναι σε θέση να παράγουν πολύπλοκα σχήματα κίνησης.

2.2.1 Seek

Η συμπεριφορά Seek παίρνει ως είσοδο δύο σημεία *x*, *y* και υπολογίζει το διάνυσμα $\underline{u}$ που οδηγεί από τη *x* στη *y*.

Algorithm : Seek

Input :

- θέση του χαρακτήρα *x* [Vector3]
- θέση στόχου *y* [Vector3]
- μέγιστη επιτάχυνση *max_accel* [float]

Output : $\underline{u}$ [Vector3] – διάνυσμα που αναπαριστά την ταχύτητα που πρέπει να ακολουθήσει ο χαρακτήρας για να φτάσει στο στόχο (*y*)

```
1:    procedure seek(x, y, max_accel)
2:        u = y - x
3:        u = normalize(u) * max_accel
4:    end procedure
```

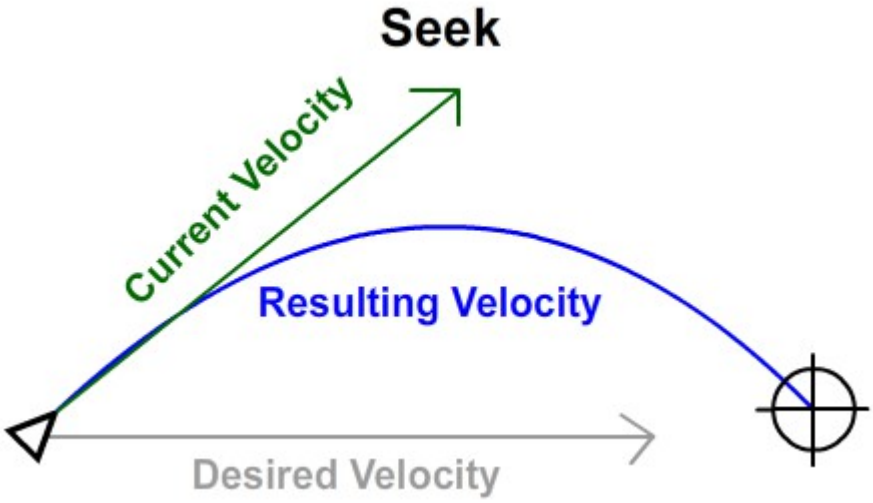
Αλγόριθμος 2.1: Συμπεριφορά Seek

Ο αλγόριθμος αρχικά υπολογίζει το διάνυσμα u αφαιρόντας το *x* από το *y*. Στη συνέχεια κανονικοποιεί το $\underline{u}$ και το πολλαπλασιάζει με *max_accel* ώστε το μήκος του να ισούται με *max_accel*, και το επιστρέφει. Αυτό έχει ως αποτέλεσμα ο χαρακτήρας να κινηθεί με τη μέγιστη επιτάχυνση προς το σημείο *y*.

Το $\underline{u}$ προστίθεται στην τρέχουσα ταχύτητα του χαρακτήρα με αποτέλεσμα αυτός σταδιακά να κινηθεί προς το στόχο. Εάν ο στόχος μετακινείται, ο χαρακτήρας θα τον ακολουθήσει.

Χρονική πολυπλοκότητα: O(1)

Χωρική πολυπλοκότητα: O(1)



Σχήμα 2.1: Με την εφαρμογή τη συμπεριφοράς αυτής σε κάθε frame, ο χαρακτήρας ακολουθεί τροχιά από την αρχική του θέση προς τη θέση του στόχου

2.2.2 Flee

Η συμπεριφορά Flee παίρνει ως είσοδο δύο σημεία *x*, *y* και υπολογίζει το διάνυσμα *u* που οδηγεί από τη *y* στη *x*. Ουσιαστικά παράγει το αντίθετο αποτέλεσμα από τη *Seek,* δηλαδή τείνει να κινήσει τον χαρακτήρα ώστε να αποφύγει το σημείο *y*.

Ο αλγόριθμος αρχικά υπολογίζει το διάνυσμα u αφαιρόντας το *y* από το *x*. Στη συνέχεια κανονικοποιεί το *u* και το πολλαπλασιάζει με *u_max* ώστε το μήκος του να ισούται με *u_max*, και το επιστρέφει. Αυτό έχει ως αποτέλεσμα ο χαρακτήρας να κινηθεί με τέτοιο τρόπο ώστε να αποφύγει το σημείο y, μέγιστη επιτάχυνση.

Το *u* προστίθεται στην τρέχουσα ταχύτητα του χαρακτήρα με αποτέλεσμα αυτός σταδιακά να κινηθεί μακριά από σημείο *y*. Εάν ο στόχος μετακινείται, ο χαρακτήρας θα τον αποφεύγει.

Algorithm : Flee

Input :

- θέση του χαρακτήρα *x* [Vector3]
- θέση στόχου *y* [Vector3]
- μέγιστη επιτάχυνση *max_accel* [float]

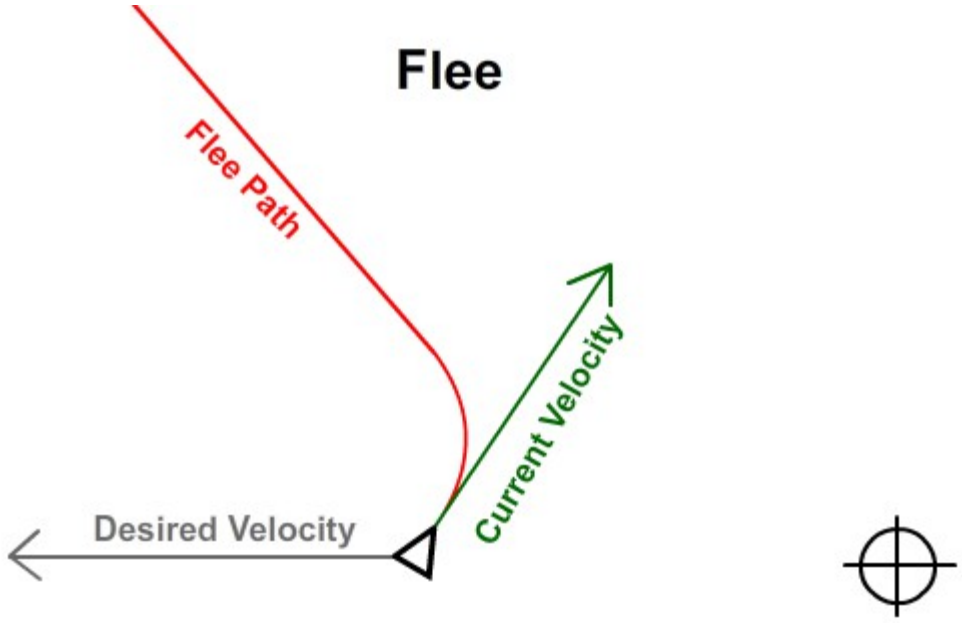
Output : $\underline{u}$ [Vector3] – διάνυσμα που αναπαριστά την ταχύτητα που πρέπει να ακολουθήσει ο χαρακτήρας για να απομακρυνθεί από τη σημείο *y*

```
1:    procedure flee(x, y, max_accel)
2:        u = x - y
3:        u = normalize(u) * max_accel
4:    end procedure
```

Αλγόριθμος 2.2: Συμπεριφορά Flee

Χρονική πολυπλοκότητα: O(1)

Χωρική πολυπλοκότητα: O(1)



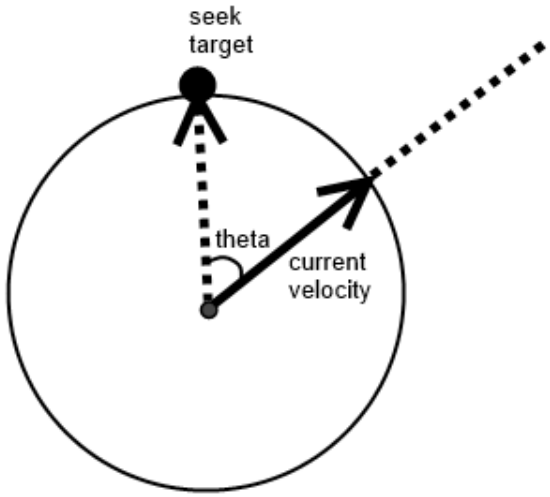
Σχήμα 2.2: Με την εφαρμογή τη συμπεριφοράς αυτής σε κάθε frame, ο χαρακτήρας προσπαθεί να κινηθεί με τέτοιο τρόπο ώστε να "ξεφύγει" από κάποιο σημείο (το οποίο μπορεί να είναι άλλος χαρακτήρας)

2.2.3 Wander

Πολλές φορές είναι επιθυμητό κάποιος χαρακτήρας να περιφέρεται μέσα στον κόσμο / περιβάλλον. Συνήθως τέτοιοι χαρακτήρες απλά περιμένουν να συμβεί κάποιο γεγονός (π.χ να δουν κάτι, κάτι να πλησιάσει). Όταν ο χρήστης βλέπει αυτή τη συμπεριφορά πρέπει η ικανότητα του χαρακτήρα να περιφέρεται στο χώρο να είναι οπτικά ευχάριστη και αρκετά ρεαλιστική.

Η συμπεριφορά αυτή έχει ως στόχο να παράξει ρεαλιστική, "ανέμελη" κίνηση ενός χαρακτήρα.

Βασίζεται στη θεώρηση μιας νοητής σφαίρας (στις 3 διαστάσεις) γύρω από τον χαρακτήρα, πάνω στον οποίο περιστρέφεται ένα σημείο target. Το σημείο αυτό δίνεται ως σημείο στόχου στη συμπεριφορά Seek, έτσι ώστε ο χαρακτήρας να κινηθεί προς αυτό.



Σχήμα 2.3: Συμπεριφορά Wander

Algorithm : Wander

Input :

- θέση του χαρακτήρα *x* [Vector3]
- Ακτίνα *radius* [float], που ισούται με την ακτίνα του κύκλου γύρω από τη θέση του χαρακτήρα που λαμβάνει υπόψη ο αλγόριθμος
- *max_angle* [float]: μέγιστη γωνιά περιστροφής του χαρακτήρα. Από αυτή υπολογίζεται επίσης η ελάχιστη γωνιά περιστροφής (-*max_angle*)
- *max_accel* [float]: μέγιστη επιτάχυνση

Output : Διάνυσμα *u* που αναπαριστά την ταχύτητα που πρέπει να ακολουθήσει ο χαρακτήρας ώστε να περιφερθεί στο χώρο.

```
1:    procedure wander(x, radius, max_angle, max_accel)
2:        theta += random_binomial() * max_angle
3:        phi += random_binomial() * max_angle
4:
5:        Vector3 target = x
6:        target.x += radius * cos(theta) * sin(phi)
7:        target.y += radius * sin(theta) * sin(phi)
8:        target.z += radius * cos(phi)
9:
10:       u = seek(x, target, max_accel)
11:   end procedure
```

Αλγόριθμος 2.3: Συμπεριφορά Wander

Αρχικά, υπολογίζονται δύο τυχαίες τιμές από [-*max_angle*, *max_angle*], οι οποίες προστίθενται στις γωνίες theta και phi, που αναπαριστούν την περιστροφή της θέσης στόχου στη σφαίρα. Στο σχήμα 2.3 φαίνεται η γωνία theta στην κάτοψη της σφαίρας. Οι τυχαίες τιμές που υπολογίζονται έχουν μεγαλύτερη πιθανότητα να βρίσκονται κοντά στο 0, έτσι ώστε ο χαρακτήρας να έχει μεγαλύτερη πιθανότητα να διατηρήσει την τρέχουσα κατεύθυνση του. Η υλοποίηση της συνάρτησης *random_binomial()* φαίνεται πιο κάτω.

Στη συνέχεια, υπολογίζεται το σημείο *target*, που βρίσκεται *πάνω* στον κύκλο.

Τέλος, καλείται η *Seek()* με παραμέτρους τη θέση του χαρακτήρα, το σημείο *target*, και τη μέγιστη επιτάχυνση *max_accel*.

Αυτό θα έχει ως αποτέλεσμα ο χαρακτήρας να περιστρέφεται περιοδικά δεξιά/αριστερά και

πάνω/κάτω σαν προχωρά. Όπως αναφέρθηκε και πιο πάνω, είναι πολύ πιο πιθανό ο χαρακτήρας να διατηρεί την κατεύθυνση του, και πολύ πιο σπάνιο να την αλλάζει απότομα.

Χρονική πολυπλοκότητα: O(1)
Χωρική πολυπλοκότητα: O(1)

Algorithm : Random Binomial

Input :

Output : *value* [float] – τυχαίος πραγματικός αριθμό (-1, 1)

```
1:    procedure random_binomial()
2:        value = random() - random()
3:    end procedure
```

Αλγόριθμος 2.4: Συνάρτηση Random Binomial

Η συνάρτηση αυτή επιστρέφει τυχαίο πραγματικό αριθμό (-1, 1) με μεγαλύτερη πιθανότητα ο αριθμός να βρίσκεται κοντά στο 0. Η συνάρτηση random( ) πρέπει να επιστρέφει τυχαίο πραγματικό αριθμό από [0, 1)

2.2.4 Path Follow

Η συμπεριφορά αυτή οδηγεί κάποιο χαρακτήρα στο να κινηθεί σε μια καθορισμένη τροχιά. Η τροχιά αυτή ορίζεται από ένα σύνολο σημειών στην οποία ο χαρακτήρας κινείται από το ένα σημείο στο άλλο με τη σειρά. Όταν ο χαρακτήρας φτάσει στο τελευταίο σημείο, υπάρχουν τρεις επιλογές:

- Να σταματήσει
- Να κινηθεί προς το πρώτο σημείο
- Να κινηθεί αντίθετα, προς το προτελευταίο σημείο και να συνεχίσει μέχρι να φτάσει στο πρώτο

Το τι από τα πιο πάνω θα γίνει καθορίζεται από την υλοποίηση του αλγορίθμου. Στο παρόν σύστημα έγινε χρήση της δεύτερης προσέγγισης.

Algorithm : Path Follow

Input :

- *path* [Vector3[]]: πίνακας με τα σημεία που ορίζουν τη διαδρομή
- *x* [Vector3]: τρέχουσα θέση του χαρακτήρα
- *index* [int]: δείκτης στο τρέχον σημείο που πρέπει ο χαρακτήρας να φτάσει
- *satisfation_radius* [float]: ακτίνα στην οποία ικανοποιείται η συνθήκη ότι ο χαρακτήρας έφτασε σε κάποιο σημείο της τροχιάς
- *max_accel* [float]: μέγιστη επιτάχυνση

Output : Διάνυσμα *u* που αναπαριστά την ταχύτητα που πρέπει να ακολουθήσει ο χαρακτήρας ώστε να ακολουθήσει την καθορισμένη τροχιά

```
1:    procedure path_follow(path, x, index,
      satisfaction_radius, max_accel)

2:        if distance(x, path[index]) < satisfaction_radius
3:           index = (index + 1) % path.length
4:
5:        u = seek(x, path[index], max_accel)
6:    end procedure
```

Αλγόριθμος 2.5: Συμπεριφορά Path Follow

Σχήμα 2.4: Χαρακτήρας που κινείται μεταξύ σημειών σε μια τροχιά

Ο αλγόριθμος λειτουργεί ως εξής:

Επιλέγοντας το επόμενο σημείο στην τροχιά βάσει του δείκτη index, καλεί τη συμπεριφορά seek ώστε να ο χαρακτήρας να κινηθεί προς αυτό. Όταν ο χαρακτήρας βρίσκεται σε

απόσταση μικρότερη του *satisfaction_radius*, θεωρείται πως ο έφτασε στο σημείο, και ο δείκτης index προχωρά στο επόμενο.

Χρονική πολυπλοκότητα: O(1)
Χωρική πολυπλοκότητα: O(1)

2.2.5 Obstacle Avoidance

Η αξιοπρεπής πλοήγηση NPC (non-player characters) απαιτεί συχνά τη δυνατότητα αποφυγής εμποδίων. Η βασική ιδέα είναι η δημιουργία μιας δύναμης που να αποφεύγει τα εμπόδια που είναι αρκετά κοντά. Ακόμη και αν το περιβάλλον έχει πολλά εμπόδια, η συμπεριφορά αυτή θα χρησιμοποιήσει ένα από αυτά κάθε φορά για να υπολογίσει τη δύναμη αποφυγής.

Ο αλγόριθμος αυτός κάνει χρήση της συνάρτηση `raycast()`, η οποία βρίσκει εάν από κάποιο σημείο, ακολουθώντας συγκεκριμένη κατεύθυνση υπάρχει σύγκρουση με κάποιο αντικείμενο, σε απόσταση *length*.

Εάν ναι, υπολογίζεται η θέση στην οποία πρέπει να κινηθεί ο χαρακτήρας ώστε να αποφύγει το εμπόδιο, που χρησιμοποιείται ως είσοδος στη συμπεριφορά *seek*.

Εάν όχι, επιστρέφεται μια ειδική τιμή *no_steering*, που σημαίνει πως ο χαρακτήρας δεν πρέπει να αλλάξει κατεύθυνση.

Χρονική πολυπλοκότητα: O(n) – O(log(n)) ανάλογα με την υλοποίηση της `raycast()`.
Χωρική πολυπλοκότητα: O(1)

Algorithm : Obstacle Avoidance

Input :

- *x* [Vector3]: τρέχουσα θέση του χαρακτήρα
- *forward* [Vector3]: διάνυσμα που δείχνει προς τα πού βλέπει ο χαρακτήρας
- *length* [float]: μήκος ακτίνας για έλεγχο εμποδίων
- *avoid_distance* [float]: απόσταση αποφυγής εμποδίων
- *max_accel* [float]: μέγιστη επιτάχυνση

Output : Διάνυσμα *u* που αναπαριστά την ταχύτητα που πρέπει να ακολουθήσει ο χαρακτήρας ώστε να αποφύγει τυχόν εμπόδια που βρίσκονται μπροστά του

```
1:    procedure obstacle_avoidance(x, forward, length,
      avoid_distance, max_accel)

2:      raycast(x, forward, length, out hit_found, out
        hit_normal, out hit_point)
3:
4:      if hit_found
5:         u = seek(x, hit_point + hit_normal *
        avoid_distance, max_accel)
6:      else
7:         u = no_steering
8:    end procedure
```

Αλγόριθμος 2.6: Συμπεριφορά Obstacle Avoidance

2.2.6 Blended Steering

Ο πιο απλός τρόπος συνδυασμού συνδυασμού συμπεριφορών είναι ο συνδυασμός των αποτελεσμάτων τους, με τη χρήση βαρών.

Ας υποθέσουμε ότι έχουμε ένα πλήθος χαρακτήρων. Οι χαρακτήρες πρέπει να κινούνται σαν μία μάζα, ωστόσο παράλληλα πρέπει να προσέχουν να μην χτυπούν ο ένας στον άλλο. Κάθε χαρακτήρας πρέπει να μένει με τους άλλους, κρατώντας όμως μια ασφαλή απόσταση. Η συνολική συμπεριφορά τους είναι ένας συνδυασμός δύο συμπεριφορών:

- Να φτάσουν στο κέντρο της μάζας της ομάδας
- Ο διαχωρισμός από τους κοντινούς χαρακτήρες

Σε καμία φάση ο χαρακτήρας δεν κάνει μόνο ένα πράγμα, αλλά πάντα λαμβάνει υπόψη του και τις δύο πιο πάνω "ανησυχίες".

Algorithm : Blended Steering

Input :

- *behaviors* [Behavior[]]: Λίστα από συμπεριφορές, στην οποία συμπεριλαμβάνεται και το βάρος της καθεμιάς
- *max_accel* [float]: μέγιστη επιτάχυνση

Output : Διάνυσμα *u* που ισούται με τον συνδυασμό των συμπεριφορών που δώθηκαν

```
1:    procedure blended(behaviors[], max_accel)

2:       accel = Vector3(0, 0, 0)
3:       for each behavior in behaviors
4:          accel += behavior.weight *
          get_steering(behavior)
5:
6:       if magnitude(accel) > max_accel
7:          accel = normalized(accel) * max_accel
8
9:       u = accel
10:   end procedure
```

Αλγόριθμος 2.7: Συμπεριφορά Blended Steering

Ο αλγόριθμος ανατρέχει κάθε συμπεριφορά και πολλαπλασιάζει το αποτέλεσμα της (που είναι ένα διάνυσμα) με το βάρος της, και στη συνέχεια το προσθέτει στο διάνυσμα accel, που είναι ο συνδυασμός των συμπεριφορών. Τέλος, γίνεται ο έλεγχος εάν η μήκος του *accel* είναι μεγαλύτερο από τη μέγιστη επιτάχυνση του χαρακτήρα. Εάν ναι, γίνεται προσαρμογή του μήκους του ώστε να ισούται με αυτή.

Σημείωση: Δεν υπάρχουν περιορισμοί σχετικά με τα βάρη – δε χρειάζεται το άθροισμα τους να ισούται με 1. Με άλλα λόγια, δεν είναι σταθμισμένος μέσος όρος.[3]

Χρονική πολυπλοκότητα: O(*n*) όπου *n* ο αριθμός των συμπεριφορών

Χωρική πολυπλοκότητα: O(1)

2.2.7 Priority Steering

Μία παραλλαγή συνδυασμού συμπεριφορών είναι η αντικατάσταση των βαρών με προτεραιότητες. Σε ένα σύστημα που βασίζεται σε προτεραιότητες, οι συμπεριφορές

οργανώνονται σε ομάδες. Το σύστημα εξετάζει την κάθε ομάδα με τη σειρά. Αν το συνολικό αποτέλεσμα μιας ομάδας είναι πολύ μικρό (μικρότερο από κάποια πολύ μικρή, αλλά προσαρμόσιμη παράμετρο), αγνοείται, και η επόμενη ομάδα εξετάζεται.

Ο αλγόριθμος προσπελάζει σειριακά την κάθε συμπεριφορά, και εξετάζει εάν το μήκος του διανύσματος της είναι μεγαλύτερο από *epsilon*. Εάν ναι, επιστρέφεται το αποτελέσμα της, αλλιώς εξετάζεται η επόμενη. Εάν δε βρεθεί συμπεριφορά που να ικανοποιεί το πιο πάνω κριτήριο, επιστρέφεται η ειδική τιμή *no_steering* (καμία μεταβολή στην κατεύθυνση του χαρακτήρα) ή επιστρέφεται το αποτέλεσμα της τελευταίας συμπεριφοράς.

Χρονική πολυπλοκότητα: Ο(*n*) όπου *n* ο αριθμός των συμπεριφορών
Χωρική πολυπλοκότητα: Ο(1)

Algorithm : Priority Steering

Input :

- *behaviors* [Behavior[]]: Λίστα από συμπεριφορές, στην οποία συμπεριλαμβάνεται και το βάρος της καθεμιάς
- *epsilon* [float]: που καθορίζει το ελάχιστο μήκος του διανύσματος μιας συμπεριφοράς

Output : Διάνυσμα *u* που ισούται με το αποτέλεσμα της πρώτης συμπεριφοράς της οποίας το μήκος του διανύσματος της που επιστρέφει ως αποτέλεσμα είναι μεγαλύτερο από *epsilon*

```
1:    procedure priority(behaviors[], epsilon)

2:      for each behavior in behaviors
3:         result = get_steering(behavior)
4:
5:         if magnitude(accel) > max_accel
6:            u = accel
7:
8:      return no_steering
9:    end procedure
```

Αλγόριθμος 2.8: Συμπεριφορά Priority Steering

2.3 Παραδοσιακός Αλγόριθμος

Η κλασική δουλειά του Reynolds[4] , που δημοσιεύτηκε το 1987, προσομοιώνει κίνηση ομάδων αυτόνομων χαρακτήρων, π.χ. πουλιών, ψαριών. Το βασικό μοντέλο αποτελείται από τρεις συμπεριφορές *(steering behaviors)*, που περιγράφουν πώς ένας χαρακτήρας κάνει ελιγμούς βάσει των θέσεων και των ταχυτήτων των κοντινών του συντρόφων.

Όταν πρωτοπαρουσιάστηκε, η προσέγγιση του Reynolds ήταν ένα τεράστιο βήμα προς τα εμπρός σε σύγκριση με τις παραδοσιακές τεχνικές που χρησιμοποιούνταν σε computer animation για κινηματογραφικές ταινίες.Το πρώτο animation που δημιουργήθηκε με το μοντέλο αυτό ήταν το *Stanley and Stella in: Breaking the Ice (1987)* ακολουθούμενο από μια πρώτη μεγάλου μήκους ταινία του Tim Burton, *Batman Returns (1992)*, όπου σμήνη νυχτερίδων πετούν στους δρόμους της Gotham City.

Οι συμπεριφορές που ακολουθούν, σε αντίθεση με τις προηγούμενες, έχουν να κάνουν με ένα σύνολο χαρακτήρων - ο κάθε χαρακτήρας λαμβάνει υπόψη του τους γύρω του.

2.3.1 Cohesion

Η συμπεριφορά αυτή προκαλεί την κίνηση προς τη μέση θέση των γειτόνων ενός χαρακτήρα, μέσα σε μια συγκεκριμένη ακτίνα.

Για κάθε χαρακτήρα *h*, ο αλγόριθμος ελέγχει εάν η απόσταση του μεταξύ της θέση *x* του χαρακτήρα που εξετάζεται, είναι μικρότερη από *radius*. Εάν ναι, η θέση του προστίθεται σε ένα διάνυσμα *center*. Όταν γίνει ο πιο πάνω έλεγχος για όλους τους χαρακτήρες, το διάνυσμα *center* διαιρείται με τον αριθμό των χαρακτήρων που ήταν μέσα στην ακτίνα που δώθηκε, και έτσι παίρνουμε το μέσο σημείο.
Τέλος, το μέσο σημείο δίνεται ως σημείο στόχου στη συμπεριφορά *seek*, ώστε να επιστραφεί το διάνυσμα που θα προκαλέσει την κίνηση του χαρακτήρα προς αυτό.

Χρονική πολυπλοκότητα: $O(n)$ όπου n ο αριθμός των χαρακτήρων
Χωρική πολυπλοκότητα: $O(1)$

Σημείωση: Η συμπεριφορά αυτή καλείται για κάθε χαρακτήρα, άρα η συνολική χρονική πολυπλοκότητα είναι $O(n^2)$. Με χρήση όμως ειδικών δομών για εύρεση των γειτόνων, όπως k-d trees[5], η πολυπλοκότητα μπορεί να μειωθεί σε $O(n \log(n))$.

Algorithm : Cohesion

Input :

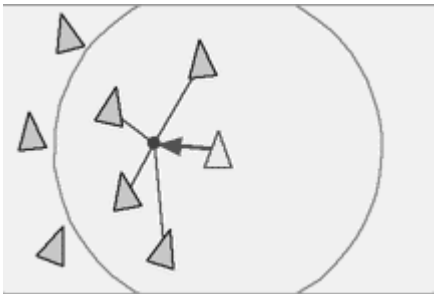
- *x* [Vector3]: θέση του χαρακτήρα
- *boids* [Boid[]]: λίστα από τους χαρακτήρες που αποτελεούν το σμήνος
- *radius* [float]: ακτίνα στην οποία λαμβάνονται υπόψη χαρακτήρες ως γείτονες γύρω από τη θέση *x*
- *max_accel* [float]: μέγιστη επιτάχυνση

Output : Διάνυσμα *u* που αναπαριστά την ταχύτητα που πρέπει να ακολουθήσει ο χαρακτήρας ώστε κινηθεί προς το κέντρο της γειτονιάς του

```
1:    procedure cohesion(x, boids[], radius, max_accel)

2:       center = Vector3(0, 0, 0)
3:       count = 0
4:
5:       for each boid in boids
6:          if distance(x, boid.position) <= radius
7:             center += boids.position
8:             count += 1
9:       center /= count
10:
11:      u = seek(x, center, max_accel)
12:   end procedure
```

Αλγόριθμος 2.9: Συμπεριφορά Cohesion



Σχήμα 2.5: Η συμπεριφορά Cohesion προκαλεί κίνηση του χαρακτήρα προς το μέσο σημείο των γειτόνων του

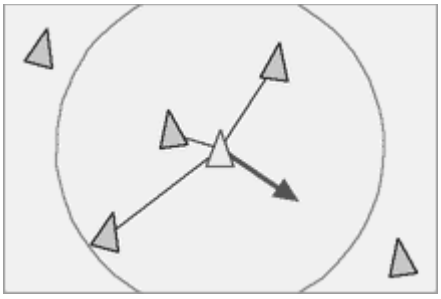
2.3.2 Separation

Η συμπεριφορά αυτή προκαλεί την απομάκρυνση προς τη μέση θέση των γειτόνων ενός χαρακτήρα, σε μια συγκεκριμένη ακτίνα. Είναι η αντίθετη της *Cohesion*.

Για κάθε χαρακτήρα *h*, ο αλγόριθμος ελέγχει εάν η απόσταση του μεταξύ της θέση *x* του χαρακτήρα που εξετάζεται, είναι μικρότερη από *radius*. Εάν ναι, η θέση του προστίθεται σε ένα διάνυσμα *center*. Όταν γίνει ο πιο πάνω έλεγχος για όλους τους χαρακτήρες, το διάνυσμα *center* διαιρείται με τον αριθμό των χαρακτήρων που ήταν μέσα στην ακτίνα που δώθηκε, και έτσι παίρνουμε το μέσο σημείο.

Τέλος, το μέσο σημείο δίνεται ως σημείο στόχου στη συμπεριφορά *flee*, ώστε να επιστραφεί το διάνυσμα που θα προκαλέσει την απομάκρυνση του χαρακτήρα προς αυτό.

Χρονική πολυπλοκότητα: O(*n*) όπου *n* ο αριθμός των χαρακτήρων
Χωρική πολυπλοκότητα: O(1)



Σχήμα 2.6: Η συμπεριφορά Separation προκαλεί απομάκρυνση ενός χαρακτήρα από τους γείτονες του

Algorithm : Separation

Input :

- *x* [Vector3]: θέση του χαρακτήρα
- *boids* [Boid[]]: λίστα από τους χαρακτήρες που αποτελεούν το σμήνος
- *radius* [float]: ακτίνα στην οποία λαμβάνονται υπόψη χαρακτήρες ως γείτονες γύρω από τη θέση *x*
- *max_accel* [float]: μέγιστη επιτάχυνση

Output : Διάνυσμα *u* που αναπαριστά την ταχύτητα που πρέπει να ακολουθήσει ο χαρακτήρας ώστε απομακρυνθεί προς το κέντρο της γειτονιάς του

```
1:    procedure separation(x, boids[], radius, max_accel)

2:       center = Vector3(0, 0, 0)
3:       count = 0
4:
5:       for each boid in boids
6:          if distance(x, boid.position) <= radius
7:             center += boids.position
8:             count += 1
9:       center /= count
10:
11:      u = flee(x, center, max_accel)
12:   end procedure
```

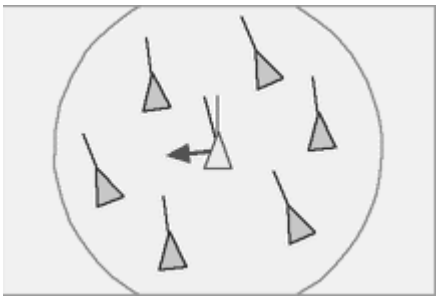
Αλγόριθμος 2.10: Συμπεριφορά Separation

2.3.3 Alignment (ή Velocity Match)

Η συμπεριφορά Alignment προκαλεί τη στοίχιση γειτονικών χαρακτήρων, δηλαδή την κίνηση τους προς την ίδια κατεύθυνση.

Για κάθε χαρακτήρα *h*, ο αλγόριθμος ελέγχει εάν η απόσταση του μεταξύ της θέση *x* του χαρακτήρα που εξετάζεται, είναι μικρότερη από *radius*. Εάν ναι, η ταχυτητα του, *v*, προστίθεται σε ένα διάνυσμα *avg_vel*. Όταν γίνει ο πιο πάνω έλεγχος για όλους τους χαρακτήρες, το διάνυσμα *avg_vel* διαιρείται με τον αριθμό των χαρακτήρων που ήταν μέσα στην ακτίνα που δώθηκε, και έτσι παίρνουμε τη μέση ταχύτητα.

Τέλος, αφαιρώντας από αυτή την ταχύτητα του χαρακτήρα, παίρνουμε το διάνυσμα u που είναι η ταχύτητα που πρέπει να ακολουθήσει ο χαρακτήρας ώστε να κινηθεί προς τα εκεί που κινούνται οι γείτονες του.



Σχήμα 2.7: Χαρακτήρας που κινείται προς την κατεύθυνση κίνησης των γειτόνων του, μέσω της συμπεριφοράς Alignment

Χρονική πολυπλοκότητα: O(*n*) όπου *n* ο αριθμός των χαρακτήρων

Χωρική πολυπλοκότητα: O(1)

Algorithm : Alignment

Input :

- $\underline{x}$ [Vector3]: θέση του χαρακτήρα
- $\underline{v}$ [Vector3]: διάνυσμα ταχύτητας χαρακήρα
- *boids* [Boid[]]: λίστα από τους χαρακτήρες που αποτελεούν το σμήνος
- *radius* [float]: ακτίνα στην οποία λαμβάνονται υπόψη χαρακτήρες ως γείτονες γύρω από τη θέση *x*
- *max_accel* [float]: μέγιστη επιτάχυνση

Output : Διάνυσμα $\underline{u}$ που αναπαριστά την ταχύτητα που πρέπει να ακολουθήσει ο χαρακτήρας ώστε κινηθεί προς τα εκεί που κινούνται οι γείτονες του, να στοιχιστεί δηλαδή με αυτούς.

```
1:    procedure alignment(x, v, boids[], radius, max_accel)

2:       avg_vel = Vector3(0, 0, 0)
3:       count = 0
4:
5:       for each boid in boids
6:          if distance(x, boid.position) <= radius
7:             avg_vel += boid.velocity
8:             count += 1
9:          endif
10:      avg_vel /= count
11:
12:      u = avg_vel - v
13:
14:      if magnitude(u) > max_accel
            u = normalized(u) * max_accel
15:   end procedure
```

Αλγόριθμος 2.11: Συμπεριφορά Alignment

2.4 Πλεονεκτήματα / Μειονεκτήματα

2.4.1 Πλεονεκτήματα

- Το μεγαλύτερο πλεονέκτημα του μοντέλου που περιγράφηκε είναι η απλότητα του. Δε βασίζεται σε πολύπλοκες στρατηγικές, τα αποτελέσματα του όμως είναι εντυπωσιακά.
- Κάθε χαρακτήρας είναι ανεξάρτητος και διαφορετικός από τους άλλους (διαφορετικές παραμέτροι στις συμπεριφορές, μέγιστη ταχύτητα, μέγιστη επιτάχυνση), άρα μπορεί να εκδηλώσει διαφορετική συμπεριφορά.
- Λόγω της απλότητας του, το μοντέλο αυτό μπορεί εύκολα να επεκταθεί με την προσθήκη νέων συμπεριφορών ή συνδυασμών από αυτές.

2.4.2 Μειονεκτήματα

- Το μοντέλο αυτό μπορεί να αναπαραστήσει μόνο συμπεριφορές όπου οι χαρακτήρες "περιφέρονται" στο περιβάλλον τους.
- Λόγω της πολυπλοκότητας του, η χρήση περισσότερων χαρακτήρων από μερικές χιλιάδες δεν έχει ακόμη επιτευχθεί.

2.5 Επεκτάσεις και χρήσεις του μοντέλου

2.5.1 Επεκτάσεις

Το βασικό μοντέλο έχει επεκταθεί με πολλούς διαφορεκούς τρόπους από τότε που ο Reynolds το πρότεινε. Για παράδειγμα, οι Dalgado-Mata et al[6] επέκτειναν το μοντέλο ώστε να ενσωματώσουν την επίδραση του φόβου σε αυτό. Οι Hartman and Benes[7] έκαναν χρήση μιας συμπληρωματικής δύναμης στη συμπεριφορά Alignment, που ονόμασαν Change of Leadership (αλλαγή ηγεσίας). Η δύναμη αυτή ουσιαστικά είναι η πιθανότητα ο χαρακτήρας να γίνει αρχηγός και να προσπαθήσει να δραπετεύσει.

2.5.2 Χρήσεις

Το μοντέλο του Reynolds χρησιμοποιείται συχνά σε γραφικά ηλεκτρονικών υπολογιστών, παρέχοντας ρεαλιστικές αναπαραστάσεις σμηνών πουλιών και άλλων πλασμάτων, όπως ψάρια ή κοπάδια ζώων. Για παράδειγμα, έχει χρησιμοποιηθεί στο βιντεοπαιχνίδι του 1998 *Half-Life*, για ιπτάμενα πλάσματα που έμοιζαν με πουλιά.

Κεφάλαιο 3

Επισκόπηση

Πίνακας Περιεχομένων

Σε αυτό το κεφάλαιο περιγράφεται μια γενική επισκόπηση του συστήματος που αναπτύχθηκε. Περιγράφονται οι τρεις διαφορετικές του λειτουργίες, ενώ στη συνέχεια γίνεται αναφορά των εργαλείων / βιβλιοθηκών που χρησιμοποιήθηκαν.

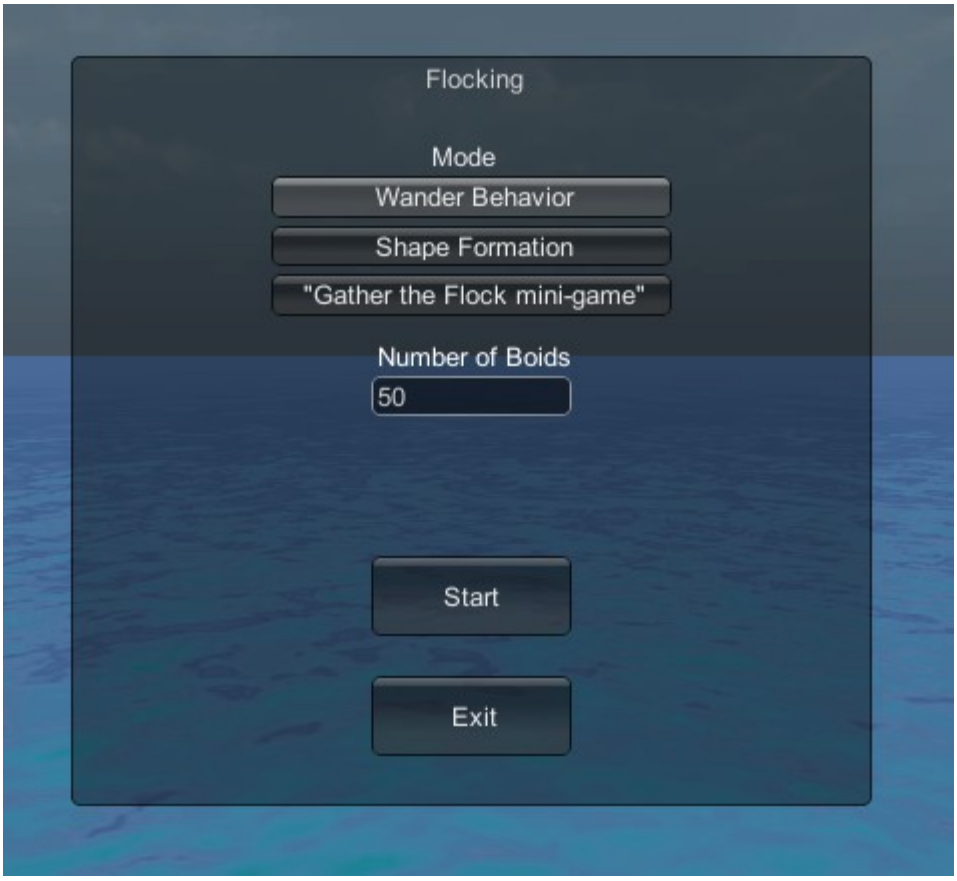
3.1 Λειτουργίες

Το σύστημα που αναπτύχθηκε περιλαμβάνει τρεις λειτουργίες. Η καθεμιά κάνει χρήση κάποιου συνδυασμού συμπεριφορών κατεύθυνσης για να επιτύχει το απαιτούμενο αποτέλεσμα.
Οι λειτουργίες είναι οι εξής:

- Λειτουργία Περιπλάνησης (Wander)
- Αναπαράσταση σχημάτων από boids

- “Gather the Flock” mini-game

Ο χρήστης επιλέγει τη λειτουργία που θέλει στο κεντρικό μενού. Η λειτουργία περιπλάνησης δίνει τη δυνατότητα επιλογής του αριθμού των χαρακτήρων, ενώ οι άλλες δύο επιλογές όχι, μιας και στη λειτουργία αναπαράστασης σχημάτων ο αριθμός των χαρακτήρων εξαρτάται από το σχήμα, ενώ στη λειτουργία παιχνιδιού ο αριθμός χαρακτήρων είναι σταθερός.



Σχήμα 3.1: Μενού επιλογής λειτουργίας

3.1.1 Λειτουργία Περιπλάνησης (Wander)

Η λειτουργία περιπλάνησης προσομοιώνει σμήνος/σμήνη πουλιών που περιφέρονται στον τρισδιάστατο χώρο.

Για την επίτευξη της συμπεριφοράς περιπλάνησης έγινε χρήση των τριών βασικών συμπεριφορών κατεύθυνσης του Reynolds: Cohesion, Separation και Alignment. Επίσης, γίνεται χρήση της Obstacle Avoidance ώστε τα πουλιά να αποφεύγουν τυχόν εμπόδια που βρίσκονται στο δρόμο τους.

Λόγω του ότι με τις τρεις συμπεριφορές Cohesion, Separation και Alignment δεν υπάρχει έλεγχος προς τα πού θα κινηθεί το σμήνος, έγινε χρήση ακόμα μιας συμπεριφοράς, της Cohesion to Anchor.

Ουσιαστικά πρόκειται για τη συνηθισμένη συμπεριφορά Cohesion, με τη διαφορά ότι κάθε χαρακτήρας (boid) προσπαθεί να ακολουθήσει ένα συγκεκριμένο χαρακτήρα (αρχηγό του σμήνους) που κινείται ανεξάρτητα του σμήνους.

Αυτό δίνει τη δυνατότητα καθορισμού οποιασδήποτε συμπεριφοράς (π.χ Seek, Flee, Path Follow) στο χαρακτήρα που λειτουργεί ως αρχηγός, και το σμήνος θα τον ακολουθήσει, διατηρώντας παράλληλα τα χαρακτηριστικά του.

Ο χρήστης μπορεί να κατευθύνει τον αρχηγό με το πληκτρολόγιο, καθώς και να αλλάξει τις παραμέτρους που αφορούν τις συμπεριφορές Cohesion, Separation και Alignment.

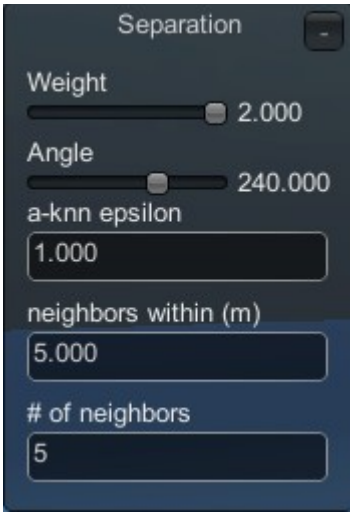
Η λειτουργία περιπλάνησης είναι ιδανική για χρήση σε εφαρμογές όπου δεν απαιτείται ακριβής έλεγχος στο σχηματισμό του σμήνους, επειδή ο τρόπος με τον οποίο μεταβάλλονται οι θέσεις των πουλιών στο σμήνος είναι μη προβλέψιμος (π.χ σε βιντεοπαιχνίδια).

Σχήμα 3.2: Σμήνος που ακολουθεί τον αρχηγό (στα αριστερά)

Φυσικά, υπάρχει η δυνατότητα καθορισμού συγκεκριμένης πορείας του σμήνους. Για παράδειγμα, εάν ο αρχηγός έχει τη συμπεριφορά Path Follow και ακολουθεί μια

συγκεκριμένη πορεία, το σμήνος, λόγω του συνδυασμού συμπεριφορών κατεύθυνσης του, θα ακολουθήσει επίσης την ίδια πορεία.

Ο χρήστης μπορεί να αλλάξει τις παραμέτρους για τις συμπεριφορές Cohesion, Separation και Alignment (Velocity Match), όπως φαίνεται στο σχήμα 3.3.



Σχήμα 3.3: Παραμέτροι για τη συμπεριφορά Separation

Οι παραμέτροι αυτές έχουν την εξής σημασία:

- Weight: Ο βαθμός που επηρεάζει το αποτέλεσμα της συμπεριφοράς αυτής το τελικό αποτέλεσμα. Π.χ εάν το βάρος της συμπεριφοράς Separation είναι 0, οι χαρακτήρες δε θα προσπαθούν να αποφύγουν τους κοντινούς τους γείτονες, με αποτέλεσμα όλοι οι χαρακτήρες στο τέλος να συμμαζευτούν σε ένα σημείο.
- A-knn epsilon: Τιμή που καθορίζει το πόσο ακριβής θα είναι ο υπολογισμός των k κοντινότερων γειτότων (k nearest neighbors)[8]. Εάν είναι 0 τότε είναι ισοδύναμο με τη χρήση του knn, ενώ εάν πάρει μεγάλες τιμές οι χαρακτήρες θα θεωρούν σχεδόν τυχαίους χαρακτήρες ως γείτονες τους.
- Neighbors within: Ακτίνα που καθορίζει τη γειτονιά ενός χαρακτήρα. Για τη

συμπεριφορά Separation, είναι η απόσταση στην οποία κάποιος χαρακτήρας προσπαθεί να αποφεύγει τους κοντινούς του γείοτονες. Εάν ανατεθεί μεγάλη τιμή, το σμήνος θα γίνει πολύ αραιό, ενώ αν στη συνέχεια ανατεθεί μικρή τιμή το σμήνος θα ξαναενωθεί, δημιουργώντας πολλά υποσμήνη.

- Number of neighbors: Αριθμός γειτόνων που λαμβάνονται υπόψη για τον αλγόριθμο aknn (ουσιαστικά η παράμετρος *k*). Όσο μεγαλύτερο είναι το k τόσο πιο αργός θα είναι ο υπολογισμός των γειτόνων, αλλά τα αποτελέσματα θα είναι *διαφορετικά* (όχι απαραίτητα καλύτερα) απ'ότι όταν το *k* είναι μικρό.

3.1.2 Αναπαράσταση σχημάτων

Η λειτουργία αναπαράστασης σχημάτων έχει ως στόχο την αναπαράσταση τρισδιάστατων σχημάτων από boids. Για παράδειγμα, το σμήνος πουλιών μπορεί να κινείται στο χώρο κρατώντας το σχήμα μιας φάλαινας, ενώ στη συνέχεια να αλλάξει το σχήμα του σε κάτι εντελώς διαφορετικό.

Οι συμπεριφορές κατεύθυνσης που χρησιμοποιήθηκαν συνδυάστηκαν ως μια συμπεριφορά Priority Steering με την εξής προτεραιότητα:

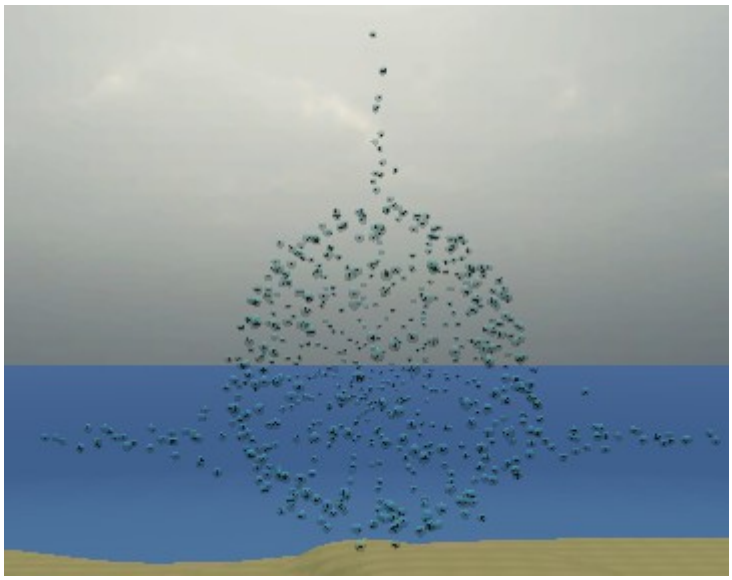
- Obstacle Avoidance για αποφυγή τυχόν εμποδίων
- Separation ώστε τα πουλιά να μην είναι πολύ κοντά το ένα στο άλλο
- Pattern Steering ώστε το κάθε πουλί να έχει τη δική του θέση προορισμού

Επίσης, γίνεται χρήση κάποιου ανεξάρτητου χαρακτήρα από το σμήνος που λειτουργεί ως αρχηγός του, όπως και στη λειτουργία περιπλάνησης, ώστε να υπάρχει η δυνατότητα το σμήνος να κινείται στο χώρο ενώ διατηρεί το σχήμα του.

Για την επίτευξη της δημιουργίας και διατήρησης του επιθυμητού σχήματος, ανατίθεται σε κάθε χαρακτήρας του σμήνους μια συγκεκριμένη (και μοναδική) θέση σε σχέση με τη θέση του αρχηγού.

Τα σημεία φορτώνονται από αρχεία που περιγράφουν τρισδιάστατα μοντέλα.

Η αναπαράσταση σχημάτων είναι χρήσιμη όταν χρειάζεται πλήρης έλεγχος του σχήματος του σμήνους, π.χ σε διαφημίσεις, ταινίες αλλά και σε βιντεοπαιχνίδια.



Σχήμα 3.4: Σμήνος πουλιών σε σχήμα φάλαινας (σε πρόσοψη)

3.1.3 "Gather the Flock" mini-game

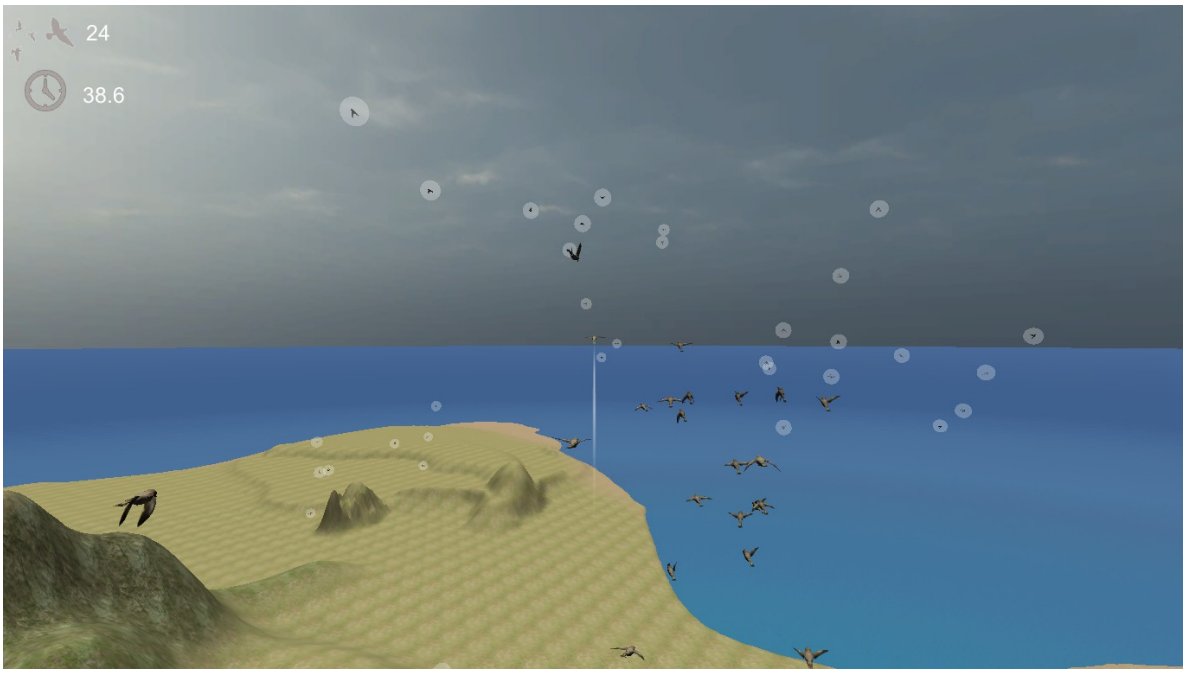
Η τρίτη λειτουργία του συστήματος είναι ένα απλό mini-game στο οποίο γίνεται χρήση των συμπεριφορών κατεύθυνσης.

Ο παίκτης παίρνει τον έλεγχο ενός πουλιού, και έχει τη δυνατότητα να κινείται μέσα σε ένα νησί, όπου υπάρχουν άλλα πουλιά. Πλησιάζοντας τα, τα πουλιά γίνονται μέλη του σμήνους του και τον ακολουθούν.

Στόχος του παιχνιδιού είναι ο παίκτης να μαζέψει όσα περισσότερα πουλιά σε 60 δευτερόλετα. Οι καλύτερες βαθμολογίες αποθηκεύονται.

Στο πάνω αριστερό μέρος της οθόνης αναγράφονται το πόσα πουλιά έχουν γίνει μέλη του σμήνους του παίκτη καθώς και ο χρόνος που απομένει.

Τα πουλιά τα οποία δεν ανήκουν στο σμήνος εμφανίζονται μέσα σε γκρίζα σφαίρα, η οποία εξαφανίζεται όταν γίνουν μέλη του. Συνολικά υπάρχουν 200 πουλιά στο παιχνίδι.



Σχήμα 3.5: Στιγμιότυπο από το παιχνίδι

3.2 Εργαλεία και βιβλιοθήκες

3.2.1 Unity3D

Το Unity3D είναι μια πλατφόρμα ανάπτυξης παιχνιδιών (game engine). Χρησιμοποιείται για την ανάπτυξη ηλεκτρονικών παιχνιδιών για web plugins, επιτραπέζιους υπολογιστές, κονσόλες και κινητές συσκευές.

Το σύστημα που δημιουργήθηκε για την παρούσα διπλωματική εργασία βασίζεται σε αυτή την πλατφόρμα, λόγω των ευκολιών αλλά και της ταχύτητας που προσφέρει στην ανάπτυξη, αφού παρέχει πληθώρα βιβλιοθηκών έτοιμου κώδικα, editor και πολλά άλλα.

3.2.2 Alglib

Η alglib είναι μία βιβλιοθήκη αριθμητικής ανάλυσης και επεξεργασίας δεδομένων. Υποστηρίζει πολλές γλώσσες προγραμματισμού (C++, C#, Pascal) και διάφορα

λειτουργικά συστήματα (Windows, Linux, Solaris). Οι δυνατότητες της περιλαμβάνουν, μεταξύ άλλων:

- Ανάλυση δεδομένων (ταξινόμηση/παλινδρόμηση, νευρωνικά δίκτυα)
- Παρεμβολή (interpolation) και γραμμική/μη γραμμική τοποθέτηση ελαχίστων τετραγώνων
- Μετασχηματισμοί Fourier και πολλοί άλλοι αλγόριθμοι

Επίσης υποστηρίζει τον αλγόριθμο knn[5] (k nearest neighbors) και aknn (approximate knn), κάνοντας χρήση K-D δέντρων (K-D trees). Ο αλγόριθμος αυτός χρησιμοποιήθηκε για την εύρεση της γειτονιάς χαρακτήρων στις συμπεριφορές Cohesion, Separation και Alignment.

3.2.3 Leap Motion

Η συσκευή Leap Motion[2] παρέχει δυνατότητες ανίχνευσης κίνησης των χεριών μέσω αισθητήρων, με πολύ καλή ακρίβεια και ταχύτητα. Η εν λόγω συσκευή μπορεί να χρησιμοποιηθεί στις λειτουργίες που αναπτύχθηκαν για έλεγχο του αρχηγού του σμήνους από το χρήστη. Με την κίνηση του χεριού αριστερά/δεξιά/πάνω/κάτω ο αρχηγός του σμήνους στρίβει με όμοιο τρόπο (και συνεπώς στρίβει και το σμήνος που ακολουθεί).

Σχήμα 3.6: Η συσκευή Leap Motion

Κεφάλαιο 4

Υλοποίηση

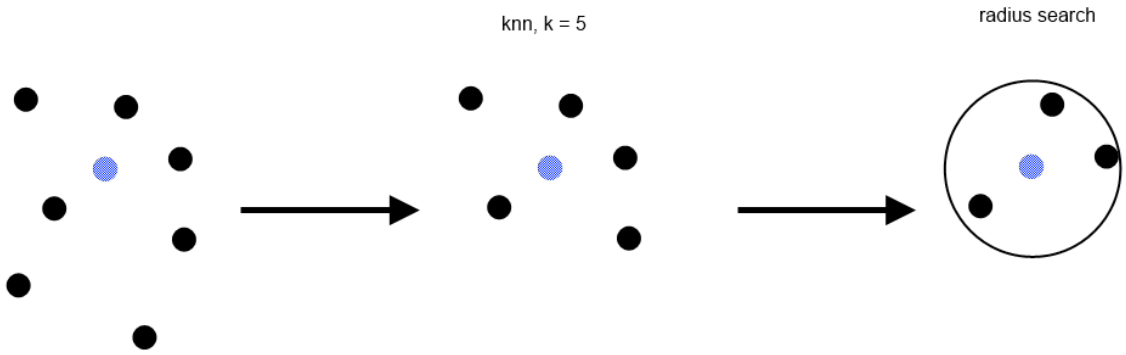
Πίνακας Περιεχομένων

Στο παρόν κεφάλαιο θα γίνει αναλυτική εξήγηση του πώς υλοποιήθηκε κάθε λειτουργία του συστήματος καθώς και επιμέρους υπολειτουργιών που χρειάστηκε να αναπτυχθούν.

Σε κάθε λειτουργία, οι χαρακτήρες ενός σμήνους έχουν συγκεκριμένη συμπεριφορά, συνδυασμό των συμπεριφορών Cohesion, Separation, Alignment, Path

Όπως είδαμε στο κεφάλαιο 2, οι τρεις συμπεριφορές κατεύθυνσης που έχουν να κάνουν με ομάδες από boids (Cohesion, Separation, Alignment) υπολογίζουν τα αποτελέσματα τους λαμβάνοντας υπόψη γείτονες σε μια συγκεκριμένη ακτίνα. Στο σύστημα που αναπτύχθηκε, έγινε συνδυασμός της πιο πάνω μεθόδου με τον αλγόριθμο που υπολογίζει τους k κοντινότερους γείτονες (k-nearest neighbor search ή knn). Για κάθε χαρακτήρα στο σμήνος, αρχικά γίνεται χρήση του knn για εύρεση των k κοντινότερων γειτόνων, βάσει της απόστασης τους με αυτόν στον τρισδιάστατο χώρο. Στη συνέχεια, απορρίπτονται όσοι από αυτούς δε βρίσκονται μέσα σε μια συγκεκριμένη απόσταση. Τέλος υπολογίζεται η μέση

θέση και ταχύτητα τους.



Σχήμα 4.1: Η επιλογή της "γειτονιάς" ενός χαρακτήρα: Αρχικά γίνεται χρήση του knn και στη συνέχεια του radius search

Λόγω της φύσης της εφαρμογής, δεν είναι απαραίτητος ο υπολογισμός του πραγματικού κέντρου και μέσης ταχύτητας της γειτονιάς ενός χαρακτήρα αλλά τιμών “κοντά” σε αυτές. Γι'αυτό αντί του knn γίνεται χρήση του aknn (approximate knn), με αντάλλαγμα βελτίωσης της ταχύτητας ή/και εξοικονόμηση μνήμης. Ένα άλλο θετικό του αλγορίθμου aknn είναι το πως υπάρχει η πιθανότητα εμφάνισης “περίεργης” συμπεριφοράς από κάποιο χαρακτήρα, κάτι που δε θα συνέβαινε στον κλασικό knn. Για παράδειγμα, μπορεί να υπάρχουν δύο σμήνη και κάποιος χαρακτήρας να αλλάξει σμήνος, δηλαδή να φύγει από το σμήνος στο οποίο ανήκει, να κινηθεί προς το άλλο σμήνος και να γίνει μέλος του. Αυτό συμβαίνει επειδή στη συγκεκριμένη περίπτωση επηρεάζεται η συμπεριφορά Cohesion και υπολογίζει διαφορετικό κέντρο της γειτονιάς του χαρακτήρα από το πραγματικό, που τυγχαίνει να είναι κοντά στο άλλο σμήνος, κι έτσι ο χαρακτήρας κινείται προς αυτό.

Στην πλατφόρμα Unity3D, όπως και στα περισσότερα game engines, υπάρχει η έννοια του Game Object, δηλαδή κάποιου αντικειμένου στο παιχνίδι (π.χ κάποιος χαρακτήρας, ένας εχθρός). Η ανάθεση κάποιας συμπεριφοράς σε κάποιο αντικείμενο γίνεται με σύνδεση του με κάποιο script, στο οποίο υπάρχει ο κώδικας για την εν λόγω συμπεριφορά.

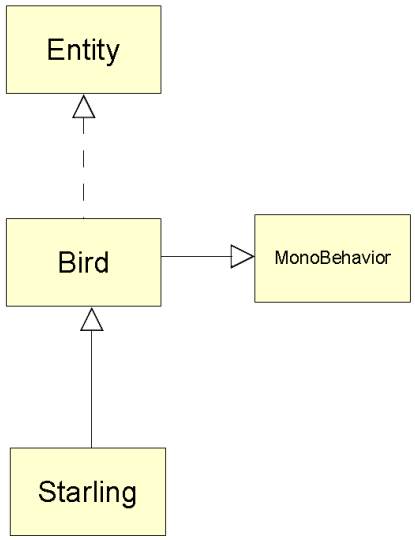
Μερικά από τα χαρακτηριστικά κάποιου game object στην εν λόγω πλατφόρμα είναι:

- Name: όνομα για αυτό το αντικείμενο.

- Tag: Ετικέτα για κάποιο αντικείμενο. Συνήθως κάποια ετικέτα χρησιμοποιείται για πολλά αντικείμενα, συνήθως παρόμοια, ή που έχουν κάτι κοινό. Για παράδειγμα, στην παρούσα υλοποίηση κάθε πουλί έχει το την ετικέτα “Starling”.
- Transform: Δίνει στο αντικείμενο τη δυνατότητα να ορίσει τη θέση του στον τρισδιάστατο χώρο του, την περιστροφή του καθώς και την κλίμακα του (position, orientation, scale).
- Rigidbody: Χρησιμοποιείται όταν το αντικείμενο αυτό πρέπει να επηρεάζεται από κανόνες φυσικής. Περιλαμβάνει μάζα, επιλογή εάν το αντικείμενο θα επηρεάζεται από τη βαρύτητα και άλλες παραμέτρους.
- Collider: Τρισδιάστατο σχήμα που τοποθετείται γύρω από το αντικείμενο και χρησιμοποιείται για ανίχνευση συγκρούσεων (collision detection).

Μια συμπεριφορά για να μπορεί να ανατεθεί σε κάποιο game object πρέπει να κληρονομεί από την κλάση *MonoBehavior*. Κληρονομώντας από αυτή, κάποια κλάση αποκτά πρόσβαση σε διάφορες συναρτήσεις / αντικείμενα που της παρέχονται μέσω του κληροδότη της.

Τα αντικείμενα που αναπαριστούν πουλιά ανήκουν στην κλάση *Starling*, η οποία κληρονομεί από την κλάση Bird, η οποία κληρονομεί από τη MonoBehavior. Η κλάση Bird υλοποιεί επίσης τη διαπροσωπεία Entity, όπως φαίνεται στο διάγραμμα 4.2.



Σχήμα 4.2: Η κλάση Starling και οι κληροδότες της

Με τον τρόπο που υλοποιείται η πιο πάνω ιεραρχία κλάσεων, επιτρέπεται η εύκολη προσθήκη άλλω ειδών πουλιών (που θα κληρονομούν από την Bird) και θα συμπεριφέρονται διαφορετικά. Στο σύστημα που αναπτύχθηκε υπάρχει μόνο ένας πουλιών, Starling.

Κάθε αντικείμενο τύπου Bird περιλαμβάνει ένα αντικείμενο τύπου BirdState. Το αντικείμενο αυτό καθορίζει εξ' ολοκλήρου τη συμπεριφορά κάποιου πουλιού. Διαφορετικές συμπεριφορές μπορούν να δημιουργηθούν υλοποιώντας τη διαπροσωπεία BirdState.

Από τη διαπροσωπεία BirdState κληρονομούν δύο abstract κλάσεις, οι SingleBirdState και MultipleBirdState. Η πρώτη αφορά συμπεριφορά για ένα αντικείμενο Bird ενώ η δεύτερη μπορεί να οριστεί ως συμπεριφορά σε ένα σύνολο αντικειμένων τύπου Bird. Οι δύο αυτές κλάσεις καθορίζουν το πώς θα κινηθεί κάποιο πουλί, χρησιμοποιώντας συνδυασμό συμπεριφορών πλοήγησης.

Για κάθε μια από τις τρεις λειτουργίες του συστήματος που περιγράφηκαν στο Κεφάλαιο 3 (Λειτουργία Περιπλάνησης, Αναπαράσταση σχημάτων από boids και “Gather the Flock” mini-game), δημιουργήθηκαν οι κλάσεις FlockWander, FlockFormation και FlockGame, οι οποίες κληρονομούν από την MultipleBirdState.

4.1 Η κλάση Main

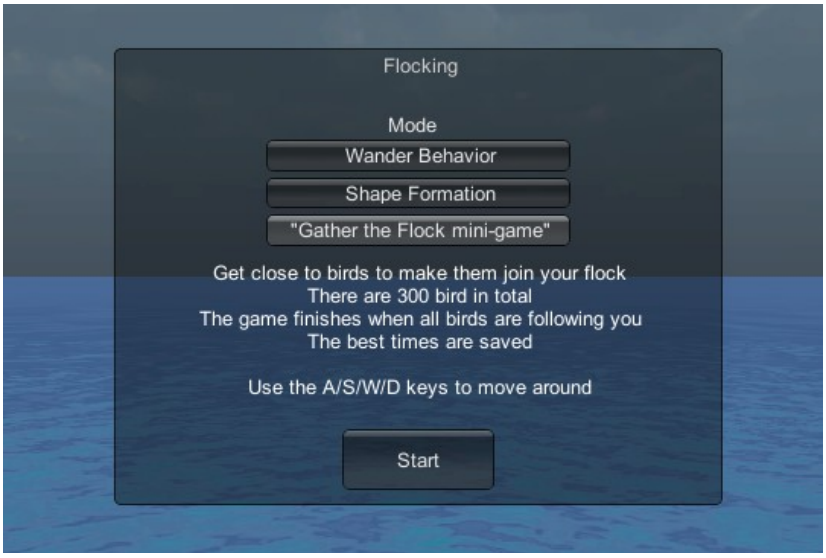
Η κλάση *Main* λειτουργεί ως κεντρικό σημείο της εφαρμογής. Στην κλάση αυτή γίνονται οι εξής λειτουργίες:

- Εμφανίζεται το κυρίως μενού καθώς και τα υπόλοιπα μενού της κάθε λειτουργίας.
- Τοποθέτηση κάμερας ανάλογα με τη λειτουργία, τη μέση θέση του σμήνους
- Διάβασμα των αρχείων με τους καλύτερους χρόνους για το παιχνίδι “Gather the Flock” καθώς και αποθήκευση τους
- Δημιουργία αντικειμένων που αναπαριστούν πουλιά και τοποθέτηση τους στο χώρο
- Διαχείριση αντικειμένου *state* τύπου MultipleBirdState που αντιστοιχεί στη λειτουργία που είναι ενεργή την παρούσα στιγμή

Το script που περιέχει την εν λόγω κλάση χρησιμοποιείται σε ένα game object με τον όνομα Main, ώστε να υπάρχει ένα μοναδικό αντικείμενο (instance) της κλάσης αυτής στην εφαρμογή.

4.1.1 Εμφάνιση μενού

Στο κεντρικό μενού εμφανίζεται μια λίστα με τις λειτουργίες που μπορεί ο χρήστης να επιλέξει. Ανάλογα με το τι θα επιλεγεί, δημιουργείται το αντικείμενο *state*, και εμφανίζονται τα ανάλογα μενού / εικονίδια στην οθόνη.

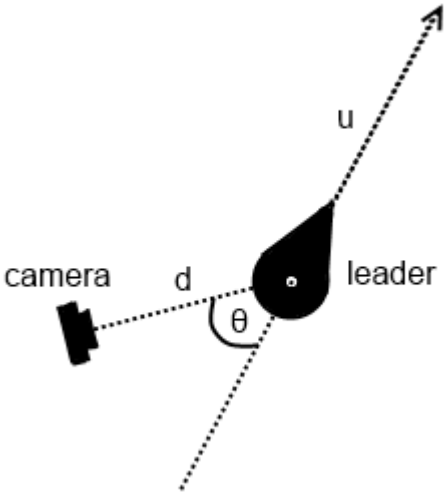


Σχήμα 4.3: Κυρίως μενού όπου η λειτουργία παιχνιδιού είναι επιλεγμένη

4.1.2 Τοποθέτηση κάμερας

Η κάμερα τοποθείται σε απόσταση *d* από το κέντρο του αρχηγού του σμήνους, με περιστροφή *θ* μοίρες γύρω από το διάνυσμα ταχύτητας του *u*. Ο χρήστης μπορεί να αλλάξει την απόσταση αυτή με τον τροχό του ποντικιού του και την περιστροφή με το αριστερό / δεξί βέλος του πληκτρολογίου του.

Στη λειτουργία Περιπλάνησης υπάρχει επίσης η επιλογή η κάμερα να ακολουθεί κάποιο τυχαίο πουλί στο σμήνος.



Σχήμα 4.4: Τοποθέτηση κάμερας σε σχέση με τον αρχηγό του σμήνους

4.2 Συμπεριφορές Κατεύθυνσης

Μια συμπεριφορά κατεύθυνσης πρέπει να υλοποιεί τη διαπροσωπεία Steering, η οποία φαίνεται πιο κάτω:

```
public interface Steering
{
     SteeringOutput GetSteering();
}
```

Η δομή `SteeringOutput` περιέχει μία μεταβλητή – το διάνυσμα ταχύτητας που επιστρέφει η κάθε συμπεριφορά κατεύθυνσης. Επίσης, υπάρχει το στατικό πεδίο `SteeringOutput.None` που επιστρέφει την ειδική τιμή *`no_steering`*, δηλαδή πως ο χαρακτήρας δεν πρέπει να μεταβάλει την ταχύτητα του.

Οι διάφορες συμπεριφορές κατεύθυνσης υλοποιούν την πιο πάνω διαπροσωπεία και τη συνάρτηση `GetSteering()`. Οι διάφορες παραμέτροι μπορούν να περαστούν στην κάθε συμπεριφορά μέσω του κατασκευαστή τους (constructor).

4.2.1 Συμπεριφορές σμηνών – Cohesion, Separation, Alignment

Οι συμπεριφορές σμηνών – δηλαδή εκείνες που έχουν να κάνουν με ομάδες από boids, κληρονομούν από την abstract κλάση `BoidSteering` (που υλοποιεί τη διαπροσωπεία Steering), που περιέχει διάφορες μεταβλητές / παραμέτρους:

Type	Name	Description
Entity	character	Χαρακτήρας στον οποίο αναφέρεται η συμπεριφορά αυτή
Flock	flock	Κλάση που περιέχει λίστα με τους χαρακτήρες που ανήκουν στο σμήνος
float	neighborhoodMaxDistance	Απόσταση στην οποία λαμβάνονται υπόψη άλλοι χαρακτήρες ως γείτονες (σε μέτρα)

float	neighborhoodMinDotProduct	Ελάχιστο εσωτερικό γινόμενο. Καθορίζει τη γωνία στην οποία γύρω χαρακτήρες θα θεωρηθούν ως γείτονες
int	maxNeighborhoodSize	Μέγιστος αριθμός χαρακτήρων που θα εξεταστούν ως γείτονες
double	aknnApproxVal	Παράμετρος για τον αλγόριθμο aknn

4.2.2 Η κλάση Flock

Η κλάση Flock περιέχει εσωτερικά μια λίστα με τους χαρακτήρες που ανήκουν στο σμήνος καθώς επίσης και διάφορες συναρτήσεις που υπολογίζουν τη μέση θέση της γειτονιάς ενός χαρακτήρα και τη μέση ταχύτητα των γειτόνων του. Περιέχει επίσης ένα k-d δέντρο, το οποίο χρησιμοποιεί για τους πιο πάνω υπολογισμούς.

Η δημιουργία του δέντρου γίνεται με τη συνάρτηση `RebuildKdTree()`, η οποία καλείται μία φορά σε κάθε frame από τις κλάσεις κληρονόμους της `MultipleBirdState`, ώστε τα αποτελέσματα της να επαναχρησιμοποιηθούν για κάθε χαρακτήρα τους σμήνους.

4.3 Αναπαράσταση σχημάτων

Όπως προαναφέρθηκε, στη λειτουργία Αναπαράστασης Σχημάτων από boids, τα πουλιά σχηματίζουν κάποιο σχήμα που φορτώνεται από αρχείο που το περιγράφει.

Για τη λειτουργία αυτή, δημιουργήθηκε η συμπεριφορά κατεύθυνσης `PatternSteering`, η κλάση `FormationManager` και η διαπροσωπεία `FormationPattern`.

4.3.1 FormationManager

Κλάση που εσωτερικά αποθηκεύει ένα αριθμό χαρακτήρων (boids). Οι χαρακτήρες αυτοί θα σχηματίζουν το ίδιο pattern. Επίσης, περιέχει μια μεταβλητή τύπου `FormationPattern`, που καθορίζει το pattern που οι χαρακτήρες θα σχηματίσουν.

Η κλάση `FormationManager` κάνει χρήση της `GetSlotPosition()` της κλάσης `FormationPattern` (περιγράφεται στη συνέχεια) για να βρίσκει τη θέση στην οποία πρέπει να κινηθεί ο κάθε χαρακτήρας. Στη συνέχεια, χρησιμοποιώντας τη συμπεριφορά *Seek*, επιστρέφει το διάνυσμα ταχύτητας που πρέπει να έχει ο χαρακτήρας για να κινηθεί πως τη θέση αυτή.

4.3.2 FormationPattern

Διαπροσωπεία που ορίζει ένα pattern. Διαφορετικά patterns κληρονομούν από αυτή και υλοποιούν τις συναρτήσεις που ορίζει. Η λογική λειτουργίας των patterns είναι η εξής: Κάθε χαρακτήρας (boid) ανατίθεται σε κάποιο slot, το οποίο αντιστοιχείται σε κάποια θέση γύρω από ένα κεντρικό σημείο ***center***.

Π.χ:

Όνομα χαρακτήρα	**Αριθμός slot**	**Θέση (σε σχέση με το σημείο *center*)**
boid_0	0	(0, 10, 0)
boid_1	1	(5, 7, 2.5)

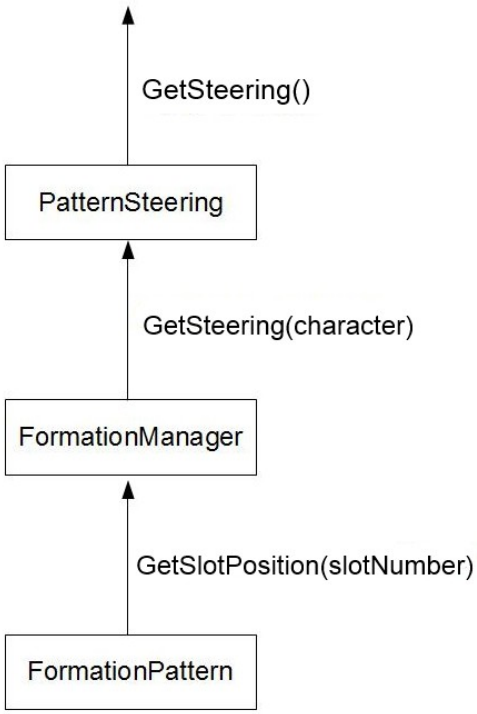
Ο αντιστοιχία των θέσεων στον τρισδιάστατο κόσμο με τον αριθμό του slot γίνεται από τις κλάσεις κληρονόμους της `FormationPattern`.

Η θέση για κάποιο slot μπορεί να ανακτηθεί με χρήση της πιο κάτω συνάρτησης:

```
Vector3 GetSlotPosition(int slotNumber);
```

την οποία υλοποιούν τα διάφορα patterns.

Το ότι η θέση για κάποιο slot είναι σχετική του σημείου *center*, προσφέρει το πλεονέκτημα ότι δε χρειάζεται η ενημέρωση της θέσης κάθε slot όταν αλλάζει το σημείο center. Ως θέση *center* χρησιμοποιείται η θέση του αρχηγού του σμήνους.



Σχήμα 4.5: Χρήση Patterns

4.3.3 PatternSteering

Κλάση που αναπαριστά τη συμπεριφορά pattern. Δέχεται ως παραμέτρους ένα χαρακτήρα (boid) και ένα αντικείμενο `FormationManager`, από το οποίο λαμβάνει το διάνυσμα ταχύτητας που πρέπει να ακολουθήσει ο χαρακτήρας ώστε να κινηθεί στη θέση που του εχει ανατεθεί από το pattern που ο `FormationManager` περιέχει.

4.3.4 ModelFormation

Κλάση που υλοποιεί τη διαπροσωπεία `FormationPattern`. Δέχεται ως είσοδο ένα σύνολο *n* σημείων στον τρισδιάστατο χώρο τα οποία αντιστοιχεί σε αριθμούς από [0, *n*). Τα σημεία αυτά φορτώνονται από αρχεία Wavefront obj.

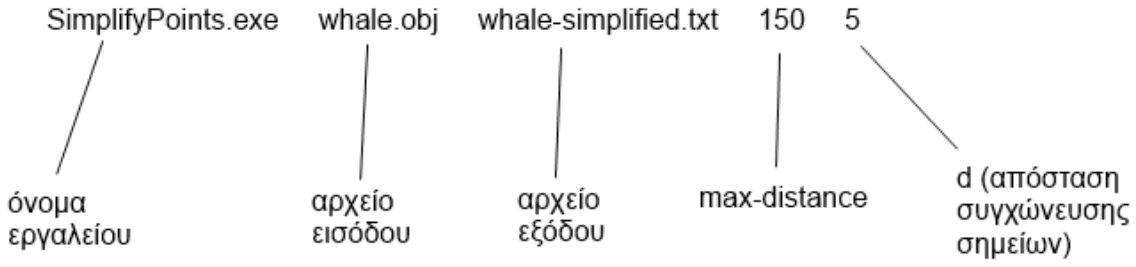
Το .obj είναι μορφή αρχείου που ορίζει γεωμετρία το οποίο αναπτύχθηκε από τη Wavefront Technologies. Η μορφή των δεδομένων είναι απλή, και περιλαμβάνει τη θέση της κάθε κορυφής, τη θέση UV για textures, τις καθέτους των κορυφών και προς τα πού βλέπει κάθε

πολύγωνο (face).

Λόγω του ότι τα αρχεία obj περιέχουν πολύ περισσότερη πληροφορία απ'ότι χρειάζεται το σύστημα (μόνο τις θέσεις των κορυφών), υλοποιήθηκε το εργαλείο `SimplifyPoints`, το οποίο διαβάζει ένα αρχείο obj και παράγει ως έξοδο ένα αρχείο μόνο με τις κορυφές του.

Μπορεί να τυγχαίνει πολλά σημεία σε κάποιο αρχείο .obj βρίσκονται πολύ κοντά το ένα στο άλλο. Αυτό είναι πρόβλημα για το λόγο ότι αν κάποια σημεία είναι πολύ κοντά τότε οι χαρακτήρες που αντιστοιχούν σε αυτά τα σημεία θα είναι πολύ κοντά ο ένας στον άλλο, και πιθανότατα ο ένας *μέσα* στον άλλο. Έτσι, το εργαλείο που αναφέρθηκε προηγουμένως, κάνει μείωση των σημείων που βρίσκονται σε απόσταση *d* συγχονεύοντας τα (η παράμετρος αυτή δίνεται από το χρήστη – για την εφαρμογή αυτή έγινε χρήση $d = 5m$).

Λόγω του ότι κάποια μοντέλα είναι πολύ μεγάλα ενώ άλλα μπορεί να είναι πολύ μικρά, το εργαλείο δέχεται επίσης μία παράμετρο *max-distance*, που καθορίζει τη μέγιστη απόσταση που μπορεί να έχει κάποιο σημείο από το (0, 0, 0). Τα σημεία τροποποιούνται με κατάλληλο τρόπο ώστε το μακρύτερο σημείο να έχει απόσταση *max-distance*.



Σχήμα 4.6: παράδειγμα εκτέλεσης του εργαλείου SimplifyPoints

Algorithm : Point Merge

Input :

- *vertices[]* [Vector3[]]: λίστα σημείων
- *distance* [float]: απόσταση συγχώνευσης

Output : Λίστα με τα συγχνευμένα σημεία

```
1:    procedure point_marge(vertices[], distance)

2:      Για κάθε σημείο vertex
           υπολόγισε πόσα σημεία γύρω από αυτό βρίσκονται
3:      σε ακτίνα d (συμπεριλαμβανομένου του vertex)
4:
5:      Για κάθε γειτονικό σημείο v του vertex σε απόσταση
        distance
6:         διέγραψε το v από τη λίστα των σημείων
7:
        πρόσθεσε στη λίστα τον μέσο όρο των σημείων
        εάν δεν καμία έγινε συγχώνευση τερμάτισε τον
        αλγόριθμο, αλλιώς επανέλαβε τη διαδικασία

8:    end procedure
```

Αλγόριθμος 4.1: Αλγόριθμος συγχώνευσης σημείων Point Merge

Κεφάλαιο 5

Πειραματικές Μετρήσεις

Πίνακας Περιεχομένων

Στο κεφάλαιο αυτό παρουσιάζονται ποσοτικές καθώς και ποιοτικές μετρήσεις που έγιναν στο σύστημα που αναπτύχθηκε. Συγκεκριμένα, η λειτουργία Περιπλάνησης θεωρήθηκε ως καταλληλότερη για διεξαγωγή των πειραμάτων, καθώς αυτή είναι που έχει ως στόχο τη ρεαλιστική αναπαράσταση σμηνών.

Οι ποσοτικές μετρήσεις αφορούν στοιχεία όπως FPS (frames per second) και ρυθμό μείωσης τους όσο αυξάνεται το μέγεθος του προβλήματος, ενώ οι ποιοτικές μετρήσεις έχουν να κάνουν με το πόσο ρεαλιστικό φαίνεται το αποτέλεσμα στο χρήστη, καθώς και το πώς συμπεριφέρονται τα πουλιά στο σμήνος.

Οι παραμέτροι για τις μετρήσεις είναι:

Γενικές παραμέτροι	
boids	Αριθμός χαρακτήρων
Update%	Ποσοστό χαρακτήρων για τους οποίους ανανεώνονται οι πληροφορίες για τους γείτονες τους σε κάθε frame. Οι υπόλοιποι χρησιμοποιούν τα αποτελέσματα που υπολογίστηκαν σε προηγούμενα frames, μέχρι να έρθει η σειρά τους να ανανεωθούν
Παραμέτροι συμπεριφορών κατεύθυνσης (Cohesion, Separation, Alignment)	
Weight	Βαρύτητα της συμπεριφοράς. Όσο μεγαλύτερη τόσο περισσότερο το αποτέλεσμα της επηρεάζει το τελικό αποτέλεσμα
k	Αριθμός k κοντινότερων γειτόνων
Radius	Ακτίνα στην οποία χαρακτήρες θεωρούνται γείτονες (από τους k κοντινότερους)
Angle	Ορίζει τη μέγιστη γωνία στην οποία οι γείτονες ενός χαρακτήρα λαμβάνονται υπόψη. Εάν ισούται με 360^{o} τότε ο χαρακτήρας λαμβάνει υπόψη του και γείτονες που είναι πίσω του, ενώ αν για παράδειγμα είναι 180^{o}, ο λαμβάνει υπόψη μόνο τους γείτονες του που είναι μπροστά του (90^{o} στα αριστερά και 90^{o} στα δεξιά).
Aknn-approx	Παράμετρος προσέγγισης για τον αλγόριθμο aknn. Όταν είναι 0 γίνεται χρήση του knn, ενώ όσο μεγαλύτερες τιμές χρησιμοποιούνται τόσο πιο ανακριβής είναι ο υπολογισμός των κοντινότερων γειτόνων,

	με βελτιώσεις στην επίδοση

5.1 Ανανέωση πληροφοριών γειτονιάς ανά frame

Το πείραμα αυτό έχει ως στόχο τη μέτρηση του κατά πόσο το αποτέλεσμα αλλάζει οπτικά με την αλλαγή της παραμέτρου Update%. Όταν ισούται με 100%, υπολογίζονται οι γείτονες για κάθε πουλί στο σμήνος σε *κάθε* frame, ενώ όταν ισούται με 20% για παράδειγμα, στο frame 0 ενημερώνονται τα πρώτα 20% των πουλιών, ενώ για το υπόλοιπο 80% χρησιμοποιείται η προηγούμεν τιμή που υπολογίστηκε. Στο επόμενο frame, θα ενημερωθεί το επόμενο 20% κτλ. Άρα χρειάζονται συνολικά 5 frames για να ανανεωθούν όλα τα πουλιά στο σμήνος. Επίσης μετρήθηκε ο ρυθμός μείωσης της επίδοσης όσο αλλάζει ο αριθμός των πουλιών καθώς και το ποσοστό ενημέρωσης τους ανά frame.

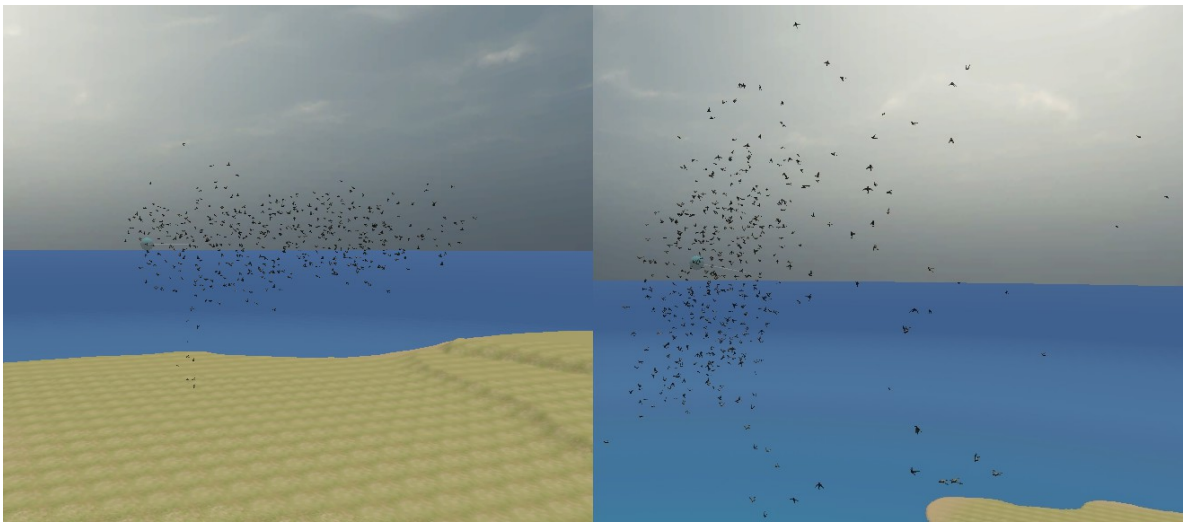
Αυτό επιφέρει μεγάλη βελτίωση στην επίδοση, με το μειονέκτημα πως εάν το ποσοστό είναι πολύ χαμηλό, η συμπεριφορά των πουλιών να μη φαίνεται “σωστή”. Στόχος είναι η εύρεση μιας ικανοποιητικής τιμής για διάφορα μεγάθη προβλήματος (αριθμός πουλιών) ώστε η επίδοση να είναι καλή ενώ παράλληλα το αποτέλεσμα να είναι ικανοποιητικό.

Για τις συμπεριφορές χρησιμοποιήθηκαν οι εξής παραμέτροι:

	Cohesion	**Separation**	**Alignment**	**Cohesion to Leader**
Weight	1.0	2.0	0.25	0.25
k	10	5	5	1
Radius	30m	5m	20m	∞
Angle	180^o	180^o	180^o	360^o
Aknn-approx	1.0	1.0	25.0	0.0

Για τη συμπεριφορά Alignment η παράμετρος Aknn-approx παίρνει τιμή πολύ μεγάλη σε σχέση με τις δύο άλλες συμπεριφορές. Αυτό συμβαίνει επειδή παρατηρήθηκε πως η αλλαγή των τιμών της δεν επέφερε ορατές αλλαγές ενώ επέφερε μικρές βελτιώσεις στην επίδοση. Η συμπεριφορά Cohesion to Leader χρησιμοποιείται ώστε τα πουλιά να μην απομακρύνονται από τον αρχηγό του σμήνους.

Όταν ο αριθμός των πουλιών είναι μέχρι 300, η διαφορά στη στη συμπεριφορά των πουλιών ακόμα και με ενημέρωση του 20% δεν είναι εύκολα ορατή, άρα η τιμή αυτή κρίνεται ικανοποιητική. Στα 500 όμως, η συμπεριφορά τους αλλάζει δραματικά – υπάρχει συνεχής κίνηση των πουλιών γύρω από τον αρχηγό, ενώ άλλα περιφέρονται σε μια απόσταση από αυτόν. Επίσης, το σμήνος από οριζόντιο γίνεται κάθετο!



Σχήμα 5.1: 500 πουλιά, 60% ενημέρωση (αριστερά) ένταντι 20% ενημέρωση (δεξιά) σε κάθε frame. Το σμήνος από οριζόντιο γίνεται κάθετο.

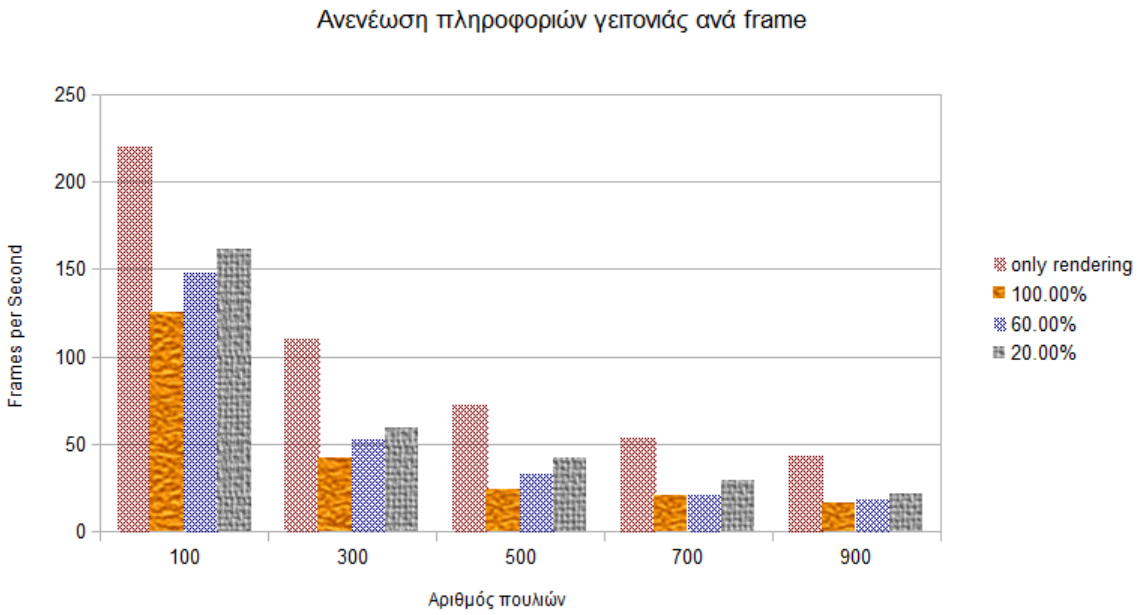
Με 700 πουλιά, φαίνεται μια μικρή διαφορά από την 100% ενημέρωση έναντι της 60%. Το σμήνος γίνεται ελάχιστα λίγο πιο αραιό καθώς και υπάρχει περισσότερη κίνηση σε αυτό. Με 20% ενημέρωση, όπως και στην περίπτωση των 300 πουλιών, υπάρχει συνεχής κίνηση και γύρω από τον αρχηγό καθώς και μετατροπή του σμήνους από οριζόντιο σε κάθετο, σε μεγαλύτερο όμως βαθμό.

Όταν ο αριθμός των πουλιών φτάσει τα 1000, η επίδοση έχει μειωθεί πλέον πάρα πολύ ακόμα και με ενημέρωση του 20% των πουλιών σε κάθε frame.

Στο σχήμα 5.2 βλέπουμε πώς μεταβάλλονται τα frames ανά δευτερόλεπτο ανάλογα με το αριθμό καθώς και με το ποσοστό των πουλιών που ενημερώνονται ανά frame. Από τις

μετρήσεις φαίνεται πως στα 300 πουλιά, με 60% ανανέωση το frame rate είναι πολύ ικανοποιητικά επίπεδα (51 fps).

Επίσης, στη γραφική παράσταση φαίνεται το μέγιστο frame rate για κάθε αριθμό πουλιών που μπορούμε να επιτύχουμε. Η μέτρηση έγινε μόνο με rendering των πουλιών και όχι προσομοίωση κάποιας συμπεριφοράς.



Σχήμα 5.2: Μέτρηση επίδοσης ανάλογα με τον αριθμό καθώς και με το ποσοστό πουλιών για τα οποία ενημερώνονται οι γείτονες ανά frame

Από τις μετρήσεις φαίνεται πως τα αποτελέσματα είναι αρκετά καλά σε σχέση με την καλύτερη περίπτωση (only rendering).

5.2 Γωνιά γειτονιάς – με χρήση Cohesion to Leader

Σε αυτό το πείραμα παρατηρήθηκε η συμπεριφορά των πουλιών καθώς και το σχήμα του σμήνους όταν άλλαζε η γωνία στην οποία το κάθε πουλί λαμβάνει υπόψη άλλα πουλιά ως γείτονες του. Χρησιμοποιήθηκαν συνδυασμοί των 360^{o}, 240^{o}, 180^{o} και 120^{o} για κάθε συμπεριφορά.

Οι παραμέτροι που χρησιμοποιήθηκαν είναι:

	Cohesion	**Separation**	**Alignment**	**Cohesion to Leader**
Weight	1.0	2.0	0.25	0.25
k	10	5	5	1
Radius	30m	5m	20m	∞
Angle				360
Aknn-approx	1.0	1.0	25.0	0.0

Η συμπεριφορά Cohesion to Leader προκαλεί τα πουλιά του σμήνους να κινούνται προς τον αρχηγό. Οι παραμέτροι της διατηρήθηκαν σταθερές κατά τη διεξαγωγή του πειράματος.

5.2.1 Πείραμα 1: Cohesion 360, Separation 360, Alignment 360

Όταν οι γωνίες είναι 360^{o} και για τις τρεις συμπεριφορές, ο κάθε χαρακτήρας λαμβάνει υπόψη του χαρακτήρες που βρίσκονται σε μια σφαίρα γύρω από αυτόν. Ως εκ τούτου, το σμήνος παίρνει μορφή σφαίρας γύρω από τον αρχηγό.

Σχήμα 5.3: Πείραμα 1: Όταν οι γωνίες είναι 360 μοίρες και για τις τρεις συμπεριφορές, το σμήνος παίρνει τη μορφή σφαίρας γύρω από τον αρχηγό.

5.2.2 Πείραμα 2: Cohesion 360, Separation 360, Alignment 180

Σε αυτή την περίπτωση οι γωνίες για τις συμπεριφορές Cohesion και Separation είναι 360 ενώ για την Alignment είναι 180. Αυτό σημαίνει πως κάποιος χαρακτήρας θα τείνει να κινηθεί προς την κατεύθυνση που κινούνται οι γείτονες που βρίσκονται μπροστά του (και όχι και πίσω του, όπως στην προηγούμενη περίπτωση). Οπτικά αυτό ωθεί το σμήνος να κινείται σε σχήμα παραλληλόγραμμου, ειδικά όταν υπάρχει αλλά στην κατεύθυνση του αρχηγού. Αν όμως ο αρχηγός δεν αλλάζει κατεύθυνση, το σχήμα γίνεται σφαίρα όπως και πριν.

Σχήμα 5.4: Πείραμα 2: Γωνίες 360 για Cohesion και Sepation και 180 για Alignment

5.2.3 Πείραμα 3: Cohesion 120, Separation 180, Alignment 240

Σχήμα 5.5: Πείραμα 3: Υποσμήνος απομακρύνεται προσωρινά από το κεντρικό σμήνος. Η τροχιά που ακολουθεί φαίνεται με τη λευκή καμπύλη

Με παραμέτρους 120, 180 και 240 μοίρες για τις συμπεριφορές Cohesion, Separation και Alignment αντίστοιχα, το σμήνος εκδηλώνει την εξής συμπεριφορά: Σε τακτά χρονικά διαστήματα, ένα μικρότερο σμήνος (υποσμήνος) "ξεφεύγει" από το κεντρικό σμήνος και κινείται προς μια άλλη κατεύθυνση. Λόγω όμως τη συμπεριφοράς Cohesion προς τον αρχηγό του σμήνους, το υποσμήνος αυτό επιστρέφει και γίνεται ένα με το μεγαλύτερο σμήνος.

5.2.4 Πείραμα 4: Cohesion 120, Separation 120, Alignment 120

Με τη διαρρύθμιση αυτή (120^{o} και για τις 3 συμπεριφορές) το αποτέλεσμα μοιάζει με αυτό του πειράματος 3: συχνά, μικρά υποσμήνη ξεφεύγουν από το σμήνος, πετούν μόνα τους προς διαφορετική κατεύθυνση από αυτό και στη συνέχεια επιστρέφουν. Η διαφορά σε αυτή

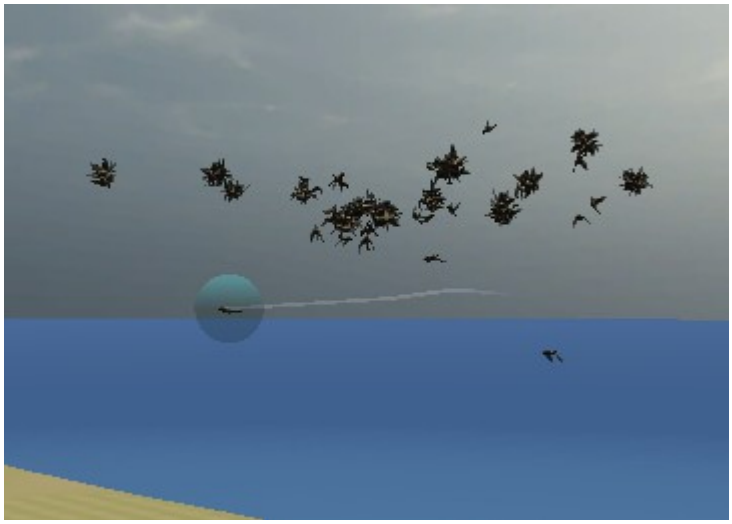
την περίπτωση είναι πως τα υποσμήνη που σχηματίζονται τείνουν να απομακρύνονται πολύ περισσότερο απ' ότι πρηγουμένως.

Σχήμα 5.6: Πείραμα 4: Όταν οι γωνίες για όλες τις συμπεριφορές είναι 120 μοίρες, δημιουργούνται υποσμήνη που απομακρύνονται πολύ από το σμήνος πρωτού επιστρέψουν σε αυτό.

5.2.5 Πείραμα 5: Cohesion 240, Separation 120, Alignment 120

Σε όλα τα προηγούμενα πειράματα, η γωνία για τη συμπεριφορά Separation ήταν μεγαλύτερη ή ίση από τη γωνία της συμπεριφοράς Cohesion. Η λεπτομέρεια αυτή είναι σημαντική, επειδή σε αντίθετη περίπτωση, τα πουλιά στο σμήνος θα προσπαθούν περισσότερο να κινηθούν προς του πολύ κοντινούς τους γείτονες αντί να απομακρύνονται από αυτούς. Αυτό έχει ως αποτέλεσμα πολλά πουλιά να προσπαθούν να κινηθούν προς κάποιο σημείο, με το αποτέλεσμα το ένα να μπαίνει μέσα στο άλλο, όπως φαίνεται στην

εικόνα.



Σχήμα 5.7: Πείραμα 5: Όταν η γωνία της Separation είναι μικρότερη από της Cohesion, οι πολύ κοντινοί γείτονες αντί να απομακρύνονται ο ένας από τον άλλο, κινούνται ο ένας προς τον άλλο.

5.3 Μορφή σμήνους

Στο πείραμα αυτό παρουσιάζεται πώς αλλάζει η μορφή του σμήνους μεταβάλλοντας τις διάφορες παραμέτρους των συμπεριφορών Cohesion, Separation και Alignment. Επίσης, γίνεται χρήση των συμπεριφορών Cohesion to Leader (τα πουλιά να κινούνται προς το σημείο που βρίσκεται ο αρχηγός), Separation from Leader (τα πουλιά να απομακρύνονται από το σημείο που βρίσκεται ο αρχηγός) και Alignment to Leader (τα πουλιά να κινούνται με το ίδιο διάνυσμα ταχύτητας με τον αρχηγό).

5.3.1 Υποσμήνη σε μορφή σφαίρας

Με τις ακόλουθες παραμέτρους το σμήνος χωρίζεται σε αριθμό υποσμηνών, που έχουν μορφή σφαίρας. Οι συμπεριφορές Cohesion to Leader και Separation from Leader δε χρησιμοποιήθηκαν.

	Cohesion	Separation	Alignment	Alignment to Leader
Weight	1.0	2.0	0.25	0.5
k	10	5	5	1
Radius	30m	5m	20m	∞
Angle	240	240	240	360
Aknn-approx	1.0	1.0	25.0	0.0

Σχήμα 5.8: Το σμήνος χωρίζεται σε μικρότερα υποσμήνη που έχουν μορφή σφαίρας.

5.3.2 Σμήνος σε μορφή ημισφαιρίου

Με τη χρήση των Cohesion to Leader και Separation from Leader, με τις παραμέτρους που φαίνονται πιο κάτω, το σμήνος παίρνει τη μορφή ημισφαιρίου πίσω από τον αρχηγό. Αυτό συμβαίνει επειδή τα πουλιά προσπαθούν να σχηματίσουν σφαίρα γύρω από τον αρχηγό λόγω της συμπεριφοράς Cohesion to Leader, αλλά η Separation to Leader τα αποτρέπει από το να πλησιάσουν (μιας και έχει μεγαλύτερο βάρος από την προηγούμενη) με αποτέλεσμα να σχηματίζουν ημισφαίριο. Η τιμή για την παράμετρο radius της Separation from Leader καθορίζει ουσιαστικά την ακτίνα του ημισφαιρίου.

	Cohesion	Separation	Alignment	Cohesion to Leader	Separation from Leader
Weight	1.0	2.0	0.25	0.5	2
k	10	5	5	1	1
Radius	30m	5m	20m	∞	25m
Angle	240	240	240	360	360
Aknn-approx	1.0	1.0	25.0	0.0	0.0

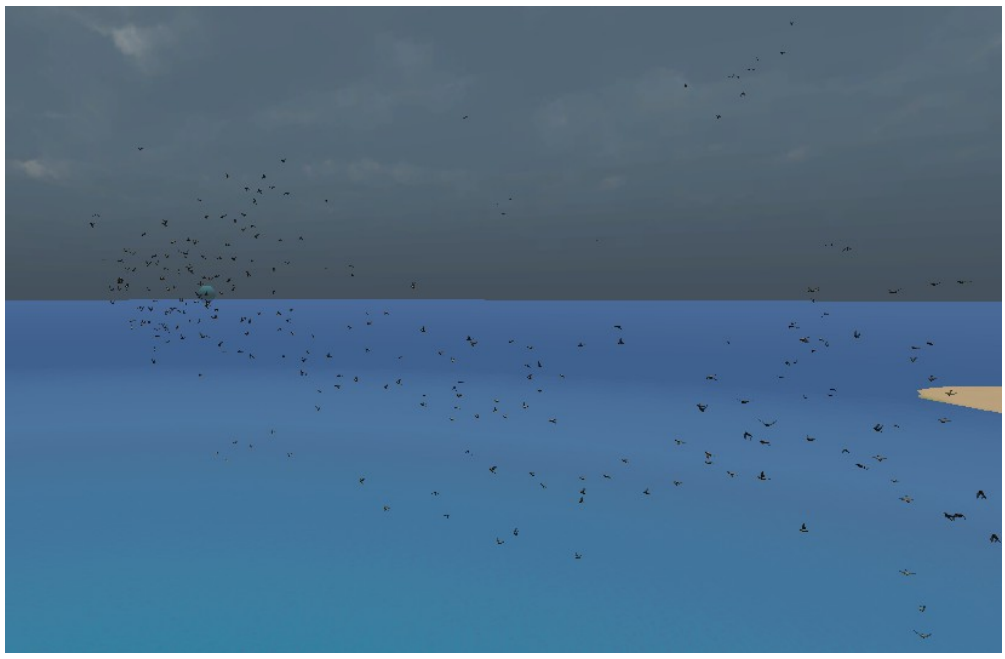
Σχήμα 5.9: Σχηματισμός ημισφαιρίου

5.3.3 Ακολουθίες πουλιών

Με τις ακόλουθες παραμέτρους μεγάλο μέρος του σμήνος σχηματίζει "ουρές" πίσω από τον αρχηγό.

	Cohesion	**Separation**	**Alignment**	**Cohesion to Leader**
Weight	1.0	2.0	0.25	0.25
k	10	5	5	1
Radius	30m	5m	20m	∞
Angle	180	240	180	360
Aknn-approx	1.0	1.0	25.0	0.0

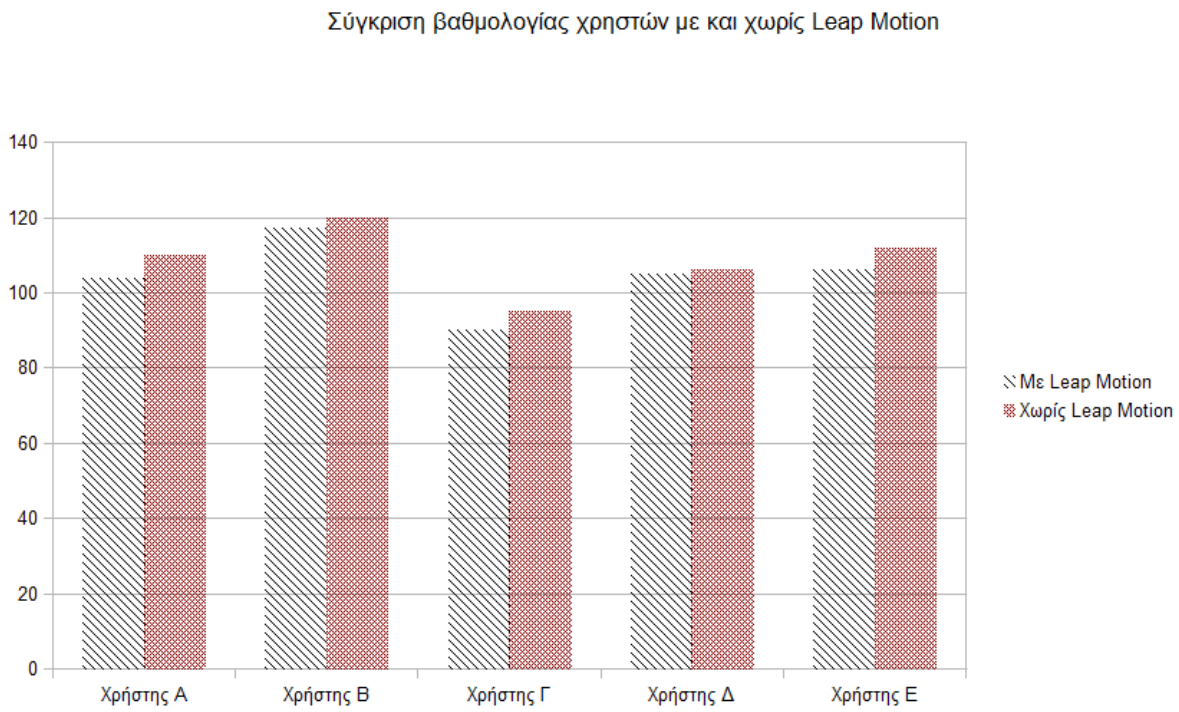
Αυτό συμβαίνει για το λόγο ότι η γωνία της συμπεριφοράς Cohesion (και Alignment σε μικρότερο βαθμό) είναι 180^{o}, με αποτέλεσμα τα πουλιά να ακολουθούν τους μπροστινούς τους γείτονες, κι έτσι να σχηματίζονται ουρές.



Σχήμα 5.10: Σχηματισμός ουρών από πουλιά πίσω από τον αρχηγό.

5.4 Πείραμα χρήσης Leap Motion

Στο πείραμα αυτό έγινε η σύγκριση των της βαθμολογίας χρηστών στο παιχνίδι που υλοποιήθηκε με και χωρίς τη χρήση της συσκευής leap motion.



Σχήμα 5.11: Σύγκριση βαθμολογίας χρηστών με και χωρίς Leap Motion

Φαίνεται πως γενικά οι βαθμολογίες με τη χρήση της συσκευής Leap Motion είναι λίγο χαμηλότερες απ'ότι χωρίς τη χρήση της. Αυτό οφείλεται στο ότι οι χρήστες δεν ήταν εξοικιωμένοι με τη συσκευή, ενώ αντίθετα ήταν συνηθισμένοι στη χρήση του πληκτρολογίου για το παίξιμο παιχνιδιών.

Παρόλα αυτά ήταν περισσότερο ικανοποιημένοι με τη χρήση του Leap Motion. Βάσει των σχολίων τους φάνηκε ότι άρεσε ο έλεγχος του παιχνιδιού μέσω της κίνησης του χεριού τους (χωρίς να χρειάζεται δηλαδή να πατούν πλήκτρα). Κατά την αλληλεπίδραση τους με τη συσκευή θέωρησαν πιο ρεαλιστικό αυτό το σενάριο χρήσης αφού όταν κινούσαν το χέρι τους ο παίκτης ανταποκρινόταν ανάλογα (π.χ με την κίνηση του χεριού δεξιά ο παίκτης

περιστρεφόταν προς τα δεξιά).

Κεφάλαιο 6

Συμπεράσματα – Μελλοντική εργασία

Πίνακας Περιεχομένων

6.1 Συμπεράσματα

Οι λειτουργίες που υλοποιήθηκαν στην παρούσα διπλωματική εργασία, αν και η καθεμιά είναι αρκετά διαφορετική από τις άλλες και με διαφορετικό στόχο (προσομοίωση συμπεριφορών σμήνους, αναπαράσταση σχημάτων και δημιουργία ενός απλού παιχνιδιού), έχουν ως κοινό τη χρήση συνδυασμού των συμπεριφορών κατεύθυνσης του Reynolds.
Αυτό δείχνει πως το μοντέλο αυτό είναι αρκετά ευέλικτο και μπορεί για να χρησιμοποιηθεί για διαφόρων ειδών εφαρμογές, με πολύ μικρές τροποποιήσεις στις παραμέτρους των συμπεριφορών.

6.2 Μελλοντική Εργασία

Μελλοντικά θα μπορούσαν να γίνουν βελτιώσεις στον υπολογισμό των k κοντινότερων γειτόνων. Στην παρούσα φάση γίνεται χρήση K-D δέντρου αλλά δεν αξιοποιούνται τυχόν επιπλέον threads που διαθέτει η μηχανή. Η χρήση threads για την ανάκτηση των κοντινότερων γειτόνων αλλά και για το κτίσιμο του K-D δέντρου αναμένεται πως θα επιφέρει σημαντικές βελτιώσεις στην επίδοση, και θα επιτρέψει μεγαλύτερους αριθμούς χαρακτήρων.
Επίσης θα ήταν ενδιαφέρον η μελέτη και ενσωμάτωση στο σύστημα επιπλέον συμπεριφορών κατεύθυνσης – π.χ η πιθανότητα κάποιοι χαρακτήρες να ξεφύγουν από το σμήνος καθώς και ομάδες χαρακτήρων να έχουν τη δική τους “προσωπικότητα”, δηλαδή

παραμέτρους διαφορετικές από υπόλοιπες ομάδες χαρακτήρων.

Τέλος, η δημιουργία διαφορετικών τύπων πουλιών θα επέτρεπε τη δημιουργία πολύπλοκων συνδυασμών από συμπεριφορές. Για παράδειγμα, κάποιο είδος πουλιών Α θα μπορούσε να "κυνηγάει" πουλιά είδους Β, ενώ τα δεύτερα να προσπαθούν να ξεφύγουν. Ή όταν ένα πουλί είδους Γ πλησιάσει κάποιο είδους Δ, να αλλάξει τις παραμέτρους του ώστε να έρθει πιο κοντά σε πουλιά του είδους του για προστασία. Οι συνδυασμοί που μπορούν να γίνουν είναι πάρα πολλοί.

Αναφορές

[1] Unity3D Game Engine https://unity3d.com/

[2] Leap Motion https://www.leapmotion.com/

[3] Artificial Intelligence for Games by Ian Millington

[4] Reynolds CW. Flocks, herds and schools: a distributed behavioral model

[5] k-d trees http://en.wikipedia.org/wiki/K-d_tree

[6] Delgado-Mata, Carlos; Martinez, Jesus Ibanez; Bee, Simon; Ruiz-Rodarte, Rocio; Aylett, Ruth (2007). “On the use of Virtual Animals with Artificial Fear in Virtual Environments”

[7] Hartman, Christopher; Beneš, Bedřich (July 2006). "Autonomous boids". Computer Animation and Virtual Worlds

[8] K-Nearest Neighbors algorithms
http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Παράρτημα Α

Ο κώδικας βρίκεται στο CD, στο φάκελο code.