ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Πτυχιακή Εργασία

Ανάπτυξη multiplayer παιχνιδιού στην μηχανή γραφικών Unity.

Ιωάννης-Αλκίνοος Μοιράγιας-Μποφίλιος
Α.Μ: Ε/10100

Πειραιάς 2016

Επιβλέπων καθηγητής
Καθηγητής Νικήτας -Μαρίνος Σγούρος

Περιεχόμενα

1. ΕΙΣΑΓΩΓΗ 1

1.1 Ιστορία Βιντεοπαιχνιδιών----------1

1.2 Παρουσίαση παιχνιδιού ----------3

1.2.1 Χειρισμός παιχνιδιού----------3

1.2.2 Σκηνές παιχνιδιού ----------3

1.3 Βήματα Υλοποίησης Παιχνιδιού----------4

1.4 Περίληψη Προγραμμάτων ----------5

1.4.1 Autodesk Maya ----------6

1.4.2 Zbrush ----------6

1.4.3 Adobe Photoshop ----------6

1.4.4 Unity ----------6

1.4.5 Visual Studio 2015 ----------6

2. ΔΗΜΙΟΥΡΓΙΑ ΣΤΟΙΧΕΙΩΝ ΠΑΙΧΝΙΔΙΟΥ (GAME ASSETS) 7

2.1 Εισαγωγή----------7

2.2 Βήματα δημιουργίας μοντέλου ----------7

2.2.1 Δημιουργία πλέγματος(mesh)----------7

2.2.2 Δημιουργία UV χάρτη(UV-mapping) ----------7

2.2.3 Δημιουργία υφών(texture maps)----------8

2.3 Διαδικασία μοντελοποίησης ----------8

2.3.1 Box modelling ----------8

2.3.2 3-D Sculpting ----------8

2.4 Διαδικασία UV-mapping ----------10

2.4.1 Μέγιστη εκμετάλλευση των pixels του UV-MAP----------10

2.4.2 Επιλογή των σημείων χωρισμού του UV-map ----------11

2.4.3 Δημιουργία πιο ομαλού UV-map----------11

2.5 Δημιουργία υφών(texturing) ----------11

2.5.1 Albedo Map----------12

2.5.2 Normal Map ----------12

2.5.3 Ambient Occlusion Map ----------13

2.5.4 Emission Map ----------14
5.5.5 Transparency Map ----------15
2.6 Animation ----------16
2.6.1. Δημιουργία skeleton ----------16
2.6.2 Δημιουργία καρέ-κλειδιών----------18
3. ΣΧΕΔΙΑΣΜΟΣ ΠΑΙΧΝΙΔΙΟΥ 19
3.1 Περιβάλλον εργασίας Unity ----------19
3.2 Δημιουργία πίστας----------19
3.2.1 Δημιουργία σχηματισμού εδάφους πίστας (terrain) ----------19
3.2.2 Εμπλουτισμός πίστας----------21
3.2.4 Δημιουργία Skybox ----------21
3.2.5 Δημιουργία ωκεανού και ποταμιού ----------22
3.2.6 Δημιουργία Prefabs ----------22
3.3 Πρόσθεση εξαρτημάτων στα μοντέλα(components) ----------23
3.3.1 Material----------23
3.3.2 Collider ----------24
3.3.3 Rigidbody ----------25
3.3.4 Camera ----------25
3.3.5 Reflection Probe----------25
3.3.6 Particle System ----------26
3.3.7 Trail renderer ----------26
3.4 Κωδικοποίηση παιχνιδιού(scripting) ----------27
3.5 Σύστημα φυσικής αυτοκινήτου----------29
3.6 Σύστημα Animation Οδηγού----------30
3.7 Δημιουργία GUI παιχνιδιού----------32
3.8 Δημιουργία Game Logic.----------34
3.8.1 Προεπισκόπηση πίστας και αρχή αγώνα----------34
3.8.2 Σύστημα ελέγχου δίκαιου αγώνα ----------35
3.9 Ήχοι και μουσική----------37
3.9.1 Audio Source και Audio Listener ----------37
3.9.2 Μουσική ----------38
3.9.3 Ήχος μηχανής αυτοκινήτου----------38
3.10 Δημιουργία Multiplayer----------39

3.10.1 Βασικές Έννοιες ----------39
3.10.2 Εισαγωγή βασικών multiplayer-components. ----------39
3.10.3 Συγχρονισμός μεταβλητών ----------40
3.10.4 Αρχικοποίηση και ταυτοποίηση παίκτη ----------41
3.10.5 Interpolation----------42
3.11 Βελτίωση Framerate ----------43
3.11.1 Profiler ----------44
3.11.2 Occlusion Culling----------45
3.12 Debugging ----------45

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας διπλωματικής είναι να αναλυθεί βήμα προς βήμα η διαδικασία δημιουργίας ενός βιντεοπαιχνιδιού από το στάδιο της προτυποποίησης μέχρι την τελική του υλοποίηση. Στα πλαίσια της εργασίας αναπτύχθηκε επίσης ένα τριδιάστατο multiplayer racing παιχνίδι, του οποίου εικόνες χρησιμοποιούνται στο κείμενο για την καλύτερη εξήγηση των σταδίων ανάπτυξης ενός παιχνιδιού. Η εργασία χωρίζεται σε τρία τμήματα.
Στο πρώτο μέρος γίνεται μια σύντομη ιστορική αναδρομή στην ιστορία των βιντεοπαιχνιδιών και πώς αυτά έχουν αναπτυχτεί με τα χρόνια. Επίσης γίνεται μια παρουσίαση του παιχνιδιού καθώς και μια σύντομη περιγραφή των προγραμμάτων που χρησιμοποιήθηκαν για την υλοποίησή του.
Στο δεύτερο μέρος αναλύεται το πώς υλοποιείται το καλλιτεχνικό κομμάτι ενός βιντεοπαιχνιδιού δηλαδή η δημιουργία τριδιάστατων μοντέλων και textures. Για τα μοντέλα γίνεται επεξήγηση των διαφόρων τεχνικών που χρησιμοποιούνται σήμερα για την δημιουργία τους και τα θετικά ή αρνητικά της κάθε μεθόδου. Για τα textures γίνεται ανάλυση του κάθε είδους texture και πώς αυτά δημιουργούνται με την τεχνική uv-mapping. Η ενότητα κλείνει με το θέμα του animation των μοντέλων, δηλαδή πώς δημιουργείται αληθοφανής κίνηση σε ένα μοντέλο.
Στο τρίτο και τελευταίο τμήμα της εργασίας παρουσιάζεται η υλοποίηση του παιχνιδιού στο εργαλείο Unity. Γίνεται εκτενής ανάλυση στο πώς χρησιμοποιείται η Unity, και πώς εκμεταλλεύεται κάποιος τα εργαλεία και τις διεργασίες που του παρέχει. Τέλος εμβαθύνει στο πώς χρησιμοποιείται η γλώσσα προγραμματισμού C# για να αποδώσει λειτουργικότητα στο παιχνίδι, αλλά και για να λυθούν διάφορα προβλήματα που εμφανίστηκαν στην πορεία δημιουργίας του.

ABSTRACT

The aim of this project is to analyse step by step the creation of a video game, from prototype phase until its creation. The project also includes the design of a 3-D multiplayer racing game, whose pictures are used throughout this text, to better explain the phases of design during video game development. The text is split into three parts.
The first part contains a brief history of video games and how they have developed over the years. A small presentation of the game and the programs used to make it is also done.
The second part analyses the artistic parts of a video game like how 3-D models and textures are made. For 3-D models I briefly review the different techniques which are used to make them, as well as the positive and negative aspects of each technique. For textures I analyse the different kind of textures and how they are created with the uv-mapping technique. This section ends with the topic of model-animation, namely, how to give life to an object.
The third and last section of this text is where the game comes together in the Unity game engine. I do an in depth analysis on how Unity is used and how someone can utilise the tools and processes that Unity provides. Finally I go in depth on how you can use C# programming language to make the game come to life and to solve various problems which arose during the design process.

1. ΕΙΣΑΓΩΓΗ

1.1 Ιστορία Βιντεοπαιχνιδιών

Το πρώτο βιντεοπαιχνίδι θεωρείται το <<Tennis For Two>> το οποίο δημιουργήθηκε το 1958 από τον Αμερικάνο φυσικό William Higinbotham για την ετήσια έκθεση του εθνικού εργαστηρίου του Brookhaven. Το παιχνίδι είχε προβληθεί στην οθόνη ενός παλμοσκοπίου(oscilloscope) και αναπαριστούσε ένα παιχνίδι τέννις το οποίο απεικονιζόταν από τα πλάγια. Πρόκειται για ένα παιχνίδι για δύο παίκτες που ο καθένας από αυτούς έλεγχε την γωνία και το πότε θα χτυπήσει την μπάλα με ένα τηλεχειριστήριο. Παρόλο που τα γραφικά του ήταν πολύ απλοϊκά, είχε τεράστια επιτυχία στην έκθεση και απέδειξε την αγάπη που έχει ο κόσμος για τον συγκεκριμένο τρόπο ψυχαγωγίας. Η στιγμή όπου τα παιχνίδια εδραιώθηκαν στο ευρύ κοινό παγκοσμίως, ήταν την δεκαετία 70’-80’ με την εμφάνιση των arcades αλλά και των πρώτων οικιακών κονσόλων.

Εικόνα 1.1 : Το παιχνίδι Tennis For Two. Πηγή: wikipedia.org

Σήμερα τα βιντεοπαιχνίδια έχουν εξελιχτεί σε μια από τις μεγαλύτερες βιομηχανίες διασκέδασης μαζί με τον κινηματογράφο και την μουσική. Με το πέρασμα των χρόνων και την εξέλιξη της τεχνολογίας καταφέρνουν να αναπαριστούν όλο και πιο πολύπλοκα και εντυπωσιακά γραφικά. Χαρακτηριστικό παράδειγμα αποτελεί η πέμπτη γενιά κονσόλων η οποία επέτρεψε την εύκολη αναπαράσταση τρισδιάστατων γραφικών. Μέχρι τότε οι κονσόλες έπαιζαν κυρίως παιχνίδια δισδιάστατων γραφικών εξαιτίας των περιορισμών τους λόγω hardware. Αν και τα πρώτα τρισδιάστατα γραφικά ήταν αρκετά <<τετραγωνισμένα>>, με την βελτίωση του hardware σήμερα, μπορούμε να αναπαραστίσουμε αρκετά ρεαλιστικά σκηνικά και μοντέλα στις οθόνες μας.

Εικόνα 1.2: Η τεχνολογική εξέλιξη της ηρωίδας των παιχνιδιών Tomb Raider από το 1996 μέχρι το 2015. Πηγή: Google Images

Αξίζει να σημειωθεί ότι εξελίσσεται συνεχώς ολόκληρη η εμπειρία των βιντεοπαιχνιδιών και όχι μόνο τα γραφικά. Τα τελευταία δύο χρόνια για παράδειγμα έχουμε δεί την εμφάνιση των πρώτων virtual reality headsets. Πρόκειτε για κράνη που φοράει στο κεφάλι του ο χρήστης τα οποία περιέχουν τριγύρω μια εννιαία οθόνη, η οποία καλύπτει ολόκληρο το οπτικό πεδίο του χρήστη. Με αυτόν τον τρόπο μπορεί να προσομοιώνει ολόκληρους τεχνητούς κόσμους και να μπερδεύει τον χρήστη ότι όντως βρίσκεται μέσα σε αυτόν, γιατί δεν καλύπτει περιορισμένο κομμάτι του οπτικού πεδίου όπως μια τηλεόραση.
Μια ακόμα σημαντική εξέλιξη ήταν ότι με την εδραίωση του ίντερνετ τα παιχνίδια απέκτησαν περισσότερα κοινωνικά χαρακτηριστικά. Πλέον μπορεί κάποιος να παίζει από το σπίτι του ένα παιχνίδι με αντιπάλους από όλο τον κόσμο χωρίς να χρειάζεται να βρίσκεται στο ίδιο δωμάτιο με αυτούς ή να μοιράζεται μια τηλεόραση.

Εικόνα 1.3: Το VR headset της Sony. Πηγή: playstation.com

1.2 Παρουσίαση παιχνιδιού

Το παιχνίδι που δημιουργήθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας είναι ένα τριδιάστατο παιχνίδι αγώνων αυτοκινήτων που τρέχουν σε μια ανώμαλη πίστα ενός εξωτικού νησιού. Περιέχει δύο game modes. Ένα singleplayer στο οποίο ο παίκτης προσπαθεί να κάνει τον γύρο της πίστας σε όσο το δυνατόν καλύτερο χρόνο μπορεί, και ένα multiplayer mode στο οποίο μπορεί να τρέξει έναν αγώνα εναντίον ενός φίλου του μέσω δικτύου. Το παιχνίδι προσπαθεί να αποδώσει μια μίξη ρεαλιστικών αλλά και arcade στοιχείων. Ενώ το περιβάλλον της πίστας και τα γραφικά του παιχνιδιού όπως το νερό και η εμφάνιση των αυτοκινήτων είναι αρκετά ρεαλιστικά, το gameplay του παιχνιδιού όπως η κίνηση και οι συγκρούσεις των αυτοκινήτων είναι πιο ψεύτικη και θυμίζουν συγκρουόμενα στο λούνα πάρκ. Σκοπός του παίκτη είναι να τερματίσει πρώτος ακόμα και με αθέμιτα μέσα όπως για παράδειγμα να ρίξει τον αντίπαλό του έξω από την πίστα. Όταν ένα αυτοκίνητο βγεί από την πίστα του παιχνιδιού ή πέσει στο νερό καταστρέφεται και ξαναεμφανίζεται σε μια περιοχή δίπλα εκεί που καταστράφηκε για να συνεχίσει τον αγώνα.

1.2.1 Χειρισμός παιχνιδιού

Για την χρήση του βασικού μενού του παιχνιδιού ο χρήστης χρησιμοποιεί το ποντίκι για να κάνει την επιλογή του.
Κατά την διάρκεια ενός αγώνα τα κουμπιά που μπορεί να πατήσει ο παίκτης είναι τα εξής:

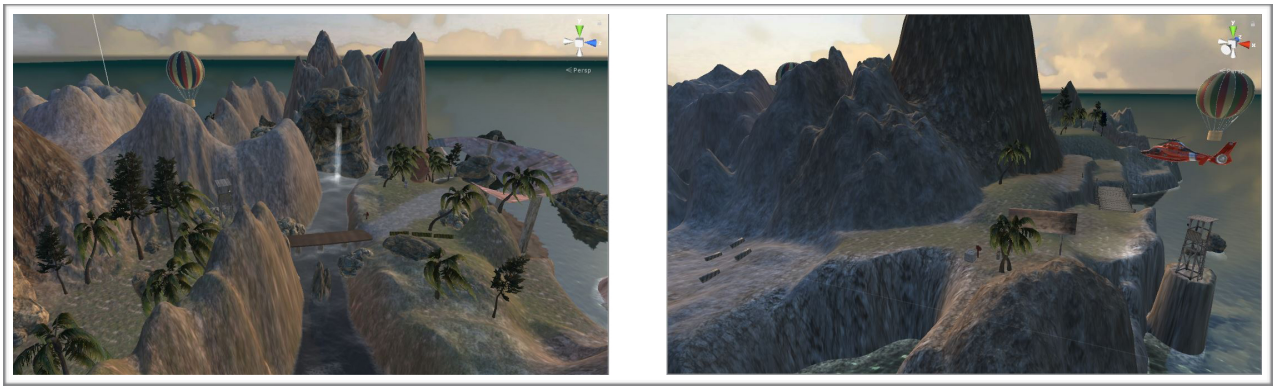
- a → Δήλωση του παίκτη ως <<έτοιμος>> ώστε να ξεκινήσει ο αγώνας
- s → Σταμάτημα του παιχνιδιού(pause game).
- esc → Επιστροφή στο μενού σε περίπτωση που το παιχνίδι είναι σταματημένο(έχει πατηθεί προηγουμένως το a).
- enter → Εναλλαγή κάμερας μεταξύ του εσωτερικού και του εξωτερικού του αυτοκινήτου.
- spacebar → Drifting του αυτοκινήτου
- d → Αυτοκαταστροφή του αυτοκινήτου
- Πάνω βελάκι → Γκάζι
- Κάτω βελάκι → Φρένο
- Αριστερό/Δεξί βελάκι → Στροφή τιμονιού αριστερά ή δεξιά αντίστοιχα.

1.2.2 Σκηνές παιχνιδιού

Την στιγμή που ο χρήστης ανοίγει την εφαρμογή ξεκινάει αυτόματα η πρώτη σκηνή του παιχνιδιού, το βασικό μενού. Σε αυτό μπορεί ο παίκτης να διαλέξει την εκδοχή του παιχνιδιού που θέλει να παίξει(singleplayer ή multiplayer) καθώς και την επιλογή help όπου δίνονται οδηγίες σχετικά με τα κουμπιά του παιχνιδιού. Τέλος υπάρχει η επιλογή exit που κλείνει την εφαρμογή.
Αν ο παίκτης κάνει κλίκ στην επιλογή singleplayer ή multiplayer τότε φορτώνεται η σκηνή της πίστας του αγώνα. Σε περίπτωση που διαλέξει την επιλογή singleplayer τότε ξεκινάει απευθείας η διαδικασία προεπισκόπησης της πίστας και το παιχνίδι περιμένει από τον παίκτη να πατήσει το κουμπί a για να ξεκινήσει. Στην περίπτωση του multiplayer πρέπει πρώτα ο ένας από τους δύο παίκτες να πατήσει το κουμπί Host και ο άλλος το κουμπί connect ώστε να συνδεθεί σε αυτόν που κάνει το host του παιχνιδιού. Στην συνέχεια το παιχνίδι θα περιμένει έως ότου και οι δύο παίκτες πατήσουν το κουμπί a έτσι ώστε να γίνει η έναρξη του αγώνα.
Η πίστα του παιχνιδιού αποτελείται από τέσσερα τμήματα. Το πρώτο είναι το τμήμα της παραλίας το οποίο περιέχει την γραμμή αφετηρίας. Στην συνέχεια είναι το τμήμα του δάσους όπου οι παίκτες θα αντιμετωπίσουν δέντρα-εμπόδια και απότομες στροφές.

Εικόνα 1.4: Η παραλία και το δάσος του νησιού.

Παρακάτω έρχονται αντιμέτωποι με το τρίτο μέρος τις “θανατηφόρες” γέφυρες τις οποίες πρέπει να προσέχουν για να μην πέσουν στο νερό. Το τελευταίο μέρος της διαδρομής είναι η περιοχή δίπλα στο ηφαίστειο στο οποίο οι παίκτες μπορούν να διαλέξουν είτε την χαμηλή διαδρομή, που περνάει δίπλα από την θάλασσα, ή την ψηλή, που περνάει δίπλα από το βουνό.



Εικόνα 1.5: Οι θανατηφόρες γέφυρες και το τελευταίο τμήμα της διαδρομής.

Στο multiplayer παιχνίδι οι 2 παίκτες πρέπει να τρέξουν έναν αγώνα τριών γύρων και αυτός που θα τερματίσει πρώτος είναι ο νικητής. Στην περίπτωση του singleplayer ο παίκτης μπορεί να κάνει όσες φορές θέλει τον γύρο της πίστας ώστε να βελτιώσει τον χρόνο του.

1.3 Βήματα Υλοποίησης Παιχνιδιού

Παρακάτω περιγράφεται σε συντομία η σειρά βημάτων που ακολούθησα για τον σχεδιασμό του παιχνιδιού που δημιουργήθηκε στα πλαίσια της παρούσας διπλωματικής.

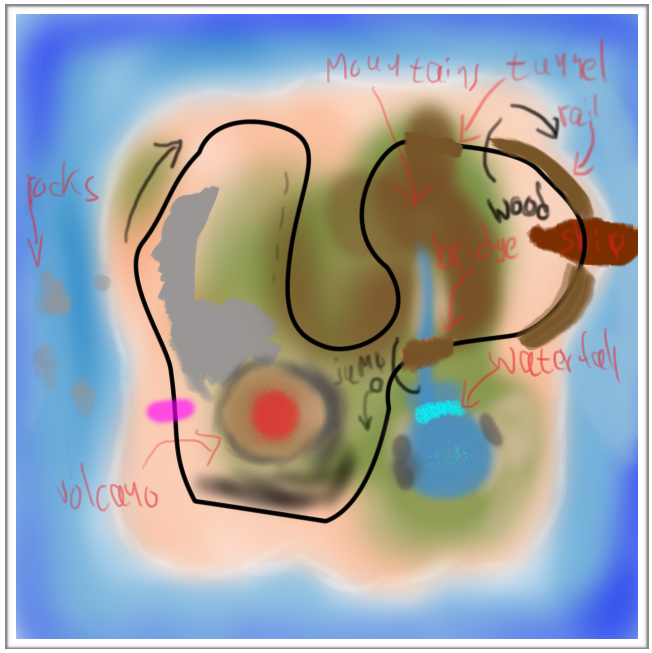
Στάδιο Ιδεών και Προτυποποίησης:

1. Δημιουργία ενός prototype του παιχνιδιού με placeholder art.

Αρχικά δημιουργείται μια πρώτη έκδοση του παιχνιδιού στην οποία δεν υπάρχουν τα κανονικά μοντέλα αλλά αντί για αυτά χρησιμοποιούνται κύβοι/σφαίρες/κύλινδροι που αναπαριστούν τα πραγματικά μοντέλα. Επίσης δημιουργούνται μερικά απλά scripts για να πάρει μια ιδέα ο σχεδιαστής πώς θα δουλεύει το παιχνίδι.

2. Αναζήτηση και δημιουργία concept και reference art.

Σε αυτό το στάδιο δημιουργήθηκαν τα σκίτσα του παιχνιδιού, τα οποία θα χρησιμοποιηθούν στην συνέχεια σαν αναφορά για την δημιουργία των τρισδιάστατων μοντέλων και της πίστας. Επίσης αυτό το στάδιο περιλαμβάνει την αναζήτηση από το διαδίκτυο ή από άλλες πηγές εικόνων και ιδεών οι οποίες μπορούνε επίσης να βοηθήσουν ως πηγή έμπνευσης ή αναφοράς.



Εικόνα 1.6: Πρώτο σκίτσο αναφοράς του νησιού που δημιουργήθηκε στο photoshop.

Στάδιο Σχεδιασμού

1. Δημιουργία των τρισδιάστατων μοντέλων του παιχνιδιού.

Μετατροπή των σχεδίων που έγιναν στο προηγούμενο στάδιο σε τρισδιάστατα μοντέλα με την χρήση του Maya και του Zbrush.

2. Δημιουργία των υφών(textures) που χρησιμοποιούνε τα μοντέλα.

Ανάλογα με τις οπτικές απαιτήσεις του κάθε μοντέλου δημιουργούνται τα κατάλληλα texture maps στο photoshop.

3. Σχεδιασμός των animations του οδηγού.

Εισαγωγή κίνησης στο μοντέλο του οδηγού και του τιμονιού με την χρήση του skeletal animation στο Maya.

Στάδιο Υλοποίησης στην Unity

1. Σχεδιασμός του terrain.

Σχεδιασμός του βασικού νησιού με την terrain engine της Unity.

2. Εισαγωγή μοντέλων στην πίστα.

Κατάλληλη τοποθέτηση του κάθε μοντέλου πάνω στην πίστα.

3. Προγραμματισμός των διαφόρων λειτουργιών.

Δημιουργία αλληλεπίδρασης στο παιχνίδι με την δημιουργία scripts σε γλώσσα C#.
Όλα τα παραπάνω βήματα θα αναλυθούνε σε βάθος στα επόμενα κεφάλαια.

1.4 Περίληψη Προγραμμάτων

Για την δημιουργία του τελικού παιχνιδιού χρησιμοποιήθηκαν διάφορα σχεδιαστικά και προγραμματιστηκά εργαλεία τα οποία περιγράφονται παρακάτω:

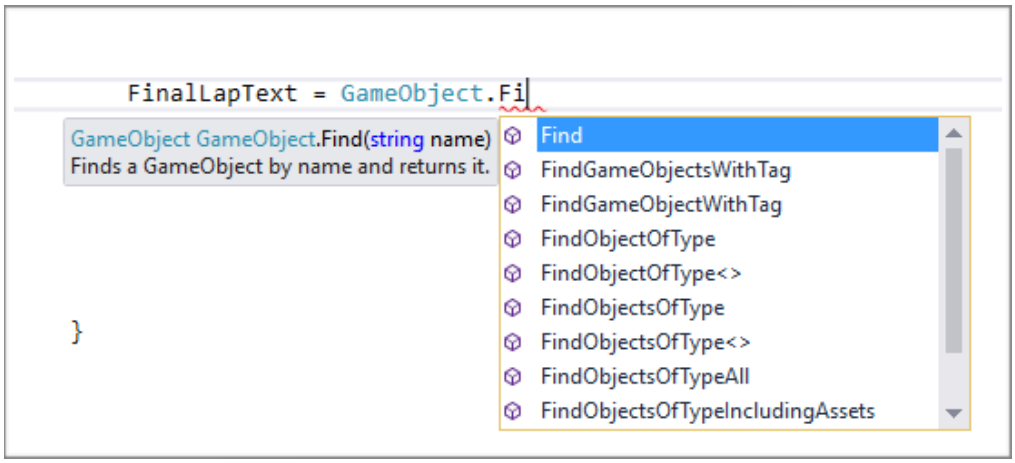
1.4.1 Autodesk Maya
Το Maya είναι ένα πρόγραμμα τριδιάστατου μοντελισμού και animation. Είναι πολύ διαδεδομένο στην βιομηχανία των βιντεοπαιχνιδιών και του κινηματογράφου. Πέρα από την δημιουργία τρισδιάστατων μοντέλων και animation μπορεί να προσομοιώσει διάφορα ρεαλιστικά εφφέ όπως νερό και φωτιά.

1.4.2 Zbrush
Φτιαγμένο από την Pixologic, το Zbrush είναι ένα πρόγραμμα τρισδιάστατης γλυπτικής με το οποίο μπορεί ο χρήστης να δημιουργήσει τρισδιάστατα μοντέλα. Η βασική του διαφορά με το Autodesk Maya είναι ότι το Zbrush προσπαθεί να προσομοιώσει μια εμπειρία πιο κοντά στην παραδοσιακή γλυπτική για αυτό και συχνά το προτιμούνε σχεδιαστές με λιγότερη εμπειρία στους υπολογιστές και περισσότερη εμπειρία στις παραδοσιακές τέχνες όπως ζωγραφική και γλυπτική. Μια ακόμα σημαντική δυνατότητά του που χρησιμοποιήθηκε είναι το texture painting το οποίο είναι η διαδικασία με την οποία βάφει κάποιος τα μοντέλα τα οποία δημιούργησε.

1.4.3 Adobe Photoshop
Ένα πρόγραμμα επεξεργασίας εικόνας. Χρησιμοποιήθηκε κυρίως για την επεξεργασία και την διόρθωση των textures των μοντέλων.

1.4.4 Unity
Η Unity είναι μια δωρεάν μηχανή γραφικών που χρησιμοποιείται για την δημιουργία παιχνιδιών για υπολογιστές, κονσόλες κινητά και ιστοσελίδες. Η πρώτη της έκδοση βγήκε το 2005 και από τότε κάθε χρόνο βγαίνουν νέες εκδόσεις και προσθέτονται συνέχεια νέες δυνατότητες. Η Unity είναι κατάλληλη τόσο για την δημιουργία τρισδιάστατων όσο και δισδιάστατων παιχνιδιών. Η μεγάλη δύναμη της έγκειται στην ευκολία να δημιουργήσεις ένα παιχνίδι μια φορά και να το τρέξεις εύκολα σε όποια κονσόλα ή υπολογιστή θέλεις.

1.4.5 Visual Studio 2015
Το visual studio της microsoft είναι ένα IDE(Integrated Development Environment) για την γρήγορη και εύκολη συγγραφή κώδικα. Βασικό στοιχείο του visual studio είναι το intellisense σύστημα το οποίο επιταχύνει την διαδικασία του προγραμματισμού. Αυτό το πετυχαίνει ελέγχοντας το κείμενο που έχει ήδη γράψει ο προγραμματιστής και διορθώνοντας ορθογραφικά λάθη ή δίνοντας προτάσεις για το πώς να ολοκληρωθεί μια εντολή του κώδικα.



Εικόνα 1.7: Το σύστημα intellisense προτείνει και εξηγεί την συνέχεια του κώδικα.

2. ΔΗΜΙΟΥΡΓΙΑ ΣΤΟΙΧΕΙΩΝ ΠΑΙΧΝΙΔΙΟΥ (GAME ASSETS)

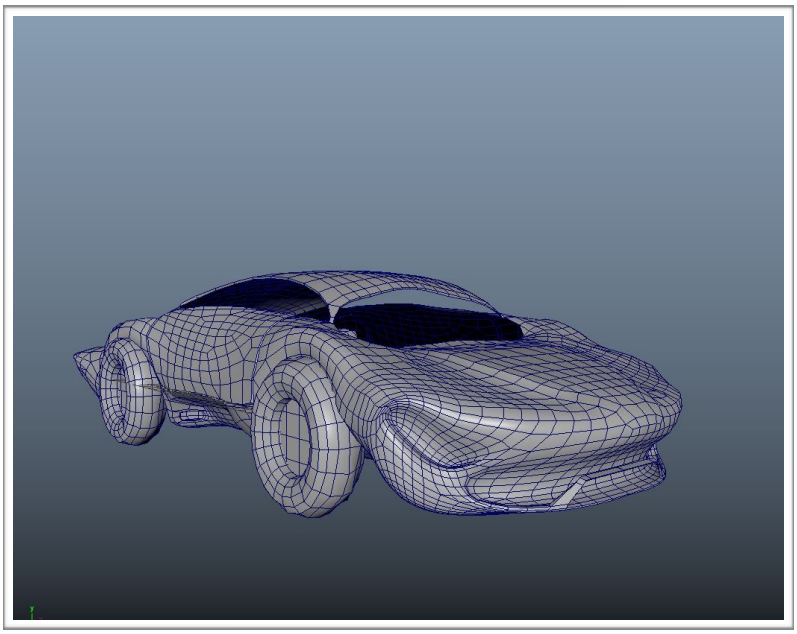
2.1 Εισαγωγή

Για την δημιουργία οποιουδήποτε βιντεοπαιχνιδιού χρειάζονται τα κατάλληλα σχεδιαστικά εργαλεία τα οποία θα δημιουργήσουν αυτά που θα βλέπει ο χρήστης. Σήμερα υπάρχουν πολλά προγράμματα που εξυπηρετούν τον σκοπό αυτό, από δωρεάν ανοιχτού λογισμικού όπως το Blender, μέχρι και πιο επαγγελματικές προτάσεις όπως το Autodesk Maya. Για την δημιουργία ενός ολοκληρωμένου στοιχείου χρησιμοποιείται συχνά ο συνδυασμός διαφόρων προγραμμάτων αξιοποιώντας τις δυνατότητες του κάθε προγράμματος.

2.2 Βήματα δημιουργίας μοντέλου

2.2.1 Δημιουργία πλέγματος(mesh)

Ο πυρήνας του κάθε μοντέλου είναι το πλέγμα του. Ένα mesh αποτελείται από τις κορυφές(vertices), τις ακμές (edges) και τις πλευρές (faces). Όσο πιο πολλές πλευρές έχει ένα mesh τόσο πιο πολύπλοκη λεπτομέρεια μπορεί να αναπαραστήσει και τόσο περισσότερο χρόνο θα χρειαστεί για να εμφανιστεί στην οθόνη.



Εικόνα 2.1: Το πλέγμα του αυτοκινήτου

2.2.2 Δημιουργία UV χάρτη(UV-mapping)

Αφού έχουμε δημιουργήσει το βασικό πλέγμα του μοντέλου μας πρέπει ύστερα να δημιουργήσουμε μια δισδιάστατη αναπαράσταση του. Αυτό γίνεται με μια διαδικασία που λέγεται UV-χαρτογράφηση(UV-mapping). Αυτή η δισδιάστατη αναπαράσταση θα χρησιμοποιηθεί στην συνέχεια για να δωθεί υφή(texture) στο μοντέλο μας αλλά και για άλλες οπτικές του βελτιώσεις.

2.2.3 Δημιουργία υφών(texture maps)

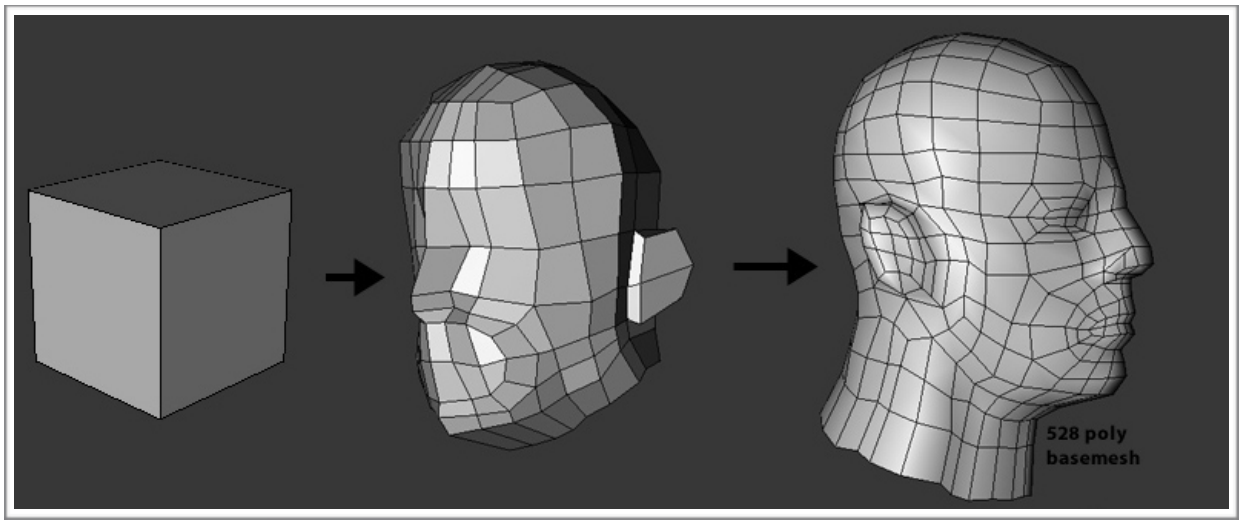
Τέλος αφού έχουμε το μοντέλο και το UV-map έτοιμο, το τελευταίο βήμα είναι η δημιουργία των χαρτών υφών. Ανάλογα με το επιθυμητό τελικό οπτικό αποτέλεσμα υπάρχει ένας τεράστιος αριθμός ειδών texture maps.

2.3 Διαδικασία μοντελοποίησης

Ώς διαδικασία μοντελοποίησης ορίζεται η δημιουργία του πλέγματος του μοντέλου μας. Μέχρι και τις αρχές του 2000 ένας ήταν ο τρόπος με τον οποίον όλοι οι σχεδιαστές φτιάχναν τρισδιάστατα μοντέλα και αυτός λεγόταν box-modelling. Τα τελευταία χρόνια, με την δυνατότητα πλέον των υπολογιστών να μπορούν να επεξεργαστούν πλέγματα με εκατομμύρια πολύγονα εμφανίστηκε και μια δεύτερη τεχνική, η τρισδιάστατη γλυπτική(3-d sculpting). Σήμερα χρησιμοποιούνται και οι δύο τεχνικές εξίσου και είναι στο χέρι του σχεδιαστή να διαλέξει ποιά προτιμάει.

2.3.1 Box modelling

Στην μέθοδο box modelling ο σχεδιαστής ξεκινάει από ένα πολύ απλό mesh (συνήθως έναν κύβο) και από τον απλό κύβο, εξωθώντας και υποδιαιρώντας τις πλευρές του δημιουργεί την γενική μορφή του μοντέλου που θέλει να δημιουργήσει. Το καλό με την τεχνική αυτή είναι ότι δίνει στον σχεδιαστή περισσότερο έλεγχο κατά την δημιουργία του μοντέλου μιας και αυτός είναι που ελέγχει το πόσες θα είναι οι τελικές πλευρές του μοντέλου. Το μειονέκτημά της είναι ότι είναι πιο χρονοβόρο και πιο δύσκολο αν θες να δημιουργήσεις μοντέλα με μη κυβική μορφή, όπως για παράδειγμα ανθρωπόμορφα μοντέλα.

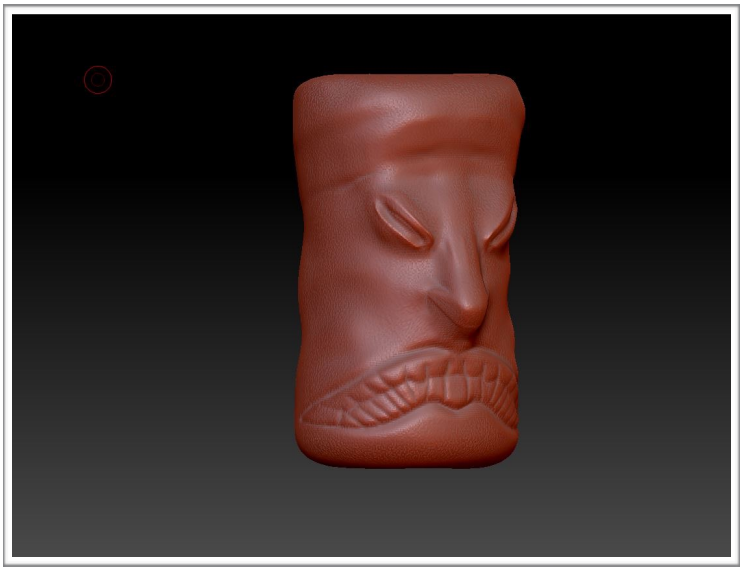


Εικόνα 2.2: Διαδικασία Box Modelling Πηγή: www.turx.com

2.3.2 3-D Sculpting

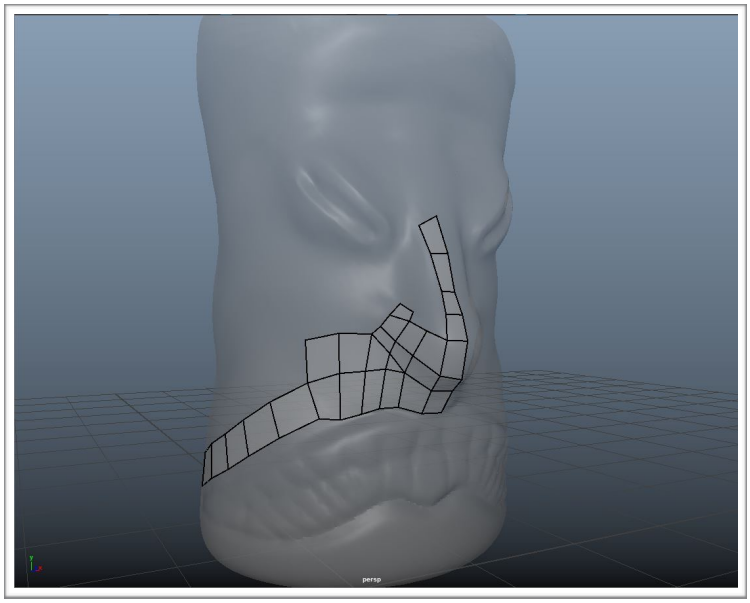
Η μέθοδος τρισδιάστατης γλυπτικής θα μπορούσαμε να πούμε ότι είναι το ανάποδο από το box modelling. Εδώ ο χρήστης ξεκινάει από μια πολύπλοκη υφή με χιλιάδες πλευρές από την αρχή(π.χ. μια σφαίρα με πολλές ακμές). Όπως προϊδεάζει το όνομά της η τεχνική αυτή θυμίζει πολύ παραδοσιακή γλυπτική γιατί ο χρήστης με διάφορα εργαλεία πλάθει ένα τριδιάστατο μοντέλο. Σε αυτή την τεχνική ο σχεδιαστής δεν έχει σχεδόν καθόλου έλεγχο

στον τελικό αριθμό των ακμών του μοντέλου αλλά έχει πιο γρήγορα ένα όμορφο οπτικό αποτέλεσμα.



Εικόνα 2.3: Πλέγμα τοτέμ με 126976 πλευρές

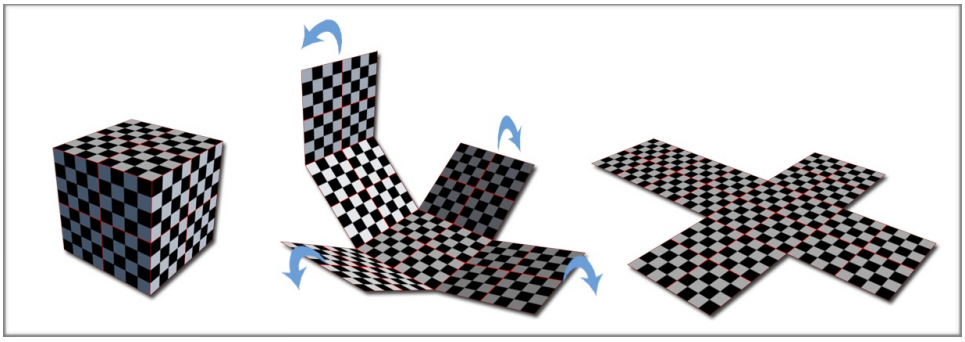
Το βασικό μειονέκτημα αυτής της τεχνικής είναι ότι επειδή το τελικό πλέγμα θα είναι υπερβολικά πολύπλοκο με πολλές χιλιάδες πλευρές, οι οποίες θα πρέπει ύστερα να μειωθούν, ώστε να μπορεί να χρησιμοποιηθεί στο παιχνίδι ή σε άλλο πρόγραμμα. Αυτή η τεχνική λέγεται retopology και αυτό που κάνει ο χρήστης είναι στην ουσία να σχεδιάζει τετραγωνάκια πάνω στο ήδη πολύπλοκο πλέγμα που έχει δημιουργήσει τα οποία αυτά τα τετραγωνάκια στο τέλος θα δημιουργήσουν ένα απλούστερο πλέγμα.



Εικόνα 2.4: Εφαρμόζωντας retopology στο πολύπλοκο πλέγμα

2.4 Διαδικασία UV-mapping

Για την δημιουργία της τελικής εμφάνισης των μοντέλων μας θα πρέπει να τοποθετηθεί πάνω τους μια υφή (texture). Τα textures είναι δισδιάστατες εικόνες και αφού θέλουμε να τα τοποθετήσουμε πάνω σε ένα τριδιάστατο μοντέλο θα πρέπει πρώτα να δημιουργήσουμε έναν <<χάρτη>> ο οποίος θα λέει στην μηχανή γραφικών μας πού να τοποθετήσει το κάθε pixel της υφής πάνω στο μοντέλο. Ο <<χάρτης>> αυτός λέγεται UV-map και όπως προαναφέρθηκε η διαδικασία ονομάζεται UV-Mapping. Αφού δημιουργήσουμε το UV-map το χρησιμοποιούμε σαν γνώμονα για την δημιουργία των διαφόρων texture maps που θα μας χρειαστούν.

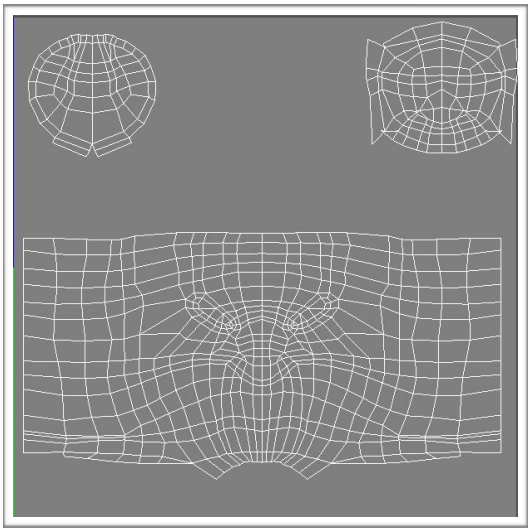


Εικόνα 2.5: Διαδικασία UV-mapping. Πηγή: http://brokenverts.com

Αν και έχουν αναπτυχθεί αλγόριθμοι για την αυτόματη δημιουργία ενός UV-map ακόμα και σήμερα προτιμάται το να δημιουργηθούν από τον ίδιο τον χρήστη. Ακολουθούνε οι βασικότεροι λόγοι για τους οποίους συμβαίνει αυτό:

2.4.1 Μέγιστη εκμετάλλευση των pixels του UV-MAP

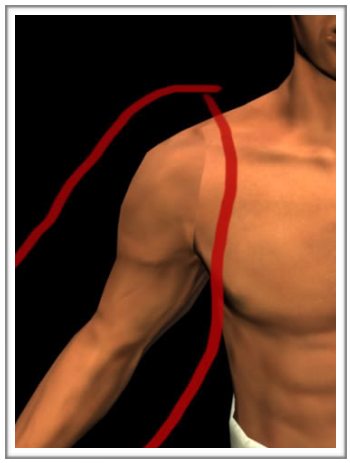
Όταν ο χρήστης δημιουργεί ο ίδιος τα UV-maps έχει τον πλήρη έλεγχο των σημείων της υφής στο οποίο θα δώσει έμφαση. Για παράδειγμα αν υπάρχει ένα μοντέλο στο παιχνίδι στο οποίο θα φαίνεται μόνο η μπροστινή πλευρά θα είναι συνετό αυτή η πλευρά να πιάνει περισσότερο χώρο στο UV-map ώστε να βελτιωθεί το οπτικό της αποτέλεσμα. Για παράδειγμα στο τοτέμ που χρησιμοποιήθηκε στο παιχνίδι η πάνω και η κάτω πλευρά που δεν φαίνονται σχεδόν ποτέ κατά την διάρκεια του παιχνιδιού πιάνουν ελάχιστο χώρο στο UV-map όπως φαίνεται και στην εικόνα.



Εικόνα 2.6: UV-map τοτέμ

2.4.2 Επιλογή των σημείων χωρισμού του UV-map

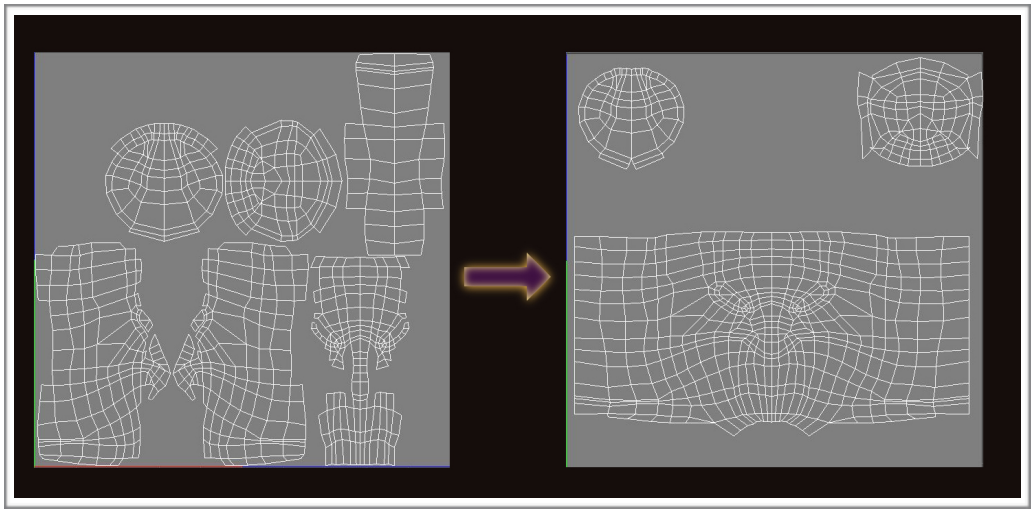
Πολλές φορές δεν είναι εφικτό ή δεν συμφέρει όλο το UV-map να είναι ένα εννιαίο κομμάτι(όπως στο παράδειγμα με τον κύβο). Σε αυτές τις περιπτώσεις το UV-map χωρίζεται σε τμήματα τα οποία ονομάζονται <<κέληφη>> (UV-shells). Η επιλογή αυτών των UV-shells έχει σημασία γιατί στα σημεία που χωρίζονται τα shells (τα UV-borders) αν αυτά δεν έχουν επιλεχτεί σωστά μπορεί να δημιουργηθούν προβλήματα όπως στην εικόνα. Για να μην δημιουργούνται τέτοια προβλήματα πρέπει ο σχεδιαστής να επιλέγει ως UV-borders σημεία τα οποία δεν φαίνονται ή τουλάχιστον δεν φαίνονται συχνά στον χρήστη.



Εικόνα 2.7: Κακή επιλογή UV-border.

2.4.3 Δημιουργία πιο ομαλού UV-map

Όταν ο υπολογιστής δημιουργεί το δικό του UV-map συνήθως θα δημιουργεί UV-borders σε τυχαία σημεία και η τελική εικόνα δεν θα είναι <<ευανάγνωστη>> στον τελικό χρήστη όπως φαίνεται και στην εικόνα:
Για να ελέγξουμε ότι το UV-map μας είναι ομαλό και σωστό μπορούμε να βάλουμε μια υφή σκακιέρας(checkerboard texture) στο μοντέλο μας. Θα πρέπει το μοντέλο μας να αποτελείται από όσο το δυνατόν γίνεται ομαλότερα μικρά τετράγωνα. Σε σημεία που υπάρχουν παραμορφώσεις και τα τετράγωνα κόβωνται ή έχουν διαστρεβλωθεί σημαίνει ότι είναι προβληματικό το UV-map εκεί πέρα.



Εικόνα 2.8: Μετατροπή αυτόματου UV-map(αριστερά) στο τελικό ευανάγνωστο(δεξιά)

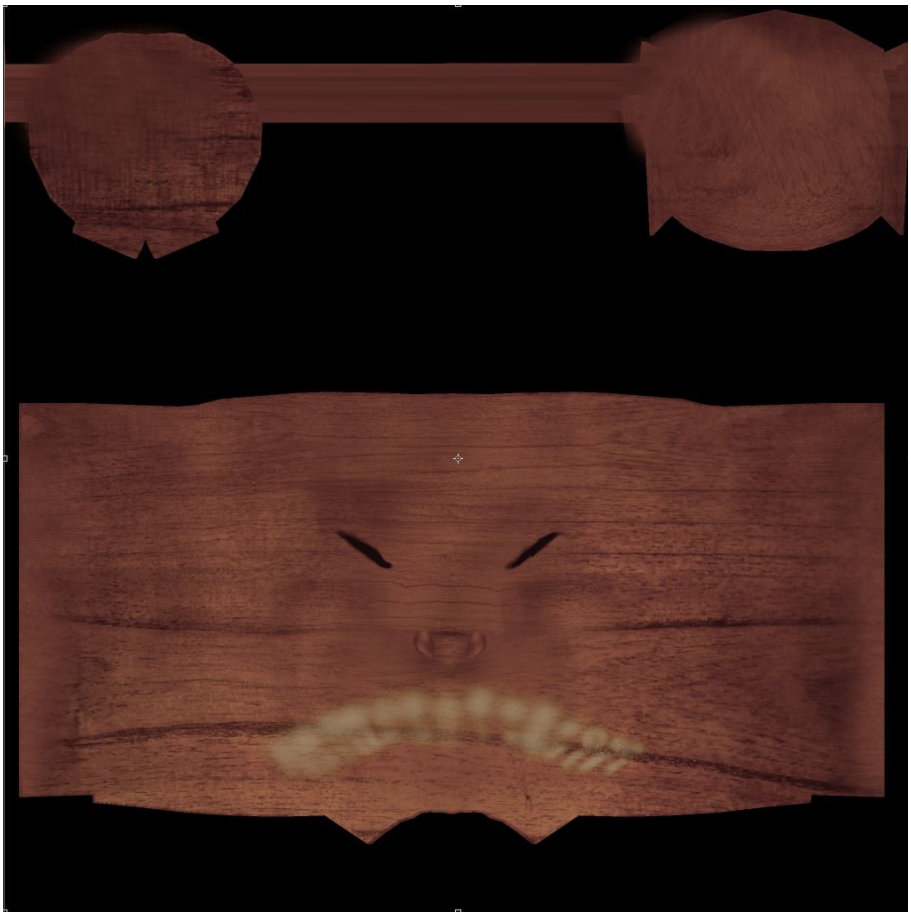
2.5 Δημιουργία υφών(texturing)

Αφού έχουμε δημιουργήσει ένα ομαλό και ευανάγνωστο UV-map για το μοντέλο μας ήρθε η ώρα να δημιουργήσουμε τις υφές οι οποίες θα του δώσουν την τελική του εμφάνιση. Για την δημιουργία αυτών των υφών μπορεί να χρησιμοποιηθεί οποιοδήποτε πρόγραμμα ζωγραφικής στο οποίο μπορούμε να ζωγραφίσουμε τα textures χρησιμοποιώντας ως

γνώμονα το UV-map. Ακολουθούν τα είδη των texture maps που χρησιμοποιήθηκαν στο παιχνίδι

2.5.1 Albedo Map

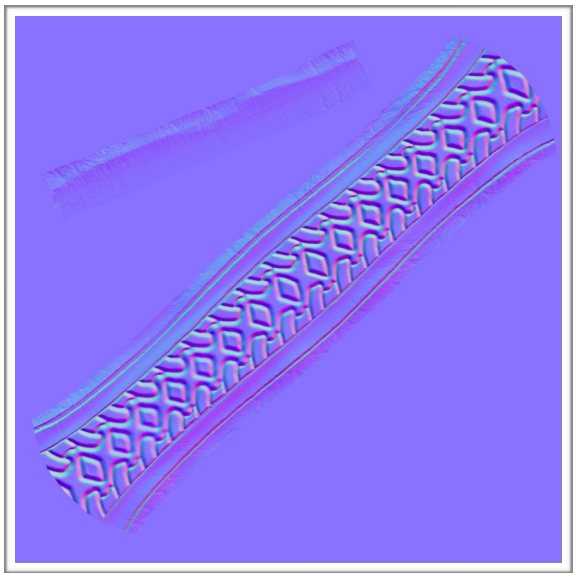
Η συγκεκριμένη υφή είναι μια RGB εικόνα η οποία θα δώσει το βασικό χρώμα στο μοντέλο. Πέρα των τριών χρωματικών καναλιών (κόκκινο, κίτρινο, μπλέ) μπορεί να χρησιμοποιηθεί ένα επιπλέον alpha channel το οποίο χρησιμοποιείται για να αλλάξει την διαφάνεια του μοντέλου(για παράδειγμα για τα τζάμια του αυτοκινήτου).



Εικόνα 2.9: Albedo Map του τοτέμ

2.5.2 Normal Map

Τα normal maps είναι μια υφή η οποία δημιουργεί την ψευδαίσθηση της πολυπλοκότητας σε ένα 3d μοντέλο χωρίς όμως αυτή να υπάρχει εκεί στην πραγματικότητα. Όπως έχει προαναφερθεί όσες περισσσότερες πλευρές έχει ένα τριδιάστατο μοντέλο τόσο πιο βαρύ θα είναι για τον υπολογιστή να το εμφανίσει στην οθόνη. Μια μερική λύση σε αυτό το πρόβλημα είναι τα normal maps τα οποία αυτό που κάνουν στην ουσία είναι να δημιουργούν την ψευδαίσθηση του βάθους σε κάποια μοντέλα χωρίς όμως να υπάρχουν πραγματικά πλευρές στο μοντέλο εκεί. Για παράδειγμα στις ρόδες του αυτοκινήτου για να δημιουργηθεί η ψευδαίσθηση ότι υπάρχουν τα βαθουλώματα αντί να δημιουργούνται εκατοντάδες επιπλέον μικρές πλευρές μέσα στις ρόδες απλά χρησιμοποιούμε ένα normal map όπως βλέπουμε και στην παρακάτω εικόνα.

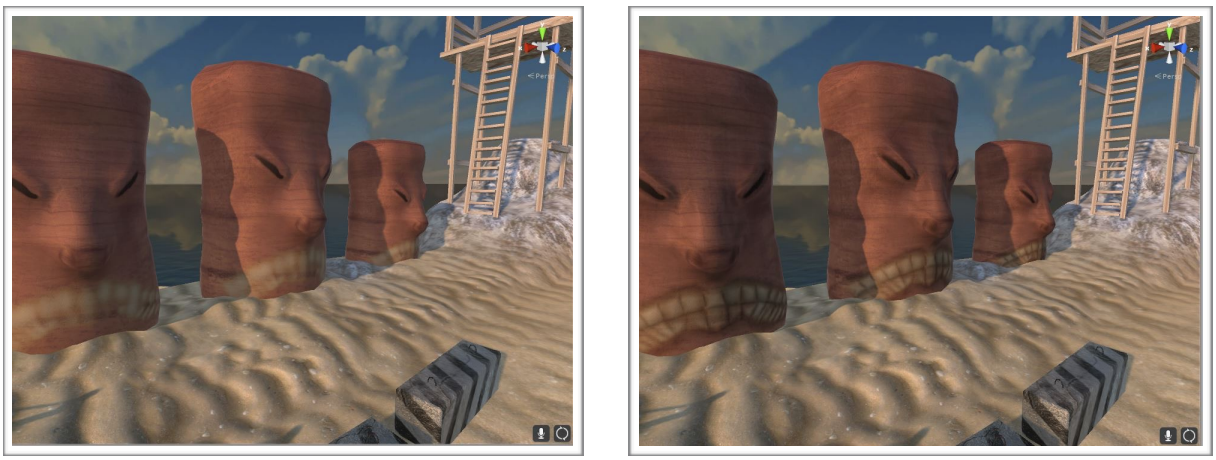


Εικόνα 2.10: Ενώ το πλέγμα είναι απλό όπως φαίνεται από το γαλάζιο περίγραμμα(αριστερά) υπάρχει η ψευδαίσθηση του βάθους με την χρήση του normal map (δεξιά)

2.5.3 Ambient Occlusion Map

Η συγκεκριμένη υφή είναι μια εικόνα μόνο με αποχρώσεις του γκρί(greyscale image) η οποία δίνει την ψευδαίσθηση της σκιάς στα βαθουλώματα του μοντέλου. Όσο πιο σκοτεινά είναι τα pixel στην εικόνα τόσο λιγότερο φώς θα πέφτει στην αντίστοιχη περιοχή του μοντέλου από το σύστημα φωτισμού της μηχανής γραφικών.

Εικόνα 2.11: Η εικόνα του Ambient Occlusion Texture



Εικόνα 2.12: Τονισμός στην σκιά των δοντιών του τοτέμ πριν και μετά την χρήση Ambient Occlusion map.

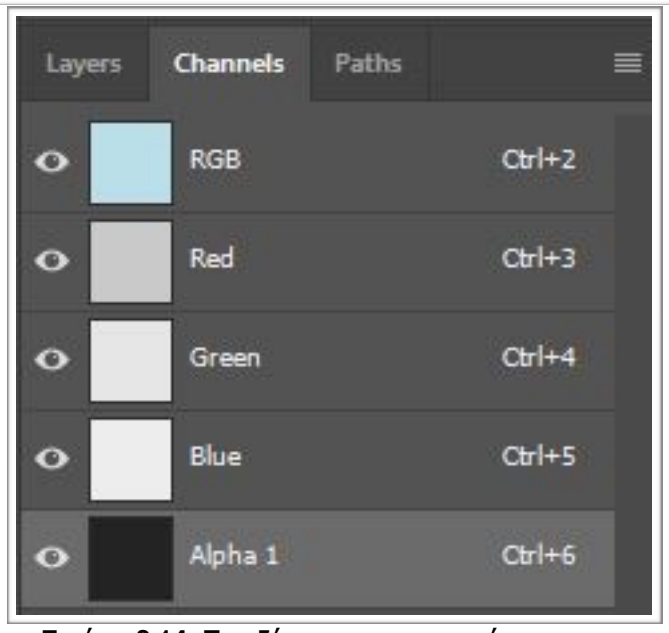
2.5.4 Emission Map

Τα emission maps χρησιμοποιούνται για να κάνουν συγκεκριμένα μέρη του μοντέλου να εκπέμπουν φωτισμό. Όσο πιο άσπρο είναι ένα pixel του texture τόσο πιο πολύ φως θα εκπέμπει. Επίσης η συγκεκριμένη υφή δέχεται και χρώματα για να προσομοιώσει φώτα μεγάλης γκάμας χρωμάτων. Είναι ιδιαίτερα χρήσιμη για τα πίσω φώτα του αυτοκινήτου τα οποία έχουν προγραμματιστεί έτσι ώστε όταν ο χρήστης πατάει το κάτω βελάκι για να φρενάρει το αυτοκίνητο, να αλλάζει το texture του emission και να το κάνει από μαύρο άσπρο έτσι ώστε να αρχίσουν να εκπέμπουν φώς.

Εικόνα 2.13: Αριστερά το emission texture είναι μαύρο και δεξιά κόκκινο.

5.5.5 Transparency Map

Η υφή διαφάνειας (transparency map) είναι χρήσιμη σε περιπτώσεις που θέλουμε κάποιο κομμάτι του μοντέλου να είναι διαφανές είτε μερικώς είτε τελείως. Για παράδειγμα στην υφή του σώματος του αυτοκινήτου το κομμάτι των τζαμιών έχει περίπου 20% διαφάνεια ενώ το υπόλοιπο 100%. Με αυτόν τον τρόπο δημιουργείτε η ψευδαίσθηση της διαφάνειας του τζαμιού και μπορεί ο οδηγός να βλέπει έξω από το αυτοκίνητο. Για την δημιουργία της υφής διαφάνειας δεν χρησιμοποιείτε επιπλέον υφή όπως στις παραπάνω αλλά χρησιμοποιείτε ένα παραπάνω κανάλι πέραν του κόκκινου-μπλέ-πράσινου στο albedo map. Ονομάζουμε το κανάλι άλφα-κανάλι(alpha channel) και εκείνο ανάλογα με την ένταση των pixels πάνω του ορίζει την διαφάνεια των αντίστοιχων περιοχών του μοντέλου. Όσο πιο μαύρο είναι τόσο πιο διαφανής θα είναι η επιφάνεια.



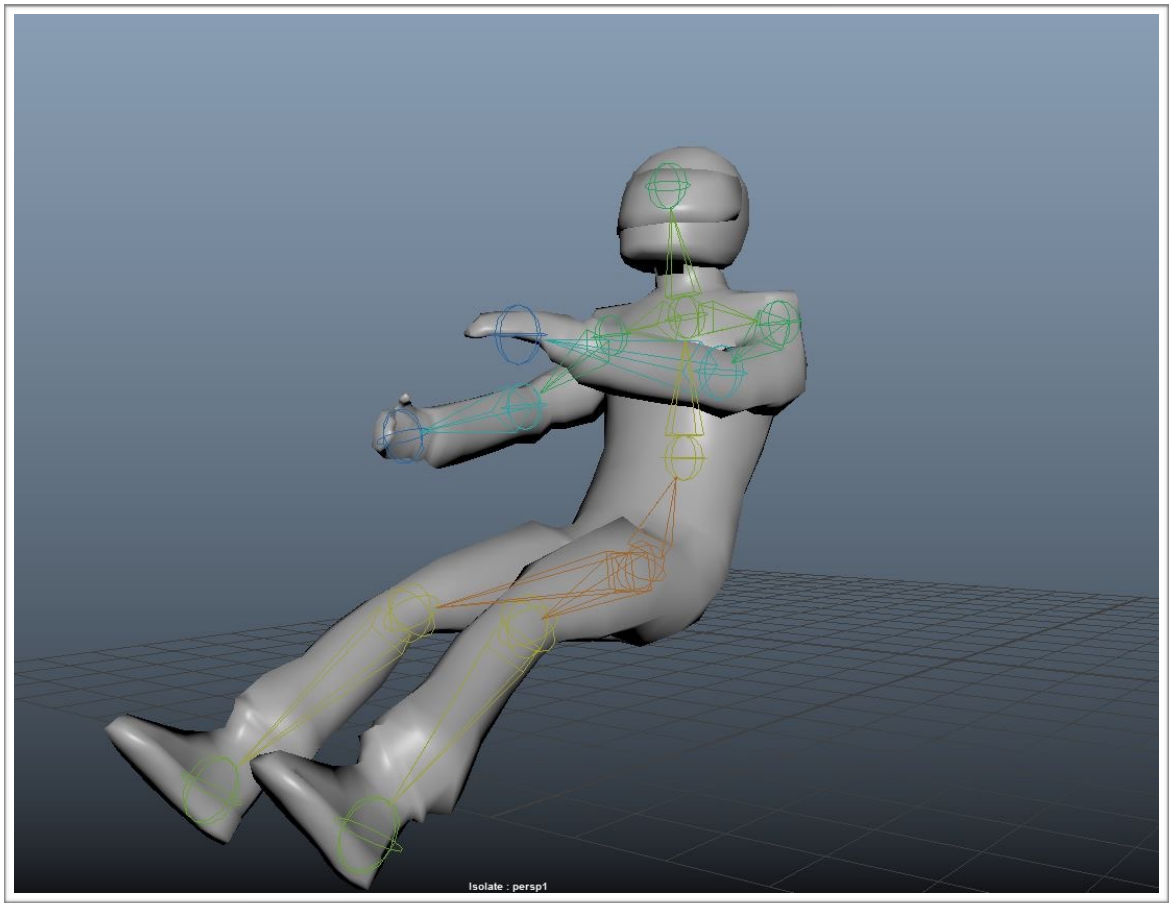
Εικόνα 2.14: Τα τζάμια του αυτοκινήτου και τα κανάλια χρώματος της υφής μαζί με το κανάλι άλφα.

2.6 Animation

Πολλές φορές θέλουμε τα τρισδιάστατα μοντέλα να κάνουν πολύπλοκες κινήσεις κατά την διάρκεια του παιχνιδιού, όπως στην περίπτωση μας τα χέρια του οδηγού να στρίβουν μαζί με το τιμόνι όταν το αυτοκίνητο στρίβει. Η φιλοσοφία του τριδιάστατου animation δεν διαφέρει πολύ από το παραδοσιακό animation που χρησιμοποιείται σε ταινίες κινουμένων σχεδίων. Και εδώ η λογική είναι να δημιουργηθούν στιγμιότυπα στον χρόνο με διαφορετικές καταστάσεις του μοντέλου οι οποίες αν παιχτούν ύστερα σε συνέχεια, προσδίδουν ζωντάνια σε αυτό. Για την δημιουργία του animation του οδηγού χρησιμοποιήθηκε το πρόγραμμα Autodesk Maya και ακολουθούν τα βήματα για την δημιουργία του τελικού αποτελέσματος.

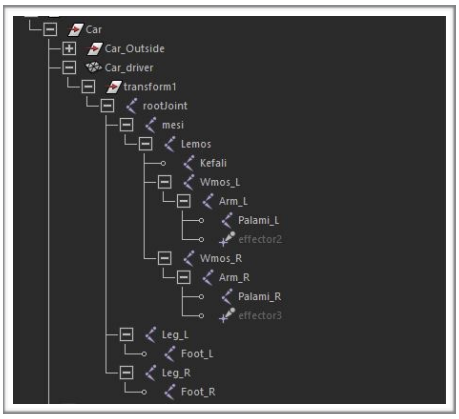
2.6.1. Δημιουργία skeleton

Για να δημιουργήσουμε αληθοφανή animations στον χαρακτήρα πρώτα πρέπει να δημιουργήσουμε έναν σκελετό με βάση τον οποίον θα κινείται. Ο σκελετός όπως και στην πραγματικότητα αποτελείται από αρθρώσεις και κόκκαλα. Το καθένα από αυτά είναι υπεύθυνο στο να κινεί τα σημεία του μοντέλου που βρίσκονται γύρω από αυτό. Κουνώντας και γυρίζοντας λοιπόν τις αρθρώσεις και τα κόκκαλα μπορεί ο σχεδιαστής να αναπαραστίσει πολύπλοκες ανθρώπινες κινήσεις.



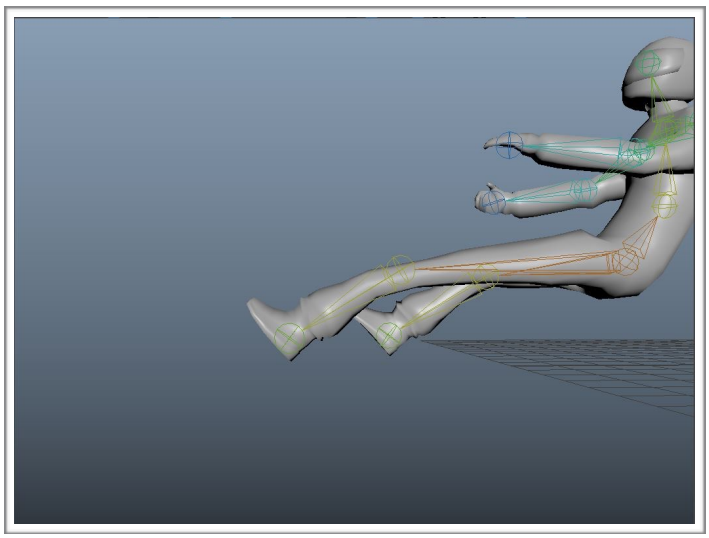
Εικόνα 2.15: Ο σκελετός του οδηγού

Ένα ακόμη βασικό χαρακτηριστικό του σκελετού είναι το ιεραρχικό του σύστημα. Κάθε σκελετός αποτελείται από την άρθρωση-ρίζα(root joint) και από τα παιδιά-αρθρώσεις(child-joints). Όταν ο σχεδιαστής αλλάζει την περιστροφή ή την τοποθεσία ενός γονέα-άρθρωσης(parent joint), τότε επηρεάζονται με τον ίδιο τρόπο και τα παιδιά-αρθρώσεις του. Υπάρχουν δύο τεχνικές για την δημιουργία της κίνησης του συστήματος σκελετού, η προς τα μπρός κινηματική(forward kinematics) και η προς τα πίσω κινηματική(inverse kinematics).



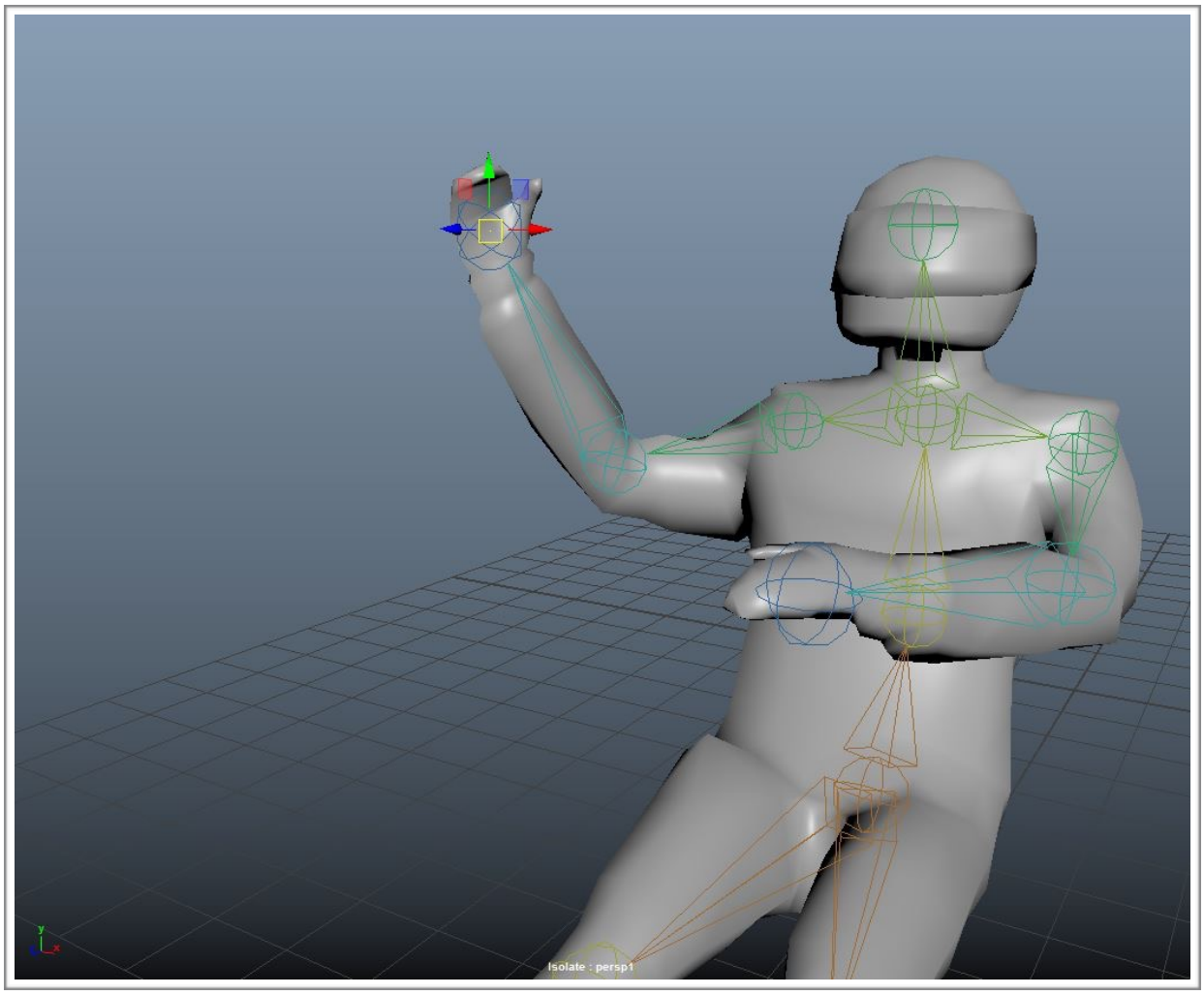
Εικόνα 2.16: Η Ιεραρχεία του σκελετού

Στην προς τα εμπρός κινηματική ο σχεδιαστής μπορεί να επηρεάσει την κάθε άρθρωση ξεχωριστά και να αποδώσει την επιθυμητή πόζα στον χαρακτήρα. Ξεκινώντας πάντα από την ρίζα του σκελετού και μετακινούμενος προς τα φύλλα της ιεραρχίας ο σχεδιαστής επηρεάζει συνήθως το rotation των αρθρώσεων και όχι το transform. Όταν επηρεάζεται το transform μιας άρθρωσης τότε μεγαλώνει το κόκκαλο μεταξύ αυτής και του γονέα της και αυτό έχει ως αποτέλεσμα να μεγαλώνει και εκείνο το κομμάτι του σώματος οπότε η αλλαγή του transform δεν χρησιμοποιείται συχνά. Βλέπουμε ένα τέτοιο παράδειγμα στην παρακάτω εικόνα.



Εικόνα Χ: Παραμόρφωση ποδιού

Πολλές φορές χρησιμοποιείται και μια εναλλακτική τεχνική η Inverse Kinematics. Σε αυτή την τεχνική ο χρήστης δημιουργεί έναν μοχλό (IK handle) μεταξύ 2 ή περισσότερων αρθρώσεων και κουνώντας αυτό τον μοχλό το πρόγραμμα υπολογίζει από μόνο του πού θα πρέπει να βρίσκονται και πώς πρέπει να στρέφονται οι ενδιάμεσες αρθρώσεις. Αυτό δίνει το πλεονέκτημα της ταχύτητας στον σχεδιαστή ώστε να μην χρειάζεται να αλλάζει ο ίδιος το κάθε joint ξεχωριστά.



Εικόνα 2.18: Κίνηση του IK Handle που επηρεάζει την παλάμι μέχρι τον ώμο.

2.6.2 Δημιουργία καρέ-κλειδιών

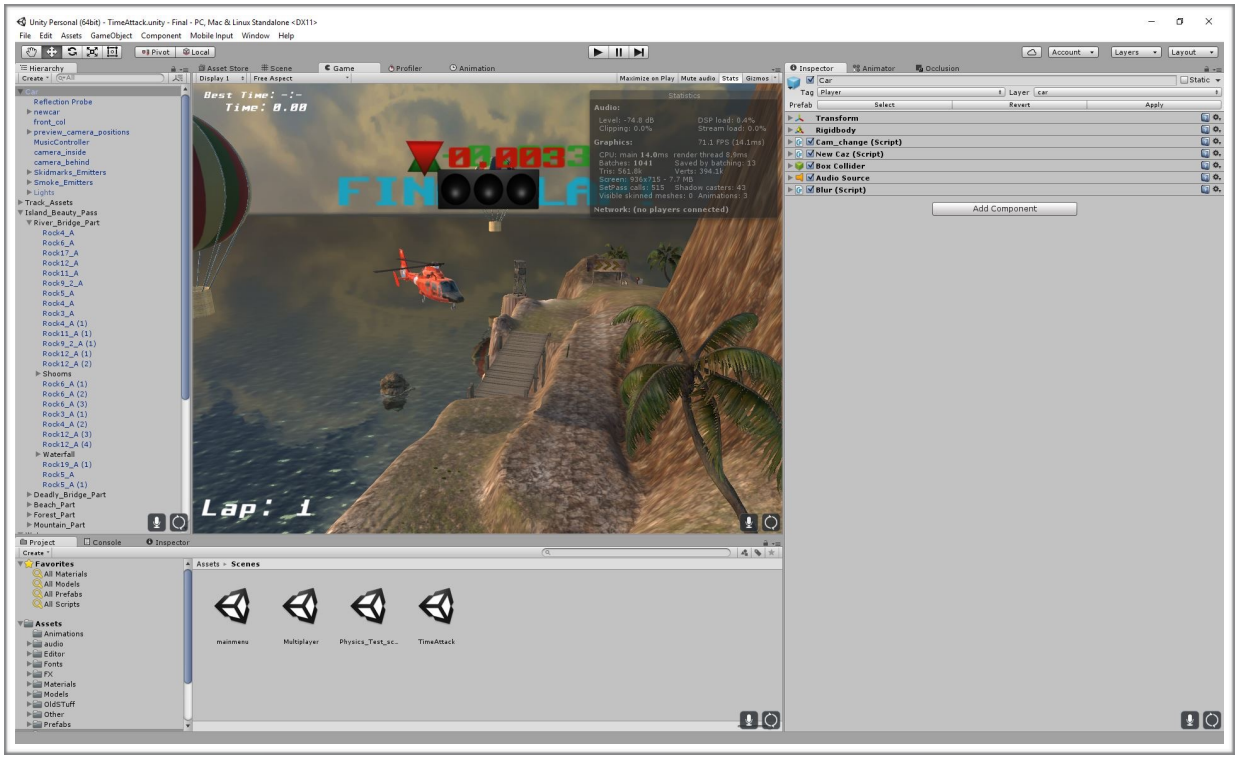
Αφού έχει δημιουργηθεί ο σκελετός του μοντέλου μας επόμενο βήμα για την δημιουργία του τελικού animation είναι να δημιουργήσουμε τα κατάλληλα καρέ-κλειδιά(keyframes) ώστε να δημιουργηθεί η επιθυμητή κίνηση. Για να το κάνουμε αυτό κουνάμε και γυρίζουμε σταδιακά τις αρθρώσεις και τους IK μοχλούς του χαρακτήρα και για κάθε αλλαγή που κάνουμε θέτουμε και ένα keyframe στην συγκεκριμένη αλλαγή. Αφού βάλουμε αρκετά keyframes ώστε να δημιουργηθεί μια ενδιαφέρουσα και ομαλή κίνηση το animation είναι έτοιμο.

3. ΣΧΕΔΙΑΣΜΟΣ ΠΑΙΧΝΙΔΙΟΥ

Στην συνέχεια περιγράφεται η πορεία ανάπτυξης του παιχνιδιού στην Unty.

3.1 Περιβάλλον εργασίας Unity

Το βασικό περιβάλλον εργασίας της Unity αποτελείται από 5 παράθυρα.
-Το παράθυρο σκηνής(scene) στο οποίο μπορούμε να μετακινηθούμε και να δούμε ολόκληρο τον κόσμο του παιχνιδιού.
-Το παράθυρο Inspector στο οποίο μπορούμε να πάρουμε πληροφορίες για το μοντέλο που έχουμε επιλέξει την συγκεκριμένη στιγμή.
-Το παράθυρο ιεραρχείας(hirierarchy) όπου φαίνονται όλα τα μοντέλα/ήχοι/εφφέ που έχουμε βάλει στην πίστα μας σε ιεραρχική δομή.
-Το παράθυρο project στο οποίο είναι αποθηκευμένα τα στοιχεία του παιχνιδιού(game assets) που έχουμε δημιουργήσει.
-Το παράθυρο παιχνιδιού(game) στο οποίο μπορούμε να τρέξουμε το παιχνίδι.
Αν και υπάρχουν πολλά περισσότερα παράθυρα όπως θα δούμε και παρακάτω, αυτά τα πέντε είναι αυτά που χρησιμοποιούνται περισσότερο.



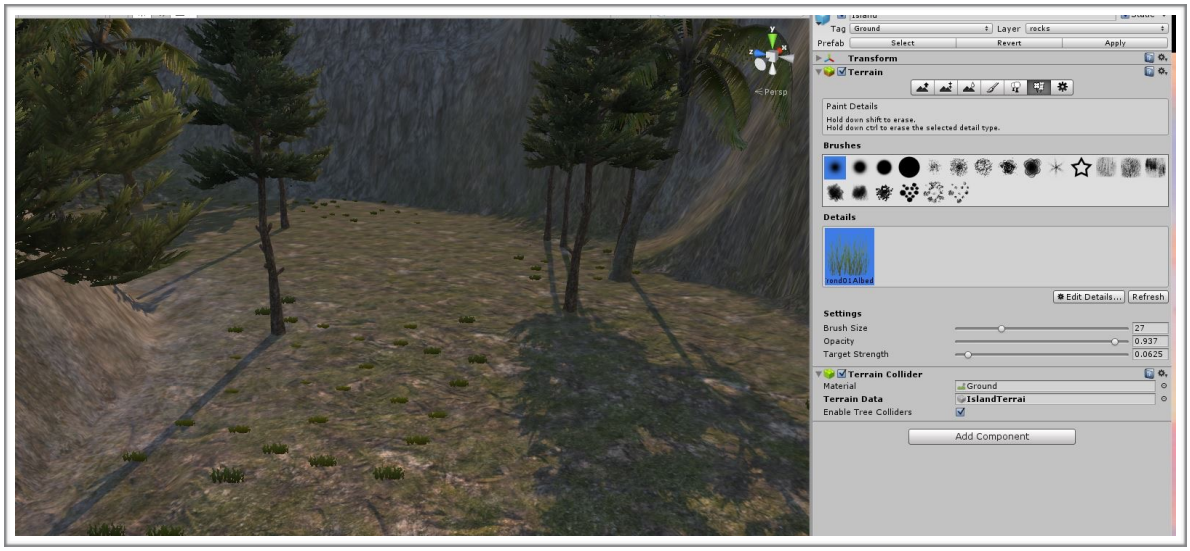
Εικόνα 3.1: Το περιβάλλον εργασίας

3.2 Δημιουργία πίστας

3.2.1 Δημιουργία σχηματισμού εδάφους πίστας (terrain)

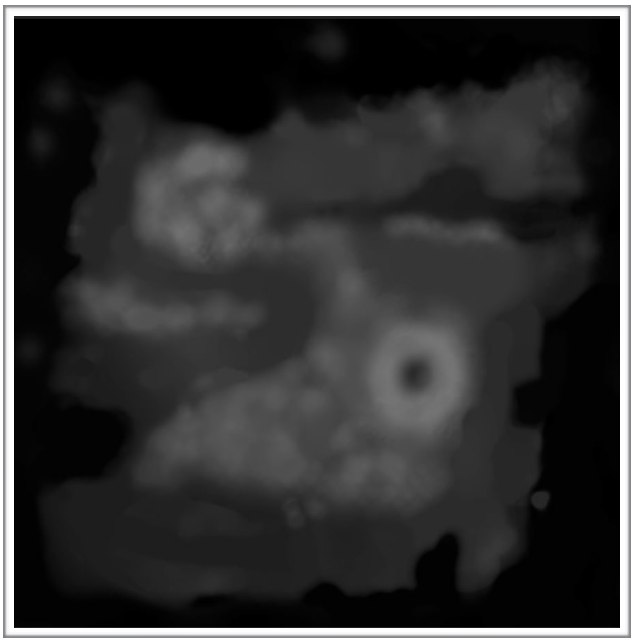
Για να δημιουργηθεί το νησί, η βασική πίστα του παιχνιδιού δηλαδή, χρησιμοποιήθηκε η μηχανή εδάφους(terrain engine) της Unity. Η συγκεκριμένη μηχανή βοηθάει στην γρήγορη

δημιουργία πολύπλοκων μορφολογιών εδάφους όπως πεδιάδες, λόφους βουνά κ.ά. Οι βασικές λειτουργίες του terrain είναι οι εξής:
-Ανέβασμα και κατέβασμα του εδάφους
-Βάψιμο του εδάφους με συγκεκριμένη υφή



Εικόνα 3.2: Τοποθέτηση βλάστησης με το Terrain-Engine.

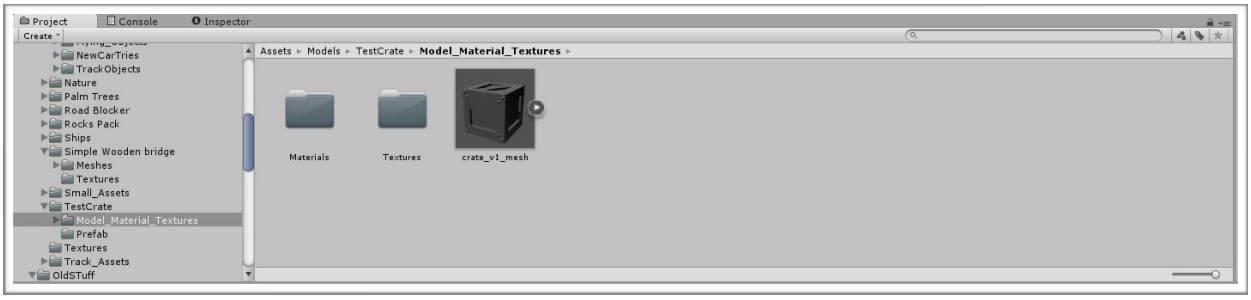
-Πρόσθεση βλάστησης (δέντρα, φυτά κ.ά)Μια επιπλέον δυνατότητα του terrain της Unity είναι η δυνατότητα να χρησιμοποιηθεί μια υφή ύψους(heightmap) ως χάρτης για την δημιουργία του εδάφους. Τα heightmaps είναι μια μορφή υφής greyscale της οποίας ανάλογα με το πόσο σκοτεινό ή άσπρο είναι το κάθε pixel επηρεάζει το πόσο ψηλά ή χαμηλά είναι το έδαφος στην αντίστοιχη περιοχή. Με αυτόν τον τρόπο μπορούμε να επεξεργαστούμε το έδαφος του παιχνιδιού σε ένα εξωτερικό πρόγραμμα επεξεργασίας εικόνας (όπως το Photoshop) και να δημιουργήσουμε εδαφικές ανωμαλίες που θα έπαιρνε περισσότερο χρόνο να επιτευχθούν μέσα στην μηχανή της Unity.



Εικόνα 3.3: Heightmap νησιού.

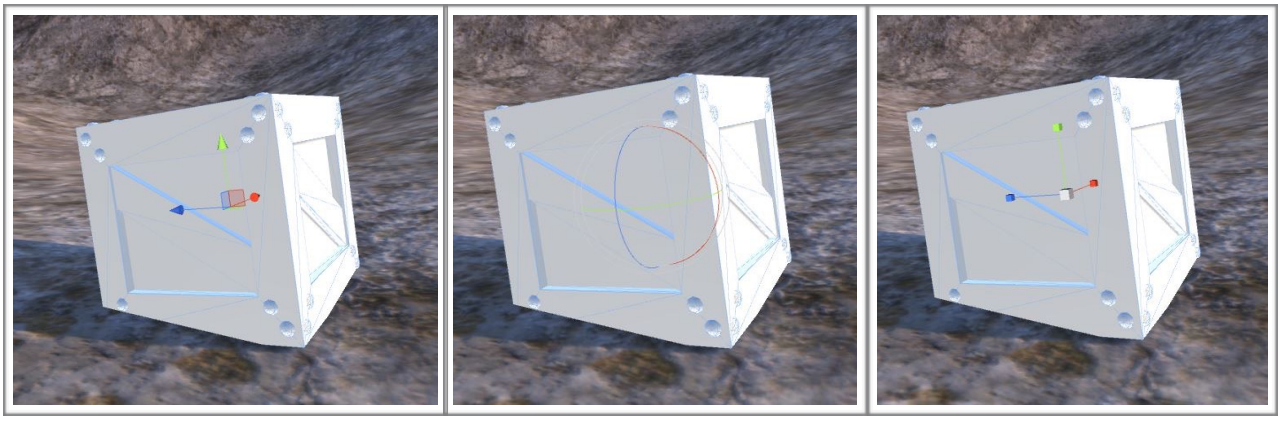
3.2.2 Εμπλουτισμός πίστας

Αφού έχουμε στήσει τον βασικό σκελετό της πίστας(το terrain δηλαδή), ήρθε η ώρα για την εισαγωγή των μοντέλων πάνω στο έδαφος που δημιουργήσαμε. Στην Unity η εισαγωγή ενος μοντέλου γίνεται πολύ απλά <<τραβώντας>> το μοντέλο από τον φάκελο assets πάνω στο παράθυρο scene. Ύστερα χρησιμοποιώντας τους μοχλούς για κίνηση,περιστροφή και μεγέθυνση στο παράθυρο scene, τοποθετούμε το μοντέλο στο σημείο που θέλουμε.



Εικόνα 3.4: Αναζήτηση μοντέλου στα Assets-Folder

Υπάρχουν 3 βασικοί μοχλοί τους οποίους χρησιμοποιούμε για την μετατροπή του μοντέλου. Ο πρώτος είναι ο μοχλός τοποθέτησης(transform handle) ο οποίος αλλάζει την τοποθεσία του μοντέλου στον χώρο. Ο δεύτερος, ο μοχλός περιστροφής(rotation handle) επιτρέπει την σφαιρική περιστροφή του μοντέλου στον χώρο. Ο τρίτος είναι ο μοχλός μεγέθους(size handle) ο οποίος επιτρέπει την μεγέθυνση και την σμίκρυνση του μοντέλου.



Εικόνα 3.5: Οι 3 μοχλοί μετατροπής του μοντέλου

3.2.4 Δημιουργία Skybox

Το skybox είναι ένας τεράστιος τριδιάστατος κύβος ο οποίος περικλύει την πίστα του παιχνιδιού με σκοπό να δώσει την ψευδαίσθηση του ουρανού και του ορίζοντα σε εξωτερικούς χώρους όπως το νησί στο παιχνίδι. Για την δημιουργία του χρειάζονται 6 υφές, μια για την κάθε πλευρά του κύβου.

Εικόνα 3.6: Η εμφάνιση του skybox texture

3.2.5 Δημιουργία ωκεανού και ποταμιού

Για την δημιουργία των στοιχείων του νερού της πίστας χρησιμοποιήθηκε το Unity water το οποίο αποτελεί την έτοιμη λύση της Unity για αληθοφανές νερό. Όπως βλέπουμε και στις παρακάτω εικόνες, το Unity water δίνει πολλές επιλογές όπως το ύψος και τη συχνότητα των κυμάτων ή το ποσοστό αντανάκλασης και διαφάνειας του νερού.

Εικόνα 3.7: Η εμφάνιση του ωκεανού με τις αντανακλάσεις

3.2.6 Δημιουργία Prefabs

Συνήθως αφού εισάγουμε ένα αντικείμενο μέσα στην σκηνή θα δεχτεί προσθέσεις από components ή μεταβολλές σε διάφορες τιμές του όπως μέγεθος περιστροφή κτλ. Μια εύκολη λύση για να μην χρειάζεται να κάνουμε τις ίδιες αλλαγές σε κάθε ίδιο μοντέλο που θέλουμε να βάλουμε είναι να αποθηκεύσουμε αυτό το αντικείμενο με όλα του τα δεδομένα και τις προδιαγραφές για να το ξαναχρησιμοποιήσουμε. Αυτό το αντικείμενο που θα αποθηκευτεί λέγεται prefab. Μια σημαντική χρήση ενός prefab είναι αν θέλουμε να αλλάξουμε την τιμή σε όλα τα ίδια prefabs φτάνει να αλλάξουμε την τιμή του ενός και θα αλλάξουν αυτόματα και τα υπόλοιπα.

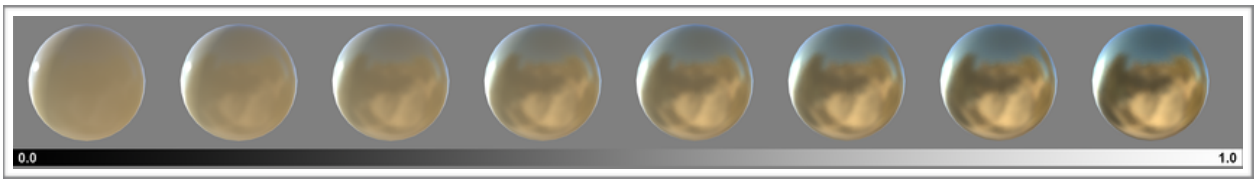
3.3 Πρόσθεση εξαρτημάτων στα μοντέλα(components)

Αφού έχουμε τοποθετήσει τα μοντέλα στα σημεία που θέλουμε το επόμενο βήμα είναι να τους προσθέσουμε components τα οποία θα τους δώσουν διάφορες λειτουργίες όταν τρέχει το παιχνίδι. Στην συνέχεια περιγράφονται τα βασικότερα components που χρησιμοποιούνται στα παιχνίδι.

3.3.1 Material

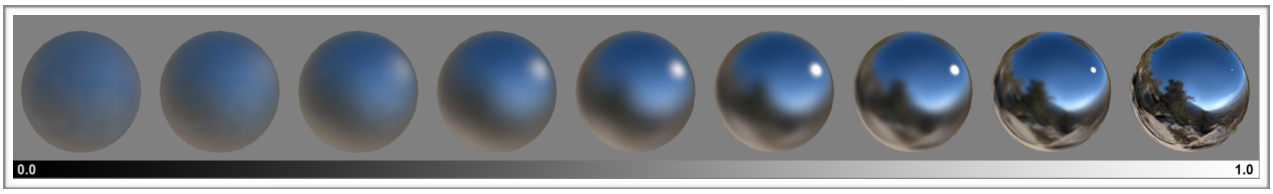
Είναι το Component εκείνο το οποίο περιέχει όλες τις υφές (Albedo map, Normal Map κ.τ.λ.) τις οποίες χρησιμοποιεί το εκάστοτε μοντέλο. Επίσης μας δίνει την δυνατότητα με sliders να αλλάξουμε το πόσο μεταλλικό είναι το μοντέλο και πόσο ομαλή είναι η επιφάνειά του.

Το metallic slider επηρεάζει το πόσο μεταλλικό θα δείχνει το μοντέλο. Όσο πιο μεταλλικό είναι το μοντέλο τόσο λιγότερο θα επηρεάζεται από το βασικό του χρώμα(το albedo map δηλαδή) και τόσο περισσότερο το χρώμα του θα επηρεάζεται από τις αντανακλάσεις του περιβάλλοντος.



Εικόνα 3.8: Η εμφάνιση μιας σφαίρας με βάση την τιμή του metallic slider. Πηγή: Unity-Manual

Το smoothness slider μας δίνει την δυνατότητα να επηρεάσουμε το πόσο ομαλή είναι η επιφάνεια του μοντέλου σε επίπεδο μικροσκοπίου. Στην ουσία δεν αλλάζει η πραγματική επιφάνεια του μοντέλου απλά αλλάζει ο τρόπος με τον οποίον το μοντέλο θα αντανακλάει το φώς. Όπως και στην πραγματικότητα έτσι και στην Unity όταν κάποιο αντικείμενο έχει ανώμαλη επιφάνεια σε μικροσκοπικό επίπεδο τότε το φώς αντανακλάται σε τυχαίες κατεθύνσεις και έτσι οι αντανακλάσεις του χάνουν την λεπτομέρειά τους. Αντίθετα όταν το αντικείμενο είναι πολύ ομαλό όπως ένας καθρέφτης τότε το φώς αντανακλάται ομοιόμορφα και έτσι βλέπουμε την αντανάκλαση του περιβάλλοντος πάνω του.



Εικόνα 3.9: Η εμφάνιση μιας σφαίρας με βάση την τιμή του smoothness slider. Πηγή: Unity-Manual

3.3.2 Collider

Τα colliders είναι τα γεωμετρικά σχήματα τα οποία χρησιμοποιούνται από την Unity για τον υπολογισμό των φυσικών συγκρούσεων μεταξύ μοντέλων. Υπάρχουν 2 ειδών colliders τα primitive και τα mesh colliders. Στην πρώτη κατηγορία ανήκουν τα βασικά γεωμετρικά σχήματα όπως ο κύβος η σφαίρα και ο κύλινδρος. Μπορούν σε ένα μοντέλο να συνδυαστούν δύο ή παραπάνω από αυτά για να δημιουργηθούν πιο πολύπλοκα σχήματα. Η δεύτερη κατηγορία των mesh colliders είναι εκείνα τα οποία το σχήμα του collider ακολουθάει ακριβώς το σχήμα της τοπολογίας του μοντέλου. Είναι συνετό να χρησιμοποιούνται κυρίως τα primitive colliders ή συνδυασμός αυτών, μιας και είναι πιο εύκολο και ελαφρύ για τον υπολογιστή να υπολογίσει τις συγκρούσεις μεταξύ τους αφού περιέχουν πολύ λιγότερες πλευρές.
Βασικό χαρακτηριστικό του κάθε collider είναι το υλικό φύσης του (physics material). Αυτό ορίζει πώς θα αντιδράει με τα υπόλοιπα colliders με τα οποία θα έρθει σε επαφή. Στο physics material μπορούμε να ορίσουμε την τριβή και το bounciness του collider. Η τριβή επηρεάζει το πόση κινητική ενέργεια απορροφάει το collider αυτό από άλλα colliders που το ακουμπάνε. Το bounciness κάνει το collider να δουλεύει σαν τραμπολίνο. Αν η τιμή του είναι μηδέν τότε το collider αυτό ακινητοποιεί το collider που προσπίπτει πάνω του. Αν η τιμή του είναι 1 τότε το εκσφενδονίζει με την ίδια ταχύτητα με την οποία προσκρούστηκε πάνω του.

Εικόνα 3.10: Τα colliders του αυτοκινήτου με πράσινο περίγραμμα

Ένα ακόμη χαρακτηριστικό ενός collider είναι αν είναι trigger-collider ή όχι. Τα trigger colliders δεν αντιδρούν ούτε επηρεάζονται από την φυσική κίνηση άλλων colliders. Αυτό που κάνουν είναι να ελέγχουν απλά πότε ένα άλλο collider ακουμπάει ή βρίσκεται μέσα τους. Για παράδειγμα στο αυτοκίνητο στην παραπάνω εικόνα το μικρό box-collider στο μπροστινό σώμα του αυτοκινήτου είναι ένα trigger collider. Δεν αντιδράει στην κίνηση του αυτοκινήτου αλλά ελέγχει πότε το μπροστινό μέρος του αυτοκινήτου ακουμπάει με το collider του εδάφους. Αυτό βοηθάει αργότερα στην κωδικοποίηση του παιχνιδιού όταν ορίζουμε πράγματα που δεν πρέπει να γίνονται αν το αυτοκίνητο δεν πατάει στο έδαφος(όπως το να στρίβει στον αέρα).

3.3.3 Rigidbody

Το rigidbody component επιτρέπει το μοντέλο στο οποίο τοποθετείται να κινείται σύμφωνα με τους νόμους της φυσικής. Επιτρέπει δηλαδή την εφαρμογή δυνάμεων και ροπών πάνω στο μοντέλο. Τα rigidbodies απαιτούν τουλάχιστον ένα collider πάνω στο μοντέλο για να προσομοιώσουν ρεαλιστική κίνηση και συγκρούσεις σε ένα μοντέλο.

3.3.4 Camera

Τα camera-components είναι υπεύθυνα για το οτιδήποτε βλέπει ο τελικός χρήστης στην οθόνη του. Είναι στην ουσία το component το οποίο κάνει render τα μοντέλα που βρίσκονται μέσα στο οπτικό του πεδίο. Συχνά χρησιμοποιείται σε συνδυασμό με διάφορα script components για να του δώσουν επιπλέον λειτουργία όπως Depth of Field και Camera Blur.

Εικόνα 3.11: Το camera component του εσωτερικού του αμαξιού.

3.3.5 Reflection Probe

Τα reflection probes είναι μια κάμερα 360° η οποία καταγράφει τα πάντα τριγύρω της. Ύστερα αποθηκεύει αυτή την πληροφορία σε έναν κύβο με 6 υφές(μια για κάθε πλευρά) ο οποίος μπορεί να χρησιμοποιηθεί από αντικείμενα για να αντανακλάσουν με ρεαλισμό τα πράγματα τριγύρω τους. Χρησιμοποιείται στο αυτοκίνητο για να του αποδώσει λίγη έξτρα μεταλλική εμφάνιση χρησιμοποιώντας την αντανάκλαση από το περιβάλλον τριγύρω του.

Εικόνα 3.12: Reflection Probe

3.3.6 Particle System

Το σύστημα σωματιδίων(particle system) χρησιμοποιείται όταν θέλουμε να απεικονίσουμε εφφέ ή πράγματα τα οποία είναι δύσκολο να αναπαρασταθούνε με ένα τριδιάστατο μοντέλο όπως κάποιο υγρό ή κάποια φωτιά τα οποία έχουν ανώμαλες και πολύ απότομες παραμορφώσεις συνέχεια. Αυτό που κάνει το σύστημα σωματιδίων είναι να εκπέμπει από μια πηγή πολλές μικρές εικόνες οι οποίες όταν εκπέμπονται μαζί δημιουργούν την ψευδαίσθηση του απαραίτητου εφφέ. Για παράδειγμα για την εξάτμιση του αυτοκινήτου στην παρακάτω εικόνα χρησιμοποιήθηκε μια απλή δισδιάστατη εικόνα καπνού η οποία εκπέμπεται συνεχόμενα από την εξάτμιση του αυτοκινήτου με πολύ γρήγορο ρυθμό. Πέρα από δισδιάστατες εικόνες το particle system μπορεί να εκπέμπει και τρισδιάστατα μοντέλα αλλά αυτό είναι συνήθως ασύμφορο διότι είναι πολύ πιο βαριά για τον υπολογιστή.

Εικόνα 3.13: Η εξάτμιση του αυτοκινήτου

3.3.7 Trail renderer

Σε περίπτωση που θέλουμε να αφήσουμε ίχνη ή σημάδια από τις περιοχές που περνάει ένα αντικείμενο χρησιμοποιούμε το component trail renderer. Αυτό που κάνει είναι να κρατάει στην μνήμη όλα τα σημεία από τα οποία περνάει το gameobject στο οποίο ειναι τοποθετημένο και να αφήνει πίσω το διδιάστατο texture το οποίο του έχουμε αποδώσει. Στα πλαίσια της πτυχιακής χρησιμοποιήθηκε το trail renderer για τα σημάδια που αφήνει το αυτοκίνητο στον δρόμο όταν κάνει ντρίφτινγκ.

Εικόνα 3.14: Τα skidmarks του αυτοκινήτου

3.4 Κωδικοποίηση παιχνιδιού(scripting)

Η κίνηση του αμαξιού όταν ο χρήστης πατάει τα βελάκια, η περιστροφή των ρόδων του αυτοκινήτου όταν αυτό τρέχει, η αλλαγή του GUI, όλα τα παραπάνω λειτουργούνε με κομμάτια κώδικα τα οποία γράφει ο σχεδιαστής του παιχνιδιού. Τα κείμενα κώδικα(scripts) ίσως είναι το πιο σημαντικό component σε ένα αντικείμενο. Είναι εκείνα τα οποία θα δώσουν την τελική λειτουριγκότητα στο παιχνίδι. Η Unity παρέχει τρείς γλώσσες για τον προγραμματισμό του παιχνιδιού: Javascript, C# και Boo. Επίσης επιτρέπει τον συνδυασμό των παραπάνω.

Final.CSharp TyreRotation

```
1   using UnityEngine;
2   using System.Collections;
3
4   public class TyreRotation : MonoBehaviour {
5
6       public int rawSpeed=1;
7       public NewCaz script;
8       public CheckGrounded ground;
9       Vector3 tyre_vel = new Vector3();
10      //private bool tyre_grounded;
11      // Use this for initialization
12      void Start () {
13          script = transform.root.GetComponent<NewCaz>();
14
15      }
16
17      // Update is called once per frame
18      void FixedUpdate () {
19          //Get the Velocity and check if it's grounded
20          tyre_vel = script.localVel;
21          //tyre_grounded = ground.grounded;
22
23
24          //Apply Turn
25          transform.Rotate(Vector3.right, Time.deltaTime*tyre_vel.x*rawSpeed);
26      }
27  }
28
```

Εικόνα 3.15: Ένα απλό κομμάτι κώδικα για την περιστροφή μιας ρόδας.

Κάθε αρχείο κώδικα αποτελείται από 2 βασικά τμήματα. Την δήλωση μεταβλητών και τις συναρτήσεις. Πέρα από τις συναρτήσεις τις οποίες μπορεί να ορίσει και να δημιουργήσει ο προγραμματιστής η Unity παρέχει μια σειρά από βασικές συναρτήσεις οι οποίες χρησιμοποιούνται στα περισσότερα κείμενα κώδικα:

- Update():

Αυτή η συνάρτηση καλείται μια φορά σε κάθε frame. Χρησιμοποιείται κυρίως για να ελέγξει αλλαγές στην τιμή ή την κατάσταση μιας μεταβλητής όσο το παιχνίδι τρέχει. Για παράδειγμα μια συνηθισμένη χρήση της είναι να ελέγχει πότε ο χρήστης πατάει ένα κουμπί.

- FixedUpdate():

Παρόμοια με την Update τρέχει και αυτή σε κάθε frame. Η διαφορά είναι ότι μεσολαβεί ένα σταθερό χρονικό διάστημα μεταξύ από δύο <<τρεξίματα>> της FixedUpdate. Για αυτόν τον λόγο προτιμάται σε περιπτώσεις που έχουμε να ασκήσουμε αλλαγές στο σύστημα φυσικής του παιχνιδιού όπως το να εφαρμόσουμε δύναμη ή ροπή σε ένα μοντέλο. Το μειονέκτημά της σε σχέση με την Update() είναι ότι είναι πιο βαριά στον επεξεργαστή.

- Start():

Η συγκεκριμένη συνάρτηση τρέχει με το που ενεργοποιηθεί το script. Χρησιμοποιείται κυρίως για αρχικοποίηση μεταβλητών και για καλέσουμε scripts από άλλα scripts.

- OnTriggerEnter():

Καλείται μια φορά όταν το collider που βρίσκεται μέσα στην παρένθεση εισέρχεται στο trigger-zone στο οποίο βρίσκεται το συγκεκριμένο script.

- OnTriggerStay():

Καλείται συνεχόμενα όσο το collider που βρίσκεται μέσα στην παρένθεση παραμένει μέσα στο trigger-zone.

- OnTriggerExit():

Παρόμοια με την OnTriggerEnter η OnTriggerExit() καλείται μια φορά όταν το collider εξέρχεται τελείως από το trigger-zone.

- OnGUI():

Η συγκεκριμένη συνάρτηση επιτρέπει την εμφάνιση δισδιάστατων εικόνων και κουμπιών πάνω στο παιχνίδι όπως ο μετρητής ταχύτητας του αυτοκινήτου και το πόσους γύρους έχει κάνει το αυτοκίνητο.

Final.CSharp

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class tyres_grounded : MonoBehaviour
5  {
6      public bool front_grounded = false;
7
8      void OnCTriggerEnter(Collider other)
9      {
10
11         front_grounded = true;
12
13     }
14     void OnTriggerStay(Collider other2)
15     {
16
17         if (other2.gameObject.tag == "Ground")
18         {
19             front_grounded = true;
20             //Debug.Log("tyres touches Ground");
21         }
22         else
23         {
24             front_grounded = false;
25         }
26
27     }
28
29     void OnTriggerExit(Collider other3)
30     {
31
32         //Debug.Log(other3);
33         front_grounded = false;
34     }
35
36
```

Εικόνα 3.16: Παράδειγμα των τριών OnTrigger συναρτήσεων που ελέγχουν πότε το αυτοκίνητο ακουμπάει ή αφήνει το έδαφος.

Εικόνα 3.17: Το GUI του παιχνιδιού

3.5 Σύστημα φυσικής αυτοκινήτου

Για την κίνηση του αυτοκινήτου στην πίστα χρησιμοποιήθηκαν 2 colliders-κύβοι. Ένα μεγάλος κύβος ο οποίος είναι γύρω από το περίγραμμα του αυτοκινήτου ο οποίος είναι και το βασικό collider που χρησιμοποιείται για να στέκεται το αμάξι πάνω στο έδαφος και για συγκρούσεις με τοίχους και άλλα αμάξια.Το δεύτερο collider είναι ένας μικρότερος κύβος που βρίσκεται κάτω από το καπό του αυτοκινήτου. Και τα 2 collider χρησιμοποιούνε ακτίνες (raycast) για να ελέγχουν αν το αυτοκίνητο ακουμπάει στο έδαφος. Τα raycast είναι ένα διάνυσμα με συγκεκριμένο μήκος αρχή και τέλος στον τριδιάστατο χώρο του παιχνιδιού τα οποία εκπέμπονται σε κάθε καρέ και ελέγχουν ποιά collider διαπερνούνε. Στην δική μας περίπτωση με το αυτοκίνητο εκπέμπονται δύο ακτίνες, μια από το κέντρο κάθε collider. Το μπροστινό collider ελέγχει αν οι ρόδες ακουμπάνε στο έδαφος με την μεταβλητή “TyresGrounded” και το μεγάλο collider αν ακουμπάει γενικά οποιοδήποτε σημείο του αυτοκινήτου στο έδαφος με την μεταβλητή IsGrounded. Με αυτές τις 2 μεταβλητές δίνεται η δυνατότητα για πιο καλό έλεγχο των περιπτώσεων κίνησης του αυτοκινήτου.

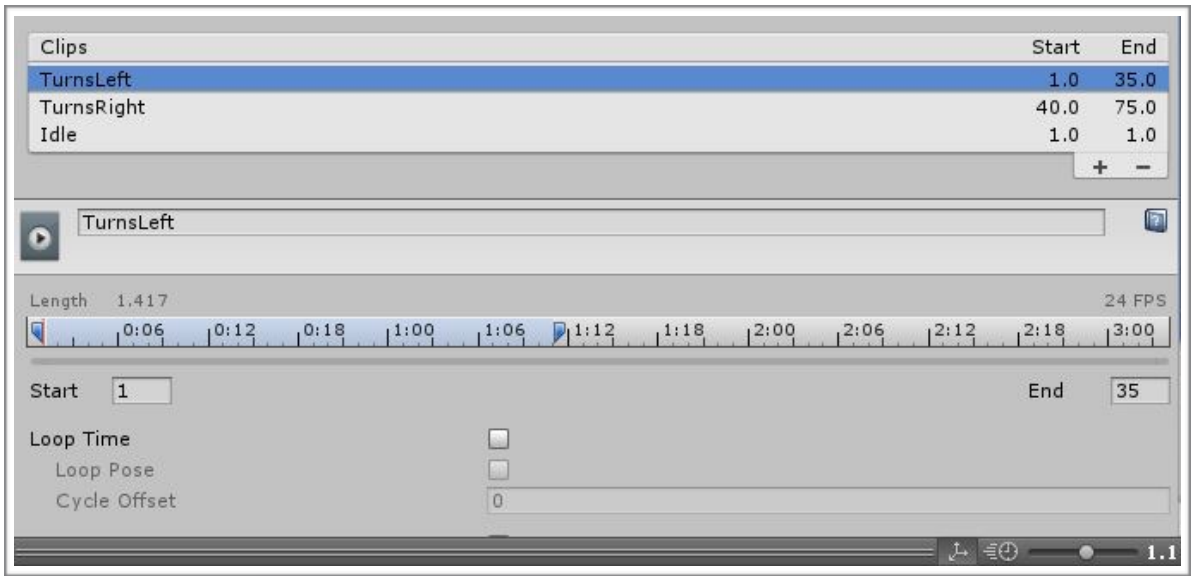
Σε συνδυασμό με τα 2 colliders χρησιμοποιείται ένα rigidbody το οποίο όπως έχει προαναφερθεί είναι το component που δίνει ρεαλισμό στην φυσική του αυτοκινήτου. Δύο είναι τα βασικά χαρακτηριστικά ενός rigidbody. Το πρώτο είναι η μάζα και είναι εκείνο που επηρεάζει το βάρος του αντικειμένου καθώς και το πόσο πολύ επηρεάζεται από συγκρούσεις με αντικείμενα διαφορετικής μάζας. Όσο μεγαλύτερη μάζα έχει ενα αντικείμενο τόσο πιο πολύ επηρεάζει κάποιο αντικείμενο με μικρότερη. Για παράδειγμα όταν το αυτοκίνητο με μάζα 1000 χτυπάει ένα βαρέλι με μάζα 200 τότε το βαρέλι επηρεάζει ελάχιστα την κίνηση του αυτοκινήτου ενώ το αυτοκίνητο εκσφενδονίζει το βαρέλι μπροστά. Το δεύτερο χαρακτηριστικό λέγεται drag και θα μπορούσε να περιγραφεί ως η τριβή του αντικειμένου με άλλα. Όσο μεγαλύτερο είναι το drag τόσο πιο ισχυρή η δύναμη της τριβής και τόσο περισσότερη δύναμη χρειάζεται να του τοποθετηθεί για να αρχίσει να κινείται.

Εικόνα 3.18: Η μάζα του βαρελιού είναι πολύ μικρότερη άρα δεν εμποδίζει κατά πολύ την πορεία του αμαξιού και παρασέρνεται.

3.6 Σύστημα Animation Οδηγού

Αποκοπή animation clip

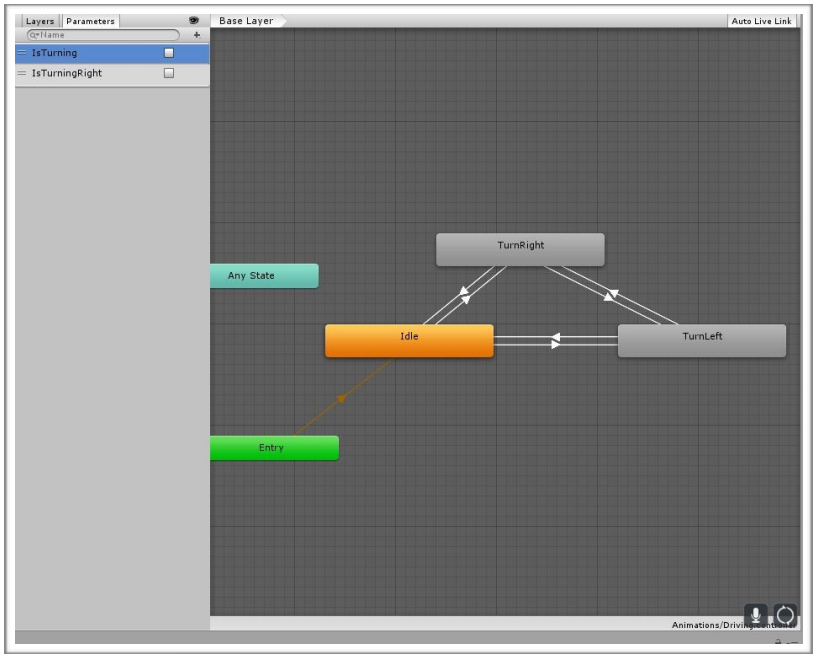
Αφού έχουμε δημιουργήσει όλα τα animations που χρειαζόμαστε για τον οδηγό του αυτοκινήτου σε κάποιο εξωτερικό πρόγραμμα, στην συνέχεια χωρίζουμε τα animation σε συγκεκριμένα μήκη από καρέ(frame ranges). Στην προκειμένη περίπτωση έχουμε 3 κινήσεις που έχουμε δημιουργήσει για τον χαρακτήρα στο Maya οπότε αφού έχουμε βάλει το μοντέλο μέσα στην Unity πρέπει να κόψουμε αυτά τα 3 animations στα frames που αντιστοιχούν. Οι κινήσεις οι οποίες θα κάνει ο οδηγός είναι:
-Να κάθεται ακίνητος (idle)
-Να στρίβει το τιμόνι δεξιά(TurnsRight)
-Να στρίβει το τιμόνι αριστερά(TurnsLeft)
Καθένα από τις παραπάνω 3 κινήσεις εμπεριέχεται σε συγκεκριμένα καρέ στο animation-slider του μοντέλου οπότε σκοπός μας είναι να κρατήσουμε μόνο εκείνα τα καρέ τα οποία περιέχουν την συγκεκριμένη κίνηση.



Εικόνα 3.19: Κόβωντας τα ξεχωριστά animations για κάθε κίνηση.

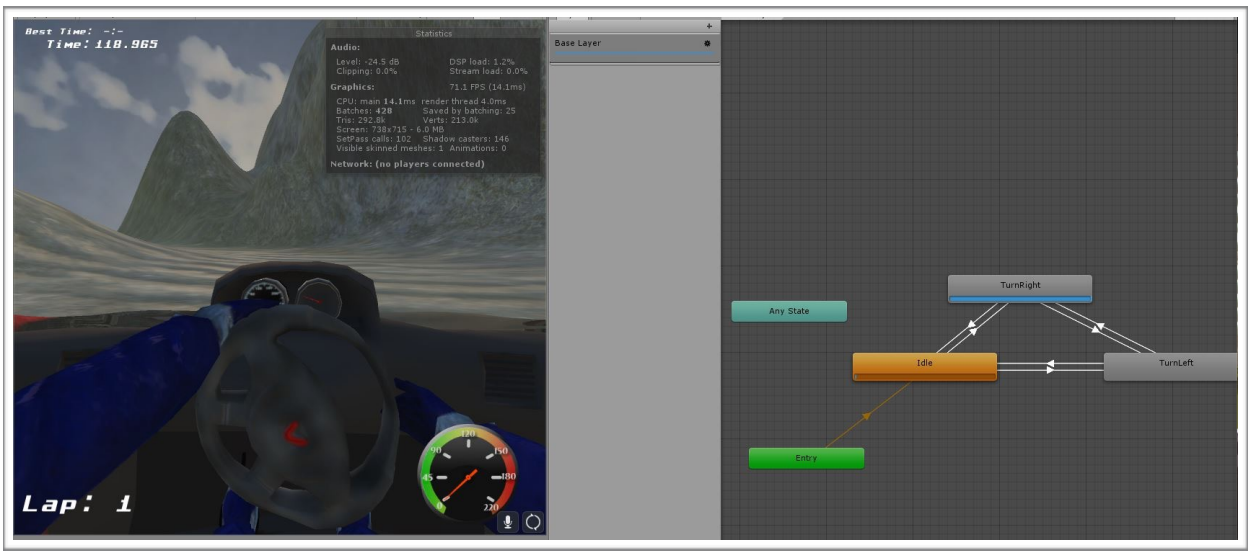
Παράθυρο Animator

Αφού λοιπόν έχουμε ξεχωρίσει τα animations που θέλουμε να κάνει ο χαρακτήρας μας στυν συνέχεια πρέπει να βρούμε έναν τρόπο ώστε να γίνεται μια ομαλή μετάβαση από το ένα στο άλλο ανάλογα με το πλήκτρο που πατάει ο χρήστης. Για να λυθεί το συγκεκριμένο πρόβλημα η Unity παρέχει το παράθυρο Animator. Το animator δείχνει τις μεταβάσεις από το ένα animation στο άλλο στην μορφή διαγράμματος καταστάσεων. Κάθε animation που δημιουργήσαμε στο προηγούμενο βήμα αποτελεί και μια τέτοια κατάσταση και το animator μας δίνει την δυνατότητα να θέσουμε τους όρους βάσει των οποίων θα μπορέσει να μεταβεί το μοντέλο μας από την μια κατάσταση στην άλλη.

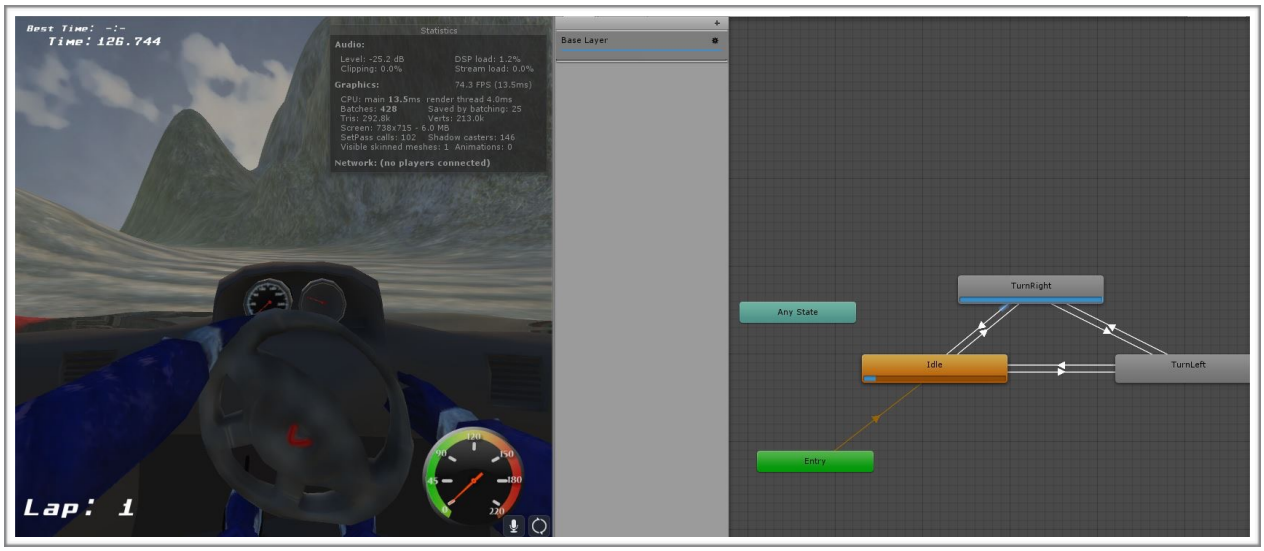


Εικόνα 3.20: Το παράθυρο Animator

Όπως βλέπουμε και στην παραπάνω εικόνα το παράθυρο περιέχει 2 μέρη. Το αριστερό το οποίο έχει τις παραμέτρους σύμφωνα με τις οποίες αλλάζουν οι καταστάσεις, και το δεξιό στο οποίο βρίσκεται το διάγραμμα καταστάσεων το οποίο περιέχει τα 3 animations που δημιουργήσαμε. Οι παράμετροι μπορεί να είναι τύπου Float,Int,Bool ή Trigger ανάλογα με το αποτέλεσμα που θέλουμε να πετύχουμε. Οι 2 παράμετροι στην παραπάνω περίπτωση είναι δύο μεταβλητές σωστού/λάθους(boolean). Όταν ο χρήστης πατάει το αριστερό βέλος η μεταβλητή isTurning γίνεται αληθής και γίνεται η μετάβαση από την Idle ή την TurnsRight προς την TurnsLeft. Όταν ο χρήστης πατάει το δεξιό βέλος η μεταβλητή isTurningRight γίνεται αληθής και τότε ο χρήστης μεταβαίνει στην κατάσταση TurnsRight. Όταν ο χρήστης δεν πατάει τίποτα τότε και οι δύο καταστάσεις είναι ψευδής άρα πηγαίνει στην κατάσταση Idle.



Εικόνα 3.21: Μετάβαση στην κατάσταση TurnsRight



Εικόνα 3.22:Επιστροφή στην Idle

3.7 Δημιουργία GUI παιχνιδιού

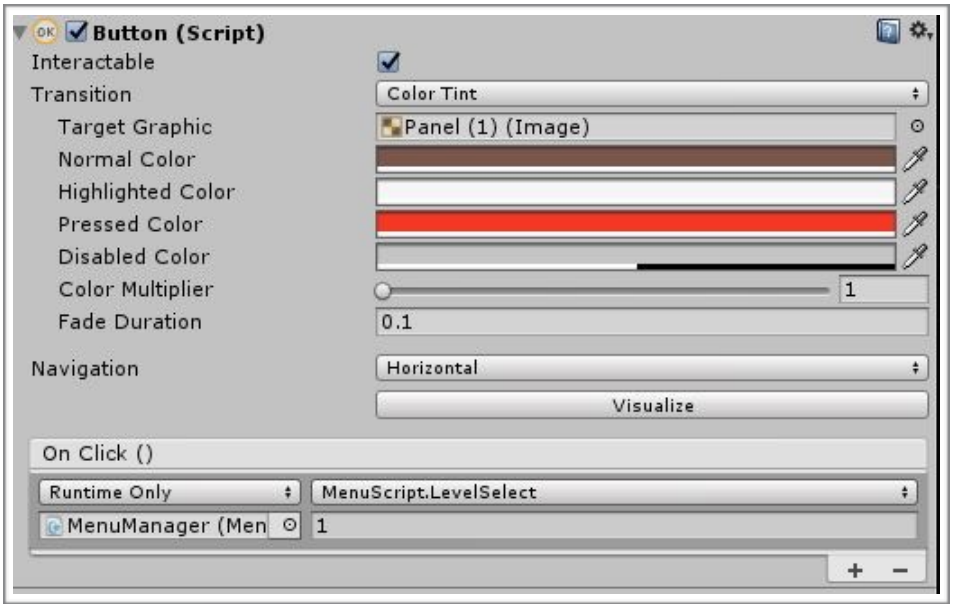
Το GUI(graphical user interface) αποτελεί όλα τα δισδιάστατα γραφικά που φαίνονται πάνω στην οθόνη του χρήστη και του δίνουν διάφορες πληροφορίες για το παιχνίδι όπως σε ποιόν γύρο βρίσκεται και με πόση ταχύτητα τρέχει το αυτοκίνητο. Για τον σχεδιασμό των στοιχείων του GUI χρησιμοποιήθηκε το Photoshop.

Εικόνα 3.23: Σχεδιασμός ταχύμετρου στο Photoshop.

Εικόνα 3.24: Πρώτη όψη του menu.

Για την εμφάνιση των γραφικών στην οθόνη χρησιμοποιήθηκε η OnGUI() συνάρτηση και το Canvas σύστημα της unity. Το canvas είναι εκείνο το GameObject μέσα στο οποίο βρίσκονται όλα τα UI στοιχεία του παιχνιδιού. Τα πιο συνηθισμένα UI elements που χρησιμοποιούνται είναι εικόνες, κείμενο και κουμπιά. Πέρα από την εμφάνιση πάνω στην οθόνη του χρήστη το canvas σύστημα μπορεί να χρησιμοποιηθεί και για να τοποθετηθούν δισδιάστατες εικόνες στον τριδιάστατο χώρο όπως γίνεται και στην περίπτωση του βασικού μενού του παιχνιδιού.

Τα σημαντικότερα στοιχεία του canvas είναι τα κουμπιά(buttons) τα οποία έχουν πάνω τους μια On Click() επιλογή στην οποία μπορούμε να τοποθετήσουμε μια δική μας συνάρτηση ώστε να τρέξει. Στην δική μας περίπτωση τρέχει την συνάρτηση levelselect. Η levelselect παίρνει σαν παράμετρο έναν ακέραιο αριθμό και φορτώνει την πίστα που αντιστοιχεί σε αυτό τον αριθμό. Για να δημιουργήσουμε λοιπόν ένα λειτουργικό μενού δημιουργούμε το canvas system με 4 κουμπιά θέτουμε ως συνάρτηση σε κάθε κουμπί την levelload() και περνάμε σε κάθε κουμπί τον αριθμό που αντιστοιχεί στην πίστα που θέλουμε να φορτώσει. Ο αριθμός της πίστας ορίζεται στα build settings του παιχνιδιού.



Εικόνα 3.25: Οι επιλογές του κουμπιού ενός canvas

3.8 Δημιουργία Game Logic.

Μέχρι στιγμής στο βασικό παιχνίδι έχουμε καταφέρει να αποδώσουμε τους νόμους της φυσικής του αυτοκινήτου καθώς και τις κατάλληλες κινήσεις στον οδηγό για τον επιπλέον ρεαλισμό. Για να γίνει όμως το παιχνίδι ένα παιχνίδι αγώνων πρέπει να δημιουργηθεί η επιπλέον λογική η οποία θα ελέγχει πράγματα όπως:

- Έχει το αμάξι τερματίσει τον αγώνα;
- Ο αγώνας να ξεκινάει αφού γίνει μια εναέρια εισαγωγή της πίστας.
- Έχει κάνει το αμάξι ολόκληρο τον γύρο ή έκλεψε;

Με άλλα λόγια θα δημιουργήσουμε ένα σύστημα το οποίο θα διασφαλίσει ότι το παιχνίδι μας δουλεύει όπως το θέλουμε και είναι λειτουργικό.

3.8.1 Προεπισκόπηση πίστας και αρχή αγώνα

Όταν ο χρήστης μπαίνει για πρώτη φορά στην πίστα θέλουμε να του δωθεί η ικανότητα να πάρει μια γεύση από αυτήν και να μην ξεκινήσει κατευθείαν ο αγώνας. Έτσι στην αρχή αφού φορτώσει η πίστα απενεργοποιούμε το script το οποίο δίνει την δυνατότητα στον χρήστη να ελέγχει το αμάξι και αντίθετα τρέχουμε ένα άλλο το οποίο κάνει μια προεπισκόπηση διαφόρων σημείων της πίστας. Για να πετύχουμε την συγκεκριμένη λειτουργία ο πιο απλός τρόπος είναι η δημιουργία άδειων gameobjects στην πίστα από τα οποία μπορούμε να χρησιμοποιήσουμε την πληροφορία τοποθεσίας και περιστροφής και να τηλεμεταφέρουμε την κάμερα σε αυτά μετά από κάποιο χρονικό διάστημα. Στην συνέχεια όταν ο χρήστης δεν θέλει να συνεχίσει να βλέπει την πίστα αρκεί να πατήσει ένα κουμπί(στην προκειμένη περίπτωση το a) το οποίο σταματά την λειτουργία της προεπισκόπησης. Όταν συμβεί αυτό τότε απενεργοποιείται η κάμερα που κάνει προεπισκόπηση και ενεργοποιείται η βασική κάμερα του αυτοκινήτου. Στην συνέχεια εμφανίζονται τα δισδιάστατα γραφικά από ένα φανάρι το οποίο όταν ανάψει πράσινο σηματοδοτεί την αρχή του αγώνα. Με το που γίνει αυτό ενεργοποιείται το script που δίνει λειτουργία στο αυτοκίνητο και ο αγώνας ξεκινάει.

Εικόνα 3.26: Η εμφάνιση του φαναριού πριν το ξεκίνημα του αγώνα

Ένα πολύ βασικό εργαλείο για την επίτευξη των παραπάνω λειτουργιών είναι οι συναρτήσεις coroutines. Η κύρια χρησιμότητά τους είναι ότι σε αντίθεση με τις απλές συναρτήσεις οι οποίες τρέχουν και τελειώνουν σε κάθε frame που τρέχει το παιχνίδι, οι coroutines μπορούν να τρέχουν για έναν μεγάλο αριθμό από frames. Αυτό μας επιτρέπει να τους δίνουμε λειτουργίες όπως να τηλεμεταφέρουν την κάμερα προεπισκόπησης σε ένα σημείο μετά από ένα χρονικό διάστημα 3 δευτερολέπτων ώστε ο χρήστης να μπορεί να

αναλύσει το σημείο που βλέπει. Ή στην αρχή του αγώνα η εμφάνιση των χρωμάτων του φαναριού να γίνεται με κάποια καθυστέρηση και όχι όλα ταυτόχρονα. Για να το πετύχουν αυτό οι coroutines χρησιμοποιούν σε κάποιο σημείο στο εσωτερικό τους το keyword yield. Όταν η συνάρτηση φτάνει στο yield τότε σταματάει και συνεχίζει στο επόμενο frame. Το yield μπορεί να χρησιμοποιηθεί σε συνδυασμό με την κλάση WaitForSeconds ώστε να εισαχθεί μια καθυστέρηση στην συνέχεια του τρεξίματος της συνάρτησης.

```
IEnumerator Fanari()
{
    fanaria[1].enabled = true;
    yield return new WaitForSeconds(1);
    fanaria[2].enabled = true;
    fanaria[1].transform.parent.GetComponent<AudioSource>().Play();
    yield return new WaitForSeconds(1);
    fanaria[3].enabled = true;
    fanaria[1].transform.parent.GetComponent<AudioSource>().Play();
    yield return new WaitForSeconds(1);
    fanaria[4].enabled = true;
```

Εικόνα 3.27: Η coroutine που εμφανίζει το φανάρι πριν τον αγώνα

Όπως βλέπουμε και στο παραπάνω παράδειγμα για την εμφάνιση του φαναριού χρησιμοποιείται η coroutine Fanari(). Στον πίνακα fanaria έχουμε εισάγει τα δισδιάστατα γραφικά του φαναριού. Κάθε δευτερόλεπτο που περνάει εμφανίζεται το κατάλληλο γραφικό και παίζεται ένας ήχος. Η yield return new WaitForSeconds(1) εμποδίζει την συνάρτηση να τα τρέξει όλα μαζί ταυτόχρονα και εισάγει καθυστέρηση ενός δευτερολέπτου.

3.8.2 Σύστημα ελέγχου δίκαιου αγώνα

Ο πιο απλός τρόπος για να προσθέσουμε ένα σύστημα γύρων στο παιχνίδι είναι να έχουμε μια μεταβλητή σε κάθε αμάξι που λέγεται currentLap και κάθε φορά που περνάει από την γραμμή τερματισμού να την αυξάνουμε κατά ένα. Τότε όμως τίθεται το θέμα του ελέγχου του χρήστη για το αν έχει όντως κάνει τον γύρο της πίστας ή απλά μπαινοβγαίνει στην γραμμή τερματισμού για να αυξάνει τον τωρινό του γύρο. Για να λυθεί το συγκεκριμένο πρόβλημα χρησιμοποιήθηκε μια λύση που χρησιμοποιεί checkpoints από τα οποία πρέπει να περάσει ο χρήστης ώστε ο αγώνας να είναι δίκαιος. Η υλοποίησή της έχει ως εξής:
Το script κάθε αυτοκινήτου περιέχει έναν πίνακα μεταβλητών bool, με μέγεθος ίσο με τα checkpoints της κάθε πίστας. Όταν το αυτοκίνητο περνάει από ένα checkpoint τότε το στοιχείο στον πίνακα με δείκτη τον αριθμό του checkpoint γίνεται true. Όταν το αμάξι περνάει από την γραμμή τερματισμού θα πρέπει να έχει περάσει από όλα τα checkpoints δηλαδή να είναι όλες οι τιμές στον πίνακα true αλλιώς παραμένει στον ίδιο γύρο. Ένα πρόβλημα που δεν λύνει η παραπάνω λογική είναι η περίπτωση το αμάξι να περάσει από όλα τα checkpoints αλλά μετά να πάρει την πίστα ανάποδα και να περάσει από την γραμμή τερματισμού. Στην περίπτωση του δικού μας παιχνιδιού αυτό δεν αποτελεί πρόβλημα γιατί υπάρχει ένα σημείο της πίστας από το οποίο δεν μπορείς να την πάρεις ανάποδα. Μια λύση σε περίπτωση πιθανής επέκτασης του παιχνιδιού στο μέλλον θα ήταν όταν το αμάξι ξαναπερνάει από ένα checkpoint έχοντας ήδη περάσει το επόμενό του (τρέχει ανάποδα δηλαδή) τότε η μεταβλητή που αντιστοιχεί σε εκείνο το checkpoint και όλες οι επόμενες να γίνονται false ώστε όταν γυρίσει στην αρχή να μην του προστεθεί γύρος.

Εικόνα 3.28: Τα checkpoints με πράσινο περίγραμμα.

Πέρα από την χρησιμότητά τους να ελέγχουν αν ο αγώνας είναι δίκαιος τα checkpoints χρησιμοποιούνται και για το respawn των αυτοκινήτων σε περίπτωση που καταστραφούν. Κάθε αυτοκίνητο έχει μια μεταβλητή με όνομα currentCheckpoint που κρατάει στην μνήμη το τελευταίο checkpoint από το οποίο πέρασε. Έτσι σε περίπτωση που το αυτοκίνητο πέσει στην θάλασσα ή ο χρήστης το καταστρέψει ο ίδιος γιατί έχει κολλήσει σε κάποιο μέρος της πίστας το αυτοκίνητο θα ξαναεμφανιστεί σε ένα σημείο κοντά στο checkpoint και όχι στην γραμμή εκκίνησης. Για την υλοποίηση του παραπάνω υπάρχει ένας πίνακας με κενά gameobjects ίσα σε αριθμό όσα και τα checkpoints. Κάθε gameobject τοποθετείται σε ένα σημείο δίπλα στα checkpoints και αντιστοιχείται με αυτό. Όταν το αυτοκίνητο καταστραφεί το παιχνίδι ελέγχει από ποιό checkpoint πέρασε το αυτοκίνητο τελευταία φορά. Στην συνέχεια τηλεμεταφέρει το αυτοκίνητο στο κενό gameobject που αντιστοιχεί στο checkpoint αυτό χρησιμοποιώντας τις πληροφορίες τοποθεσίας και περιστροφής του gameobject.

```
//Reset the car speed to 0. And wait 5 seconds until it respawns somewhere.
car.GetComponent<Rigidbody>().velocity = new Vector3(0, 0, 0);
yield return new WaitForSeconds(3.0f);

car.transform.position = carscript.respawnPointArray[carscript.currentCheckpoint].position;
car.transform.rotation = carscript.respawnPointArray[carscript.currentCheckpoint].rotation;
```

Εικόνα 3.29: Ακινητοποίηση του αυτοκινήτου και τηλεμεταφορά του στο σημείο δίπλα από το current checkpoint

3.9 Ήχοι και μουσική

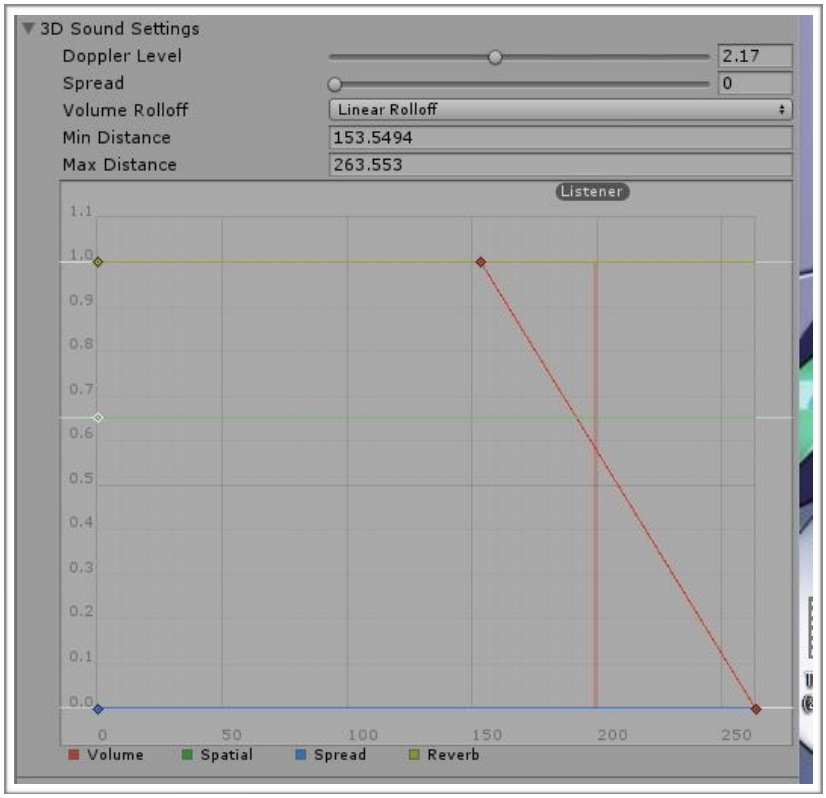
Ένα βασικό στοιχείο κάθε βιντεοπαιχνιδιού για να το κάνει πιο ενδιαφέρον είναι η ήχοι που αυτό το παιχνίδι θα παίζει κατά την διάρκειά του. Αυτοί μπορεί να είναι σε μορφή μουσικής που θα παίζει όσο ο χρήστης παίζει την πίστα ή σε μορφή απλών ήχων όπως για παράδειγμα η μηχανή ενός αυτοκινήτου. Παρακάτω αναλύονται οι επιλογές που έγιναν για τους εκάστοτε ήχους του παιχνιδιού στα πλαίσια της παρούσας εργασίας.

3.9.1 Audio Source και Audio Listener

Το component Audio Source είναι υπεύθυνο για τον οποιονδήποτε ήχο ακούει ο χρήστης στην σκηνή. Σε κάθε component audio source προστίθεται ένα audioclip, δηλαδή ένα αρχείο μορφής ήχου το οποίο θα παίζεται. Ανάλογα με την περίσταση μπορούμε είτε τον ήχο να τον κάνουμε να ακούγεται συνέχεια ως ambient ήχος τσεκάροντας την επιλογή Loop είτε να ακούγεται σε συγκεκριμένες περιπτώσεις ενεργοποιώντας τον με κάποιο κομμάτι κώδικα. Σημαντική δυνατότητα των audio sources είναι ότι επιτρέπουν την προσομοίωση και διδιάστατου αλλά και τριδιάστατου ήχου με ένα slider με όνομα <<Spatial Blend>>. Όσο πιο δεξιά τοποθετηθεί το slider τόσο πιο τριδιάστατος θα είναι ο ήχος. Τριδιάστατος ήχος σημαίνει ότι ανάλογα με την απομάκρυνση που έχει το audio-listener από το audio source τόσο θα μειώνεται ο ήχος ενώ διδιάστατος ήχος θα ακούγεται παντού το ίδιο. Στην παρακάτω εικόνα βλέπουμε ένα audio source πώς φαίνεται στην Unity. Αποτελείται από 2 ομόκεντρες σφαίρες. Η εσωτερική ορίζει την περιοχή στην οποία ο ήχος θα ακούγεται στο μέγιστο και η εξωτερική το μέγιστο όριο στο οποίο θα ακούγεται ο ήχος. Εφόσον ο ήχος είναι τριδιάστατος όταν ο χρήστης θα βρίσκεται στην εξωτερική αλλά όχι στην εσωτερική σφαίρα τότε θα ακούει τον ήχο σε λιγότερη ένταση. Σε αντίθετη περίπτωση θα άκουγε την ίδια ένταση εφόσον βρισκόταν μέσα σε οποιαδήποτε σφαίρα. Στην προκειμένη περίπτωση ο χρήστης θα ακούει τον ήχο του ωκεανού περισσότερο όταν βρίσκεται πάνω στην γέφυρα και λιγότερο όταν την πλησιάζει ή φεύγει μακρυά της.

Εικόνα 3.30: Ένα audio source με ήχο ωκεανού(μπλέ περιγράμματα)

Το audio listener είναι το component το οποίο <<ακούει>> τους ήχους που παράγονται στην σκηνή και τοποθετείται συνήθως πάνω στην κάμερα του παίκτη. Όταν ένα audio listener βρίσκεται μέσα στην περιοχή ενός audio source τότε ακούει τον ήχο που αυτό παράγει. Επιπλέον κάθε audiosource component έχει και το παρακάτω διάγραμμα το ανανεώνεται συνεχώς και δείχνει σε τί απόσταση βρίσκεται το listener από το audio source και πώς αυτό επηρεάζει την ένταση του ήχου.



Εικόνα 3.31: Η ένταση του ήχου μειώνεται λόγω απόστασης Listener-Audiosource

3.9.2 Μουσική

Για την μουσική που ακούει ο χρήστης ενώ τρέχει στον αγώνα επιλέχτηκε μια δωρεάν λίστα με τραγούδια που βρέθηκε από το Asset Store με όνομα <<Metal Mayhem Music Pack>>. Στην συνέχεια δημιουργήθηκε ένα script το οποίο δημιουργεί έναν πίνακα με όλα τα τραγούδια αυτά και παίζει στην τύχη κάποιο. Όλα τα τραγούδια είναι διδιάστατοι ήχοι μιας και παράγονται από το αυτοκίνητο και όχι από κάποιο σημείο της πίστας. Είναι σημαντικό να χρησιμοποιούνται τριδιάστατοι ήχοι μόνο όποτε χρειάζεται για ρεαλισμό γιατί χρησιμοποιούν περισσότερους πόρους από τους διδιάστατους.

3.9.3 Ήχος μηχανής αυτοκινήτου

Ενδιαφέρουσα περίπτωση ήχου αποτελεί εκείνος της μηχανής του αυτοκινήτου. Ο ήχος της μηχανής θέλουμε να αλλάζει ανάλογα με το πόσο γρήγορα πηγαίνει το αυτοκίνητο για να προσομοιώσει με πιο ρεαλιστικό τρόπο έναν πραγματικό κινητήρα. Για να πετύχουμε αυτό το εφφέ δημιουργήθηκε ένα script το οποίο επηρεάζει τον τόννο του ήχου (pitch) ανάλογα με την ταχύτητα που τρέχει το αυτοκίνητο. Το συγκεκριμένο script χρησιμοποιεί το audio source <<Acceleration High>> που παρέχεται στα standards assets της Unity. Στην συνέχεια διαιρεί την τωρινή ταχύτητα του αυτοκινήτου με την μέγιστη δυνατή ώστε να δημιουργηθεί ένα ποσοστό πόσο κοντά είναι η ταχύτητα στο μέγιστο. Ανάλογα με το ποσοστό αυτό αλλάζει την παράμετρο pitch του audio source.

3.10 Δημιουργία Multiplayer

Αφού έχουμε μια ικανοποιητική κατάσταση του παιχνιδιού που δουλεύει σε έναν υπολογιστή, το επόμενο βήμα είναι να προσθέσουμε το multiplayer κομμάτι του παιχνιδιού. Αν και σε γενικές γραμμές ο προγραμματισμός των multiplayer στοιχείων των παιχνιδιών είναι συχνά πολύπλοκος η Unity παρέχει το HLAPI(high level API) για να διευκολύνει την γρήγορη δημιουργία multiplayer στοιχειών προσφέροντας γρήγορες λύσεις στα πιο συνηθή προβλήματα. Ας δούμε λοιπόν πώς δουλεύει το multiplayer στην Unity.

3.10.1 Βασικές Έννοιες

Η όλη προϋπόθεση για να δημιουργηθεί ένα multiplayer παιχνίδι είναι να δημιουργηθεί μια σύνδεση μεταξύ ενός διακομιστή(server) και ενός ή περισσότερων πελατών(clients) οι οποίοι θα τρέχουν ταυτόχρονα το παιχνίδι και θα στέλνουν τα κατάλληλα μηνύματα μεταξύ τους ώστε να συγχρονίζονται. Ο διακομιστής είναι εκείνος ο οποίος είναι υπεύθυνος για αυτόν τον συγχρονισμό. Κάθε πελάτης στέλνει τις τοπικές αλλαγές που έγιναν σε αυτόν, τις στέλνει στον server και ύστερα ο server ενημερώνει τους υπόλοιπους πελάτες για την αλλαγή. Στην περίπτωση της Unity ο υπολογιστής-διακομιστής λέγεται οικοδεσπότης(host). Ο υπολογιστής-οικοδεσπότης δουλεύει και ως οικοδεσπότης αλλά και ως πελάτης(σε περίπτωση που θέλει να παίξει αυτός που κάνει host το παιχνίδι).
Μια άλλη βασική έννοια είναι του τοπικού παίκτη και της τοπικής εξουσίας. Κάθε παίκτης έχει ένα ή περισσότερα αντικείμενα τα οποία μόνο εκείνος μπορεί να τα ελέγχει. Με αυτόν τον τρόπο δημιουργείται η έννοια της τοπικής εξουσίας. Όταν προστίθεται ένας παίκτης και ταυτίζεται με ένα αντικείμενο τοπικής εξουσίας τότε έχουμε την έννοια του τοπικού παίκτη. Πέρα από την τοπική εξουσία υπάρχει και η server-εξουσία η οποία υπάρχει για αντικείμενα τα οποία δεν ελέγχουν οι παίκτες όπως για παράδειγμα αντικείμενα στην πίστα τα οποία μπορούν να μετακινηθούν αν ο παίκτης χτυπήσει πάνω τους.

3.10.2 Εισαγωγή βασικών multiplayer-components.

Για να μπορέσει το παιχνίδι να γίνει multiplayer χρειάζεται η κατάλληλη προετοιμασία της σκηνής. Το πρώτο βήμα είναι η δημιουργία ενός άδειου GameObject στην σκηνή στο οποίο θα τοποθετηθεί ένα NetworkManager και ένα NetworkManagerHud component. Το Network manager είναι υπεύθυνο στο να διευθύνει όλες τις διαδικτυακές καταστάσεις του παιχνιδιου ενώ το NetworkManagerHud δημιουργεί ένα απλό menu όταν ξεκινάει το παιχνίδι που επιτρέπει στον χρήστη να δημιουργήσει server ή να συνδεθεί σε έναν server ως client εισάγοντας την IP του server. Όταν ο χρήστης συνδεθεί τότε ο network manager εμφανίζει στην σκηνή έναν κλώνο του prefab που έχει τοποθετηθεί στο πεδίο Spawn Info του NetworkManager.

Εικόνα 3.32: Το menu πριν την δημιουργία multiplayer παιχνιδιού.

Εικόνα X: Η επιλογή Player Prefab.

3.10.3 Συγχρονισμός μεταβλητών

Όπως προαναφέρθηκε η πιο σημαντική λειτουργία του HLAPI είναι να συγχρονίζει τις μεταβλητές μεταξύ των clients και του server, αλλά ποιές είναι οι μεταβλητές αυτές; Επειδή το bandwidth και η ταχύτητα μεταφοράς δεδομένων σε μια σύνδεση μεταξύ υπολογιστών είναι περιορισμένα, πρέπει να φροντίσουμε να στέλνουμε μόνο τις απολύτως απαραίτητες μεταβλητές. Για παράδειγμα είναι σημαντικό όλοι οι clients να ανανεώνουν συνέχεια την τοποθεσία που βρίσκεται ένα κουτί στην πίστα το οποίο μπορεί να μετακινηθεί, ενώ αντίθετα ένα δέντρο δεν υπάρχει λόγος να στέλνεται η τοποθεσία του συνέχεια στο δίκτυο γιατί δεν πρόκειτε να μετακινηθεί ποτέ. Η ανανέωση τοποθεσίας και περιστροφής μεταξύ client και server είναι η πιο βασική μεταβλητή που πρέπει να συγχρονίζεται μεταξύ των υπολογιστών ώστε η αλλαγή τοποθεσίας που γίνεται σε έναν υπολογιστή να συμβαίνει και στους υπόλοιπους.

Για να επιτευχθεί ο συγχρονισμός των μεταβλητών που επιθυμεί ο σχεδιαστής το HLAPI της Unity απαιτεί λίγα απλά βήματα.

```
using UnityEngine.Networking;
using UnityEngine.UI;


public class PlayerID : NetworkBehaviour {

    //Sync Name Over Network
    [SyncVar]
    public string playerUniqueName;
    [SyncVar]
    public int playerNumber;
```

Εικόνα 3.34: NetworkBehaviour script

1) Στην αρχή του script να κάνει import την βιβλιοθήκη UnityEngine.Networking
2) Να αλλάξει το script στην αρχή από MonoBehaviour σε NetworkBehaviour.
3) Να δηλώσει την μεταβλητή που θέλει να συγχρονίζεται ως [SyncVar].

Όταν η μεταβλητή δηλωθεί ως syncvar τότε αυτόματα όποτε αλλάζει στον server θα αλλάζει και σε όλους τους clients. Για να ενημερώσει ο client τον server για μια τοπική αλλαγή θα πρέπει να χρησιμοποιήσει μια ειδική command συνάρτηση. Οι command συναρτήσεις πρέπει πριν την δήλωσή τους να έχουν το κείμενο [Command] και το όνομά τους να ξεκινάει από τα γράμματα Cmd.

3.10.4 Αρχικοποίηση και ταυτοποίηση παίκτη

Για να δούμε όλα τα παραπάνω πώς δουλεύουν στην πράξη ας κάνουμε μια ανάλυση στο πώς πετυχαίνεται η αρχικοποίηση ενός παίκτη του παιχνιδιού που σχεδιάστηκε στα πλαίσια της πτυχιακής. Στην αρχή δηλώνουμε τις τρεις SyncVar μεταβλητές που θέλουμε να συγχρονίζονται από τον server στους clients με ονόματα playerUniqueName, playerNumber και wantsToRace. Η πρώτη είναι μια string μεταβλητή με το όνομα του παίκτη, η δεύτερη μια ακέραιη με τον αριθμό του παίκτη και η τρίτη μια boolean μεταβλητή που δηλώνει ότι ο παίκτης είναι έτοιμος να τρέξει. Και οι τρεις παραπάνω μεταβλητές θέλουμε να ενημερώνονται στον server από τους clients. Χρησιμοποιούμε λοιπόν τις command συναρτήσεις της παρακάτω εικόνας με τις οποίες κάνουμε update τον server για τις τιμές των μεταβλητών.

```
[Command]
void CmdTellServerMyName(string name, int number)
{
    playerUniqueName = name;
    playerNumber = number;

}
```

Εικόνα 3.35: Μια command συνάρτηση που αλλάζει τις μεταβλητές playerUniqueName και PlayerNumber από τον client στον server.

```
[Command]
void CmdTellServerIWantToRace()
{
    wantsToRace = true;
}

[Command]
void CmdTellServerMyName(string name, int number)
{
    playerUniqueName = name;
    playerNumber = number;

}
```

Εικόνα Χ: Οι εντολές command που ενημερώνουν τον server.

Μια επίσης χρήσιμη συνάρτηση είναι η OnStartLocalPlayer() η οποία καλείται αυτόματα όταν ένας χρήστης συνδεθεί στο παιχνίδι. Μέσα σε αυτήν τρέχουμε τις συναρτήσεις GetNetIdentity() και SetIdentity(). Η πρώτη βλέπει ποιά είναι η ταυτότητα του παίκτη στο δίκτυο και την λέει στον server. Η δεύτερη χρησιμοποιείται για να συγχρονίσει την ταυτότητα του κάθε παίκτη με όλους τους υπόλοιπους. Με αυτόν τον τρόπο αποφεύγονται περιπτώσεις που ο ένας παίκτης να βλέπει τον εαυτό του σαν Player1 και οι άλλοι να τον

βλέπουν σαν Player2. Τέλος η SetIdentity φροντίζει ώστε παίκτες με διαφορετικό αριθμό να έχουν διαφορετικό χρώμα αυτοκινήτου ώστε να ξεχωρίζουν.

```
public override void OnStartLocalPlayer()
{
    GetNetIdentity();
    SetIdentity();
}

//Get this object's Net ID and sync it to the server
[Client]
void GetNetIdentity()
{
    playerNetID = GetComponent<NetworkIdentity>().netId;
    CmdTellServerMyName(MakeUniqueName(), MakeUniqueNumber());
}

void SetIdentity()
{
    myTransform.name = playerUniqueName;
    if(playerNumber != 0)
    {
        //Set car color
        script.rendBody.material.mainTexture =  multiplayerTex[playerNumber-1];
        script.rendBody2.material.mainTexture =  multiplayerTex[playerNumber-1];
        script.rendBody3.material.mainTexture =  multiplayerTex[playerNumber-1];

        //Move car to spawnpoint (disable rigidbody and collider so that it doesn't jump when it spawns)
        carCollider.enabled = false;
        carRB.isKinematic = true;
        transform.position = spawnPoints[playerNumber - 1].position;
        carCollider.enabled = true;
        carRB.isKinematic = false;
    }
}
```

Εικόνα 3.33: Οι συναρτήσεις Get και SetNetIdentity.

3.10.5 Interpolation

Το κύριο πρόβλημα που αντιμετωπίζουν τα διαδικτυακά βιντεοπαιχνίδια είναι η καθυστέρηση που χρειάζονται τα πακέτα για να σταλούν μεταξύ των δύο απομακρισμένων υπολογιστών που παίζουν το ίδιο παιχνίδι. Ας πάρουμε για παράδειγμα τον συγχρονισμό της τοποθεσίας ενός αυτοκινήτου μεταξύ 2 απομακρυσμένων υπολογιστών. Ας υποθέσουμε ότι για να σταλθεί ένα πακέτο από τον υπολογιστή A στον B και παλι πίσω χρειάζεται 45ms καθυστέρηση και ότι το αυτοκίνητο το κινεί ο υπολογιστής B. Τότε όμως μέσα σε αυτά τα 45ms έχει διαμεσολαβήσει ένα μικρό αλλά σημαντικό χρονικό διάστημα στο οποίο ο υπολογιστής A δεν γνωρίζει την τοποθεσία του αυτοκινήτου και το κρατάει ακίνητο για 45ms μέχρις ότου να έρθει νέο πακέτο από τον B για την καινούργια τοποθεσία του αυτοκινήτου. Αν και ο χρόνος φαίνεται πολύ μικρός στην πραγματικότητα αν δεν χρησιμοποιηθεί μια μέθοδος εξομάλυνσης ο χρήστης στον υπολογιστή A θα βλέπει το αμάξι να <<τηλεμεταφέρεται>> σε μικρές αποστάσεις.
Για την λύση του παραπάνω προβλήματος χρησιμοποιήθηκε η μέθοδος της παρεμβολής (interpolation). Αυτό που κάνει η συγκεκριμένη τεχνική είναι να προσπαθεί να μαντέψει τις ενδιάμεσες καταστάσεις που έχει το αυτοκίνητο μέχρις ότου να πάρει το update από τον server όπου όταν το παίρνει κάνει ομαλή μετάβαση(lerping) στην πραγματική κατάσταση

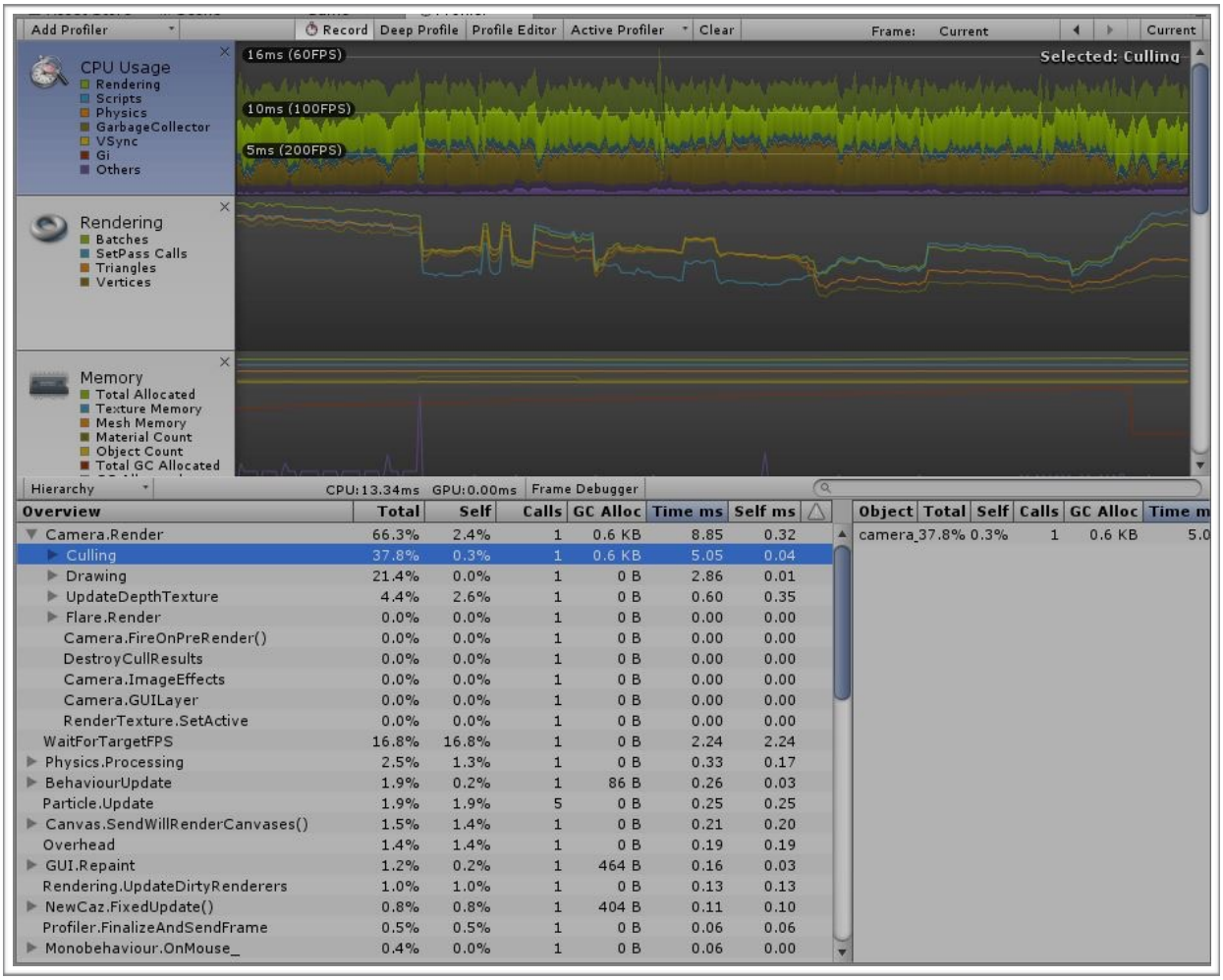
του server. Αν πάρουμε πάλι το παραπάνω παράδειγμα με τους υπολογιστές Α και Β τότε αυτό που κάνουμε είναι πέρα από το να λέει ο Β στον Α την τοποθεσία του αμαξιού να του λέει επίσης και την ταχύτητα του. Με αυτόν τον τρόπο ο Α τοποθετεί το αυτοκίνητο στο σημείο που του είπε ο Β και μέχρι να πάρει νέα δεδομένα για την τοποθεσία του αυτοκινήτου τότε υπολογίζει ο Α το πού πιστεύει ότι βρίσκεται το αμάξι με βάση την ταχύτητα του αμαξιού και τους νόμους της φυσικής του παιχνιδιού. Με αυτόν τον τρόπο ομαλοποιείται η κίνηση του αμαξιού και έχουμε ένα πιο όμορφο οπτικό αποτέλεσμα για τον τελικό χρήστη του παιχνιδιού.

3.11 Βελτίωση Framerate

Ένα πολύ σημαντικό στοιχείο ενός παιχνιδιού για να είναι διασκεδαστικό στον τελικό χρήστη είναι να μπορεί να έχει ένα ικανοποιητικό framerate(καρέ ανά δευτερόλεπτο). Όπως και στον κινηματογράφο, ο χρήστης αυτό που βλέπει είναι στην ουσία πάρα πολλές εικόνες οι οποίες κινούνται πολύ γρήγορα ταυτόχρονα και έτσι μπερδεύεται το μάτι και νομίζει ότι βλέπει μια συνεχή κίνηση. Έτσι και στα βιντεοπαιχνίδια αυτό που γίνεται είναι ότι η κάρτα γραφικών δημιουργεί με μεγάλη ταχύτητα εικόνες με βάσει τα πλήκτρα που πατάει ο χρήστης και τους κανόνες του παιχνιδιού.
Το πρόβλημα στην περίπτωση των βιντεοπαιχνιδιών είναι ότι επειδή δεν γίνεται να γνωρίζεις το πώς θα εξελιχτεί το παιχνίδι ή τί πλήκτρα θα πατήσει ο χρήστης τα καρέ αυτά πρέπει να δημιουργηθούν real-time την ώρα που παίζεται το παιχνίδι, δεν είναι δηλαδή pre-rendered όπως στις ταινίες. Εφόσον λοιπόν το rendering στην περίπτωσή μας γίνεται αναγκαστικά σε πραγματικό χρόνο(real-time rendering) υπάρχουν κάποιοι περιορισμοί. Για παράδειγμα η GPU έχει κάποια όρια μνήμης πέραν των οποίων αν προσπαθήσεις να φορτώσεις περισσότερα μοντέλα ή υφές από ότι αντέχει τότε δημιουργούνται πτώσεις στο framerate του παιχνιδιού τα οποία επιδεινώνουν την εμπειρία του χρήστη.

3.11.1 Profiler

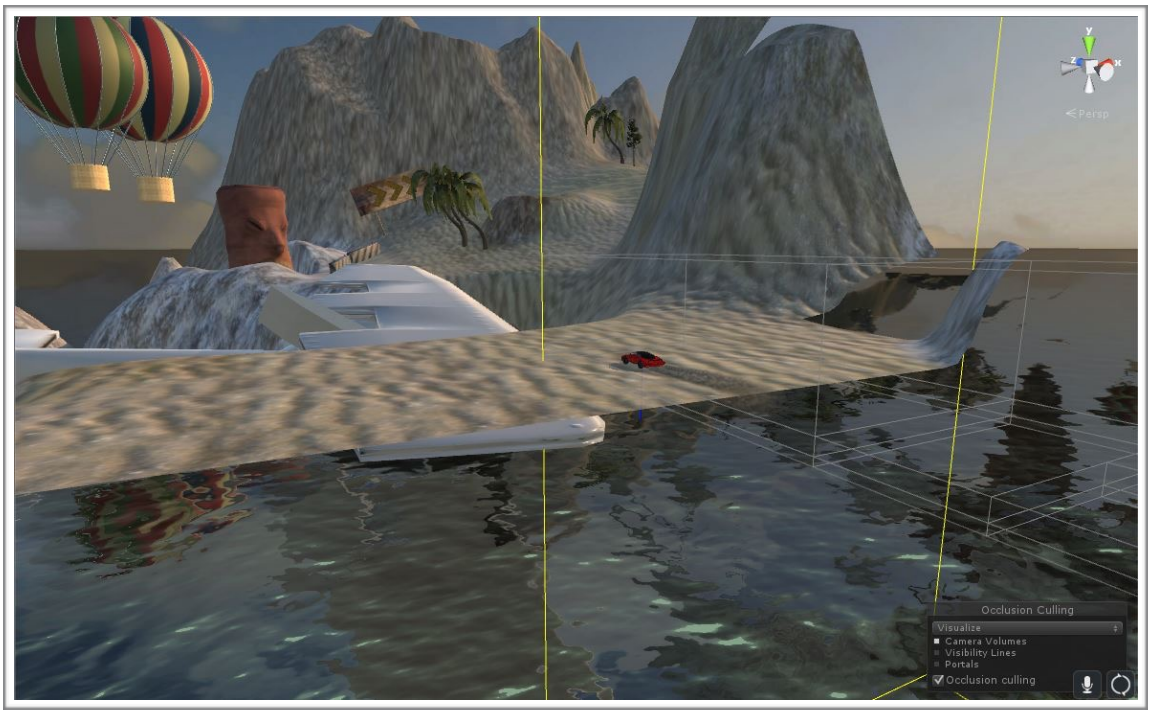
Μια μέθοδος που παρέχει η Unity για τον εντοπισμό του υπαίτιου τέτοιων πτώσεων στα καρέ του παιχνιδιού είναι το profiler παράθυρο. Το συγκεκριμένο παράθυρο επιτρέπει στον χρήστη να ελέγχει την απόδοση της κάρτας γραφικών του επεξεργαστή και των υπόλοιπων τμημάτων hardware του υπολογιστή. Επίσης μπορεί ο χρήστης να ελέγξει το ποσοστό που σπαταλάει η Unity για να δημιουργήσει το κάθε κομμάτι του παιχνιδιού(λογική, rendering, audio κ.τλ).
Το παράθυρο του profiler χωρίζεται σε 2 βασικά τμήματα. Στο πάνω κομμάτι βλέπουμε την απόδοση του κάθε στοιχείου του παιχνιδιού για κάθε καρέ σε μορφή διαγράμματος. Μπορούμε για παράδειγμα στο διάγραμμα της CPU να ελέγξουμε λόγω πιανής διαδικασίας πέφτουν τα fps σε κάποια καρέ. Το κάτω τμήμα του profiler δείχνει με λεπτομέρεια το κάθε στοιχείο και την κάθε συνάρτηση που χρησιμοποιεί πόρους από το σύστημα και τί ποσοστό πόρων χρησιμοποιεί.



Εικόνα X: Το Profiler παράθυρο.

3.11.2 Occlusion Culling

Το occlusion culling είναι μια τεχνική η οποία χρησιμοποιείται σχεδόν σε όλα τα σημερινά τρισδιάστατα βιντεοπαιχνίδια. Αυτό που κάνει είναι να υπολογίζει ποιά μοντέλα δεν βρίσκονται μέσα στο οπτικό πεδίο της κάμερας του παίκτη και να τα κόβει από το rendering pipeline ελαφρύνοντας έτσι την κάρτα γραφικών αφού θα πρέπει να κάνει render λιγότερα μοντέλα και υφές. Στην Unity το occlusion culling γίνεται με αυτόματο τρόπο ανοίγοντας το occlusion παράθυρο και πατώντας bake.



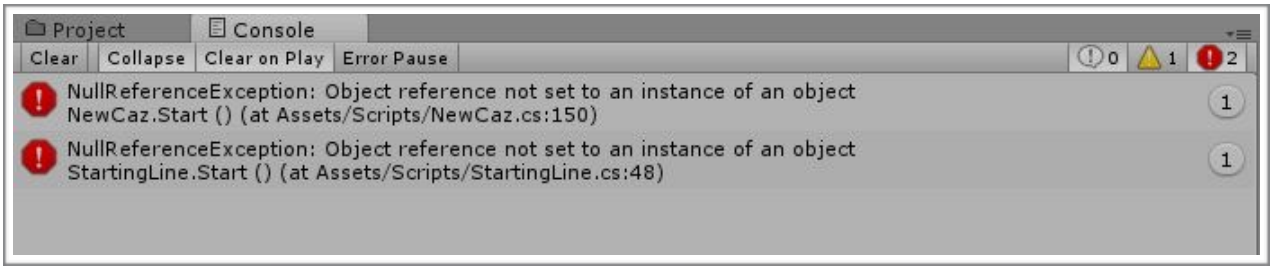
Εικόνα Χ: Χρησιμοποιώντας το occlusion culling η κάρτα γραφικών φορτώνει μόνο τα μοντέλα που είναι μπροστά από το αυτοκίνητο και ορατά στην κάμερα.

3.12 Debugging

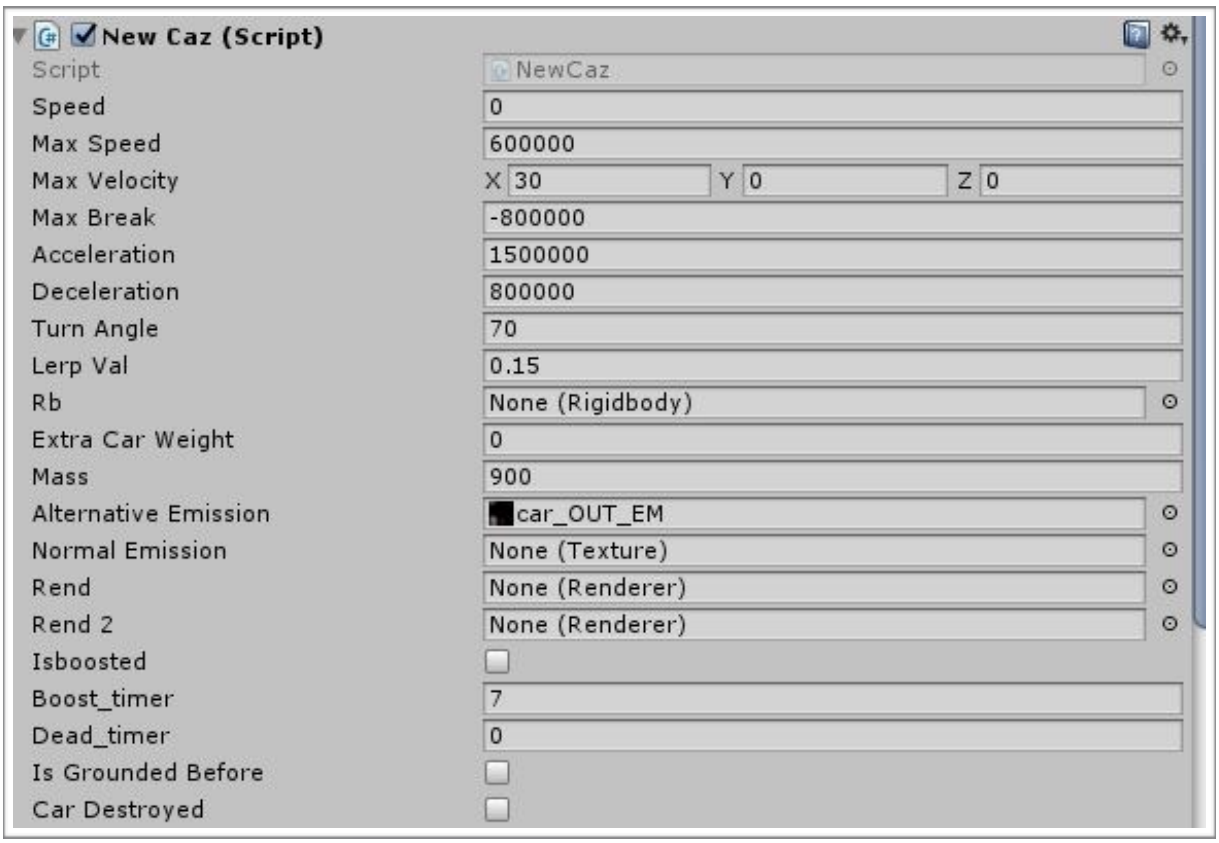
Το debugging στην Unity όπως και στις γλώσσες προγραμματισμού είναι η διαδικασία εκείνη με την οποία βρίσκουμε και εξαφανίζουμε τα διάφορα bugs ή προβλήματα στον κώδικα του παιχνιδιού. Είναι μια διαδικασία που χρησιμοποιείται συνεχώς κατά την ανάπτυξη του παιχνιδιού από το αρχικό πρωτότυπο μέχρι την τελειοποίηση της τελικής έκδοσης. Για το debugging η Unity παρέχει το Console παράθυρο το οποίο ενημερώνει τον προγραμματιστή για οποιοδήποτε πρόβλημα εμφανίζεται κατά την διάρκεια του τρεξίματος του παιχνιδιού.

Ένα συνηθισμένο ζήτημα κατά την διάρκεια του debugging ενός προγράμματος είναι να μπορούμε να ελέγχουμε την τιμή που έχει μια μεταβλητή οποιαδήποτε στιγμή. Στην Unity αυτό για να το πετύχουμε υπάρχουν δύο τρόποι. Ο πρώτος τρόπος είναι να δηλώσουμε την μεταβλητή ως public στο script οπότε όταν ανοίγουμε το script στο inspector μπορούμε να δούμε την τιμή του ανά πάσα στιγμή. Μια εναλλακτική λύση είναι να εμφανίσουμε την

τιμή του script στην κονσόλα μέσω της εντολής Debug.Log(). Βάζουμε την μεταβλητή μέσα στην εντολή αυτή και όταν το script φτάσει στην γραμμή αυτή θα εμφανίσει στην κονσόλα την τιμή της μεταβλητής που εισάγαμε.



Εικόνα Χ: Το Console παράθυρο ενημερώνει τον προγραμματιστή για κάποιο πρόβλημα και τον πληροφορεί ποιό script και ποιά γραμμή σε αυτό το script το δημιουργεί.



Εικόνα Χ: Διάφορες public μεταβλητές που βλέπουμε τις τιμές τους στο inspector παράθυρο και ανανεώνονται κάθε frame.

Βιβλιογραφία

https://en.wikipedia.org/wiki/UV_mapping

http://blog.digitaltutors.com/understanding-inverse-and-forward-kinematics/

https://en.wikipedia.org/wiki/Skeletal_animation

https://en.wikipedia.org/wiki/Tennis_for_Two

https://en.wikipedia.org/wiki/Fifth_generation_of_video_game_consoles