ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη τρισδιάστατου παιχνιδιού ανοιχτού περιβάλλοντος με αυτονομία αντιπάλων

Διπλωματική εργασία

Φράγκος Νικόλαος

Εξεταστική επιτροπή

Επ.Καθ. Αικατερίνη Μανιά (επιβλέπουσα)

Επ.Καθ. Μιχαήλ Γ. Λαγουδάκης

Επ.Καθ. Αντώνιος Δεληγιαννάκης

Χανιά 2010

Περίληψη

Η παρούσα διπλωματική πραγματεύεται την ανάπτυξη ενός ολοκληρωμένου ηλεκτρονικού παιχνιδιού για υπολογιστές με την χρήση του προγραμματιστικού περιβάλλοντος Unity 3D. Το παιχνίδι εντάσσεται στην κατηγορία "βολών αυτοκινήτου" (vehicular combat). Μέσα σε μια ειδικά διαμορφωμένη πίστα τοποθετούνται οκτώ αυτοκίνητα τα οποία μπορούν να εξοπλιστούν με πύραυλους. Ο χρήστης έχοντας τον έλεγχο ενός εκ των οχημάτων προσπαθεί να εξοντώσει τα υπόλοιπα.

Η ανάπτυξη της εν λόγω εφαρμογής είναι ένα εγχείρημα το οποίο προϋποθέτει έρευνα σε πολλούς τομείς. Αρχικά, για να λειτουργήσει μια γεωμετρία ως αυτοκίνητο θα πρέπει να υπακούει στους αντίστοιχους νόμους της φυσικής. Για την δημιουργία ενός τέτοιου μοντέλου εξομοίωσης απαιτείται μελέτη των δυνάμεων που ασκούνται στους τροχούς των πραγματικών αυτοκινήτων. Η τριβή καθορίζει το κράτημα στο οδόστρωμα, η ροπή την ταχύτητα, και η κάθετη δύναμη την συμπεριφορά κατά την στρέψη των τροχών. Εκτός όμως από την εξομοίωση τα οχήματα που ελέγχονται από τον υπολογιστή χρειάζονται τεχνητή νοημοσύνη. Με αυτόν τον τρόπο θα είναι σε θέση να ανταποκριθούν στους κανόνες του παιχνιδιού. Τα αυτοκίνητα μπορούν να βρίσκονται σε μια από τις εξής καταστάσεις: καταδίωξη αντίπαλου, αποφυγή αντίπαλου, εξόπλιση πυραύλων, εξόπλιση υγείας. Εάν τα οχήματα βρίσκονται συνέχεια στην κατάσταση «καταδίωξη» το παιχνίδι γίνεται πολύ δύσκολο για τον χρήστη, ενώ εάν βρίσκονται συνέχεια στην «αποφυγή» είναι πολύ εύκολο. Για να υπάρξει ισορροπία στις αποφάσεις που παίρνει ο υπολογιστής χρησιμοποιήθηκαν δύο μέθοδοι. Η πρώτη είναι μια μηχανή πεπερασμένων καταστάσεων και η δεύτερη είναι ένα σύστημα ασαφούς λογικής. Η διαφορά τους έγκειται στον τρόπο μετάβασης από τη μια κατάσταση στην άλλη. Η μηχανή πεπερασμένων καταστάσεων χρησιμοποιεί συγκεκριμένες λογικές συνθήκες. Για αυτό το λόγο οι αποφάσεις της καθίστανται προβλέψιμες και κατά συνέπεια και η συμπεριφορά των αντιπάλων. Η προβλέψιμη συμπεριφορά θα προκαλέσει δυσαρέσκεια στον χρήστη. Αντιθέτως η ασαφής λογική παράγει απρόβλεπτες αποφάσεις καθώς χρησιμοποιεί λεκτικούς όρους και ποσοστά. Πριν ξεκινήσει κάθε αγώνας ο χρήστης είναι σε θέση να διαλέξει την μέθοδο αποφάσεων που θα ακολουθήσει ο υπολογιστής.

Επίσης στους κανονισμούς του παιχνιδιού προστέθηκε μια πρωτοτυπία για αυτήν την κατηγορία παιχνιδιών. Κάθε τριάντα δευτερόλεπτα εμφανίζεται, σε τυχαία θέση, μέσα στην πίστα μία σημαία. Όλα τα οχήματα υπολογίζουν την βέλτιστη διαδρομή από την θέση που βρίσκονται μέχρι την σημαία, έτσι ώστε να την προσεγγίσουν όσο το δυνατόν γρηγορότερα. Το όχημα που θα φτάσει πρώτο κερδίζει πόντους, ενώ το τελευταίο αποκλείεται από το παιχνίδι. Η εύρεση του καλύτερου\γρηγορότερου μονοπατιού γίνεται με τον αλγόριθμο A*. Για να λειτουργήσει ο A*, χρειάζεται ένα σύνολο σημείων, μια αρχική και μια τελική θέση. Επειδή το παιχνίδι δεν διαθέτει ένα περιορισμένο περιβάλλον με κουτάκια όπως για παράδειγμα το σκάκι, θα πρέπει να

δημιουργηθεί τεχνητό πλέγμα. Όταν κληθεί ο αλγόριθμος, σε κάθε αυτοκίνητο αντιστοιχίζεται μια αρχική θέση στο πλέγμα. Ως τελική θέση θεωρείται πάντα η θέση της σημαίας. Διαθέτοντας αυτά τα δεδομένα ο A* είναι σε θέση να προσδιορίσει την συντομότερη διαδρομή μεταξύ αυτοκινήτου και σημαίας.

Τέλος, εκτός από το προγραμματιστικό κομμάτι δημιουργήθηκε μια γραφική διεπαφή για το παιχνίδι. Τα μενού και οι εικόνες που τα πλαισιώνουν σχεδιάστηκαν με σκοπό την εύκολη και ευχάριστη πλοήγηση του χρήστη.

Περιεχόμενα

Πίνακας Εικόνων

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγή

Στόχος της παρούσας διπλωματικής ήταν η υλοποίηση ενός τρισδιάστατου παιχνιδιού για υπολογιστές με έμφαση στην δημιουργία αυτόνομων αντιπάλων. Ο όρος “ανοικτού περιβάλλοντος” που βρίσκεται στον τίτλο αναφέρεται στο γεγονός ότι ο χώρος του παιχνιδιού είναι ελευθέρα προσβάσιμος από τον χρήστη, δεν υπάρχουν δηλαδή περιορισμοί στις θέσεις που μπορεί να καταλαμβάνει ο χρήστης, όπως συμβαίνει σε ένα παιχνίδι σκάκι ή Othello. Ο όρος “αυτόνομοι αντίπαλοι” αναφέρεται στο γεγονός ότι οι αντίπαλοι στο παιχνίδι μας είναι ικανοί να πάρουν την καλύτερη απόφαση για την οντότητα τους με βάση τις εκάστοτε συνθήκες του παιχνιδιού. Το εν λόγω παιχνίδι μπορεί να κατηγοριοποιηθεί στα παιχνίδια βολών αυτοκινήτων (vehicular combat) [10], στα παιχνίδια της οποίας το κέντρο του ενδιαφέροντος είναι τα αυτοκίνητα ή άλλου τύπου οχήματα. Αυτά είναι εξοπλισμένα με διάφορα οπλικά συστήματα με τα οποία προσπαθούμε να καταστρέψουμε τα οχήματα που ελέγχονται από τον υπολογιστή ή από άλλους παίκτες. Συνήθως υπάρχει πληθώρα διαθέσιμων οχημάτων κάθε ένα με τις δικές του ιδιότητες. Παραδοσιακά στηρίζονται στην γρήγορη εξέλιξη και έντονη δράση του αγώνα και όχι τόσο στην υπόθεση.

1.2 Περιγραφή παιχνιδιού

Στην παρούσα υλοποίηση, έχουμε ένα σύνολο αυτοκινήτων τα οποία είναι διαθέσιμα στον χρήστη, ο οποίος επιλέγει το αυτοκίνητο που προτιμά και μπορεί να τρέξει σε έναν αγώνα. Οι προϋποθέσεις υπό τις οποίες κερδίζει ο χρήστης είναι να εξολοθρεύσει όλους τους αντιπάλους με κάθε δυνατό μέσο. Μπορεί δηλαδή να τους καταστρέψει είτε με βολές πυραύλων, είτε με το να συγκρουστεί με δύναμη πάνω τους. Στην δεύτερη περίπτωση βέβαια, είναι πολύ πιθανό ότι και το αυτοκίνητο του χρήστη θα υποστεί φθορές.

Για λόγους πρωτοτυπίας στο παιχνίδι, προστέθηκε ένα επιπλέον χαρακτηριστικό. Κάθε 30 δευτερόλεπτα μέσα στην πίστα εμφανίζεται μια σημαία την οποία επιδιώκουν να πάρουν όλα τα συναγωνιζόμενα αυτοκίνητα. Όταν ο χρήστης ή κάποιος από τους αντιπάλους του καταφέρει να πάρει τη σημαία, τότε το αυτοκίνητο το οποίο απέχει τη μεγαλύτερη Ευκλείδεια απόσταση από τη σημαία, αυτομάτως καταστρέφεται. Με αυτόν τον τρόπο το παιχνίδι γίνεται περισσότερο ανταγωνιστικό και ενδιαφέρον.

Στα παιχνίδια αυτού του τύπου οι πίστες δεν έχουν την κλασσική δομή μιας πίστας αγώνων. Δεν υπάρχει προκαθορισμένη διαδρομή, αλλά ένα ανοικτό περιβάλλον που θυμίζει αρένα, στο οποίο έχουν προστεθεί ποικίλα εμπόδια όπως κτήρια, βράχια κ.α. προκειμένου να γίνει ελκυστικότερο το περιβάλλον. Η πίστα σχεδιάστηκε από το μηδέν, με την βοήθεια προγραμμάτων μοντελοποίησης.

Ο χρήστης μπορεί να επιλέξει μεταξύ πληθώρας αυτοκινήτων, τα οποία διαφέρουν στο χρώμα και το έμβλημα, δεν υπάρχει όμως διαφοροποίηση στις επιδόσεις ή στις δυνατότητες τους. Επίσης η γεωμετρία είναι ίδια για όλα τα αυτοκίνητα και μάλιστα από τρίτο δημιουργό.

Διάσπαρτα μέσα στην πίστα υπάρχουν "κουτιά" τα οποία δίνουν στο χρήστη βοήθεια, με την έννοια ότι αναπληρώνουν την ενέργεια και αυξάνουν ή αναβαθμίζουν τον αριθμό των όπλων του χρήστη. Όταν ένας χρήστης πάρει μία αναβάθμιση, αυτή είναι ξανά διαθέσιμη μετά από κάποια δευτερόλεπτα.

Οι πιθανές εκδοχές τερματισμού του παιχνιδιού είναι τρείς. Κατά την πρώτη, ο χρήστης θα χάσει το παιχνίδι εάν το αυτοκίνητο του έχει την μεγαλύτερη ευκλείδεια απόσταση από την σημαία, συγκριτικά με τα υπόλοιπα αυτοκίνητα. Η απόσταση μετράται την στιγμή που κάποιο όχημα θα φτάσει πρώτο στην σημαία. Επίσης αποκλεισμός μπορεί να συμβεί εάν μηδενιστούν οι πόντοι ενέργειας/υγείας του χρήστη. Η τελευταία εκδοχή είναι ο χρήστης να εξοντώσει όλους τους αντιπάλους του. Υπολογίζεται μια βαθμολογία, η οποία εξαρτάται από διάφορους παράγοντες που θα αναλυθούν εκτενέστερα σε παρακάτω κεφάλαιο, η οποία αποθηκεύεται σε ένα πίνακα στον οποίο κρατούνται οι υψηλότερες βαθμολογίες.

1.3 Δομή της εργασίας

Η δομή της εργασίας είναι η εξής:

Στο δεύτερο κεφάλαιο παρουσιάζεται η διοχέτευση υλικού στις κάρτες γραφικών καθώς και οι βιβλιοθήκες OpenGL και DirectX. Αναλύεται επίσης η απεικόνιση της γεωμετρίας στην οθόνη του υπολογιστή, η χρήση σκιαστών, υφών καθώς και τα βασικά φώτα (point, spot, directional). Στην συνέχεια αναλύεται η θεωρία γύρω από τις μηχανές γραφικών, φυσικής και παιχνιδιών και παρουσιάζουμε τις μηχανές Source, Ego, UDK, FIFE, Panda3D. Επιπλέον, γίνεται αναφορά σε τεχνικές λήψης αποφάσεων με την χρήση μηχανών πεπερασμένων καταστάσεων και ασαφούς λογικής. Τέλος παρουσιάζουμε τον αλγόριθμο Α* και τη βελτίωση του με χρήση ουράς προτεραιότητας αντί συνδεδεμένης λίστας.

Στο τρίτο κεφάλαιο αναλύονται τα συστατικά της μηχανής παιχνιδιών Unity 3D. Αρχικά γίνεται μία γενική αποτύπωση της οργάνωσης των αντικειμένων σε στοιχεία (Components) και στην δόμηση όλων των οντοτήτων (Game Objects) με τα προαναφερθέντα στοιχεία (Components). Ονομαστικά, τα Components που παρουσιάζονται είναι τοποθέτηση (Transform), σωματίδια (Particles), ήχος, γεωμετρία, κίνηση (Animation), δίκτυο, φώτα, υφές, κάμερα, προκατασκευές (Prefabs), έδαφος (Terrain) και Scripting. Ιδιαίτερη μνεία γίνεται στους σκιαστές της Unity, διάχυσης/ανάκλασης (Diffuse/Specular), διαφάνειας (Transparent), ανώμαλης επιφάνειας (Bump). Επίσης η Unity διαθέτει μια πολύ ισχυρή μηχανή φυσικής, την Ageia η οποία αναπτύσσεται από την εταιρία nVidia. Οι μηχανές φυσικής για να λειτουργήσουν χρειάζονται τις οντότητες που λέγονται συγκρουστές (Colliders). Στο εν λόγω κεφάλαιο εξηγούμε πως γίνεται η ανίχνευση συγκρούσεων με Colliders αλλά και η εξομοίωση φυσικής με την χρήση άκαμπτων σωμάτων (RigidBodies). Τέλος έχουμε τις γενικές ρυθμίσεις του παιχνιδιού Quality, Render, Time, Input και την ολοκλήρωση του παιχνιδιού. Με τον όρο “ολοκλήρωση του παιχνιδιού” εννοούμε την διαδικασία κατά την οποία το παιχνίδι θα μετατραπεί από ένα σύνολο αρχείων της μηχανής παιχνιδιών σε αυτόνομο πρόγραμμα.

Στο τέταρτο κεφάλαιο αναλύεται η αρχιτεκτονική των επιπέδων και η γραφική διεπαφή του χρήστη. Κάθε παιχνίδι χρειάζεται μια γραφική διεπαφή για να δίνει τις απαραίτητες πληροφορίες στον χρήστη και σε αυτό το κεφάλαιο αναφέρονται όλα τα μενού και η λειτουργία τους, καθώς και ο τρόπος με τον οποίο συνδέονται μεταξύ τους.

Στην αρχή του πέμπτου κεφαλαίου γίνεται αναφορά στην ρύθμιση της γεωμετρίας του αυτοκινήτου προκειμένου να γίνεται αντιληπτό από τη μηχανή φυσικής. Στην συνέχεια αναλύεται ο τρόπος με τον οποίο υπολογίζεται η ταχύτητα που θα αναπτύξει το αυτοκίνητο αναλογικά με την αντίδραση του χρήστη. Στη συνέχεια περιγράφεται ο τρόπος με τον οποίο οι αντίπαλοι αντιλαμβάνονται και αποφεύγουν τα εμπόδια, μια διαδικασία που ονομάζεται Steering behaviors-Obstacle avoidance. Δηλαδή, από τη θεωρία για τις μηχανές πεπερασμένων καταστάσεων και την ασαφή λογική, περνάμε στην εφαρμογή τους. Μέσω της χρήσης διαγραμμάτων φανερώνεται ο τρόπος “σκέψης” των αντιπάλων καθώς και ολόκληρη η δομή της τεχνητής νοημοσύνης του παιχνιδιού. Τέλος παραθέτοντας κώδικα παρουσιάζεται ο αλγόριθμος A* και η βελτίωση που επιτυγχάνεται χρησιμοποιώντας ουρά προτεραιότητας.

Στο έκτο κεφάλαιο παρουσιάζονται τα αποτελέσματα δοκιμών του παιχνιδιού οι οποίες βοήθησαν στην εξαγωγή συμπερασμάτων σχετικά με τη σταθερότητα, την ευχρηστία του και στο να γίνουν διορθώσεις με βάση τις παρατηρήσεις των χρηστών. Το ερωτηματολόγιο που χρησιμοποιήθηκε είναι το QUIS από το πανεπιστήμιο του Maryland.

Στο έβδομο και τελευταίο κεφάλαιο γίνεται ανακεφαλαίωση της διπλωματικής διατριβής, αναφέρονται τα συμπεράσματα της εργασίας και επισημαίνονται μελλοντικές επεκτάσεις.

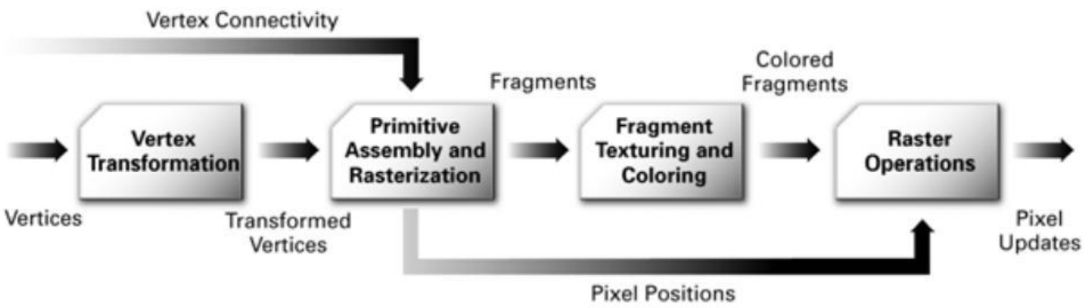
Κεφάλαιο 2

Τεχνολογία Γραφικών

Σε αυτό το κεφάλαιο παρουσιάζονται οι τεχνολογίες γραφικών που χρησιμοποιούνται στην ανάπτυξη παιχνιδιών αλλά και στην δημιουργία animations και εφέ με εφαρμογές στον κινηματογράφο.

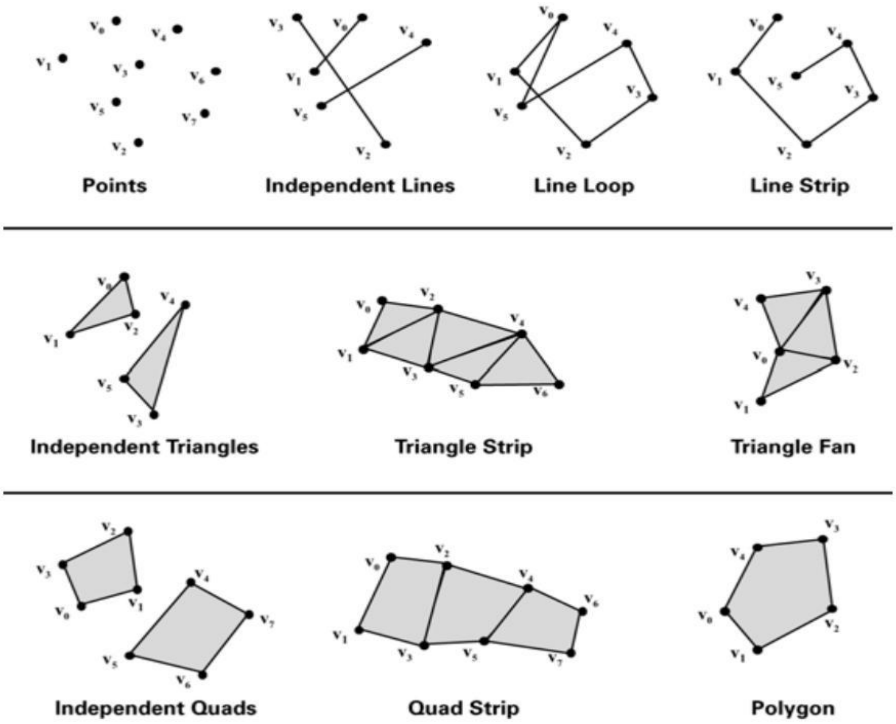
2.1 Διοχέτευση υλικού γραφικών (The Graphics Hardware Pipeline)

Η διοχέτευση (pipeline) είναι μία ακολουθία από στάδια τα οποία εκτελούνται παράλληλα με καθορισμένο τρόπο. Η Εικόνα 2.1 απεικονίζει σχηματικά τη διοχέτευση που χρησιμοποιείται από τις σύγχρονες κάρτες γραφικών σε επίπεδο υλικού.



Εικόνα 2.1: Η διοχέτευση υλικού γραφικών [24]

Η τρισδιάστατη εφαρμογή στέλνει στη GPU μία ακολουθία από κόμβους οι οποίοι αποτελούν γεωμετρικά σύνολα κυρίως πολύγονα, γραμμές και σημεία. Όπως φαίνεται και στην παρακάτω εικόνα υπάρχουν πολλοί τρόποι περιγραφής των γεωμετρικών συμβόλων.



Εικόνα 2.2: Τύποι γεωμετρικών συνόλων [24]

Κάθε κορυφή έχει μια θέση αλλά και διάφορες άλλες ιδιότητες όπως βασικό χρώμα, δευτερεύον ή κατοπτρικό (specular) χρώμα, μια ή περισσότερες ομάδες συντεταγμένων που αφορούν τις υφές (texture) και ένα "κανονικό" (normal) διάνυσμα. Το κανονικό διάνυσμα προσδιορίζει ποια είναι η κατεύθυνση της επιφάνειας των κορυφών και η τυπική χρήση του στους υπολογιστές σχετίζεται με το φωτισμό.

2.1.1 Μετασχηματισμός κορυφών

Ο μετασχηματισμός κορυφών αποτελεί το πρώτο στάδιο επεξεργασίας στην διοχέτευση υλικού γραφικών. Κατά το στάδιο αυτό εκτελείται μια ακολουθία μαθηματικών διαδικασιών σε κάθε κορυφή, οι οποίες περιλαμβάνουν το μετασχηματισμό της θέσης της κορυφής σε μια θέση της οθόνης, για χρήση από την μονάδα που κάνει απεικόνιση διανυσματικών γραφικών με χρήση εικονοστοιχείων (rasterizer), παράγοντας τις συντεταγμένες για εφαρμογή των υφών στην επιφάνεια και φωτίζοντας την κορυφή για να καθοριστεί το χρώμα της.

2.1.2 Πρωταρχική συναρμολόγηση και απεικόνιση

Οι μετασχηματισμένες κορυφές προχωρούν σειριακά στο επόμενο στάδιο, το οποίο ονομάζεται πρωταρχική συναρμολόγηση και απεικόνιση (primitive assembly and rasterization). Αρχικά συναθροίζονται οι κορυφές και σχηματίζουν γεωμετρικά σχήματα βασισμένα στις πληροφορίες που έχουν οι κορυφές με αποτέλεσμα τη δημιουργία μιας ακολουθίας από τρίγωνα, γραμμές, ή σημεία. Τα γεωμετρικά σχήματα ενδέχεται να απαιτήσουν περικοπή στην ορατή περιοχή του τρισδιάστατου χώρου (view frustum). Ο απεικονιστής (rasterizer) μπορεί επίσης να απορρίψει τα πολύγωνα ελέγχοντας εάν οι επιφάνειές του κοιτούν προς τα μέσα ή προς τα έξω, μια διαδικασία γνωστή ως ξεδιάλεγμα (culling).

Στα πολύγωνα τα οποία απομένουν μετά από τα βήματα της περικοπής και ξεδιαλέγματος πρέπει να γίνει απεικόνιση διανυσματικών γραφικών με χρήση εικονοστοιχείων (rasterization). Η απεικόνιση (rasterization) είναι η διαδικασία μέσω της οποίας καθορίζεται το σύνολο των εικονοστοιχείων που καλύπτονται από ένα γεωμετρικό σχήμα. Τα πολύγωνα, οι γραμμές και τα σημεία απεικονίζονται με χρήση εικονοστοιχείων, σύμφωνα με τους κανόνες που έχουν οριστεί για το κάθε ένα τύπο. Αποτελέσματα της απεικόνισης (rasterization) είναι ένα σύνολο θέσεων εικονοστοιχείου καθώς επίσης και ένα σύνολο που ορίζουν ένα κομμάτι (fragment). Δεν υπάρχει καμία σχέση μεταξύ του αριθμού των κορυφών που έχει ένα γεωμετρικό σχήμα και του αριθμού των κομματιών που παράγονται όταν γίνεται απεικόνιση(rasterization). Παραδείγματος χάριν, ένα τρίγωνο αποτελούμενο από τρείς κορυφές θα μπορούσε να καταλάβει ολόκληρη την οθόνη και επομένως να παράγει εκατομμύρια κομμάτια.

Η διάκριση μεταξύ ενός κομματιού και ενός εικονοστοιχείου είναι σημαντική. Ένα εικονοστοιχείο αντιπροσωπεύει τα περιεχόμενα ενός frame buffer σε μια συγκεκριμένη θέση, όπως το χρώμα, το βάθος και οποιεσδήποτε άλλες τιμές που συνδέονται με εκείνη την θέση. Ένα κομμάτι είναι η κατάσταση που ενδεχομένως χρειάζεται για να ενημερώσει ένα συγκεκριμένο εικονοστοιχείο.

Ο όρος τεμάχιο (fragment) χρησιμοποιείται επειδή η απεικόνιση (rasterization) χωρίζει κάθε γεωμετρικό σχήμα, για παράδειγμα ένα τρίγωνο, σε μικρά κομμάτια για κάθε εικονοστοιχείο που καλύπτει το σχήμα. Ένα τεμάχιο έχει για κάθε εικονοστοιχείο μια σχετική θέση, μια τιμή βάθους και ένα σύνολο παραμέτρων όπως χρώμα, κατοπτρικό (specular) χρώμα και ένα ή περισσότερα σύνολα συντεταγμένων υφής. Αυτές οι παράμετροι προέρχονται από τις μετασχηματισμένες κορυφές που απαρτίζουν το ιδιαίτερο γεωμετρικό σχήμα το οποίο χρησιμοποιείτε για την παραγωγή των κομματιών. Μπορούμε να σκεφτούμε κάθε κομμάτι ως ένα “πιθανό εικονοστοιχείο”. Εάν ένα κομμάτι περνά τις διάφορες δοκιμές απεικόνισης (rasterization) τότε ενημερώνει ένα εικονοστοιχείο στον frame buffer [24].

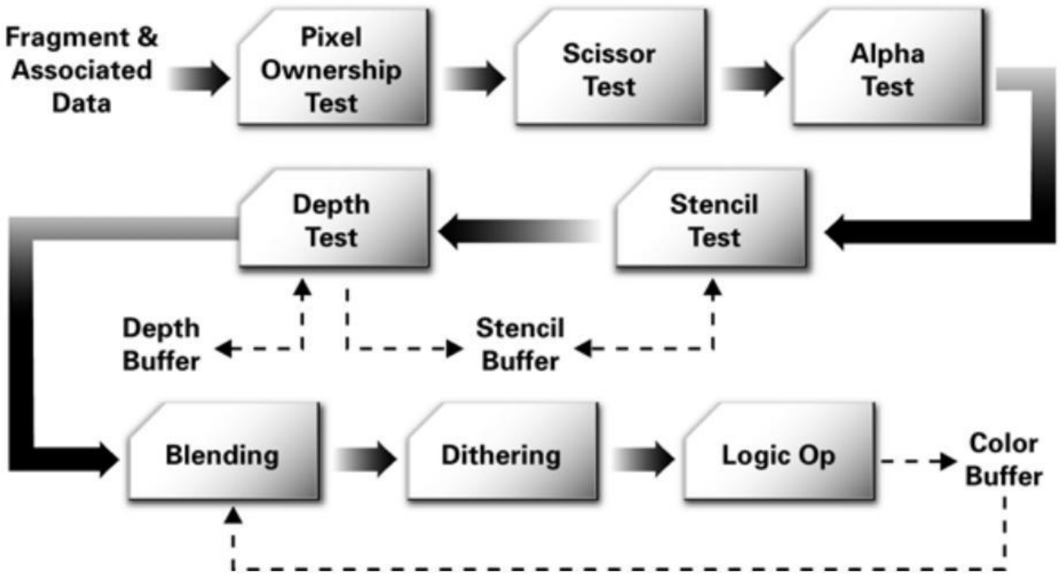
2.1.3 Παρεμβολή, υφές και χρωματισμός

Όταν ένα σχήμα έχει απεικονιστεί σε μια συλλογή μηδέν ή περισσότερων κομματιών, το στάδιο της παρεμβολής, των υφών και του χρωματισμού, μεταβάλλει τις παραμέτρους των κομματιών, όπου κρίνεται αναγκαίο, εκτελεί μια ακολουθία εφαρμογής υφών και μαθηματικών διαδικασιών, και καθορίζει το τελικό χρώμα του κάθε κομματιού. Εκτός από τον καθορισμό του τελικού χρώματος, το στάδιο αυτό μπορεί επίσης να δημιουργήσει ένα νέο κομμάτι ή ακόμη και να απορρίψει ένα κομμάτι με σκοπό την ανανέωση του αντίστοιχου εικονοστοιχείου στο frame buffer. Επιτρέποντας στο στάδιο αυτό να μπορεί να απορρίψει ένα κομμάτι, εκπέμπονται ένα ή κανένα χρωματισμένα κομμάτια για κάθε κομμάτι που λαμβάνει από την είσοδο.

2.1.4 Διαδικασίες απεικόνισης (rasterization)

Το στάδιο απεικόνισης διαδικασιών εκτελεί μια τελική ακολουθία διαδικασιών ανά κομμάτι αμέσως πριν ενημερωθεί ο frame buffer. Αυτές οι διαδικασίες αποτελούν ένα τυποποιημένο μέρος των βιβλιοθηκών OpenGL και Direct3D. Κατά τη διάρκεια αυτού του σταδίου, οι κρυμμένες επιφάνειες αποβάλλονται μέσω μιας διαδικασίας γνωστή ως δοκιμή βάθους(depth testing). Άλλα αποτελέσματα, όπως η ανάμειξη (blending) και η σκίαση βασισμένη στο stencil, εμφανίζονται επίσης κατά τη διάρκεια αυτού του σταδίου.

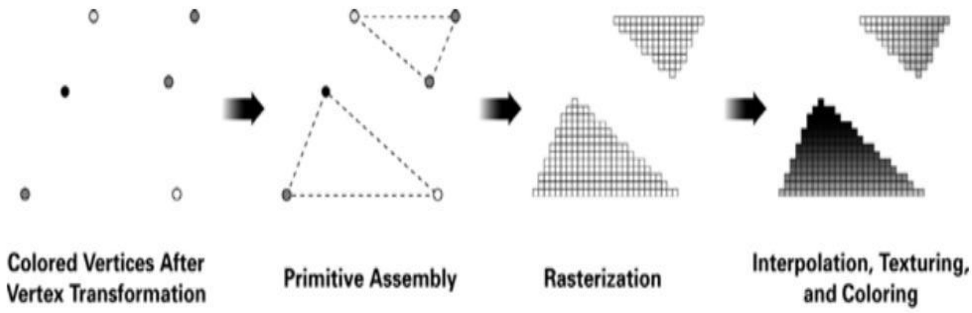
Το στάδιο raster διαδικασιών ελέγχει κάθε κομμάτι βασιζόμενο σε διάφορες δοκιμές, του ψαλιδιού (scissor test), άλφα (alpha test), του stencil, και των δοκιμών βάθους (depth test). Αυτές οι δοκιμές περιλαμβάνουν το τελικό χρώμα ή το βάθος του κομματιού, τη θέση εικονοστοιχείου και τις ανά εικονοστοιχείο τιμές του βάθους και του stencil. Εάν οποιαδήποτε δοκιμή αποτύχει, το στάδιο απορρίπτει το κομμάτι (fragment) χωρίς να ενημερώνει για την τιμή του χρώματος του εικονοστοιχείου. Σε περίπτωση επιτυχούς δοκιμής, η τιμή βάθους του εικονοστοιχείου μπορεί να αντικατασταθεί με την τιμή βάθους του κομματιού. Μετά το πέρας των δοκιμών, μια διαδικασία ανάμειξης (blending operation) συνδυάζει το τελικό χρώμα του κομματιού (fragment) με την αντίστοιχη τιμή χρώματος του εικονοστοιχείου. Τέλος, καταγράφεται σε έναν frame buffer ότι η λειτουργία αντικαθιστά το χρώμα του εικονοστοιχείου με το αναμεμειγμένο χρώμα [24].



Εικόνα 2.3: Τυπικές raster διαδικασίες OpenGL και Direct3D [24]

2.1.5 Οπτικοποιώντας την διοχέτευση γραφικών (Graphics Pipeline)

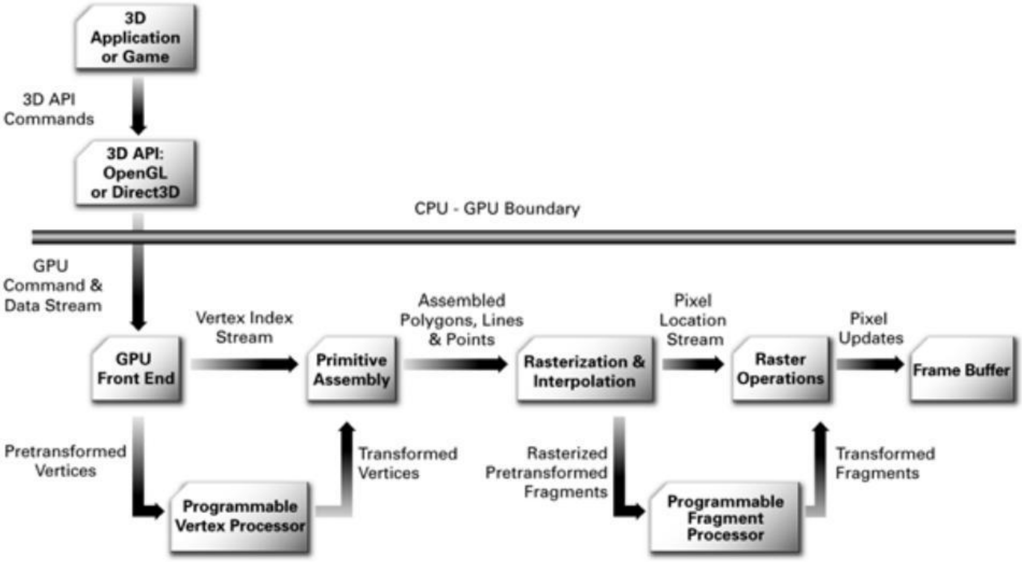
Παρακάτω φαίνονται τα στάδια της διοχέτευσης. Στην εικόνα, απεικονίζονται δύο τρίγωνα (rasterized). Η διαδικασία ξεκινά με το μετασχηματισμό και το χρωματισμό των κορυφών, συνεχίζεται τη συνάθροιση των σχημάτων και τη δημιουργία τριγώνων από τις κορυφές, όπως υποδεικνύουν οι διακεκομμένες γραμμές. Στο επόμενο βήμα ο απεικονιστής «γεμίζει» τα τρίγωνα με κομμάτια και τέλος λαμβάνονται υπόψη οι καταχωρημένες τιμές των κορυφών και χρησιμοποιούνται για την εφαρμογή υφών και χρώματος [24]. Να σημειωθεί ότι από έναν πολύ μικρό αριθμό κορυφών δημιουργούνται πολλά κομμάτια.



Εικόνα 2.4: Αρχικό στάδιο της διοχέτευσης γραφικών [24]

2.1.6 Προγραμματιζόμενη διοχέτευση γραφικών

Η σύγχρονη τάση που κυριαρχεί στο σχεδιασμό του υλικού των γραφικών είναι η ενσωμάτωση αυξημένης προγραμματιστικής δυνατότητας στο επίπεδο της μονάδας επεξεργασίας γραφικών (GPU). Στην Εικόνα 2.5 παρουσιάζονται τα στάδια επεξεργασίας των κορυφών και κομματιών στη διοχέτευση μιας προγραμματιζόμενης GPU.

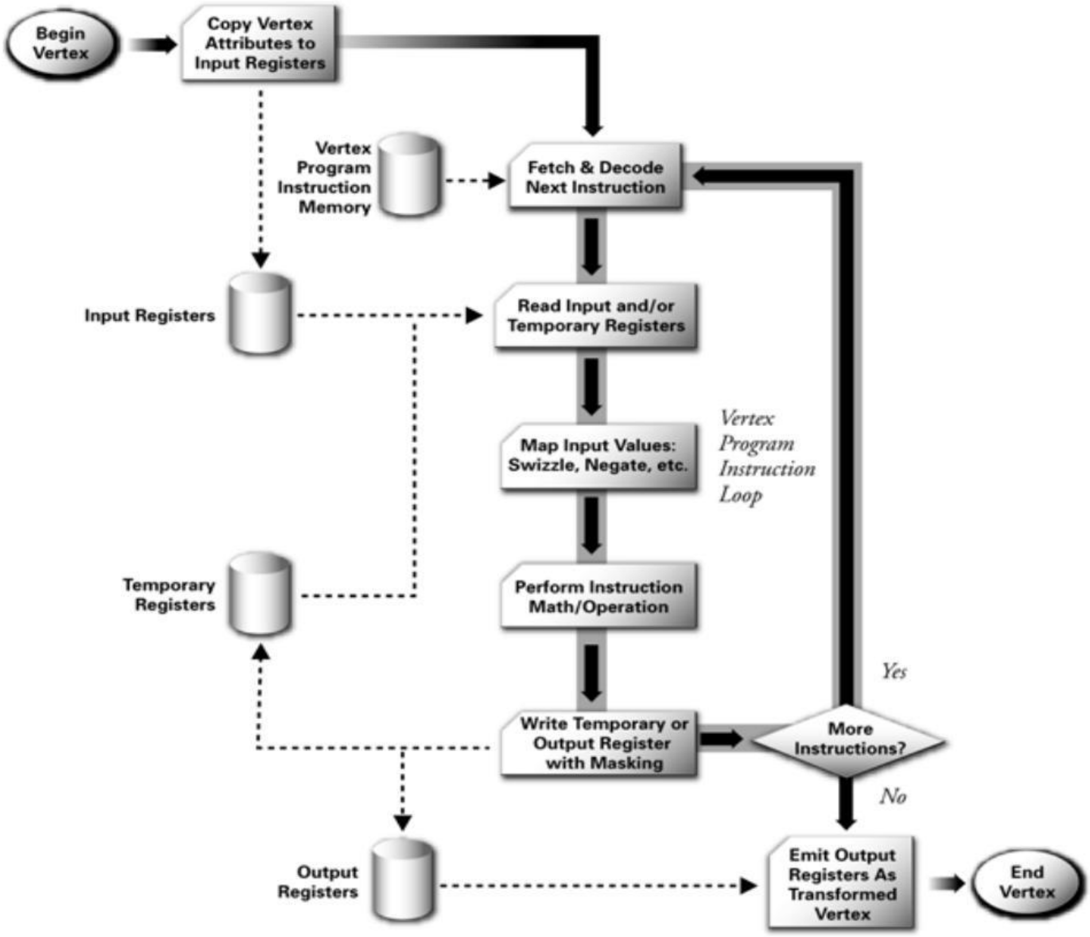


Εικόνα 2.5: Σχηματική απεικόνιση της διοχέτευσης γραφικών [24]

Είναι φανερό από την παραπάνω εικόνα ότι τόσο η επεξεργασία των κορυφών όσο και των κομματιών έχουν διασπαστεί σε προγραμματιζόμενες μονάδες. Ο προγραμματιζόμενος επεξεργαστής κορυφών είναι η μονάδα υλικού που εκτελεί τα προγράμματα Cg (C for graphics) κορυφών, ενώ ο προγραμματιζόμενος επεξεργαστής κομματιών είναι η μονάδα που εκτελεί τα Cg προγράμματα κομματιών.

2.1.7 Προγραμματιζόμενος επεξεργαστής κορυφών

Η Εικόνα 2.6 παρουσιάζει ένα διάγραμμα ροής για ένα τυπικό προγραμματιζόμενο επεξεργαστή κορυφών. Το μοντέλο ροής δεδομένων για την επεξεργασία κορυφών ξεκινά με την εισαγωγή των ιδιοτήτων των κορυφών (θέση, χρώμα, συντεταγμένες υφών κ.α.) στον επεξεργαστή κορυφών. Έπειτα, ο επεξεργαστής κορυφών επαναλαμβάνει την εκτέλεση μιας εντολής και την εκτελεί έως ότου το πρόγραμμα κορυφών να τερματιστεί. Οι εντολές έχουν πρόσβαση σε διάφορες διακριτές ομάδες καταχωρητών οι οποίοι περιλαμβάνουν τις τιμές των διανυσμάτων, όπως αυτών της θέσης, της κανονικής (normal) θέσης, ή του χρώματος. Οι ιδιότητες των καταχωρημένων κορυφών είναι μόνο για ανάγνωση και περιλαμβάνουν ορισμένες ομάδες ιδιοτήτων για τις κορυφές, ιδιαίτερα για την εφαρμογή. Οι προσωρινοί καταχωρητές μπορούν να διαβαστούν, να εγγραφούν και χρησιμοποιούνται για να πραγματοποιηθούν οι υπολογισμοί των ενδιάμεσων αποτελεσμάτων, κυρίως για την αποθήκευση των προσωρινών τιμών των μεταβλητών. Τα αποτελέσματα των καταχωρητών είναι μόνο για εγγραφή, για τα οποία είναι υπεύθυνο το λογισμικό. Όταν το πρόγραμμα των κορυφών τερματιστεί, το αποτέλεσμα των καταχωρητών εξόδου περιλαμβάνει τις καινούργιες μετασχηματισμένες κορυφές. Μετά την εγκατάσταση και την απεικόνιση του τριγώνου, οι παρεμβαλλόμενες τιμές για κάθε καταχωρητή περνάνε στον επεξεργαστή κομματιών [24].

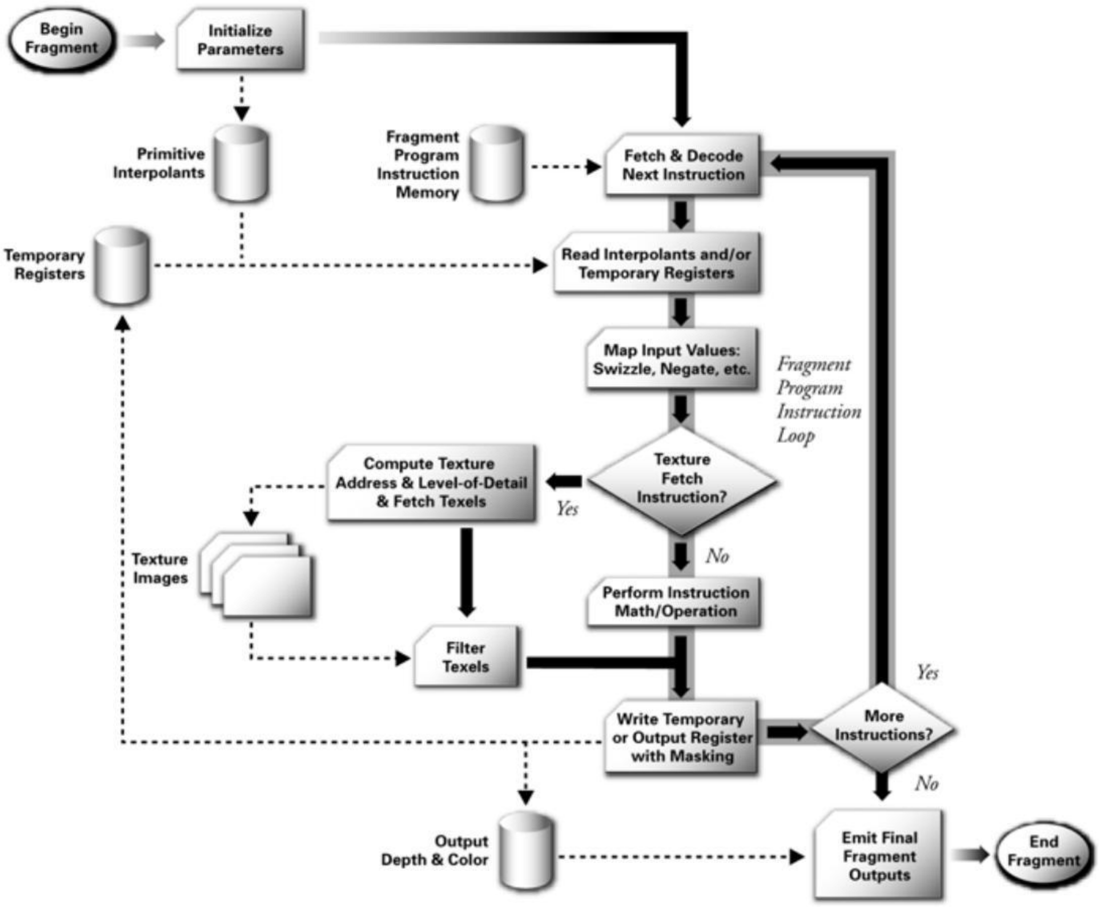


Εικόνα 2.6: Διάγραμμα ροής για τον προγραμματιζόμενο επεξεργαστή κορυφών [24]

2.1.8 Προγραμματιζόμενος επεξεργαστής κομματιών

Ο προγραμματιζόμενος επεξεργαστής κομματιών απαιτεί πολλές από τις ίδιες μαθηματικές διαδικασίες που χρειάζονται και οι επεξεργαστές κορυφών. Η διαφορά τους είναι ότι υποστηρίζει και επιπλέον διαδικασίες για την εφαρμογή υφών. Οι διαδικασίες για την εφαρμογή υφών δίνουν τη δυνατότητα στον επεξεργαστή να έχει πρόσβαση στην υφή της εικόνας χρησιμοποιώντας τις συντεταγμένες και έπειτα, να επιστρέφει ένα φιλτραρισμένο δείγμα της υφής της εικόνας.

Στην εικόνα που ακολουθεί φαίνεται ένα διάγραμμα ροής για ένα σύγχρονο προγραμματιζόμενο επεξεργαστή κομματιών. Όπως και στον επεξεργαστή κορυφών, έτσι και σε αυτήν την περίπτωση η ροή των δεδομένων περιλαμβάνει την εκτέλεση μιας σειράς εντολών έως ότου να τερματιστεί το πρόγραμμα. Επίσης, υπάρχει μια ομάδα από καταχωρητές εισόδου, οι οποίοι, σε αντίθεση με τις ιδιότητες των κορυφών, περιλαμβάνουν και τις ενδιάμεσες παραμέτρους για κάθε κομμάτι, οι οποίες έχουν οριστεί από πριν και προήλθαν από τις παραμέτρους των κορυφών που αποτελούν ένα σχήμα. Οι διαδικασίες της εγγραφής για τους καταχωρητές εξόδου περιλαμβάνουν το χρώμα και μερικές φορές την νέα τιμή βάθους του κάθε κομματιού. Οι εντολές των προγραμμάτων των κομματιών περιλαμβάνουν και ενώσεις με την υφή.



Εικόνα 2.7: Διάγραμμα ροής για τον προγραμματιζόμενο επεξεργαστή κομματιών [24]

2.2 Διεπαφές γραφικών (3D APIs)

2.2.1 Direct3D

Το Direct3D είναι κομμάτι του DirectX API της Microsoft και είναι διαθέσιμο μόνο για το λειτουργικό σύστημα Windows (Windows 95 ή νεότερο) και για το ανοιχτού κώδικα πρόγραμμα Wine. Αποτελεί επίσης τη βάση για τη γραφική διεπαφή των παιχνιδομηχανών Xbox και Xbox 360. Το Direct3D χρησιμοποιείται για να παράγει (render) τρισδιάστατα γραφικά σε εφαρμογές στις οποίες η χρονική απόδοση παίζει πολύ σημαντικό ρόλο, όπως για παράδειγμα στα παιχνίδια. Επίσης επιτρέπει στις εφαρμογές να εκτελούνται σε ολόκληρη την οθόνη αντί σε ένα μικρό παράθυρο, αν και μπορούν να ανοίξουν και μέσα σε παράθυρο εάν προγραμματιστούν για αυτόν τον σκοπό. Το Direct3D χρησιμοποιεί επιτάχυνση υλικού, “ hardware acceleration”, εάν φυσικά είναι διαθέσιμη από την κάρτα γραφικών, επιτρέποντας την επιτάχυνση σε ολόκληρη την διοχέτευση γραφικών ή σε ένα κομμάτι της. Αναδεικνύει εξελιγμένες δυνατότητες γραφικών όπως z-buffering, anti-aliasing, alpha blending, mipmapping, atmospheric effects, και perspective-correct texture mapping. Η ανάλυση των προαναφερθέντων τεχνολογιών είναι πέρα από τους στόχους της παρούσας εργασίας [14].

2.2.2 OpenGL

Το OpenGL (Open Graphics Library) είναι μια διεπαφή για τη δημιουργία εφαρμογών που παράγουν 2D και 3D γραφικά ανεξαρτήτως της πλατφόρμας στην οποία λειτουργούν. Η διεπαφή διαθέτει περισσότερες από διακόσιες πενήντα κλήσεις συναρτήσεων οι οποίες μπορούν να δημιουργήσουν πολύπλοκα περιβάλλοντα χρησιμοποιώντας βασικά γεωμετρικά σχήματα. Το OpenGL αναπτύχθηκε από την Silicon Graphics Inc. (SGI) το 1992 και χρησιμοποιείται σε εφαρμογές CAD, εικονικής πραγματικότητας, scientific visualization, information visualization και εξομοιωτές [16]. Επίσης χρησιμοποιείται και για παιχνίδια, όπου ανταγωνίζεται το Direct3D της Microsoft.

2.2.3 Direct3D εναντίον OpenGL

Το OpenGL βρίσκει εφαρμογή σε επαγγελματικές εφαρμογές ενώ το αντίπαλο δέος, το DirectX χρησιμοποιείται κυρίως για την ανάπτυξη παιχνιδιών. Ο όρος “επαγγελματικές εφαρμογές” αναφέρεται στην επαγγελματική παραγωγή και απεικόνιση γραφικών, όπως οπτικά εφέ σε ταινίες ή επιστημονική απεικόνιση και εξομοίωση, σε αντίθεση με τα παιχνίδια στα οποία τα παραγόμενα γραφικά είναι για προσωπική και όχι για επαγγελματική χρήση από τον τελικό χρήστη.

Γενικά, το Direct3D είναι σχεδιασμένο για να οπτικοποιεί τρισδιάστατες διεπαφές υλικού (3D hardware interfaces). Ο προγραμματιστής παιχνιδιών δεν χρειάζεται να ασχοληθεί με το υλικό του υπολογιστή στο οποίο αναπτύσσει την εφαρμογή του. Αντίθετα το OpenGL αποτελεί ένα τρισδιάστατο σύστημα απεικόνισης με επιτάχυνση υλικού (3D accelerated hardware rendering system) το οποίο μπορεί να προσομοιωθεί μέσα σε εφαρμογές. Η σχεδίαση των δύο αυτών διεπαφών είναι βασισμένη σε θεμελιωδώς ξεχωριστές ιδέες [9].

Το πρώτο απαιτεί από την εφαρμογή να διαχειριστεί το υλικό του υπολογιστή, ενώ το δεύτερο χρειάζεται μια επιπλέον υλοποίηση οδηγού (driver) η οποία το διαχειρίζεται ανεξάρτητα. Για το OpenGL αυτό σημαίνει μικρότερη δυσκολία στην ανάπτυξη κώδικα για το API, εντούτοις αυξάνεται η δυσκολία για την ανάπτυξη ενός οδηγού (driver) ο οποίος θα έχει καλή απόδοση. Όταν χρησιμοποιείται το Direct3D, ο μηχανικός πρέπει να διαχειριστεί τους πόρους του υλικού αυτόνομα. Σε αυτήν την περίπτωση όμως, η υλοποίηση είναι απλούστερη και οι μηχανικοί έχουν την ευελιξία να κατανείμουν τους πόρους του συστήματος με τον αποδοτικότερο για την εφαρμογή τρόπο. Πέρα από κάποιες μικρές λειτουργικές διαφορές, οι οποίες δημιουργήθηκαν με την πάροδο των χρόνων, οι δύο διεπαφές παρέχουν πλέον, σχεδόν το ίδιο επίπεδο λειτουργικότητας.

2.3 Δίκτυα γεωμετρίας (Meshes)

Κάθε παιχνίδι διαθέτει το αντίστοιχο περιβάλλον, το οποίο μπορεί να είναι ένας δρόμος, ένα σπίτι ή μία βιομηχανική εγκατάσταση. Όταν πρωτοεμφανίστηκαν τα ηλεκτρονικά παιχνίδια και επειδή οι κάρτες γραφικών δεν ήταν ιδιαίτερα ισχυρές, χρησιμοποιούσαν φωτογραφίες ή κινούμενες εικόνες για να αναπαραστήσουν το περιβάλλον του παιχνιδιού. Κάποια στιγμή έγινε μετάβαση από τις δύο, στις τρεις διαστάσεις και έτσι δημιουργήθηκε η ανάγκη για τρισδιάστατη γεωμετρία. Για την οπτικοποίηση ενός αυτοκινήτου ή ενός ανθρώπου στον υπολογιστή απαιτείται πολύπλοκη γεωμετρία που δεν αντιστοιχεί στον συνδυασμό απλών σχημάτων όπως κύβοι και σφαίρες. Για αυτό το λόγο μπορούμε να σχεδιάσουμε πολύπλοκη γεωμετρία μέσω χρήσης ειδικών προγραμμάτων, δηλαδή δίκτυα γεωμετρίας, τα οποία αναπαριστούν το αντικείμενο μας και δεν βασίζονται σε στοιχειώδη γεωμετρικά σχήματα.

2.4 Φωτισμός (Lighting)

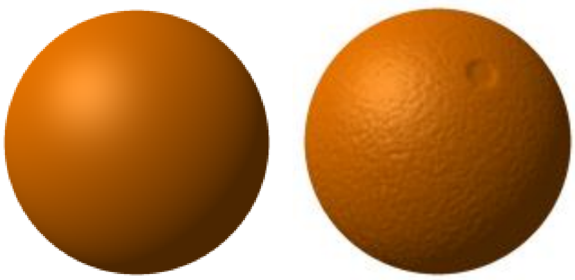
Ο φωτισμός είναι ο σημαντικότερος παράγοντας στην τεχνολογία των γραφικών καθώς από αυτό καθορίζεται το πόσο ρεαλιστικά θα απεικονίζεται η γεωμετρία ενός αντικειμένου. Στην σημερινή εποχή το επίπεδο ρεαλισμού έχει φτάσει στο σημείο να μην μπορούμε να διακρίνουμε τη διαφορά μεταξύ των γραφικών και της πραγματικότητας. Κάποιοι από τους αλγόριθμους που χρησιμοποιούνται κυρίως για την παραγωγή οπτικών εφέ σε ταινίες είναι οι: ray tracing, global illumination, final gather, το μεγαλύτερο μειονέκτημα των οποίων είναι ο χρόνος υπολογισμού (rendering). Οι κάρτες γραφικών δεν μπορούν να εκτελέσουν αυτούς τους αλγόριθμους σε πραγματικό χρόνο λόγω των απαιτήσεων τους σε υπολογιστική ισχύ, με αποτέλεσμα στα παιχνίδια να εφαρμόζεται η τεχνική του χάρτη φωτισμού

(Lightmap). Ο χάρτης φωτισμού είναι μία δομή δεδομένων που περιέχει πληροφορίες για σκιές και φωτεινότητα επιφανειών και είναι στατικός, δηλαδή οι σκιές δε μεταβάλλονται κατά τη διάρκεια του παιχνιδιού.

2.5 Υφές (Textures)

Υφή ονομάζουμε τις ιδιότητες και το αίσθημα που μας αφήνει μία επιφάνεια όταν την αγγίζουμε, και τις χρησιμοποιούμε για να αποδώσουμε ρεαλισμό στην τρισδιάστατη απεικόνιση. Οι υφές όμως, είναι εικόνες που πρέπει να χαρτογραφηθούν πάνω στα τρισδιάστατα αντικείμενα. Η χαρτογράφηση (texture mapping) είναι μια μέθοδος για να αποδώσουμε λεπτομέρεια ή να χρωματίσουμε ένα γραφικό φτιαγμένο από υπολογιστή ή ένα τρισδιάστατο μοντέλο. Υπάρχουν πολλά είδη χαρτογράφησης υφών. Παρακάτω αναφέρονται κάποια από τα βασικότερα και ευρέως χρησιμοποιούμενα.

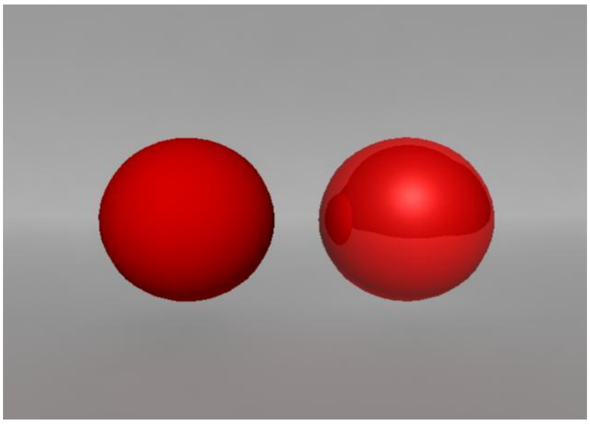
Πολυχαρτογράφηση (multitexturing) ονομάζεται η χρησιμοποίηση περισσότερων από μιας υφών πάνω σε ένα τρισδιάστατο μοντέλο. Για παράδειγμα, μια ρόδα έχει άλλο texture για την ζάντα και άλλο για το λάστιχο, η γεωμετρία της όμως παραμένει ενιαία. Η χαρτογράφηση βάθους (Bump mapping) επιτρέπει σε μία υφή να ελέγχει το ύψος της επίστρωσης με αποτέλεσμα η επιφάνεια να μας δίνει την εντύπωση του βάθους, χωρίς να επηρεάζεται η γεωμετρία. Αυτό που συμβαίνει στην ουσία είναι ότι το bump mapping παρεμβαίνει στην διαδικασία φωτισμού και έτσι οι αλγόριθμοι νομίζουν ότι η επιφάνεια έχει βάθος [6]. Οι δημοφιλέστερες μέθοδοι είναι το normal και parallax mapping.



Εικόνα 2.8: Αριστερά μία σφαίρα χωρίς bump, δεξιά με bump [6]

2.6 Σκιαστές (Shaders)

Ένας σκιαστής στον τομέα των γραφικών είναι ένα σετ από εντολές λογισμικού, οι οποίες χρησιμοποιούνται από τους γραφικούς πόρους κυρίως για να εκτελέσουν σχεδιαστικά εφέ. Τα εφέ αυτά αφορούν κυρίως την αντίδραση των διαφόρων υλικών με το φως. Για παράδειγμα ο σκιαστής Lambert είναι για τις ματ επιφάνειες ενώ ο Blinn για μεταλλικές [19].



Εικόνα 2.9: Αριστερά σφαίρα με lambert, δεξιά με blinn

2.7 Κινούμενο σχέδιο (Animation)

Στα ελληνικά αποδίδεται είτε με τον όρο Κινούμενη Εικόνα, είτε περιφραστικά, ως Απόδοση Κίνησης στην Εικόνα. Animation ονομάζεται η ταχεία προβολή μιας σειράς εικόνων (δισδιάστατης ή τρισδιάστατης μακέτας) ή θέσεων ενός μοντέλου, με σκοπό και τελικό αποτέλεσμα την ψευδαίσθηση της κίνησης. Αποτελεί οφθαλμαπάτη της κίνησης και αυτό συμβαίνει εξ αιτίας του φαινομένου διατήρησης της εικόνας στο μάτι επί 1/12 του δευτερολέπτου (μεταίσθημα ή μετείκασμα). Η κίνηση μπορεί να δημιουργηθεί και να παρουσιαστεί με πολλούς τρόπους, με την πιο διαδεδομένη μέθοδο απεικόνισης της κινούμενης εικόνας να αποτελείται από ένα πρόγραμμα βίντεο ή κινουμένου σχεδίου.

Στις περισσότερες ταινίες κινουμένων σχεδίων του 20ού αιώνα χρησιμοποιούνταν η παραδοσιακή τεχνική απόδοσης της κίνησης στο σχέδιο, με βάση την οποία κάθε ξεχωριστό πλαίσιο μιας ταινίας που έχει δημιουργηθεί με την παραδοσιακή τεχνική, αποτελείται από μια φωτογραφία ή ένα σχέδιο, που σχεδιάζεται πρώτα σε ένα χαρτί. Κάθε σχέδιο διαφέρει ελάχιστα από το προηγούμενο, προκειμένου να δημιουργηθεί η αίσθηση της κίνησης,. Τα σχέδια των animator αποτυπώνονται σε μια διαφάνεια εκτύπωσης που ονομάζεται κυψέλη, η οποία γεμίζεται με χρώματα προσδιορισμένων αποχρώσεων και τόνων από την πίσω πλευρά του σχεδίου. Οι ολοκληρωμένες διαφάνειες των χαρακτήρων φωτογραφίζονται μία-προς-μία σε φιλμ κινούμενης εικόνας με μια ραμφοειδή κάμερα, μπροστά από ένα ζωγραφισμένο φόντο.

Η παραδοσιακή μέθοδος της διαφάνειας ήταν πια ξεπερασμένη στις αρχές του 21ου αιώνα. Σήμερα, τα σχέδια των δημιουργών και τα φόντα σαρώνονται ή ζωγραφίζονται απευθείας σε ένα υπολογιστικό σύστημα. Χρησιμοποιούνται διάφορα λογισμικά για τον χρωματισμό των σχεδίων και για την προσομοίωση των κινήσεων και των εφέ στην κάμερα. Το τελικό κομμάτι της διαδικασίας είναι η εξαγωγή σε κάποιο από τα μεταφορικά μέσα, από το παραδοσιακό φιλμ των 35 mm, αλλά και σε πιο σύγχρονα μέσα όπως το ψηφιακό βίντεο. Η εμφάνιση των παραδοσιακών διαφανειών διατηρείται ακόμα, και η ουσιαστική δουλειά των animator παραμείνει απαραίτητη τα τελευταία 70 χρόνια. Κάποιοι παραγωγοί, χρησιμοποιούν τον όρο "tradigital" (παραδοσιακό-ψηφιακό) για να περιγράψουν την τεχνική των διαφανειών που κάνουν εκτεταμένη χρήση της τεχνολογίας των Η/Υ.

Τα ηλεκτρονικά παιχνίδια χρησιμοποιούν animation σε κάποιες περιπτώσεις όπως στην αναπαράσταση ανθρώπων. Σε ένα εξειδικευμένο πρόγραμμα δημιουργούμε τη γεωμετρία και στην συνέχεια το περπάτημα, με κύριο σκοπό και ενδιαφέρον να δημιουργηθεί ένας πλήρης κύκλος περπατήματος. Όταν ολοκληρωθεί η διαδικασία, η γεωμετρία και το animation μεταφέρονται στην μηχανή παιχνιδιών. Να σημειωθεί σε αυτό το σημείο πως εάν δεν υπάρχει μηχανή παιχνιδιών, η εισαγωγή animation είναι μια πολύ χρονοβόρα και κουραστική διαδικασία. Στη συνέχεια η γεωμετρία μας προγραμματίζεται ώστε να "αναπαράγει" το animation όταν ο χρήστης πατάει κάποιο πλήκτρο. Για παράδειγμα κάθε φορά που ο χρήστης δίνει την εντολή στον χαρακτήρα που ελέγχει να περπατήσει, τότε ξεκινά η ακολουθία του animation και επαναλαμβάνεται για όσο κρατιέται πατημένο το κουμπί..Επειδή υπάρχει η περίπτωση ο χρήστης να πατήσει πολλά κουμπιά γρήγορα και τα διάφορα animations να μπλεχτούν παράγοντας ανεπιθύμητα αποτελέσματα πάντα δημιουργείται και ένα script για τον έλεγχο τους [3].

2.8 Περιβάλλοντα ανάπτυξης γραφικών (Μηχανές)

Καθώς αυξάνεται η ζήτηση για εντυπωσιακότερα παιχνίδια, αυξάνεται η πολυπλοκότητά τους, λόγω του ότι η γεωμετρία γίνεται ακριβέστερη, τα εφέ περισσότερα και ο κώδικας εκτενέστερος. Μεγάλος αριθμός ανθρώπων θα πρέπει να συνεργαστούν με τον αποτελεσματικότερο τρόπο για να παραχθεί το παιχνίδι σε λογικά χρονικά πλαίσια, ως εκ τούτου η ανάγκη για ένα ενιαίο σύστημα παραγωγής παιχνιδιών είναι μεγάλη. Τα συστήματα αυτά ονομάζονται Μηχανές και παρέχουν ένα σύνολο εργαλείων και βιβλιοθηκών κώδικα για την ευκολότερη ανάπτυξη παιχνιδιών και οποιονδήποτε άλλων εφαρμογών απεικόνισης. Οι Μηχανές χωρίζονται σε τρείς κατηγορίες, γραφικών, φυσικής και παιχνιδιών ανάλογα με τον τομέα εξειδίκευσης τους.

2.8.1 Μηχανές γραφικών

Οι μηχανές γραφικών ή αλλιώς μηχανές 3D αποτελούν βιβλιοθήκες που παρέχουν συναρτήσεις για εύκολη απεικόνιση γραφικών. Η χρήση τους επεκτείνεται πέρα από την ανάπτυξη παιχνιδιών, στην δημιουργία εξομοιωτών για εκπαιδευτικούς σκοπούς, αρχιτεκτονικές απεικονίσεις και γενικότερα οτιδήποτε χρειάζεται γραφικά σε πραγματικό χρόνο. Το μεγάλο τους μειονέκτημα είναι ότι δεν παρέχουν επιπλέον εργαλεία για τη διαχείριση ήχου, δικτύου και άλλων συστημάτων απαραίτητων για ένα παιχνίδι. Η απουσία αυτών των εργαλείων δεν σημαίνει πως εάν χρησιμοποιήσουμε μηχανή γραφικών δε μπορεί να δημιουργηθεί ένα δικτυακό παιχνίδι. Ο κώδικα όμως για την σύνδεση των υπολογιστών και την ανταλλαγή των μεταξύ τους πληροφοριών, θα πρέπει να γραφτεί εξολοκλήρου από τους δημιουργούς του παιχνιδιού. Μερικές μηχανές γραφικών είναι οι OGRE3D, IRRLicht, Horde3D.

2.8.2 Μηχανές φυσικής

Προκειμένου να υπάρχει ρεαλισμός στα παιχνίδια είναι απαραίτητη η εισαγωγή σε αυτά των νόμων της φυσικής. Για την εξομοίωση χρησιμοποιούνται αρκετά συστατικά όπως η Νευτώνεια φυσική, η βαρύτητα και η ανίχνευση συγκρούσεων, ο συνδυασμός των αποδίδει πολύ εντυπωσιακά αποτελέσματα. Θα πρέπει όμως να λαμβάνεται υπόψη και η πλευρά της διασκέδασης. Δηλαδή, ένα παιχνίδι με απόλυτα ρεαλιστικό μοντέλο φυσικής ίσως είναι αρκετά δύσκολο στο χειρισμό του, γεγονός που θα δυσαρεστήσει τον χρήστη. Σε κάθε περίπτωση, γίνεται προσπάθεια να βρεθεί η χρήση τομή μεταξύ ρεαλισμού και ευκολίας στον χειρισμό.

Μια μηχανή φυσικής είναι ένα πρόγραμμα το οποίο παρέχει συναρτήσεις για εξομοίωση συγκεκριμένων φυσικών συστημάτων. Μερικά από αυτά είναι η φυσική άκαμπτου και μαλακού σώματος (rigid and soft body dynamics), ανίχνευση συγκρούσεων (collision detection) και δυναμική υγρών (fluid dynamics). Μηχανές φυσικής χρησιμοποιούνται σε πολλούς τομείς όπως γραφικά υπολογιστών, ηλεκτρονικά παιχνίδια και ταινίες.

Υπάρχουν δύο κατηγορίες μηχανών φυσικής, οι πραγματικού χρόνου και οι υψηλής ακρίβειας. Οι υψηλής ακρίβειας μηχανές χρειάζονται μεγάλη υπολογιστική ισχύ και χρησιμοποιούνται από επιστήμονες και σχεδιαστές ταινιών animation. Οι μηχανές πραγματικού χρόνου κάνουν πιο απλούς υπολογισμούς έτσι ώστε να μπορέσει ένας προσωπικός υπολογιστής να εκτελέσει μια εφαρμογή με ικανοποιητικό ρυθμό(game play frame rate).

Όσον αφορά τα παιχνίδια, διαθέτουν δύο κύρια μέρη. Το πρώτο είναι η ανίχνευση συγκρούσεων και το δεύτερο η δυναμική εξομοίωση (dynamics simulation), το οποίο είναι υπεύθυνο για τον υπολογισμό των δυνάμεων που ασκούνται στα διάφορα σώματα. Οι σύγχρονες μηχανές φυσικής διαθέτουν και σύστημα εξομοίωσης υγρών.

2.8.3 Μηχανές παιχνιδιών

Μια μηχανή παιχνιδιών είναι ένα λογισμικό σχεδιασμένο για τη δημιουργία και ανάπτυξη ηλεκτρονικών παιχνιδιών. Η κύρια λειτουργικότητα περιλαμβάνει μια μηχανή φωτισμού (rendering engine) για γραφικά δύο και τριών διαστάσεων, μία μηχανή φυσικής, συστήματα ήχου, scripting, animation και δικτύου. Επίσης παρέχεται ένα εύχρηστο γραφικό περιβάλλον με πληθώρα εργαλείων, για την απλή και εύκολη ανάπτυξη του παιχνιδιού [13]. Όλα τα παιχνίδια πλέον αναπτύσσονται με την βοήθεια μηχανών, μερικές από τις πιο γνωστές αναφέρονται παρακάτω:

Source, Ego

Δύο από τις μεγαλύτερες μηχανές παιχνιδιών, με την πρώτη αναπτύσσεται από την Valve και τη δεύτερη από την Codemasters. Πολλά εμπορικά επιτυχημένα παιχνίδια έχουν δημιουργηθεί με τη χρήση αυτών των μηχανών. Η τιμή απόκτηση τους όμως είναι απαγορευτική και ουσιαστικά είναι διαθέσιμες μόνο για μεγάλα studio παραγωγής.

Unreal engine

Ίσως η καλύτερη μηχανή παιχνιδιών αυτή τη στιγμή. Έχοντας ξεκινήσει από το 1998 με το παιχνίδι πρώτου προσώπου Unreal, η μηχανή έγινε η βάση δημιουργίας για πολλά επιτυχημένα παιχνίδια: *Unreal Tournament*, *Deus Ex*, *Turok*, *Tom Clancy's Rainbow Six 3: Raven Shield*, *Tom Clancy's Rainbow Six: Vegas*, *Gears of War*, *BioShock*, *BioShock 2*,*Mirror's Edge*, *Batman: Arkham Asylum*. Παρόλο που αρχικά προοριζόταν μόνο για παιχνίδια πρώτου προσώπου, έχουν παραχθεί και διαφορετικού τύπου παιχνίδια όπως stealth (*Tom Clancy's Splinter Cell*), MMORPG (*Vanguard: Saga of Heroes*), RPGs (*Mass Effect*, *The Last Remnant*, και *Harry Potter and the Philosopher's Stone)* [23].

Με τον πυρήνα της να είναι γραμμένος σε C++, η Unreal έχει πολύ μεγάλο βαθμό φορητότητας. Τα παιχνίδια που παράγονται μπορούν να τρέξουν σε πολλές πλατφόρμες προσωπικών υπολογιστών, όπως Microsoft Windows, Linux, Mac OS και Mac OS X, αλλά και σε κονσόλες όπως Dreamcast, Xbox, Xbox 360, PlayStation 2, και PlayStation 3.

Διαθέτει δική της Scripting Engine και η γλώσσα προγραμματισμού είναι η Unreal Script. Επιπλέον παρέχεται πληθώρα εργαλείων για σχεδιαστές και μηχανικούς. Η τελευταία έκδοση, είναι η Unreal Engine 3, η οποία υποστηρίζει την τεχνολογία Microsoft's DirectX 9 για 32/64-bit Windows XP/Windows Vista/Windows 7 και Xbox 360 πλατφόρμες, DirectX 10 και DirectX 11 για 32/64-bit Windows Vista/Windows 7, και OpenGL για 32/64-bit Linux, Mac OS X και PlayStation 3.

Η Unreal Engine τον Νοέμβριο του 2009 ξεκίνησε να διατίθεται δωρεάν για μη εμπορική χρήση. Οι μεγάλες απαιτήσεις της μηχανής σε πόρους συστήματος αλλά και η πολυπλοκότητα των εργαλείων της, την έκαναν ακατάλληλη για την συγκεκριμένη εργασία.

Fife engine

Η FIFE είναι μια μηχανή εξειδικευμένη στα δυσδιάστατα παιχνίδια με ισομετρική προοπτική, όπως αυτή που χρησιμοποιείται κατά κόρον στα παιχνίδια στρατηγικής. Ο κώδικας της μηχανής βρίσκεται ακόμα σε πρώιμο beta στάδιο και είναι διαθέσιμη με την GPL άδεια που σημαίνει ότι ο οποιοσδήποτε μπορεί να μεταφορτώσει τον κώδικα και να συνεισφέρει στην βελτίωση της. Τη στιγμή που το API βρίσκεται σε συνεχή τροποποίηση είναι δύσκολο να καθορίσουμε πόσο εύκολο είναι να χρησιμοποιηθεί το προϊόν από τους σχεδιαστές παιχνιδιών. Η FIFE έχει σχεδιαστεί για Windows και Linux πλατφόρμες.

Panda3D

Η μηχανή γραφικών Panda3D από τα αρχικά του Platform Agnostic Networked Display Architecture, είναι μια ισχυρή μηχανή για σχεδίαση σε Linux, Sun, Windows. Αρχικά αναπτύχθηκε από τα studios εικονικής πραγματικότητας της Walt Disney και χρησιμοποιήθηκε για να υλοποιήσουν το MMOG παιχνίδι Toon town Online. Μετά από αυτό η μηχανή έγινε ανοικτού κώδικα. Ο πυρήνας της είναι γραμμένος σε C++ και διαθέτει διεπαφή για σύνδεση με την γλώσσα Python. Μερικά από τα εργαλεία που παρέχει η Panda είναι τα εξής:

- Ανάλυση απόδοσης
- Εξερεύνηση του γράφου σκηνής
- Debugging
- Εισαγωγή και εξαγωγή αρχείων π.χ. γεωμετρία
- Τρισδιάστατο ήχο, χρησιμοποιώντας είτε το FMOD, OpenAl ή Miles sound system
- Collision detection
- Μηχανή φυσικής Open Dynamics Engine

Παρόλα τα θετικά στοιχεία που συγκεντρώνει η Panda3D, υπάρχουν και κάποια αρνητικά. Ο ρυθμός ανανέωσης του παιχνιδιού, μπορεί να μειωθεί δραματικά με την προσθήκη πολύπλοκης γεωμετρίας. Επίσης τα σωματιδιακά εφέ (Particles) αλλά και ο φωτισμός βρίσκονται σε πρώιμο στάδιο και δεν εκμεταλλεύονται πλήρως τις δυνατότητες των σημερινών καρτών γραφικών. Τέλος η εξομοίωση της συμπεριφοράς ενός αυτοκινήτου καθίσταται πολύ δύσκολη καθώς δεν παρέχονται εξειδικευμένοι συγκρουστές (colliders) για τροχούς επίγειων οχημάτων.

Unity3D

Η Unity3D αποτελεί ένα ολοκληρωμένο προγραμματιστικό περιβάλλον ανάπτυξης παιχνιδιών. Παρέχει πληθώρα εργαλείων για την επεξεργασία της γεωμετρίας, των υφών, του φωτισμού και των ήχων. Επιπλέον ενσωματώνει την μηχανή φυσικής Ageia που αναπτύσσεται από την Nvidia. Διαθέτει συγκρουστές (colliders) εξειδικευμένους για τροχούς οχημάτων κάτι που την καθιστά κατάλληλη για εξομοίωση αυτοκινήτων. Σε επίπεδο βιβλιοθηκών γραφικών, η Unity3D υποστηρίζει

DirectX και OpenGL ενώ τα παιχνίδια που παράγονται μπορούν να "τρέξουν" στην πλατφόρμα Windows και Macintosh. Η αρχιτεκτονική της εν λόγω μηχανής θα αναλυθεί περαιτέρω στο Κεφάλαιο 3 καθώς χρησιμοποιήθηκε για τις ανάγκες τις παρούσας διπλωματικής.

2.9 Τεχνητή Νοημοσύνη (Artificial intelligence)

Η τεχνητή νοημοσύνη είναι παρούσα σχεδόν σε όλα τα ηλεκτρονικά παιχνίδια.. Κάθε οντότητα που δεν ελέγχεται από τον χρήστη θα πρέπει να υπάγεται στον έλεγχο της τεχνητής νοημοσύνης. Για παράδειγμα σε ένα παιχνίδι αγώνων αυτοκινήτων τα αντίπαλα αυτοκίνητα ελέγχονται από τον υπολογιστή και άρα είναι σε θέση να καταλάβουν πότε πρέπει να πατήσουν φρένο ή γκάζι και πότε να στρίψουν. Η ρύθμιση της δυσκολίας των αντιπάλων δεν αποτελεί ένα τετριμμένο ζήτημα. Εάν οι αντίπαλοι είναι πολύ δύσκολοι θα υπάρξει δυσαρέσκεια από το χρήστη ενώ αντίθετα εάν είναι εύκολοι δεν θα υπάρχει ενδιαφέρον. Κατά τη διάρκεια των ετών αναπτύχτηκαν διάφορες τεχνικές για την ανταγωνιστικότερη συμπεριφορά των αντιπάλων. Παρακάτω γίνεται μια σύντομη αναφορά στις μηχανές πεπερασμένων καταστάσεων και στη συνέχεια αναπτύσσεται η τεχνική της ασαφούς λογικής. Τέλος γίνεται αναφορά στον αλγόριθμο A*, έναν από τους πιο διαδεδομένους αλγόριθμους για την εύρεση του συντομότερου μονοπατιού.

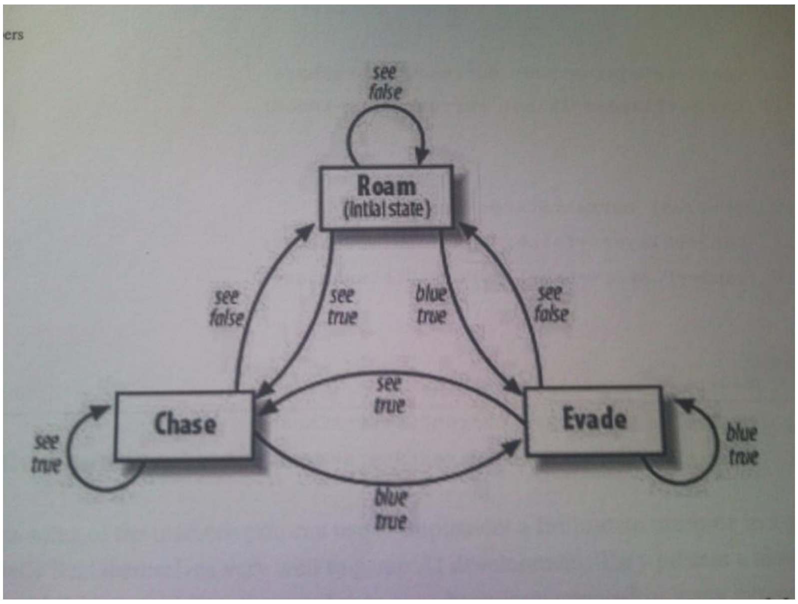
2.9.1 Λήψη αποφάσεων (Decision making)

Οι οντότητες ενός παιχνιδιού που ελέγχονται από τον υπολογιστή θα πρέπει να διαθέτουν κάποιου είδους τεχνητή νοημοσύνη. Με αυτόν τον τρόπο το παιχνίδι γίνεται πιο ενδιαφέρον και οι αντίπαλοι δυσκολότεροι. Αυτό που κάνει τους αντιπάλους αυτόνομους είναι η δυνατότητα να παίρνουν αποφάσεις ανάλογα με τις εκάστοτε συνθήκες του παιχνιδιού. Όσο καλύτερες είναι οι αποφάσεις που παίρνουν, τόσο πιο "έξυπνοι" φαίνονται στον χρήστη.

Μηχανές πεπερασμένων καταστάσεων (F.S.M.)

Μία FSM είναι μια γενική μηχανή που μπορεί να βρίσκεται κάθε φορά σε μια από πολλές προκαθορισμένες καταστάσεις, οι οποίες καθορίζουν ανάλογα και τη συμπεριφορά της Υπάρχουν σύνολα από συνθήκες που καθορίζουν σε ποια κατάσταση θα μεταβεί η FSM. Οι FSM χρησιμοποιούνται σε παιχνίδια από πολύ παλιά. Τα φαντάσματα στο παιχνίδι Pac Man είναι FSMs, οι οποίες διαθέτουν τρεις καταστάσεις: ελεύθερος περίπατος, επίθεση στον χρήστη, αποφυγή του χρήστη. Σε κάθε κατάσταση συμπεριφέρονται διαφορετικά και η μετάβαση σε κάποια κατάσταση καθορίζεται από τη συμπεριφορά του χρήστη. Για παράδειγμα όταν ο χρήστης πάρει το «ειδικό χάπι» η κατάσταση των φαντασμάτων μεταβάλλεται και από επίθεση γίνεται αποφυγή .

Αν και οι FSMs υπάρχουν στον κόσμο των υπολογιστών για πάρα πολλά χρόνια, χρησιμοποιούνται ακόμα ευρύτατα και στα σημερινά παιχνίδια. Η ευκολία στην υλοποίηση αλλά και την αποσφαλμάτωση είναι δύο από τους λόγους που τις κάνουν ελκυστικές λύσεις σε προβλήματα τεχνητής νοημοσύνης.



Εικόνα 2.10: Παράδειγμα διαγράμματος FSM [4]

Παρόλο που οι FSM είναι χρήσιμες σε πολλές περιπτώσεις, έχουν και αυτές ελαττώματα, ένα από τα οποία είναι το γεγονός ότι οι μεταβάσεις είναι προκαθορισμένες. Με λίγα λόγια, εάν ο χρήστης παίζει πολύ ένα παιχνίδι, θα αρχίζει να καταλαβαίνει πως αντιδρούν αντίπαλοι σε διάφορες συνθήκες και θα είναι σε θέση να τους κερδίζει συνέχεια. Τα αποτέλεσμα είναι ο χρήστης μετά από κάποιο χρονικό διάστημα να χάσει το ενδιαφέρον του για το παιχνίδι.

Πρέπει λοιπόν οι αντίπαλοι να είναι ικανοί αντιδρούν αυτόνομα χωρίς να είναι γνωστή ακριβώς η απόφαση τους σε μία κατάσταση. Την αδυναμία της FSM σε αυτό το σημείο καλύπτει η τεχνική της ασαφούς λογικής.

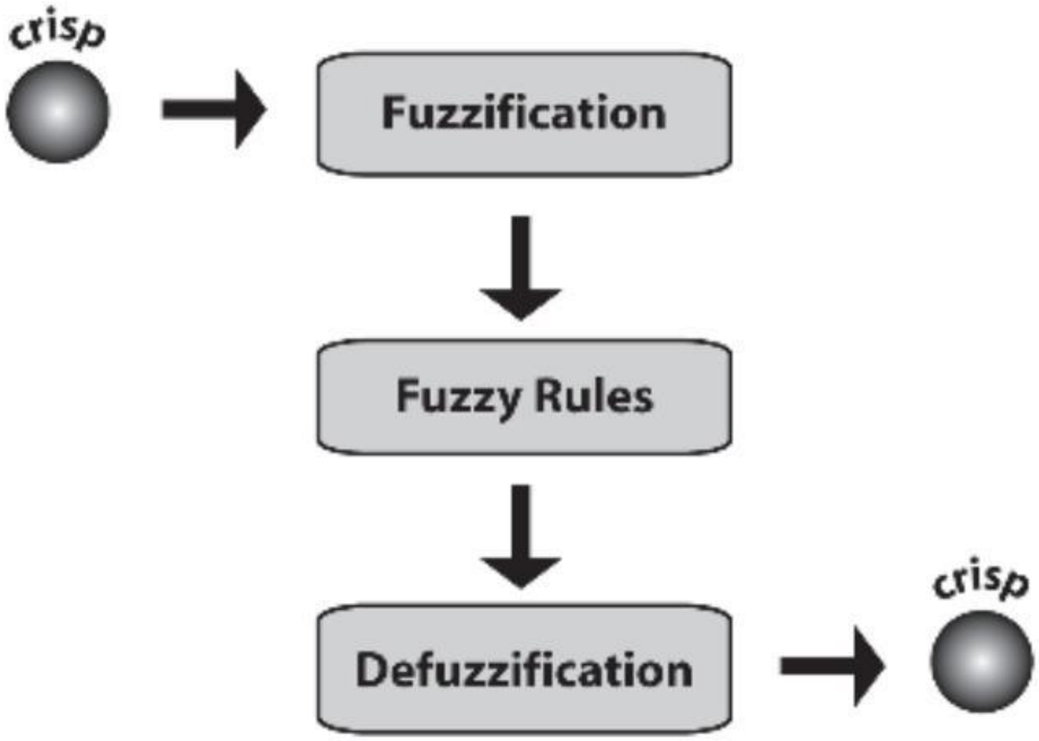
Ασαφής λογική (Fuzzy Logic)

Η ασαφής λογική είναι ένας τρόπος περιγραφής ενός προβλήματος στον υπολογιστή με τον τρόπο που αντιμετωπίζεται από τους ανθρώπους. Η ουσία της ασαφούς λογικής είναι ότι όλα είναι θέμα ποσοστού .

Οι άνθρωποι πολύ συχνά αναλύουν καταστάσεις, ή λύνουν προβλήματα, με ανακρίβεια. Δηλαδή, μπορεί να μην υπάρχουν όλα τα γεγονότα, κάποια από αυτά να είναι αμφιλεγόμενα, ή να μην διαθέσιμα ακριβή δεδομένα ή μετρήσεις.

Για παράδειγμα, έστω ότι γίνεται ένας φιλικός αγώνας basket σε μια γειτονιά. Για να κρίνει κάποιος ποίον αντίπαλο θα μαρκάρει, ενδεχομένως να βασίσει την απόφαση του ανάλογα με το ύψος και την επιδεξιότητα του αντιπάλου. Ίσως να κρίνει ότι είναι ψηλός και γρήγορος ή όχι και τόσο ψηλός και δυσκίνητος. Η ασαφής λογική δίνει τη δυνατότητα παρουσίασης και να λύσης προβλημάτων χρησιμοποιώντας λεκτικούς όρους. Όταν κάποιος λέει ότι ο αντίπαλός του είναι ψηλός μεσαίου αναστήματος ή πολύ ψηλός, πιθανότατα δεν έχει προκαθορισμένα όρια για την κάθε κατηγορία [4].

Τα βασικά στάδια της διαδικασίας που ακολουθούνται παρουσιάζονται στην παρακάτω εικόνα.



Εικόνα 2.11: Διάγραμμα ροής Fuzzy Logic [4]

Το πρώτο κομμάτι ονομάζεται Fuzzification, στο οποίο τα δεδομένα (πραγματικοί αριθμοί) μετατρέπονται σε ασαφή δεδομένα. Δηλαδή από 1,7μ, 1,8μ σε Ψηλός ή Πολύ Ψηλός. Εφόσον όλα μας τα δεδομένα έχουν εκφραστεί με ασαφή τρόπο, μπορούν να συνδυαστούν δημιουργώντας έτσι κανόνες που καθορίζουν το βαθμό στον οποίο κάθε κανόνας είναι έγκυρος. Για παράδειγμα Ψηλός και Γρήγορος σημαίνει Καλή Επιλογή (κανόνας 1) ενώ Ψηλός και Αργός, Μέτρια Επιλογή (κανόνας 2). Ο πρώτος κανόνας μπορεί να έχει ποσοστό συμμετοχής 70% και ο δεύτερος 50% και ονομάζονται Fuzzy Rules. Μέχρι στιγμής έχουν δημιουργηθεί κάποιοι κανόνες και τα ποσοστά τους. Θα πρέπει τώρα να εξαχθεί ένα τελικό νούμερο έτσι ώστε να γίνει η επαναφορά των ασαφών δεδομένων σε κανονικά ώστε να μπορούν να χρησιμοποιηθούν. Αυτή η διαδικασία ονομάζεται defuzzification [4].

2.9.2 Εύρεση καλύτερου μονοπατιού (Pathfinding with A*)

Το pathfinding είναι ένα από τα βασικά προβλήματα στην τεχνητή νοημοσύνη των παιχνιδιών. Ένα φτωχό pathfinding μπορεί να κάνει τους χαρακτήρες ενός παιχνιδιού να αντιδρούν μη ρεαλιστικά και αφελώς. Αυτό που κάνει τον αλγόριθμο A* δημοφιλή είναι το γεγονός ότι εγγυάται να βρει την καλύτερη διαδρομή δεδομένων δύο σημείων, με την προϋπόθεση ότι το μονοπάτι υπάρχει [2]. Επίσης είναι σχετικά γρήγορος, όμως δεν είναι κατάλληλος για όλες τις περιπτώσεις. Για παράδειγμα εάν στον ανοιχτό τρισδιάστατο χώρο, υπάρχει ανεμπόδιστη γραμμή μεταξύ δύο σημείων τότε ο A* είναι περιττός. Με τον όρο περιττός εννοούμε ότι για τη συγκεκριμένη περίπτωση θα μπορούμε να χρησιμοποιήσουμε έναν απλούστερο και γρηγορότερο αλγόριθμο, διότι ο A* θα επιβάρυνε τον επεξεργαστή χωρίς ουσιαστικό λόγο.

O A* χρησιμοποιεί best-first search και βρίσκει το μονοπάτι ελάχιστου κόστους δεδομένου ενός αρχικού και ενός τελικού κόμβου .

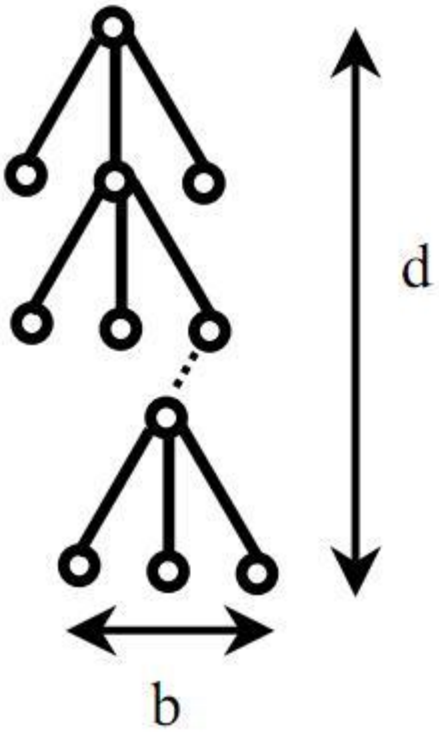
Για να καθορίσει το κόστος χρησιμοποιεί δύο συναρτήσεις f(x) = g(x)+h(x)

- Το κόστος του μονοπατιού από τον αρχικό κόμβο μέχρι τον κόμβο στον οποίο βρισκόμαστε. Συμβολίζεται με $g(x)$.
- Μια αποδεκτή ευρεστική εκτίμηση της απόστασης του παρόντος κόμβου από τον τελικό. Συμβολίζεται με $h(x)$.

Καθώς ο A* διασχίζει (traverses) τον γράφο ακολουθεί το μονοπάτι με το χαμηλότερο κόστος, κρατώντας μία ταξινομημένη ουρά προτεραιότητας με εναλλακτικά μονοπάτια. Εάν σε κάποιο σημείο το κομμάτι του μονοπατιού στο οποίο βρίσκεται έχει μεγαλύτερο κόστος από κάποιο άλλο κομμάτι που συνάντησε, τότε εγκαταλείπει το παρόν κομμάτι και συνεχίζει να διασχίζει από το κομμάτι με το χαμηλότερο κόστος. Αυτό συνεχίζεται μέχρι να φτάσει στο στόχο του [2].

Αναλυτικότερα, ξεκινώντας από τον αρχικό κόμβο, διατηρεί μια ουρά προτεραιότητας με τους κόμβους από τους οποίους έχει περάσει, γνωστή και ως open list. Όσο χαμηλότερο είναι το $f(x)$ για δεδομένο (x) τόσο μεγαλύτερη η προτεραιότητα. Σε κάθε βήμα του αλγόριθμου ο κόμβος με το μικρότερο $f(x)$ διαγράφεται από την ουρά, οι τιμές $f(x)$ και $g(x)$ των γειτόνων ανανεώνονται και οι κόμβοι προστίθενται στην ουρά.. Ο αλγόριθμος συνεχίζει μέχρι ένας από τους τελικούς κόμβους να έχει το χαμηλότερο f(x) ή να αδειάσει η ουρά. Για να εξαχθεί το τελικό μονοπάτι κάθε κόμβος αποθηκεύει τον κόμβο γονέα. Με αυτόν τον τρόπο, ξεκινώντας από το τέλος στην αρχή λαμβάνεται ολόκληρη η διαδρομή.

Κάθε φορά που υπολογίζεται την συνάρτηση h(x), επεκτείνεται ο χώρος αναζήτησης κατά b κόμβους. Για όλους τους b κόμβους θα γίνει άλλη μια επέκταση κατά b κόμβους και ούτω καθεξής. Η επέκταση θα σταματήσει μόνο όταν φτάσει στο επίπεδο που βρίσκεται ο στόχος μας.



Εικόνα 2.12: Δενδρικός χώρος αναζήτησης Α*

Υπολογισμός συνολικού αριθμού κόμβων:

Επίπεδο	Αριθμός κόμβων
0	1
1	b
2	b^2
3	b^3
d	b^d

$N = 1 + b + b^2 + b^3 + \ldots + b^d$ *(1)*

$b * N = b + b^2 + b^3 + b^4 \ldots + b^{d+1}$*(2)*

Αφαιρώντας κατά μέλη (1) - (2) έχουμε:

$$(1-b) * N = b^{d+1} - 1 \Leftrightarrow N = \frac{b^{d+1} - 1}{(1-b)}$$

Από τους παραπάνω υπολογισμούς προκύπτει ότι η πολυπλοκότητα για την χειρότερη περίπτωση είναι O(b^d). Όμως η πολυπλοκότητα μπορεί να γίνει πολυωνυμική εάν ικανοποιούνται οι εξής τρείς συνθήκες. Ο χώρος αναζήτησης είναι δένδρο, υπάρχει μόνο μια κατάσταση τερματισμού και η ευρεστική συνάρτηση ικανοποιεί την συνθήκη $|h(x) - h^*(x)| = O(\log h^*(x))$, όπου h*(x) είναι η βέλτιστη ευρεστική. Η βέλτιστη, ή αλλιώς τέλεια, ευρεστική δεν κάνει εκτίμηση αλλά επιστρέφει την πραγματική απόσταση από τον κόμβο X (κόμβος επεξεργασίας) μέχρι τον τελικό κόμβο.

<u>Ο ψευδοκώδικας του A* [2]</u>

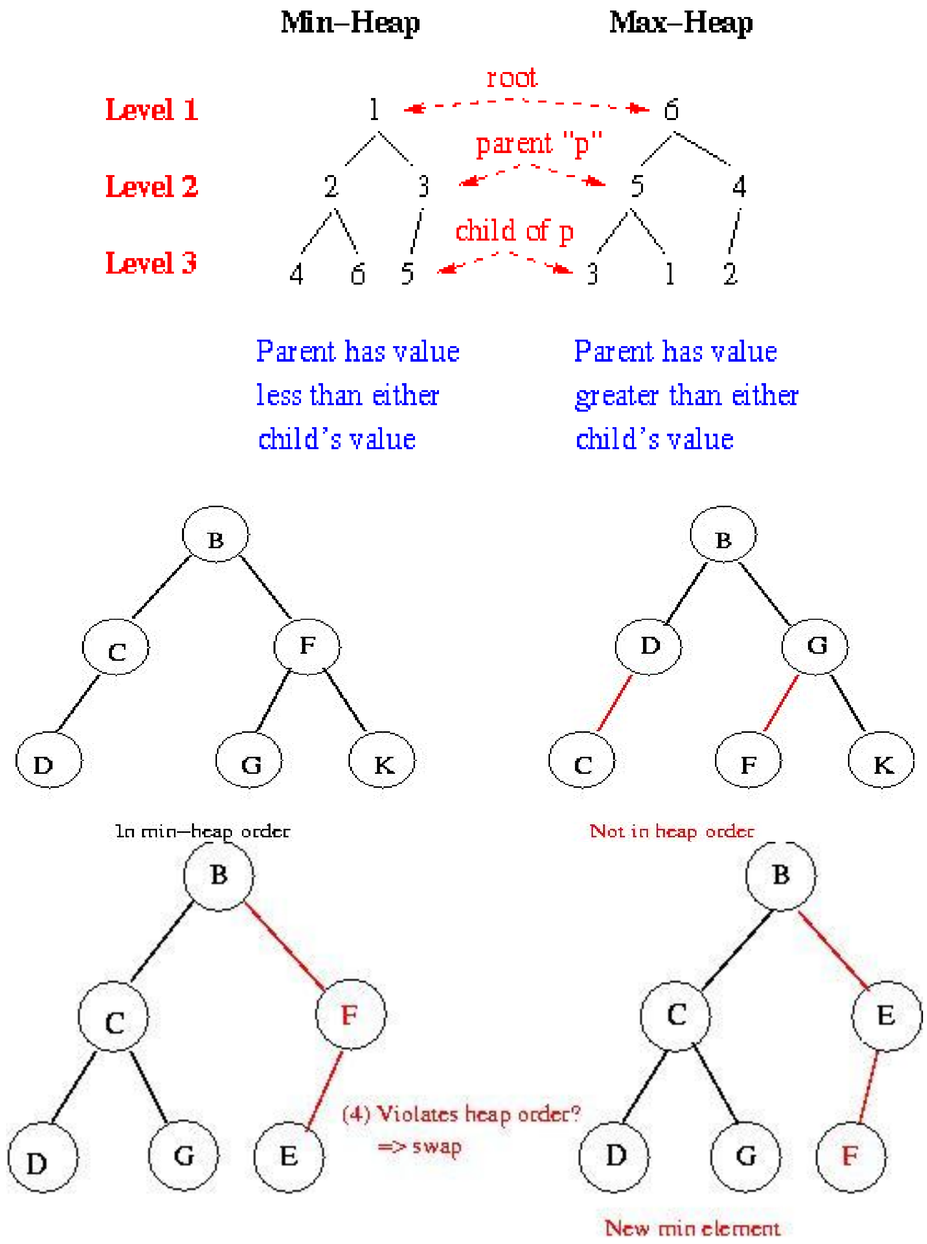
```
//----------------------------------------------------------------------------------------------------

function A*(start,goal)
     closedset:= the empty set                      // The set of nodes already evaluated.
     openset:= set containing the initial node // The set of tentative nodes to be evaluated.
     g_score[start]:= 0                             // Distance from start along optimal path.
     h_score[start]:= heuristic_estimate_of_distance(start, goal)
     f_score[start]:= h_score[start]                   // Estimated total distance from start to goal through y.
     while openset is not empty
/*σημείο βελτίωσης*/
     x:= the node in openset having the lowest f_score[] value
//-------------
         if x = goal
             return reconstruct_path(came_from[goal])
         remove x from openset
         add x to closedset
         foreach y in neighbor_nodes(x)
             if y in closedset
                 continue
             tentative_g_score:= g_score[x] + dist_between(x,y)

             if y not in openset
                 add y to openset
                 tentative_is_better:= true
             elseif tentative_g_score < g_score[y]
                 tentative_is_better:= true
             else
                 tentative_is_better:= false
             if tentative_is_better = true
                 came_from[y]:= x
                 g_score[y]:= tentative_g_score
                 h_score[y]:= heuristic_estimate_of_distance(y, goal)
                 f_score[y]:= g_score[y] + h_score[y]
     return failure

 function reconstruct_path(current_node)
     if came_from[current_node] is set
         p = reconstruct_path(came_from[current_node])
         return (p + current_node)
     else
         return current_node
//-------------------------------------------------------------
```

2.9.3 Βελτίωση του αλγόριθμου Α*

Μια συνηθισμένη βελτίωση του Α* είναι η χρησιμοποίηση ουράς προτεραιότητας (priority queue) για την ανοιχτή λίστα αντί για απλή συνδεδεμένη λίστα. Με αυτόν τον τρόπο όταν αναζητείται ο κόμβος με το μικρότερο f_score η χρονική πολυπλοκότητα από O(N) γίνεται O(1). Η θεωρητική συνολική χρονική πολυπλοκότητα του αλγορίθμου δεν επηρεάζεται, όμως πρακτικά αποφεύγονται πολλές περιττές συγκρίσεις.

Η ουρά προτεραιότητας είναι μια δομή Αφηρημένου Τύπου Δεδομένων με κλειδιά η οποία υποστηρίζει δύο βασικές λειτουργίες: εισαγωγή ενός νέου στοιχείου και αφαίρεση του στοιχείου με το μικρότερο/μεγαλύτερο κλειδί. Οι διάφορες υλοποιήσεις του Α.Τ.Δ. μπορούν να γίνουν με πίνακα, συνδεδεμένη λίστα ή σωρό. Παρακάτω αναλύεται η τρίτη μέθοδος.

Η δομή που ονομάζεται σωρός (heap) μπορεί να υποστηρίξει αποδοτικά τις βασικές λειτουργίες μιας ουράς προτεραιότητας. Σε ένα σωρό, οι εγγραφές αποθηκεύονται σε ένα πίνακα έτσι ώστε κάθε κλειδί να είναι εγγυημένα μεγαλύτερο από τα κλειδιά που βρίσκονται σε δύο άλλες συγκεκριμένες θέσεις. Με τη σειρά τους, καθένα από αυτά τα κλειδιά πρέπει να είναι μεγαλύτερο από δύο ακόμη κλειδιά, και ούτω καθεξής. Αυτή η διάταξη γίνεται αρκετά ευδιάκριτη αν φανταστεί κανείς ότι τα κλειδιά βρίσκονται σε κάποια δομή δυαδικού δέντρου με ακμές από κάθε κλειδί προς τα δύο κλειδιά τα οποία είναι γνωστό ότι είναι μικρότερα από αυτό. Ουσιαστικά ο σωρός είναι ένα σύνολο κόμβων με κλειδιά τοποθετημένα σε ένα πλήρες δυαδικό δένδρο το οποίο είναι διατεταγμένο σε σωρό και αναπαρίσταται ως πίνακας. Το πλήρες δένδρο είναι ένα δένδρο του οποίου κάθε επίπεδο, εκτός ίσως από το τελευταίο, είναι γεμάτο και όλοι οι κόμβοι είναι όσο πιο αριστερά γίνεται [18].



Εικόνα 2.13: Παραδείγματα δομής ελάχιστου/μέγιστου σωρού [15][11][12]

Κεφάλαιο 3

Unity 3D

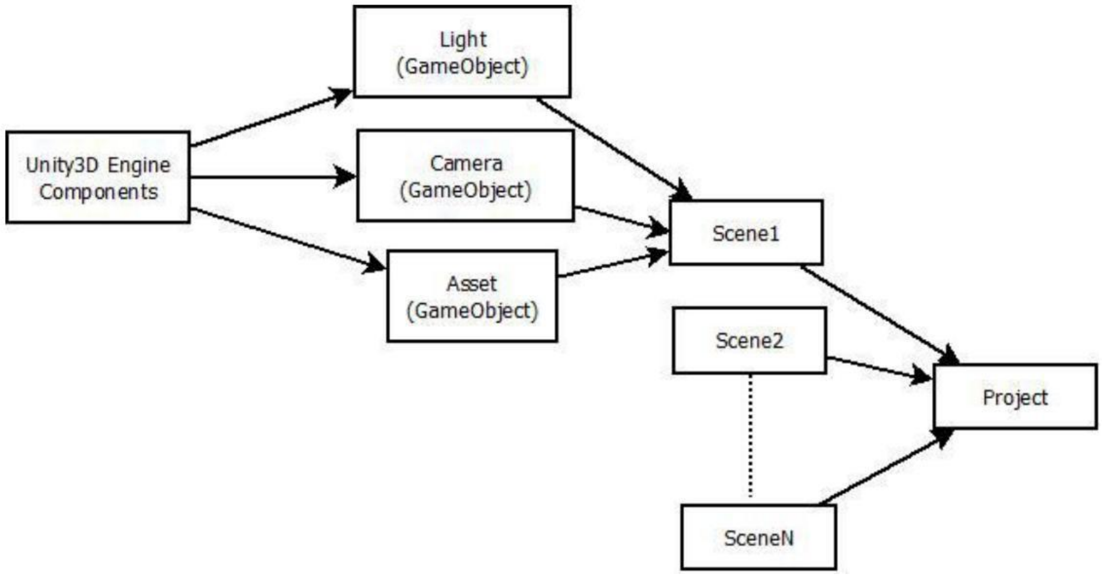
3.1 Εισαγωγή

Η Unity 3D αποτελεί ένα ολοκληρωμένο περιβάλλον ανάπτυξης παιχνιδιών που παρέχει πληθώρα εργαλείων για γρηγορότερη ανάπτυξη και αποσφαλμάτωση. Ορισμένα από αυτά είναι η γραφική διεπαφή για τον έλεγχο των αντικειμένων, πρόσβαση σε όλες τις λεπτομέρειες και ιδιότητες της γεωμετρίας και άμεση προεπισκόπηση του παιχνιδιού.

Ο λόγος για τον οποίο επιλέχθηκε η συγκεκριμένη μηχανή είναι ότι η τεκμηρίωση της διεπαφής για τον προγραμματισμό είναι λεπτομερέστατη αλλά και γιατί η Unity διαθέτει μια πάρα πολύ ενεργή κοινότητα χρηστών από την οποία αντλήσαμε σημαντικές πληροφορίες και βοήθεια. Επίσης η αρχιτεκτονική και η οργάνωση της είναι απλή με αποτέλεσμα να μπορούμε να εστιάζουμε άμεσα στο αντικείμενο μας χωρίς προβλήματα. Η μόνη παράμετρος στην οποία υστερεί η Unity έναντι στις πιο εξελιγμένες μηχανές όπως η UDK είναι ο τομέας των γραφικών. Η έκδοση της Unity(Unity 2.5) που χρησιμοποιήθηκε διαθέτει DirectX 9, ενώ η UDK DirectX 11. Αυτό σημαίνει ότι το εάν αναπτύσσαμε το παιχνίδι μας στην UDK, θα είχε πιο εντυπωσιακά γραφικά. Παράλληλα όμως θα ανέβαιναν και οι απαιτήσεις σε υπολογιστικούς πόρους με αποτέλεσμα να εκτελείται δυσκολότερα σε υπολογιστές παλαιότερης τεχνολογίας.

3.2 Δομή της μηχανής

Για να εργαστούμε σε βάθος με διάφορες πτυχές της μηχανής θα πρέπει μελετηθεί και να γίνει κατανοητή η αρχιτεκτονική της. Κάθε παιχνίδι αποτελεί ένα project και κάθε project περιέχει πολλές σκηνές (scenes), οι οποίες μπορεί να αποτελούν πίστες για το παιχνίδι ή γραφική διεπαφή ή ακόμα και συνδυασμό των δύο, για την δημιουργία Heads Up Display. Όποιο αντικείμενο υπάρχει σε μία σκηνή (scene) χαρακτηρίζεται ως GameObject, τα οποία μπορεί να έχουν πολλά χαρακτηριστικά που ονομάζονται συστατικά (Components) [22].



Εικόνα 3.1: Ιεραρχία δομικών μονάδων [24]

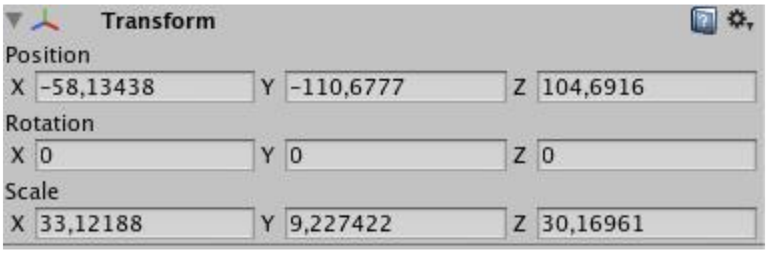
Εικόνα 3.2: Τα components από τα οποία αποτελείται η Unity 3D [22]

3.3 Τοποθέτηση (Transform)

Το component Transform πρέπει να υπάρχει υποχρεωτικά σε κάθε αντικείμενο που βρίσκεται σε μία σκηνή και καθορίζει τη θέση, την περιστροφή και τη διάσταση του αντικειμένου.

Ιδιότητες:

- position x,y,z: προσδιορίζει την θέση του αντικειμένου στον χώρο
- scale x,y,z: αυξομειώνει το μέγεθος του αντικειμένου είτε είναι γεωμετρία είτε αντικείμενο γραφικής διεπαφής (GUI)
- rotation x,y,z: περιστροφή αντικειμένου είτε είναι γεωμετρία είτε αντικείμενο γραφικής διεπαφής (GUI).

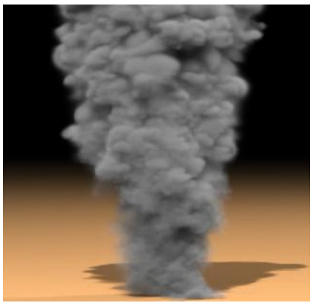


Εικόνα 3.3: Transform component

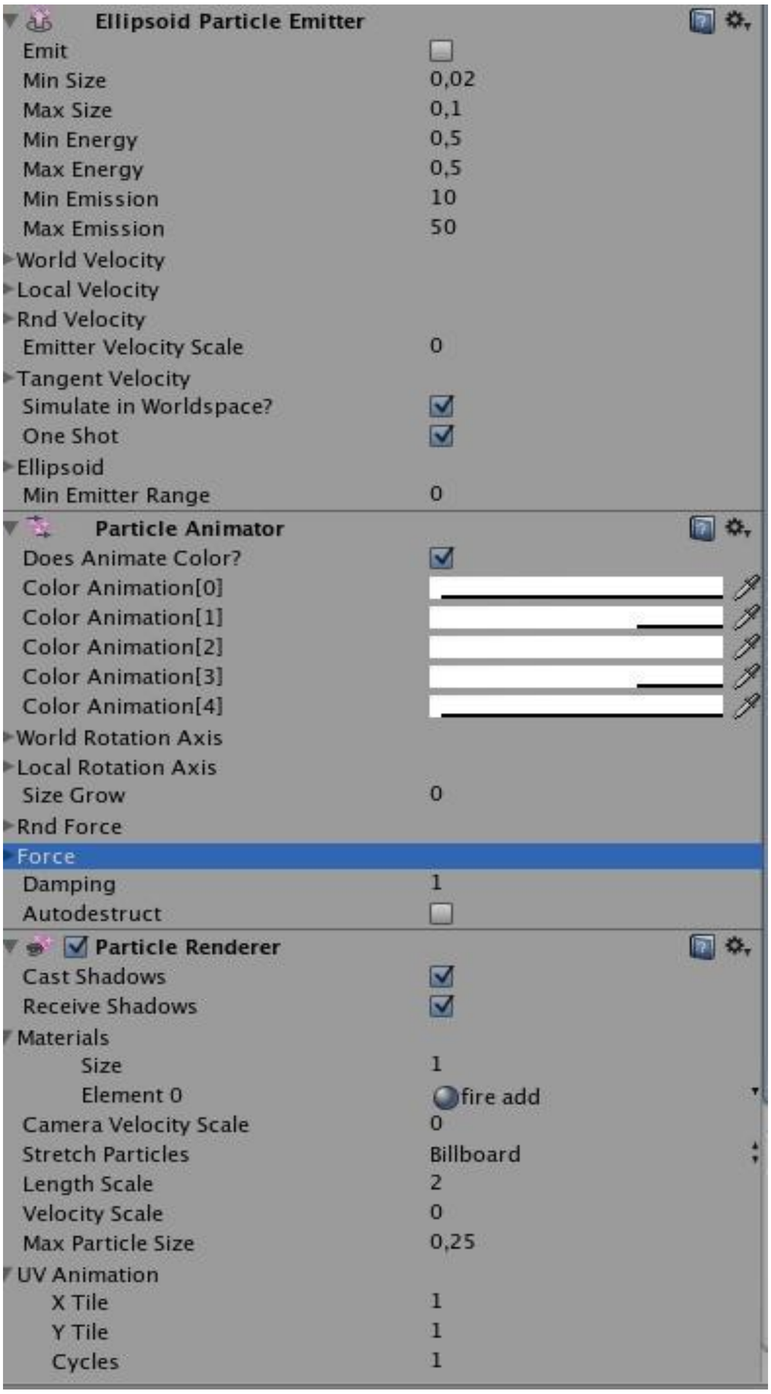
3.4 Σωματίδια (Particles)

Τα particles είναι σωματιδιακά εφέ για την δημιουργία εκρήξεων, σκόνης, καπνού κ.α. Τα τρία βασικότερα συστατικά είναι: Emitter, Animator, Renderer, κάθε ένα από τα οποία διαθέτει πολύ μεγάλο αριθμό μεταβλητών και επιλογών.

Η θέση και η κίνηση των σωματιδίων στον τρισδιάστατο χώρο ελέγχεται από τον emitter, ο οποίος λειτουργεί ως πηγή σωματιδίων και η θέση του καθορίζει το σημείο γέννησης τους. Μερικές από τις παραμέτρους που μπορούμε να μεταβάλουμε είναι το ελάχιστο/μέγιστο μέγεθος, η ελάχιστη/μέγιστη ενέργεια και η ταχύτητα. Μετά τη δημιουργία τους, ο animator θα αναλάβει για την περαιτέρω κίνηση τους. Για να αποδοθεί μεγαλύτερος ρεαλισμός μπορούν να εφαρμοστούν στα σωματίδια διάφορες δυνάμεις όπως βαρύτητα, έλξη(drag), διαταραχή(turbulence) και να συμπεριφέρονται ως καπνός, έκρηξη ή σκόνη. Τέλος ο particle renderer αναλαμβάνει να εμφανίσει τα particles στην οθόνη. Οι ρυθμίσεις του αφορούν την εμφάνιση των σωματιδίων αλλά και την αλληλεπίδραση με το φως [22].



Εικόνα 3.4: Παράδειγμα καπνού με Particles [1]



Εικόνα 3.5: Particle component

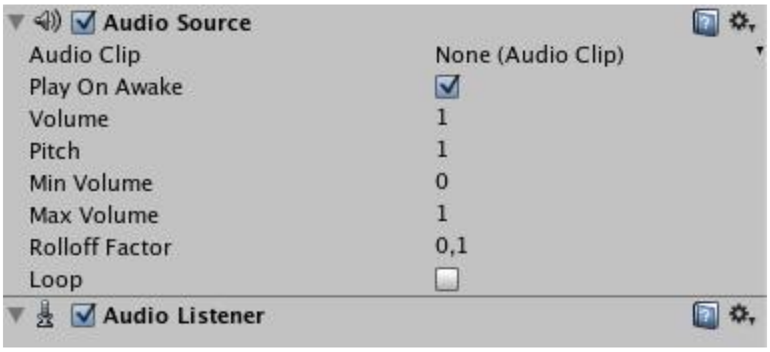
3.5 Ήχος

Η υπομονάδα ήχου διαθέτει δύο συστατικά.

Audio Listener: Μονάδα που λαμβάνει ήχους από τη σκηνή. Ουσιαστικά βοηθάει στην αυξομείωση του ήχου ανάλογα με το που βρισκόμαστε μέσα σε μία σκηνή.

Audio Source: Παρέχει την δυνατότητα σύνδεσης του αντικειμένου μας με ένα ηχητικό αρχείο και να ελέγξουμε την αναπαραγωγή του μέσω κώδικα.

Εάν θέλουμε μπορούμε να φτιάξουμε ένα πιο εξελεγμένο σύστημα ήχου όπως για παράδειγμα διαχείριση πολλών ήχων που παίζουν ταυτόχρονα έτσι ώστε να μην γίνονται ενοχλητικοί. Ένα τέτοιο σύστημα θα αναπτυχθεί εξ ολοκλήρου από εμάς με την χρήση κώδικα, αφού η Unity δεν μας παρέχει επιπλέον εργαλεία περά από τα δύο προαναφερθέντα συστατικά.



Εικόνα 3.6: Audio Source/Listener component

3.6 Unity GUI

Τα μενού, αλλά και όλες οι γραφικές διεπαφές στο παιχνίδι θα πρέπει να έχουν μια μορφή, η οποία καθορίζεται από πολλές παραμέτρους όπως το χρώμα και το μέγεθος των γραμμάτων.

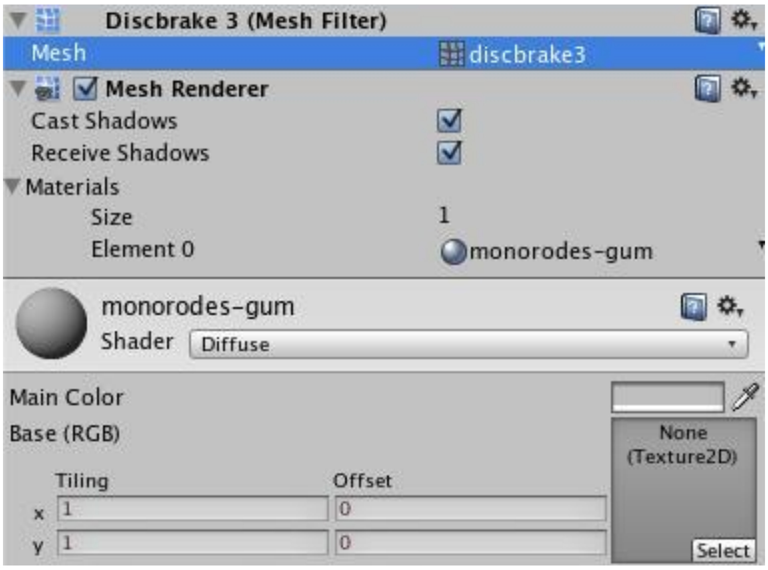
GUI Style, GUI Skin: Παρέχουν μεταβλητές για την ρύθμιση της εμφάνισης της γραφικής διεπαφής. Ένα GUI Skin μπορεί να περιέχει πολλά GUI Styles, ένα για τα κουμπιά άλλο για τις επικεφαλίδες και ούτω καθεξής.

Εικόνα 3.7: Gui skin parameters

3.7 Εισαγωγή δικτύων γεωμετρίας (Mesh)

Η γεωμετρία δημιουργείται με τη βοήθεια κάποιου προγράμματος μοντελοποίησης. Μερικά από τα καλύτερα προγράμματα είναι τα: 3ds Max, Maya, Blender, Cinema 4D, και Cheetah3D. Αφού ολοκληρώσουμε τη μοντελοποίηση ανοίγουμε την Unity και κάνουμε εισαγωγή της γεωμετρίας από το μενού, οπότε και γίνεται διαθέσιμη για χρήση σε οποιοδήποτε επίπεδο. Τοποθετώντας τη γεωμετρία σε κάποιο επίπεδο μπορούμε να προσθέσουμε επιπλέον components με τη δυνατότητα να κάνουμε περισσότερα. Να σημειωθεί ότι εάν εισάγουμε την γεωμετρία σε μία πίστα χωρίς επιπλέον components, θα έχουμε ένα ακίνητο αντικείμενο. Σε ένα παιχνίδι όμως χρειάζεται αλληλεπίδραση, άλλωστε αυτός είναι ο λόγος που προσθέτουμε επιπλέον components. Για παράδειγμα η γεωμετρία ενός αυτοκινήτου χρειάζεται ένα component physics και ένα ήχου, ώστε να κινείται ρεαλιστικά αλλά και να ακούγεται ότι η μηχανή του δουλεύει.

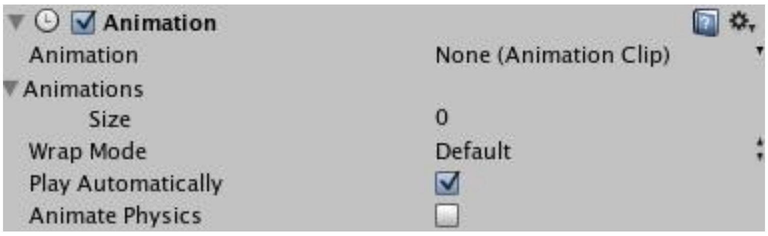
Όταν εισάγουμε γεωμετρία στην Unity, αυτομάτως αποκτά δύο components: το ένα είναι υπεύθυνο να σχηματίσει την γεωμετρία της πίστας και το άλλο είναι υπεύθυνο για την εμφάνισή της. Δηλαδή τι σκιαστή (shader) και χρώμα θα χρησιμοποιηθεί και εάν το αντικείμενο αυτό θα αλληλεπιδρά με το φώς και θα ρίχνει σκιές σε άλλα αντικείμενα. Τα δύο αυτά components ονομάζονται Mesh filter και Mesh renderer αντίστοιχα.



Εικόνα 3.8: Mesh component

3.8 Κίνηση (Animation)

Το component αυτό διαχειρίζεται animations τα οποία εισάγονται από κάποιο πρόγραμμα μοντελοποίησης. Ο έλεγχος της ροής γίνεται μέσα από script το οποίο είναι ευθύνη του προγραμματιστή.



Εικόνα 3.9: Animation component

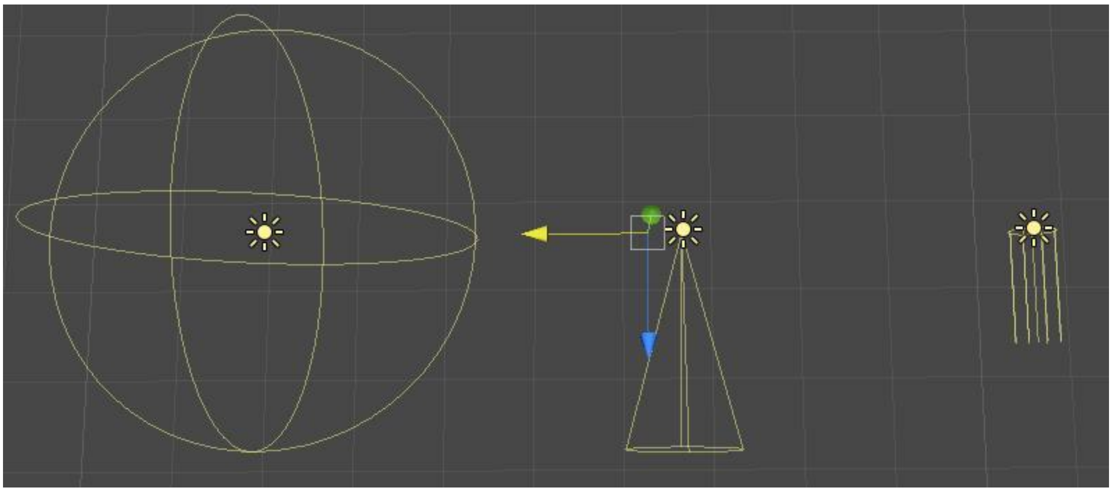
3.9 Δίκτυο (Network)

Κάθε αντικείμενο που θέλουμε να ελέγχεται μέσω δικτύου πρέπει υποχρεωτικά να έχει αυτό το component. Παρέχει πολύ εύκολο έλεγχο και σε υψηλό επίπεδο καλύπτοντας έτσι την ανάγκη για δημιουργία sockets χειροκίνητα.

Εικόνα 3.10: Network view component

3.10 Φώτα, Υφές, Κάμερα (Lights, textures, camera)

Η Unity διαθέτει τρεις διαφορετικούς τύπους πηγών φωτισμού: Point, Spot και directional light, οι οποίοι χρησιμοποιούνται σε προγράμματα τρισδιάστατης μοντελοποίησης και αποτελούν τη βάση για το φωτισμό σε όλες τις πτυχές των γραφικών υπολογιστών. Το Point light από το σημείο στο οποίο βρίσκεται εκπέμπει φως προς όλες τις κατευθύνσεις, σε αντίθεση με το Spot light το οποίο ρίχνει το φως σε συγκεκριμένο σημείο και μάλιστα σε σχήμα κώνου. Αυτοί οι δύο τύποι φωτισμού λειτουργούν καλά για μικρές περιοχές. Επειδή στη συγκεκριμένη εφαρμογή πρέπει να φωτίσουμε μια ολόκληρη πίστα, χρησιμοποιούμε το directional light, το οποίο είναι ουσιαστικά ένα περιστρεφόμενο επίπεδο στην κατεύθυνση που θέλουμε να πέφτει το φώς και με αποτέλεσμα όλα τα αντικείμενα στη σκηνή να φωτίζονται με τον ίδιο τρόπο από μια συγκεκριμένη κατεύθυνση. Όλα τα φώτα μοιράζονται κάποιες βασικές ιδιότητες, εκ των οποίων οι κυριότερες είναι η ένταση και το χρώμα φωτός, ο τύπος και η ανάλυση σκιών και το μέγεθος.



Εικόνα 3.11: Point, Spot, Directional Light

Όσον αφορά τα textures η Unity υποστηρίζει σχεδόν όλες τις νέες τεχνολογίες και συγκεκριμένα τις Bump mapping, Reflection mapping, Parallax mapping και render-to-texture. Με την τελευταία ιδιότητα μπορούμε να χρησιμοποιήσουμε ένα video ως υφή πάνω σε ένα αντικείμενο.

Όταν εισάγουμε γεωμετρία σε μια σκηνή αυτομάτως προστίθεται το component Mesh renderer, το οποίο παρουσιάσαμε προηγουμένως. Σε αυτό υπάρχει μια επιλογή που λέγεται Materials, τα οποία είναι textures και shaders ρυθμισμένα από τον προγραμματιστή έτσι ώστε να χρησιμοποιούνται κατευθείαν στην γεωμετρία. Για παράδειγμα ένα Material μπορεί να έχει Diffuse share για να πετύχουμε το εφέ της ματ επιφάνειας και για texture μια εικόνα βράχων. Έχοντας την αντίστοιχη γεωμετρία προσθέτουμε το material και το αντικείμενο μας είναι πλέον ολοκληρωμένο.

Η κάμερα αποτελεί την οπτική γωνία του παιχνιδιού. Όταν δημιουργούμε ένα καινούργιο επίπεδο η Unity τοποθετεί στην σκηνή μια κάμερα την οποία ονομάζει MainCamera. Εάν θέλουμε να την διαγράψουμε, μπορούμε να το κάνουμε, αλλά θα πρέπει να ορίσουμε κάποια άλλη ως MainCamera στη θέση της προηγούμενης. Εάν η σκηνή δεν έχει κανένα game Object κάμερας τότε δεν θα βλέπουμε τίποτα όταν θα τρέξουμε το παιχνίδι. Μερικές από τις ιδιότητες της κάμερας είναι το βάθος πεδίου, η απόσταση και η γωνία μέχρι τη οποία «βλέπει» και το χρώμα φόντου το οποίο είναι το χρώμα που δείχνει η κάμερα στην περίπτωση πού δεν υπάρχει γεωμετρία στο οπτικό της πεδίο. Επίσης οι κάμερες ελέγχονται πλήρως μέσω κώδικα script κάτι που είναι πολύ χρήσιμο σε περιπτώσεις που η κάμερα ακολουθεί κάποιο αντικείμενο και πρέπει να προσαρμόζει την θέση της ανάλογα με τις κινήσεις του αντικειμένου.

3.11 Σκιαστές (Unity Shaders)

3.11.1 Σκιαστής διάχυσης (Diffuse)

Ο σκιαστής διάχυσης υπολογίζει ένα απλό φωτιστικό μοντέλο (Lambertian). Ο φωτισμός σε μια επιφάνεια ελαττώνεται καθώς η γωνία μεταξύ αυτού και του φωτός ελαττώνεται, εξαρτάται μόνο από αυτήν την γωνία και δεν αλλάζει όταν η κάμερα μετακινείται ή περιστρέφεται. Είναι σκιαστής φωτισμένος ανά εικονοστοιχείο και το πιο φτηνό μοντέλο σκιαστή.

3.11.2 Σκιαστής ανάκλασης (Specular)

Ο σκιαστής ανάκλασης υπολογίζει το φωτιστικό μοντέλο Blinn. Κυρίως εφαρμόζεται σε λείες επιφάνειες όπου είναι γνωστό ότι το μεγαλύτερο μέρος των ακτινών ανακλάται. Ένα παράδειγμα είναι οι γυαλισμένες μεταλλικές επιφάνειες των αυτοκινήτων.

3.11.3 Σκιαστής διαφάνειας (Transparent)

Μπορεί να κάνει το δικτύωμα της γεωμετρίας να είναι μερικώς ή εντελώς διαφανές διαβάζοντας το κανάλι Alpha της κύριας υφής. Το μαύρο είναι εντελώς διάφανο και το άσπρο εντελώς αδιαφανές. Εάν η κύρια υφή δεν έχει Alpha κανάλι το αντικείμενο θα είναι αδιαφανές.

3.11.4 Σκιαστής διάχυσης/ανάκλασης ανώμαλης επιφάνειας (Bumped Diffuse/Specular)

Ο σκιαστής ανώμαλης επιφάνειας υπολογίζει την διάχυση/ανάκλαση όπως στον σκιαστή διάχυσης/ανάκλασης. Για την ανώμαλη επιφάνεια χρησιμοποιεί μια υφή η οποία ονομάζεται χάρτης ανώμαλης επιφάνειας (Bump or Normal map) για να δημιουργήσει το εφέ του βάθους χωρίς αλλαγή στη γεωμετρία. Στον χάρτη ανώμαλης επιφάνειας, κάθε τιμή χρώματος του εικονοστοιχείου αντιπροσωπεύει την γωνία του διανύσματος που είναι κάθετο στην επιφάνεια του πολυγώνου (surface normal). Έπειτα χρησιμοποιώντας αυτήν την τιμή υπολογίζεται ο φωτισμός αντί της γεωμετρίας. Ο χάρτης ανώμαλης επιφάνειας αντιπαρέρχεται την γεωμετρία του δικτυώματος (mesh geometry) όταν υπολογίζεται ο φωτισμός του αντικειμένου.

3.11.5 Σκιαστής νερού (water shader)

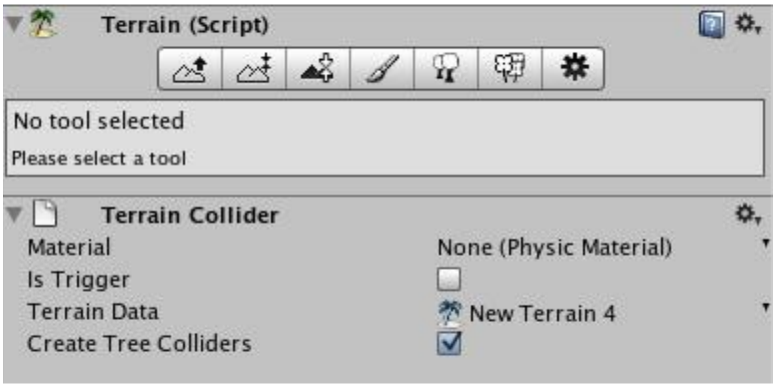
Ο σκιαστής νερού χρησιμοποιεί μια επίπεδη επιφάνεια, τοποθετημένη οριζόντια σαν το δικτύωμα νερού. Αλλάζοντας τις ιδιότητες του σκιαστή μπορούμε να το προσαρμόσουμε στις εκάστοτε ανάγκες. Για το εφέ του κυματισμού χρησιμοποιούνται δύο υφές οι οποίες συνεχώς αλλάζουν θέση και ταχύτητα.

3.12 Προκατασκευές (Prefabs)

Τα Prefabs είναι από τα πιο σημαντικά εργαλεία που παρέχει η Unity. Κάθε αντικείμενο που υπάρχει στο παιχνίδι διαθέτει μία πληθώρα στοιχείων(Components). Επειδή δεν είναι δυνατόν κάθε φορά που εμφανίζεται ένα αντικείμενο στο παιχνίδι να προσθέτουμε όλα αυτά τα στοιχεία ορίζουμε ένα prefab, δηλαδή μία οντότητα που περιλαμβάνει όλα τα components που χρειαζόμαστε και όλες οι ρυθμίσεις έχουν γίνει κατά την ανάπτυξη του παιχνιδιού.

3.13 Έδαφος (Terrain)

Η μετάφραση της λέξεως Terrain είναι έδαφος. Η δημιουργία μεγάλων περιοχών από κάποιο πρόγραμμα μοντελοποίησης είναι μια χρονοβόρα διαδικασία. Εκτός από αυτό απαιτείται πολύ μεγάλος αριθμός πολυγώνων για την αναπαράσταση υψωμάτων ή κοιλωμάτων στο έδαφος, κάτι που θα κάνει το παιχνίδι μας αργό. Η Unity διαθέτει ένα εργαλείο για τη δημιουργία Terrain. Όπως όλα στην Unity έτσι και το Terrain αποτελεί ένα GameObject που περιέχει κάποια Components.



Εικόνα 3.12: Terrain component

Το script που διαθέτει προσφέρει κάποια εργαλεία για τη διαμόρφωση του εδάφους. Μπορούμε να ανυψώσουμε ή να χαμηλώσουμε κάποιο σημείο ή ακόμα να το εξομαλύνουμε σε περίπτωση που έχουν δημιουργηθεί πολύ απότομες αυξομειώσεις. Επίσης μπορούμε να τοποθετήσουμε διάφορες εικόνες (textures) κάνοντας πολυχαρτογράφηση (blending ή multitexturing) και να καταλήξουμε στον χρωματισμό εδάφους που επιθυμούμε. Μία ακόμη επιλογή είναι η προσθήκη δένδρων. Κατασκευάζοντας ένα δέντρο σε ένα πρόγραμμα μοντελοποίησης, το εισάγουμε στην Unity και στην συνέχεια με το εργαλείο “Place Trees” και με αυτοματοποιημένο τρόπο δημιουργούμε πολλά όμοια δέντρα. Μπορούμε να ρυθμίσουμε το μέγιστο ύψος και το πλάτος καθώς και την διασπορά γύρω από αυτές τις τιμές ώστε να υπάρχει μια ανομοιομορφία. Επιπλέον μπορούμε να προσθέσουμε εφέ δύο διαστάσεων τα λεγόμενα (paint details), κάθε ένα από τα οποία είναι ένα texture το οποίο η Unity μπορεί να πολλαπλασιάσει όπως ακριβώς συμβαίνει και με τα δέντρα. Τέλος, για το φωτισμό και τη σκίαση του εδάφους χρησιμοποιείται η τεχνικής χαρτογράφησης φωτός (lightmapping) και αυτό διότι το έδαφος είναι στατικό αντικείμενο και οι δυναμικές σκιές θα επιβάρυναν χωρίς λόγο την κάρτα γραφικών.

3.14 Φυσική (Physics)

Η Unity διαθέτει τη μηχανή φυσικής Ageia PhysX για την προσομοίωση της συμπεριφοράς των αντικειμένων ως πραγματικά στερεά άκαμπτα σώματα.

3.14.1 Ανίχνευση συγκρούσεων

Όπως έχει αναφερθεί σε προηγούμενο κεφάλαιο οι μηχανές αυτές αναλαμβάνουν έκτος των άλλων και την ανίχνευση συγκρούσεων. Για να γίνει αυτό πρέπει να οριοθετήσουμε με συγκεκριμένο τρόπο την γεωμετρία. Η Unity παρέχει τους συγκρουστές (colliders), οι οποίοι ανήκουν στην κατηγορία των συστατικών (Components) και μπορούν να τοποθετηθούν σε οποιαδήποτε γεωμετρία. Οι συγκρουστές είναι πέντε: σφαιρικός, ορθογώνιος, κυλινδρικός, εξειδικευμένος για τροχούς και ένας collider που παίρνει το σχήμα της γεωμετρίας μας (Mesh collider) και κάθε ένας διαθέτει τέσσερα βασικά χαρακτηριστικά:

Physic Material: προσομοιώνει την επιφάνεια ενός φυσικού υλικού. Για παράδειγμα άλλη συμπεριφορά έχει ένα ξύλινος κύβος πάνω σε άσφαλτο και άλλη ένας μεταλλικός.

Is Trigger: Ένας collider μπορεί να ενημερώνει ότι έγινε σύγκρουση με άλλη γεωμετρία χωρίς όμως η μηχανή φυσικής να κάνει κάτι επιπλέον. Εάν θέλουμε να συμβαίνει αυτό ενεργοποιούμε την επιλογή isTrigger.

Size, Center: Το μέγεθος και το κέντρο του συγκρουστή.

Παρακάτω αναλύουμε τον εξειδικευμένο συγκρουστή για ρόδες που είναι ένα από τα σημαντικότερα πλεονεκτήματα της Unity για την ανάπτυξη παιχνιδιών με αυτοκίνητα, έναντι των άλλων μηχανών παιχνιδιών.

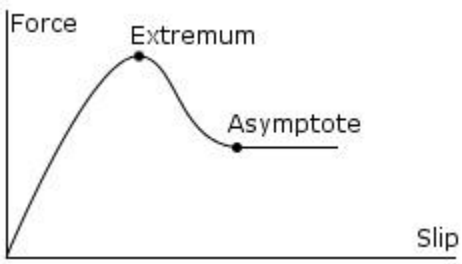
Ιδιότητες wheel collider

Center:	Το κέντρο του τροχού.
Radius:	Η ακτίνα του τροχού.
Suspension Distance:	Μέγιστη απόσταση για την επέκταση της ανάρτησης, σε τοπικές συντεταγμένες. Η ανάρτηση πάντα επεκτείνεται κατά μήκος του άξονα y.
Suspension Spring:	Η εξομοίωση του ελατηρίου.
Spring:	Η δύναμη του ελατηρίου προσπαθεί να φτάσει μια απόσταση στόχο. Μια μεγαλύτερη τιμή κάνει το ελατήριο να προσεγγίσει αυτήν την απόσταση πιο γρήγορα.
Damper:	Η μεταβλητή αυτή επηρεάζει την απορρόφηση κραδασμών. μια μεγαλύτερη τιμή κάνει το ελατήριο να επιστρέφει πιο αργά στην θέση του.
Target Position:	Η απόσταση στόχου κατά μήκος της μέγιστης απόστασης επέκτασης. Η τιμή μηδέν αντιστοιχεί σε πλήρως εκτεταμένη ανάρτηση και η τιμή ένα σε πλήρως συμπιεσμένη.
Mass:	Η μάζα του τροχού.
Forward/Sideways Friction:	Προσημείωση τριβής για τους τροχούς.

Η ανίχνευση σύγκρουσης για τη ρόδα γίνεται ρίχνοντας μια ακτίνα από το κέντρο προς τα κάτω, πάνω στον τοπικό άξονα Y. Το όχημα ελέγχεται μέσω κώδικα script χρησιμοποιώντας μεταβλητές όπως οι: motorTorque, brakeTorque και steerAngle. Ο υπολογισμός της τριβής γίνεται ξεχωριστά από τις άλλες εξομοιώσεις της μηχανής για περισσότερο ρεαλισμό με βάση ένα μοντέλο καμπυλών.

Η τριβή του ελαστικού μπορεί να περιγραφεί από την καμπύλη στην παρακάτω εικόνα. Υπάρχουν ξεχωριστές καμπύλες για την μπροστινή και πλάγια τριβή που δημιουργείται πάνω στον τροχό. Και στις δύο κατευθύνσεις πρώτα υπολογίζεται η ολίσθηση με βάση την ταχύτητα του τροχού και του δρόμου και στην συνέχεια από την τιμή της ολίσθησης βρίσκουμε την δύναμη που ασκείται στο λάστιχο στο σημείο επαφής.

Η καμπύλη παίρνει την ολίσθηση και δίνει μια ποσότητα δύναμης ως έξοδο. Η προσέγγιση γίνεται με spline δύο κομματιών. Το πρώτο κομμάτι ξεκινάει από το (0,0) μέχρι το *(ExtremumSlip, ExtremumValue)*, στο οποίο η εφαπτομένη της καμπύλης είναι μηδέν. Το δεύτερο κομμάτι ξεκινάει από το *(ExtremumSlip, ExtremumValue)* μέχρι το *(AsymptoteSlip, AsymptoteValue)* όπου η εφαπτομένη είναι πάλι μηδέν [22].

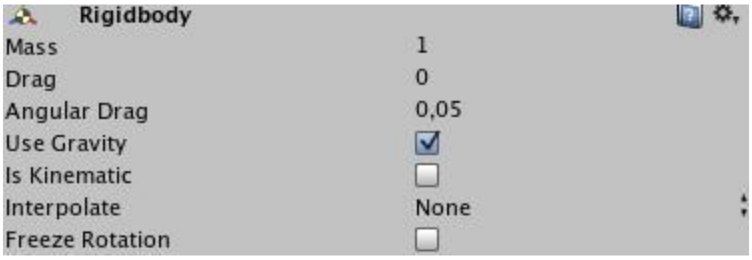


Εικόνα 3.13: Τυπική μορφή καμπύλης ροπής για τροχούς

Αυτό που συμβαίνει με τους πραγματικούς τροχούς είναι ότι για μικρή τιμή ολίσθησης μπορούν να ασκήσουν μεγάλες δυνάμεις, αφού το ελαστικό αντισταθμίζει την ολίσθηση με τη διαστολή. Όταν η ολίσθηση μεγαλώσει πολύ οι δυνάμεις μειώνονται καθώς το λάστιχο γλιστράει ή σπινάρει [22].

3.14.2 Εξομοίωση Φυσικής

Εκτός όμως από την ανίχνευση συγκρούσεων οι μηχανές φυσικής εξομοιώνουν και φυσικά συστήματα. Το βασικότερο από αυτά είναι η επίδραση της βαρύτητας. Για να υπολογίσει η μηχανή τη βαρύτητα σε ένα αντικείμενο θα πρέπει να τοποθετήσουμε στη γεωμετρία το Component: Rigidbody. Η γεωμετρία θα συμπεριφέρεται ανάλογα με ρύθμιση των μεταβλητών, οι οποίες φαίνονται στην παρακάτω εικόνα.



Εικόνα 3.14: Rigidbody component

Εάν θέλουμε να προσθέσουμε στο παιχνίδι μας αντικείμενα όπως πύραυλοι ή ρουκέτες μπορούμε να χρησιμοποιήσουμε το Component: Constant Force, το οποίο ασκεί μια συνεχή νευτώνεια δύναμη στο άκαμπτο σώμα (rigidbody) που διαθέτει η γεωμετρία.

Εικόνα 3.15: Constant force component

3.15 Βασικό Scripting

Κάθε Game Object μπορεί να έχει έναν αυθαίρετο αριθμό από scripts. Η Unity μας δίνει τη δυνατότητα προγραμματισμού σε τρείς γλώσσες: C#, BOO, Javascript. Οι τρείς γλώσσες υλοποιούν πλήρως το API της Unity με τη διαφορά ότι η Javascript κληρονομεί αυτομάτως την κλάση MonoBehaviour ενώ στις άλλες δύο πρέπει να το δηλώσουν σε κάποιο σημείο του κώδικα. Επειδή δεν είναι δυνατό στα πλαίσια της παρούσας αναφοράς να παρουσιάσουμε ολόκληρο το API της Unity, θα αναφερθούμε στα σημαντικότερα σημεία.

3.15.1 Εκκίνηση\Αρχικοποίηση

Ας θεωρήσουμε ότι έχουμε ένα αντικείμενο στην πίστα μας με ένα script ως Component και ξεκινάμε τη λειτουργία του παιχνιδιού. Αρχικά εκτελείται η συνάρτηση Awake() και στη συνέχεια η συνάρτηση Start(). Ο λόγος που υπάρχουν δύο συναρτήσεις εκκίνησης είναι κυρίως πρακτικός. Υπάρχουν περιπτώσεις όπου έχουμε αντικείμενα που εξαρτώνται από άλλα και θέλουμε κατά την εκκίνηση να αρχικοποιηθούν με κάποια σειρά. Οι δύο παραπάνω συναρτήσεις δεν είναι απαραίτητο να υλοποιούνται σε κάθε script που φτιάχνουμε. Μπορούμε να τις χρησιμοποιήσουμε με κάθε δυνατό συνδυασμό αρκεί να θυμόμαστε ότι πρώτα εκτελούνται όλες οι Awake() από όλα τα script στην πίστα μας και στη συνέχεια οι Start(). Αφού ολοκληρωθεί η εκτέλεση τους, ο έλεγχος πηγαίνει στην συνάρτηση Update().

3.15.2 Κύκλος παιχνιδιού

Η συνάρτηση Update() μπορεί να διεκπεραιώσει κάθε δυνατή λειτουργία εκτός από τη γραφική διεπαφή του χρήστη(GUI) για την οποία υπεύθυνη είναι η συνάρτηση OnGUI(). Για να κατανοήσουμε τη λειτουργία της Update() θα πρέπει να εξηγήσουμε τον όρο frame rate. Στα κινούμενα σχέδια, frame rate είναι ο αριθμός τον διαφορετικών εικόνων που περνούν μπροστά από την οθόνη στην διάρκεια ενός δευτερολέπτου. Για να έχουμε κινούμενη εικόνα πρέπει ο ρυθμός των διαφορετικών εικόνων να είναι μεγαλύτερος από είκοσι τέσσερις εικόνες το δευτερόλεπτο. Στα παιχνίδια ο ορισμός είναι λίγο διαφορετικός. Ένα frame είναι ο χρόνος που χρειάζεται ένα σύστημα για να ολοκληρώσει έναν πλήρη κύκλο εργασιών. Εάν το frame rate είναι 60Hz, το σύστημα επαναπροσδιορίζει όλες τις απαραίτητες εισόδους και ανανεώνει τις απαραίτητες εξόδους εξήντα φορές το δευτερόλεπτο. Η συνάρτηση Update() λοιπόν επαναλαμβάνεται σε κάθε frame και με αυτόν τον τρόπο επιτυγχάνεται ομαλή κίνηση των αντικειμένων στον τρισδιάστατο χώρο. Επειδή το frame rate εξαρτάται και από το υλικό του υπολογιστή, μπορούμε μέσω κώδικα να θέσουμε ένα target frame rate και το σύστημα να προσπαθεί να το φτάσει όποτε έχει την δυνατότητα.

Όταν ένα αντικείμενο εμπλέκεται στην εξομοίωση φυσικής η ανανέωση των μεταβλητών του σε κάθε frame δεν είναι πολύ αποδοτική. Το fixed frame rate είναι ένας προκαθορισμένος ρυθμός πολύ πιο γρήγορος από τον κανονικό και αφορά τα αντικείμενα που έχουν το component RigidBody. Για να μπορέσει ένα αντικείμενο να ενημερώσει τις τιμές του με βάση το fixed frame rate θα πρέπει να υλοποιεί την συνάρτηση FixedUpdate() η οποία πέρα από το ρυθμό ανανέωσης δεν έχει καμία άλλη διάφορα από την Update.

3.15.3 Γραφική διεπαφή (Graphical User Interface)

Για την υλοποίηση μιας γραφικής διεπαφής υπεύθυνη είναι η συνάρτηση OnGUI() η οποία μάλιστα είναι και η μοναδική συνάρτηση μέσα από την οποία μπορούν να κληθούν στοιχεία GUI. Μερικά από αυτά είναι κουμπιά (buttons), ετικέτες (labels), περιοχές φράσης (textfiels), περιοχές κειμένου (textareas) και δημιουργία παραθύρων. Η συνάρτηση OnGUI(), όπως και η συνάρτηση Update(), εκτελείται σε κάθε frame και όχι μια φορά. Η ιδιότητα αυτή μας επιτρέπει να συνδέσουμε μεταβλητές άλλων συναρτήσεων με την OnGUI() και να παρατηρούμε τις ενημερωμένες τιμές στην οθόνη.

3.15.4 Συναρτήσεις προγραμματιστή

Εκτός από τις συναρτήσεις που μας παρέχει η Unity, μπορούμε να γράψουμε και τις δικές μας, οι οποίες όμως για μπορέσουν να λειτουργήσουν θα πρέπει να καλούνται μέσα από κάποια συνάρτηση της μηχανής όπως η Update(). Μέσω του κώδικα είμαστε σε θέση να ελέγξουμε κάθε ιδιότητα του αντικειμένου στο οποίο έχουμε συνδέσει το script. Επίσης οι ιδιότητες των επιπλέον συστατικών (Components) που μπορεί να προσθέσουμε σε ένα αντικείμενο είναι και αυτές διαθέσιμες για επεξεργασία μέσω κώδικα.

Οι συναρτήσεις του προγραμματιστή είναι απαραίτητες και αποτελούν τον συνδετικό κρίκο μεταξύ όλων των οντοτήτων στο παιχνίδι. Χωρίς αυτές το αποτέλεσμα θα ήταν μία στατική απεικόνιση γεωμετρίας.

3.16 Γενικές ρυθμίσεις

Ο γενικές ρυθμίσεις επηρεάζουν ολόκληρο το παιχνίδι ανεξάρτητα από το επίπεδο στο οποίο βρισκόμαστε.

3.16.1 Quality

Η ρύθμιση οπτικής ποιότητας διαθέτει 6 επίπεδα από Fastest μέχρι Fantastic. Ο επιλογές αφορούν την ανάλυση των σκιών και των υφών(textures) καθώς και το φωτισμό. Με αυτόν τον τρόπο οι παλαιότερης τεχνολογίας υπολογιστές θα μπορούν να παίξουν το παιχνίδι με καλά γραφικά, ενώ οι καινούργιας τεχνολογίας θα αποδώσουν την μέγιστη ποιότητα γραφικών.

3.16.2 Render

Με την επιλογή Render μπορούμε να προσθέσουμε ομίχλη στην πίστα μας και να αλλάξουμε την πυκνότητα και το χρώμα της. Ο περιβαλλοντικός φωτισμός (ambient lighting) επηρεάζει το φωτισμό σε ολόκληρο το επίπεδο και είναι βοηθητικός. Αν δηλαδή θέλουμε να πετύχουμε καλό οπτικό αποτέλεσμα πρώτα θα χρησιμοποιήσουμε τα κύρια φώτα της Unity (point, spot, directional) και στη συνέχεια θα κάνουμε τις τελικές μικρορυθμίσεις με τον περιβαλλοντικό φωτισμό. Τέλος μπορούμε να δημιουργήσουμε ουρανό με δικές μας φωτογραφίες. Η οντότητα που χρησιμοποιεί η Unity λέγεται Skybox και ανήκει στην κατηγορία των σκιαστών. Πιο συγκεκριμένα, είναι σκιαστής RenderFX/Skybox και για να είναι πλήρης χρειάζεται έξι φωτογραφίες έτσι ώστε να δημιουργήσει την τελική πανοραμική απεικόνιση του ουρανού.

3.16.3 Time

Στη ρύθμιση Time μας παρέχεται μια σημαντική επιλογή. Μπορούμε να αλλάξουμε τον κύκλο επανάληψης του FixedUpdate που αναφέρθηκε παραπάνω και ανάλογα με τις δυνατότητες του υπολογιστή να τον αυξήσουμε προκειμένου να πετύχουμε ρεαλισμό ή να τον επιβραδύνουμε για καλύτερη απόδοση.

3.16.4 Input

Η επιλογή αυτή ρυθμίζει την είσοδο του πληκτρολογίου ή του μοχλού παιχνιδιών. Ουσιαστικά αντιστοιχούμε τα κουμπιά του πληκτρολογίου/μοχλού με τους άξονες και τα κουμπιά δράσης (fire buttons) της Unity έτσι ώστε η ανάπτυξη κώδικα να είναι ανεξάρτητη από τη συσκευή εισόδου. Για παράδειγμα, όταν γράφουμε ένα script δεν χρησιμοποιούμε μια συνάρτηση όπως η GetKey(Ctrl), αλλά GetKey(FireButton1). Με αυτόν τον τρόπο εάν βάλουμε την λειτουργία του πλήκτρου Ctrl στο πλήκτρο Alt δεν θα χρειαστεί να αλλάξει κάτι στον κώδικα. Η Unity βέβαια παρέχει συναρτήσεις για να πάρουμε την είσοδο από κάποιο συγκεκριμένο πλήκτρο εάν το επιθυμούμε.

3.16.5 Ολοκλήρωση παιχνιδιού

Έχοντας ολοκληρώσει την ανάπτυξη μπορούμε να εξάγουμε το παιχνίδι μας σε τέτοια μορφή ώστε να εκτελείται χωρίς την ύπαρξη της μηχανής. Οι διαθέσιμες εξαγώγιμες μορφές είναι:.exe για λειτουργικό Microsoft Windows, executable για Mac OSX PPC ή Intel, Wii, iPhone. Επίσης υπάρχει και η δυνατότητα εξαγωγής σε μορφή κατάλληλη για να εκτελεστεί το παιχνίδι σε κάποιον browser (Mozilla,Safari). Η δυνατότητα αυτή προϋποθέτει την εγκατάσταση μιας επιπλέον εφαρμογής στον browser.

Κεφάλαιο 4

Σχεδιασμός παιχνιδιού και γραφική διεπαφή χρήστη

4.1 Εισαγωγή

Στο παρών κεφάλαιο παρουσιάζουμε το βασικό σχεδιασμό του παιχνιδιού καθώς και τη γραφική διεπαφή του χρήστη. Αναλύονται βήμα προς βήμα όλα τα μενού του παιχνιδιού καθώς και η λειτουργία τους.

4.2 Σενάριο

Το σενάριο στα παιχνίδια βολών αυτοκινήτου παίζει δευτερεύοντα ρόλο. Συνήθως υπάρχει ένα τουρνουά και καθώς ο χρήστης κερδίζει πίστες αναβαθμίζεται, μέχρι να αντιμετωπίσει τον τελευταίο αντίπαλο («αρχηγό»). Στην παρούσα εργασία υλοποιήθηκε μόνο μία αρένα για μάχες οπότε δεν υπήρχε λόγος υλοποίησης κάποιου σεναρίου.

Ο στόχος είναι απλός: επιβίωση. Εάν ο χρήστης καταφέρει να μείνει στο παιχνίδι έχοντας εξολοθρεύσει όλους τους αντιπάλους τότε κερδίζει. Τα αυτοκίνητα μπορούν να εξοπλιστούν με πυραύλους και να κάνουν επιθέσεις στα αντίπαλα αυτοκίνητα. Όταν ο μετρητής υγείας μηδενιστεί τότε το όχημα φεύγει από το παιχνίδι. Εκτός από το κομμάτι της μάχης κάθε τριάντα δευτερόλεπτα εμφανίζεται στην πίστα μία σημαία. Η σημαία είναι μια αφορμή για να αναγκαστούν οι αντίπαλοι να έρθουν κοντά ο ένας με τον άλλον και έτσι αναγκάζονται να συμμετάσχουν όλοι. Δηλαδή εξαλείφουμε την περίπτωση κάποιος να κρυφτεί σε μια γωνία μέχρι να μείνει μόνο ένας αντίπαλος. Την στιγμή που κάποιος θα πάρει την σημαία υπολογίζονται οι αποστάσεις όλων των αυτοκινήτων από την σημαία. Το αυτοκίνητο που έχει την μεγαλύτερη ευκλείδεια απόσταση, χάνει το παιχνίδι. Στην περίπτωση που ο χρήστης πάρει την σημαία του δίνεται μια επιβράβευση στην βαθμολογία του. Στην συνέχεια αναφέρονται εκτενέστερα οι κανόνες και ο τρόπος βαθμολόγησης.

4.3 Γραφική διεπαφή (Graphical User Interface)

4.3.1 Αρχικό μενού

Η αρχική σελίδα της εφαρμογής, παρουσιάζεται στην παρακάτω εικόνα. Ο χρήστης έχει τη δυνατότητα να πληροφορηθεί για τα κουμπιά χειρισμού, να ρυθμίσει κάποιες παραμέτρους του παιχνιδιού και φυσικά να παίξει.

Εικόνα 4.1: Αρχικό μενού

Οι λειτουργίες των κουμπιών είναι οι εξής:

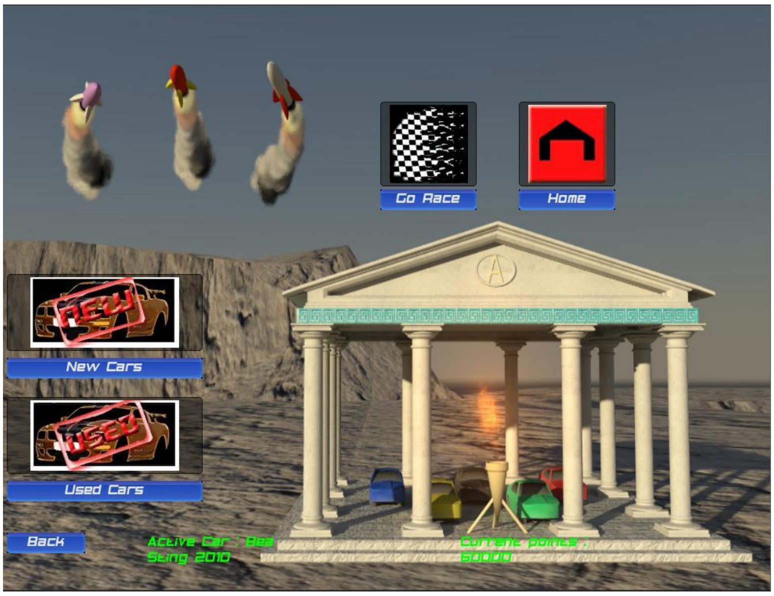
Start Game: Εκκίνηση του παιχνιδιού για ένα χρήστη.

Options: Οι ρυθμίσεις των παραμέτρων του παιχνιδιού. Λόγω της μεγάλης έκτασης του παιχνιδιού οι επιλογές περιορίστηκαν στην ρύθμιση της έντασης της μουσικής και των λοιπών ηχητικών εφέ.

How to play: Με αυτήν την επιλογή ο χρήστης μεταφέρεται σε μια οθόνη που του δίνει πληροφορίες για τον χειρισμό του αυτοκινήτου, αλλά και για τους κανόνες του παιχνιδιού.

Quit: Έξοδος από το παιχνίδι.

4.3.2 Κύριο μενού



Εικόνα 4.2: Κύριο μενού

Το κύριο μενού αποτελείται από τις εξής επιλογές:

New Cars: Μεταφορά στο μενού επιλογής καινούργιου αυτοκινήτου.

Used Cars: Μεταφορά στο μενού επιλογής μεταχειρισμένου αυτοκινήτου.

Home: Μεταφορά στο γκαράζ του χρήστη.

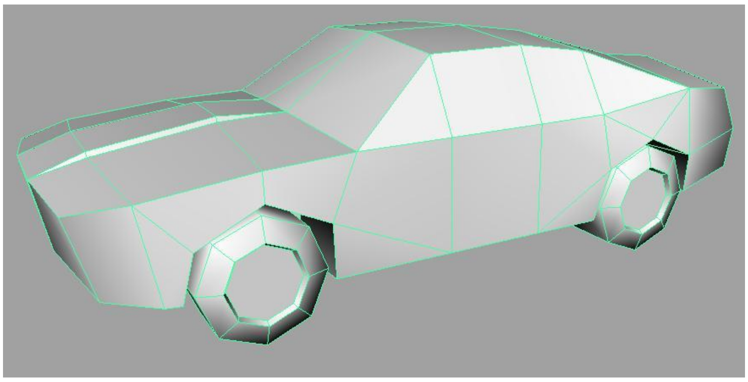
Go Race: Μεταφορά στο μενού επιλογής αγώνα

Επίσης στο κάτω μέρος της οθόνης βρίσκεται η ένδειξη Active Car. Το active car δείχνει ανά πάσα στιγμή το αυτοκίνητο που έχει επιλέξει ο χρήστης. Όταν ο χρήστης επιλέξει να τρέξει σε κάποιον αγώνα, τότε αυτόματα θα φορτωθεί το συγκεκριμένο αυτοκίνητο. Δεξιότερα βρίσκεται η ένδειξη Current Points που πληροφορεί για τους πόντους που έχει μαζέψει ο χρήστης. Οι πόντοι κερδίζονται παίζοντας σε αγώνες και μπορούν να χρησιμοποιηθούν για την αγορά καινούργιων αυτοκινήτων.

4.3.3 Ανάλυση των επιμέρους μενού

New and Used Cars

Τα δύο αυτά μενού περιέχουν όλα τα διαθέσιμα αυτοκίνητα του παιχνιδιού. Ο διαχωρισμός μεταξύ νέων και μεταχειρισμένων είναι καθαρά για τυπικούς λόγους και η διαφορά που έχουν είναι μόνο στην τιμή αγοράς.



Εικόνα 4.3: Η γεωμετρία του αυτοκινήτου

Για να υπάρξει πληθώρα επιλογών και ταυτόχρονα οικονομία στο χρόνο ανάπτυξης, η γεωμετρία είναι ακριβώς ίδια για όλα τα αυτοκίνητα. Αυτό που αλλάζει είναι η υφή. Κάθε όχημα έχει τη δική του υφή (texture) η οποία διαφοροποιείται είτε μέσω του χρώματος είτε μέσω του εμβλήματος που διαθέτει.

Επιλέγοντας ένα αυτοκίνητο από τη λίστα, ο χρήστης μεταφέρεται στο “Showroom” το οποίο αποτελεί και αυτό με τη σειρά του ένα νέο μενού. Στο “Showroom” μπορούμε να δούμε το επιλεγμένο όχημα και κάποιες πληροφορίες που το συνοδεύουν. Σε αυτό το στάδιο, υπάρχει η επιλογή αγοράς αυτοκινήτου επιστροφής στο προηγούμενο μενού. Σε περίπτωση αγοράς ενός αυτοκινήτου, εμφανίζεται ένα μήνυμα το οποίο μας ρωτά τον χρήστη εάν θέλει “να μπούμε στο αυτοκίνητο τώρα”. Εάν επιλέξει “να μπούμε στο αυτοκίνητο τώρα” τότε αυτομάτως ανανεώνεται το active car, αλλιώς παραμένει ως έχει. Τη στιγμή που αγοράζει ένα αυτοκίνητο, αυτό προστίθεται στην προσωπική συλλογή του χρηστή στο γκαράζ, απ’ όπου είναι διαθέσιμο οποιαδήποτε στιγμή.

Εικόνα 4.4: Showroom

Home

Εικόνα 4.5: Το μενού Home

Το μενού αυτό περιέχει τις εξής τρεις επιλογές:

Garage: Το γκαράζ του χρήστη. Εδώ αποθηκεύεται η προσωπική του συλλογή των αυτοκινήτων.

Hall of fame: Κατάλογος με όλες τις βαθμολογίες από τους αγώνες που έχουν γίνει.

Credits: Για να δημιουργηθεί το παιχνίδι χρειάστηκε να χρησιμοποιήσουμε αρκετό υλικό από τρίτους, είτε αυτό ήταν γεωμετρία είτε textures. Όλες οι πηγές, αναφέρονται σε αυτό το μενού.

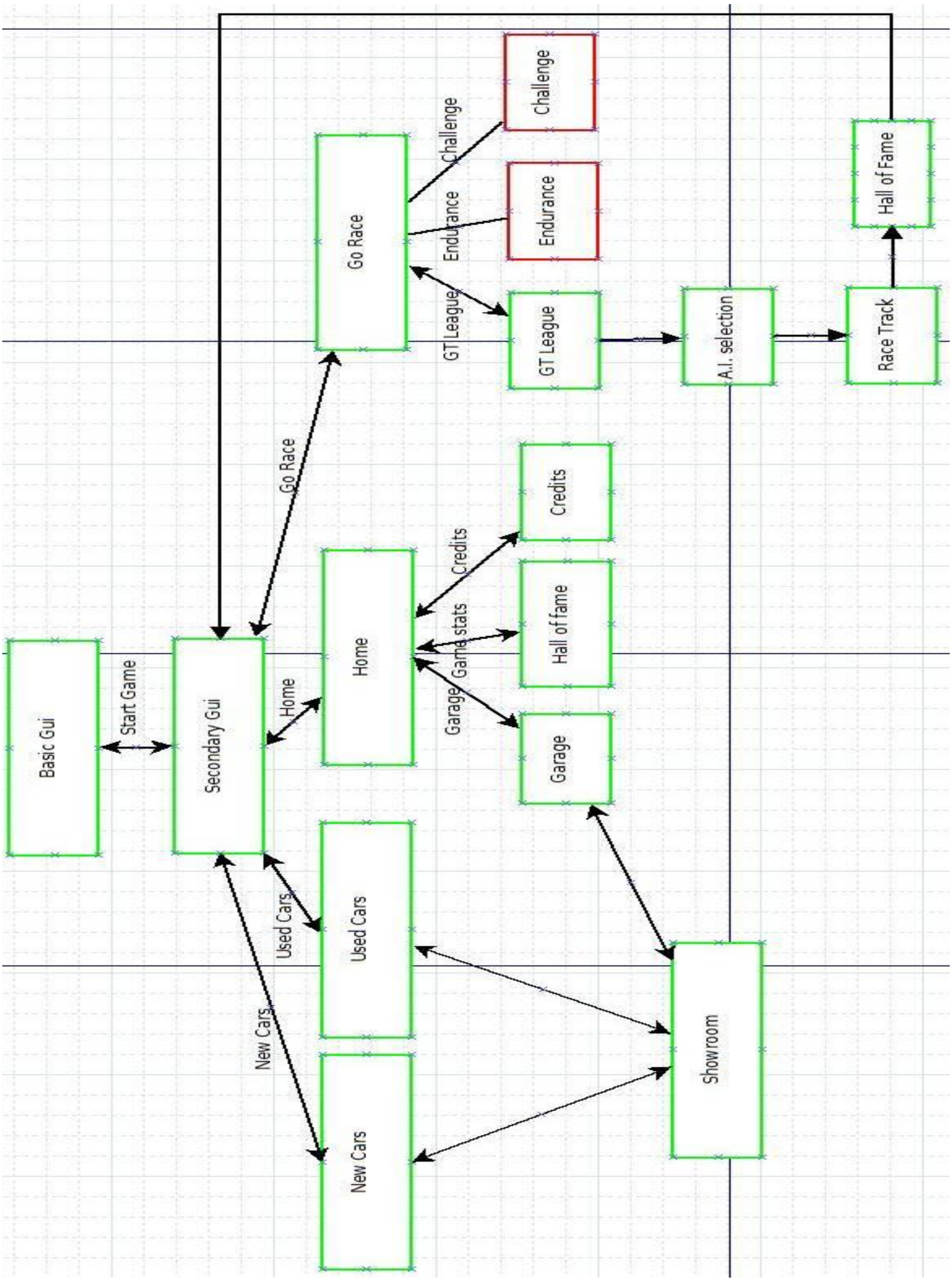
Go Race

Εικόνα 4.6: Το μενού Go Race

Σε αυτό το μενού διαλέγουμε έναν αγώνα για να τρέξουμε. Παρόλο που εμφανίζονται τρείς επιλογές μόνο η “GT League” είναι διαθέσιμη, καθώς οι υπόλοιπες αφορούν μελλοντική επέκταση του παιχνιδιού. Πατώντας το κουμπί “GT League” μεταφερόμαστε στην παρακάτω οθόνη όπου μπορούμε να διαλέξουμε τι τύπο τεχνητής νοημοσύνης θα χρησιμοποιήσουν οι αντίπαλοι μας.

Εικόνα 4.7: Επιλογή τεχνητής νοημοσύνης

Πατώντας το κουμπί “Race !!!” ξεκινάει ο αγώνας.



Εικόνα 4.8: Αρχιτεκτονική επιπέδων

*Σημείωση : τα κόκκινα κουτιά δηλώνουν μελλοντική επέκταση του παιχνιδιού

4.3.4 Κύριο γραφικό περιβάλλον παιχνιδιού

Εικόνα 4.9: Gameplay

Heads Up Display

Το κύριο γραφικό περιβάλλον αναφέρεται σε όσα βλέπει ο χρήστης στην διάρκεια ενός αγώνα, το λεγόμενο και Heads Up Display (HUD). Οι πληροφορίες που παρέχονται για τις επιδόσεις του αυτοκινήτου είναι οι στροφές του κινητήρα, η ταχύτητα στο κιβώτιο και η ταχύτητα του οχήματος σε km/h. ToRadar που υπάρχει στην πάνω δεξιά γωνία δείχνει τη σχετική θέση χρήστη-αντίπαλων. Η μπλε τελεία στο κέντρο του Radar είναι ο χρήστης ενώ η κόκκινη ο αντίπαλος. Κάτω αριστερά εμφανίζονται ενδείξεις για την υγεία και των αριθμό πυραύλων που διαθέτει ο χρήστης.

Μενού παύσης

Στην περίπτωση που ο χρήστης κατά τη διάρκεια του παιχνιδιού πατήσει το πλήκτρο Esc, εμφανίζεται ένα μενού με τρεις επιλογές:

Resume: συνέχιση του αγώνα

Restart: επανεκκίνηση του αγώνα

Quit: εγκατάλειψη και επιστροφή στον μενού επιλογής αγώνα.

4.4 Αναβαθμίσεις πυρομαχικών και υγείας

Η μέγιστη τιμή της υγείας είναι 100. Εάν η τιμή της υγείας κάποιου αυτοκινήτου φτάσει στο 0 (μηδέν), τότε αυτό καταστρέφεται και φεύγει από το παιχνίδι. Η υγεία μπορεί να μειωθεί με δύο τρόπους. Είτε με σύγκρουση του αυτοκινήτου με άλλη γεωμετρία (αυτοκίνητο κτήριο, κ.α.), είτε όταν βάλλεται από πυραύλους και το πολυβόλο κάποιου άλλου αυτοκινήτου.

Αναλυτικότερα για τα οπλικά συστήματα, υπάρχουν τρείς κατηγορίες πυραύλων και ένα επιπλέον πολυβόλο. Το πολυβόλο έχει άπειρο αριθμό πυρομαχικών αλλά προκαλεί την μικρότερη ζημιά (0.5 βαθμοί). Οι Speed είναι οι πιο γρήγοροι πύραυλοι, αλλά και οι πιο αδύναμοι (5 βαθμοί) και η τροχιά τους είναι η ευθεία γραμμή. Οι πύραυλοι Homing είναι μεσαίας ισχύος (10 βαθμοί), ενώ το χαρακτηριστικό τους είναι ότι ακολουθούν το στόχο τους μέχρι να τον χτυπήσουν ή να συγκρουστούν με κάποιο εμπόδιο. Είναι σχετικά δύσκολο να τους αποφύγει κάποιος, αλλά εφικτό. Τέλος οι Power είναι οι πύραυλοι με την μεγαλύτερη ισχύ (20 βαθμοί), είναι όμως και οι πιο αργοί.

Τα οπλικά συστήματα σε νούμερα

	Ταχύτητα	Δύναμη	Πόντοι ζημιάς
Machine gun	-	0.5	0.5
Homing missile	60	10	10
Speed missile	180	5	5
Power missile	130	20	20

Όπως αναφέραμε και στο πρώτο κεφάλαιο κάθε τριάντα δευτερόλεπτα εμφανίζεται στην πίστα μια σημαία την οποία όλοι θέλουν να πάρουν. Επειδή οι αντίπαλοι του χρήστη ελέγχονται από τον υπολογιστή γνωρίζουν την ακριβή θέση της σημαίας. Προκειμένου να βοηθηθεί ο χρήστης να βρει ποιο εύκολα το στόχο του, για όσο υπάρχει στην πίστα η σημαία, ακριβώς πάνω από το αυτοκίνητο εμφανίζεται ένα βέλος που δείχνει τη σωστή κατεύθυνση.

Στην πάνω αριστερή γωνία της οθόνης παρέχονται πληροφορίες για τον κοντινότερο αντίπαλο, όπως το όνομα, η ποσότητα υγείας και η τρέχουσα κατάσταση τεχνητής νοημοσύνης. Οι καταστάσεις τεχνητής νοημοσύνης είναι τέσσερις, κυνήγι ή αποφυγή κοντινότερου αντιπάλου και αναζήτηση υγείας ή πυρομαχικών.

Για να αναβαθμίσουμε την υγεία αλλά και τα πυρομαχικά μας μέσα στην πίστα υπάρχουν διάσπαρτα κάποιοι κύβοι (Packs). Συλλέγοντας έναν κύβο, ανάλογα με την κατηγορία του, η οποία καθορίζεται από το χρώμα του, παρέχει μία αναβάθμιση. Ο κύβος Health (μπλε) αυξάνει την υγεία του χρήστη κατά 25 μονάδες, αλλά σε καμία περίπτωση το σύνολο δεν ξεπερνάει τις 100. Οι κύβοι για τους πυραύλους παρέχουν 2 πυραύλους κάθε φορά, ανάλογα με την κατηγορία του κύβου. Το κόκκινο αντιστοιχεί στον Speed, το μωβ στον Homing και τέλος, λευκό για τον Power. Αφού πάρει κάποιος μία αναβάθμιση, αυτή κρύβεται έτσι ώστε να μην είναι διαθέσιμη για κανέναν άλλο. Μετά από ένα χρονικό διάστημα δέκα δευτερολέπτων επανεμφανίζεται και γίνεται διαθέσιμη στα αυτοκίνητα.

4.5 Νίκη / Ήττα

Όπως σε όλα τα παιχνίδια έτσι και εδώ μπορούμε είτε να κερδίσουμε είτε να χάσουμε. Ο χρήστης εγκαταλείπει το παιχνίδι σε δύο περιπτώσεις. Η πρώτη είναι να μηδενιστεί η ενέργεια του αυτοκινήτου. Η δεύτερη είναι το αυτοκίνητό του να βρίσκεται πιο μακριά από την σημαία σε σχέση με τα άλλα αυτοκίνητα όταν κάποιος καταφέρει να την πάρει. Προκειμένου να κερδίσει ο χρήστης, το αυτοκίνητό του θα πρέπει να είναι το τελευταίο που βρίσκεται στην πίστα. Στην περίπτωση αυτή, αφού τελειώσει η μάχη ζητείται από το χρήστη να εισάγει ένα όνομα και πατώντας “Enter” βλέπει τη βαθμολογία του σε σχέση με τα καλύτερα score μέχρι στιγμής. Επιπλέον δίνεται μια χρηματική αμοιβή των 10000 credits η οποία μπορεί να χρησιμοποιηθεί για την απόκτηση καινούργιου αυτοκινήτου.

Αναλυτικότερα το σύστημα βαθμολόγησης έχει ως εξής. Το ποσό της ζημιάς που προκαλείται από τον χρήστη σε έναν αντίπαλο μετατρέπεται σε πόντους υπέρ του χρήστη. Για παράδειγμα ο Power πύραυλος προκαλεί ζημιά 20 πόντων στην υγεία του αυτοκινήτου. Άρα ο χρήστης κερδίζει 20 πόντους κάθε φορά που χτυπά έναν αντίπαλο με Power πύραυλο. Βέβαια εάν ο αντίπαλος χτυπήσει με Power πύραυλο τον χρήστη, εκτός από μείωση στην υγεία, θα υπάρξει και μείωση στη βαθμολογία του. Τα χτυπήματα από τρακάρισμα και χρήση πολυβόλου δε λαμβάνονται υπόψη στην βαθμολογία καθώς προκαλούν πολύ μικρή ζημιά. Επίσης εάν ο χρήστης εξολοθρεύσει έναν αντίπαλο, δηλαδή του δώσει τη χαριστική βολή, τότε κερδίζει επιπλέον 100 πόντους, ενώ στην περίπτωση που πάρει την σημαία κερδίζει 200 πόντους. Τέλος οι αναβαθμίσεις ενέργειας (Health packs) έχουν αντίκτυπο στην βαθμολογία και μάλιστα αρνητικό καθώς την μειώνουν κατά 5 μονάδες.

Πίνακας 1 Περιπτώσεις αύξησης βαθμολογίας

	Αύξηση βαθμών
Χτύπημα στόχου με Power missile	+20
Χτύπημα στόχου με Speed missile	+5
Χτύπημα στόχου με Homing missile	+10
Υγεία τερματισμού	+Το ποσό της υγείας
Τελικό χτύπημα σε αντίπαλο	+100 (Για κάθε αντίπαλο)
Σημαία	+200 (Για κάθε σημαία)

Πίνακας 2 Περιπτώσεις μείωσης βαθμολογίας

	Μείωση βαθμών
Χτύπημα από Power missile	-20
Χτύπημα από Speed missile	-5
Χτύπημα από Homing missile	-10
Αναπλήρωση υγείας	-5

Εικόνα 4.10: Hall of fame

Κεφάλαιο 5

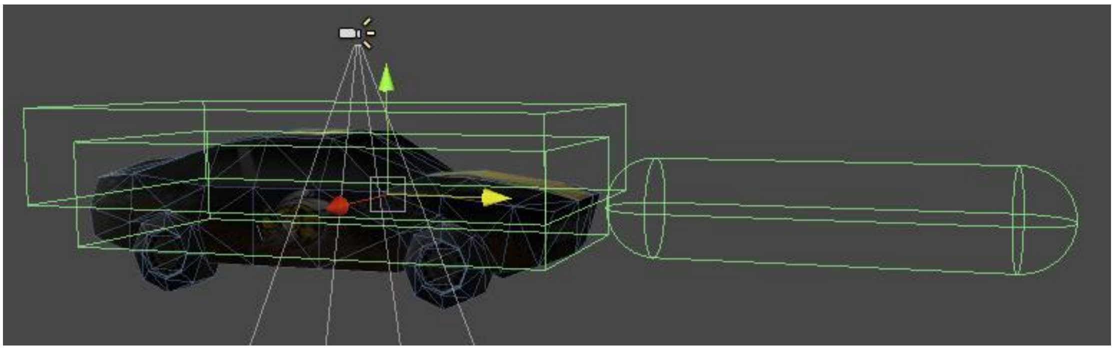
Υλοποίηση παιχνιδιού

5.1. Εισαγωγή

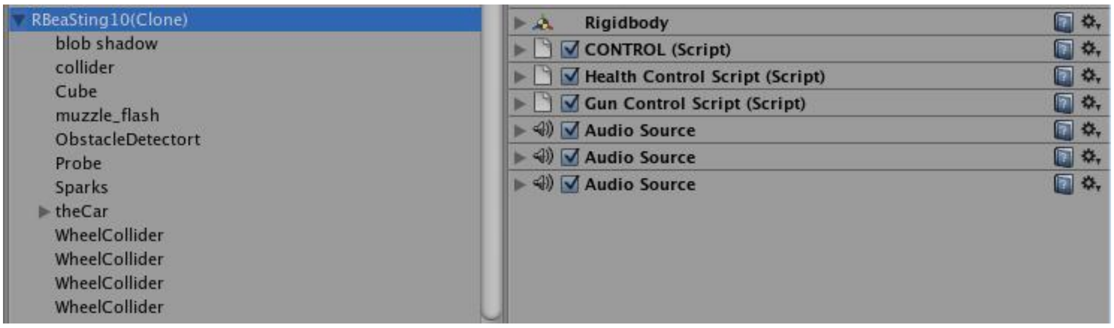
Στο Κεφάλαιο αυτό παρουσιάζεται η διαδικασία η οποία ακολουθήθηκε για να μπορέσει η γεωμετρία του αυτοκινήτου να αναγνωριστεί από την μηχανή φυσικής. Στην συνέχεια περιγράφεται ο τρόπος υπολογισμού της ποσότητας της ροπής στους τροχούς και της αλλαγής ταχυτήτων. Επιπλέον για τα αντίπαλα αυτοκίνητα που ελέγχονται από τον υπολογιστή έχει αναπτυχθεί ένα σύστημα αποφυγής εμποδίων, αλλά και ένα σύστημα αποφάσεων με FSM και Fuzzy Logic. Τέλος παρουσιάζεται η υλοποίηση του αλγόριθμου A* ο οποίος επιλύει το πρόβλημα για την εύρεση της βέλτιστης διαδρομής μεταξύ αυτοκινήτου και σημαίας.

5.2 Οργάνωση αυτοκινήτου (Car Setup)

Όπως αναφέραμε και στο πρώτο κεφάλαιο η γεωμετρία του αυτοκινήτου είναι από τρίτο δημιουργό. Για να εισαγάγουμε τη γεωμετρία μας στην Unity κάνουμε δεξί κλικ-> Import new Asset στην περιοχή του Project. Στην συνέχεια μπορούμε να την τοποθετήσουμε απευθείας σε μία πίστα ή να την καλούμε να εμφανιστεί δυναμικά χρησιμοποιώντας κώδικα. Μέχρι στιγμής υπάρχει ένα ακίνητο αυτοκίνητο το οποίο δεν αναγνωρίζει η μηχανή φυσικής και έτσι μπορούμε να περάσουμε από μέσα του. Για να δώσουμε λοιπόν τις επιθυμητές ιδιότητες θα πρέπει να προσθέσουμε κάποια Components. Να σημειωθεί ότι για να χρησιμοποιήσουμε το αυτοκίνητο δημιουργούμε ένα Prefab που περιέχει όλα τα απαραίτητα χαρακτηριστικά. Με αυτόν τον τρόπο, μέσω κώδικα καλούμε το Prefab και έτσι δεν χρειάζεται να κάνουμε κάθε φορά τις ίδιες ρυθμίσεις.



Εικόνα5.1: Το αυτοκίνητο και οι colliders



Εικόνα 5.2: Ιδιότητες

Στην κορυφή της ιεραρχίας έχουμε ένα GameObject με ξεχωριστό όνομα για κάθε όχημα. Το συστατικό Rigidbody, δίνει την εντολή στην μηχανή φυσικής να εφαρμόσει πάνω στο αντικείμενο την ιδιότητα της βαρύτητας. Για τον έλεγχο του αυτοκινήτου ο κώδικας χωρίζεται σε τρία Scripts : CONTROL.js, Health Control Script.js, Gun Control Script.js. Το πρώτο είναι υπεύθυνο για τον χειρισμό του οχήματος, την αποφυγή εμποδίων, και την λήψη αποφάσεων (FSM, Fuzzy Logic). Τα επόμενα δύο διαχειρίζονται την υγεία και τα όπλα αντίστοιχα. Στις επόμενες παραγράφους θα αναλύσουμε τα υπόλοιπα GameObjects που συνοδεύουν την προκατασκευή (Prefab) του αυτοκινήτου.

Στα παιχνίδια το ζητούμενο είναι η ομαλή λειτουργία και η καλή απόδοση. Κανονικά καθώς το φώς που υπάρχει στην σκηνή ρίχνει τις ακτίνες του στο αυτοκίνητο θα έπρεπε να υπολογίζεται συνέχεια η σκιά του. Εντούτοις για να μην επιβαρύνουμε το σύστημα προβάλουμε μία έτοιμη σκιά κάτω από το αυτοκίνητο μέσω ενός βοηθήματος που μας παρέχει η Unity ονόματι Projector. Η τεχνική αυτή υπάρχει ακόμα και στα σημερινά παιχνίδια γιατί πετυχαίνουμε ένα αποδεκτό οπτικό αποτέλεσμα και πολύ καλή απόδοση. Το GameObject που αναλαμβάνει την παραπάνω λειτουργία είναι το blob_shadow και σε αυτό έχει τοποθετηθεί το συστατικό Projector.

Προκειμένου η μηχανή φυσικής να καταλαβαίνει που βρίσκονται τα όρια του αυτοκινήτου χρησιμοποιούμε ένα συστατικό ονόματι Box Collider. Τοποθετήσαμε τον συγκρουστή (Collider) κατάλληλα, ώστε να περιβάλει όλο το αμάξωμα εκτός από τους τροχούς. Ουσιαστικά λοιπόν η μηχανή φυσικής αντιλαμβάνεται τα οχήματα μας, ως κουτιά.

Σε ένα παιχνίδι βολών αυτοκινήτου οι ανατροπές είναι πολύ συχνές. Η οντότητα Cube είναι ένας Trigger. Όταν μέσα στον Trigger παρεμβληθεί το Terrain, δηλαδή το έδαφος τότε πιθανότατα το αυτοκίνητο να ανατράπηκε και πρέπει να το επαναφέρουμε. Χρησιμοποιούμε την λέξη «πιθανότατα» διότι υπάρχει περίπτωση το αυτοκίνητο να επανέλθει χωρίς την δική μας παρέμβαση, χτυπώντας με άλλη γεωμετρία. Για να μπορέσει να γίνει αυτό έχουμε προσθέσει στην οντότητα το παρακάτω Script.

Roof180.js

```
function OnTriggerEnter (other: Collider)
{
	if(other.tag == "terrainTag")
	{
		Invoke("RestoreCar",4);
	}
}
function OnTriggerExit (other: Collider)
{
	CancelInvoke("RestoreCar");
}
function RestoreCar()
{
	SendMessageUpwards("Restore",1);
}
```

Με την SendMessageUpwards στέλνουμε μήνυμα στην CONTROL.js για να ξεκινήσει η Restore() η οποία θα επαναφέρει το αυτοκίνητο.

```
function Restore()
{
	transform.rotation.z = 0;
	transform.rotation.x = 0;
	transform.rotation.y = 0;
	rigidbody.velocity = Vector3.zero;
	transform.position.y = 15;
	rigidbody.freezeRotation = true;
	yield WaitForSeconds(0.5);
	rigidbody.freezeRotation = false;
}
```

Το επόμενο GameObject είναι το muzzle flash. Η ονομασία αυτή προέρχεται από την λάμψη που βλέπουμε μπροστά από ένα όπλο όταν πυροβολεί. Στην προκειμένη περίπτωση θέλαμε να φτιάξουμε το εφέ της αγωνιστικής εξάτμισης το οποίο δημιουργείται με αντίστοιχο τρόπο στα παιχνίδια.. Ουσιαστικά αυτό που δημιουργήσαμε ήταν μια γρήγορη εναλλαγή μιας φωτογραφίας (Εικόνα 5.3) κάθε φορά που αλλάζουμε ταχύτητα.

Εικόνα 5.3: Εικόνα για την λάμψη στην εξάτμιση

Στην συνέχεια για την αποφυγή εμποδίων τοποθετήσαμε το GameObject Obstacle Detector. Το συστατικό (Component) που περιέχει είναι ένας Capsule collider ο οποίος λειτουργεί ως Trigger. Με την βοήθεια του είμαστε σε θέση να ανιχνεύσουμε πιθανές συγκρούσεις. Κάθε φορά που παρεμβάλλεται γεωμετρία μέσα στην περιοχή του Trigger λαμβάνουμε ένα σήμα το οποίο εκμεταλλευόμαστε μέσω Script με την συνάρτηση OnTriggerEnter(Collider : other). Θεωρητικά το μήκος του θα έπρεπε να μεταβάλλεται ανάλογα με την ταχύτητα. Δηλαδή όσο μεγαλύτερη ταχύτητα έχουμε τόσο πιο μακριά πρέπει να ελέγχουμε για εμπόδια. Όμως οι γρήγορες εναλλαγές στην ταχύτητα των αυτοκινήτων στο παιχνίδι, μετέβαλλαν πολύ συχνά το μήκος του ανιχνευτή. Αυτό είχε ως αποτέλεσμα οι αντίπαλοι να μπερδεύονται νομίζοντας ότι το εμπόδιο χάνεται και εμφανίζεται σε κλάσματα δευτερολέπτου. Πρακτικά λοιπόν, ρυθμίσαμε τον ανιχνευτή να έχει σταθερό μήκος. Το πώς ακριβώς γίνεται η ανίχνευση και αποφυγή συγκρούσεων θα αναλυθεί σε παρακάτω ενότητα.

Όπως έχουμε αναφέρει τα οχήματα διαθέτουν πληθώρα όπλων, ανάμεσα σε αυτά και το πολυβόλο. Καθώς τα αυτοκίνητα ανταλλάσουν πυρά θα πρέπει να φαίνεται το σημείο σύγκρουσης μεταξύ σφαιρών και γεωμετρίας. Το GameObject Sparks έχει για συστατικό (Component) ένα σύστημα σωματιδίων (Particle system) για την εμφάνιση του ζητούμενου εφέ. Όταν μια σφαίρα από το πολυβόλο χτυπήσει σε γεωμετρία τότε ενεργοποιείται το σύστημα σωματιδίων και εμφανίζεται υο εφέ. Σε αυτό το σημείο να τονίσουμε ότι για να υπάρξει σύγκρουση θα πρέπει η γεωμετρία να διαθέτει κάποιου είδους συγκρουστή (Collider).

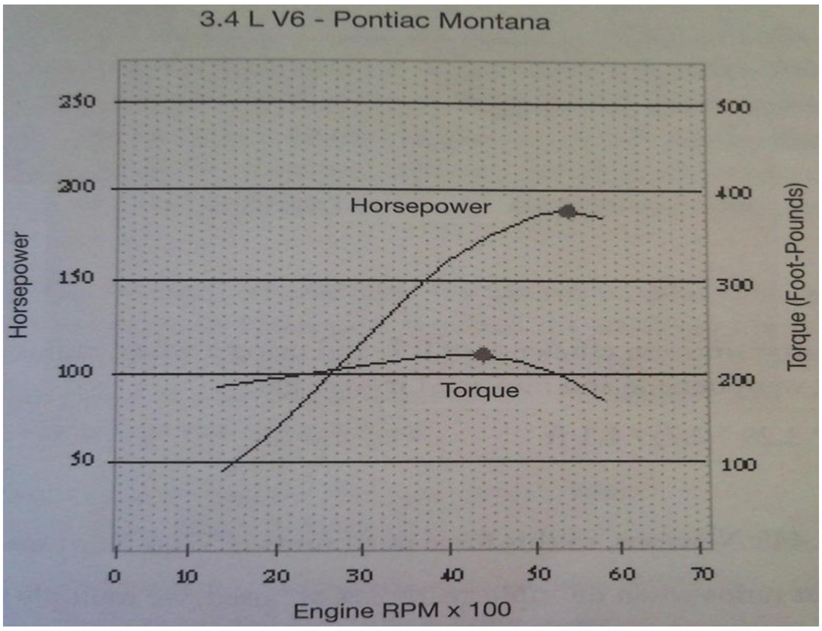
Η οριοθέτηση του οχήματος μέχρι στιγμής περιλάμβανε όλα τα μέρη του, εκτός από τους τροχούς. Οι τροχοί χρειάζονται ειδική διαχείριση καθώς είναι περιστρεφόμενα κομμάτια και μάλιστα η περιστροφή τους εξαρτάται από την ταχύτητα. Η Unity μας παρέχει ετοίμους συγκρουστές κατάλληλους για τροχούς οχημάτων εδάφους “Wheel Colliders”. Αναλυτικότερα για τους wheel colliders μπορείτε να ανατρέξετε στο κεφάλαιο 3. Για να μπορέσουμε να πετύχουμε την μέγιστη δυνατή ευθυγράμμιση με την κανονική γεωμετρία των τροχών, οι συγκρουστές δημιουργούνται λίγο πριν ξεκινήσει ο αγώνας μέσω του script CONTROL.js. Εναλλακτική λύση αποτελεί η χρήση σφαιρικών συγκρουστών (Sphere Colliders). Όμως η προσομοίωση των αναρτήσεων και ο υπολογισμός της ροπής των τροχών θα έπρεπε να δημιουργηθεί από την αρχή κάτι που θα καθυστερούσε κατά πολύ την ανάπτυξη του υπόλοιπου παιχνιδιού.

5.3 Φυσική αυτοκινήτου (Car Physics)

Ένα σύγχρονο παιχνίδι επιβάλλεται να χρησιμοποιεί ένα ρεαλιστικό μοντέλο φυσικής. Στην δική μας περίπτωση προσπαθήσαμε να εξομοιώσουμε την συμπεριφορά ενός αυτοκινήτου με βάση τους νόμους της φυσικής.

5.3.1 Μεταφορά δύναμης στους τροχούς: Μηχανή και κιβώτιο ταχυτήτων

Για να μπορέσουμε να υπολογίσουμε την ταχύτητα με την οποία θα τρέχει το αυτοκίνητο μας θα πρέπει πρώτα να μελετήσουμε την λειτουργία των πραγματικών αυτοκινήτων. Σε μια μηχανή εσωτερικής καύσης, η ροπή που παράγεται δεν είναι σταθερή, αλλά είναι μια συνάρτηση της ταχύτητας της μηχανής σε περιστροφές/λεπτό (Revolutions Per Minute). Να σημειώσουμε σε αυτό το σημείο ότι η συνάρτηση είναι μη-γραμμική και για να την υπολογίσουμε με ακρίβεια πρέπει να τοποθετήσουμε το αυτοκίνητο σε δυναμόμετρο. Στην συνέχεια ακολουθεί η σύνδεση με το κιβώτιο ταχυτήτων, το οποίο καθορίζει το ποσοστό ροπής που θα μεταφερθεί στους άξονες, το διαφορικό και τέλος οι τροχοί. Σε ένα πραγματικό αυτοκίνητο λοιπόν η τιμή της ροπής καθορίζεται από τον κινητήρα. Σε ένα παιχνίδι όμως δεν διαθέτουμε μηχανή εσωτερικής καύσης και για αυτό θα παράγουμε τις απαραίτητες τιμές με διαφορετικό τρόπο.



Εικόνα 5.4: Καμπύλη ροπής

Η καμπύλη (Εικόνα 5.4) θα μπορούσε να αποθηκευτεί σε ένα πίνακα και να χρησιμοποιηθεί σε ένα παιχνίδι. Όμως επειδή στην δική μας υλοποίηση δεν θέλουμε να φτιάξουμε εξομοιωτή, δεν χρειαζόμαστε μεθόδους με τόσο μεγάλη ακρίβεια. Μια απλή προσεγγιστική συνάρτηση είναι ικανή να παράγει τα επιθυμητά αποτελέσματα.

Στην παραπάνω εικόνα φαίνεται επίσης ότι και η ιπποδύναμη είναι μεταβλητό μέγεθος το οποίο εξαρτάται από τις RPM. Η σχέση μεταξύ ροπής και η ιπποδύναμης είναι:

$$H = \frac{\tau RPM}{5252}$$

Μετά από τον κινητήρα η ροπή μεταφέρεται στο κιβώτιο ταχυτήτων. Το κιβώτιο διαθέτει κάποιες αναλογίες για τις ταχύτητες του.

Πίνακας 3 Αναλογίες κιβωτίου ταχυτήτων

	Montana Gear Ratios	Game Gear Ratios
1st	2.92	2.3
2nd	1.57	1.78
3rd	1.00	1.30
4th	0.71	1.00
Reverse	2.39	2.3
Final drive	3.29:1	3.4:1

Η αναλογία 2.92 σημαίνει ότι για να πάρει μία στροφή το γρανάζι στο κιβώτιο, θα πρέπει ο κινητήρας να πάρει 2.92 στροφές. Στην εφαρμογή μας όμως πρέπει να μάθουμε την αναλογία κινητήρα – τροχών. Για αυτόν το λόγο πολλαπλασιάζουμε την αναλογία της κάθε ταχύτητας με την αναλογία του διαφορικού (final drive). Για παράδειγμα εάν έχουμε 3000RPM τον κινητήρα, θα έχουμε 3000/(1.57*3.29) = 580.3 RPM στους τροχούς [17].

Η χρησιμότητα της παραπάνω θεωρίας θα γίνει άμεσα κατανοητή με την παρουσίαση του κώδικα για την κίνηση του αυτοκινήτου.

Εάν οι υπολογιστές διέθεταν μία μηχανή εσωτερικής καύσης, θα υπολογίζαμε άμεσα τη ροπή στους τροχούς και το αυτοκίνητό μας θα μπορούσε να κινηθεί ανάλογα με το πάτημα του κουμπιού του χρήστη. Επειδή κάτι τέτοιο όμως δε συμβαίνει, αντί να ξεκινήσουμε από την μηχανή που δυστυχώς δεν υπάρχει, θα ξεκινήσουμε από τους τροχούς. Πιο συγκεκριμένα από τους Wheel Colliders οι οποίοι μας δίνουν και τα R.P.M. αλλά και τη ροπή που τους ασκείται.

Κάθε τροχός έχει διαφορετικές στροφές και έτσι προτιμούμε να παίρνουμε το μέσο όρο των RPM όλων των τροχών με την συνάρτηση ComputeRpmFromWheels(). Στην περίπτωση που το αυτοκίνητο δεν κινείται, οι στροφές των τροχών είναι μηδενικές. Για να αποφύγουμε αυτό το πρόβλημα έχουμε προσθέσει ένα ελάχιστο εικονικό όριο στροφών έτσι ώστε πότε να μην πάρουμε μηδέν R.P.M. ακόμα και όταν το αυτοκίνητο βρίσκεται σε ακινησία.

```
// Compute average RPM of all motorized wheels
function ComputeRpmFromWheels(): float
{
	var rpm = 0.0;
	var count = 0;
	for( var w in wheels )
	{
		if( w.powered )
		{
			rpm += w.coll.rpm;
			++count;
		}
	}

	if( count != 0 )
	{
		rpm /= count;
	}
return rpm;
}
```

Η μεταβλητή motor έχει εύρος 0 … 1 και αντιστοιχεί στο πάτημα πλήκτρου που ελέγχει το γκάζι. Η συνάρτηση ComputeAxisTorque() θα πάρει σαν όρισμα την ποσότητα του γκαζιού (motor). Στη συνέχεια υπολογίζουμε τις στροφές του κινητήρα με τη βοήθεια των αναλογιών που αναφέρθηκαν παραπάνω, από τις στροφές των τροχών. Έχοντας τις στροφές του κινητήρα χρησιμοποιούμε την καμπύλη ροπής για να βρούμε τη ροπή του κινητήρα. Για να πάρει την τελική τιμή η ροπή του κινητήρα την πολλαπλασιάζουμε με την ποσότητα του γκαζιού (motor), το οποίο αποτελεί μία μικροβελτίωση που προστέθηκε μετά από δοκιμές.

Επίσης υπάρχει ένα ελάχιστο και ένα μέγιστο όριο στροφών. Εάν ξεπεράσουμε το ανώτατο όριο στροφών κινητήρα τότε ανεβάζουμε ταχύτητα. Το αντίστροφο συμβαίνει για το ελάχιστο όριο. Το μόνο που απομένει πλέον είναι να υπολογίσουμε την ροπή στους τροχούς.

Ροπή στου τροχούς = ((ροπή μηχανής)/ταχύτητα στο κιβώτιο/αναλογία διαφορικού)

```
wheelsRpm = ComputeRpmFromWheels();
var axisTorque = ComputeAxisTorque( Mathf.Abs( motor ) );
ComputeMotorRpm( wheelRPM );
AutomaticTransmission();
// A simple parabola to approximate the shape of motor torque
// curve. In real cars you probably would do this with explicit
// table that maps rpm to torque.
function MotorTorqueCurve( motorrpm: float )
{
	var x = (motorrpm/motorRpmRange) * 2 - 1;
	var y = 0.5*( - x * x + 2 ) + 5;
	return Mathf.Lerp( minTorque, maxTorque, y );
```

```
}
// Given acceleration pedal input, computes the torque that
// we want to apply to wheel axles.
function ComputeAxisTorque( accelPedal: float ): float
{
	// Compute current motor rpm, based on wheels rpm
	wheelRPM = Mathf.Abs( ComputeRpmFromWheels() );
	ComputeMotorRpm( wheelRPM );
	AutomaticTransmission();
	// Figure out motor torque based on motor rpm
	var motorTorque = accelPedal * MotorTorqueCurve( engineRPM );
	// Compute wheel axle torque from motor torque
	return motorTorque / gearRatios[currentGear] / finalDriveRatio;
}
// Compute motor RPM from wheel RPM
function ComputeMotorRpm( rpm: float )
{
	if(currentGear !=0)
	{
		engineRPM = rpm * gearRatios[currentGear] * finalDriveRatio;
	}
	else
	{
		engineRPM = -rpm * gearRatios[currentGear] * finalDriveRatio;
	}
}
```

Αφού λοιπόν υπολογίσαμε τη ροπή που έχουν οι τροχοί, εφαρμόζουμε την τιμή της στους Wheel Colliders και το αυτοκίνητο κινείται. Βέβαια η τελική τιμή είναι λίγο διαφορετική λόγω των μικροβελτιώσεων για να δοθεί καλύτερα η αίσθηση της ταχύτητας.

```
wheels[0].coll.motorTorque = 2 * axisTorque * motor;
wheels.coll.motorTorque = 2 * axisTorque * motor;
wheels.coll.motorTorque = 2 * axisTorque * motor;
wheels.coll.motorTorque = 2 * axisTorque * motor;
```

5.3.2 Κάθετη δύναμη (Downforce)

Καθώς το αυτοκίνητο αναπτύσσει μεγάλες ταχύτητες, σε περίπτωση απότομης στροφής υπάρχει μεγάλη πιθανότητα για ανατροπή. Η μηχανής φυσικής μας παρέχει πολλές ευκολίες όμως δε μπορεί να προσθέσει επιπλέον δυνάμεις σε ένα αντικείμενο. Θα πρέπει εμείς, χειροκίνητα να υπολογίσουμε και να προσθέσουμε την κάθετη δύναμη που θα ασκηθεί στο αυτοκίνητο έτσι ώστε να μην ανατραπεί. Η δύναμη εφαρμόζεται πάνω στο Component, Rigidbody που όπως έχουμε πει, είναι ο συνδετικός κρίκος μεταξύ της γεωμετρίας και της μηχανής φυσικής.

```
function ApplyDownPressure()
{
```

```
var groundedCount = 0;
var sidewaysSlip = 0.0;
var forwardSlip = 0.0;
for( var w in wheels )
{
	if( w.coll.isGrounded )
	{
		++groundedCount;
		var hit: WheelHit;
		var wc: WheelCollider = w.coll;
		wc.GetGroundHit(hit);
		sidewaysSlip += hit.sidewaysSlip;
		forwardSlip += hit.forwardSlip;
		//temp = w.coll.radius;
	}
}
if( groundedCount )
{
	sidewaysSlip /= groundedCount;
	forwardSlip /= groundedCount;
}
var smoother = Time.deltaTime * 5.0;
smoothSidewaysSlip = Mathf.Lerp( smoothSidewaysSlip, sidewaysSlip,
smoother );
smoothForwardSlip = Mathf.Lerp( smoothForwardSlip, forwardSlip,
smoother );
if( groundedCount )
{
	var downPressure = Vector3(0,0,0);
	downPressure.y=-Mathf.Pow(rigidbody.velocity.magnitude,1.2)*
	downPressureFactor;
	downPressure.y=Mathf.Max(downPressure.y,maxDownPressureForce
	);
var localPos = Vector3(0,0,0);
localPos.x = smoothSidewaysSlip / 5.0;
localPos.x = -Mathf.Sign(localPos.x) * localPos.x * localPos.x;
var hwidth = bcollider.size.x / 2.0;
localPos.x = Mathf.Clamp( localPos.x, -hwidth, hwidth );
var worldPos = rigidbody.position + rigidbody.rotation * localPos;
downPressure = transform.TransformDirection( downPressure );

rigidbody.AddForceAtPosition(downPressure,worldPos,
ForceMode.Acceleration );

rigidbody.AddForceAtPosition(downPressure,
Vector3(worldPos.x+10,worldPos.y,worldPos.z+10),
ForceMode.Acceleration );

rigidbody.AddForceAtPosition( downPressure, Vector3(worldPos.x-
10,worldPos.y,worldPos.z+10), ForceMode.Acceleration );
```

```
rigidbody.AddForceAtPosition(downPressure,
Vector3(worldPos.x+10,worldPos.y,worldPos.z-10),
ForceMode.Acceleration);

rigidbody.AddForceAtPosition( downPressure, Vector3(worldPos.x-
10,worldPos.y,worldPos.z-10), ForceMode.Acceleration );

	}
}
```

5.4 Αποφυγή εμποδίων (Steering behaviors – Obstacle avoidance)

Steering behavior, ονομάζουμε τη συμπεριφορά στρέψης των τροχών ενός αυτοκινήτου σε διάφορες συνθήκες. Μία από αυτές είναι η αποφυγή εμποδίων. Κατά τη διάρκεια του παιχνιδιού θα πρέπει τα αντίπαλα αυτοκίνητα να είναι σε θέση να καταδιώξουν οποιοδήποτε άλλο αυτοκίνητο. Τα αυτοκίνητα μέσα στην πίστα είναι επτά και η κίνηση τους γρήγορη και συνεχής, επομένως πρέπει να επιλέξουμε αποδοτικές μεθόδους έτσι ώστε η απόδοση του συστήματος να είναι ομαλή.

Για να το επιτύχουμε αυτό χρησιμοποιήσαμε έναν τροποποιημένο Line Of Sight (L.O.S.) αλγόριθμο. Ουσιαστικά υπολογίζουμε το πόσο πρέπει να στρίψουν οι ρόδες του αυτοκινήτου έτσι ώστε να αντικρίζει το στοχευόμενο αντικείμενο σε ευθεία γραμμή.

```
inputSteer = RelativeWaypointPosition.x / RelativeWaypointPosition.magnitude;
```

Στην συνέχεια αποφασίζουμε για το εάν πρέπει να δώσουμε ώθηση στο αυτοκίνητο ή να το φρενάρουμε. Οι περιπτώσεις που ελέγχουμε είναι τρείς.

Εάν η μεταβλητή inputSteer είναι μικρότερη από 0.1 τότε σημαίνει ότι το αυτοκίνητο πηγαίνει σχεδόν ευθεία και μπορούμε να πατήσουμε γκάζι.

```
if ( Mathf.Abs( inputSteer ) < 0.1 ) )
{
		inputHandbrake = 0;
		inputMotor = 1;
		inputBrake = 0 ;
}
```

Επίσης εάν η ταχύτητα του αυτοκινήτου είναι χαμηλή τότε όσο και αν στρίβουμε μπορούμε και πάλι να του δώσουμε ώθηση.

```
else if(rigidbody.velocity.magnitude * 3600/1000 <30 )
{
```

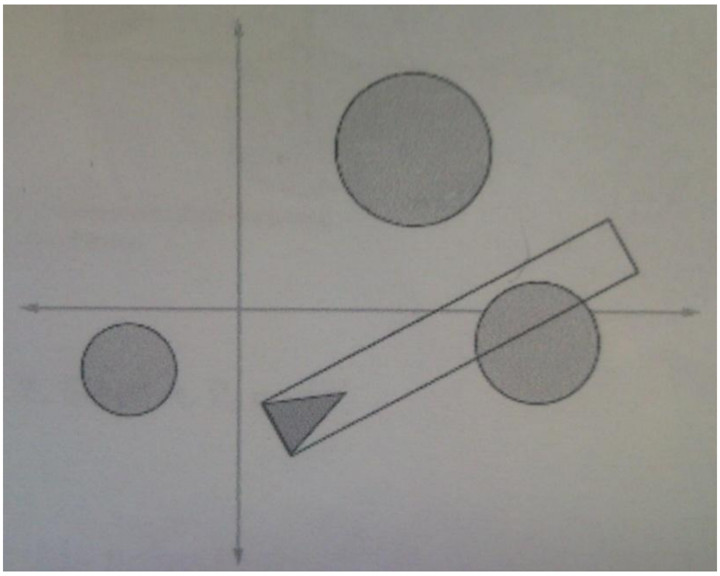
```
        inputHandbrake = 0;
        inputBrake = 0;
        inputMotor = 1;
}
```

Στην περίπτωση που καμία συνθήκη από τις παραπάνω δεν ικανοποιείται τότε το αυτοκίνητο φρενάρει.

```
else{
    inputHandbrake = 1;
    inputBrake = 1;
    inputMotor = 0;
}
```

Η στρατηγική αυτή έχει ένα μεγάλο μειονέκτημα, γιατί ενώ είναι από τους πιο γρήγορους αλγόριθμους, δεν μπορεί αυτομάτως να αποφεύγει εμπόδια. Πρέπει να προσθέσουμε επιπλέον κώδικα ο οποίος θα είναι υπεύθυνος για την αποφυγή εμποδίων και θα αλλάζει την πορεία του οχήματος όποτε είναι απαραίτητο.

Για να το πετύχουμε αυτό, εκτός από κώδικα χρειαζόμαστε και κάποια βοηθήματα που μας παρέχει η Unity3D. Όπως αναφέραμε στο προηγούμενο κεφάλαιο η οντότητα του αυτοκινήτου έχει ενσωματωμένη μία άλλη οντότητα με όνομα Obstacle Detector. Αυτή η οντότητα είναι ένας Capsule Collider με την επιλογή trigger.Δηλαδή δεν συγκρούεται με άλλη γεωμετρία αλλά μας ενημερώνει εάν συμβαίνει κάτι τέτοιο. Όπως δηλώνει και το όνομα του "Detector", χρησιμεύει σαν κουτί ανίχνευσης. Το πλάτος του είναι ίδιο με αυτό του αυτοκινήτου, ενώ το μήκος μεταβάλλεται ανάλογα με την ταχύτητα. Όσο πιο γρήγορα κινείται το αυτοκίνητο τόσο μακρύτερος γίνεται ο ανιχνευτής.

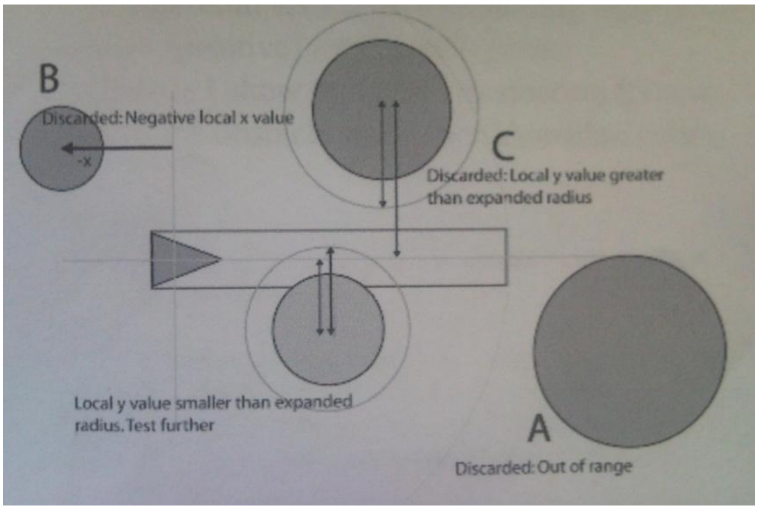


Εικόνα 5.5: Setup for the obstacle avoidance steering behavior [5]

Τα εμπόδια και τις πληροφορίες που αυτά περιέχουν, μπορούμε να τις πάρουμε χρησιμοποιώντας κώδικα και έτσι να έχουμε όλες τις πληροφορίες που χρειαζόμαστε για την αποφυγή τους. Το σχήμα τους είναι πάντα κυκλικό διότι χρειαζόμαστε μόνο το κέντρο και την ακτίνα τους ως πληροφορίες. Απλοποιούνται έτσι πολύ οι υπολογισμοί μας αλλά και εκτελούνται γρηγορότερα.

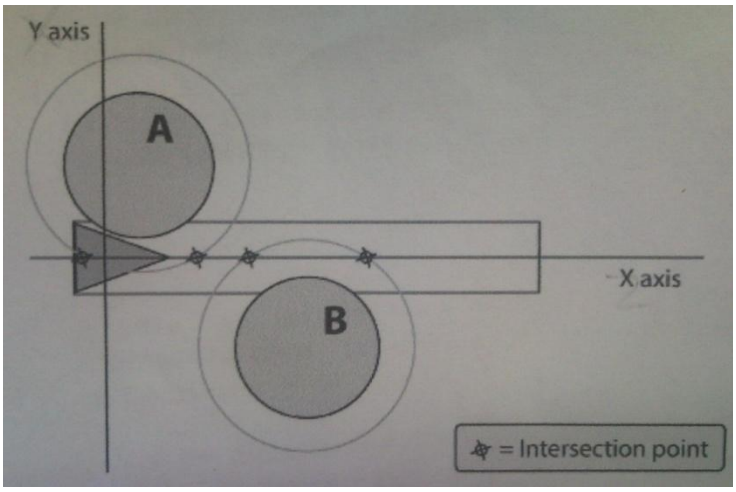
Αρχικά το όχημα θα πρέπει να λαμβάνει υπόψη του μόνο τα εμπόδια που βρίσκονται στην ευρύτερη περιοχή του ανιχνευτή. Ο αλγόριθμος μας θα ελέγξει όλα τα εμπόδια στην πίστα και μόνο όσα είναι μέσα στην περιοχή του θα συνεχίσουν στους επόμενους ελέγχους. Στην συνέχεια μετατρέπουμε τις παγκόσμιες συντεταγμένες των εμποδίων σε τοπικές συντεταγμένες σε σχέση με το αυτοκίνητο. Αυτό διευκολύνει κατά πολύ όλους τους υπολογισμούς μας, όπως για παράδειγμα, τα εμπόδια με αρνητική τοπική τιμή στο άξονα Ζ απορρίπτονται αμέσως. Σε όσα εμπόδια απομένουν κάνουμε μία επέκταση ακτίνας κατά το ήμισυ του πλάτους του ανιχνευτή. Συγκρίνουμε την τοπική τιμή X με την επεκταμένη ακτίνα των εμποδίων. Εάν η τοπική τιμή X είναι μεγαλύτερη από την επεκταμένη ακτίνα τότε σημαίνει ότι δεν πρόκειται να συγκρουστούμε και έτσι απορρίπτουμε το εμπόδιο [5].

Η παρακάτω εικόνα θα βοηθήσει στο να γίνουν ξεκάθαρα τα βήματα που έχουμε αναφέρει μέχρι τώρα.



Εικόνα 5.6: Steps for obstacle avoidance [5]

Σε αυτό το σημείο έχουμε μόνο τα εμπόδια που πρόκειται να παρεμβληθούν στην διαδρομή. Τώρα πρέπει να βρούμε ποιο είναι το κοντινότερο σημείο σύγκρουσης. Για άλλη μια φορά μας βοηθούν οι τοπικές συντεταγμένες. Προηγουμένως επεκτείναμε την ακτίνα των εμποδίων. Με βάση αυτό θα χρησιμοποιήσουμε μια απλή δοκιμή παρεμβολής ευθείας/κύκλου(line/circle intersection test). Θα υπάρχουν δύο σημεία παρεμβολής όπως φαίνεται στην παρακάτω εικόνα από τα οποία θα βρούμε ποιό είναι το κοντινότερο. Σημειώνεται ότι τα σημεία πίσω από το αυτοκίνητο απορρίπτονται [5].



Εικόνα 5.7: Intersection points [5]

Έχοντας πλέον το επιθυμητό σημείο, μπορούμε να αλλάξουμε την κατεύθυνση του αυτοκινήτου μας ώστε να αποφύγουμε το εμπόδιο. Το πώς θα μεταβληθεί η κατεύθυνση εξαρτάται από την θέση που έχει σε σχέση με το εμπόδιο.

```
if(activateStuck && Mathf.Floor(rigidbody.velocity.magnitude* 3600/1000 ) < 1 &&
activateStuck2 )
{
        activateStuck2 = false;
        Invoke("Unstuck",4);
}
if(probeCheck)
{
        //Closest Intersecting Obstacle
        var closestIntersectingObject ;
        //this will be used to track the distance to the CIB
        var distToClosestIP: float = Mathf.Infinity;//maxDouble;
        //this will record the transformed local coordinates of the CIB
        var localPosOfClosestObstacle;
        //m_dDBox.height=4+Mathf.Clamp(rigidbody.velocity.magnitude,0,4)*4;
        //m_dDBox.center. =  Mathf.Clamp(rigidbody.velocity.magnitude,0,8);
        m_dDBox.height = 25;
        m_dDBox.center.z = 12;
        var i: int = 0;
        currentObstacle = 0;

        for( i=0 ; i < obstacles.length ; i++ )
        {
            var                                  relativeObstaclePosition:Vector3=
            transform.InverseTransformPoint(                                  Vector3(
            obstacles[currentObstacle].position.x,                transform.position.y,
            obstacles[currentObstacle].position.z ) );
           if(relativeObstaclePosition.z >= 0  )
          {
```

```
//if the distance from the z axis to the object's position is
//less than it's radius
// + half the width of the detection box then there is a
//potential intersection
var expandedRadius: float = m_dDBox.radius +
obstacleColliders[currentObstacle].radius +20 ;
//Debug.Log("                                    realtive:
//"+Mathf.Abs(relativeObstaclePosition.x)+"   expandedRadius
//: "+expandedRadius);
if(Mathf.Abs(relativeObstaclePosition.x) < expandedRadius )
{
    //now to do a line/circle intersection test. The center
    //of the circle
    //is represented by (cZ,cX).The intersection points
    //are given by the formula
    // z = cZ +/-sqrt(r^2-cX^2) for x=0.
    //We only need to look at the smallest positive value of z
    //because that will be the closest point of intersection.

var cZ: float = relativeObstaclePosition.z;
var cX: float = relativeObstaclePosition.x;
//we only need to calculate the sqrt part of the above equation once
var sqrtPart: float = Mathf.Sqrt(expandedRadius*expandedRadius - cX*cX);
var ip: float = cZ - sqrtPart;
if(ip <= 0)
{
        ip = cZ + sqrtPart;
}
//test to see if this is the closest so far.If it is, keep a
//record of the obstacle and its local coordinates
if(ip < distToClosestIP)
{
distToClosestIP = ip;

closestIntersectingObject = obstacleColliders[currentObstacle];
localPosOfClosestObstacle = relativeObstaclePosition;

}

}
}
currentObstacle++;
}

//if we have found an intersecting obstacle, calculate a steering
//force away from it
var steeringForce: Vector3;
if(closestIntersectingObject)
{
```

```
        //the closer the agent is to an object, the stronger the steering force
        //should be
        var multiplier: float = 1.0 + (m_dDBox.height -
        localPosOfClosestObstacle.z)/m_dDBox.height;

        //calculate the lateral force
        steeringForce.x = (closestIntersectingObject.radius -
        localPosOfClosestObstacle.x)*multiplier*100;

        //apply a braking force proportional to the obstacle's distance from
        //the vehicle
        var brakingWeight: float = 100;
        var gmt: Vector3 = transform.position;
        gmt.z = gmt.z+5;
        steeringForce.z = (closestIntersectingObject.radius -
        localPosOfClosestObstacle.z )*brakingWeight ;

        inputHandbrake = 0;
        inputSteer = Mathf.Clamp(steeringForce.x, -1.0, 1.0);

        inputBrake = 0;
        inputMotor = 0.5;
        return;
        }
}
return ;
```

5.5 Λήψη αποφάσεων (Decision making)

Παρακάτω αναλύουμε πλήρως το σύστημα λογικής που αναπτύξαμε για τις ανάγκες του παιχνιδιού.

5.5.1 Μηχανές πεπερασμένων καταστάσεων (Finite State Machines)

Οι αντίπαλοι μας θέλουμε να είναι αυτόνομοι. Δηλαδή θα πρέπει να παίρνουν αποφάσεις για τις διάφορες καταστάσεις που συμβαίνουν στο παιχνίδι. Ο βασικότερος τρόπος για να γίνει αυτό είναι με την χρήση μίας μηχανής πεπερασμένων καταστάσεων (F.S.M.). Ως είσοδο στην μηχανή λαμβάνουμε τις μεταβλητές που αφορούν την υγεία, τον αριθμό των πυρομαχικών και το εάν δεχόμαστε πυρά ή όχι. Σε όποια κατάσταση και αν βρισκόμαστε ελέγχουμε όλες τις μεταβλητές και αποφασίζουμε την επόμενη μετάβαση της μηχανής. Οι διαθέσιμες καταστάσεις είναι οι εξής τέσσερις :

1:Καταδίωξη αντιπάλου

```
if(health.GetHealth() < 35 )
{
	state = loadHealth;
}
else if(health.GetReceivingDamage())
{
	state = evade;
}
else if(ammo.GetSpeedAmount() + ammo.GetHomingAmount() + ammo.GetPowerAmount()  < 3)
{
	state = loadWeapons;
}
```

2:Αποφυγή αντιπάλου

```
if(health.GetHealth() < 35 )
{
	state = loadHealth;
}
else if(!health.GetReceivingDamage())
{
	state = chase;
}
else if(ammo.GetSpeedAmount() + ammo.GetHomingAmount() + ammo.GetPowerAmount() < 3)
{
	state = loadWeapons;
}
```

3:Αναπλήρωση υγείας

```
if(health.GetHealthTaken())
{
	health.SetHealthTaken(false);
	if(ammo.GetSpeedAmount() + ammo.GetHomingAmount() + ammo.GetPowerAmount() < 3)
	{
			state = loadWeapons;
	}
	else if(health.GetReceivingDamage())
	{
		state = evade;
	}
	else
	{
		state = chase;
```

```
        }
}
```

4:Εξοπλισμός πυραύλων

```
if(ammo.GetWeaponsTaken())
{
        ammo.SetWeaponsTaken(false);
        if(health.GetHealth() < 35 )
        {
                state = loadHealth;
        }
        else if(health.GetReceivingDamage())
        {
                state = evade;
        }
        else
        {
                state = chase;
        }
}
```

Παρατηρούμε ότι υπάρχει μια προτεραιότητα ανάμεσα στις καταστάσεις. Πρώτα ελέγχουμε την υγεία και εάν είναι κάτω από 35 μονάδες τότε μεταβαίνουμε στην κατάσταση LoadHealth ανεξάρτητα από τις τιμές των άλλων μεταβλητών. Σε περίπτωση που δεν ικανοποιήθηκε η πρώτη συνθήκη ελέγχουμε το εάν δεχόμαστε πυρά και στην συνέχεια την ποσότητα των πυρομαχικών. Τέλος για να παραμείνουμε στην ίδια κατάσταση θα πρέπει καμία συνθήκη να μην ικανοποιείται.

Οι F.S.M.s αποτελούν το πρώτο βήμα στην κατανόηση της τεχνητής νοημοσύνης. Η λειτουργία τους όμως, δίνει προβλέψιμα αποτελέσματα, κάτι που είναι ορατό και δυσάρεστο στον χρήστη. Για ένα καλύτερο σύστημα αποφάσεων χρησιμοποιήσαμε ασαφή λογική. Με αυτόν τον τρόπο δεν γνωρίζουμε εκ των προτέρων την απόφαση των αντιπάλων, καθώς κάθε φορά εκτιμούν μια κατάσταση και στην συνέχεια αποφασίζουν.

5.5.2 Ασαφής λογική (Fuzzy Logic)

Η ασαφής λογική βρίσκει πολλές εφαρμογές σε διάφορα πεδία, από συστήματα ελέγχου μέχρι την τεχνητή νοημοσύνη. Παρόλο που υπάρχει αρκετή αμφισβήτηση από τον επιστημονικό κόσμο για την αποτελεσματικότητα της, χρησιμοποιείται συχνά σε ηλεκτρονικά παιχνίδια. Ως εναλλακτική λύση υπάρχει η Bayesian logic η οποία χρησιμοποιεί επιπλέον και στατιστικά δεδομένα για τη λήψη αποφάσεων.

Στη δική μας εφαρμογή, η ασαφής λογική καλύπτει πλήρως τις απαιτήσεις μας. Παρακάτω παρουσιάζουμε τη διαδικασία υπολογισμού για την κατάσταση “Αναπλήρωση υγείας”.

Fuzzification

Ορίζουμε την ασαφή μεταβλητή “ Προτεραιότητα ” η οποία θα αντιπροσωπεύει το βαθμό προτεραιότητας της κάθε κατάστασης. Σε πρώτη φάση συλλέγονται όλα τα απαραίτητα δεδομένα.

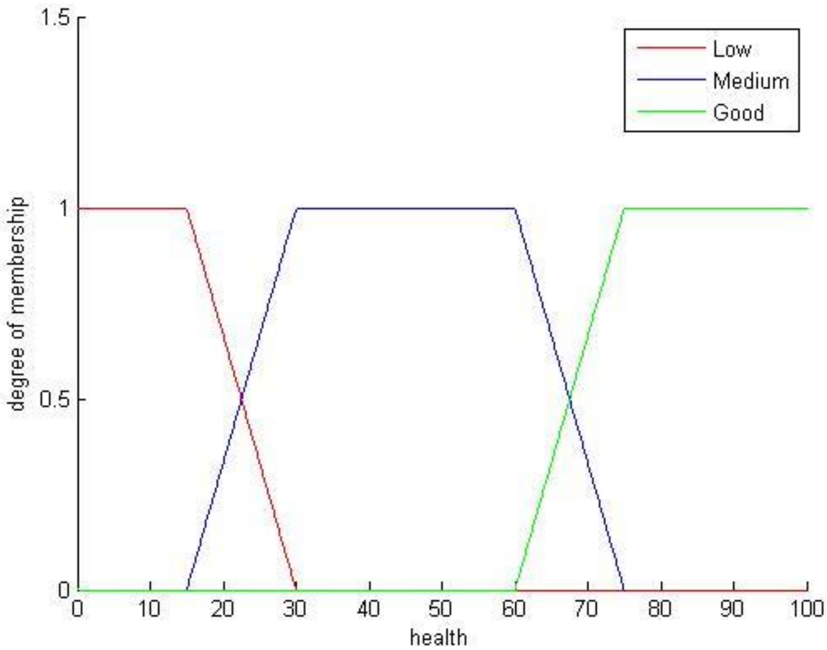
- Υγεία αυτοκινήτου
- Αριθμός πυρομαχικών (ισούται με το άθροισμα όλων των πυραύλων που διαθέτουμε)
- Υγεία πλησιέστερου αντιπάλου
- Αριθμός πυρομαχικών πλησιέστερου αντιπάλου
- Πλησιέστερη αναπλήρωση υγείας
- Πλησιέστερη αναπλήρωση πυρομαχικών

Για κάθε μια από τις παραπάνω μεταβλητές κατασκευάζουμε γραφικές παραστάσεις. Στον άξονα X τοποθετούμε την τιμή της εκάστοτε μεταβλητής ενώ ο άξονας Y αντιπροσωπεύει το ποσοστό συμμετοχής. Παρατηρώντας προσεκτικά τις γραφικές παραστάσεις (Εικόνα 5.8, 5.9) υπάρχουν τρεις ξεχωριστές καμπύλες. Κάθε καμπύλη αντιστοιχεί σε έναν λεκτικό όρο. Θα αναλύσουμε λεπτομερώς την περίπτωση υπολογισμού της προτεραιότητας υγείας. Οι υπόλοιπες περιπτώσεις υπολογίζονται με αντίστοιχο τρόπο. Οι λεκτικοί όροι για το γράφημα ποσοστό συμμετοχής – υγεία (degree of membership – health), είναι Χαμηλή, Μέτρια, Καλή. Έχοντας την τιμή της υγείας υπολογίζουμε το ποσοστό συμμετοχής που αντιστοιχεί σε κάθε μία καμπύλη. Για παράδειγμα εάν η υγεία ισούται με 22.5 τότε θα έχουμε τα εξής ποσοστά :

Ποσοστά υγείας:

- Χαμηλή = 0.5
- Μέτρια = 0.5
- Καλή = 0

lowHealth = FuzzyReverseGrade(tempHealth,15,30);
mediumHealth = FuzzyTrapezoid(tempHealth,15,30,60,75);
goodHealth = FuzzyGrade(tempHealth,60,75);



Εικόνα 5.8: Ποσοστό υγείας

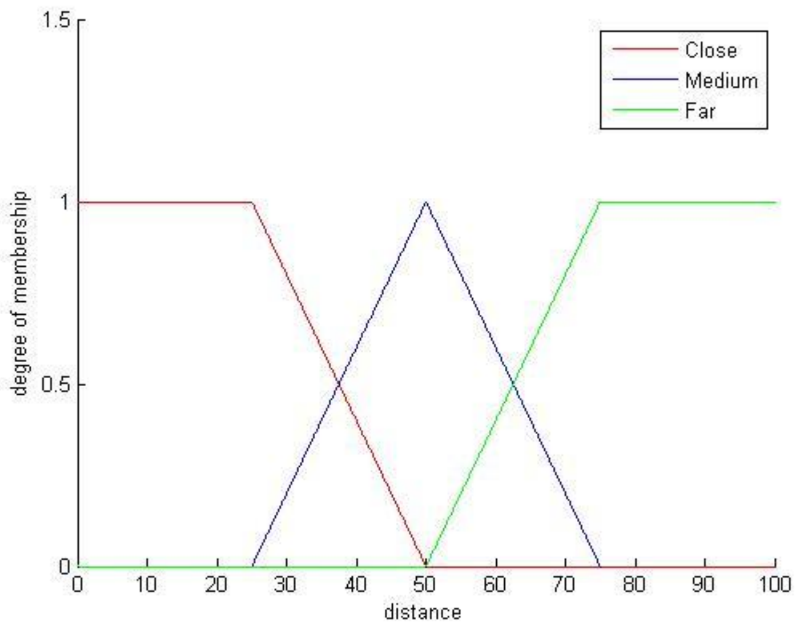
Η κρισιμότητα της αναπλήρωσης υγείας είναι σχετική με την απόσταση που βρίσκεται ένα κουτάκι (ή αλλιώς Power up) που θα μας δώσει υγεία. Άρα φτιάχνουμε πάλι ένα γράφημα με τρεις καμπύλες, με τον άξονα X να έχει την απόσταση και τον Y το ποσοστό. Να σημειώσουμε ότι μπορούμε να φτιάξουμε όσες καμπύλες θέλουμε, για την δική μας περίπτωση είναι αρκετές τρεις.

Έστω λοιπόν ότι η απόσταση είναι ίση με 50 μονάδες μήκους. Τα αντίστοιχα ποσοστά για τους λεκτικούς όρους είναι :

Ποσοστά απόστασης από το κοντινότερο “ κουτί ” αναπλήρωσης υγείας:

- Κοντά : 0
- Μέτρια : 1
- Μακριά: 0

closeDistance = FuzzyReverseGrade(distance,25,50);
mediumDistance = FuzzyTriangle(distance,25,50,75);
farDistance = FuzzyGrade(distance,50,75);



Εικόνα 5.9: Ποσοστό απόστασης

Έχοντας πλέον και αυτά τα δεδομένα μπορούμε να περάσουμε στο επόμενο στάδιο όπου θα συνδυάσουμε τους λεκτικούς όρους και τα ποσοστά σύμφωνα με ένα σύνολο κανόνων.

Fuzzy Rules

Η ασαφής μεταβλητή "Προτεραιότητα" όπως αναφέραμε παραάνω αντιπροσωπεύει τον βαθμό κρισιμότητας της κάθε κατάστασης. Όπως και για τις προηγούμενες μεταβλητές, θα κατασκευάσουμε ένα γράφημα με καμπύλες. Επειδή χρειαζόμαστε περισσότερη ακρίβεια, θα χρησιμοποιήσουμε τέσσερις καμπύλες οι οποίες είναι οι εξής: Χαμηλή, Φυσιολογική, Υψηλή, Κρίσιμη.

Οι λεκτικοί όροι και τα ποσοστά του προηγούμενου βήματος θα συνδυαστούν με το ασαφές Και (Fuzzy AND) το οποίο μας επιστρέφει την μικρότερη από τις δύο συγκρινόμενες τιμές. Για παράδειγμα ο πρώτος κανόνας είναι ο συνδυασμός Χαμηλή υγεία (Έστω 0.5) – Κοντινή απόσταση (Έστω 0.7) και αντιστοιχούν στην καμπύλη Κρίσιμη (Critical). Το αποτέλεσμα είναι ο κανόνας αυτός να έχει τιμή 0.5. Δηλαδή η τιμή της μεταβλητής "προτεραιότητα" είναι το σημείο του άξονα X όπου η καμπύλη Κρίσιμη (Critical) έχει τιμή 0.5. Η αντιστοιχία των όρων μπορεί να αλλάξει κατά βούληση και επαφίεται στον προγραμματιστή και στις ανάγκες του παιχνιδιού. Παρακάτω παραθέτουμε την αντιστοιχία που χρησιμοποιείται στην εφαρμογή μας.

```
rule1 = FuzzyAND(lowHealth,closeDistance);//critical
rule2 = FuzzyAND(lowHealth,mediumDistance);//high
rule3 = FuzzyAND(lowHealth,farDistance);//high
rule4 = FuzzyAND(mediumHealth,closeDistance);//normal
```

```
rule5 = FuzzyAND(mediumHealth,mediumDistance);//low
rule6 = FuzzyAND(mediumHealth,farDistance);//low
rule7 = FuzzyAND(goodHealth,closeDistance);//low
rule8 = FuzzyAND(goodHealth,mediumDistance);//low
rule9 = FuzzyAND(goodHealth,farDistance);//low
```

Για το παράδειγμα μας τα αποτελέσματα είναι:

- *rule1 = 0//critical*
- *rule2 = 0.5//high*
- *rule3 = 0//high*
- *rule4 = 0//normal*
- *rule5 = 0.5//low*
- *rule6 = 0;//low*
- *rule7 = 0;//low*
- *rule8 = 0;//low*
- *rule9 = 0;//low*

Επειδή οι περιπτώσεις Υψηλή (High) και Χαμηλή (Low) είναι παραπάνω από μία θα πρέπει να τις ξανά συνδυάσουμε ώστε να πάρουμε μια μόνο τιμή. Προκειμένου να καταλήξουμε σε μία τιμή χρησιμοποιούμε το ασαφές Η (Fuzzy OR) που μας επιστρέφει την μεγαλύτερη από τις δύο συγκρινόμενες τιμές.

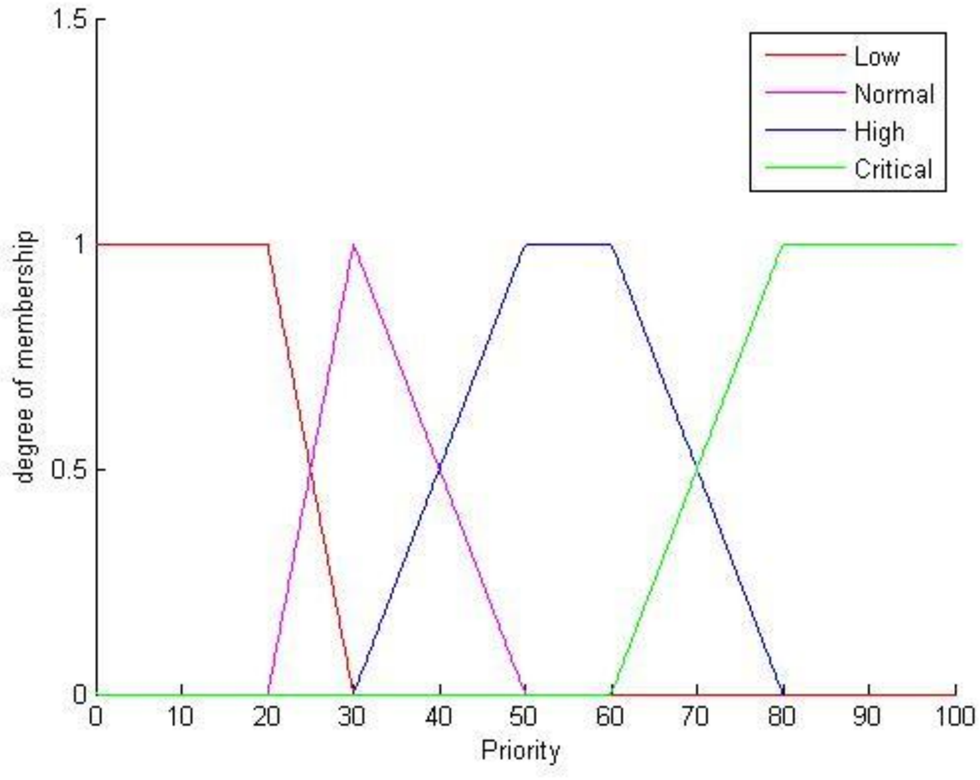
```
// --- Critical
critical = rule1;
// --- High
high = FuzzyOR(high,rule2);
high = FuzzyOR(high,rule3);
// --- Normal
normal = rule4;
// --- Low
low = FuzzyOR(low,rule5);
low = FuzzyOR(low,rule6);
low = FuzzyOR(low,rule7);
low = FuzzyOR(low,rule8);
low = FuzzyOR(low,rule9);
```

Για το παράδειγμα μας τα αποτελέσματα είναι:

- *Critical = 0*
- *High = 0.5*
- *Normal = 0*
- *Low = 0.5*

Defuzzification

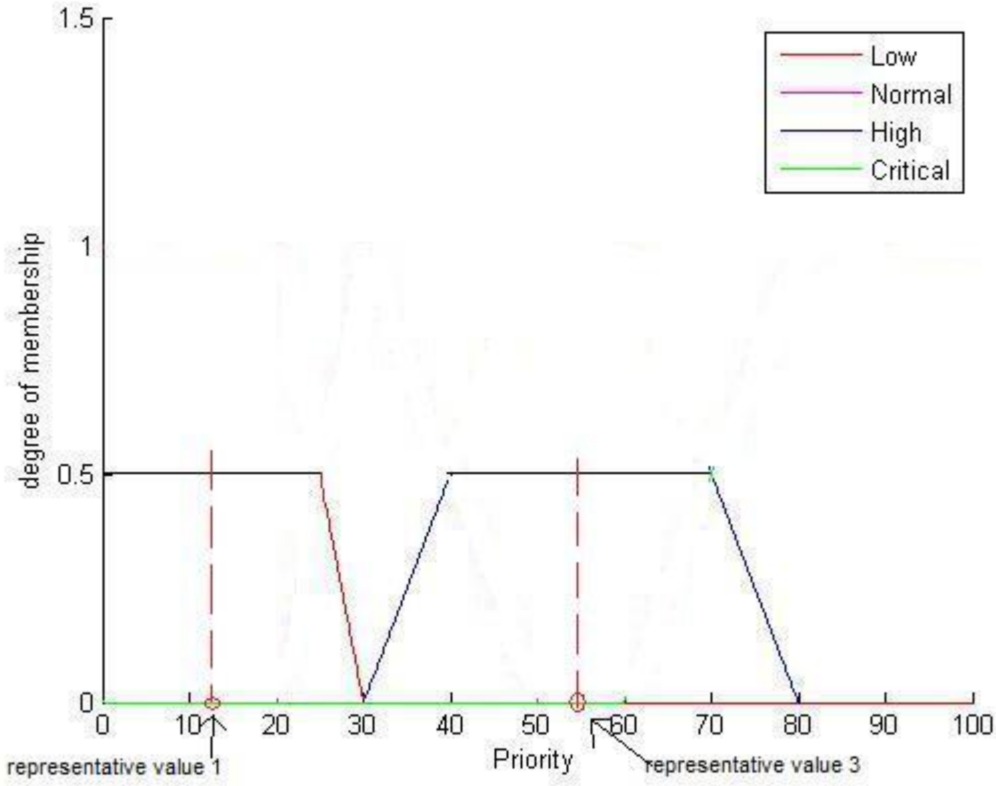
Έχοντας πλέον τα αποτελέσματα των κανόνων, είμαστε σε θέση να πάρουμε την τελική τιμή. Με τη βοήθεια της παρακάτω γραφικής παράστασης βρίσκουμε το ποσοστό συμμετοχής της κάθε κατηγορίας της μεταβλητής "Προτεραιότητα"



Εικόνα 5.10α: Ποσοστό προτεραιότητας

Σε αυτό το σημείο μπορούμε να χρησιμοποιήσουμε διάφορες μεθόδους για να πάρουμε το αποτέλεσμα μας. Οι τρείς πιο γνωστές είναι: Mean Of Maximum, Centroid, Average Of Maxima. Κάθε μια δίνει λίγο διαφορετικά αποτελέσματα, όμως αυτό δεν αποτελεί πρόβλημα την στιγμή που η ίδια μέθοδος εφαρμόζεται σε όλες τις περιπτώσεις. Εμείς επιλέξαμε την μέθοδο Average Of Maxima γιατί μας παρέχει ένα αξιόπιστο αποτέλεσμα και ταυτόχρονα είναι γρήγορη. Αντιθέτως η Mean Of Maximum δεν είναι αρκετά ακριβής ενώ η Centroid χρειάζεται πολλούς υπολογισμούς για να μας δώσει αποτέλεσμα.

Σύμφωνα με τα δεδομένα του παραδείγματος μας οι καμπύλες για την μεταβλητή "Προτεραιότητα" θα μετατραπούν σε αυτές της εικόνας 5.10β. Η μέθοδος Average Of Maxima θα επιστρέψει για κάθε καμπύλη μια αντιπροσωπευτική τιμή για τον άξονα Χ. Η τιμή αυτή αντιστοιχεί στο κέντρο της μικρής βάσης των τραπέζιων που σχηματίστηκαν. Για τις κατηγορίες Normal και Critical η αντιπροσωπευτική τιμή θα είναι μηδέν αφού το ποσοστό συμμετοχής τους είναι μηδενικό.



Εικόνα 5.10β: Ποσοστό προτεραιότητας

representativeValue1 = DefuzzyReverseGrade(low, 20, 30) ;
representativeValue2 = DefuzzyTriangle(normal,20,30, 50);
representativeValue3 = DefuzzyTrapezoid(high,30,50, 60, 80);
representativeValue4 = DefuzzyGrade(critical, 60, 80,100) ;

Το τελικό βήμα είναι να συνδυάσουμε τις μεταβλητές Low, Normal, High, Critical με τις αντιπροσωπευτικές τιμές με την παρακάτω συνάρτηση.

*Result = ((representativeValue1*low + representativeValue2*normal + representativeValue3*high + representativeValue4*critical)/low+normal+high+critical);*

Παρόμοια διαδικασία ακολουθείται και για τις υπόλοιπες τρεις καταστάσεις (Καταδίωξη αντιπάλου, Αποφυγή αντιπάλου, Εξοπλισμός πυραύλων).

Αφού υπολογίσουμε όλα τα ποσοστά “Προτεραιότητας” για τις καταστάσεις, τις συγκρίνουμε και διαλέγουμε την μεγαλύτερη. Μεγαλύτερη «Προτεραιότητα» σημαίνει για παράδειγμα ότι η Αναπλήρωση υγείας πρέπει να γίνει άμεσα έναντι του να κυνηγήσουμε το αντίπαλο που βρίσκεται μπροστά μας.

Ένα από τα πλεονεκτήματα του Fuzzy Logic είναι και το γεγονός ότι μπορούμε να αλλάξουμε τις καμπύλες και τους κανόνες. Αυτός είναι ένας τρόπος για να πετύχουμε διαφορετικά επίπεδα δυσκολίας. Για παράδειγμα όσο δυσκολεύει το επίπεδο, οι αντίπαλοι γίνονται πιο επιθετικοί αν αυξήσουμε την "Προτεραιότητας" για την κατάσταση Καταδίωξη.

5.6 Ο αλγόριθμος Α*

5.6.1 Εύρεση συντομότερου μονοπατιού

Οι αγώνες του παιχνιδιού πραγματοποιούνται σε ανοιχτό περιβάλλον και όπως έχουμε αναφέρει προηγουμένως κάθε τριάντα δευτερόλεπτα εμφανίζεται μια σημαία την οποία όλα τα αυτοκίνητα τρέχουν να πάρουν. Επειδή οι αντίπαλοι πρέπει να είναι ανταγωνιστικοί ο Α* υπολογίζει το καλύτερο μονοπάτι έτσι ώστε οι αντίπαλοι να φτάσουν στη σημαία όσο το δυνατόν γρηγορότερα.

Το επίπεδο στο οποίο γίνονται οι αγώνες περιέχει ένα αντικείμενο (GameObject) με όνομα Flag. Αυτό το αντικείμενο περιέχει το script FlagScript το οποίο διαχειρίζεται οτιδήποτε έχει σχέση με τη σημαία και τον Α*. Οι αντίπαλοι μας είναι επτά, επομένως κάθε φορά που εμφανίζεται η σημαία ο Α* θα πρέπει να εκτελεστεί επτά φορές. Η εκτέλεση γίνεται σειριακά και όχι παράλληλα, δηλαδή ολοκληρώνεται ο Α* για το πρώτο αυτοκίνητο, στη συνέχεια για το δεύτερο και ούτω καθεξής. Η παράλληλη εκτέλεση του αλγορίθμου προκαλούσε προβλήματα στην ομαλή λειτουργία του παιχνιδιού κάτι που μπορεί να μην οφείλεται τόσο στην πολυπλοκότητα του αλγορίθμου όσο στην απαίτηση για μαζική και άμεση κατοχύρωση μνήμης (memory allocation).

Αρχικά έπρεπε να αντιμετωπίσουμε ένα τεχνικό πρόβλημα. Ο Α* λειτουργεί σε περιβάλλοντα με «κουτάκια» (tiled environment) και δεν είναι κατάλληλος για την δική μας πίστα. Για αυτό το λόγο πριν μπουν τα αυτοκίνητα στην πίστα δημιουργούμε ένα τεχνητό πλέγμα. Στον δίσκο είναι αποθηκευμένα όλα τα δεδομένα των κόμβων και κατά το χρόνο εκτέλεσης(runtime) τοποθετούμε τους κόμβους στις θέσεις τους. Οι κόμβοι που βρίσκονται μέσα σε κτήρια αυτομάτως χαρακτηρίζονται ως μη προσπελάσιμοι. Τα δεδομένα των κόμβων κρατούνται στην κλάση NodeInfo και το κόστος κάθε ακμής στην EdgeInfo.

Πριν ξεκινήσει τη λειτουργία του ο Α* χρειάζεται να γνωρίζουμε τον αρχικό και τελικό κόμβο. Ο τελικός κόμβος υπολογίζεται από δύο random() συναρτήσεις μια για κάθε διάσταση.

```
while(true)
    {
        x = Mathf.Round( (UnityEngine.Random.value)*19);
        y = Mathf.Round( (UnityEngine.Random.value)*19);

        if( (grid[x][y] as NodeInfo).accessible )
        {
            break;
```

```
		}
	}
```

Για τον αρχικό κόμβο πρέπει να γνωρίζουμε τη θέση του εκάστοτε αυτοκινήτου στην πίστα. Επειδή οι κόμβοι είναι συγκεκριμένοι (400 στον αριθμό) δεν είναι δυνατό να αντιστοιχούν σε όλες τις πιθανές θέσεις των αυτοκινήτων.

Η συνάρτηση function FindStart(goX: float,goZ: float): NodeInfo αναλαμβάνει να βρει τον πιο κοντινό διαθέσιμο κόμβο για κάθε αυτοκίνητο. Αρχικά βρίσκουμε τον κόμβο ο οποίος έχει απόσταση μικρότερη από ένα κατώφλι. Στη συνέχεια υπολογίζουμε την ευκλείδεια απόσταση του αυτοκίνητου από τον εν λόγω κόμβο αλλά και από όλους τους γειτονικούς του. Τελικά διαλέγουμε ως αρχικό, τον κόμβο με τη μικρότερη απόσταση από το αυτοκίνητο.

```
// Find the node that is closer to the car position
function FindStart(goX : float ,goZ : float) : NodeInfo
{
	var tempX : float = goX;
	var tempY : float = goZ;
	var tempGridX : int = 0;
	var tempGridY : int = 0;
	var closestNeighborDistance : float = Mathf.Infinity;
	var closestNeighbor : int = 0;

	var ok1:boolean = false;
	var ok2:boolean = false;
	var ok3:boolean = false;
	var ok4:boolean = false;
	var ok5:boolean = false;
	var ok6:boolean = false;
	var ok7:boolean = false;
	var ok8:boolean = false;

	for(i=0;i<grid.length;i++)
	{
		var temp = grid[1][i] as NodeInfo;

		if(Mathf.Abs(tempX-temp.posX) <= threshold )
		{
			tempGridY = i;

			break;
		}

	}

	for(j=0;j<grid.length;j++)
	{
		temp = grid[j][tempGridY] as NodeInfo;
```

```
        if(Mathf.Abs(tempY-temp.posY) <= threshold )
        {
                tempGridX = j;

                break;
        }

}
zero=Heuristic(grid[tempGridX][tempGridY].posX,grid[tempGridX]
[tempGr  idY].posY,tempX,tempY );
        try
        {
                one=Heuristic(grid[tempGridX-1][tempGridY-
                1].posX,grid[tempGridX-1][tempGridY-1].posY   ,tempX,tempY
                );
                ok1=true;
        }
        catch(Exception){ one = Mathf.Infinity;}
        try
        {

                two=Heuristic(grid[tempGridX+1][tempGridY+1].posX,grid
                [tempGridX+1][tempGridY+1].posY ,tempX,tempY );
                ok2=true;
        }
        catch(Exception){ two = Mathf.Infinity;}
        try
        {
                three=Heuristic(grid[tempGridX-
                1][tempGridY+1].posX,grid[tempGridX-
                1][tempGridY+1].posY ,tempX,tempY );
                ok3=true;
        }
        catch(Exception){ three = Mathf.Infinity;}
        try
        {
                four=Heuristic(grid[tempGridX+1][tempGridY-
                1].posX,grid[tempGridX+1][tempGridY-1].posY ,tempX,tempY
                );
                ok4=true;
        }
        catch(Exception){ four = Mathf.Infinity;}
        try
        {
                five=Heuristic(grid[tempGridX-
                1][tempGridY].posX,grid[tempGridX-1][tempGridY].posY
                ,tempX,tempY );
                ok5=true;
        }
```

```
catch(Exception){ five = Mathf.Infinity;}
try
{
	six=Heuristic(grid[tempGridX][tempGridY-
	1].posX,grid[tempGridX][tempGridY-1].posY ,tempX,tempY );
	ok6=true;
}
catch(Exception){ six = Mathf.Infinity;}
try
{
	seven=Heuristic(
	grid[tempGridX+1][tempGridY].posX,grid[tempGridX+1][te
	mpGridY].posY ,tempX,tempY );
	ok7=true;
}
catch(Exception){ seven = Mathf.Infinity;}
try
{
	eight=Heuristic(
	grid[tempGridX][tempGridY+1].posX,grid[tempGridX][temp
	GridY+1].posY ,tempX,tempY );
	ok8=true;
}
catch(Exception){ eight = Mathf.Infinity;}

if(	(grid[tempGridX][tempGridY] as NodeInfo).accessible &&
zero < closestNeighborDistance )
{
	closestNeighbor = 0;
	closestNeighborDistance = zero ;
}
if(ok1 && (grid[tempGridX-1][tempGridY-1] as NodeInfo).accessible
&& one < closestNeighborDistance )
{
	closestNeighbor = 1;
	closestNeighborDistance = one ;
}
if(	ok2	&&	(grid[tempGridX+1][tempGridY+1]	as
NodeInfo).accessible && two < closestNeighborDistance)
{
	closestNeighbor = 2;
	closestNeighborDistance = two;
}
if(	ok3	&&	(grid[tempGridX-1][tempGridY+1]	as
NodeInfo).accessible && three < closestNeighborDistance)
{
	closestNeighbor = 3;
	closestNeighborDistance = three;
}
```

```
        if( ok4 && (grid[tempGridX+1][tempGridY-1] as
        NodeInfo).accessible && four  < closestNeighborDistance)
        {
                closestNeighbor = 4;
                closestNeighborDistance = four;
        }
        if( ok5 && (grid[tempGridX-1][tempGridY] as NodeInfo).accessible
        && five < closestNeighborDistance)
        {
                closestNeighbor = 5;
                closestNeighborDistance =  five;
        }
        if( ok6 &&  (grid[tempGridX][tempGridY-1] as NodeInfo).accessible
        && six < closestNeighborDistance)
        {
                closestNeighbor = 6;
                closestNeighborDistance = six;
        }
        if( ok7 &&  (grid[tempGridX+1][tempGridY] as NodeInfo).accessible
        && seven < closestNeighborDistance)
        {
                closestNeighbor = 7;
                closestNeighborDistance = seven;
        }
        if( ok8 &&  (grid[tempGridX][tempGridY+1] as NodeInfo).accessible
        && eight < closestNeighborDistance)
        {
                closestNeighbor = 8;
                closestNeighborDistance = eight;
        }
if(closestNeighbor == 0)
{
        return grid[tempGridX][tempGridY];
}

else if(closestNeighbor == 1)
{
        return grid[tempGridX-1][tempGridY-1];
}
else if(closestNeighbor == 2)
{
        return grid[tempGridX+1][tempGridY+1];
}
else if(closestNeighbor == 3)
{
        return grid[tempGridX-1][tempGridY+1];
}
else if(closestNeighbor == 4)
{
        return grid[tempGridX+1][tempGridY-1];
```

```
        }
        else if(closestNeighbor == 5)
        {
                return grid[tempGridX-1][tempGridY];
        }
        else if(closestNeighbor == 6)
        {
                return grid[tempGridX][tempGridY-1];
        }
        else if(closestNeighbor == 7)
        {
                return grid[tempGridX+1][tempGridY];
        }
        else if(closestNeighbor == 8)
        {
                return grid[tempGridX][tempGridY+1];
        }


}
```

Έχοντας στη διάθεση μας το πλέγμα, την αρχική και τελική θέση μπορούμε να ξεκινήσουμε τον Α* για την εύρεση του συντομότερου μονοπατιού. Αρχικά αδειάζουμε την openList και closedList σε περίπτωση που υπάρχουν υπολείμματα μνήμης από προηγούμενη εκτέλεση. Στη συνέχεια βάζουμε στην openList μια κενή καταχώρηση. Αυτό συμβαίνει γιατί η openList είναι μια priority queue σε αντίθεση με την closed που είναι συνδεδεμένη λίστα. Η priority queue είναι ένας αφηρημένος τύπος δεδομένων που μπορεί να υλοποιηθεί με διάφορους τρόπους. Στην παρούσα εργασία χρησιμοποιούμε ελάχιστο σωρό(minHeap). Για να λειτουργήσει ο κώδικας για το σωρό χρειάζεται να έχουμε πάντα στην πρώτη θέση μια κενή καταχώρηση. Έπειτα στην openList μπαίνει ο αρχικός κόμβος και ξεκινάει ο βασικός βρόχος (loop) του αλγορίθμου. Η συνθήκη του βρόχου είναι αυτός να εκτελείται συνέχεια μέχρι η openList να μην έχει καταχωρήσεις πλην της κενής καταχώρησης που προσθέσαμε στην αρχή.

Στο επόμενο βήμα θα πρέπει να αναζητήσουμε στην openList τον κόμβο με το χαμηλότερο κόστος, να τον αφαιρέσουμε και να τον χρησιμοποιήσουμε στις επόμενες διαδικασίες. Εδώ ακριβώς φαίνεται η διαφορά μεταξύ συνδεδεμένης λίστας και ελάχιστου σωρού. Η χρονική πολυπλοκότητα αναζήτησης και διαγραφής για συνδεδεμένη λίστα είναι O(n) και Θ(1) αντίστοιχα. Για τον ελάχιστο σωρό (με δυαδικό δένδρο), η αναζήτηση έχει κόστος Θ(1) και η διαγραφή Θ(logn). Να υπενθυμίσουμε ότι μας ενδιαφέρει πάντα ο κόμβος με το χαμηλότερο κόστος. Σε αντίθετη περίπτωση, ο σωρός δεν θα είχε την ίδια πολυπλοκότητα.

```
currentNode = deleteMin();
```

Αφού πάρουμε τον κόμβο με το μικρότερο κόστος, ελέγχουμε πρώτα εάν αυτός είναι ο τελικός κόμβος. Εάν ναι, τότε κάνουμε διάσχιση από τον τελικό κόμβο μέχρι τον αρχικό φτιάχνοντας το μονοπάτι μας σε συνδεδεμένη λίστα την οποία στο τέλος αντιστρέφουμε και ο αλγόριθμος τελειώνει.

```
//End case -- result has been found, return the traced path
        if(currentNode.posX == end.posX && currentNode.posY == end.posY
)
        {
            var curr: NodeInfo = currentNode;
            ret.clear();
            var counter: int = 0;
            try{
                while(true)
                {
                    ret.push(grid[curr.gridPosX][curr.gridPosY]);
                    cur=grid[curr.parentGridPosX]
                    curr.parentGridPosY];
                    counter++;

                    if(counter == 400   )
                    {
                        Debug.Log("error");
                        break;
                    }
                    else    if(curr.gridPosX==start.gridPosX    &&
                    curr.gridPosY==start.gridPosY)
                    {
                        Debug.Log("path complete");
                        break;
                    }
                }
            }
            catch(Exception){
            }
            ret.reverse();
            return 1;
        }
```

Εάν όχι τότε βάζουμε τον κόμβο στην closedList και παίρνουμε τους γείτονες του. Η συνάρτηση Neighbors (grid: Array, node: NodeInfo): Array θα μας επιστρέψει μια λίστα με όλους τους προσβάσιμους γειτονικούς κόμβους και στην συνέχεια θα επεξεργαστούμε κάθε έναν από αυτούς μέσα σε ένα for loop. Παράλληλα θα γεμίσει μια άλλη λίστα που περιέχει το κόστος G για κάθε γείτονα.

```
closedList.push(currentNode);
var neighbors = Neighbors(grid, currentNode);
for(i=0; i<neighbors.length;i++)
{ … }
```

Στον καινούργιο βρόχο που ανοίξαμε ελέγχουμε εάν ο γείτονας βρίσκεται στην closedList. Εάν συμβαίνει αυτό τότε έχουμε ήδη επεξεργαστεί αυτόν τον κόμβο και συνεχίζουμε με τον επόμενο κόμβο.

```
if(FindGraphNode1(closedList,neighbor.id))
{
        continue;
}
```

Αν ο γειτονικός κόμβος δε βρίσκεται στην κλειστή λίστα τότε υπολογίζουμε το κόστος G για την πρόσβαση σε αυτόν. Το κόστος G είναι το άθροισμα του κόστους G του «πατέρα» συν το κόστος G για την πρόσβαση στον γειτονικό κόμβο.

```
var gScore = currentNode.g +neighborGscore[i];
```

Η επόμενη περίπτωση που ελέγχουμε είναι για το αν ο γειτονικός κόμβος δεν βρίσκεται στην openList. Εάν δεν βρίσκεται τότε υπολογίζουμε το ευρεστικό κόστος που είναι η ευκλείδεια απόσταση από τον γειτονικό μέχρι τον τελικό κόμβο. Αποθηκεύουμε πληροφορίες όπως οι συντεταγμένες του κόμβου πατέρα, για να γνωρίζουμε την διαδρομή που ακολουθούμε. Επίσης αποθηκεύουμε το κόστος G και H καθώς και το συνολικό κόστος F και τέλος προσθέτουμε το γειτονικό κόμβο στην openList.

```
if(!FindGraphNode2(openList,neighbor.id))
{
        //This the first time we have arrived at this node, it must be the best
        //Also, we need to take the h (heuristic) score since we haven't done so yet
        neighbor.h = Heuristic(neighbor.posX,neighbor.posY,end.posX,end.posY);

        neighbor.parentID = currentNode.id;
        neighbor.parentGridPosX = currentNode.gridPosX;
        neighbor.parentGridPosY = currentNode.gridPosY;

        neighbor.g = gScore;
        neighbor.f = neighbor.g + neighbor.h;
        openList.push(neighbor);
        insert(neighbor);
}
```

Στην περίπτωση που ο γειτονικός κόμβος βρίσκεται στην openList εξετάζουμε εάν το κόστος G από τον παρόντα κόμβο πατέρα είναι καλύτερο από του προηγούμενου. Εάν συμβαίνει αυτό τότε υπολογίζουμε τα νέα κόστη(G,H,F) αποθηκεύουμε το νέο «πατέρα» και ανανεώνουμε τη θέση του κόμβου στον σωρό. Τη λειτουργία της ανανέωσης θέσης αναλαμβάνει η συνάρτηση swim(k: int)η οποία θα αναλυθεί παρακάτω.

```
else if(gScore < neighbor.g)
```

```
{
//We have already seen the node, but last time it had a worse g (distance from start)

        neighbor.h    =    Heuristic(neighbor.posX,neighbor.posY,end.posX,end.posY);

        neighbor.parentID = currentNode.id;
        neighbor.parentGridPosX = currentNode.gridPosX;
        neighbor.parentGridPosY = currentNode.gridPosY;

        neighbor.g = gScore;
        neighbor.f = neighbor.g + neighbor.h;

        swim(neighbor.priorityQueueID,neighbor);
}
```

Έχοντας κάνει όλους τους ελέγχους για όλους τους κόμβους που περιέχονται στην openList ο αλγόριθμος θα μας επιστρέψει το συντομότερο μονοπάτι.

5.6.2 Βελτίωση Α* με ουρά προτεραιότητας

Η συναρτήσεις για την ουρά προτεραιότητας είναι εξειδικευμένες για τη συγκεκριμένη εργασία.

Για την εισαγωγή στοιχείου, προσθέτουμε τον κόμβο στο τέλος του σωρού και καλούμε την συνάρτηση swim(k: int). Η swim κάνει τακτοποίηση σωρού από κάτω προς τα πάνω, θα συγκρίνει το νέο στοιχείο και θα το τοποθετήσει στο κατάλληλο σημείο έτσι ώστε στην κορυφή του σωρού να έχουμε και πάλι τον κόμβο με το μικρότερο κόστος.

Η συνάρτηση deleteMin(): NodeInfo αναλαμβάνει τη διαγραφή του πρώτου και μόνο στοιχείου το οποίο και επιστρέφει. Για να γίνει η διαγραφή ανταλλάσουμε το πρώτο με το τελευταίο στοιχείο του σωρού. Στη συνέχεια επαναπροσδιορίζουμε τη θέση του «νέου πρώτου στοιχείου» με την συνάρτηση sink(k:int, N:int) η οποία κάνει τακτοποίηση σωρού από πάνω προς τα κάτω. Αφού γίνει και αυτό το βήμα ο σωρός μας είναι ταξινομημένος εκτός από το τελευταίο του στοιχείο που πριν βρισκόταν στην πρώτη θέση. Διαγράφουμε λοιπόν το στοιχείο αυτό από τον σωρό και το επιστρέφουμε.

Οι συναρτήσεις υπεύθυνες για την διαχείριση του σωρού είναι οι παρακάτω:

```
private var N: int = 0;
private var minNodef: NodeInfo;
private function  insert(newElement: NodeInfo)
{
        N=N+1;
        newElement.priorityQueueID = N;
        swim(N);
}
private function  deleteMin(): NodeInfo
```

```
{
   if (N == 0) throw new Exception("Priority queue underflow");
   exch(1, N);
         sink(1, N-1);
   minNodef = openList.pop();
   N=N-1;
   return minNodef;
}
private function  greater( i: int, j:int): boolean
{
         return (openList[i].f > openList[j].f);
}
private function  exch( i:int, j:int)
{
         var id1 = openList[i].priorityQueueID;
         openList[i].priorityQueueID = openList[j].priorityQueueID;
         openList[j].priorityQueueID = id1;

         var  t: NodeInfo = openList[i];
         openList[i] = openList[j];
         openList[j] = t;

}
 private function  swim(k:int)
 {
         while (k > 1 && greater(k/2, k))
         {
                   exch(k, k/2);
                   k = k/2;
         }
}
private function  sink( k:int, N:int)
{
      while (2*k <= N) {
         var j:int = 2*k;
         if (j < N && greater(j, j+1)){ j++;}
         if (!greater(k, j)){ break;}
         exch(k, j);
         k = j;
      }
}
```

Εικόνα 5.11α: Κάτοψη πίστας

Εικόνα 5.11β: Κάτοψη πίστας με ορατό το πλέγμα για τον A*

Κεφάλαιο 6

Αξιολόγηση συστήματος

6.1 Μέθοδος αξιολόγησης

Για την αξιολόγηση της εργασίας, έγινε χρήση του ερωτηματολόγιου χρηστικότητας QUIS(Questionnaire for User Interaction Satisfaction). Οι εθελοντές που συμμετείχαν απάντησαν σε ερωτήσεις που αφορούσαν την εμφάνιση, την λειτουργικότητα, την ευκολία εκμάθησης και τις γενικές δυνατότητες της εφαρμογής.

Στην διαδικασία αξιολόγησης συμμετείχαν δέκα χρήστες (πέντε άνδρες και πέντε γυναίκες), ηλικίας από 18 έως 24 χρονών, από διαφορετικούς επαγγελματικούς χώρους, οι οποίοι έχουν διαφορετική εξοικείωση με ψηφιακές εφαρμογές.

Το ερωτηματολόγιο QUIS περιλαμβάνει 39 ερωτήσεις, χωρισμένες σε πέντε ενότητες. Οι απαντήσεις δίνονται σε κλίμακα από ένα (1) έως εννιά (9) ή δεν γνωρίζω/ δεν απαντώ (NA).Παρακάτω παραθέτουμε τα ερωτηματολόγιο.

Ερωτηματολόγιο Αξιολόγησης Διαλογικής Χρήσης Συστήματος

(QUIS - Πανεπιστήμιο του Maryland, 1997)

Αριθμός ερ/γίου: _____ Σύστημα: _________________ Ηλικία: ______ Φύλο: ___ άρρεν ___ θήλυ___

<u>ΜΕΡΟΣ 1: Γενική εντύπωση του χρήστη</u>

Παρακαλώ κυκλώστε τους αριθμούς που αντικατοπτρίζουν ακριβέστερα τις εντυπώσεις σας από τη χρήση αυτού του υπολογιστικού συστήματος. Δεν ξέρω/δεν απαντώ = NA.

1.1. Γενική αντίδραση του Συστήματος:

απαράδεκτη υπέροχη

1 2 3 4 5 6 7 8 9 NA

1.2. Γενική αντίδραση του Συστήματος:

μπερδεύει ικανοποιεί

1 2 3 4 5 6 7 8 9 NA

1.3. Γενική αντίδραση του Συστήματος:

χαζό ευχάριστο

1 2 3 4 5 6 7 8 9 NA

1.4. Γενική αντίδραση του Συστήματος:

δύσκολο εύκολο

1 2 3 4 5 6 7 8 9 NA

1.5. Γενική αντίδραση του Συστήματος:

άκαμπτο ευέλικτο

1 2 3 4 5 6 7 8 9 NA

<u>ΜΕΡΟΣ 2: Οθόνη</u>

2.1. Ο σχεδιασμός της οθόνης βοήθησε:

2.1.1. Ο σχεδιασμός της οθόνης βοήθησε: ποτέ πάντα

1 2 3 4 5 6 7 8 9 NA

2.1.2. Ποσότητα πληροφορίας που εμφανίζονταν στην οθόνη:

ανεπαρκής επαρκής

1 2 3 4 5 6 7 8 9 NA

2.1.3. Δόμηση της πληροφορίας που εμφανίζονταν στην οθόνη:

χαοτική οργανωμένη

1 2 3 4 5 6 7 8 9 NA

2.2. Αλληλουχία οθονών:

2.2.1. Αλληλουχία οθονών: μπερδεμένη σαφής

1 2 3 4 5 6 7 8 9 NA

2.2.2. Επόμενη οθόνη στη σειρά: απρόβλεπτη προβλέψιμη

1 2 3 4 5 6 7 8 9 NA

2.2.3. Επιστροφή στην προηγούμενη οθόνη: αδύνατη εύκολη

1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλια σας για τη δομή των οθονών εδώ:

ΜΕΡΟΣ 3: Ορολογία και επικοινωνία με το Σύστημα

3.1. Τα μηνύματα εμφανίζονται στην οθόνη με:

3.1.1. Τα μηνύματα εμφανίζονται στην οθόνη με:

ασυνέπεια συνέπεια

1 2 3 4 5 6 7 8 9 NA

3.2. Τα μηνύματα που εμφανίζονται στην οθόνη είναι:

3.2.1. Τα μηνύματα που εμφανίζονται στην οθόνη είναι:

μπερδεμένα σαφή

1 2 3 4 5 6 7 8 9 NA

3.3. Ο υπολογιστής σας ενημερώνει για το τι κάνει:

3.3.1. Ο υπολογιστής σας ενημερώνει για το τι κάνει:

ποτέ πάντα

1 2 3 4 5 6 7 8 9 NA

3.3.2. Εκτελώντας μια κίνηση οδηγούμαστε σε προβλέψιμο αποτέλεσμα:

ποτέ πάντα

1 2 3 4 5 6 7 8 9 NA

3.3.3. Ο έλεγχος της ποσότητας της ανάδρασης του Συστήματος:

αδύνατος εύκολος

1 2 3 4 5 6 7 8 9 NA

3.3.4 Καθυστερήσεις: απαράδεκτες παραδεκτές

1 2 3 4 5 6 7 8 9 NA

3.4. Μηνύματα λαθών:

3.4.1.Μηνύματα λαθών: δεν βοηθούν πολύ βοηθητικά

1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλιά σας για την ορολογία και την επικοινωνία με το Σύστημα εδώ:

____
____
____

<u>ΜΕΡΟΣ 4: Εκμάθηση χρήσης</u>

4.1. Η εκμάθηση χρήσης του συστήματος είναι:

4.1.1. Η εκμάθηση χρήσης του συστήματος είναι:

δύσκολη εύκολη

1 2 3 4 5 6 7 8 9 NA

4.1.2. Χρόνος για την εκμάθηση χρήσης του συστήματος:

πολύς λίγος

1 2 3 4 5 6 7 8 9 NA

4.2. Η εξερεύνηση των δυνατοτήτων με τη μέθοδο “προσπάθειας και λάθους (trial and error)”:

4.2.1. Η εξερεύνηση των δυνατοτήτων με τη μέθοδο “προσπάθειας και λάθους (trial and error)”:

απογοητεύει αποδίδει

1 2 3 4 5 6 7 8 9 NA

4.3. Οι εργασίες γίνονται σε λογική αλληλουχία:

4.3.1. Οι εργασίες γίνονται σε λογική αλληλουχία:

ποτέ πάντα

1 2 3 4 5 6 7 8 9 NA

4.3.2. Τα βήματα για την ολοκλήρωση της εργασίας ακολουθούν μια λογική σειρά

ποτέ πάντα

1 2 3 4 5 6 7 8 9 NA

4.3.3 Ανάδραση όταν ολοκληρωθεί η εργασία:

ασαφής σαφής

1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλιά σας για την εκμάθηση της χρήσης εδώ:

____
____
____

ΜΕΡΟΣ 5: Δυνατότητες του Συστήματος

5.1. Η ταχύτητα του Συστήματος είναι:

5.1.1. Η ταχύτητα του Συστήματος είναι: ικανοποιητική πολύ αργή

1 2 3 4 5 6 7 8 9 NA

5.2. Το Σύστημα είναι σταθερό:

5.2.1. Το Σύστημα είναι σταθερό: ποτέ πάντα

1 2 3 4 5 6 7 8 9 NA

5.2.2. Χειρισμοί – Λειτουργίες αναξιόπιστα αξιόπιστα

1 2 3 4 5 6 7 8 9 NA

5.3. Η ευκολία χειρισμού εξαρτάται από την εμπειρία του χρήστη:

5.3.1. Η ευκολία χειρισμού εξαρτάται από την εμπειρία του χρήστη:

ποτέ πάντα

1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλιά σας για τις δυνατότητες του Συστήματος εδώ:

____
____
____

ΜΕΡΟΣ 6: Πολυμέσα

6.1. Η ποιότητα των εικόνων/ φωτογραφιών είναι:

κακή καλή

1 2 3 4 5 6 7 8 9 NA

6.1.1.Εικόνες/ Φωτογραφίες:

θολές καθαρές

1 2 3 4 5 6 7 8 9 NA

6.1.2. Η φωτεινότητα της εικόνας/ φωτογραφίας:

σκοτεινή φωτεινή

1 2 3 4 5 6 7 8 9 NA

6.2. Η ηχητική απόδοση είναι:

6.2.1. Η ηχητική απόδοση είναι:

ασταθής εύρυθμη

1 2 3 4 5 6 7 8 9 NA

6.2.2. Η ηχητική απόδοση είναι:

αλλοιωμένη ξεκάθαρη

1 2 3 4 5 6 7 8 9 NA

6.3. Τα χρώματα που χρησιμοποιούνται είναι:

αφύσικα φυσικά

1 2 3 4 5 6 7 8 9 NA

6.3.1. Η ποσότητα των χρωμάτων που είναι διαθέσιμη είναι :

ανεπαρκής επαρκής

1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλιά σας για τα πολυμέσα εδώ:

____

____

____

ΜΕΡΟΣ 7: Εγκατάσταση συστήματος

*Σημείωση: Το παιχνίδι μας, δεν διαθέτει εγκατάσταση.

6.2 Αποτελέσματα

Τα αποτελέσματα που προέκυψαν συνέβαλαν στις τελικές ρυθμίσεις του παιχνιδιού, αλλά και στην εξαγωγή χρησίμων συμπερασμάτων για τις μελλοντικές επεκτάσεις του. Παρακάτω παραθέτουμε συνοπτικά τα αποτελέσματα του ερωτηματολογίου για κάθε ενότητα.

ΜΕΡΟΣ 1: Γενική εντύπωση του χρήστη

Ερώτηση	Μέσος όρος βαθμολογίας
1.1	8.5
1.2	7
1.3	9
1.4	8
1.5	7.5

ΜΕΡΟΣ 2: Οθόνη

Ερώτηση	Μέσος όρος βαθμολογίας
2.1.1	8.5
2.1.2	9
2.1.3	9
2.2.1	6
2.2.2	7
2.2.3	9

ΜΕΡΟΣ 3: Ορολογία και επικοινωνία με το Σύστημα

Ερώτηση	Μέσος όρος βαθμολογίας
3.1.1	8.5
3.2.1	9
3.3.1	-
3.3.2	9
3.3.3	-
3.3.4	8
3.4.1	-

ΜΕΡΟΣ 4: Εκμάθηση χρήσης

Ερώτηση	Μέσος όρος βαθμολογίας
4.1.1	5

4.1.2	6.5
4.2.1	8
4.3.1	9
4.3.2	9
4.3.3	-

ΜΕΡΟΣ 5: Δυνατότητες του Συστήματος

Ερώτηση	Μέσος όρος βαθμολογίας
5.1.1	8
5.2.1	9
5.2.2	8
5.3.1	7.5

ΜΕΡΟΣ 6: Πολυμέσα

Ερώτηση	Μέσος όρος βαθμολογίας
6.1	9
6.1.1	9
6.1.2	7
6.2.1	7
6.2.1	7
6.3	9
6.3.1	9

*Σημείωση: Επειδή η εφαρμογή μας είναι παιχνίδι κάποιες από τις ερωτήσεις δεν ήταν δυνατόν να απαντηθούν

6.3 Συμπεράσματα

Τα συμπεράσματα από το ερωτηματολόγιο θα αναλυθούν ανά μέρη και βάσει των εντυπώσεων των χρηστών.

ΜΕΡΟΣ 1: Γενική εντύπωση του χρήστη

Όλοι οι χρήστες έμειναν ικανοποιημένοι από την συνολική αντίδραση του συστήματος. Το παιχνίδι προσφέρει ευχάριστο και κατανοητό περιβάλλον και αποτελεί μία ολοκληρωμένη εφαρμογή για τον τελικό χρήστη. Δυσκολία αντιμετώπισαν οι αρχάριοι, κάτι το οποίο είναι φυσιολογικό.

ΜΕΡΟΣ 2: Οθόνη

Ο σχεδιασμός της οθόνης παρέχει πολλές πληροφορίες για την καθοδήγηση του χρήστη. Η αλληλουχία όμως των οθονών φάνηκε να τους μπερδεύει αν και όταν εξοικειώθηκαν με το σύστημα η λειτουργία των μενού έγινε ξεκάθαρη.

ΜΕΡΟΣ 3: Ορολογία και επικοινωνία με το Σύστημα

Τα μηνύματα που εμφανίζονται στην οθόνη έχουν συνέπεια και σαφήνεια. Οι ονομασίες όλων των κουμπιών υποδηλώνουν ξεκάθαρα την λειτουργία τους. Καθυστερήσεις στις μεταβιβάσεις μεταξύ των μενού δεν υπάρχουν, όμως ανάλογα με τις επιδόσεις του υπολογιστή επηρεάζεται η απόδοση του παιχνιδιού κατά την διάρκεια του αγώνα.

ΜΕΡΟΣ 4: Εκμάθηση χρήσης

Για την εκμάθηση χρήσης του συστήματος η βαθμολογία ήταν 5. Αυτό οφείλεται στο γεγονός ότι οι χρήστες που δεν είχαν εμπειρία σε ηλεκτρονικά παιχνίδια δυσκολεύτηκαν αρχικά για να μάθουν τον χειρισμό του αυτοκινήτου. Κατά τα άλλα, τα βήματα για την ολοκλήρωση μιας διαδικασίας είναι απλά και κατανοητά.

ΜΕΡΟΣ 5: Δυνατότητες του Συστήματος

Το σύστημα αποδείχτηκε σταθερότατο κάτι που οφείλετε στην δομή της Unity. Επίσης ευκολία στο χειρισμό εξαρτάται κατά κύριο λόγο στην εμπειρία του χρήστη σε αντίστοιχα παιχνίδια. Οι αρχάριοι, δυσκολεύτηκαν να καταλάβουν την χρησιμότητα των μενού και τον χειρισμό του αυτοκινήτου, ενώ οι υπόλοιποι προσαρμόστηκαν γρήγορα.

ΜΕΡΟΣ 6: Πολυμέσα

Η ποιότητα των εικόνων άφησε άκρως θετικές εντυπώσεις καθώς δημιουργήθηκαν με την χρήση προ επεξεργασμένων γραφικών εφάμιλλα των κινηματογραφικών ταινιών. Επίσης όλες οι φωτογραφίες δέχτηκαν επιπλέον επεξεργασία με ειδικό πρόγραμμα έτσι ώστε η φωτεινότητα και η αντίθεση να είναι στα σωστά επίπεδα. Τα ηχητικά εφέ αλλά και η μουσική χαρακτηρίστηκαν ως κατάλληλες για το παιχνίδι. Αντίθετα η ηχητική απόδοση χαρακτηρίστηκε ως ελαφρώς αλλοιωμένη όταν γινόταν υπέρθεση των εφέ με την μουσική.

Κεφάλαιο 7

Αποτελέσματα και μελλοντικές επεκτάσεις

7.1 Στόχος και αποτελέσματα

Στόχος της παρούσας διπλωματικής ήταν η δημιουργία ενός τρισδιάστατου παιχνιδιού για υπολογιστές με αυτόνομους αντιπάλους. Στα πλαίσια αυτά αναπτύξαμε διάφορα υποσυστήματα για μπορέσει το παιχνίδι να σταθεί ως μια αυτόνομη και πλήρης εφαρμογή.

Αρχικά για να κινηθεί το αυτοκίνητο χρειάστηκε να μελετήσουμε τη βασική φυσική και μηχανολογία τους. Χρησιμοποιώντας πληροφορίες από το βιβλίο Start your engines καταφέραμε να φτιάξουμε ένα αποδεκτό μοντέλο συμπεριφοράς για τα αυτοκίνητα μας. Στην συνέχεια, οι αντίπαλοι μας θα έπρεπε να έχουν μια έξυπνη αλλά και απρόβλεπτη συμπεριφορά έτσι ώστε να υπάρχει ένα επίπεδο πρόκλησης για τον χρήστη. Επειδή τα αυτοκίνητα κινούνται συνέχεια υλοποιήσαμε έναν αλγόριθμο Line Of Sight έτσι ώστε να μην επιβαρύνουμε πολύ το σύστημα μας. Με αυτόν τρόπο ένας αντίπαλος προσεγγίζει κάποιον άλλο πολύ γρήγορα όμως δεν μπορεί να αποφύγει τα όποια εμπόδια έχει η πίστα. Για να λύσουμε το πρόβλημα, αναπτύξαμε ένα αλγόριθμο Obstacle avoidance έτσι ώστε τα αυτοκίνητα να κινούνται ανεμπόδιστα σε ολόκληρο το χώρο της πίστας. Επιπλέον επειδή θέλαμε αυτόνομους, αντίπαλους χρησιμοποιήσαμε Fuzzy Logic για να τους δώσουμε την δυνατότητα να εκτιμούν μια κατάσταση και να παίρνουν αποφάσεις τις οποίες εμείς δε μπορούμε να προβλέψουμε. Επίσης υλοποιήσαμε μια πρωτότυπη λειτουργία για τα παιχνίδια βολών αυτοκινήτου. Κάθε 30 δευτερόλεπτα εμφανίζεται στην πίστα μια σημαία . Όποιο αυτοκίνητο βρίσκεται στην μεγαλύτερη ευκλείδεια απόσταση την στιγμή που κάποιος πάρει την σημαία τότε φεύγει από το παιχνίδι. Αυτό σημαίνει ότι οι αντίπαλοι μας θα έπρεπε να είναι σε θέση να φτάσουν την σημαία όσο πιο γρήγορα γίνεται. Ο καταλληλότερος αλγόριθμος για την συγκεκριμένη προδιαγραφή είναι ο A*. Κάθε φορά που εμφανίζεται η σημαία όλα τα αυτοκίνητα υπολογίζουν το συντομότερο μονοπάτι προς αυτή. Αν και ο A* είναι ένας γρήγορος αλγόριθμος, η κλασσική του δομή μπορεί να βελτιωθεί ακόμα περισσότερο. Εμείς επιλέξαμε να αλλάξουμε την δομή δεδομένων της ανοιχτής λίστας, από απλή λίστα σε ουρά προτεραιότητας έτσι ώστε να επιταχύνουμε την αναζήτηση του κόμβου με το μικρότερο κόστος στο πρώτο βήμα του αλγορίθμου.

Εκτός από την πλευρά των αλγορίθμων δημιουργήθηκαν πολλά μενού έτσι ώστε η γραφική διεπαφή να είναι απλή και ευχάριστη. Όλα τα κομμάτια του παιχνιδιού αναπτύχθηκαν από το μηδέν εκτός από το αυτοκίνητο, κάποια γεωμετρία μέσα στην πίστα και τις υφές που είναι από τρίτους δημιουργούς. Τέλος, γνωρίζοντας ότι ένα παιχνίδι που σέβεται τον εαυτό του, χρειάζεται ένα εισαγωγικό video, δημιουργήσαμε ένα animation που παρουσιάζει κάποια αυτοκίνητα με προ επεξεργασμένα γραφικά και διάφορες εικόνες από την ανάπτυξη του παιχνιδιού.

7.2 Μελλοντικές επεκτάσεις και βελτιώσεις

Η εφαρμογή μας μπορεί να βελτιωθεί μελλοντικά κυρίως αυξάνοντας την διάρκεια του παιχνιδιού. Δηλαδή να δημιουργηθούν πολλές πίστες και ένα τουρνουά έτσι ώστε να υπάρχει ενδιαφέρον και ανταγωνισμός. Επίσης τα αυτοκίνητα θα μπορούσαν να έχουν διαφορετικές επιδόσεις και ειδικά όπλα. Όσον αφορά την τεχνητή νοημοσύνη το Fuzzy Logic μπορεί να βελτιωθεί προσθέτοντας και άλλους κανόνες και καταστάσεις. Μια εναλλακτική λύση είναι να χρησιμοποιήσουμε για σύστημα αποφάσεων Bayesian Network βάζοντας έτσι μια πιο επιστημονική μέθοδο που χρησιμοποιεί πιθανότητες. Επίσης στον A* να υλοποιήσουμε την ουρά προτεραιότητας με σωρό οποίος να αντί για δυαδικό δέντρο να είναι Fibonacci heap χαμηλώνοντας ακόμα πιο πολύ τον χρόνο εκτέλεσης του αλγορίθμου.

Το Unity API μας δίνει την δυνατότητα να φτιάξουμε εργαλεία που θα μπορούν να ξαναχρησιμοποιηθούν για την πιο γρήγορη ανάπτυξη παιχνιδιών. Για παράδειγμα, πολύ αποδοτικό θα ήταν ένα σύστημα για την παραγωγή ελέγχου διαδρομής και τη διαχείριση τους. Δηλαδή οι κόμβοι του A* να ρυθμίζονται μέσω γραφικής διεπαφής και όχι άμεσα από κώδικα. Τέλος η σύνδεση πολλών χρηστών μέσω δικτύου και η δυνατότητα να αναλαμβάνουν αποστολές είτε ως σύμμαχοι, είτε ως αντίπαλοι θα έκανε ακόμα πιο ενδιαφέρον το παιχνίδι.

7.3 Επίλογος

Ένα εφηβικό όνειρο πραγματοποιήθηκε. Η διαδικασία ήταν χρονοβόρα, κουραστική αλλά το αποτέλεσμα συναρπαστικό. Χρειάστηκε να γίνει έρευνα σε πολλούς τομείς της επιστήμης των γραφικών και να αποκτηθούν γνώσεις χειρισμού πολλών επαγγελματικών προγραμμάτων. Τεχνητή νοημοσύνη, θεωρία ασαφούς λογικής, εύρεση καλύτερου μονοπατιού, είναι μερικά μόνο από τα θέματα που πραγματεύτηκε αυτή η διπλωματική. Προγράμματα όπως το Maya, Photoshop, After Effects που χρησιμοποιούνται για την παραγωγή παιχνιδιών και ταινιών από μεγάλα studio χρησιμοποιήθηκαν και εδώ. Η πολύπλευρη φύση της εργασίας έδωσε την δυνατότητα για συνδυασμό και εφαρμογή όλων όσων αποκτήθηκαν κατά την διάρκεια των σπουδών αλλά και πρόσφερε σημαντική εμπειρία.

Η βιομηχανία παιχνιδιών και ταινιών έχει επιστημονικό και καλλιτεχνικό ενδιαφέρον. Καλλιτέχνες και μηχανικοί μπορούν να συνεργαστούν και να πετύχουν εντυπωσιακά αποτελέσματα. Το μόνο σίγουρο, είναι ότι υπάρχει πολύ μέλλον για τα γραφικά και ο δρόμος είναι ανοιχτός.

Παράρτημα Α

Αντίστοιχα εμπορικά παιχνίδια

Η εφαρμογή που υλοποιήσαμε δεν είναι η μοναδική στην κατηγορία της. Κατά καιρούς έχουν κυκλοφορήσει πολλά παιχνίδια με μάχες αυτοκινήτων με ιδιαίτερη επιτυχία, εδώ θα αποτίσουμε φόρο τιμής σε δύο από τα πιο ιστορικά.

Carmageddon II: Carpocalypse Now

Το πρώτο παιχνίδι είναι το Carmageddon II: Carpocalypse Now**.** Κυκλοφόρησε στις 30 Νοεμβρίου του 1998 από το βρετανικό studio, Stainless Games [7][8]. Αποτελεί συνέχεια του παιχνιδιού Carmageddon το οποίο είχε κυκλοφορήσει ένα χρόνο πριν. Ο χρήστης ελέγχει ένα αυτοκίνητο ειδικά τροποποιημένο για συγκρούσεις και αγωνίζεται ενάντια σε ένα σύνολο αντιπάλων σε διάφορες τοποθεσίες όπως μέσα σε πόλεις, ορυχεία, βιομηχανικές περιοχές.

Υπάρχουν τρείς τρόποι για να κερδίσουμε:

- Ολοκληρώνοντας τον αγώνα, δηλαδή περνώντας από όλα τα checkpoints πριν τελειώσει ο χρόνος.
- Καταστρέφοντας όλους τους αντιπάλους.
- Σκοτώνοντας όλους τους ανθρώπους/ζόμπι στο επίπεδο [7][8].

Για την ανάπτυξη χρησιμοποιήθηκε ένα πολύ ρεαλιστικό και καινοτόμο μοντέλο φυσικής που ήταν μάλιστα από τα πρώτα παιχνίδια που χρησιμοποιούσαν παραμορφώσημες επιφάνειες για το μοντέλο ζημιών των αυτοκινήτων. Το παιχνίδι προκάλεσε ποικίλες αντιδράσεις λόγω των έντονων σκηνών βίας με αποτέλεσμα σε κάποιες χώρες να κυκλοφορήσει με ρομπότ αντί για ανθρώπους με πράσινο αντί για κόκκινο αίμα ή ακόμα και να απαγορευτεί.

Twisted Metal III

Το δεύτερο παιχνίδι είναι το Twisted Metal III. Κυκλοφόρησε στις 31 Οκτωβρίου του 1998 από την 989 Studios η οποία αργότερα συγχωνεύτηκε με την Sony Computer Entertainment of America [21][22].

Το Twisted Metal III είναι ένα παιχνίδι βολών αυτοκινήτου. Ο χρήστης μπορεί να διαλέξει ανάμεσα σε δώδεκα αυτοκίνητα κάθε ένα με το δικό του “ειδικό” όπλο. Υπάρχουν επιπλέον διαθέσιμα πολλά είδη όπλων όπως πολυβόλα, πύραυλοι και βόμβες διάσπαρτα μέσα στις πίστες. Το τουρνουά αποτελείται από οκτώ επίπεδα και στόχος είναι πάντα η εξόντωση των αντιπάλων [21][22].

Το παιχνίδι δέχτηκε πολύ αρνητικές κριτικές κυρίως για το μη-ρεαλιστικό μοντέλο φυσικής του και για τις πρόχειρα σχεδιασμένες πίστες. Παρ’ όλα αυτά έκανε μεγάλη εμπορική επιτυχία με αποτέλεσμα να συνεχίζεται η κυκλοφορία της σειράς μέχρι και σήμερα.

Αναφορές

[1] *A Vortex Particle Method for Smoke, Water, and Explosions*. Ανάκτηση από Stanford: http://graphics.stanford.edu/papers/vortex_particle-sig05/vortex_particles_smoke.jpg

[2] *A* search algorithm*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/A*

[3] *Animation*. Ανάκτηση από Wikipedia: http://el.wikipedia.org/wiki/Animation

[4] Bourg, D. M., & Seeman, G. (2004). *AI for Game Developers.*

[5] Buclkland, M. (2005). *Programming Game AI by Example.*

[6] *Bump mapping*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Bump_mapping

[7] *Carmageddon*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Carmageddon

[8] *Carmageddon_II:_Carpocalypse_Now*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Carmageddon_II:_Carpocalypse_Now

[9] *Comparison of OpenGL and Direct3D*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Comparison_of_OpenGL_and_Direct3D

[10] Ernest Adams, R. A. (2006). *Fundamentals of Game Design.* Prentice Hall.

[11] *figures*. Ανάκτηση από The George Washington University: http://www.seas.gwu.edu/~simhaweb/cs153/lectures/module2/figures/fig6.jpg

[12] *figures*. Ανάκτηση από The George Washington University: http://www.seas.gwu.edu/~simhaweb/cs153/lectures/module2/figures/fig1.jpg

[13] *Game engine*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Game_engine

[14] *Microsoft Direct3D*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Direct3D

[15] *Min-Max Heaps*. Ανάκτηση από University Of Florida: http://www.cise.ufl.edu/~mssz/DatStrucAlg/Min-MaxHeaps.gif

[16] *OpenGL*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/OpenGL

[17] Parker, J. (2005). *Start Your Engines: Developing Driving and Racing Games.* Paraglyph Press.

[18] Sedgewick, R. *Αλγόριθμοι σε Java.*

[19] *Shaders*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Shaders

[20] *Twisted_metal*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Twisted_metal

[21] *Twisted_Metal_3*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Twisted_Metal_3

[22] *Unity (game engine*) reference manual. Ανάκτηση από Unity3D: http://unity3d.com/support/documentation/Components/index.html

[23] *Unreal Engine*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Unreal_Engine

[24] Μελέτιος, Σ. (2009). *Ανάπτυξη τρισδιάστου παιχνιδιού με χρήση μηχανής φυσικής*. Χανιά.

Προγράμματα

Μηχανή γραφικών: Unity 3D 2.5, http://unity3d.com/

Μοντελοποίηση: Autodesk Maya 2009, Autodesk 3D Studio Max 2009, http://usa.autodesk.com/

Animation, Rendering: Autodesk Maya 2009, http://usa.autodesk.com/

Επεξεργασία υφών και εικόνων: Adobe Photoshop CS4, http://www.adobe.com/

Επιπλέον εφέ στο εισαγωγικό βίντεο: Adobe After Effects CS4, http://www.adobe.com/

Μοντάζ για το εισαγωγικό βίντεο: CyberLink Power Director 8, http://www.cyberlink.com/

Forums

http://forum.unity3d.com/

http://madseeds.com/forum/