ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ - ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ: ΣΥΣΤΗΜΑΤΩΝ ΚΙ ΑΥΤΟΜΑΤΟΥ ΕΛΕΓΧΟΥ

Διπλωματικη Εργασια

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών της Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών

Δημητριου Τρουπη
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 227394

Θέμα

Μοντελοποίηση και Προσομοίωση Αστικής Κυκλοφοριακής Κίνησης Μέσω Φυσικής των Οχημάτων για Σχεδίαση Αποδοτικών Οδικών Συστημάτων

Επιβλέπων
Αναπληρωτής Καθηγητής Μουστάκας Κωνσταντίνος

Αριθμός Διπλωματικής Εργασίας:

Πάτρα, Ιανουάριος 2021

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η διπλωματική εργασία με θέμα

Μοντελοποίηση και Προσομοίωση Αστικής Κυκλοφοριακής Κίνησης Μέσω Φυσικής των Οχημάτων για Σχεδίαση Αποδοτικών Οδικών Συστημάτων

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Δημήτριου Τρουπή

(Α.Μ.: 227394)

παρουσιάστηκε δημόσια και εξετάστηκε στο τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις

___/___/____

Ο Επιβλέπων

Μουστάκας Κωνσταντίνος
Αναπληρωτής Καθηγητής

Ο Διευθυντής του Τομέα

Σγάρμπας Κυριάκος
Αναπληρωτής Καθηγητής

Στοιχεία διπλωματικής εργασίας

Θέμα: **Μοντελοποίηση και Προσομοίωση Αστικής Κυκλοφοριακής Κίνησης Μέσω Φυσικής των Οχημάτων για Σχεδίαση Αποδοτικών Οδικών Συστημάτων**

Φοιτητής: **Δημήτριος Τρουπής**

Ομάδα επίβλεψης
Αναπληρωτής Καθηγητής Μουστάκας Κωνσταντίνος

Εργαστήριο Απεικόνισης Πληροφορίας και Εικονικής Πραγματικότητας

Η εργασία αυτή γράφτηκε στο XΗLATEX και χρησιμοποιήθηκε η γραμματοσειρά GFS Didot του Greek Font Society

Plagiarism Disclaimer: I hereby declare that this thesis is my own and autonomous work. All sources and aids used have been indicated as such. All texts either quoted directly or paraphrased have been indicated by in-text citations.Full bibliographic details are given in the reference list which also contains internet sources containing URL and access date. This work has not been submitted to any other examination authority.

Περίληψη

Στην εργασία αυτή, αναπτύσσεται ένα λογισμικό σχεδίασης, προσομοίωσης κι αποτίμησης οδικών συστημάτων με χρήση της μηχανής παιχνιδιών Unity3D. Ο χρήστης μέσω του εργαλείου σχεδιασμού παράγει αρχεία χάρτη που ορίζουν ένα οδικό σύστημα, τα οποία έπειτα εισάγονται στο εργαλείο προσομοίωσης ώστε να παρακολουθήσει την εξέλιξη της κυκλοφορίας σε πραγματικό χρόνο, με τη βοήθεια οπτικοποίησης στατιστικών που υλοποιείται με HLSL shaders. Το εργαλείο σχεδιασμού περιλαμβάνει γραφική διεπαφή χρήστη (GUI) με λειτουργίες εισαγωγής εικόνας, σχεδιασμού κυβικών καμπυλών Bézier και ορισμού λωρίδων. Η προσομοίωση υλοποιεί ένα μικροσκοπικό μοντέλο κυκλοφορίας με συμπεριφορές όπως ακολούθηση μονοπατιού και παραχώρηση προτεραιότητας. Η φυσική των οχημάτων επίσης λαμβάνεται υπ'όψην σε μεγάλο βαθμό για πιο ρεαλιστική συμπεριφορά των οχημάτων αλλά και για εξαγωγή δεδομένων κατανάλωσης καυσίμου. Αξιοποιώντας την ενσωματωμένη μηχανή φυσικής PhysX, ρυθμίζουμε μοντέλα ελατηρίου - αποσβεστήρα για την ανάρτηση καθώς και ένα μοντέλο ολίσθησης για την φυσική των ελαστικών. Ο κινητήρας και η μετάδοση επίσης προσομοιώνονται με ένα μοντέλο βασισμένο στη σύμπλεξη δίσκων, στη μείωση/αύξηση και μεταφορά ροπής και στη ροπή αδράνειας.

Λέξεις κλειδιά: Προσομοίωση κυκλοφορίας, προσομοίωση οχήματος, πολεοδομικός σχεδιασμός, οικονομία καυσίμου, visual analytics

Abstract

In this work, a software for design, simulation and evaluation of road systems is developed using the Unity3D game engine. The user via the design tool creates map files that define a road system, which are then imported into the simulation tool so as to examine the traffic progression in real-time, with the help of statistics visualization implemented in HLSL shaders. The design tool includes a graphical user interface (GUI) with operations for image importing, design of cubic Bézier curves and lane definition. The simulation implements a microscopic traffic model, with behaviours such as path following and yielding. Vehicle physics are also seriously considered for more realistic vehicle behaviour and for extraction of fuel consumption data. Utizing the built-in PhysX physics engine, we adjust spring-damper models for the suspension as well as a slip model for the tire physics. The engine and transmission (powertrain) are also simulated with a model based on clutch engage, torque transfer and moment of inertia.

Keywords: Traffic simulation, vehicle simulation, urban planning, fuel economy, visual analytics

Ευχαριστίες

Θα ήθελα αρχικά να εκφράσω την ευγνωμοσύνη μου στην οικογένειά μου για την στήριξη και υπομονή τους όλα αυτά τα χρόνια. Να ευχαριστήσω επίσης τον επιβλέποντά μου κ. Κωνσταντίνο Μουστάκα, καθώς πέρα από την ευκαιρία της εκπόνησης αυτής της εργασίας, τα επιστημονικά πεδία που εισήγαγε στο Τμήμα με ενέπνευσαν στο να πάρω τον δικό μου δρόμο.

Περιεχόμενα

1 Εισαγωγή

1.1 Εισαγωγικό σημείωμα και σκοπός εργασίας

Σε έναν ραγδαία αναπτυσσόμενο κόσμο, η ανάγκη για ρεαλιστική προσομοίωση πολύπλοκων και μεγάλων συστημάτων είναι ολοένα και πιο επιτακτική. Η επιστήμη και η τεχνολογία της εποχής μας έχουν να αντιμετωπίσουν συστήματα αμέτρητων παραμέτρων, μεγάλης σημαντικότητας, μεγάλης επικινδυνότητας, μεγάλων ποσοτήτων ενέργειας και καθοριστικών οικονομικών επιπτώσεων. Είναι προφανές λοιπόν το πως γεννάται η ανάγκη για ακριβή πρόβλεψη και εκτίμηση αυτών των παραμέτρων, για αποδοτικότερο σχεδιασμό σε θεωρητικό αλλά και σε επίπεδο υλοποίησης.

Ένα από αυτά τα πολύπλοκα συστήματα αφορά στην μετακίνηση των ανθρώπων για λόγους επαγγελματικούς ή και αναψυχής, που λόγω της μαζικότητάς της έχει μεγάλες συνέπειες, θετικές ή αρνητικές, σε θέματα οικονομίας, ψυχικής και σωματικής υγείας, ποιότητας ζωής, περιβάλλοντος αλλά και αποδοτικότητας και αποτελεσματικότητας της μετακίνησης καθ' αυτής.

Σχήμα 1: Μαζική κίνηση σε εθνικό δίκτυο. Πηγή: medicalbag.com

Σε αυτήν την εργασία αναπτύσσεται ένα λογισμικό προσομοίωσης της κυκλοφοριακής κίνησης σε αστικό ή υπεραστικό περιβάλλον μέσω μοντελοποίησης της συμπεριφοράς των οδηγών σε ατομικό, ή κατά την ορολογία αυτού του είδους προσομοιώσεων, μικροσκοπικό επίπεδο. Έμφαση έχει δοθεί στην επιλογή αλγορίθμων και στον τρόπο υλοποίησής τους για να μπορεί να επιτευχθεί το παραπάνω ταυτόχρονα σε σχετικά μεγάλο αριθμό οχημάτων, ώστε τα

αποτελέσματα της προσομοίωσης να είναι στατιστικά εμπιστέψιμα. Δίνεται η δυνατότητα στο χρήστη να σχεδιάσει οδικά συστήματα (δρόμους, διασταυρώσεις, κόμβους κ.α.) μέσω ειδικού εργαλείου που αναπτύχθηκε.

Στη συνέχεια το σχεδιασμένο αυτό οδικό σύστημα περνάει μέσα από την προσομοίωση, κατά την οποία μεγάλος αριθμός οχημάτων το διασχίζει και αλληλεπιδρά με όσο το δυνατόν πιο ρεαλιστικό τρόπο. Μεγάλο ρόλο σε αυτό έχει η μοντελοποίηση της δυναμικής των οχημάτων, τόσο όσον αφορά τις δυνάμεις αλληλεπίδρασης με το έδαφος και τον αέρα, όσο και με τη φυσική της μετάδοσης της κίνησης στους τροχούς. Μέσω του δεύτερου γίνεται και προσεγγιστικός υπολογισμός της κατανάλωσης σε καύσιμο, είτε στιγμιαία είτε καθ' όλην τη διάρκεια ύπαρξης του οχήματος στο μελετούμενο οδικό σύστημα. Τέλος, αναπτύχθηκε εργαλείο οπτικής και στατιστικής ανάλυσης, ώστε ο χρήστης να μπορεί να αποτιμήσει την απόδοση του συστήματος σύμφωνα με τις προδιαγραφές που θεωρεί σημαντικότερες (π.χ. συχνότητα συμφορήσεων, συνολική κατανάλωση καυσίμου κ.α.)

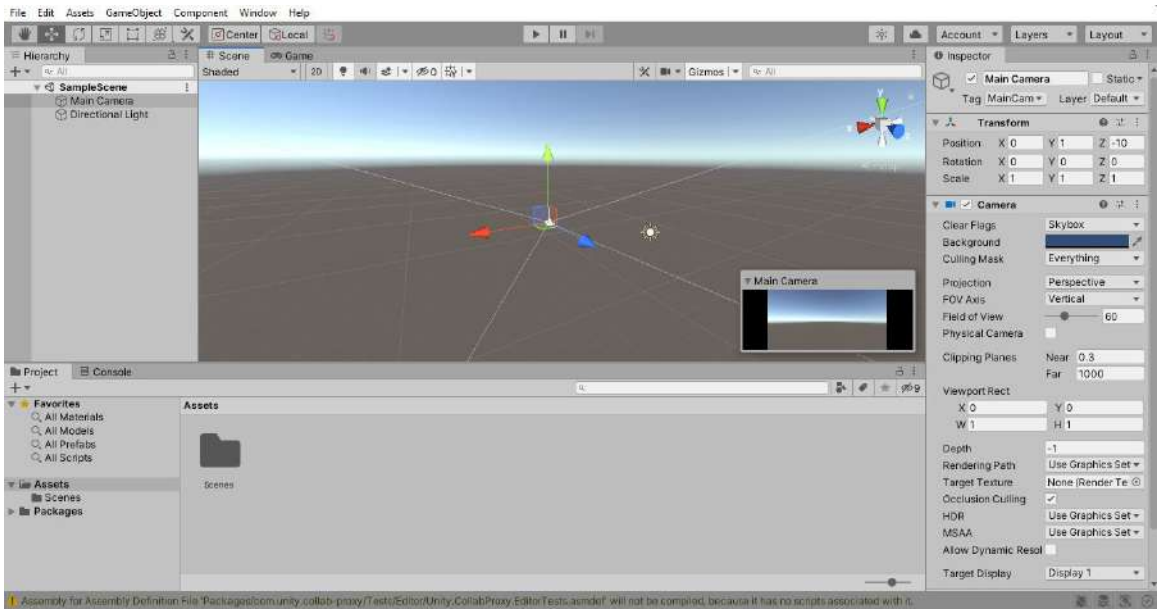
1.2 Τεχνικά στοιχεία

1.2.1 Μηχανή γραφικών

Η οπτικοποίηση στην παρούσα εργασία γίνεται με χρήση τριδιάστατων γραφικών σε πραγματικό χρόνο. Είναι δυνατό κανείς να αναπτύξει εφαρμογές τριδιάστατων στην εποχή μας ξεκινώντας από το απόλυτο μηδέν, όπως γινόταν άλλωστε και στα πρώτα χρόνια εμφάνισής τους στους οικιακούς υπολογιστές. Ωστόσο κάτι τέτοιο θα ήταν πολύ χρονοβόρο, δεδομένης της εμφάνισης και εξέλιξης πανίσχυρων εργαλείων που αναλαμβάνουν την ρεαλιστική τριδιάστατη οπτικοποίηση αντικειμένων και περιβαλλόντων. Τα εργαλεία αυτά, πέρα από την ποιότητα της ίδιας της οπτικοποίησης και την επιτάχυνση της διαδικασίας ανάπτυξης, έρχονται και με πληθώρα άλλων πλεονεκτημάτων, όπως απουσία σημαντικών εν δυνάμει σφαλμάτων λειτουργίας (bugs) και εγγυημένη συμβατότητα μεταξύ των διαφόρων συσκευών αναπαραγωγής. Επίσης χωρίζονται σε διάφορες κατηγορίες, κυρίως ως προς το επίπεδο που βοηθούν κι εμπλέκονται στην διαδικασία ανάπτυξης, καθώς και στις δυνατότητες που προσφέρουν. [1]

Εδώ θα χρησιμοποιήσουμε την Unity3D[2], προϊόν της Unity Technologies, η οποία ανήκει στις λεγόμενες μηχανές παιχνιδιών (game engines), δηλαδή λογισμικά που βοηθούν στην ανάπτυξη

παιγνίων υπολογιστή. Φυσικά, η τεχνολογία αυτή μπορεί να χρησιμοποιηθεί για πληθώρα άλλων εφαρμογών, όπως οπτικοποιήσεις γενικού σκοπού, προσομοιώσεις, εκπαιδευτικές εφαρμογές κ.α. Μια μηχανή παιχνιδιών περιλαμβάνει τη μηχανή γραφικών, τη μηχανή φυσικής (που θα δούμε στην επόμενη παράγραφο), εργαλεία δικτύωσης (networking), εργαλεία διεπαφής με το σύστημα (system I/O), μηχανή ήχου και πολλά άλλα βοηθήματα. Επίσης, οι μηχανές παιχνιδιών περιλαμβάνουν και γραφική διεπαφή με το χρήστη (editor) για ευκολότερη τοποθέτηση αντικειμένων στο χώρο, άμεση οπτική προεπισκόπηση καθώς και διαχείριση του εκάστοτε περιβάλλοντος, χωρίς την ανάγκη χρήσης κώδικα και χειροκίνητων υπολογισμών. Αυτό έρχεται σε αντίθεση με, για παράδειγμα, τις βιβλιοθήκες (libraries) που συνήθως παρέχουν τα παραπάνω εργαλεία μόνο με τη χρήση κώδικα.



Σχήμα 2: Η γραφική διεπαφή (editor) της Unity σε μια νέα εργασία (project)

Στην παραπάνω εικόνα βλέπουμε την γραφική διεπαφή που μας παρέχεται ξεκινώντας μια νέα εργασία (project) στη Unity. Στα αριστερά υπάρχει η ιεραρχία, που είναι μια λίστα με τα αντικείμενα που αποτελούν το περιβάλλον και που οργανώνονται με σχέσεις ”γονέων” και ”παιδιών”. Δεξιά είναι ο inspector, ένα πεδίο στο οποίο αναλύονται οι παράμετροι κάθε αντικειμένου και μας δίνεται η δυνατότητα να τις επεξεργαστούμε. Κάτω υπάρχει μια γενική επισκόπηση των αρχείων και της διάρθρωσης της εργασίας, και τέλος στο κέντρο υπάρχει η

άποψη σκηνής (scene view), στην οποία βλέπουμε μια προεπισκόπηση του περιβάλλοντος που έχουμε δημιουργήσει.

Η μηχανή γραφικών που περιλαμβάνεται είναι υψηλών προδιαγραφών και μας δίνει τη δυνατότητα επίτευξης υψηλής ποιότητας γραφικών μέσω ειδικών εντολών και παραμετροποίησης των ιδιοτήτων υλικών και φωτισμού, όπως επίσης και τη δυνατότητα δημιουργίας σκιαστών (shaders) που είναι ειδικά προγράμματα επεξεργασίας γραφικών και θα χρησιμοποιηθούν στην οπτικοποίηση των στατιστικών, όπως θα δούμε σε επόμενο κεφάλαιο.

Η χρήση της μηχανής γίνεται μεν μέσω της διεπαφής που παρουσιάστηκε παραπάνω, αλλά και με τη χρήση κώδικα. Ο κώδικας γράφεται στην γλώσσα C# [4], μια γλώσσα προγραμματισμού γενικού σκοπού που έχει δημιουργηθεί από τη Microsoft. Η C# στη συγκεκριμένη περίπτωση χρησιμοποιείται ως scripting γλώσσα: αυτό σημαίνει πως γράφοντας C# δεν προγραμματίζουμε απευθείας το σύστημα που εκτελεί τον κώδικα (native) όπως θα κάναμε υπό κανονικές συνθήκες, παρά δίνουμε εντολές σε κώδικα της Unity που όντως τρέχει στο σύστημα, για να εκτελεστεί. Η Unity επίσης έχει φροντίσει να ενσωματώσει το .NET, ένα οικοσύστημα λογισμικού που συμπληρώνει την C# με βιβλιοθήκες κι εργαλεία, κάνοντας έτσι την εμπειρία σχεδόν πανομοιότυπη με την καθιερωμένη ανάπτυξη λογισμικού σε C#.

1.2.2 Μηχανή φυσικής

Όπως αναφέρθηκε, η μηχανή φυσικής αποτελεί ένα βασικό κομμάτι μιας τυπικής μηχανής παιχνιδιών. Η Unity στη συγκεκριμένη περίπτωση δεν υλοποιεί δική της μηχανή φυσικής, αλλά, όπως είναι σύνηθες, ενσωματώνει μια έτοιμη λύση του εμπορίου. Η μηχανή φυσικής αυτή ονομάζεται PhysX[3], παράγεται από την Nvidia, είναι ανοιχτού κώδικα κι είναι μια από τις κυρίαρχες μηχανές φυσικής του χώρου, μαζί με την Havok και τη Bullet.

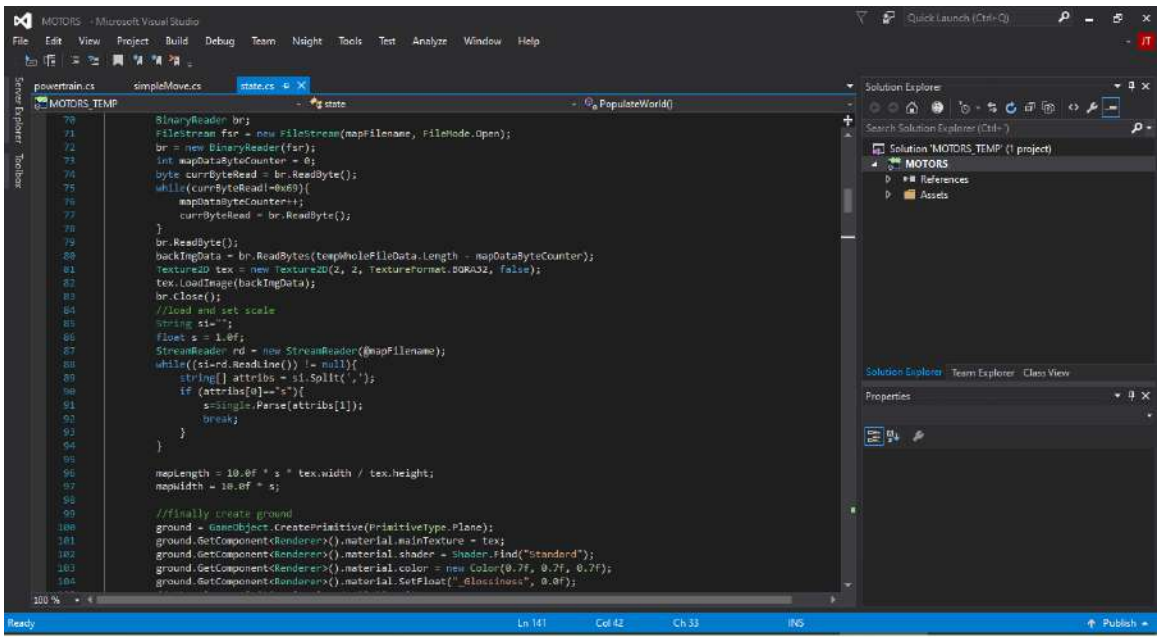
Μια τυπική μηχανή φυσικής παρέχει προσομοίωση κυρίως Νευτώνιας μηχανικής, και συγκεκριμένα ρευστών, στερεών και παραμορφώσιμων αντικειμένων. Αναλόγως την εφαρμογή, δίνεται σημασία στην ποιότητα της λύσης, στην σταθερότητα της λύσης, στην ταχύτητα εύρεσης αυτής καθώς και στην βέλτιστη υλοποίηση των αλγορίθμων αριθμητικής επίλυσης λαμβάνοντας υπόψην τις ιδιότητες του υλισμικού (hardware) στο οποίο αυτή εκτελείται. Όσον αφορά στην ενσωμάτωση που μας παρέχει η Unity, δεν υπάρχουν πολλές δυνατότητες μετατροπής στον

τρόπο λειτουργίας της μηχανής φυσικής, ωστόσο βασικές παράμετροι όπως το χρονικό βήμα της προσομοίωσης (timestep) είναι διαθέσιμες προς αλλαγή. Περισσότερες λεπτομέρειες της χρήσης της μηχανής φυσικής θα δούμε σε επόμενο κεφάλαιο, όπου παρουσιάζεται αναλυτικά η προσομοίωση της φυσικής των οχημάτων.

1.2.3 Επιπλέον εργαλεία

Επιπλέον της Unity, διάφορα άλλα εργαλεία έχουν χρησιμοποιηθεί για την ανάπτυξη του κώδικα αλλά και την παραγωγή των αρχείων εικόνας κι ήχου που χρησιμοποιούνται στην εργασία.

Ο κώδικας συγγράφεται στο Visual Studio[5], ένα IDE (Integrated Development Environment) που περιλαμβάνει πολλές ευκολίες για συγγραφή και διαχείριση κώδικα, και το οποίο η Unity υποστηρίζει σε μεγάλο βαθμό.



Σχήμα 3: Η γραφική διεπαφή (editor) του Visual Studio με ανοιχτό αρχείο της εργασίας

Το Visual Studio δεν χρησιμοποιήθηκε μόνο στα πλαίσια της Unity. Κάποια στιγμή, όπως θα δούμε σε επόμενα κεφάλαιο, θα χρειαστεί να "ανοίξουμε" διαλόγους (dialogs) του λειτουργικού συστήματος για αναζήτηση (browsing) αρχείων. Η Unity δεν προσφέρει κάτι

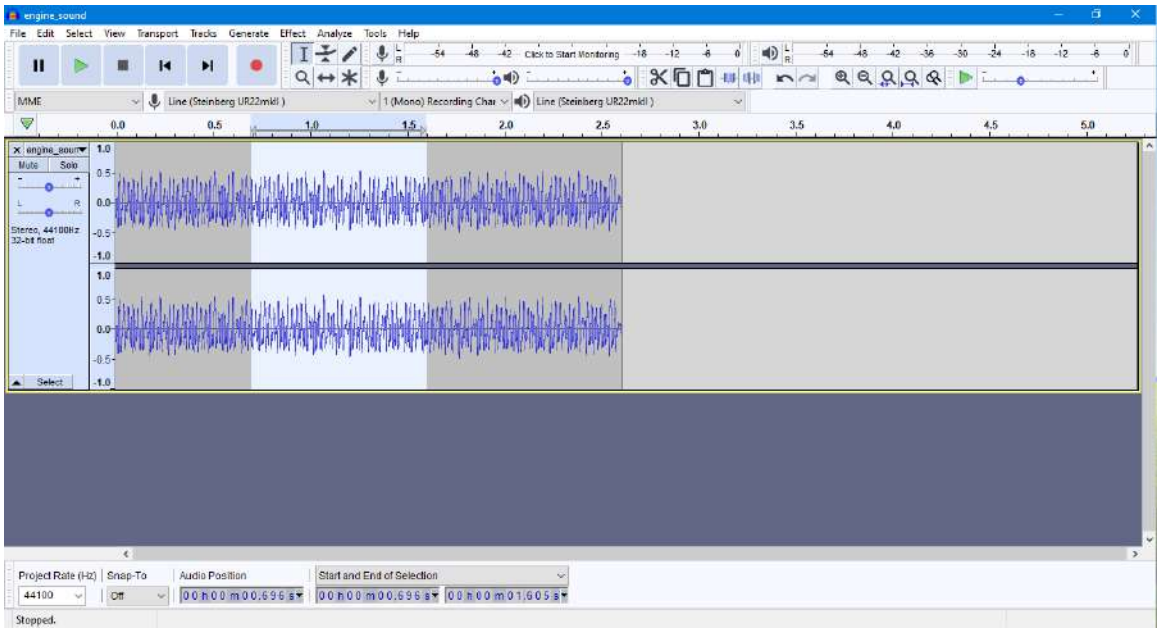
τέτοιο, συνεπώς θα πρέπει να βρούμε κάποιον εξωτερικό τρόπο. Εδώ γίνεται χρήση του .NET, το οποίο προσφέρει συναρτήσεις και διαλόγους για άνοιγμα αλλά και αποθήκευση αρχείων. Εφόσον αυτά όπως είπαμε δεν υλοποιούνται μέσα στη Unity, τα υλοποιούμε σε ξεχωριστό project. Το παράγωγο αυτού του project είναι ένα αρχείο τύπου .dll (Dynamically Linked Library), το οποίο περιέχει εκτελέσιμο κώδικα που μπορεί να εκτελεστεί από ένα τρίτο πρόγραμμα (που στην περίπτωσή μας είναι η Unity). Ο κώδικας του project αυτού είναι πολύ απλός, και προσφέρει απλά συναρτήσεις με τις οποίες μπορούμε να ανοίξουμε τέτοιους διαλόγους A.1.

Οι δυο συναρτήσεις που ορίζει, είναι κι αυτές που είναι διαθέσιμες προς τον τρίτο που θα χρησιμοποιήσει το .dll αρχείο που παράγεται. Έτσι, όταν έρθει η ώρα να ανοίξουμε ένα αρχείο από τη Unity, χρησιμοποιούμε άμεσα αυτές τις συναρτήσεις A.2.

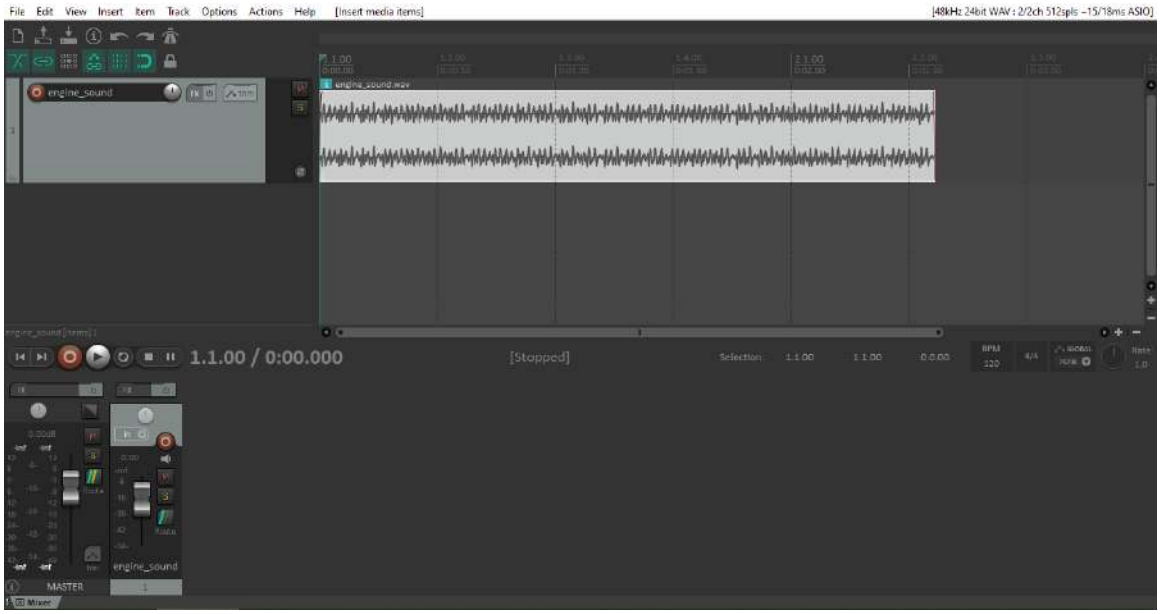
Αρχικά εισάγουμε στο project το namespace NETWindowsDialogs που δηλώθηκε παραπάνω και το οποίο περιλαμβάνει τις συναρτήσεις που μας ενδιαφέρουν. Έπειτα, όταν χρειαστούμε για παράδειγμα να ανοίξουμε ένα αρχείο όπως στην περίπτωση της έναρξης της προσομοίωσης, καλούμε την αντίστοιχη συνάρτηση. Η συνάρτηση όπως φαίνεται, δέχεται μια παράμετρο, το φίλτρο (filter). Αυτό μας δίνει τη δυνατότητα να αφήσουμε το χρήστη να διαλέξει μόνο αρχεία συγκεκριμένου τύπου. Στην περίπτωσή μας είναι αρχεία τύπου .motors, τα οποία όπως θα δούμε αργότερα είναι ειδικά αρχεία που περιέχουν τον χάρτη που πρόκειται να προσομοιωθεί. Να αναφέρουμε τέλος, πως για να ανιχνεύσει η Unity το εν λόγω .dll αρχείο, δεν έχουμε παρά να το βάλουμε σε κάποιον από τους φακέλους του project, κι αυτή θα μας το κάνει διαθέσιμο.

Για την παραγωγή των αρχείων ήχου, χρησιμοποιήθηκαν τα προγράμματα Audacity[6] και REAPER[7].

Το Audacity πρόκειται για ένα πρόγραμμα επεξεργασίας ήχου, με πολύ χαμηλές απαιτήσεις συστήματος και κυρίως είναι χρήσιμο για απλές και σύντομες επεξεργασίες, όπως χωρισμό αρχείων ήχου σε μικρότερα, εφαρμογή ηχητικών φίλτρων κ.α. Αντιθέτως, το REAPER αποτελεί ένα DAW (Digital Audio Workstation), με κατά πολύ ανώτερες δυνατότητες και ευρεία χρήση



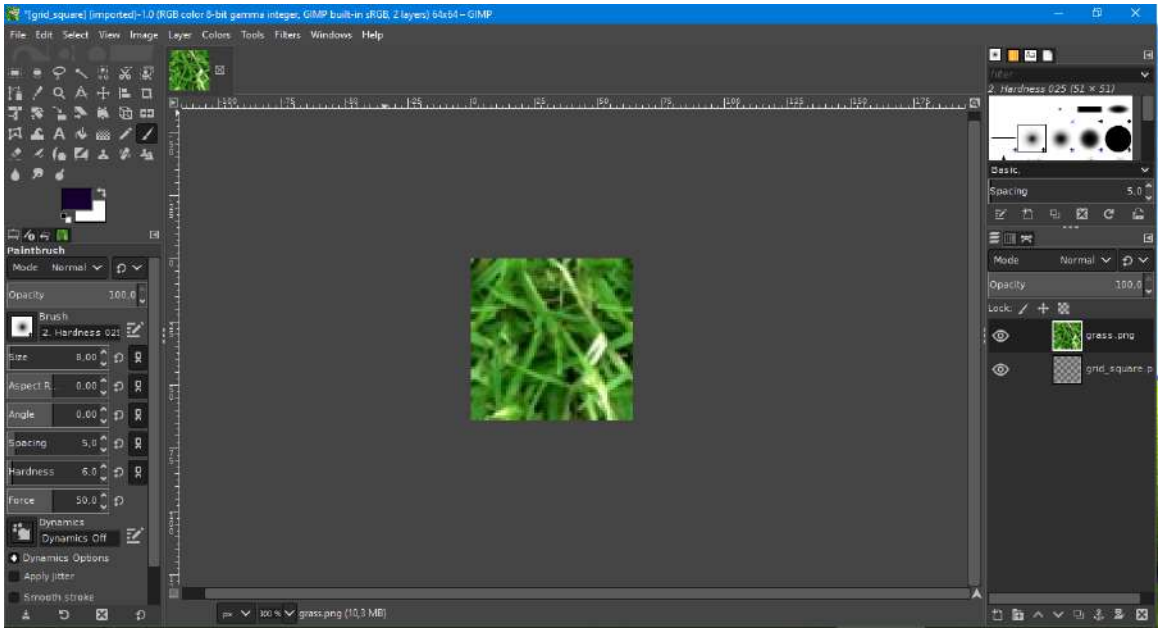
Σχήμα 4: Η γραφική διεπαφή του Audacity με ανοιχτό αρχείο της εργασίας



Σχήμα 5: Η γραφική διεπαφή του REAPER με ανοιχτό αρχείο της εργασίας

στη βιομηχανία της μουσικής παραγωγής κ.α. Μέσω του REAPER δημιουργήσαμε τον ήχο του κινητήρα, όπως θα δούμε στην αντίστοιχη παράγραφο.

Τέλος, για την παραγωγή των αρχείων εικόνας, καταλυτική ήταν η προσφορά του GIMP[8], το οποίο είναι πρόγραμμα επεξεργασίας τύπου raster. Με το GIMP δημιουργήθηκαν ή επεξεργάστηκαν οι εικόνες που είτε αποτελούν textures μοντέλων μέσα στο περιβάλλον, είτε είναι μέρος των στοιχείων της γραφικής διεπαφής της εργασίας (πλήκτρα, μπάρες προόδου κ.λπ.)



Σχήμα 6: Η γραφική διεπαφή του GIMP με ανοιχτό αρχείο της εργασίας

1.3 Μέθοδοι προσομοίωσης κυκλοφορίας κι επισκόπηση βιβλιογραφίας

Η προσομοίωση της κυκλοφορίας και της κίνησης στους δρόμους (traffic simulation) είναι ένα πρόβλημα το οποίο έχει μελετηθεί πολύ, ειδικά τα τελευταία χρόνια που η διαθέσιμη υπολογιστή ισχύς καθιστά δυνατή την σχεδίαση περίπλοκων μοντέλων και την ταυτόχρονη μελέτη μεγάλου αριθμού οχημάτων σε πραγματικό χρόνο.

Ένα σύστημα δεκάδων έως και χιλιάδων οχημάτων είναι αρκετά περίπλοκο και χαοτικό σε σύγκριση με ένα μηχανικό ή ηλεκτρικό σύστημα, καθώς επηρεάζεται σημαντικά από πολλούς και διάφορους παράγοντες που είναι αδύνατο να μοντελοποιήσουμε και να προβλέψουμε. Συνεπώς,

διάφορες μέθοδοι έχουν αναπτυχθεί ανά τους καιρούς, οι οποίες αναγκαστικά συμβιβάζονται σε κάποιον τομέα, αναλόγως τις απαιτήσεις, ώστε να παραμείνουν υλοποιήσιμες υπολογιστικά.

Οι υπάρχουσες αλλά και πιθανές μέθοδοι είναι τόσες πολλές, που χρήσιμο είναι να τις κατατάξουμε σύμφωνα με διάφορες ιδιότητές τους ώστε να έχουμε μια εποπτική εικόνα για τα υπέρ και τα κατά της κάθε μιας. Η κατάταξη γίνεται για παράδειγμα με το πως διαχειρίζονται την έννοια του χώρου και του χρόνου, καθώς και το πόσο επικεντρώνονται ξεχωριστά σε κάθε όχημα, ή αντιμετωπίζουν όλη τη ροή οχημάτων ως μια οντότητα. Στην παρακάτω εικόνα γίνεται ένας τέτοιος διαχωρισμός.

Types Of Simulation In Transportation

Time	State	Space		
		Continuous	Discrete	N/A
Continuous	Disc.	Real Transportation Systems * Traffic flow, pedestrians Dynamic traffic assignment		Discrete Event Systems * queueing inventory manufacturing
	Cont.	PDE Traffic flow models Pedestrian models		ODE vehicle motion car suspension queueing (fluid approx)
Discrete	Disc.		Cellular Automata * Traffic, pedestrians Land use Urban sprawl Random Number Generation	Discrete Event Simulation * queueing inventory manufacturing
	Cont.	Car-following models * Microscopic traffic flow models *	Numerical PDE methods Godunov, Variational	Numerical ODE methods Euler, Runga-Kutta time-series * ARIMA
N/A	Disc. or Cont.	Monte Carlo method * : use of pseudo-random number Simulation of static probabilistic problems Integration, Optimization		Econometric models trip generation, distribution, modal split Optimization static traffic assignment

Σχήμα 7: Τύποι προσομοίωσης κυκλοφορίας. Πηγή: Jorge Laval – wikipedia.org

Βλέπουμε πως ο κύριος διαχωρισμός γίνεται με βάση το αν ο χώρος, ο χρόνος και η κατάσταση ορίζονται διακριτά ή συνεχώς. Για παράδειγμα, μια μακροσκοπική μέθοδος που χρησιμοποιείται συχνά για συστήματα πολύ μεγάλης κλίμακας (χιλιάδες οχήματα, περίπλοκα

οδικά συστήματα σε εύρος ενός δήμου κ.λπ.) είναι η θεώρηση πως η ροή των οχημάτων στους δρόμους προσεγγίζεται από τη ροή ενός ρευστού σε ένα σύστημα σωλήνων [10]. Σε αυτήν την περίπτωση έχουμε πολύ μικρή ακρίβεια στη νοημοσύνη του κάθε οχήματος, που περιορίζεται στις αλληλεπιδράσεις μεταξύ των σωματιδίων του ρευστού, και περιγράφεται από μερικές διαφορικές εξισώσεις, η επίλυση των οποίων γίνεται μέσω αριθμητικών μεθόδων ολοκλήρωσης. Λόγω αυτής της απλοποίησης όμως μπορούμε να κρατήσουμε χαμηλά το υπολογιστικό κόστος και να έχουμε ικανοποιητικά αποτελέσματα που μπορούν να φανερώσουν άγνωστα πιθανά προβλήματα σε ένα υπάρχον σχέδιο.

Σε αυτήν την εργασία προσπαθούμε να σχεδιάσουμε ένα μοντέλο που πληροί κάποιες επιθυμητές προδιαγραφές και ταυτόχρονα αντιμετωπίζει κάποια συγκεκριμένα προβλήματα αντί να τα προσπερνά. Στον παραπάνω πίνακα, θα κατατάσσαμε το μοντέλο στην πράσινη περιοχή, δηλαδή σε αυτά που λειτουργούν σε μικροσκοπικό επίπεδο, που ''ακολουθούν'' το κάθε όχημα ξεχωριστά και του εφαρμόζουν ανεξάρτητη αλλά και επηρεαζόμενη από τα υπόλοιπα οχήματα νοημοσύνη. Γνωστό λογισμικό σε αυτήν την κατηγορία είναι το SUMO[14] (Simulation of Urban MObility) το οποίο εφαρμόζει μικροσκοπικά και συνεχή μοντέλα για την προσομοίωση της κυκλοφορίας.

Άλλος μοντέρνος προσομοιωτής με μικροσκοπική προσομοίωση είναι ο CARLA [15], που είναι λογισμικό ανοικτού κώδικα για την ανάπτυξη, την εκπαίδευση και την αποτίμηση συστημάτων αυτόνομης οδήγησης. Παρέχει τη δυνατότητα σχεδίασης χαρτών, με μια βιβλιοθήκη από αντικείμενα ανοικτών δικαιωμάτων που μπορούν να χρησιμοποιηθούν για την δημιουργία κάποιου περιβάλλοντος πόλεως, με ρυθμιζόμενες περιβαλλοντικές συνθήκες, χρήση μηχανής φυσικής για την προσομοίωση της συμπεριφοράς οχημάτων και προσομοίωση καθιερωμένων συστημάτων ανίχνευσης (LIDAR, GPS. αισθητήρες απόστασης κ.α). Επίσης παρέχει διεπαφή με γνωστά πρωτόκολλα επικοινωνίας.

Το Traffic Flow Dynamics [11] [12] είναι ένα βιβλίο στον τομέα της δυναμικής της ροής κυκλοφοριακής κίνησης το οποίο περιλαμβάνει πληροφορίες για μακροσκοπικά και μικροσκοπικά μοντέλα. Τα μακροσκοπικά μοντέλα περιγράφουν τη ροή σε όρους πυκνότητας, ενώ τα μικροσκοπικά σε όρους οχήματος/οδηγού. Τελικά παρουσιάζει το θεωρητικό πλαίσιο στην πράξη μέσω εφαρμογών όπως εκτίμηση χρόνου ταξιδιού, έξυπνα συστήματα μεταφοράς, καθώς και ένα μοντέλο κατανάλωσης

καυσίμου και εκπομπών. Το παραπάνω βιβλίο εφαρμόζεται μέσω μικροσκοπικού μοντέλου στον online προσομοιωτή traffic-simulation.de [13] που αφήνει στον χρήστη αρκετές διαθέσιμες παραμέτρους, όπως την ροή εισόδου των οχημάτων, το ποσοστό των βαρέων οχημάτων (πχ φορτηγά) καθώς και παραμέτρους που αφορούν τον οδηγό/όχημα (επιθετικότητα, επιτάχυνση κ.α). Δίνει τέλος τη δυνατότητα επιλογής από κάποια συνήθη προκαθορισμένα σχήματα δρόμων (κυκλικός κόμβος κ.α).

Εδώ δανειζόμαστε χαρακτηριστικά από τις προαναφερθείσες δουλειές και βελτιώνουμε σε συγκεκριμένα επίπεδα ενδιαφέροντος.

Αρχικά, λαμβάνουμε σοβαρά υπ' όψην τη φυσική των οχημάτων, κάτι που συχνά αγνοείται για λόγους μείωσης πολυπλοκότητας του προγράμματος αλλά και μικρότερου υπολογιστικού κόστος ανά όχημα. Η ρεαλιστική φυσική παίζει σημαντικό ρόλο στην αλληλεπίδραση αλλά και στην ατομική συμπεριφορά των οχημάτων, καθώς πέρα από τις περιπτώσεις συγκρούσεων και αποφυγής αυτών, η γεωμετρία του δρόμου επηρεάζει άμεσα τις δυνατότητες επιτάχυνσης, επιβράδυνσης και αλλαγής κατεύθυνσης των οχημάτων. Η προσομοίωση του κινητήρα εσωτερικής καύσης που υλοποιήθηκε, εισάγει νέες μεταβλητές στο πρόβλημα, καθώς ένα συγκεκριμένο σχέδιο μπορεί να εξαναγκάζει τους οδηγούς σε αρκετά μη οικονομική οδήγηση, μέσω συχνών συμφορήσεων ή συχνών μεταβολών ταχύτητας.

Έπειτα, στο κάθε όχημα παρέχεται ένα σύνολο ατομικών δεδομένων και μεταβλητών, που αφορούν τις φυσικές του ιδιότητες, παραμέτρους ρύθμισης της συμπεριφοράς του οδηγού αλλά και τον ίδιο χειρισμό του οχήματος. Το τελευταίο γίνεται με αναλυτικό τρόπο, με ανάλυση της τροχιάς που πρόκειται να διασχίσει και υπολογισμό κατάλληλων τιμών για το χειρισμό του τιμονιού και των πεταλιών γκαζιού και φρένου. Ανά πάσα στιγμή επίσης αναλύεται η γειτονική περιοχή για ύπαρξη άλλων οχημάτων και ανάλογη προσαρμογή της συμπεριφοράς.

Τέλος, η αναπαράσταση των δρόμων σχεδιάστηκε έτσι ώστε να δίνει όσο το δυνατόν μεγαλύτερη ευελιξία χωρίς περιορισμό στη γεωμετρία και να είναι δυνατός ο σχεδιασμός και η μελέτη μη συμβατικών οδικών συστημάτων. Πολλές φορές σε αντίστοιχα λογισμικά βλέπουμε, για παράδειγμα, πως περιοριζόμαστε σε διασταυρώσεις τεσσάρων δρόμων και σχήματος τέλειου σταυρού, ή σε κυκλικούς κόμβους με αρκετά προβλεπόμενο σχήμα και εισόδους/εξόδους. Προφανώς,

το πως αντιλαμβάνεται η προσομοίωση την οντότητα του δρόμου επηρεάζει σημαντικά και το μοντέλο της νοημοσύνης των οδηγών, συνεπώς αυτά τα δυο αναπτύχθηκαν παράλληλα σε σημαντικό βαθμό ώστε να υπάρχει συμβατότητα μεταξύ τους.

2 Σχεδίαση Δρόμων

2.1 Καμπύλες Bezier

Όπως αναφέρθηκε, σκοπός στο σχεδιασμό δρόμων είναι να παρέχεται μεγάλη ευελιξία στη γεωμετρία για να μπορούν να αποτιμηθούν και να μελετηθούν μη συμβατικά σχέδια, ενώ παράλληλα να είναι δυνατόν για τον κάθε οδηγό να αντιληφθεί και να καθορίσει ποια τροχιά θα πρέπει να ακολουθήσει.

Κατάλληλες για την περίπτωση αυτή είναι οι καμπύλες Bézier, στο όνομα του μηχανικού Pierre Bézier, που τις χρησιμοποίησε για το σχεδιασμό αυτοκινήτων στην αυτοκινητοβιομηχανία της Renault [16]. Οι καμπύλες αυτές είναι παραμετρικές καμπύλες που περιγράφονται από εξισώσεις μιας παραμέτρου t, η οποία βρίσκεται στο εύρος $[0, 1]$. Ένα σημείο πάνω στην καμπύλη έχει μια μοναδική τιμή της παραμέτρου, με το 0 να σημαίνει την αρχή της καμπύλης και το 1 το τέλος της.

Μια καμπύλη Bézier ορίζεται από το αρχικό σημείο, το τελικό σημείο κι έναν αριθμό σημείων ελέγχου. Με τη μεταβολή της παραμέτρου t μετακινούμαστε μεταξύ αρχικού-τελικού σημείου με τετραγωνική παρεμβολή (quadratic interpolation) παράγοντας έτσι μια καμπύλη 2ου βαθμού, με το σημείο ελέγχου να καθορίζει την καμπυλότητα.

Με τρία σημεία έχουμε τη λεγόμενη τετραγωνική (quadratic) Bézier, ενώ αυξάνοντας τα σημεία ελέγχου ανεβαίνουμε τάξη και μπορούμε να έχουμε περισσότερες αλλαγές καμπυλότητας. Εδώ θα χρησιμοποιήσουμε την κυβική Bézier (cubic) που ορίζεται από τέσσερα σημεία, δηλαδή το αρχικό, το τελικό και δύο σημεία ελέγχου (στο εξής θα αναφερόμαστε και στα τέσσερα ώς σημεία ελέγχου). Όντας 3ου βαθμού, μπορούμε να αλλάξουμε μια φορά καμπυλότητα.

Η γενική μορφή μιας τέτοιας καμπύλης στις N διαστάσεις είναι η εξής:

$$\begin{bmatrix} B_0(t) \\ B_1(t) \\ \vdots \\ B_N(t) \end{bmatrix} = (1-t)^3 \begin{bmatrix} P_{00}(t) \\ P_{01}(t) \\ \vdots \\ P_{0N}(t) \end{bmatrix} + 3(1-t)^2 t \begin{bmatrix} P_{10}(t) \\ P_{10}(t) \\ \vdots \\ P_{1N}(t) \end{bmatrix} + 3(1-t)t^2 \begin{bmatrix} P_{20}(t) \\ P_{21}(t) \\ \vdots \\ P_{2N}(t) \end{bmatrix} + t^3 \begin{bmatrix} P_{30}(t) \\ P_{31}(t) \\ \vdots \\ P_{3N}(t) \end{bmatrix} \quad (1)$$

όπου $B_k(t)$ το σημείο της καμπύλης που θέλουμε να υπολογίσουμε, και P_{0k}, P_{1k}, P_{2k} και P_{3k} τα αντίστοιχα σημεία ελέγχου στην k διάσταση. Όπως θα δούμε στη συνέχεια, οι καμπύλες αυτές βρίσκονται στη βάση της σχεδίασης των δρόμων, μέσω καμπυλών που ορίζουν τα όρια των λωρίδων και κατ' επέκταση τις τροχιές των οχημάτων.

Η υλοποίηση των καμπυλών Bézier στη C# γίνεται κατ' αρχάς δηλώνοντας μια κλάση με αντίστοιχο όνομα και μέληA.3. Έτσι μπορούμε να φτιάξουμε συναρτήσεις οι οποίες να δημιουργούν καμπύλες Bezier από κάποια δεδομένα σημεία ελέγχου. Μια τέτοια συνάρτηση είναι η CubicBezierFromPoints A.4.

Η συνάρτηση αυτή παίρνει ως ορίσματα τέσσερα σημεία ελέγχου, εφόσον μιλάμε για καμπύλη 3ου βαθμού, καθώς και μια παράμετρο res, εκ του resolution. Αυτή είναι η ανάλυση στην οποία θα χωρίσουμε την καμπύλη, δηλαδή ο αριθμός των βημάτων. Μέσα στην συνάρτηση, υπολογίζουμε την παράμετρο t σε κάθε βήμα, η οποία είναι ο λόγος του τρέχοντος βήματος προς τα συνολικά βήματα που θα γίνουν. Αυτό θα δώσει στην παράμετρο ένα εύρος από το 0 έως το 1. Έπειτα, για κάθε σημείο υπολογίζουμε τις συντεταγμένες κατά τον ορισμό της καμπύλης, και το προσθέτουμε σε μια λίστα, την οποία και θα επιστρέψουμε στο τέλος της συνάρτησης.

Η ανάλυση όπως βλέπουμε δίνεται σαν μεταβλητή τύπου float, παρόλο που αφορά ακέραιο αριθμό βημάτων. Ο λόγος θα φανεί σε επόμενη παράγραφο, που θα μιλήσουμε για την χρήση αυτής της συνάρτησης ως μέρος της δημιουργίας της τροχιάς των οχημάτων.

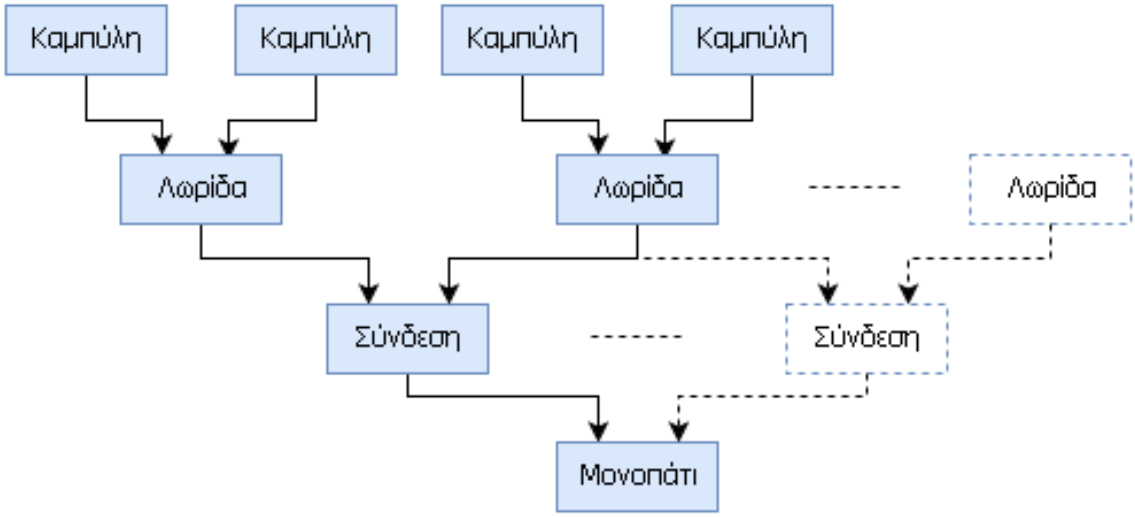
Μια ακόμα χρήσιμη συνάρτηση που δημιουργήθηκε και αφορά την καμπύλη Bezier, είναι η BezierPointAtParameterVal, η οποία μας επιστρέφει το σημείο που βρίσκεται στην καμπύλη για συγκεκριμένη τιμή της παραμέτρου A.5.

Παρόμοια με την προηγούμενη, αυτή η συνάρτηση δέχεται ως ορίσματα τα τέσσερα σημεία της καμπύλης, καθώς και μια τιμή της παραμέτρου t. Ακολουθώντας πάλι τον ορισμό, επιστρέφει το σημείο με τις συντεταγμένες $[x, y, z]$ που αντιστοιχούν στη συγκεκριμένη τιμή της παραμέτρου. Να σημειωθεί πως επειδή στην εργασία αυτή καταπιανόμαστε με δρόμους οι οποίοι κείτονται σε επίπεδες επιφάνειες, δε λαμβάνουμε υπόψην το ύψος, συνεπώς τα σημεία που χειριζόμαστε είναι της μορφής $[x, 0, z]$.

Στην επόμενη παράγραφο θα δούμε πως χρησιμοποιείται το συγκεκριμένο είδος καμπύλης για την αναπαράσταση των δρόμων, και σε ποιες οντότητες αυτή χωρίζεται.

2.2 Αναπαράσταση δρόμων

Η αναπαράσταση του οδικού συστήματος είναι μείζωνος σημασίας και είναι πολύ στενά συνδεδεμένη με την ίδια την προσομοίωση, αφού είναι στην ουσία το τι θα "βλέπουν" οι πράκτορες όσο υπολογίζουν την επόμενή τους κίνηση. Ταυτόχρονα, πρέπει για λόγους χρηστικούς, να έχουν και κάποια αντιστοίχιση με την καθημερινή αντίληψη που έχει ο άνθρωπος για ένα οδικό σύστημα. Η παρακάτω εικόνα δείχνει το γενικότερο πλάνο της οργάνωσης των οντοτήτων που αποτελούν το οδικό σύστημα:



Σχήμα 8: Η οργάνωση του οδικού συστήματος

Όπως φαίνεται λοιπόν, τελικά η οντότητα που αντιπροσωπεύει ένα κομμάτι δρόμου, είναι το μονοπάτι (path). Το μονοπάτι αποτελείται από τη σύνδεση (connection) πολλών λωρίδων (lane), οι οποίες ορίζονται από δυο καμπύλες (curve), την αριστερή και τη δεξιά. Συνεπώς κάθε λωρίδα αποτελείται από ακριβώς δυο καμπύλες, κάθε σύνδεση από ακριβώς δυο λωρίδες, και κάθε μονοπάτι από έναν αόριστο αριθμό συνδέσεων. Θα ορίσουμε τις αντίστοιχες οντότητες

στον κώδικα.

Ξεκινάμε με τον ορισμό του ορίου της λωρίδας A.6. Είναι μια κλάση που περιλαμβάνει μια αναφορά (reference) σε ένα GameObject (αυτό αφορά τη Unity), ένα ID που χρησιμεύει στο να ταυτοποιεί το συγκεκριμένο όριο-καμπύλη, καθώς και τα τέσσερα σημεία ελέγχου που ορίζουν την καμπύλη. Επίσης ορίζονται και δυο constructors, που συντομεύουν τη δημιουργία ενός αντικειμένου από αυτήν την κλάση με μια ταυτόχρονη αρχικοποίηση κάποιων παραμέτρων του.

'Επειτα ακολουθεί ο ορισμός της λωρίδας A.7. Ο ορισμός αυτός είναι λίγο πιο περίπλοκος. Αρχικά βλέπουμε πως κι εδώ έχουμε μια μεταβλητή που κρατά τον αριθμό ταυτοποίησης της λωρίδας (laneID). Έπειτα, έχουμε δυο ακέραιες μεταβλητές που αναφέρονται στα IDs των ορίων που ορίσαμε πριν, ένα για το αριστερό κι ένα για το δεξιό. Επίσης, έχουμε και δυο boolean μεταβλητές, με τις οποίες μπορούμε να αντιστρέψουμε (invert) τη σειρά των σημείων ελέγχου της καμπύλης. Αυτό είναι χρήσιμο για παράδειγμα αν θέλουμε να αντιστρέψουμε τη ροή κυκλοφορίας μιας λωρίδας χωρίς να χρειαστεί να την ξανασχεδιάσουμε ανάποδα από το μηδέν. Μια bool μεταβλητή (correctlyBound) κρατά το αν αυτή η λωρίδα έχει συνδεθεί σωστά κάπου, μια μεταβλητή τύπου LaneIndicator κρατά ένα αντικείμενο της Unity το οποίο δείχνει οπτικά τη θέση της λωρίδας στο χώρο, και μια λίστα ακεραίων connsTo κρατά τους δείκτες των λωρίδων με τις οποίες αυτή η λωρίδα ενώνεται.

Δυο ακόμα μεταβλητές έχουν έναν ειδικό τύπο, συγκεκριμένα η leftBorderType/rightBorderType και mode. Η πρώτη αφορά στον τύπο του εκάστοτε ορίου της λωρίδας, και η δεύτερη στη λειτουργία της. Ο τύπος του ορίου μιας λωρίδας είναι ένας enumeratorA.8, αφορά στη διαγράμμιση, που προφανώς έχει μεγάλη επίπτωση στην σημασία της στο δρόμο .

Η λειτουργία της λωρίδας πάλι ορίζεται ως enumerator A.9, και αφορά στο αν αυτή η λωρίδα είναι λωρίδα εκκίνησης, λωρίδα τερματισμού, ή τίποτα από τα δυο. Μέσω αυτής της μεταβλητής αποφασίζουμε το αν σε αυτή τη λωρίδα θα δημιουργείται ροή αυτοκινήτων, αν θα "καταστρέφονται" αυτοκίνητα ώστε να παύουν να υπάρχουν στην προσομοίωση, ή αν απλά είναι ενδιάμεση λωρίδα με καμία επίπτωση στη δημιουργία ή τερματισμό οχημάτων.

Τέλος, ορίζονται και κάποιοι constructors, που παρόμοια με τα όρια λωρίδων χρησιμεύουν στην συντομότερη δημιουργία αντικειμένων από αυτήν την κλάση, που όμως παραλείπονται χάριν συντομίας.

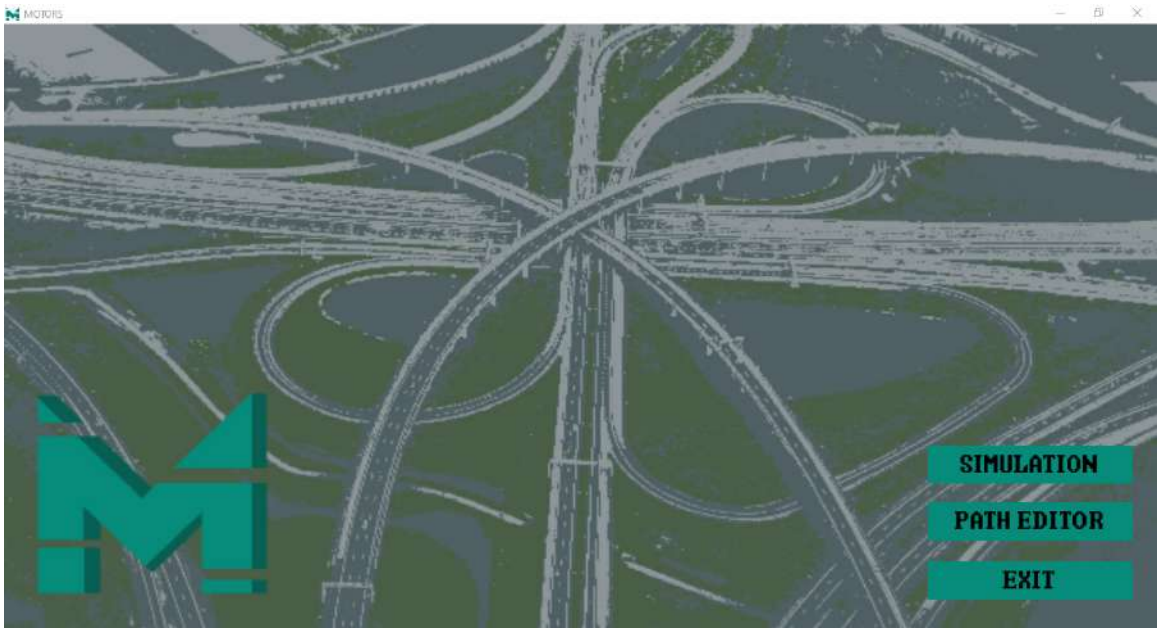
Όσον αφορά τέλος στην σύνδεση λωρίδων, ορίζεται μια αντίστοιχη κλάση A.10 η οποία περιέχει έναν δείκτη προς την λωρίδα από την οποία θα γίνει η σύνδεση, έναν προς τη λωρίδα προς την οποία θα γίνει η σύνδεση, μία bool που αφορά στο αν η λωρίδα τελειώνει σε STOP καθώς και έναν έλεγχο για το αν η σύνδεση έχει γίνει σωστά. Πάλι, με έναν χρηστικό constructor.

Ας δούμε τώρα πως χρησιμοποιούνται τα παραπάνω πρακτικά στο σχεδιασμό δρόμων, καθώς και το γενικότερο περιβάλλον της σχεδίασης. Το μονοπάτι, που δεν αναφέρθηκε, δεν είναι μέρος της σχεδίασης καθώς αποφασίζεται από το κάθε όχημα ξεχωριστά κι αφορά στη δικιά του πορεία, συνεπώς είναι μέρος μόνο της ίδιας της προσομοίωσης.

2.3 Το περιβάλλον σχεδίασης

2.3.1 Αρχική οθόνη

Για τη σχεδίαση των χαρτών της προσομοίωσης, δημιουργήθηκε ο Path Editor, ένα ξεχωριστό λογισμικό από την κύρια προσομοίωση, χρησιμοποιώντας τα εργαλεία που προσφέρει η Unity για σχεδίαση μιας γραφικής διεπαφής χρήστη (GUI). Αυτά αφορούν συνηθισμένα στοιχεία όπως κουμπιά, μπάρες προόδου, παράθυρα διαλόγων, εικόνες κι απλό κείμενο. Για να έχουμε πρόσβαση στον Path Editor, αρχικά ανοίγουμε το κύριο πρόγραμμα της εργασίας. Εκεί μας καλωσορίζει η αρχική οθόνη του προγράμματος:



Σχήμα 9: Η αρχική οθόνη του κύριου προγράμματος της εργασίας

Όπως βλέπουμε, έχουμε τρεις επιλογές: την ίδια την προσομοίωση, τον Path Editor καθώς και την έξοδο από το πρόγραμμα.

Στη Unity, η εναλλαγή αυτή έχει υλοποιηθεί με τη μέθοδο των σκηνών (Scenes). Η σκηνή στη Unity είναι μια οντότητα που περιέχει τη δικιά της ιεραρχία αντικειμένων, τις δικές της ρυθμίσεις καθώς και τη δικιά της λειτουργικότητα. Η αρχική οθόνη είναι μια απλή σκηνή, που περιέχει μόνο το φόντο και τρία κουμπιά. Η λειτουργικότητά της περιορίζεται στο να επιλέξουμε μια από τις τρεις επιλογές, η μία εκ των οποίων προφανώς κλείνει το πρόγραμμα, ενώ οι άλλες οδηγούν σε νέες σκηνές. Ο κώδικας της σκηνής αυτής είναι πολύ απλός A.11.

Αν επιλέξουμε το κουμπί της εξόδου, τότε απλώς τρέχουμε την εντολή της Unity που κλείνει το πρόγραμμα. Αν πατήσουμε το κουμπί της προσομοίωσης, τότε θα ανοίξει ένα παράθυρο διαλόγου της .NET με το οποίο θα διαλέξουμε το αρχείο χάρτη προς προσομοίωση. Αν πατήσουμε το κουμπί του Path Editor, τότε θα ανοίξει η αντίστοιχη σκηνή. Να σημειωθεί πως

όπως φαίνεται από τον κώδικα, δεν φορτώνονται άμεσα οι αντίστοιχες σκηνές, παρά φορτώνεται η σκηνή loading. Αυτή είναι μια ενδιάμεση σκηνή που χρησιμοποιείται για να δείξει στον χρήστη μια κίνηση (animation) για όση ώρα φορτώνεται η αντίστοιχη σκηνή. Η φόρτωση της σκηνής μπορεί να είναι κάποια δευτερόλεπτα, συνεπώς αυτό επιβεβαιώνει στον χρήστη πως το πρόγραμμα δεν ''κόλλησε'', παρά εκτελεί κάποιες λειτουργίες στο παρασκήνιο.

Σχήμα 10: Η σκηνη φόρτωσης (loading)

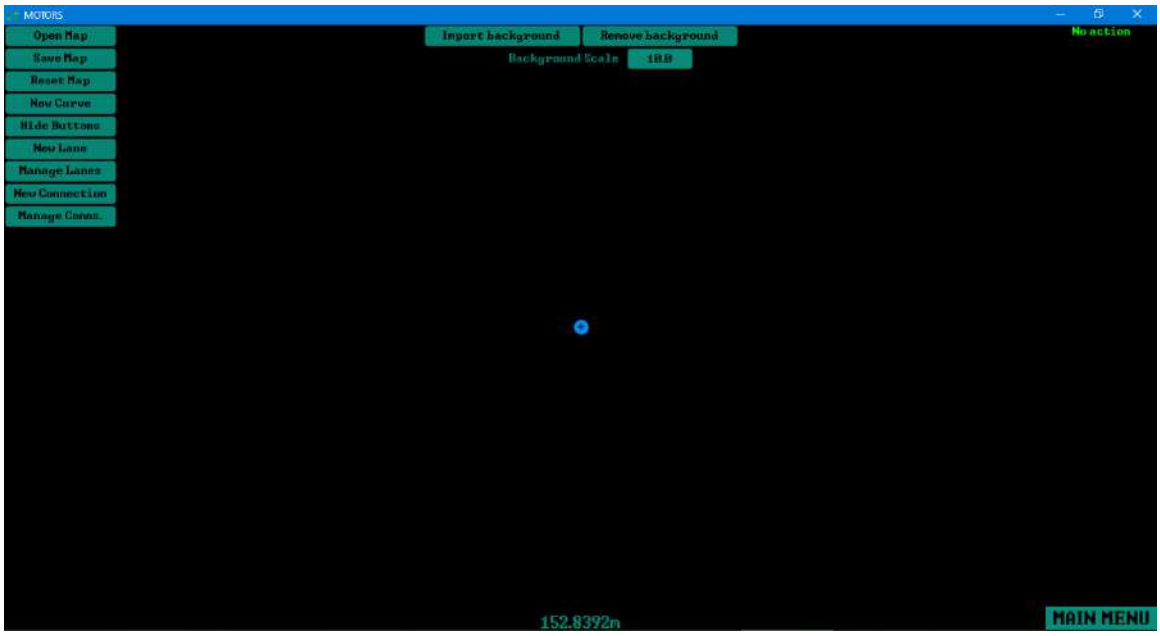
Κι αυτής της σκηνής η δομή είναι πολύ απλή, καθώς περιλαμβάνει απλά μια εικόνα ενός περιστρεφόμενου συμπαγούς δίσκου (CD), ως σύμβολο φόρτωσης δεδομένων. Με το που φορτώνει αυτή η σκηνή, φορτώνει άμεσα την τελική σκηνή, αναλόγως την τιμή της μεταβλητής choice της αρχικής σκηνής που φαίνεται στο προηγούμενο κομμάτι κώδικα. Ταυτόχρονα, εκτελεί την περιστροφή του δίσκου για να φαίνεται ξεκάθαρα πως το πρόγραμμα δεν έχει ''κολλήσει'' A.12.

Μια σημαντική λεπτομέρεια είναι πως εδώ δεν χρησιμοποιείται η εντολή LoadScene αλλά η LoadSceneAsync. Η δεύτερη, όπως μαρτυρά το όνομά της, εκτελεί φόρτωση σκηνής ασύγχρονα, δηλαδή δεν σταματά τις δικές τις λειτουργίες αλλά τις συνεχίζει ταυτόχρονα. Γι' αυτό το λόγο μπορεί και συνεχίζει να δουλεύει η συνάρτηση Update, στην οποία γίνεται η περιστροφή του δίσκου.

Προχωράμε τώρα πατώντας στο κουμπί PATH EDITOR. Αυτό ανοίγει την σκηνή φόρτωσης, κι αυτή με τη σειρά της το περιβάλλον σχεδίασης.

2.3.2 Εργαλεία σχεδίασης

Εισερχόμενοι στη σκηνή του Path Editor, βλέπουμε τα περιβάλλον σχεδίασης.

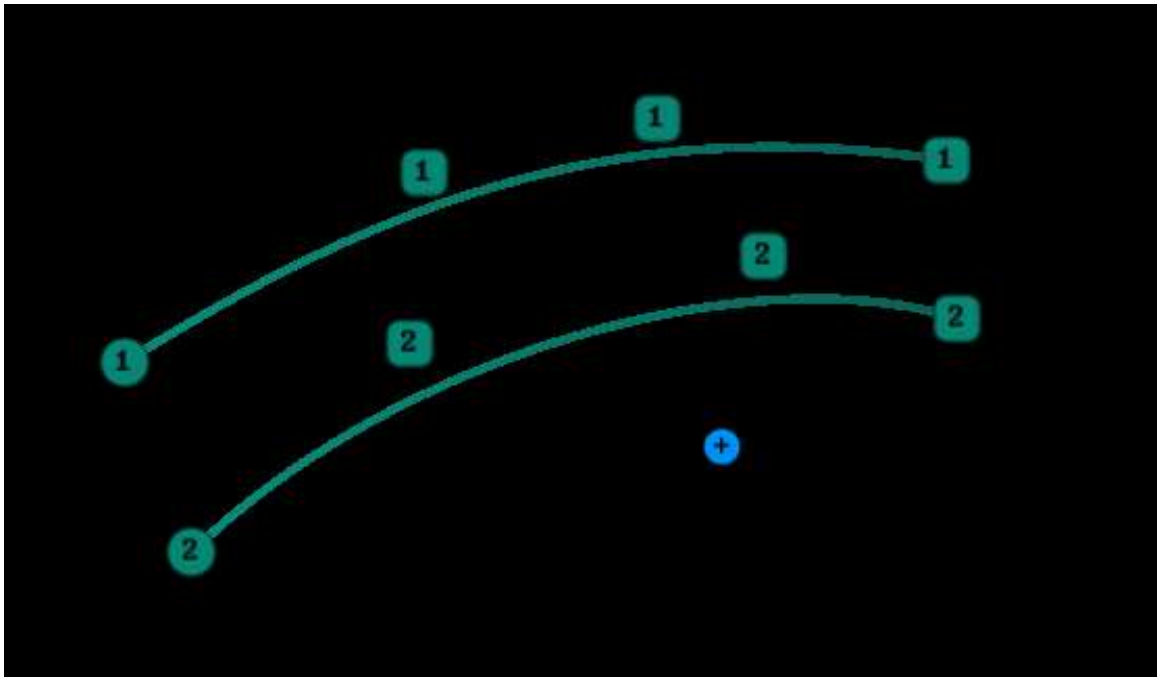


Σχήμα 11: Η αρχική κατάσταση της σκηνής του Path Editor

Στα αριστερά, βρίσκεται το μενού επιλογών για δημιουργία νέων στοιχείων GUI αλλά και διαχείρισης λειτουργιών. Στο πάνω μέρος έχουμε κάποια στοιχεία που αφορούν στο φόντο, με το οποίο μπορούμε να σχεδιάσουμε ένα χάρτη έχοντας αναφορά ένα ήδη υπάρχον σχέδιο, μια κάτοψη ή μια φωτογραφία. Πάνω δεξιά βρίσκεται μια ένδειξη που αναφέρει την τελευταία εκτελεσμένη ενέργεια, και το αν αυτή ήταν επιτυχής ή όχι. Κάτω δεξιά υπάρχει ένα κουμπί

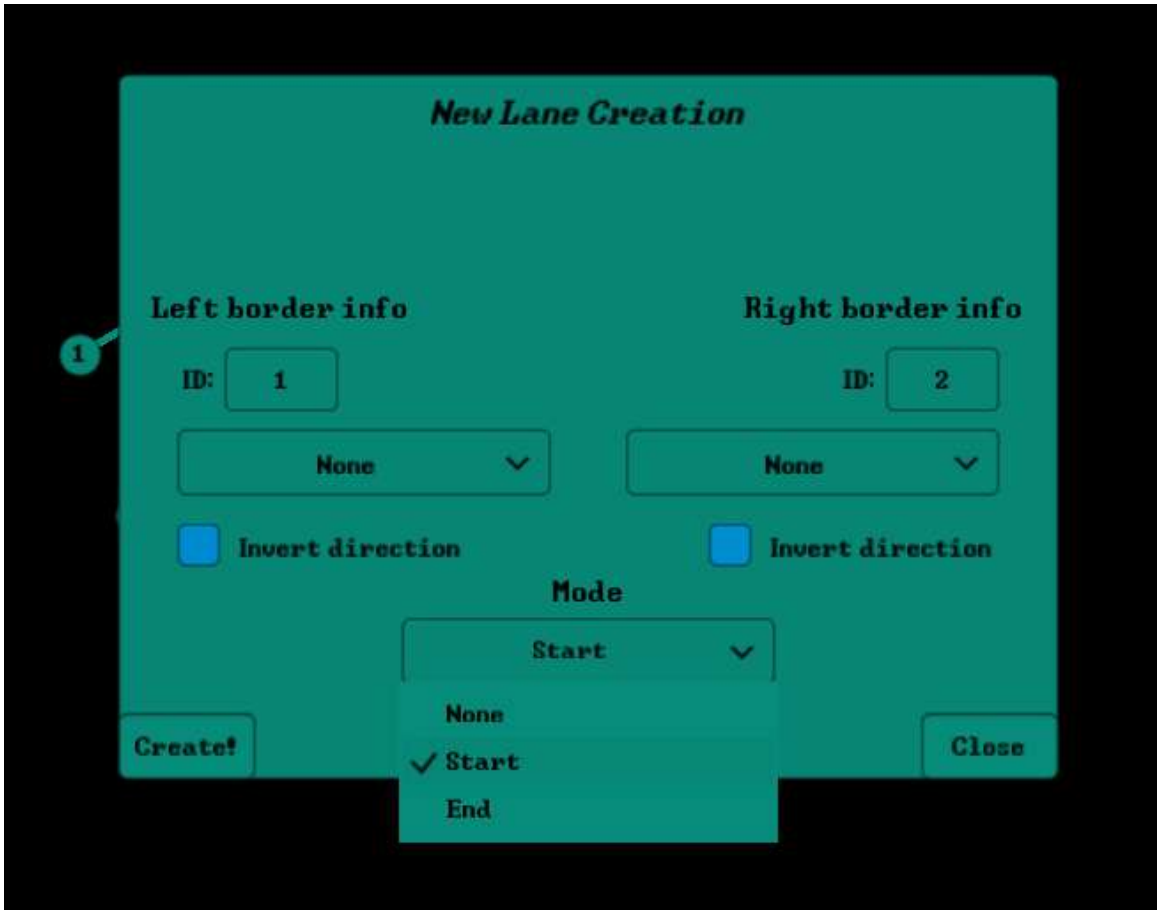
με το οποίο μπορούμε να πάμε στο αρχικό μενού. Στο κάτω μέρος υπάρχει μια ένδειξη για το μήκος, από άκρη σε άκρη, της περιοχής της κάτοψης που βλέπουμε αυτήν τη στιγμή. Τέλος, στη μέση υπάρχει μια ένδειξη του κέντρου της οθόνης για να υπάρχει ως αναφορά στο σχέδιο.

Για τη δημιουργία του χάρτη, ξεκινάμε επιλέγοντας New Curve. Θα δημιουργήσουμε μια καμπύλη, η οποία θα πάρει αυτόματα αρίθμηση. Μαζί με την καμπύλη δημιουργούνται και τέσσερα κουμπιά, ένα για κάθε σημείο ελέγχου της καμπύλης Bezier. Πατώντας τα, μπορούμε να σύρουμε το καθένα από αυτά και να φέρουμε την καμπύλη στην επιθυμητή θέση. Η καμπύλη χρησιμοποιεί το LineRenderer της Unity, που χρησιμοποιείται για σχεδιασμό γραμμών. Εφόσον θέλουμε να σχεδιάσουμε καμπύλες, χωρίζουμε την καμπύλη σε πολλά κομμάτια με τις συναρτήσεις που αναφέρθηκαν στην παράγραφο των καμπυλών Bezier, και προσθέτουμε στον LineRenderer τα ενδιάμεσα σημεία. Ας δημιουργήσουμε δυο παράλληλες καμπύλες:

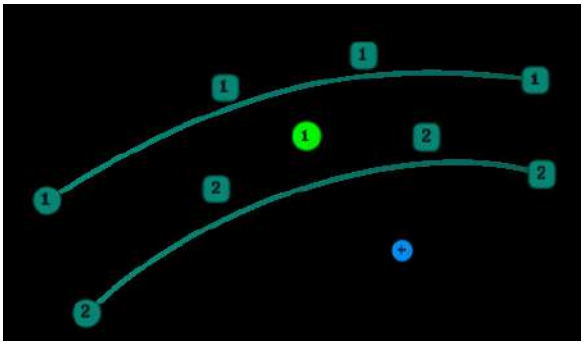


Σχήμα 12: Παράδειγμα σχεδιασμένων καμπυλών με το αντίστοιχο εργαλείο

Έπειτα, φτιάχνουμε μια νέα λωρίδα πατώντας το αντίστοιχο κουμπί αριστερά (New Lane). Εκεί καταχωρούμε το ID της αριστερής και δεξιάς καμπύλης, όπως φαίνονται από τις ενδείξεις τους. Επίσης ορίζουμε τους τύπους της διαγράμμισης, το αν τις θέλουμε στην ορθή ή αντίστροφη κατεύθυνση, καθώς και το αν πρόκειται για λωρίδα έναρξης ή τερματισμού. Στο συγκεκριμένο παράδειγμα τα ID είναι η 1 και 2 και επιλέγουμε να είναι λωρίδα έναρξης, συνεπώς προκύπτει αυτή η λωρίδα με το ID της να ισούται με 1:

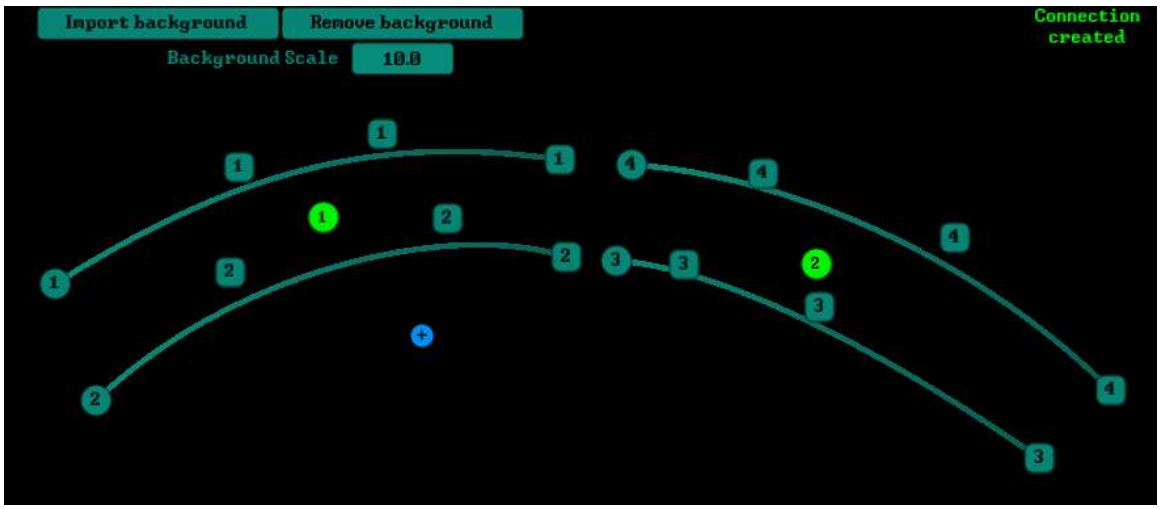


Σχήμα 13: Το παράθυρο διαλόγου της δημιουργίας λωρίδας



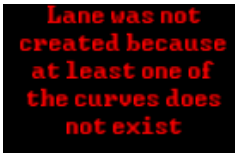
Σχήμα 14: Η δημιουργημένη λωρίδα, με μοναδικό αριθμό ταυτοποίησης

Αντίστοιχα φτιάχνουμε ένα δεύτερο ζευγάρι καμπυλών, και μια δεύτερη λωρίδα. Έπειτα, επιλέγοντας New Connection, δημιουργούμε μια νέα σύνδεση, η οποία παίρνει το ID 2. Επιλέγουμε της λωρίδες 1 και 2 και δημιουργούμε τη σύνδεση. Η ένδειξη πάνω δεξιά μας ενημερώνει για την επιτυχή δημιουργία της σύνδεσης, όπως μας είχε ενημερώσει και για τη δημιουργία καμπυλών και λωρίδων:



Σχήμα 15: Η δημιουργημένη σύνδεση, με επιβεβαίωση της επιτυχίας της σύνδεσης

'Ενα παράδειγμα αποτυχίας δημιουργίας, είναι το να δώσουμε στη δημιουργία λωρίδας ID καμπυλών που δεν υπάρχουν. Αν για παράδειγμα δώσουμε 5 και 6, τότε ενημερωνόμαστε αντίστοιχα:

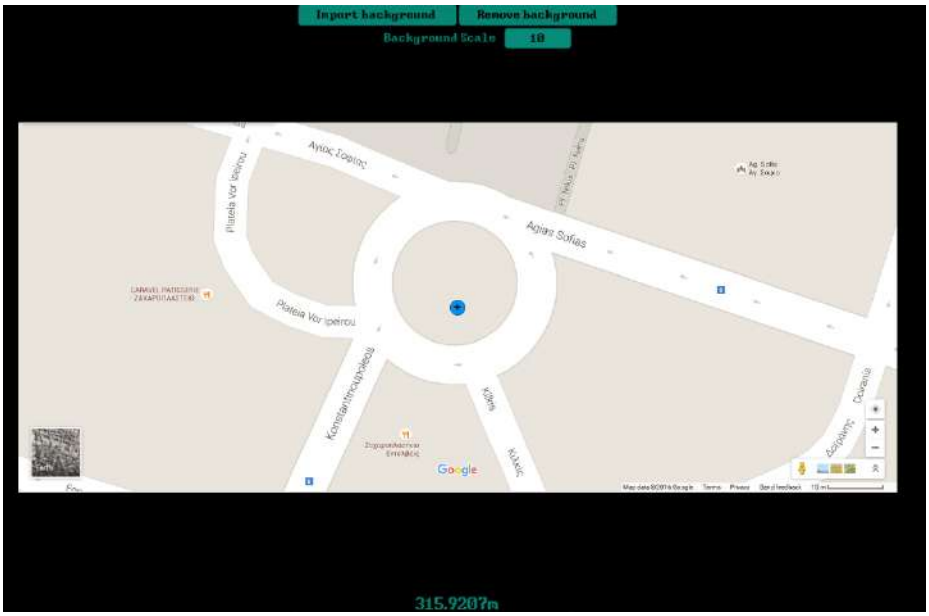


Σχήμα 16: Παράδειγμα αποτυχημένης λωρίδας, λόγω ανυπαρξίας των καμπυλών

Μπορούμε να δούμε εποπτικά το ποιες λωρίδες και συνδέσεις έχουμε φτιάξει, καθώς και να διαγράψουμε κάποιες από αυτές, με τους αντίστοιχους διαλόγους Manage Lanes και Manage Conns. Επίσης με το Reset Map σβήνονται όλες οι καμπύλες, λωρίδες και συνδέσεις από τον χάρτη. Με το Hide Buttons μπορούμε να αποκρύψουμε τα κουμπιά των καμπυλών, καθώς σε περίπλοκους χάρτες δυσκολεύει η ευδιακριτότητα των ίδιων των καμπυλών.

2.3.3 Εισαγωγή φόντου

Όπως αναφέρθηκε, υπάρχει η δυνατότητα εισαγωγής φόντου για αναφορά κατά τη σχεδίαση. Αυτό γίνεται με τη χρήση των πλήκτρων στο πάνω μέρος της οθόνης. Πατάμε λοιπόν Import Background και επιλέγουμε ένα αρχείο εικόνας μέσω του διαλόγου που ακολουθεί. Αφού η εικόνα εισαχθεί ως φόντο, μπορούμε να μετατοπιστούμε γύρω από το κέντρο μέσω των πλήκτρων W,A,S,D ώστε να φέρουμε το χάρτη στο επιθυμητό σημείο σε σχέση με το κέντρο. Αυτό μετατοπίζει κι όλες τις δημιουργημένες καμπύλες και λωρίδες. Επίσης, με τη ροδέλα του ποντικιού μπορούμε να μεγεθύνουμε την κάμερα ώστε το σχέδιό μας να έχει τη σωστή κλίμακα. Τέλος, μπορούμε να φέρουμε το φόντο στην επιθυμητή κλίμακα μέσω του πεδίου Background Scale.



Σχήμα 17: Παράδειγμα εισαγωγής φόντου, με εικόνα από το Google Maps

Η υλοποίηση Α.13 γίνεται δημιουργώντας ένα επίπεδο στο χώρο, και ορίζοντας την υφή (texture) αυτού του επιπέδου στην εικόνα που μόλις επιλέξαμε.

Αν υπάρχει ήδη επίπεδο, αντικαθίσταται το texture του, αλλιώς δημιουργείται νέο. Επίσης πρέπει να θέσουμε τον shader του επιπέδου σε Unlit/Texture, ειδάλλως θα χρησιμοποιήσει τον default shader (Standard) ο οποίος κάνει το μοντέλο να επηρεάζεται από το φωτισμό και τη σκίαση, κάτι που φυσικά δε θέλουμε.

2.3.4 Έλεγχος αρτιότητας χάρτη

Πριν φτάσουμε στο σημείο της αποθήκευσης του χάρτη σε ειδικό αρχείο και στο σημείο ανάκτησής του, πρέπει πρώτα να βεβαιωθούμε για την αρτιότητά του. Αυτό κρίνεται από το αν υπάρχουν καμπύλες που δεν έχουν τεθεί σε κάποια λωρίδα, αν υπάρχουν λωρίδες που δεν έχουν συνδεθεί μεταξύ τους, ή αν δεν υπάρχουν λωρίδες έναρξης και τερματισμού. Ο έλεγχος αρτιότητας γίνεται στη συνάρτηση CheckMapIntegrity Α.14

2.3.5 Ειδικός τύπος αρχείου χάρτη

Για την αποθήκευση του χάρτη και τη δυνατότητα μετέπειτα ανάκτησης, καθώς και τη διανομή ώστε να μπορεί να ανοιχτεί σε σύστημα άλλου χρήστη, απαραίτητη ήταν η δημιουργία κάποιου είδους αρχείου που θα ακολουθεί μια καλά ορισμένη δομή. Γι' αυτό το λόγο δημιουργήθηκε το ειδικό αρχείο τύπου MOTORS, δανειζόμενο το όνομα της εργασίας (MOdeling of Traffic for Optimization of Road Systems). Αυτό το αρχείο έχει επέκταση .motors και αυτή χρησιμοποιείται ως φίλτρο στα παράθυρα διαλόγου μέσω των οποίων ανοίγουμε και αποθηκεύουμε τέτοια αρχεία. Ας δούμε τη δομή ενός τέτοιου αρχείου:

```
1  p,1,63.5616,14.70026,2.129805,...
2  ...
3  p,15,10.66854,14.70026,-18.65211,...
4  l,1,1,False,1,2,False,4,1
5  ...
6  l,8,15,False,1,14,False,1,2
7  c,1,4,False,0,0,0
8  c,2,4,False,0,0,0
9  ...
10 c,6,3,False,0,0,0
11 s,10
12 i
13 %PNG
14 SUB
15 NUL NUL NUL
16 IHDR NUL NUL ENQ V NUL NUL STX ? BS STX NUL NUL NUL DC3 BEL NAK...
17 ...
```

Σχήμα 18: Παράδειγμα περιεχομένων και δομής αρχείου τύπου .motors

Οι χαρακτήρες p, l και c δείχνουν πως η εκάστοτε γραμμή αφορά σε καμπύλη, λωρίδα και σύνδεση αντίστοιχα. Το πρώτο επόμενο νούμερο αφορά στο ID της εκάστοτε οντότητας, και μετά ακολουθούν οι ακριβείς παράμετροι. Στην περίπτωση της καμπύλης ακολουθούν οι 12 συντεταγμένες των σημείων που την αποτελούν (3 x 4 σημεία), στη λωρίδα ακολουθεί το ID της, τα ID των καμπύλων που αποτελούν τα άκρα της καθώς και οι άλλες λεπτομέρειες, κι αντίστοιχα παρόμοιες λεπτομέρειες για τη σύνδεση. Ο χαρακτήρας s σηματοδοτεί αρχή γραμμής που περιέχει το νούμερο της κλίμακας του χάρτη. Τέλος, ο χαρακτήρας i σηματοδοτεί το τέλος των δεδομένων του χάρτη καθ' αυτού, και την αρχή των δεδομένων της εικόνας. Η εικόνα, όντας binary αρχείο που δεν είναι αναγνώσιμο από άνθρωπο, περιέχει χαρακτήρες που δεν βγάζουν

νόημα σε πρόγραμμα ανάγνωσης κειμένου όπως της φωτογραφίας, ωστόσο αντιπροσωπεύουν τα πραγματικά δεδομένα της εικόνας και διαβάζονται αντίστοιχα όπως θα δούμε στην επόμενη παράγραφο.

2.3.6 Αποθήκευση και ανάκτηση χάρτη

Έχοντας σχεδιάσει ένα χάρτη χρησιμοποιώντας τα εργαλεία που αναφέρθηκαν προηγουμένως, έχοντας σχεδιάσει καμπύλες, ορίσει λωρίδες και ενώσεις και έχοντας εισάγει φόντο και ορίσει κλίμακα, είμαστε σε θέση να αποθηκεύσουμε τον χάρτη, με δεδομένη τη μορφή που αναλύθηκε. Συνεπώς δεν έχουμε παρά να γράψουμε στο αρχείο μια γραμμή ανά καμπύλη, λωρίδα και σύνδεση, και στο τέλος να γράψουμε και τα binary δεδομένα της εικόνας.

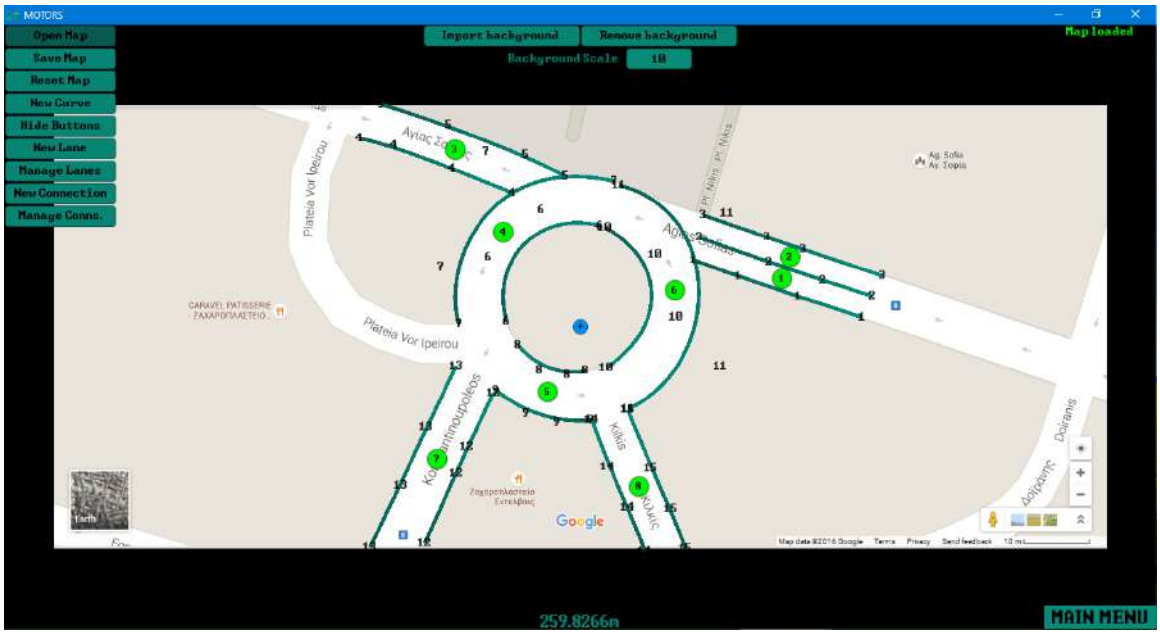
Επιλέγοντας λοιπόν από το μενού αριστερά Save Map, ανοίγει ο διάλογος στον οποίο θέτουμε το επιθυμητό όνομα του αρχείου, ενώ το φίλτρο έχει φροντίσει να βάλει τη σωστή επέκταση. Όταν επιβεβαιώσουμε την επιλογή μας, θα γίνει η αποθήκευση του αρχείου. Πριν την αποθήκευση, προηγείται εκτέλεση της συνάρτησης CheckMapIntegrity που αναλύθηκε, κι αν αποτύχει, τότε η αποθήκευση ακυρώνεται με αντίστοιχο μήνυμα. Η συνάρτηση OnSaveMapButtonClick Α.15πρώτα ελέγχει την αρτιότητα του χάρτη, κι έπειτα γράφει με τη σειρά τα δεδομένα σε ένα αρχείο .motors.

Για το άνοιγμα ενός αρχείου χάρτη, δεν έχουμε παρά να κάνουμε ακριβώς την αντίστροφη διαδικασία, δηλαδή να διαβάσουμε τα δεδομένα αυτά ανά γραμμή με τη γνωστή δομή. Πατώντας Open Map στο μενού, ανοίγει πάλι ένας διάλογος στον οποίο μπορούμε να επιλέξουμε το επιθυμητό αρχείο χάρτη. Ο αρχικός χαρακτήρας μας δίνει το σε ποια οντότητα αναφερόμαστε, και ο χαρακτήρας i σηματοδοτεί την έναρξη των δεδομένων της εικόνας του φόντου.

Η συνάρτηση υπεύθυνη για αυτήν την διαδικασία είναι η OnOpenMapButtonClick, η οποία είναι αρκετά πιο εκτενής από την OnSaveMapButtonClick, λόγω του ότι πέρα από την ανάγνωση των δεδομένων, πρέπει να δημιουργήσει από την αρχή και τα αντικείμενα που αποτελούν τον χάρτη. Τα δεδομένα του χάρτη διαβάζονται κατά τα γνωστά, ελέγχοντας μία μία γραμμή για τον αρχικό χαρακτήρα της Α.16 (οι τυπικές λεπτομέρειες διαβάσματος των γραμμών και δημιουργίας των αντικειμένων παραλείπονται).

Όταν διαβαστεί ο χαρακτήρας i όπως φαίνεται ο βρόχος θα τερματιστεί, στην οποία περίπτωση συνεχίζουμε με την ανάγνωση των δεδομένων της εικόνας. Η διαδικασία είναι σχεδόν πανομοιότυπη με τον τρόπο με τον οποίο κατασκευαζόταν το φόντο με τα δεδομένα της εικόνας που εισάγαμε. Η διαφορά είναι ότι αντί απλά να αντιγράψουμε τα δεδομένα του αρχείου στο texture του επιπέδου, πρέπει να τα βρούμε μέσα στο .motors αρχείο χειροκίνητα, ξεκινώντας να διαβάζουμε μετά τον i χαρακτήρα. Έπειτα δημιουργούμε το επίπεδο κατά τα γνωστά. A.17

Εδώ τελειώνει η περιγραφή του Path Editor, των λειτουργιών που εκτελεί και του πως αυτές υλοποιούνται συνοπτικά σε επίπεδο λογικής και κώδικα. Ας δούμε ένα παράδειγμα ενός σχεδιασμένου χάρτη, με φόντο αναφοράς και έτοιμες καμπύλες, λωρίδες και συνδέσεις, με κρυμμένα τα κουμπιά και εμφανή τα ID όλων των οντοτήτων:



Σχήμα 19: Παράδειγμα σχεδιασμένου χάρτη, μέσα στον Path Editor

Στο επόμενο κεφάλαιο θα περάσουμε στην προσομοίωση της φυσικής των οχημάτων, ώστε σε συνδυασμό με ό,τι είδαμε μέχρι τώρα να είμαστε σε θέση να αναλύσουμε μετέπειτα την ίδια την προσομοίωση της κυκλοφορίας στο κεφάλαιο 4.

3 Προσομοίωση Οχήματος

3.1 Κινητήρας εσωτερικής καύσης

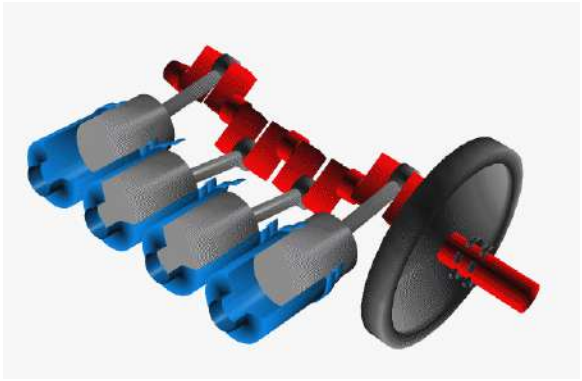
3.1.1 Αρχή λειτουργίας

Τα οχήματα που θα προσομοιωθούν στα πλαίσια αυτής της εργασίας, είναι συμβατικά οχήματα δρόμου, όπως αυτά που χρησιμοποιούνται από το μέσο άνθρωπο τον τελευταίο αιώνα.

Η αιτία μετατόπισης σε τέτοια οχήματα, είναι μια κύρια πηγή ροπής, που ονομάζεται κινητήρας. Η ενέργεια μπορεί να παρέχεται με τη μορφή συνεχόμενης ροπής, μπορεί ωστόσο να είναι και μια πηγή διαμήκους δυνάμεως, όπως στην περίπτωση των κινητήρων jet, που λειτουργούν με την αρχή της δράσης-αντίδρασης, με την βίαια εκτόξευση ενός ρευστού προς μια κατεύθυνση να έχει ως αποτέλεσμα την δύναμη προς την αντίθετη κατεύθυνση [17]. Ωστόσο τέτοιοι κινητήρες βρίσκουν εφαρμογή σε άλλους κλάδους (αεροναυπηγική, διαστημόπλοια) κι όχι τόσο σε καθημερινό επίπεδο, λόγω του μεγάλου κόστους, πολυπλοκότητας της διάταξης, χαμηλής ενεργειακής απόδοσης και θεμάτων ασφαλείας.

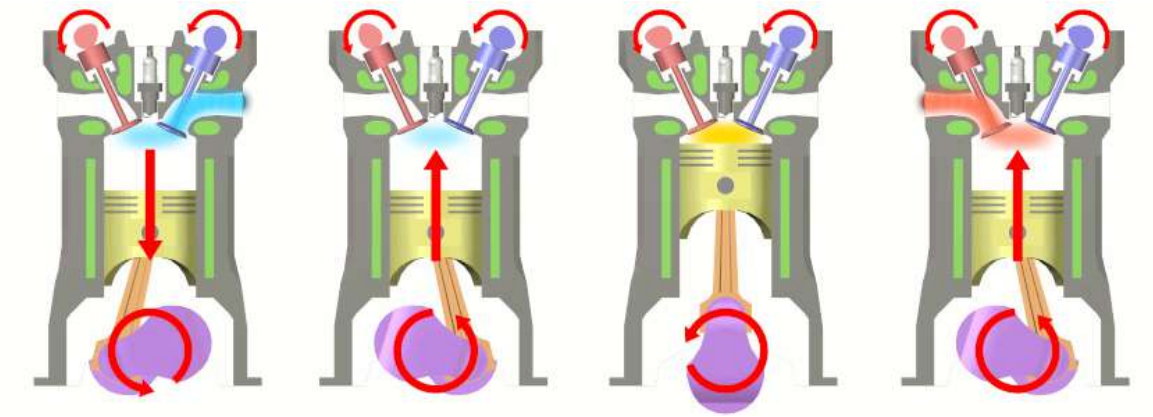
Στα καθημερινά οχήματα συγκεκριμένα χρησιμοποιείται ο παλινδρομικός κινητήρας εσωτερικής καύσης (Reciprocating Internal Combustion Engine). Στο παρελθόν, τέτοιου είδους κίνηση δημιουργούνταν από κινητήρες εξωτερικής καύσης, όπως ο κινητήρας ατμού [18]. Η βασική λογική είναι η παραγωγή κίνησης σε ένα έμβολο μέσω δημιουργίας διαφοράς πίεσης σε δυο δοχεία. Η κίνηση του εμβόλου αυτού, αν κι αρχικά είναι διαμήκης, μπορεί να μετατραπεί σε περιστροφική χάριν στο γεγονός πως είναι παλινδρομική κι όχι μονής κατεύθυνσης. Η μετατροπή αυτή γίνεται μέσω ειδικών διατάξεων που χωρίζονται σε διάφορες κατηγορίες. Ο James Watt πέρασε από διάφορους τύπους τέτοιων διατάξεων [19] για την ατμομηχανή του.

Στις μέρες μας, η διάταξη που είναι υπεύθυνη γι αυτήν την μετατροπή λέγεται στροφαλοφόρος άξονας (crankshaft). Είναι ένας άξονας που φιλοξενεί μια σειρά από στρόφαλους (cranks), στους οποίους στηρίζονται τα άκρα παλινδρομικών κυλίνδρων μέσω εμβόλων. Στην άκρη του άξονα αυτού, μπορούμε να πάρουμε πλέον καθαρή περιστροφική κίνηση, όπως φαίνεται στην εικόνα που ακολουθεί.



Σχήμα 20: Στροφαλοφόρος άξονας (κόκκινο) με κυλίνδρους (γκρι) [20]

Ας δούμε τώρα το πως παράγεται αυτή η παλινδρομική κίνηση των κυλίνδρων. Μιλήσαμε ήδη για την ατμομηχανή, που είναι είδος κινητήρα εξωτερικής καύσης, διότι η ενέργεια παρέχεται από εξωτερικά θερμαινόμενο υγρό, χωρίς ύπαρξη καυσίμων μέσα στον κινητήρα. Οι σύγχρονοι κινητήρες είναι εσωτερικής καύσης, διότι αυτή γίνεται εσωτερικά σε θάλαμο στον οποίο αναμειγνύονται αέρας με καύσιμο. Μια συσκευή που ονομάζεται αναφλεκτήρας, αναφλέγει το μίγμα αυτό μέσω μιας σπίθας υψηλής ενέργειας. Η όλη διαδικασία μπορεί να έχει πολλές παραλλαγές, ωστόσο η πιο δημοφιλής για τα αυτοκίνητα δρόμου είναι η τετράχρονη λειτουργία, που δίνει το όνομά της και στους τετράχρονους κινητήρες.



Σχήμα 21: Οι 4 χρόνοι του τετράχρονου κινητήρα. Με μωβ διακρίνεται ο στροφαλοφόρος άξονας[21]

Οι 4 χρόνοι είναι η εισαγωγή, η συμπίεση, η ανάφλεξη κι η εκτόνωση. Συγκεκριμένα στην εισαγωγή ειδικές βαλβίδες επιτρέπουν σε πολύ καλά ελεγχόμενες ποσότητες αέρα και καυσίμου να μπουν στο θάλαμο με τον κύλινδρο να κατεβαίνει ξεκινώντας τον πρώτο κύκλο του. Στην συμπίεση ο κύλινδος ανεβαίνει συμπιέζοντας το μίγμα προς τον αναφλεκτήρα, με τις βαλβίδες κλειστές. Στην ανάφλεξη η σπίθα του αναφλεκτήρα δημιουργεί μια ενεργειακή εκτόνωση που σπρώχνει τον κύλινδρο προς τα κάτω. Τέλος, στην εκτόνωση ο κύλινδρος ανεβαίνοντας στον τελευταίο κύκλο σπρώχνει τα παραγόμενα καυσαέρια προς τις βαλβίδες εξόδου, που έχουν φροντίσει να είναι ανοιχτές στη σωστή στιγμή. Ο χρονισμός αυτός των βαλβίδων γίνεται μέσω του λεγόμενου εκκεντροφόρου άξονα, που φέρει έκκεντρα τα οποία ανοίγουν τις βαλβίδες σε κατάλληλα χρονικά διαστήματα, και ο χρονισμός του οποίου είναι ένα βασικό σχεδιαστικό χαρακτηριστικό των κινητήρων.

Σχήμα 22: Έκκεντρα εκκεντροφόρου άξονα, με τη μία βαλβίδα να έχει ήδη ανοίξει [22]

Συνεπώς, το αποτέλεσμα είναι η παραγωγή ροπής σε έναν άξονα (στροφαλοφόρο) με αιτία μια δεδομένη ποσότητα μίγματος αέρα/καυσίμου που εισάγεται στο θάλαμο. Δεδομένου ότι οι αναλογίες του μίγματος αυτού είναι συγκεκριμένες για κάθε κινητήρα (μέσα σε κάποια πλαίσια), αυτό που κυρίως ελέγχουμε είναι η ποσότητα του ίδιου του μίγματος, η οποία θα έχει κι ως αποτέλεσμα περισσότερη ροπή στην έξοδο. Όπως θα δούμε, η σχέση αυτή δεν είναι μονοσήμαντη, καθώς σημαντικό ρόλο στην έξοδο έχει και το σε πόσες στροφές ανά λεπτό περιστρέφεται ήδη ο κινητήρας. Αυτό θα μπορούσαμε να το παρομοιάσουμε με το γεγονός πως ένας ποδηλάτης δεν μπορεί να δίνει την ίδια ποσότητα ροπής μέσω των ποδιών του σε όλες τις ταχύτητες: σε μεγάλες ταχύτητες μειώνεται η δύναμη που ασκεί στα πετάλια, αλλά αυξάνεται η ταχύτητα με την οποία το κάνει.

3.1.2 Καμπύλη ροπής-στροφών

Η σχέση της ροπής εξόδου με την ταχύτητα περιστροφής του κινητήρα για δεδομένη ποσότητα μίγματος εισόδου δεν είναι γραμμική. Επίσης είναι ένα αρκετά περίπλοκο σύστημα για να μοντελοποιηθεί, με πάρα πολλές μηχανικές και θερμικές διεργασίες, αν και έχουν γίνει αρκετές μελέτες επί του θέματος [23] [24]. Ως αποτέλεσμα έχουμε την πειραματική μέτρηση αυτών των τιμών μέσω διατάξεων που ονομάζονται δυναμόμετρα [25]. Δυο μεγάλες κατηγορίες δυναμομέτρων αυτοκινήτων είναι τα δυναμόμετρα κινητήρα και δυναμόμετρα πλαισίου (engine/chassis dynamometer). Στο δεύτερο μετράται η ροπή στους τροχούς μέσω κυλίνδρων στους οποίους στέκεται το όχημα. Αυτή λόγω μηχανικών απωλειών και απωλειών τριβής δεν είναι ίδια με τη ροπή που παράγει ο στρόφαλος, και την οποία μετράνε τα δυναμόμετρα κινητήρα,

κι είναι περίπου στο 80-90% αυτής [26].

Η μέτρηση που παράγουν τα δυναμόμετρα είναι τυπικά μια καμπύλη ροπής-στροφών, με τον άξονα x να αντιπροσωπεύει τις στροφές, και τον y τη ροπή. Επίσης, εξίσου συνήθης είναι κι η καμπύλη ισχύος-στροφών, ωστόσο αυτή δεν αποτελεί το ευθύ αποτέλεσμα της μέτρησης αλλά είναι παράγωγη.

Αρχικά γνωρίζουμε πως η ισχύς είναι ο ρυθμός μεταβολής του έργου:

$$P = \frac{dW}{dt} \tag{2}$$

Επίσης το έργο στην περιστροφική κίνηση, δεδομένης μιας ροπής $\vec{\tau}$ που δρα στο εύρος μιας γωνίας $\mathrm{d}\vec{\theta}$, ισούται με:

$$W = \int_{\theta_1}^{\theta_2} \vec{\tau}\, \mathrm{d}\vec{\theta} \tag{3}$$

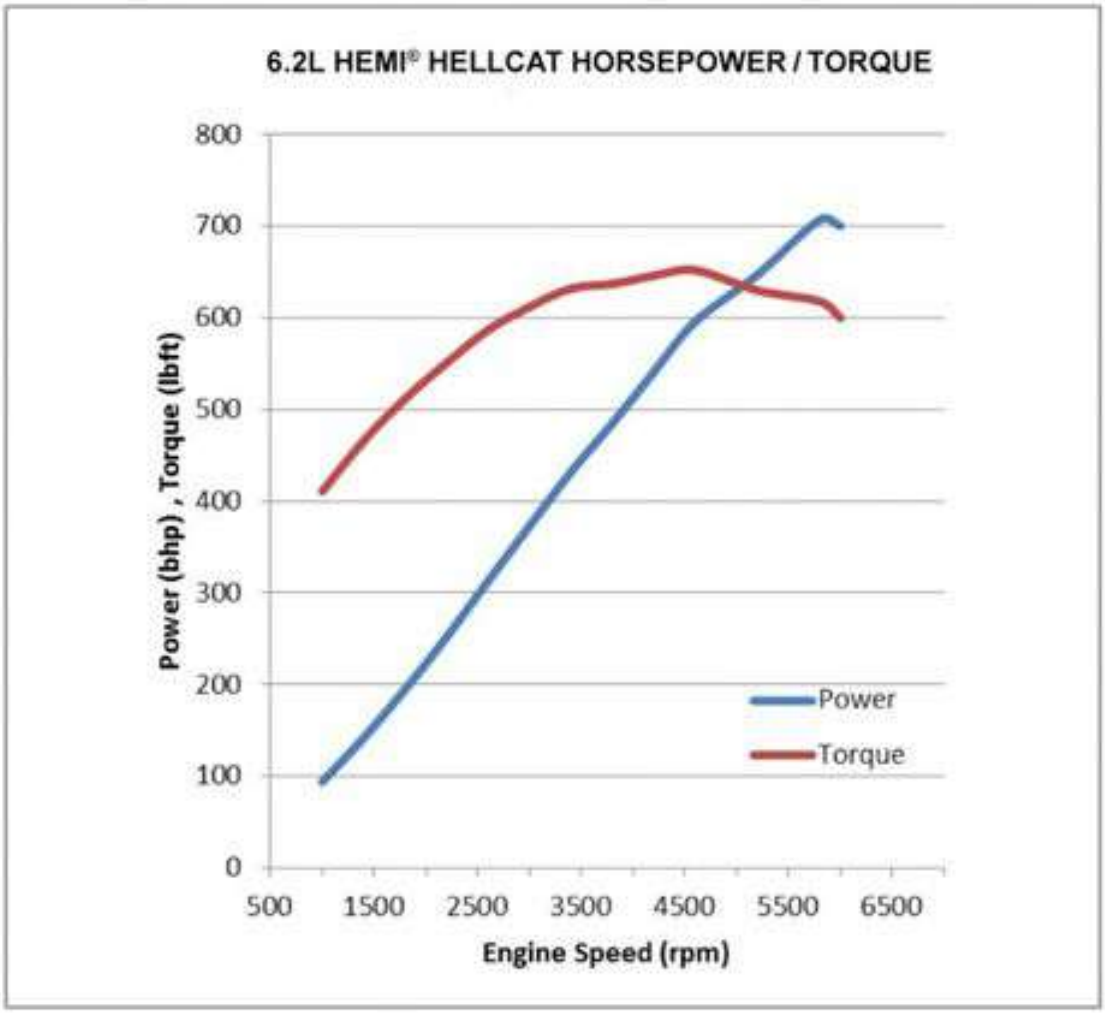
Όταν η ροπή και η περιστροφική μετατόπιση είναι στην ίδια κατεύθυνση, το εσωτερικό γινόμενο γίνεται βαθμωτό γινόμενο, αφού $cos(0) = 1$, κι άρα το έργο γίνεται:

$$W = \int_{\theta_1}^{\theta_2} \tau\, \mathrm{d}\theta \tag{4}$$

Συνεπώς από τα παραπάνω προκύπτει πως

$$P = \tau \cdot \omega \tag{5}$$

όπου θα μπορούσαν η ροπή και η περιστροφική ταχύτητα ω να είναι και διανυσματικές ποσότητες.



Σχήμα 23: Καμπύλη ροπής και ισχύος για τον κινητήρα HEMI 6.2L V8[27]

Τα δεδομένα μιας καμπύλης σαν και της παραπάνω αξιοποιούνται ώστε να μπορούμε να δώσουμε στην προσομοίωση μια ρεαλιστική έξοδο ροπής προς τους τροχούς δεδομένης εισόδου καυσίμου και δεδομένων τρεχουσών στροφών ανά λεπτό του κινητήρα. Η υλοποίηση στον κώδικα ακολουθεί παρακάτω. Έχουμε δεδομένη μια κλάση Engine η οποία περιέχει ως member έναν πίνακα από floats με όνομα torqueCurve. Στον constructor της κλάσης φτιάχνουμε την καμπύλη ροπής, παίρνοντας δεδομένα κάποια αρχικά γνωστά δείγματα, και μετά με γραμμική

παρεμβολή παράγουμε και τα ενδιάμεσα Α.18.

Η συνάρτηση torqueCurveSamples περιέχει κάποια δείγματα που εισήχθησαν χειροκίνητα, με αναφορά μια πραγματική καμπύλη αυτοκινήτου του εμπορίου Α.19.

Με δεδομένα αυτά, και με δεδομένη μια τρέχουσα περιστροφική ταχύτητα του κινητήρα, μπορούμε από τον πίνακα να υπολογίσουμε την τρέχουσα ροπή Α.20.

Η ροπή αυτή καταλήγει μεν στο άκρο του στροφαλοφόρου, αλλά απέχει αρκετά ακόμα από το να καταλήξει στις ρόδες, αφού στο ενδιάμεσο υπάρχει το σύστημα δίσκων και το κιβώτιο ταχυτήτων.

3.2 Μετάδοση

3.2.1 Σύστημα δίσκων

Στο άκρο του στροφαλοφόρου άξονα, υπάρχει μια διάταξη που λέγεται σφόνδυλος (flywheel). Το flywheel είναι ένας δίσκος μεγάλης μάζας, και συγκεκριμένα ροπής αδράνειας, και ο σκοπός του γενικά είναι η αποθήκευση περιστροφικής κινητικής ενέργειας. Στη συγκεκριμένη περίπτωση ο σκοπός του είναι εκτός των άλλων [28] η εξομάλυνση των δονήσεων του κινητήρα, και κυρίως μεταξύ των χρόνων του. Να θυμηθούμε πως η ροπή του κινητήρα παράγεται κατά το χρόνο της ανάφλεξης, συνεπώς μέχρι τον επόμενο κύκλο δεν υπάρχει κάποια ροπή στην έξοδο. Αυτά τα κενά μπορούν να προκαλέσουν δυσάρεστη εμπειρία στον οδηγό και στους επιβάτες καθώς και φθορά στα μηχανικά μέρη. Η ροπή αδράνειας του δίσκου είναι σημαντική, καθώς ορίζει την προαναφερθείσα εξομάλυνση αλλά και τη συμπεριφορά του κατά τη σύμπλεξη με το υπόλοιπο σύστημα της μετάδοσης.

Έπειτα, έχουμε το σύστημα των δίσκων[29]. Είναι μια διάταξη που μπορεί να γίνει αρκετά περίπλοκη, ωστόσο η βασική της λογική αφορά δυο δίσκους οι οποίοι βρίσκονται κολλητά ο ένας στον άλλον, με τον έναν να είναι συνδεδεμένος στον κινητήρα και τον άλλον στο υπόλοιπο όχημα (και τελικά στους τροχούς). Η μετάδοση γίνεται μέσω τριβής μεταξύ τους, και το μέγεθος της σύμπλεξης καθορίζεται από τον οδηγό μέσω του πεταλιού του συμπλέκτη. Ορίζεται μια κλάση

Gearbox που αφορά στο σύστημα μετάδοσης, και μια μεταβλητή της είναι η clutch A.21, που είναι ένας float με εύρος το [0,1] και συμβολίζει την ποσότητα της σύμπλεξης.

Με τη μεταβλητή αυτή να έχει την τιμή 1 θεωρούμε πως δεν υπάρχει καμία σύμπλεξη και δεν περνάει ροπή προς τους τροχούς, ενώ με τιμή 0 θεωρούμε πλήρη σύμπλεξη, δηλαδή σαν να υπάρχει ένας συμπαγής άξονας.

Οι φυσικές παράμετροι του δίσκου της μεριάς του κινητήρα ορίζονται στη συνάρτηση SetupPowertrain A.22.

Ο δίσκος συμβολίζεται με ένα στερεό σώμα, δηλαδή RigidBody σε όρους Unity. Αυτό το rigid body έχει μια μάζα και μια ροπή αδράνειας που τίθενται σε κάποιες τυπικές τιμές. Επίσης ορίζεται ο τανυστής ροπής αδράνειας (moment of inertia tensor) που είναι ένας 3x3 πίνακας που περιλαμβάνει τις ροπές αδράνειας γύρω από τους τρεις άξονες [30]. Ο πίνακας αυτός είναι διαγώνιος, και εφόσον μας ενδιαφέρει μόνο η περιστροφή γύρω από τον ένα άξονα, ορίζουμε μόνο την τιμή του Y. Επίσης για τον ίδιο λόγο περιορίζουμε τους βαθμούς ελευθερίας μέσω των λεγόμενων *constraints*, "παγώνοντας" την μετατόπιση σε όλους τους άξονες καθώς και τις περιστροφές γύρω από τους άξονες X και Z. Οι υπόλοιπες τρεις παράμετροι αφορούν τη Unity: το isKinematic γίνεται false ώστε να μπορούν να ασκηθούν ροπές στο στερεό σώμα και να έχει δυναμική συμπεριφορά, το useGravity = false ώστε το σώμα να μην επηρεάζεται από τη βαρύτητα, και το angularDrag είναι μια πειραματική τιμή που προσδίδει απώλειες στον δίσκο ώστε να σταματάει τελικά την περιστροφή του μετά από κάποιο χρόνο ελλείψει εξωτερικής δύναμης.

3.2.2 Κιβώτιο ταχυτήτων και σχέσεις μετάδοσης

Ένας κινητήρας εσωτερικής καύσης έχει ένα συγκεκριμένο εύρος λειτουργίας. Συνεπώς είναι πολύ δύσκολο σε αυτό το εύρος λειτουργίας να μπορούμε να εξυπηρετήσουμε και την ανάγκη για υψηλή ροπή (εκκίνηση οχήματος, ανηφόρες) και την ανάγκη για υψηλή ταχύτητα (ταξίδι σε αυτοκινητόδρομο). Γι' αυτό το λόγο υπάρχει το κιβώτιο ταχυτήτων, μια περίπλοκη διάταξη γραναζιών, αξόνων κι άλλων μηχανικών στοιχείων, που σκοπό έχει να προκαλέσει μείωση ή αύξηση στην ποσότητα ροπής και στροφών [31].

Ο λόγος της ροπής εξόδου προς τη ροπή εισόδου, κι αντίστοιχα των στροφών εξόδου και στροφών εισόδου λέγεται gear ratio. Συνήθως ένα συμβατικό κιβώτιο θα έχει 5 διαθέσιμα gear ratios για την ορθή κατεύθυνση, κι άλλο ένα που θα περιορίζεται στην όπισθεν. Η επιλογή της σχέσης γίνεται από τον οδηγό με τον αντίστοιχο μοχλό. Οι μικρές σχέσεις παρέχουν μεγαλύτερη ροπή και λιγότερες στροφές, ενώ οι μεγάλες το αντίθετο. Ένα επιπλέον γρανάζι υπάρχει συνήθως μετά τα κύρια 5 το οποίο λέγεται final drive, η σχέση του οποίου δεν είναι επιλέξιμη από τον οδηγό. Ορίζονται οι αντίστοιχες μεταβλητές στην κλάση EngineA.23.

Αντίστοιχα οι φυσικές παράμετροι της μεριάς του κιβωτίου ορίζονται κι αυτές στη συνάρτηση SetupPowertrainA.24.

Εφόσον η ροπή περάσει από το κιβώτιο μετασχηματισμένη κατά το λόγο μετασχηματισμού, έχει ένα τελικό στάδιο πριν φτάσει στους τροχούς.

3.2.3 Διαφορικό

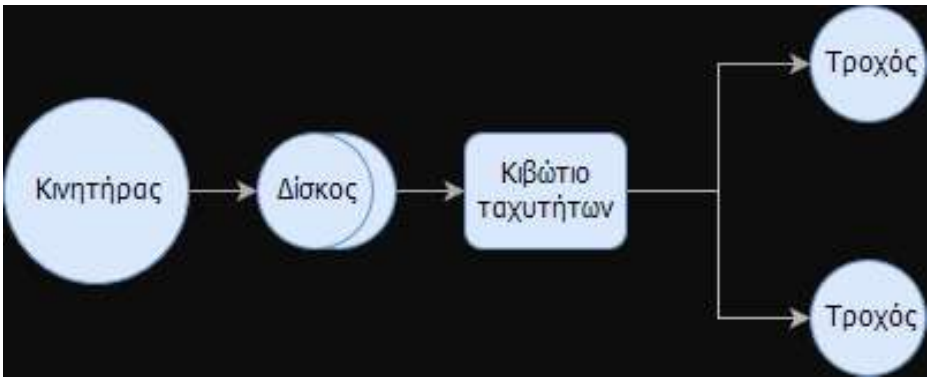
Η ροπή, ερχόμενη από το κιβώτιο ταχυτήτων μέσω του άξονα της μετάδοσης (driveshaft), θα μπορούσε να συνδεθεί απευθείας και με τους δυο τροχούς. Ωστόσο, κάτι τέτοιο θα ήταν προβληματικό στις περιπτώσεις που το όχημα δεν πηγαίνει ευθεία, καθώς κατά τη στροφή οι εσωτερικοί και εξωτερικοί τροχοί του οχήματος θα εξαναγκάζονταν σε ίδια περιστροφική ταχύτητα. Κάτι τέτοιο θα προκαλούσε ολίσθηση σε κάποιον από τους δυο, αφού προφανώς ο εξωτερικός τροχός θα πρέπει να εκτελέσει κίνηση με μεγαλύτερη περιστροφική ταχύτητα από τον εσωτερικό για να διανύσει την ίδια απόσταση στον ίδιο χρόνο. Η ολίσθηση αυτή θα προκαλούσε αστάθεια στο όχημα, ειδικά σε κατάσταση υψηλών ταχυτήτων ή/και απότομων αλλαγών πορείας. Γι' αυτό το λόγο προηγείται μια διάταξη που ονομάζεται διαφορικό [32].

Το διαφορικό φροντίζει, αναλόγως το είδος του, την κατανομή ροπής στον εσωτερικό και εξωτερικό τροχό του άξονα στον οποίο βρίσκεται. Υπάρχουν διάφορα είδη διαφορικών, αναλόγως την σύμπλεξη που επιβάλλουν μεταξύ των τροχών [33]. Στη συγκεκριμένα εργασία υλοποιήθηκε το ανοικτό διαφορικό (open differential), δηλαδή η ροπή που υπολογίζεται στην έξοδο του κιβωτίου, κατανέμεται ίσα και στους δυο τροχούς A.25.

Η παραπάνω τιμή grbx.wheelTorque που είναι και η τελική ροπή που κατανέμεται στους τροχούς, υπολογίζεται κατά την διάρκεια εκτέλεσης του προγράμματος, στη συνάρτηση UpdatePowertrain η οποία εκτελείται σε κάθε βρόχο της προσομοίωσης μέσα στη συνάρτηση Update της Unity. Στα πλαίσια αυτής της παραγράφου αναφέρουμε τη μεταβλητή hndl.throttle που είναι η ποσότητα του πεταλιού του γκαζιού που εφαρμόζει ο οδηγός, ωστόσο αναλυτικά τους χειρισμούς του οδηγού θα τους δούμε στο επόμενο κεφάλαιο.

Αρχικά, υπολογίζουμε την τρέχουσα ροπή του κινητήρα, μέσω της καμπύλης ροπής-στροφών A.26. Έπειτα υπολογίζουμε την ροπή στους τροχούς, λαμβάνοντας υπόψην όσα αναφέραμε περί κιβωτίου, σχέσεων και σύμπλεξης, ξεχωρίζοντας παράλληλα την περίπτωση όπου η σχέση που έχει εφαρμοστεί είναι η λεγόμενη "νεκρά". Στη νεκρά, θεωρούμε πως δεν υπάρχει σύμπλεξη, συνεπώς η ροπή εφαρμόζεται όλη πάνω στο σφόνδυλο του κινητήρα και στους τροχούς θα είναι 0A.27.

Συνοψίζοντας το σύστημα της μετάδοσης, το παρακάτω διάγραμμα δείχνει την πορεία της ροπής από τον κινητήρα προς τους τροχούς και τη σύνδεση των επι μέρους στοιχείων:



Σχήμα 24: Το σύστημα της μετάδοσης

3.3 Δυναμική ελαστικών και ανάρτησης

Η μόνη επαφή ενός αυτοκινήτου με το οδόστρωμα, είναι μέσω των ελαστικών. Συνεπώς η κατανόηση της δυναμικής συμπεριφοράς του ελαστικού κατά την κύλιση, καθώς και των μεταβατικών καταστάσεων κατά την αλλαγή πορείας όπου αυτό υπόκειται σε δυνάμεις στρέβλωσης, είναι καθοριστική για μία ρεαλιστική προσομοίωση.

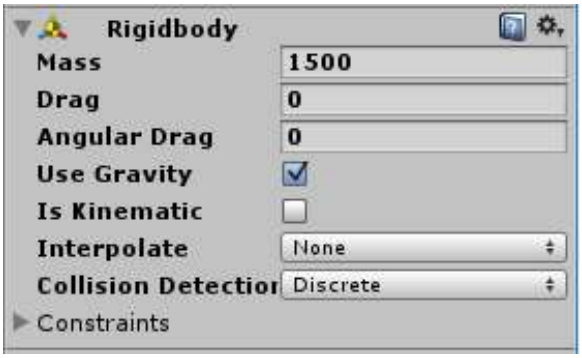
Αντίστοιχα, η στήριξη του αμαξώματος γίνεται σε μεγάλο βαθμό από τις αναρτήσεις: διατάξεις ελατηρίου-αποσβεστήρα που σχεδιάζονται λαμβάνοντας υπ' όψη θέματα ασφαλείας, άνεσης και επιδόσεων.

Και τα δύο πρόκεινται για περίπλοκα δυναμικά συστήματα, με χρόνια ερευνών να έχουν αφιερωθεί στην δημιουργία μοντέλων προσομοίωσης, είτε με στόχο την απόλυτη ακρίβεια είτε την αρκετά ρεαλιστική προσέγγιση. Η Unity, μέσω της ενσωματωμένης μηχανής φυσικής, προσφέρει προσομοίωση του συστήματος ελαστικού-ρόδας-ανάρτησης, με το component που ονομάζει WheelCollider. Αυτό που απαιτεί από εμάς είναι η ρύθμιση των κατάλληλων αριθμητικών παραμέτρων[34], καθώς και η σωστή αντιστοίχιση των αξόνων και των μεγεθών με το μοντέλο του αυτοκινήτου που χρησιμοποιείται. Συγκεκριμένα, χρησιμοποιείται αυτοκίνητο τύπου FWD (Front-Wheel Drive), συνεπώς έχουμε δυο άξονες από δυο ρόδες ο καθένας, με την κίνηση να είναι στον μπροστά άξονα.

Σχήμα 25: Το αυτοκίνητο που χρησιμοποιείται στην εργασία

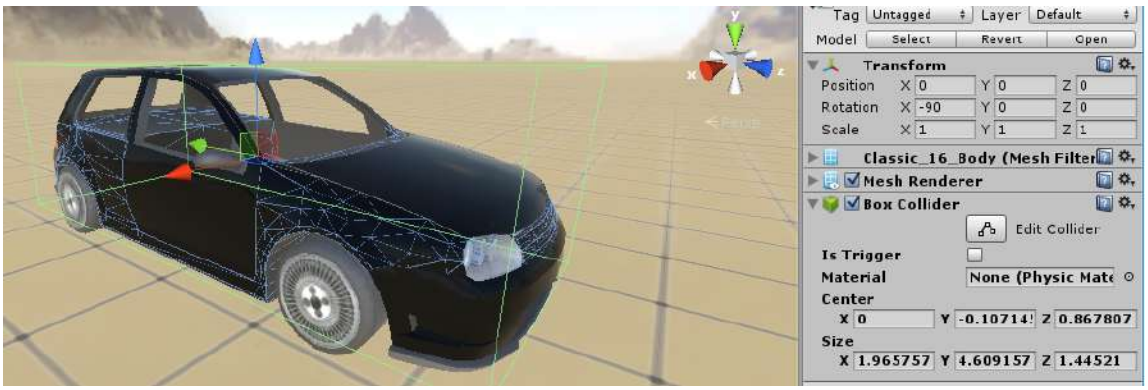
Το μοντέλο αυτό, πέρα από το κύριο αμάξωμα, έχει και πολλά υπο-μοντέλα που είναι αναγκαστικά διαφορετικές οντότητες στην ιεραρχία της Unity, ώστε να έχουν διαφορετικό

μετασχηματισμό (για παράδειγμα, για να μπορούν οι ρόδες να περιστρέφονται και να στρίβουν). Σε αυτά τα υπο-αντικείμενα που αναπαριστούν τις ρόδες, θα χρειαστεί να βάλουμε τα WheelCollider components ώστε να αποκτήσουν φυσική. Ωστόσο, πριν από αυτό πρέπει να θέσουμε ένα στερεό σώμα στο κύριο αντικείμενο του αυτοκινήτου. Το επιλέγουμε, και εισάγουμε ένα RigidBody component, με τα κατάλληλα νούμερα:



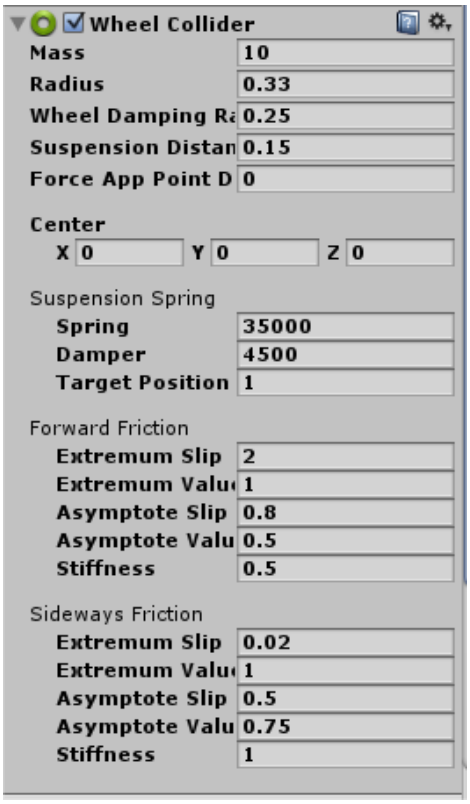
Σχήμα 26: Το κύριο component που δίνει ιδιότητες στερεού σώματος στο αυτοκίνητο

Επίσης, επιλέγουμε το υπό-αντικείμενο που αναπαριστά το κύριο αμάξωμα, και του δίνουμε ένα BoxCollider ώστε να μπορεί να γίνει μια βασική ανίχνευση σύγκρουσης με το περιβάλλον. Προφανώς το παραλληλεπίπεδο δεν καλύπτει με ακρίβεια το αμάξωμα, αλλά είναι μια πολύ καλή προσέγγιση:



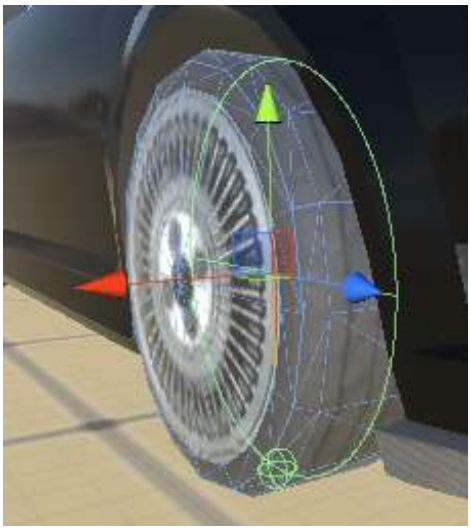
Σχήμα 27: Το BoxCollider του κυρίως αμαξώματος (πράσινο πλαίσιο)

Τώρα είμαστε έτοιμοι να ρυθμίσουμε τις τέσσερις ρόδες, επιλέγοντας το αντίστοιχο αντικείμενο και βάζοντας ένα WheelCollider:



Σχήμα 28: Το WheelCollider της κάθε ρόδας

Να σημειωθεί πως αυτό το component προσφέρει τρία πράγματα: φυσική ελαστικού, φυσική ανάρτησης αλλά και φυσική ανίχνευσης συγκρούσεων. Το τρίτο θα μπορούσε να πει κανείς πως εννοείται, καθώς για να μπορούν να λειτουργήσουν τα άλλα δυο πρέπει να ανιχνεύεται το πότε η ρόδα έρχεται σε επαφή με κάποια επιφάνεια. Ωστόσο δε χρειάζεται να εισάγουμε κάποιον collider χειροκίνητα, καθώς περιλαμβάνεται. Αυτό που πρέπει να κάνουμε είναι να ορίσουμε το μέγεθος του collider, ώστε να ταιριάζει με τις ρόδες του μοντέλου:

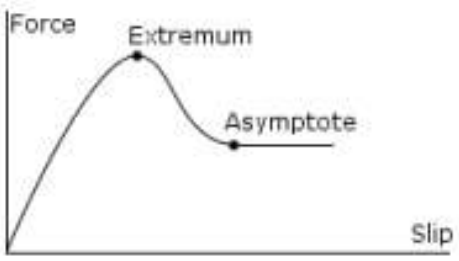


Σχήμα 29: Ο collider της ρόδας μετά από ρύθμιση

Οι παράμετροι στο πάνω μέρος του WheelCollider (Mass, Radius) αφορούν στην ρόδα καθαυτή.

Οι ρυθμίσεις Spring. Damper και TargetPosition αφορούν στην ανάρτηση, και επιλέχθηκαν μετά από δοκιμές ώστε να υπάρχει προσεγγιστικά ρεαλιστική συμπεριφορά. Είναι η δύναμη του ελατηρίου, η απόσβεση του αποσβεστήρα και η θέση ηρεμίας του ελατηρίου (στην οποία προσπαθεί να το φέρει η δύναμή του), αντίστοιχα.

Οι υπόλοιπες παράμετροι, που είναι και αρκετά πιο δύσκολες στην ρύθμιση, αφορούν στη φυσική του ελαστικού. Το μοντέλο που χρησιμοποιεί η Unity κι αντιστοιχίζει σε αυτά τα νούμερα είναι αυτό που φαίνεται παρακάτω:



Σχήμα 30: Το μοντέλο φυσικής ελαστικού της Unity

Είναι μια καμπύλη της δυνάμεως που ασκείται στο ελαστικό (άρα στη ρόδα, κι άρα στο όχημα σε εκείνο το σημείο) συναρτήσει του μεγέθους της ολίσθησης του ελαστικού σε σχέση με το οδόστρωμα. Το σκεπτικό είναι ότι για μικρές τιμές ολίσθησης το ελαστικό μπορεί και παραμορφώνεται αρκετά ώστε να έχει σταθερή επαφή με το οδόστρωμα και έτσι να προσφέρει αρκετή πρόσφυση. Μετά από ένα μέγιστο, η δύναμη που παράγεται είναι όλο και μικρότερη για μεγάλες τιμές ολίσθησης. Αυτό παρατηρείται στην πραγματικότητα, από το γεγονός πως αν στρίψουμε παραπάνω από όσο αντέχει το ελαστικό, το χειρότερο που μπορούμε να κάνουμε είναι να στρίψουμε ακόμα περισσότερο. Αντίθετα, αν στρίψουμε λιγότερο (αντίθετα από τη στροφή) τότε θα φτάσουμε πάλι στο μέγιστο της παραπάνω καμπύλης κι άρα σε κατάσταση μέγιστης πρόσφυσης.

Το μόνο που μένει είναι η εφαρμογή στις ρόδες της ροπής που έχει υπολογιστεί, καθώς και του στριψίματος, που θα δούμε στο επόμενο κεφάλαιο. Ορίζουμε μια κλάση AxleInfo που θα έχει πιο συγκεντρωτικά τις οντότητες που αποτελούν έναν άξονα A.28.

Το αυτοκίνητο θα περιλαμβάνει δυο τέτοια objects. Στη συνάρτηση Update που τρέχει στον κύριο βρόχο, θέτουμε τις ροπές και τις γωνίες στριψίματος στους αντίστοιχους άξονες A.29.

Το brakeTorque είναι η ροπή πέδησης, κάτι που επίσης μας παρέχει η Unity. Είναι διαφορετική από τη ροπή επιτάχυνσης, με την έννοια πως δεν είναι απλά αντίθετης κατεύθυνσης αλλά ακινητοποιεί τον WheelCollider στον οποίο εφαρμόζεται. Το WheelTorque είναι η ροπή

που υπολογίζεται πως φτάνει στις ρόδες, όπως είδαμε προηγουμένως.

Τέλος, το μόνο που μένει είναι το οπτικό κομμάτι: πρέπει με την πληροφορία της φυσικής που υπολογίζει η Unity να περιστρέψουμε τις ρόδες αναλόγως. Αυτό γίνεται πολύ απλά, παίρνοντας τον μετασχηματισμό από τον WheelCollider και θέτοντας το μετασχηματισμό περιστροφής του στον μετασχηματισμό περιστροφής της ρόδας A.30.

3.4 Αεροδυναμική αντίσταση

Ένας σημαντικός παράγοντας που επηρεάζει τις επιδόσεις και τη συμπεριφορά ενός οχήματος, είναι το γεγονός πως αυτό κινείται μέσα σε ένα ρευστό, που στην προκείμενη περίπτωση είναι ο αέρας. Η κίνηση μέσα στο ρευστό έχει ως αποτέλεσμα διάφορες δυνάμεις που ασκούνται με διάφορες κατευθύνσεις και σε διάφορα σημεία του οχήματος. Δεν χρειαζόμαστε κάποια ιδιαίτερη ακρίβεια στην προσομοίωση αυτών των δυνάμεων στα πλαίσια αυτής της εργασίας, αρκεί να έχουμε μια ποιοτική προσέγγιση της σημαντικότερης εξ' αυτών, η οποία ονομάζεται οπισθέλκουσα (drag).

Η οπισθέλκουσα είναι μια δύναμη που ασκείται σε ένα σώμα όταν αυτό κινείται σε ένα ρευστό, με κατεύθυνση αντίθετη της κίνησής του, όπως μαρτυρά και το όνομά της. Η τιμή της δίνεται από την εξίσωση οπισθέλκουσας (drag equation) [35] ως εξής:

$$F_d = C_d \frac{1}{2} \rho V^2 A \tag{6}$$

Όπου C_d είναι ο drag coefficient, ρ η πυκνότητα του αέρα, V η ταχύτητα του σώματος και A η επιφάνεια του σώματος όπως φαίνεται από την προβολή από τη μεριά που πλησιάζει το ρευστό. Ο αριθμός C_d είναι αδιάστατος κι ευρίσκεται πειραματικά.

Εφόσον θέλουμε απλά μια ποιοτική προσέγγιση, το βασικό που πρέπει να πάρουμε από την παραπάνω εξίσωση είναι η τετραγωνική σχέση της δύναμης με την ταχύτητα του σώματος. Δε θα δώσουμε βάση στις τιμές των παραγόντων, καθώς εμπειρικά θα πρέπει ένα όχημα αυτής της ισχύος και αυτού του μεγέθους να εμφανίζει μεγάλη αύξηση της οπισθέλκουσας περίπου στα

80km/h, και θα πρέπει να μην μπορεί να πετύχει ταχύτητα μεγαλύτερη των περίπου 180km/h. Με αυτά ως αναφορά, ορίζουμε και υλοποιούμε μια συνάρτηση ApplyAero που τρέχει στην Update A.31.

Όπως φαίνεται, ασκούμε στο στερεό σώμα του οχήματος τη δύναμη στην αντίθετη κατεύθυνση από αυτήν της κίνησης, με την ταχύτητα να είναι στο τετράγωνο και με έναν παράγοντα που περιλαμβάνει όλους τους παράγοντες της εξίσωσης οπισθέλκουσας και που δίνει εμπειρικά ρεαλιστικό αποτέλεσμα.

3.5 Παραγωγή ήχου κινητήρα

Για τον ήχο του κινητήρα, ξεκινούμε από το γεγονός πως αν είχαμε έναν κύλινδρο, αυτός θα έμπαινε στην φάση της ανάφλεξης (κι άρα της στιγμής που ακούγεται ο θόρυβος) σε κάθε δεύτερη στροφή του στροφαλοφόρου άξονα. Αυτό διότι όπως έχουμε δει, στον τετράχρονο κινητήρα ο κύλινδρος κάνει δυο ανόδους: μία για την ανάφλεξη, και μία ύστερα για την εκτόνωση.

Έχοντας λοιπόν την περιστροφική ταχύτητα του στροφαλοφόρου άξονα σε στροφές ανά λεπτό, μπορούμε να υπογίσουμε τη συχνότητα των αναφλέξεων ανά δευτερόλεπτο ως εξής:

$$F = \frac{RPM}{60}\frac{1}{2} \tag{7}$$

Ο συγκεκριμένος κινητήρας όμως έχει 4 κυλίνδρους, συνεπώς πολλαπλασιάζοντας επί 4 η συχνότητα γίνεται:

$$F = \frac{RPM}{30} \tag{8}$$

Στη Unity αυτό το υλοποιούμε δημιουργώντας μια πηγή ήχου στη θέση που είναι ο κινητήρας του αυτοκινήτου, και ρυθμίζοντας την συχνότητα του ήχου μέσω της μεταβλητής pitch. Η μεταβλητή αυτή ωστόσο δεν δείχνει απευθείας τη συχνότητα του ήχου, αλλά είναι ένας πολλαπλασιαστής της συχνότητας του αρχείου ήχου το οποίο έχει οριστεί στην πηγή. Έχοντας λοιπόν έναν ήχο του οποίου η συχνότητα φροντίζουμε μέσω του REAPER να είναι 1000Hz, η υλοποίηση σε κώδικα είναι απλή A.32.

Συνεπώς για τις τρέχουσες στροφές ανά λεπτό του στροφαλοφόρου άξονα, θα υπολογιστεί μια συχνότητα η οποία θα διαιρεθεί με την αναφορά των 1000Hz ώστε να θέσουμε στην πηγή ήχου την κατάλληλη αναλογία του pitch.

3.6 Κατανάλωση καυσίμου

Ένα πλεονέκτημα της μέχρι τώρα δουλειάς που έχει γίνει στη μοντελοποίηση του κινητήρα και του αυτοκινήτου γενικότερα, είναι πως μπορούμε αρκετά εύκολα να υπολογίσουμε την κατανάλωση του κινητήρα με μόνη μεταβαλλόμενη παράμετρο την ποσότητα του πεταλιού του γκαζιού που εφαρμόζει ο οδηγός. Αυτό διότι το γκάζι δεν υπολογίζεται τεχνητά, παρά προκύπτει από τις φυσικές απαιτήσεις που υπάρχουν ώστε το όχημα να εκτελέσει την πορεία που πρέπει.

Η κατανάλωση του καυσίμου είθισται να παρουσιάζεται σε λίτρα ανά 100 χιλιόμετρα. Αυτή δεν είναι άρα στιγμιαία τιμή, παρά μέσος όρος σε εύρος πολλών χιλιομέτρων, ώστε να μπορούμε να έχουμε καλύτερη εικόνα για την οικονομία του κινητήρα σε διάφορες συνθήκες οδήγησης χωρίς να λαμβάνουμε υπ' όψη ακραίες τιμές (σε εκκίνηση σε μια ανηφόρα θα υπάρχει τεράστια κατανάλωση, ενώ σε μια κατηφόρα μπορεί να είναι και 0). Η προσέγγιση που κάνουμε εδώ, είναι πως υπολογίζουμε ανά πάσα στιγμή έναν "στιγμιαίο" μέσο όρο, και παράλληλα υπολογίζουμε τον τρέχοντα μέσο όρο όλων αυτών των μέσων όρων ώστε να πάρουμε την τρέχουσα μέση κατανάλωση σε λ/100χμ.

Κατ' αρχάς, είναι προφανές πως η ποσότητα του καυσίμου που μπορεί να παρέχεται στον κινητήρα είναι περιορισμένη. Αυτό το εκφράζουμε ως μια μέγιστη τιμή λίτρων ανά 100χμ που θα μπορούσε να καταναλώσει ο κινητήρας στην χειρότερη περίπτωση A.33.

Τα μοντέρνα αυτοκίνητα έχουν μια αρκετά περίπλοκη διαδικασία μέσω της οποίας αποφασίζεται το πόση ποσότητα καυσίμου θα προσφερθεί στον κινητήρα, χωρίς να υπάρχει απόλυτα άμεση σχέση με την ποσότητα του γκαζιού που εφαρμόζει ο οδηγός. Ωστόσο εδώ θα θεωρήσουμε, κάνοντας μια σχετικά ρεαλιστική προσέγγιση, πως η ροή του καυσίμου είναι ακριβής αναλογία της ποσότητας του γκαζιού. Άρα μπορούμε να υπολογίσουμε την τρέχουσα κατανάλωση σε

λ/100χμ πολλαπλασιάζοντας την τιμή της μέγιστης κατανάλωσης σε λ/100χμ με την ποσότητα του γκαζιού που εφαρμόζεται αυτήν τη στιγμή (η οποία είναι στο εύρος $[0, 1]$) A.34

Έπειτα απλά προσθέτουμε αυτές τις τιμές σε έναν accumulator, και τις διαιρούμε με το πλήθος των μετρήσεων για να πάρουμε έναν μέσο όρο. Υπενθυμίζεται ότι η κάθε μέτρηση αφορά τον τρέχοντα στιγμιαίο μέσο όρο σε λ/100χμ, και τελικά υπολογίζουμε τον μέσο όρο αυτών των μέσων όρων για να πάρουμε την τελική τιμή της κατανάλωσης, την οποία και δείχνουμε στο χρήστη, όπως θα δούμε αργότερα στο περιβάλλον της προσομοίωσης.

4 Προσομοίωση Κυκλοφορίας

4.1 Μοντέλο οδηγού

Όπως αναφέραμε και στο κεφάλαιο της αναπαράστασης των δρόμων, είναι πολύ σημαντικό να υπάρχει μεγάλη συσχέτιση μεταξύ του τρόπου που αναπαρίσταται ο δρόμος μέσα στην προσομοίωση, και της ίδιας της προσομοίωσης. Αυτό διότι έννοιες όπως δρόμος, λωρίδα, όρια δρόμου, διαδρομή κλπ είναι κρίσιμες για τον υπολογισμό της συμπεριφοράς του οδηγού ακόμα και στην πραγματικότητα.

Η βασική λογική και δομή του μοντέλου του οδηγού βρίσκεται στο αρχείο agent.cs με κύρια κλάση την Driver A.35.

Αναλυτικά αυτές οι κλάσεις έχουν ως εξής:

- trajectory : Αφορά σε ό,τι έχει να κάνει με την τροχιά και τη διαδρομή που ακολουθεί το όχημα. Περιλαμβάνει δείκτες στα αριστερά και δεξιά άκρα της τρέχουσας λωρίδας, δείκτη στην ίδια την λωρίδα, δείκτη στην επόμενη λωρίδα, την λίστα με τα checkpoints της τρέχουσας διαδρομής, το τρέχον checkpoint κ.α., καθώς και συναρτήσεις που παράγουν τα checkpoints της τρέχουσας διαδρομής, συναρτήσεις μετάβασης σε επόμενη λωρίδα κ.α.

- handling : A.36Περιλαμβάνει παραμέτρους που αφορούν στον χειρισμό του οχήματος, όπως το ποσοστό πατήματος των πεταλιών γκαζιού, φρένου και συμπλέκτη, τη γωνία στριψίματος του τιμονιού, τις στροφές ανά λεπτό στις οποίες θα εκτελείται αλλαγή σχέσης στο κιβώτιο (προς τα πάνω και προς τα κάτω), μεταβλητές που επιτρέπουν ή όχι την αλλαγή σχέσεων ή την χρήση των πεταλιών (π.χ. αλλαγή σχέσης κατά τη διάρκεια άλλης αλλαγής δεν επιτρέπεται) κλπ.

- behaviour : A.37Αυτή η κλάση έχει να κάνει με την συμπεριφορά του οδηγού. Βασικές παράμετροι είναι η επιθετικότητα με την οποία στρίβει καθώς και η ταχύτητα απόκρισης σε καταστάσεις αλλαγής πορείας. Επίσης περιλαμβάνει μια μεταβλητή που ελέγχει το αν το αμάξι πρόκειται να σταματήσει ή όχι (πχ λόγω παραχώρησης προτεραιότητας ή λόγω εμποδίου στην πορεία του).

- vehicle : Εδώ ομαδοποιούνται οι παράμετροι κι οι μέθοδοι που έχουν να κάνουν με το όχημα καθ' αυτό. Θα δούμε αργότερα την κλάση αυτή σε δράση, ωστόσο να αναφέρουμε παραδείγματα μεταβλητών που περιλαμβάνει: τα αντικείμενα (GameObject) του κυρίως αμαξώματος, των τεσσάρων ροδών, η λίστα των αξόνων (ως μια λίστα από AxleInfo), μια μεταβλητή που ελέγχει αν το όχημα έφτασε στο τέλος της πορείας του για να καταστραφεί, το χρώμα του οχήματος, η πηγή ήχου, η μέγιστη γωνία στριψίματος των ροδών, η μέγιστη ροπή πέδησης κ.α.

- engine : Την κλάση Engine την είδαμε μερικώς στο κεφάλαιο της προσομοίωσης του κινητήρα. Όπως λέει και το όνομά της αφορά στον κινητήρα και ενδεικτικά περιλαμβάνει μεταβλητές για τη ροπή αδράνειας, τα όρια στροφών που μπορεί να λειτουργήσει ο κινητήρας, τη μέγιστη ροή καυσίμου, τη λίστα με τα σημεία της καμπύλης ροπής, τις στροφές κάτω από τις οποίες ο κινητήρας θα σβήσει (''ρελαντί'', stall) κ.α.

- gearbox : Τέλος, την κλάση Gearbox επίσης την είδαμε στο κεφάλαιο της προσομοίωσης του κιβωτίου ταχυτήτων, και είναι πολύ στενά συνδεδεμένη με την κλάση Engine. Περιέχει μεθόδους που θα δούμε στην πράξη στη συνέχεια, και μεταβλητές για τους λόγους των σχέσεων (gear ratios) καθώς και της τελικής σχέσης (final drive), το ποσοστό ενεργοποίησης του συμπλέκτη, τη ροπή αδράνειας του δίσκου, την τρέχουσα σχέση στο κιβώτιο, κ.α.

Συνεπώς, ο οδηγός είναι μια οντότητα που εννοιολογικά αλλά και προγραμματιστικά συμπεριλαμβάνει μια πορεία (ή τροχιά), μια ανθρώπινη συμπεριφορά, ένα σύνολο οδηγικών χαρακτηριστικών, ένα όχημα (ως φυσικό αντικείμενο), έναν κινητήρα, κι ένα κιβώτιο ταχυτήτων/σύστημα μετάδοσης. Με αυτά είμαστε έτοιμοι να δούμε το πως αρχικά δημιουργείται το όχημα, πως τοποθετείται στο σημείο δημιουργίας του και έπειτα πως πλοηγείται και συμπεριφέρεται στο σχεδιασμένο σύστημα δρόμων.

Την ενορχήστρωση της δημιουργίας των οχημάτων και του χειρισμού της προσομοίωσης την αναλαμβάνει το script state.cs το οποίο όμως θα δούμε αναλυτικά στην παράγραφο του περιβάλλοντος της προσομοίωσης. Οι επόμενες παράγραφοι ασχολούνται με το κάθε όχημα σαν ανεξάρτητη οντότητα, με περιστασιακές αναφορές στο state όπου αυτό είναι απαραίτητο, μιας και για συγκεκριμένες λειτουργίες τα οχήματα πρέπει να ελέγχουν την γενική κατάσταση του κόσμου.

4.2 Πλοήγηση στη λωρίδα

4.2.1 Αρχική τοποθέτηση οχήματος

Το κάθε όχημα χρησιμοποιεί ένα component τύπου Move που υλοποιείται από το script με το ίδιο όνομα. Όταν το όχημα δημιουργείται, τότε όπως είθισται στην Unity, εκτελείται η συνάρτηση Start όλων των components του. Αυτό βέβαια προϋποθέτει το script να κληρονομεί από την κλάση MonoBehaviour όπως έχουμε αναφέρει. Υλοποιούμε τότε την Start του οχήματος A.38.

- Η συνάρτηση InitializeParams δημιουργεί τα αντικείμενα από τις κλάσεις τους καθώς και αρχικοποιεί τις διάφορες παραμέτρους του οδηγού A.39. Επίσης εκτελεί τη συνάρτηση SetupPowertrain η οποία είναι μια διαδικαστική συνάρτηση που αρχικοποιεί τις παραμέτρους των κλάσεων του κινητήρα και του κιβωτίου A.40.

Επίσης αρχικοποιεί το speedLUT της κλάσης Handling που είναι ένα look-up table (δηλαδή ένας πίνακας αντιστοιχίας) μεταξύ της γωνίας προς την τροχιά και της μέγιστης επιτρεπτής ταχύτητας σε αυτήν τη γωνία A.41. Θα το δούμε σε χρήση αργότερα.

Τέλος, επιλέγει μια τυχαία λωρίδα από αυτές που είχαμε ορίσει στον editor ως λωρίδες εκκίνησης, και επιλέγει και την επόμενη λωρίδα, με επιτρεπτές να είναι αυτές που έχουν οριστεί ως συνδεδεμένες σε αυτήν που επιλέχθηκε ως αρχική. Όλες αυτές οι πληροφορίες υπενθυμίζεται πως περιέχονται στο .motors αρχείο που δημιουργήσαμε από τον editor μέσω του New Lane και New Connection A.42.

- Η SetColor αρχικοποιεί το όχημα σε ένα τυχαίο χρώμα. Εφόσον αποτελείται από πολλά υπο-αντικείμενα, πρέπει να βάλουμε το χρώμα σε όλα τα αντικείμενα ξεχωριστά A.43.

- Το state.dirs είναι μια λίστα από αντικείμενα της κλάσης Directions A.44. Την επεξήγηση των μελών θα τη δούμε σύντομα.

- Τέλος, η συνάρτηση Lineup είναι σημαντική καθώς ευθυγραμμίζει το αυτοκίνητο στην τροχιά που πρόκειται να ακολουθήσει, και ελέγχει το αν στο ίδιο σημείο έχει μόλις δημιουργηθεί άλλο αυτοκίνητο ώστε να ακυρώσει τη δημιουργία του νέου για αποφυγή συγκρούσεων A.45.

Η dirCompute προετοιμάζει τη μεταβλητή dirVec με την κατεύθυνση του οχήματος A.46.

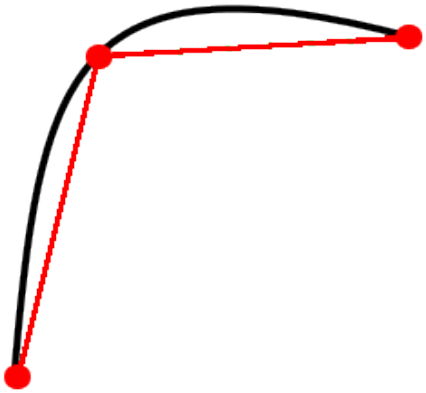
Το όχημα πλέον δημιουργήθηκε κι ευθυγραμμίστηκε και μπαίνει στη διαδικασία πλοήγησης μέσα στην τροχιά.

4.2.2 Υπολογισμός τροχιάς

Η αρχική τροχιά έχει ήδη υπολογιστεί, καθώς χρησιμοποιήθηκε ως παράμετρος στη συνάρτηση Lineup που είδαμε πριν (traj.currPath). Ο υπολογισμός γίνεται κατά την εκτέλεση της συνάρτησης GoToLaneA.47, που είναι μέθοδος της κλάσης Trajectory και εκτελέστηκε στο τέλος της InitializeParams.

Η Generate είναι αυτή που τελικά παράγει την τροχιά, χρησιμοποιώντας τις πληροφορίες του κόσμου μέσω του script state καθώς και τις βοηθητικές συναρτήσεις που αναλύσαμε στο κεφάλαιο των καμπυλών Bezier. Η τροχιά αποτελεί σταθμισμένο μέσο όρο των καμπυλών που αποτελούν το δεξιό και αριστερό άκρο της, με τυχαίες στάθμες ώστε να μην υπολογίζεται η ακριβώς ίδια τροχιά για όλα τα οχήματα της ίδιας λωρίδας A.48A.49.

Για τον αριθμό των βημάτων, χρειαζόμαστε το μήκος της καμπύλης. Κάνουμε προσέγγιση δυο τμημάτων, αθροίζοντας το μήκος των δυο ευθυγράμμων τμημάτων που ορίζονται από την αρχή της καμπύλης μέχρι το μέσο της, και από το μέσο ως το τέλος της A.50.



Σχήμα 31: Προσέγγιση δυο τμημάτων του μήκους καμπύλης Bezier.

Η παράμετρος smoothConnection δημιουργεί ομαλή σύνδεση δυο λωρίδων, καθώς παίρνει τον μέσο όρο του ήδη υπολογισμένου αρχικού σημείου P_0 της επόμενης λωρίδας και του τρέχοντος σημείου-στόχου. Αυτό γίνεται ώστε να μην υπάρχει απότομη αλλαγή πορείας κατά την αλλαγή λωρίδων, αφού είναι πολύ πιθανό δυο συνδεδεμένες λωρίδες να μην έχουν απόλυτη ευθυγράμμιση πάνω στο σημείο της σύνδεσης. Στην αρχική εκτέλεση περνάμε false στην παράμετρο, ωστόσο στις μετέπειτα δημιουργίες τροχιών (δηλαδή αφού το όχημα έχει δημιουργηθεί και αλλάζει λωρίδες) θα περνάμε true A.51. Δεν μένει παρά να δημιουργήσουμε τελικά την τροχιά A.52.

Όλα τα παραπάνω εκτελέστηκαν στην Start. Το όχημα λοιπόν μόλις έχει ξεκινήσει να κινείται στην αρχική τροχιά, ωστόσο κάποια στιγμή θα χρειαστεί να ακολουθήσει την επόμενη λωρίδα, η οποία θα έχει τη δικιά της τροχιά, συνεπώς θα πρέπει να επαναληφθεί η παραπάνω διαδικασία. Επίσης, μέσα στην ίδια τροχιά, έχουμε παράξει πολλά σημεία-στόχους (checkpoints) τα οποία ο οδηγός πρέπει να ακολουθήσει. Οι παραπάνω έλεγχοι για την αλλαγή τροχιάς/λωρίδας, αλλά και την αλλαγή τρέχοντος σημείου-στόχου μέσα στην ίδια λωρίδα, γίνονται στην FixedUpdate επαναληπτικά για όλο τον χρόνο ύπαρξης το οχήματος στον κόσμο A.53.

Αν κάποια από αυτές τις συνθήκες ικανοποιείται, ακυρώνουμε την συγκεκριμένη επανάληψη μαζί με όλες τις υπολοιπόμενες ενέργειες (ανανέωση της κατάστασης της φυσικής του οχήματος και της συμπεριφοράς του οδηγού). Ταυτόχρονα, κάνουμε την αλλαγή στην επόμενη λωρίδα ή στο επόμενο checkpoint ώστε η επόμενη επανάληψη να είναι ενημερωμένη με την τρέχουσα κατάσταση A.54.

Η CurrentTargetReached ελέγχει για το αν φτάσαμε στον τρέχοντα σημείο-στόχο. Θεωρητικά θα φτάναμε στον στόχο αν η θέση του οχήματος ήταν πανομοιότυπη με τη θέση του checkpoint. Αυτό ωστόσο είναι απίθανο να γίνει για δυο λόγους:

Πρώτον, το όχημα από άποψης κινηματικής και γεωμετρίας δεν είναι ικανό να διαγράψει όλες τις πιθανές τροχιές που μπορούν να σχεδιαστούν σε ένα επίπεδο. Για παράδειγμα, δεν είναι δυνατό ένα τετράτροχο όχημα να διαγράψει πορεία τεθλασμένης γραμμής. Αυτό σημαίνει πως κάποια σημεία, είναι αδύνατο να τα φτάσει, χωρίς να ακινητοποιηθεί και να κάνει όπισθεν.

Δεύτερον, ακόμα κι αν κινηματικά ήταν ικανό να διαγράψει οποιαδήποτε πορεία, εφόσον οι πράξεις γίνονται με αριθμούς κινητής υποδιαστολής (floating point) ο έλεγχος της ισότητας δυο αριθμών χάνει το νόημά του [36]. Υπάρχουν μέθοδοι για να προσπεραστεί αυτός ο περιορισμός των αριθμών κινητής υποδιαστολής, όπως η χρήση συναρτήσεων ισότητας που θεωρούν ισότητα την πολύ μικρή διαφορά δυο αριθμών, αλλά σε συνδυασμό με τον πρώτο λόγο παραπάνω, το καθιστά μη πρακτικό.

Συνεπώς θεωρούμε πως φτάσαμε στο checkpoint αν απλά βρισκόμαστε πολύ κοντά σε αυτό. Συγκεκριμένα ορίζουμε ως αρκετά μικρή απόσταση τα 10cm. Αν το όχημα απέχει περισσότερο, δεν έχει φτάσει ακόμα στον στόχο, αν όμως απέχει λιγότερο τότε αυξάνουμε τον δείκτη του σημείου της τροχιάς το οποίο θεωρούμε τρέχοντα στόχο.

Αντίστοιχα χειριζόμαστε και τον έλεγχο του τέλους της λωρίδας A.55.

Η συνθήκη που κρίνει τώρα το αν έχουμε φτάσει στο τέλος της λωρίδας, είναι το αν ο τρέχων στόχος είναι και το τελευταίο σημείο της τροχιάς, ελέγχοντας τον δείκτη του τρέχοντος στόχου με το μέγεθος της λίστας των σημείων της τροχιάς.

Στην περίπτωση που το σημείο δεν είναι το τελευταίο, απλά επιστρέφουμε false. Αν όμως είναι, υπάρχουν δυο περιπτώσεις:

Αν η λωρίδα στην οποία ήμαστε μέχρι τώρα είναι λωρίδα τερματισμού, τότε το όχημα έφτασε στο τέλος της διαδρομής του και καταστρέφεται θέτοντας τη μεταβλητή letDestroy του οχήματος σε true. Έπειτα, στην επόμενη επανάληψη, στην Update του state, θα καταστραφεί. Τον ακριβή τρόπο θα τον δούμε στην αντίστοιχη παράγραφο. Ο λόγος που δεν γίνεται απευθείας η καταστροφή του αντικειμένου μέσα από το script Move, είναι πως το state περιέχει διάφορα δεδομένα ανά όχημα, συνεπώς το αφήνουμε στη δικιά του ευχέρεια να κάνει το Destroy για να τακτοποιήσει και άλλες εκκρεμότητες (διαγραφή από άλλες λίστες κλπ).

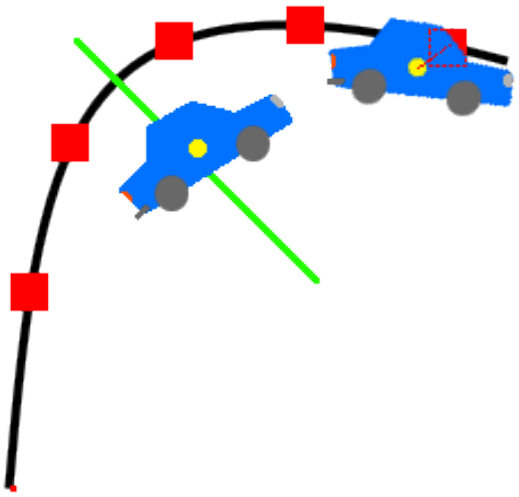
Αν η λωρίδα δεν είναι λωρίδα τερματισμού, τότε απλά επιλέγουμε την επόμενη λωρίδα. Παίρνουμε για αρχή μια αναφορά στη λίστα με τις συνδέσεις της τρέχουσας λωρίδας, κι επιστρέφουμε τη λωρίδα της λίστας που ως δείκτη έχει το τρέχον traj.nextLane. Την πρώτη φορά, αυτό είχε γίνει στην InitializeParams. Η συνάρτηση traj.NextLane είναι πρακτικά ίδια με την GoToLane, με τη διαφορά ότι εδώ έχουμε smoothConnection == true. Τέλος, ορίζουμε καινούριο δείκτη traj.nextLane που θα χρησιμοποιηθεί ομοίως στην επόμενη επανάληψη.

Επόμενη παρασκευαστική συνάρτηση που καλείται στην FixedUpdate του οδηγού, και ακριβώς πριν καλέσουμε τις συναρτήσεις χειρισμού του οχήματος, είναι η SetupCurrentTarget η οποία προετοιμάζει τον τρέχοντα στόχο. Εκτελεί δυο βασικές λειτουργίες A.56.

Αρχικά υπολογίζει τη γωνία στην οποία βρίσκεται ο στόχος. Το νούμερο αυτό το χρησιμοποιεί ο οδηγός για να στρίψει αναλόγως. Για να βρούμε τη γωνία, πρέπει πρώτα να βρούμε το διάνυσμα που ενώνει το όχημα με το σημείο στόχο, αφαιρώντας τη θέση του στόχου από τη θέση του οχήματος. Στη συνέχεια χρησιμοποιούμε τη συνάρτηση Vector2.Angle της Unity που δέχεται ως ορίσματα δυο διδιάστατα διανύσματα, κι επιστρέφει τη γωνία μεταξύ τους. Η γωνία αυτή ωστόσο δεν έχει πρόσημο [37], κι άρα δεν έχουμε πληροφορία για το αν ο στόχος βρίσκεται αριστερά ή δεξιά καθ' αυτή τη γωνία. Αυτό το βρίσκουμε με μια επιπλέον πράξη, που είναι το εξωτερικό γινόμενο μεταξύ δυο διανυσμάτων: του διανύσματος όχημα-στόχος που υπολογίσαμε πριν, και του διανύσματος κατεύθυνσης του οχήματος. Το πρόσημο του εξωτερικού γινομένου

μας δίνει τη μεριά από την οποία βρίσκεται ένα σημείο σε σχέση με το διάνυσμα αναφοράς, κι άρα αντιστρέφουμε το πρόσημο της γωνίας αναλόγως.

Έπειτα, λαμβάνει χώρα μια πολύ σημαντική πράξη, που είναι πολύ συναφής με το όριο των 10cm για την προσέγγιση σημείου που αναφέραμε προηγουμένως. Εδώ λοιπόν προκύπτει επίσης ένα πρόβλημα: αν για κάποιο λόγο το όχημα προσπεράσει ένα σημείο χωρίς να περάσει μέσα στην ακτίνα των 10cm, τότε οι προηγούμενοι υπολογισμοί θα το εξαναγκάσουν σε στροφή 180 μοιρών, αφού πλέον το σημείο είναι πίσω από το όχημα. Αυτό οδηγεί σε μια κατάσταση όπου το όχημα προσπαθεί επανειλλημένως να πλησιάσει το στόχο, με πιθανότερο σενάριο το να κάνει συνεχώς κύκλους γύρω από αυτό, χωρίς να μπορεί, λόγω κινηματικής, να πλησιάσει αρκετά κοντά. Κάτι τέτοιο είναι σχεδόν σίγουρο πως θα γίνει όταν υπάρχει μια στροφή, οπότε και το όχημα για λίγα μέτρα ίσως είναι λίγο εκτός πορείας μέχρι να συγκλίνει στην τροχιά. Η λύση σε αυτό, γίνεται αγνοώντας τον τρέχοντα στόχο αν βρισκόμαστε πλησιέστερα στον επόμενο. Η πράξη είναι ακριβώς αυτή, δηλαδή έλεγχος της απόστασης με τον τρέχοντα και τον επόμενο στόχο, και αύξηση του δείκτη του τρέχοντος στόχου αν η πρώτη είναι μεγαλύτερη. Σχηματικά, μπορούμε να φανταστούμε τη μεσοκάθετο των δυο στόχων, με τη συνθήκη να ισχύει αν περάσουμε την ευθεία:



Σχήμα 32: Αγνοούμε τον τρέχοντα στόχο αν είμαστε πλησιέστερα στον επόμενο

Το να συνεχίσει η επανάληψη πέρα από τις προαναφερθείσες συναρτήσεις, σημαίνει πως το όχημα βρίσκεται σε κανονική κατάσταση εν μέσω μιας λωρίδας, έχει επιλέξει στόχο και γνωρίζει την γωνία προς αυτόν. Συνεπώς ο οδηγός αναλαμβάνει πλέον το χειρισμό του οχήματος.

4.2.3 Χειρισμός τιμονιού

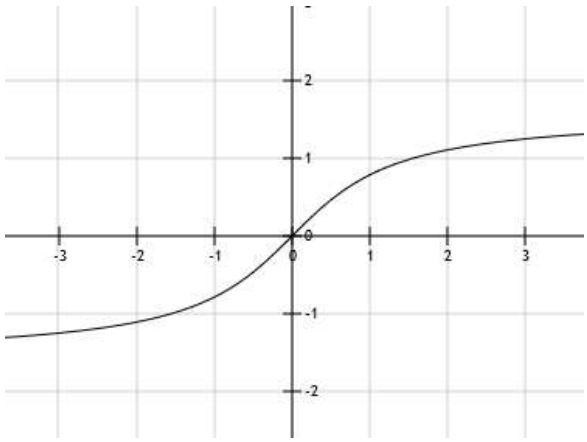
Όλος ο χειρισμός του τιμονιού γίνεται στη συνάρτηση SteerToTarget. Θεωρητικά θα αρκούσε απλά να στρίβαμε το τιμόνι στην γωνία που βρίσκεται ο στόχος. Ο λόγος που αυτό δεν αρκεί, είναι το γεγονός πως στην γενική περίπτωση που το όχημα δεν είναι ακριβώς πάνω στην τροχιά και απόλυτα ευθυγραμμισμένο, τον στόχο θα τον πλησιάσει με ευθεία κίνηση και υπο γωνία. Εφόσον οι στόχοι απέχουν πολύ λίγο (Trajectory.meterRes = 0.3f) αυτό είναι σίγουρο και δοκιμασμένο πως θα προκαλέσει στο όχημα τροχιά "ζιγκ-ζαγκ", όπου θα πλησιάζει κάθε φορά υπό γωνία την τροχιά, μόλις φτάνει στο στόχο θα ξεφεύγει από αυτήν λόγω κεκτημένης ταχύτητας, θα επιστρέφει για να ξαναπλησιάσει κ.ο.κ.

Μια πρώτη λύση σε αυτό το πρόβλημα είναι να μη δίνουμε εντολή στριψίματος στην ακριβή γωνία που βρίσκεται ο στόχος, αλλά να λαμβάνουμε υπ' όψην και την κατεύθυνση της τροχιάς. Η πληροφορία της κατεύθυνσης της τροχιάς μας βοηθά στο να δίνουμε βάρος και στο προς τα που θέλουμε να κατευθύνεται το όχημα, πέρα από το να μηδενίσει την απόσταση από τον τρέχοντα στόχο. Στην ακραία περίπτωση που χρησιμοποιούσαμε μόνο αυτήν την γωνία για διόρθωση του στριψίματος, θα είχαμε ένα όχημα όπου μπορεί να απείχε μια οποιαδήποτε απόσταση από την τροχιά, αλλά να κινούνταν παράλληλα με αυτή. Μπορούμε λοιπόν σα μέση λύση να δώσουμε εντολή στριψίματος στη γωνία που είναι ο μέσος όρος μεταξύ της γωνίας με το στόχο και της γωνίας με την κατεύθυνση της τροχιάς.

Η παραπάνω προσέγγιση ωστόσο, αν και βελτιώνει λίγο τα πράγματα, δεν παύει να έχει την ίδια ποιοτικώς συμπεριφορά. Αυτό διότι ο μέσος όρος είναι γραμμική πράξη, και μάλιστα με σταθερό νούμερο, κι άρα η πορεία του οχήματος θα είναι ίδια με πριν με τη διαφορά της λιγότερο "επιθετικής" γωνίας. Συνεπώς χρειαζόμαστε μια πράξη η οποία να ρυθμίζει τη γωνία στριψίματος προοδευτικά, ώστε σε μεγάλες αποκλίσεις να γίνεται επιθετικά μεγάλη για να διορθωθεί το σφάλμα, αλλά να μηδενίζεται όταν μηδενίζεται κι η απόκλιση.

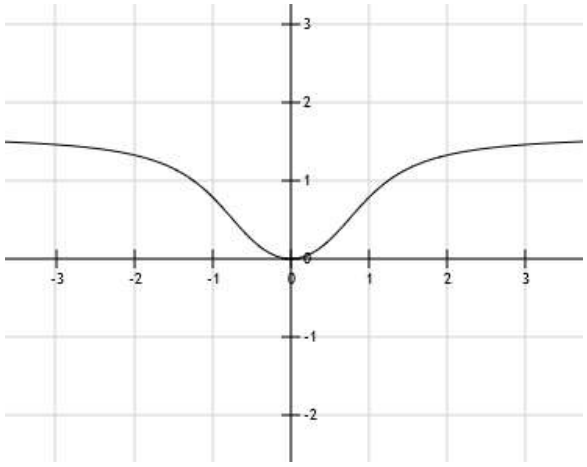
Συναρτήσεις με τέτοια συμπεριφορά υπάρχουν διαφόρων τύπων. Μια κατηγορία που μπορεί να χρησιμοποιηθεί στην περίπτωσή μας με εξαιρετικά αποτελέσματα είναι οι σιγμοειδείς συναρτήσεις (sigmoid functions). Όπως μαρτυρά το όνομά τους, η γραφική τους παράσταση μοιάζει αρκετά με το γράμμα S. Ο λόγος που ταιριάζει αυτή η συμπεριφορά στο συγκεκριμένο πρόβλημα, είναι πως κάπως αντίστοιχα ένας οδηγός στην πραγματικότητα θα πλησίαζε μια τροχιά: θα έστριβε σταδιακά μέχρι μια μέγιστη γωνία, την οποία επίσης σταδιακά θα μείωνε όσο πλησίαζε προς την τροχιά.

Μια τέτοια συνάρτηση είναι η αντίστροφη εφαπτομένη ή εφαπτομένη τόξου (arctan, atan) [38]. Στη γραφική της παράσταση[39] είναι εμφανής η σιγμοειδής συμπεριφορά της:



Σχήμα 33: Γραφική παράσταση της αντίστροφης εφαπτομένης

Η συνάρτηση έχει γενικά την σωστή συμπεριφορά, όμως το "S" είναι συμμετρικό ως προς το 0 και εκτείνεται προς τα αρνητικά. Αυτό διορθώνεται εύκολα υψώνοντας το x στο τετράγωνο. Τότε η $atan(x^2)$ γίνεται:



Σχήμα 34: Διορθωμένη atan με ύψωση του αγνώστου στο τετράγωνο

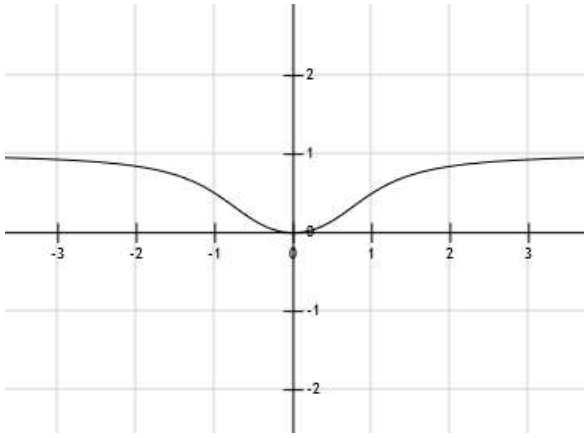
Η ύψωση στο τετράγωνο, πέρα από την εξάλειψη των αρνητικών εξόδων που θα επιτυγχανόταν και με το απόλυτο, μας δίνει και μια ακόμα αλλαγή καμπυλότητας, με αποτέλεσμα να έχουμε "S" και στα αρνητικά αλλά και στα θετικά x. Μπορούμε ήδη να φανταστούμε τον άξονα y ως επιθετικότητα στο στρίψιμο, με τον άξονα x να είναι κάποιου είδους σφάλμα. Με άλλα λόγια, μηδενίζουμε το στρίψιμο για 0 σφάλμα, και εφαρμόζουμε μέγιστο στρίψιμο για μεγάλο σφάλμα. Ταυτόχρονα η σχέση αυτή είναι αρκετά ομαλή, καθώς η συνάρτηση είναι συνεχής και παραγωγίσιμη σε όλο το διάστημα που μας ενδιαφέρει, κι άρα δεν υπάρχουν απότομες αλλαγές στην έξοδο. Μπορούμε να προσαρμόσουμε πλέον σε αυτήν τη συνάρτηση, ως x και y, τις έννοιες που μας ενδιαφέρουν. Η τελική συνάρτηση που χρησιμοποιείται είναι η εξής:

$$s = f(x) = \frac{2}{\pi} c_1 arctan(c_2 x^2) \tag{9}$$

Οι σταθερές c_1 και c_2 απλά χρησιμοποιούνται ως παράγοντες για να ρυθμίσουν τα ακριβή μεγέθη της καμπύλης, χωρίς να επηρεάζουν τη συμπεριφορά της.

Ο πολλαπλασιασμός με το $\frac{2}{\pi}$ γίνεται για να φέρει την έξοδο της συνάρτησης στο εύρος $[0, 1]$, αφού το εύρος τιμών της $arctan(x^2)$ είναι $[0, \frac{\pi}{2}]$ (και της $atan(x)$ $[-\frac{\pi}{2}, \frac{\pi}{2}]$).

Στη θέση του x θέτουμε την απόσταση traj.distanceToCurrent από τον τρέχοντα στόχο, που θεωρούμε πως είναι το σφάλμα μας. Για τις συγκεκριμένες τιμές των $c_1 = 2.0$ και $c_2 = 0.5$ η γραφική παράσταση γίνεται:



Σχήμα 35: Η συνάρτηση διόρθωσης με τις τελικές τιμές των σταθερών

Συνεπώς από την παραπάνω πράξη παίρνουμε ένα παράγοντα s (από το smoothing factor), με εύρος $[0, 1]$ και παράμετρο την απόσταση από τον στόχο. Ο s χρησιμοποιείται ως στάθμη της γωνίας προς τον στόχο στον σταθμισμένο μέσο όρο που αναφέραμε προηγούμενως. Σε αντίθεση με πριν, ο μέσος όρος αυτός δεν είναι σταθερός κι έχει τη συμπεριφορά της $f(x)$ λόγω του παράγοντα s. Έτσι, η γωνία στριψίματος του τιμονιού υπολογίζεται ως εξής:

$$a_{steer} = \frac{a_{toPathDir} + s * a_{toTarget}}{1 + s} \tag{10}$$

Το $a_{toPathDir}$ είναι η γωνία μεταξύ της τρέχουσας κατεύθυνσης του οχήματος και της κατεύθυνσης της τροχιάς. Συνεπώς, για μεγάλη απόσταση από τον στόχο το s γίνεται 1 και άρα έχουμε έναν κανονικό μέσο όρο, κι όσο το όχημα πλησιάζει προς τον στόχο το s γίνεται σιγμοειδώς 0, κι άρα η γωνία στριψίματος γίνεται ίση με την γωνία με την κατεύθυνση της τροχιάς. Το ότι ο παράγοντας s έχει πολύ μικρές τιμές κοντά στο 0, κι "αργεί" να πάρει μεγάλες τιμές, μας χρησιμεύει στο να μην κάνει ο οδηγός πολλές και απότομες διορθώσεις όταν πηγαίνει ευθεία, καθώς τα μικροσφάλματα διορθώνονται άμεσα με αντίστοιχες λεπτές διορθώσεις.

Η υλοποίηση των παραπάνω στην SteerToTarget γίνεται ως ακολούθως. Αρχικά υπολογίζουμε την γωνία της κατεύθυνσης προς την τροχιά A.57. Αντίστοιχα με πριν, η γωνία που μας δίνει η Unity δεν έχει πρόσημο, άρα κάνουμε την ίδια διαδικασία με το εξωτερικό γινόμενο για την γωνία προς τον στόχο A.58. Αφού βρούμε τη γωνία, μπορούμε πλέον να υπολογίσουμε τον παράγοντα s και το steerAngle A.59. Το steerAngle όμως μπορεί να είναι φυσικά αδύνατο για το όχημα αφού οι ρόδες έχουν περιορισμένο εύρος, άρα το περιορίζουμε στα μέγιστα όρια A.60. Πολλαπλασιάζουμε τη γωνία με το μέγεθος απόκρισης που έχουμε ορίσει στον οδηγό (Behaviour.responsiveness) για να πάρουμε την τελική γωνία A.61 και τέλος, για να χρησιμοποιήσουμε την επιθετικότητα (Behaviour.steeringAggression) αντί να θέσουμε απευθείας το finalSteerAngle στο Handling.steer, το αυξάνουμε ή μειώνουμε με ρυθμό ίσο με το μέγεθος της επιθετικότητας A.62.

4.2.4 Χειρισμός πεταλιών

Ακριβώς επόμενη συνάρτηση που εκτελείται στην επανάληψη είναι η CalculatePedals την οποία θα δούμε βήμα-βήμα παρακάτω. Ξεκινούμε αρχικοποιώντας τις τιμές των πεταλιών σε 0, και παίρνουμε και την ταχύτητα του οχήματος από το RigidBody component του A.63.

Έπειτα, επιλέγουμε ένα checkpoint που βρίσκεται μετέπειτα στην τροχιά. Το πόσο μετέπειτα, κρίνεται από το μέγεθος της ταχύτητας που κινούμαστε. Το σκεπτικό είναι ότι όσο πιο γρήγορα κινούμαστε, όπως και στην πραγματικότητα, τόσο πιο μπροστά στην πορεία μας συγκεντρώνουμε την προσοχή μας. Αν ο δείκτης του σημείου που υπολογίσαμε ξεπερνάει το μέγεθος της λίστας των σημείων της τροχιάς (επειδή πλησιάζουμε στο τέλος της) τότε το περιορίζουμε A.64.

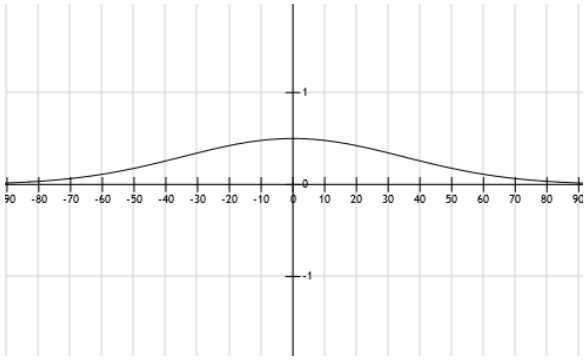
Έχοντας επιλέξει σημείο, υπολογίζουμε το διάνυσμα turnVector που ενώνει το όχημα με το σημείο αυτό, και μετά τη γωνία αυτού του διανύσματος με το διάνυσμα της κατεύθυνσης του οχήματος. Στην προκειμένη δε μας ενδιαφέρει το πρόσημο, συνεπώς μας αρκεί απλά η απόλυτη τιμή που επιστρέφει η Unity A.65.

Εδώ έρχεται σε χρήση το look-up table που είχαμε αναφέρει πιο πριν. Η λογική του είναι πως για συγκεκριμένη γωνία της πορείας, βάζουμε ένα όριο ταχύτητας. Για παράδειγμα, δεν επιτρέπουμε να κινείται το όχημα ταχύτερα από $5m/s$ $(= 18km/h)$ σε γωνίες μεγαλύτερες των 40 μοιρών. Έχοντας λοιπόν ορίσει το speedLUT στην InitializeParams, κάνουμε απλή γραμμική παρεμβολή για τον υπολογισμό της ταχύτητας που αντιστοιχεί στη γωνία A.66. Μετά εφαρμόζουμε το καθολικό όριο ταχύτητας, που υπάρχει στο state A.67. Αν έχει υπάρξει συνθήκη ακινητοποίησης, τότε η ταχύτητα-στόχος γίνεται 0 A.68.

Και τώρα υπολογίζουμε τις τιμές των πεταλιών. Αρχικά αν η τρέχουσα ταχύτητα είναι μικρότερη του στόχου, πρέπει να επιταχύνουμε. Για να γίνει αυτό, μηδενίζουμε το φρένο, και εφαρμόζουμε γκάζι που ορίζεται από μια σχέση συναρτήσει της γωνίας που έχει υπολογιστεί. Αντίστοιχα με την ομαλή συνάρτηση του τιμονιού, κι εδώ επιλέγεται μια παρόμοια σχέση:

$$throttle = 0.5 \cdot 2^{-0.006 \cdot x^2} \tag{11}$$

όπου x η υπολογισμένη γωνία οχήματος-τροχιάς. Η συνάρτηση αυτή έχει την εξής συμπεριφορά:



Σχήμα 36: Η συνάρτηση υπολογισμού του γκαζιού

όπου βλέπουμε την ομαλή μετάβαση σε σχεδόν 0 γκάζι σε πολύ μεγάλες γωνίες και το ανάποδο. Με άλλα λόγια, αν έχουμε μεγάλη γωνία σε σχέση με το που πηγαίνει ο δρόμος, πατάμε ελάχιστο γκάζι ώστε να ευθυγραμμιστούμε, κι έπειτα μπορούμε να πατήσουμε όσο χρειαστεί για να φτάσουμε το targetSpeed A.69.

Αν ωστόσο το targetSpeed είναι μικρότερο Α.70, τότε πρώτο βήμα είναι να μηδενίσουμε το γκάζι. Έπειτα, υπολογίζουμε τη διαφορά ταχύτητας σε σχέση με το targetSpeed, ώστε να φρενάρουμε αναλόγως. Τότε εφαρμόζουμε φρένο που σχετίζεται με την διαφορά ταχύτητας, χρησιμοποιώντας αυτήν τη φορά σκέτη την $arctan(x)$ όπου x το vel_diff. Να σημειωθεί πως λόγω της συνθήκης, το vel_diff θα είναι σίγουρα μη αρνητικό. Πάλι πολλαπλασιάζουμε με το $\frac{2}{\pi}$ για να έρθει ο παράγοντας στο $[-1, 1]$, και επίσης με το vhcl.maxBrakeTorque για να πάρουμε την τελική ροπή πέδησης. Επίσης, στην περίπτωση που υπάρχει εντολή για ακινητοποίηση, εφαρμόζεται λίγο φρένο παραπάνω. Αυτό διότι η $atan$ περνάει από το $[0, 0]$, άρα αν πλησιάσουμε την ακινητοποίηση (vel_diff -> 0) τότε και το φρένο μηδενίζεται, δηλαδή θα έχουμε μια πολύ αργή διόρθωση με το όχημα να πηγαίνει με απειροελάχιστες ταχύτητες.

Τέλος, υπάρχουν περιπτώσεις όπου το γκάζι επηρεάζεται από παραμέτρους του κινητήρα/κιβωτίου κι όχι λόγω διαδρομής, στροφών ή ορίων ταχυτήτων. Αυτά θα γίνουν στη συνάρτηση UpdatePowertrain στην οποία χειριζόμαστε και το κιβώτιο και θα δούμε καλύτερα στην επόμενη παράγραφο.

Κατ' αρχάς, λαμβάνουμε υπ' όψη τις engn.stallRpm, δηλαδή την κατάσταση χαμηλών στροφών κάτω από τις οποίες ο κινητήρας θα σταματήσει να λειτουργεί, και να εφαρμόσουμε όσο γκάζι χρειάζεται Α.71. Πολλά αυτοκίνητα, αν όχι πλέον όλα, αυτό το κάνουν αυτόματα, δηλαδή μπορούμε αφήνωντας το γκάζι τελείως να κινούμαστε με τις ελάχιστες δυνατές στροφές.

Επίσης, ο κινητήρας για λόγους ασφαλείας περιορίζεται στον αριθμό των στροφών ανά λεπτό που μπορεί να λειτουργεί. Αυτό το κάνει ο περιοριστής στροφών (revolution limiter, rev limiter) ο οποίος, αναλόγως και το όχημα, έχει ένα εύρος $[L_{low}, L_{high}]$ στο οποίο ενεργοποιείται κι απενεργοποιείται. Δηλαδή, φτάνοντας τις L_{high} σ/λ ενεργοποιείται, αλλά δεν απενεργοποιείται μέχρι να πέσουμε κάτω από τις L_{low}. Το εύρος αυτό μπορεί να είναι της τάξης των 0-500 σ/λ. Είχαμε ορίσει αυτά τα όρια στο powertrain.cs Α.72.

Έτσι μπορούμε με απλούς ελέγχους να χειριστούμε αυτές τις καταστάσεις Α.73. Η engn.letThrottle είναι μια bool που χρησιμοποιείται λίγο μετά για να μηδενίσει το γκάζι Α.74. Η UpdatePowertrain τρέχει μετά την CalculatePedals, συνεπώς γνωρίζουμε πως ο μηδενισμός αυτός του γκαζιού είναι τελικός ό,τι κι αν είχε υπολογίσει η δεύτερη (δηλαδή κάνει overwrite).

4.2.5 Χειρισμός σχέσεων μετάδοσης

Ο χειρισμός του κιβωτίου γίνεται στην UpdatePowertrain την οποία είχαμε δει και στο κεφάλαιο της προσομοίωσης του κινητήρα και της μετάδοσης. Η αλλαγή σχέσεων και η εφαρμογή του συμπλέκτη γίνεται πλέον αρκετά απλά, ελέγχοντας κάποιες συνθήκες.

Πριν δούμε την υλοποίηση, να μιλήσουμε για κάποιες σημαντικές μεταβλητές. Η allowGearChangeTimeout είναι ένας μετρητής χρόνου που επιτρέπει ή όχι την αλλαγή σχέσης, για να προσομοιώσουμε την καθυστέρηση που έχει ένας οδηγός στην πραγματικότητα. Η currentlyOnGearChange είναι μια bool που μας δίνει την πληροφορία του αν υπάρχει σε εξέλιξη άλλη αλλαγή. Η gearChangeTime είναι ένας float που μετράει το χρόνο της τρέχουσας αλλαγής, ώστε να ξέρουμε αν αυτή ολοκληρώθηκε, το οποίο θα γίνει όταν αυτή φτάσει την τιμή 1.0f.

Αρχικά, ελέγχουμε για το αν είμαστε κατά τη διάρκεια αλλαγής. Τότε, αυξάνουμε τον μετρητή gearChangeTime, και δίνουμε στον συμπλέκτη ανάλογη τιμή, ώστε να είναι πλήρως πατημένος στην αρχή της αλλαγής, και σταδιακά να αφήνεται για να υπάρχει ομαλή αλλαγή. Επίσης κατά τη διάρκεια αλλαγής σχέσης, δεν επιτρέπουμε να εφαρμόζεται γκάζι μέσω της engn.letThrottle. Όταν gearChangeTime $>$ 1.0f τότε η αλλαγή ολοκληρώθηκε, ενημερώνουμε πως δεν βρισκόμαστε κατά τη διάρκεια αλλαγής και μηδενίζουμε τον μετρητή A.75.

Έπειτα ανανεώνουμε το grbx.allowGearChangeTimeout A.76. Έχοντας ορίσει κάποια εύρη στροφών στα οποία θεωρούμε σωστό να εκτελείται η κάθε αλλαγήA.77 (αυτά τα ορίζουμε στην κλάση Handling κι όχι στην Engine ή στη Gearbox μιας και εννοιολογικά είναι περισσότερο θέμα χειρισμού παρά χαρακτηριστικών του οχήματος) εκτελούμε τις αλλαγές αν αυτό απαιτείται από τις τρέχουσες στροφές ανά λεπτό A.78.

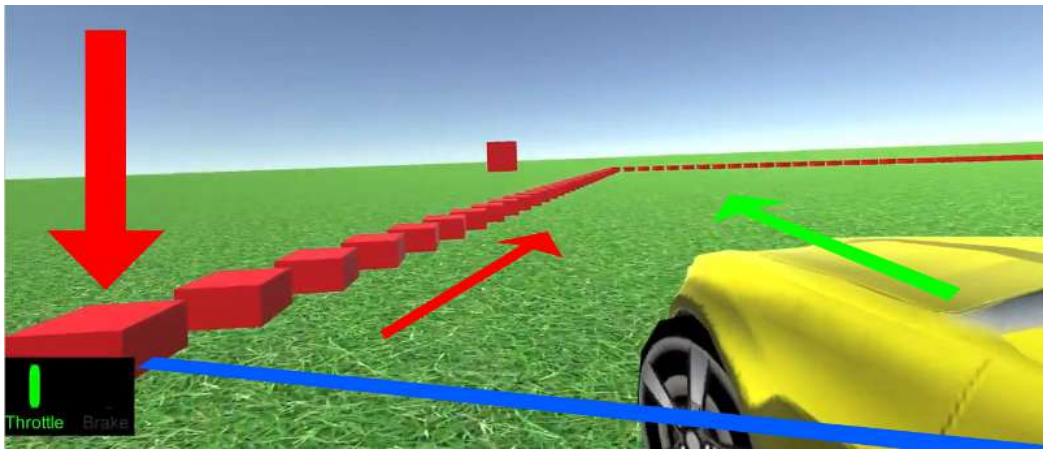
Η grbx.shiftUp υλοποιείται στο powertrain.cs. Αν η σχέση στην οποία βρισκόμαστε είναι η τελευταία (στην περίπτωσή μας η 5η) τότε απλά επιστρέφει χωρίς να κάνει τίποτα. Αντίστοιχα δεν κάνει κάτι αν δεν έχει μηδενιστεί ο μετρητής allowGearChangeTimeout. Σε αντίθετη περίπτωση αρχικοποιεί τον μετρητή, και καλεί την setGear με τον αριθμό της επόμενης σχέσης A.79.

Η grbx.setGear τελικά θέτει την μεταβλητή gear που χρησιμοποιείται κατά τα γνωστά για τον υπολογισμό της ροπής εξόδου, και ενημερώνει την currentlyOnGearChange που είδαμε πριν ως αληθή Α.80. Ακριβώς ίδια είναι η συνάρτηση shiftDown, απλώς με μείωση του αριθμού της σχέσης κι έλεγχο για το αν είμαστε στην ελάχιστη (1η) ώστε να αγνοηθεί η αλλαγή Α.81:

Μπορούμε να βλέπουμε σε πραγματικό χρόνο τις τιμές των πεταλιών, τη σχέση στο κιβώτιο, την ταχύτητα κίνησης, τις στροφές του κινητήρα καθώς και την τρέχουσα μέση κατανάλωση μέσω του ειδικού GUI στοιχείου που αναπτύχθηκε στη Unity:

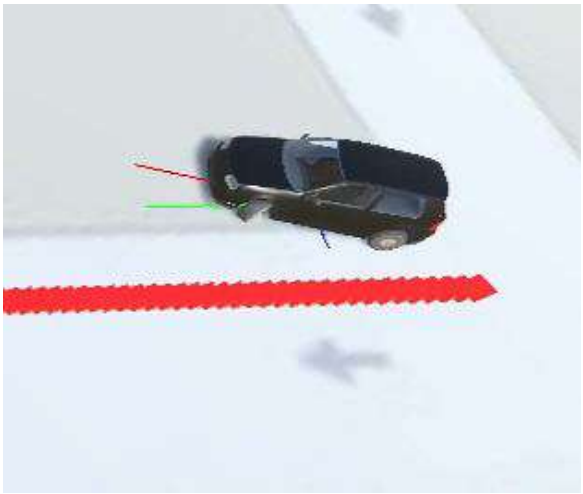
Σχήμα 37: Ενδείξεις στο GUI για πληροφορίες πεταλιών και κινητήρα

Τα διάφορα σημεία και διανύσματα που χρησιμοποιήσαμε φαίνονται στις εικόνες που ακολουθούν. Σε αυτήν την εικόνα, το κατακόρυφο κόκκινο βέλος δείχνει τον τρέχοντα στόχο. Το άλλο κόκκινο βέλος δείχνει την κατεύθυνση της τροχιάς. Ο κόκκινος κύβος που αιωρείται υποδεικνύει το σημείο που βρίσκεται πιο μπροστά στην τροχιά και μέσω του οποίου ελέγχουμε για γωνίες/στροφές. Το πράσινο βέλος είναι η κατεύθυνση του οχήματος, και η μπλε γραμμή συνδέει το όχημα με τον τρέχοντα στόχο. Επίσης φαίνεται πως λόγω ευθυγράμμισης με την τροχιά το γκάζι δεν περιορίζεται, κι επίσης λόγω της ευθείας που ανιχνεύεται δεν χρησιμοποιείται το φρένο. Το στρίψιμο τέλος δεν είναι ιδιαίτερα απότομο καθώς το όχημα δεν απέχει πολύ από την τροχιά.



Σχήμα 38: Εποπτικά τα διανύσματα και σημεία που χρησιμοποιούμε α)

Επίσης εδώ βλέπουμε με μπλε το διάνυσμα που συνδέει το όχημα με το στόχο, με κόκκινο το διάνυσμα κατεύθυνσης του οχήματος και με πράσινο το διάνυσμα κατεύθυνσης της τροχιάς.



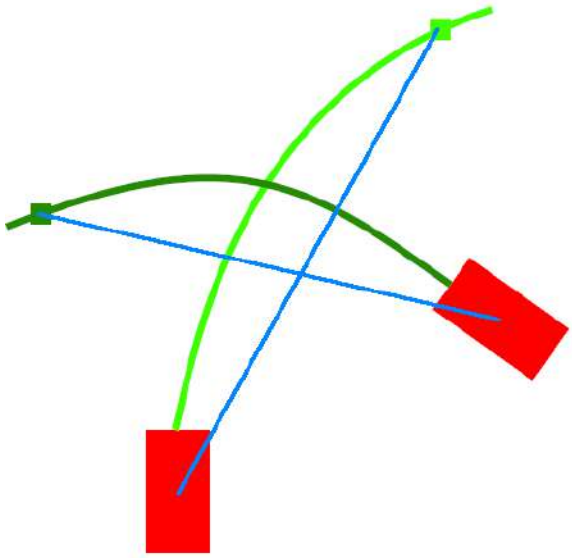
Σχήμα 39: Εποπτικά τα διανύσματα και σημεία που χρησιμοποιούμε β)

Το όχημα πλέον μπορεί να ελεγχθεί πλήρως, άρα μπορούμε να περάσουμε στη συμπεριφορά του σε σχέση με τα άλλα οχήματα που κυκλοφορούν στον δρόμο που σχεδιάσαμε.

4.3 Παραχώρηση προτεραιότητας

Είναι πολύ πιθανό κατά τη διάρκεια της προσομοίωσης, ένα όχημα να προσεγγίζει ένα άλλο με τέτοιο τρόπο ώστε να χρειαστεί παραχώρηση προτεραιότητας στο δεύτερο [40]. Την προτεραιότητα την έχει το όχημα που προσεγγίζει από τα δεξιά. Το όχημα που πρέπει να παραχωρήσει προτεραιότητα, ακινητοποιείται μέχρι να σταματήσει να ικανοποιείται η συνθήκη. Η συνάρτηση που υλοποιεί τα παραπάνω είναι η Awareness που εκτελείται στην ίδια επανάληψη με τις προηγούμενες συναρτήσεις στο component Move του οχήματος. Η διαδικασία που εκτελούμε είναι η εξής.

Σκοπός είναι να δούμε το αν η τροχιά ενός οχήματος πρόκειται να τέμνει την τροχιά κάποιου άλλου. Ιδανικά θα ελέγχαμε για τις τομές των δυο καμπυλών τους, ωστόσο η αριθμητική επίλυση της τομής δυο Bezier είναι ένα δύσκολο κομμάτι της υπολογιστικής γεωμετρίας με τους υπάρχοντες αλγορίθμους να είναι ακριβοί υπολογιστικά, να έχουν ασταθείς λύσεις κ.α. [41]. Μπορούμε όμως να προσεγγίσουμε αυτήν την πράξη με τον ίδιο τρόπο που απλοποιήσαμε τις καμπύλες για τον υπολογισμό του μήκους τους, δηλαδή προσεγγίζοντάς τις με ευθύγραμμα τμήματα. Συνεπώς, μπορούμε να ελέγχουμε, αντί για τις ίδιες τις καμπύλες, τομή ευθυγράμμων τμημάτων που εκτείνονται από τη θέση του οχήματος μέχρι ένα μετέπειτα σημείο της τροχιάς. Η θέση του σημείου μπορεί να εξαρτάται από την ταχύτητα του εκάστοτε οχήματος, με τη λογική πως δε χρειάζεται έλεγχος τομής τροχιών αν οι ταχύτητες είναι μικρές σε σχέση με την απόσταση.



Σχήμα 40: Προσέγγιση τομής των Bezier με ευθύγραμμα τμήματα

Για την τομή ευθυγράμμων τμημάτων, πρέπει να ελέγξουμε το αν και τα δυο σημεία του ενός βρίσκονται εκατέρωθεν του άλλου τμήματος κι αντίστροφα. Αυτό φαίνεται από την παρακάτω εικόνα, όπου στην πρώτη περίπτωση και τα δυο σημεία και των δυο τμημάτων βρίσκονται στην ίδια μεριά του άλλου, στη δεύτερη μόνο του ενός βρίσκονται εκατέρωθεν, και στην τρίτη και των δυο βρίσκονται εκατέρωθεν κι άρα έχουμε τομή.

Σχήμα 41: Συνθήκη τομής ευθυγράμμων τμημάτων

Τη μεριά στην οποία βρίσκεται ένα σημείο σε σχέση με την ευθεία που ορίζει ένα ευθύγραμμο τμήμα μπορούμε να τη βρούμε με τη χρήση εξωτερικού γινομένου. Το εξωτερικό γινόμενο δυο διανυσμάτων $\vec{a}$ και $\vec{b}$ υπολογίζεται ως η ορίζουσα του πίνακα με γραμμές τις συντεταγμένες του κάθε διανύσματος, κι έχει κατεύθυνση κάθετη στο επίπεδο των δυο διανυσμάτων σύμφωνα

με τον κανόνα του δεξιού χεριού:

$$\vec{a} \times \vec{b} = \begin{vmatrix} a_x & a_y \\ b_x & b_y \end{vmatrix} \hat{z} \tag{12}$$

Έχοντας τώρα τρία σημεία A, B και C, ορίζουμε:

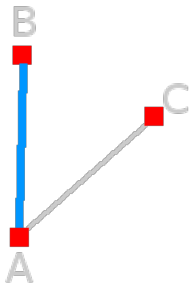
$$\vec{a} = \overrightarrow{AB} \tag{13}$$

και

$$\vec{b} = \overrightarrow{AC} \tag{14}$$

Τότε η μεριά που βρίσκεται το σημείο C σε σχέση με την ευθεία που ορίζει το AB είναι:

$$sign((B_x - A_x)(C_y - A_y) - (B_y - A_y)(C_x - A_x)) \tag{15}$$



Σχήμα 42: Μεριά ενός σημείου σε σχέση με ευθύγραμμο τμήμα

Στο script helperClasses.cs υλοποιούνται κάποιες χρήσιμες συναρτήσεις για την εύρεση των παραπάνω. Επίσης εδώ περιέχονται κι οι συναρτήσεις για τη δημιουργία καμπυλών Bezier. Ορίζουμε μια κλάση Utils που είναι static ώστε να μπορεί να γίνει η χρήση της χωρίς τη δημιουργία αντικειμένου από αυτήν. Υλοποιούνται συναρτήσεις για τον υπολογισμό της μεριάς ενός διανύσματος $\overrightarrow{p_1 p_2}$ που βρίσκεται ένα σημείο $\vec{p}$ A.82 και για το εξωτερικό γινόμενο A.83, η οποία με τη σειρά της χρησιμοποιείται για τον έλεγχο του αν δυο ευθύγραμμα τμήματα τέμνονται A.84.

Κι άρα τώρα μπορούμε να ελέγξουμε για προτεραιότητα. Τελικό αποτέλεσμα θα είναι να θέσουμε την τιμή της bhvr.stop. Συνεπώς ξεκινάμε με αρχικοποίηση A.85, επιλέγουμε ένα σημείο μετέπειτα στην τροχιά που εξαρτάται από την ταχύτητα A.86 και παίρνουμε τα δυο σημεία που θα ορίσουν το διάνυσμα του τρέχοντος οχήματος A.87.

Για κάθε ένα από τα υπόλοιπα οχήματα, παίρνουμε μια αναφορά στο Move component τους, κι ελέγχουμε για το αν πρόκειται για το τρέχον όχημα, στην οποία περίπτωση το αγνοούμε. Αλλιώς, κρατάμε σε μια μεταβλητή τη θέση του A.88.

Αντίστοιχα υπολογίζουμε το μετέπειτα σημείο του άλλου οχήματος A.89 και τελικά κάνουμε τον έλεγχο με τις παραπάνω συναρτήσεις και αν ισχύει η συνθήκη, δίνουμε εντολή στο τρέχον όχημα να σταματήσει A.90.

Να σημειωθεί πως δεν ελέγχουμε τη μεριά σε σχέση με την ευθεία της κατεύθυνσης του οχήματος, αλλά με την ευθεία που ορίζεται από την προσέγγιση της καμπύλης. Αυτό διότι η προτεραιότητα δίνεται σε σχέση με το που θα καταλήξουν τα οχήματα που εμπλέκονται αν ακολουθήσουν την τροχιά τους, κι όχι απόλυτα με το που είναι τη στιγμή του ελέγχου.

Στο επόμενο κεφάλαιο θα δούμε την υλοποίηση του κεντρικού script state.cs που περιέχει τη βασική λογική της προσομοίωσης, καθώς και την εκτέλεση της ίδιας της προσομοίωσης.

4.4 Διαδικασία προσομοίωσης

4.4.1 Υλοποίηση κύριας λογικής

Το state.cs είναι το κύριο script που αναλαμβάνει τη διαδικασία της προσομοίωσης και το χειρισμό του γραφικού περιβάλλοντος του χρήστη. Γι' αυτό περιλαμβάνει κάποιες βασικές μεταβλητές, όπως αυτήν που θα σημάνει το ξεκίνημα της προσομοίωσης, τον αριθμό των οχημάτων, τη λίστα των αντικειμένων (GameObject) των οχημάτων, το καθολικό όριο ταχύτητας, αναφορές προς τα στοιχεία της γραφικής διεπαφής κ.α A.91A.92A.93A.94A.95.

Ξεκινώντας, καλείται η Start συνάρτηση αυτού του script A.96. Οι πρώτες δυο συναρτήσεις αφορούν την συγκέντρωση κι οπτικοποίηση των στατιστικών και θα τις δούμε στο επόμενο κεφάλαιο. Η συνάρτηση LoadAndRenderGround είναι αντίστοιχη με αυτή που φόρτωνε το χάρτη όπως είδαμε στο κεφάλαιο του editor. Η διαφορά εδώ είναι πως χρησιμοποιεί τα δεδομένα του αρχείου του χάρτη για να κατασκευάσει ένα επίπεδο στον τριδιάστατο χώρο με τις παραμέτρους του αρχείου, και με texture την εικόνα που είχαμε σώσει στο αρχείο.

Θεωρούμε πως έχουμε διαβάσει τις παραμέτρους από το αρχείο, με τα δεδομένα της εικόνας να βρίσκονται στην μεταβλητή tex τύπου Texture2D, την κλίμακα του χάρτη στη μεταβλητή s και το μέγεθος του χάρτη στις μεταβλητές mapLength και mapWidth. Με αυτές τότε γίνεται η δημιουργία του επιπέδου που αντιπροσωπεύει το δάπεδο (και το δρόμο) στην LoadAndRenderGround A.97.

Σχήμα 43: Το επίπεδο που δημιουργείται από τα δεδομένα του χάρτη

Σειρά είναι της PopulateWorld, που δημιουργεί τις οντότητες των καμπυλών, των λωρίδων, των συνδέσεων και των δρόμων. Είναι παρόμοια με τη συνάρτηση που δείξαμε να κάνει το ίδιο κατά τη φόρτωση του χάρτη στον editor A.98. Μετά η SetupLaneIndices βρίσκει κι ορίζει

τους δείκτες των καμπυλών ως αριστερά και δεξιά όρια της εκάστοτε λωρίδας Α.99. Έπειτα γίνεται έλεγχος για λωρίδες εκκίνησης και τερματισμού Α.100 και τέλος βρίσκουμε τις συνδέσεις μεταξύ των λωρίδων και τις τοποθετούμε στις αντίστοιχες λίστες, τις οποίες και είδαμε να χρησιμοποιούμε κατά την επιλογή επόμενης λωρίδας από τα οχήματαΑ.101.

Πατώντας το πλήκτρο space, ξεκινάει η προσομοίωση με το πρώτο αυτοκίνητο Α.102. Η DispatchCar επιστρέφει χωρίς να κάνει τίποτα αν δεν έχει ενεργοποιηθεί προηγουμένως η running. Επιλέγει ένα σημείο για την δημιουργία του αυτοκινήτου, που είναι σίγουρα πολύ μακριά από τα υπόλοιπα, στην προκείμενη περίπτωση στο $[100, 100, 100]$. Αυτό διότι, αν και άμεσα όπως είδαμε το αυτοκίνητο τοποθετείται κι ευθυγραμμίζεται στην λωρίδα εκκίνησής του από το Move component του, αν το δημιουργούσαμε στο $[0, 0, 0]$ θα μπορούσε να έχει μια εν δυνάμει σύγκρουση με όχημα που ήδη βρίσκεται σε εκείνο σημείο.

Έπειτα δημιουργούμε το αντικείμενο του οχήματος μέσω της Instantiate συνάρτησης της Unity, και το προσθέτουμε στη λίστα. Τέλος καλούμε την Invoke στον εαυτό της με μια τυχαία καθυστέρηση από 3 έως 8 δευτερόλεπτα για να εκτελείται επ' αόριστον. Η Invoke παίρνει ορίσματα το όνομα μιας συνάρτησης που θέλουμε να εκτελέσει κι ένα χρόνο καθυστέρησης. Μετά από αυτόν τον χρόνο η Invoke καλεί τη συνάρτηση ασύγχρονα χωρίς να εμποδίζει την ροή του υπόλοιπου προγράμματος Α.103.

Μια άλλη λειτουργία της Update του state είναι η ικανότητα επιλογής αυτοκινήτου με το ποντίκι για να δούμε τα στατιστικά του. Αυτή η επιλογή γίνεται απευθείας στο τριδιάστατο περιβάλλον με τη χρήση raycasting[42]. Το raycast είναι προέκταση μιας ημιευθείας από ένα αρχικό σημείο προς μια ορισμένη κατεύθυνση, κι ίσως για περιορισμένη απόσταση. Χρησιμοποιώντας την υλοποίηση της μηχανής φυσικής, μπορούμε να δούμε το αντικείμενο που αυτή η ημιευθεία τέμνει, κι αν αυτό πρόκειται για όχημα τότε ανακτούμε το δείκτη του στη λίστα των οχημάτων ώστε να μπορούμε να δούμε τα δεδομένα που το αφορούν. Τότε αλλάζουμε τα χρώματα του αμαξώματος ώστε να είναι εμφανές στο χρήστη το ποιο όχημα επέλεξε. Πατώντας το αριστερό πλήκτρο του ποντικιού, δημιουργούμε ένα ray στην κατεύθυνση της κάμερας ξεκινώντας από το πίξελ στο οποίο πατήσαμε το πλήκτρο. Σε αυτό βοηθάει η συνάρτηση ScreenPointToRay που είναι μέρος του component της κάμερας και επιστρέφει ένα αντικείμενο τύπου ray για δεδομένες συντεταγμένες στην οθόνη Α.104.

Αν η ακτίνα πετύχει αντικείμενο, ελέγχουμε αν πρόκειται για όχημα και διαβάζουμε το δείκτη του, αφού έχουμε φροντίσει να του δίνουμε ως όνομα το δείκτη του στη λίστα των οχημάτων Α.105. Αν διαβαστεί δείκτης σωστά, τότε ρυθμίζουμε το Renderer component του αντικειμένου να κάνει μια εναλλαγή χρωμάτων με το χρόνο Α.106.

Ο CarMarkShader είναι ένας shader για τους οποίους θα μιλήσουμε πιο αναλυτικά στο κεφάλαιο της οπτικοποίησης, ωστόσο να αναφέρουμε απλά πως μεταβάλλει το χρώμα του αντικειμένου στο οποίο εφαρμόζεται με μια συνάρτηση που ρυθμίζει το πράσινο μέρος του χρώματος σε σχέση με το χρόνο με τη χρήση τριγωνομετρικών συναρτήσεων Α.107.

Τέλος η Update ελέγχει τις μεταβλητές letDestroy των οχημάτων για να αφαιρέσει από τον κόσμο όποιο έφτασε στο τέλος της διαδρομής του Α.108.

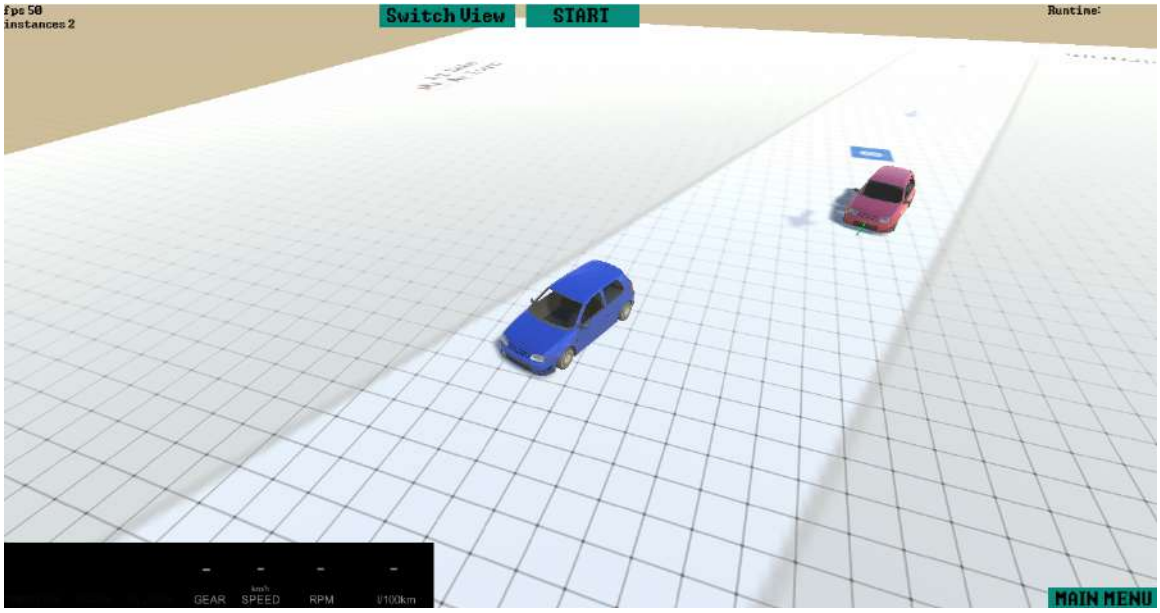
Με αυτά μπορούμε πλέον να περάσουμε στην εκτέλεση της προσομοίωσης φορτώνοντας ένα αρχείο χάρτη και ξεκινώντας με τη δημιουργία του πρώτου οχήματος.

4.4.2 Εκτέλεση της προσομοίωσης και γραφικό περιβάλλον

Ξεκινούμε ανοίγοντας το πρόγραμμα στην αρχική οθόνη του, κι επιλέγουμε SIMULATION. Τότε, όπως και στον editor, ένας διάλογος μας καλεί να επιλέξουμε αρχείο χάρτη. Επιλέγουμε το αρχείο και τότε θα φορτώσει ο κόσμος:

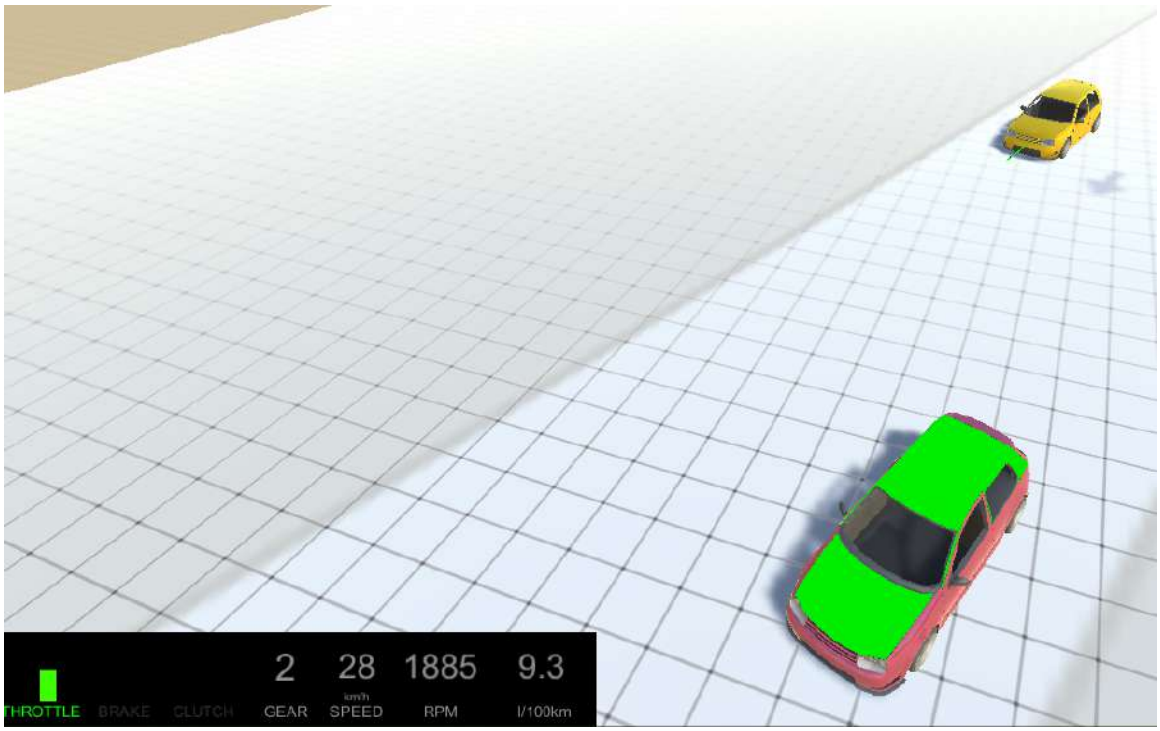
Σχήμα 44: Ανοίγοντας το αρχείο του χάρτη φορτώνει ο κόσμος της προσομοίωσης

Έπειτα ξεκινούμε την προσομοίωση και σύντομα δυο οχήματα δημιουργούνται σε μια λωρίδα εκκίνησης:



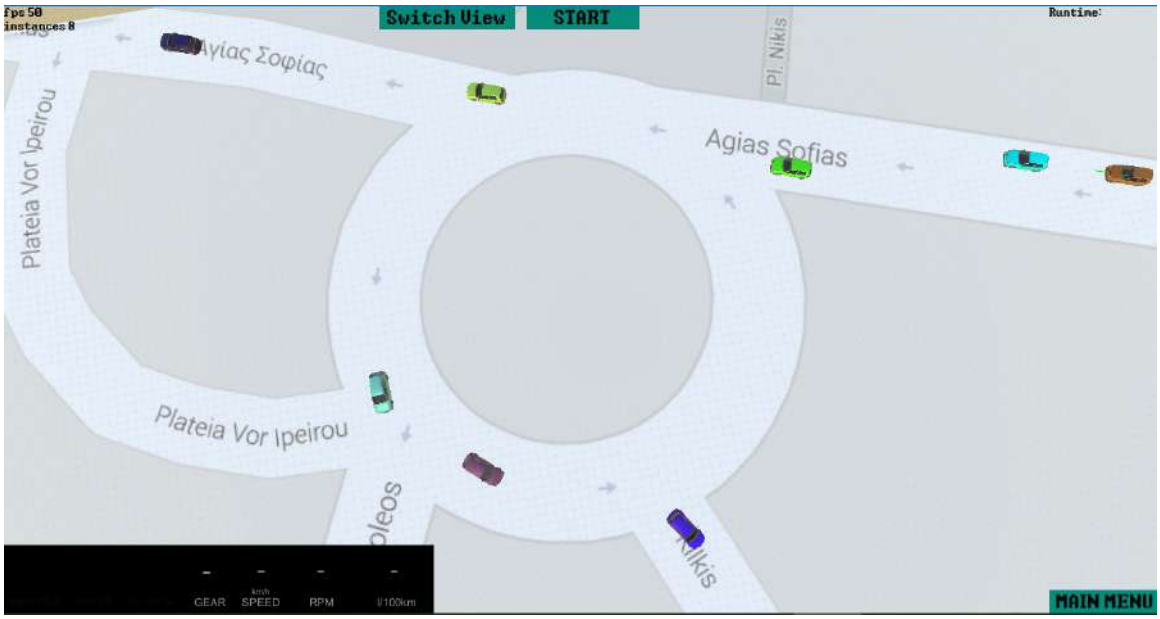
Σχήμα 45: Δυο οχήματα μόλις μετά την έναρξη της προσομοίωσης

Μπορούμε να επιλέξουμε ένα όχημα πατώντας πάνω του. Τότε κάτω αριστερά θα δούμε τα δεδομένα χειρισμού του οδηγού καθώς και δεδομένα του κινητήρα του συγκεκριμένου οχήματος:



Σχήμα 46: Μετά την επιλογή οχήματος βλέπουμε τα δεδομένα του

Αφήνοντας λίγη ώρα την προσομοίωση, κι άλλα οχήματα έχουν δημιουργηθεί και πλοηγούνται στον χάρτη:



Σχήμα 47: Η προσομοίωση σε εξέλιξη

Έχοντας χειριστεί την προσομοίωση, τώρα μπορούμε να προχωρήσουμε στην οπτικοποίηση των στατιστικών που συλλέγονται κατά τη διάρκειά της.

5 Ανάλυση Αποτελεσμάτων

5.1 Υπολογισμός στατιστικών

Θεωρητικά έχουμε πολλών ειδών δεδομένα που θα μπορούσαμε να αναλύσουμε. Έχουμε κατ' αρχάς ό,τι αφορά την ενεργειακή απόδοση του οδικού συστήματος, το οποίο μπορούμε να το αποτιμήσουμε με τα δεδομένα κατανάλωσης καυσίμου που έχουμε ήδη παράξει. Ένας πιθανός στόχος λοιπόν με τη χρήση αυτού του λογισμικού θα ήταν να ελαχιστοποιηθεί η συνολική κατανάλωση καυσίμου που παρατηρείται μετά από ένα μεγάλο χρονικό διάστημα, θεωρώντας απομονωμένο το σύστημα που μελετούμε, με τις ροές εισόδου σε αυτό να προκύπτουν από πραγματικές μετρήσεις. Επίσης, ένας στόχος θα μπορούσε να είναι η επίτευξη όσο το δυνατόν

καλύτερης ροής αυτοκινήτων στο οδικό σύστημα, άρα ο στόχος θα ήταν η μεγιστοποίηση της μέσης ταχύτητας όλων των οχημάτων μετά από ένα χρονικό διάστημα. Ένας άλλος στόχος θα ήταν όχι η αύξηση του μέσου όρου των ταχυτήτων, αλλά η αύξηση του πάνω ορίου της μέγιστης ταχύτητας που μπορεί να επιτευχθεί αν αυτό χρειαστεί. Για παράδειγμα, μπορεί να μην μας ενδιαφέρει η μαζική ροή να είναι γρήγορη, αλλά αν χρειαστεί κάποιο μεμονωμένο όχημα να περάσει από αυτήν πολύ γρήγορα (π.χ ασθενοφόρο όχημα), αυτό να μπορεί να γίνει λόγω σχεδιασμού του δρόμου.

Συγκεκριμένα εδώ θα υλοποιήσουμε ανάλυση στατιστικών μέσης ταχύτητας σε διάσπαρτα σημεία μέτρησης του χάρτη. Η οπτικοποίησή τους θα μας δώσει μια χρωματική αναπαράσταση του πως κατανέμονται οι μέσες ταχύτητες μέσα στο χάρτη μετά από κάποιο χρόνο για τον οποίο έχουμε αφήσει την προσομοίωση να τρέξει. Έπειτα ο χρήστης μπορεί να ερμηνεύσει αυτήν την πληροφορία σύμφωνα με τις δικές του σχεδιαστικές απαιτήσεις και να προσαρμόσει τον σχεδιασμό του οδικού συστήματος αναλόγως.

Η λογική που ακολουθείται εδώ ξεκινά με την εγκατάσταση σημείων μέτρησης μέσα στο χάρτη σε στρατηγικά τοποθετημένες θέσεις. Τα σημεία αυτά παίρνουν μετρήσεις σε τακτά χρονικά διαστήματα από όλα τα οχήματα που περνούν πάνω από αυτά ή κι αρκετά κοντά από αυτά, κι ενημερώνουν ένα σύνολο δεδομένων που έπειτα αξιοποιείται για την οπτικοποίηση.

Η αρχή γίνεται στην Start A.109 συνάρτηση του state με τις συναρτήσεις που δεν αναλύσαμε πριν. Η SetupVisualsLayer αφορά στην οπτικοποίηση και θα τη δούμε στην επόμενη παράγραφο.

Η SetupMeasuringPoints αρχικοποιεί τη λίστα με τα σημεία μέτρησης. Το κάθε σημείο μέτρησης είναι μια κλάση που περιλαμβάνει πληροφορίες για τη θέση του σημείου, την ακτίνα μέτρησής του (δηλαδή την ακτίνα μέσα στην οποία επηρεάζεται από οχήματα που περνούν), το άθροισμα όλων των τιμών καθώς και το πλήθος τους ώστε να μπορούμε να πάρουμε τον αριθμητικό μέσο όρο A.110.

Για να δημιουργήσουμε τα σημεία μέτρησης, παίρνουμε με τη σειρά όλες τις λωρίδες του χάρτη A.111 και υπολογίζουμε το μέσο σημείο της λωρίδας με τον ίδιο ακριβώς τρόπο που είχαμε κάνει και τον υπολογισμό της τροχιάς των αυτοκινήτων. Έχοντας τα μέσα σημεία

και στα τέσσερα σημεία ελέγχου των δυο καμπυλών, το μεσαίο σημείο βρίσκεται στο μέσο της καμπύλης που προκύπτει από τις μέσες συντεταγμένες, δηλαδή για τιμή της παραμέτρου $t = 0.5$ A.112. Τότε χρησιμοποιούμε αυτό το μεσαίο σημείο της λωρίδας για να κάνουμε προσέγγιση δυο τμημάτων και να βρούμε ως γνωστόν το προσεγγιστικό μήκος της λωρίδας, το οποίο αν το διαιρέσουμε με την επιθυμητή απόσταση των σημείων μέτρησης (1 μέτρο) παίρνουμε το πλήθος των σημείων μέτρησης της συγκεκριμένης λωρίδας A.113. Μέσω του CheckpointGenerator παράγουμε τη λίστα των θέσεων των σημείων μέτρησης A.114.

Για να υπολογίσουμε την ακτίνα επιρροής των σημείων μέτρησης, μπορούμε να βρούμε το πλάτος της λωρίδας ως την απόσταση των σημείων P_0 των καμπυλών της και να θέσουμε αυτό ως διάμετρο του κύκλου μέτρησης. Προφανώς αυτό δεν είναι απόλυτα σωστό μιας και το πλάτος της λωρίδας μπορεί να μην είναι σταθερό για όλο το μήκος της, κι ίσως θα ήταν καλύτερα να υπολογίζουμε το πλάτος κατά μήκος της κάθε φορά χρησιμοποιώντας την παράμετρο t, ωστόσο είναι μια ικανοποιητική προσέγγιση A.115. Κι οπότε είμαστε έτοιμοι να δημιουργήσουμε τα σημεία για κάθε μία από τις υπολογισμένες θέσεις και με ακτίνα το μισό πλάτος της λωρίδας A.116.

Η συγκέντρωση των μετρήσεων τώρα γίνεται από κάθε όχημα ξεχωριστά από το Move component του στην συνάρτηση CheckForMeasuringPoints, στην συνάρτηση FixedUpdate κάθε επανάληψη μαζί με τις υπόλοιπες λειτουργίες. Η συνάρτηση είναι απλή και ελέγχει το state για το αν υπάρχουν σημεία ελέγχου κοντά που απέχουν λιγότερο από την ακτίνα που τους έχουμε θέσει A.117. Αν υπάρχει κάποιο, τότε παίρνουμε μέτρηση προσθέτοντας την ταχύτητα του οχήματος στο άθροισμα των τιμών κι αυξάνουμε τον μετρητή του πλήθους των μετρήσεων.

Συνεπώς έχουμε όλα τα δεδομένα που χρειαζόμαστε, δηλαδή μια λίστα με σημεία μέτρησης και τις συγκεντρωτικές μετρήσεις του, και μπορούμε να προχωρήσουμε στην οπτικοποίηση.

5.2 Οπτικοποίηση στατιστικών

5.2.1 Πέρασμα των δεδομένων στη GPU

Τα δεδομένα που έχουμε ήδη υπολογίσει βρίσκονται σε λίστες στον κώδικα των scripts, αυτό σημαίνει πως βρίσκονται στη "μεριά" της CPU, δηλαδή στη μνήμη RAM. Λόγω της χρήσης της μηχανής Unity, μπορεί αρκετές φορές να συγχέονται οι έννοιες CPU και GPU μιας και ήδη έχουμε υλοποιήσει τριδιάστατα γραφικά χωρίς να γίνει προγραμματισμός της GPU ευθέως, ωστόσο έχει σημασία να ξέρουμε πότε κώδικας της Unity έχει πρόσβαση σε αυτήν. Για παράδειγμα, όταν αλλάξαμε χρώμα στα αυτοκίνητα για να υποδεικνύουμε αυτό που έχει επιλεγεί από το χρήστη, έμμεσα χρησιμοποιήσαμε τη GPU μιας και πειράξαμε το χρώμα του υλικού (material) που χρησιμοποιεί το μοντέλο. Η Unity μετά μέσω αυτού του material ζωγραφίζει το αντικείμενο αξιοποιώντας την GPU εσωτερικά.

Το σκεπτικό είναι πως θα δημιουργήσουμε μια ακόμα υφή (texture) για το material του αντικειμένου που είχαμε δημιουργήσει για το δάπεδο κατά τη φόρτωση του κόσμου. Υπάρχει φυσικά ήδη το texture της εικόνας που είχαμε εισάγει κατά τη σχεδίαση και το οποίο φορτώνεται μαζί με το δάπεδο, κι όπως είδαμε το ορίζουμε ως κεντρικό texture στο material του αντικειμένου κατά την εκτέλεση της LoadAndRenderGround. Με την επιπλέον υφή όμως μπορούμε να κάνουμε πράξεις με τις οποίες θα αλλάζουμε το χρώμα του δαπέδου, χρησιμοποιώντας τις τιμές των στατιστικών που έχουμε υπολογίσει.

Η συνάρτηση που δημιουργεί αυτό το νέο texture είναι η SetupVisualsLayer A.118. Το statTexSize το έχουμε ορίσει ως 256 pixel κάθετα και οριζόντια. Μεγαλύτερο μέγεθος προσφέρει μεγαλύτερη ακρίβεια στην οπτικοποίηση εφόσον τα pixel αντιστοιχούν σε μικρότερα μεγέθη στον πραγματικό κόσμο, ωστόσο αυξάνουν κατά πολύ τις πράξεις που πρέπει να γίνουν και την επικοινωνία με την GPU, κάτι που είναι πολύ κοστοβόρο σε επίπεδο υλικού και καλό είναι να αποφεύγεται [43]. Έπειτα αρχικοποιούμε τις τιμές αυτού του texture σε 0 A.119, και τις εφαρμόζουμε στο texture A.120. Θέτουμε το αντίστοιχο texture μέσα στο υλικό σε αυτό που μόλις δημιουργήσαμε A.121 και δημιουργούμε και μια λίστα από χρώματα τα οποία θα περνάμε μέσα στο υλικό σε κάθε επανάληψη. Το μέγεθος της λίστας θα είναι η επιφάνεια του texture, δηλαδή το μήκος επί το πλάτος της A.122.

Κατά τη διάρκεια της επανάληψης, εκτελούμε την UpdateStatsTex που ανανεώνει τα χρώματα σύμφωνα με τις μετρήσεις που έχουμε πάρει. Άρα για κάθε σημείο μέτρησης υπολογίζουμε το δείκτη του σημείου που αντιστοιχεί στο texture γι αυτές τις συντεταγμένες του κόσμου A.123. Δηλαδή υπολογίζουμε την κανονικοποιημένη θέση, διαιρώντας τη θέση του κόσμου με το μέγεθος του χάρτη. Η μετατόπιση κατά μισό χάρτη γίνεται διότι ο χάρτης τοποθείται αρχικά στο $[0, 0, 0]$ γύρω από το κέντρο του, συνεπώς για να είναι οι συντεταγμένες ενός σημείου μέτρησης στο ίδιο σύστημα αναφοράς, μετατοπίζονται κατά μισό μήκος και μισό πλάτος χάρτη. Επίσης οι συντεταγμένες υφής (texture coordinates) στη Unity δίνονται κανονικοποιημένες στο εύρος $[0, 1]$, συνεπώς με ένα πολλαπλασιασμό με το μέγεθος του texture μπορούμε να πάρουμε το δείκτη του texel (texture element)[44].

Υπολογίζουμε το μέσο όρο των μετρήσεων A.124 και δημιουργούμε και ένα χρώμα, με το πράσινο και το μπλε περιεχόμενο κενά, και το κόκκινο περιεχόμενο να είναι η αναλογική τιμή της μέσης ταχύτητας του σημείου μέτρησης σε σχέση με το καθολικό όριο ταχύτητας A.125. Μετά βάζουμε αυτά τα χρώματα στον πίνακα των χρωμάτων. Για να είναι πιο εμφανή τα πίξελ, δεν χρωματίζονται ένα-ένα αλλά σε ένα παράθυρο 3x3 γύρω από τους δείκτες που υπολογίσαμε A.126. Εφόσον ο πίνακας δεν είναι διδιάστατος, για να προσπελάσουμε στοιχεία με διδιάστατους δείκτες (όπως εδώ τους idxX και idxY) υπολογίζουμε τον γραμμικό δείκτη (linear index) που είναι li = i+j*sx, όπου li ο γραμμικός δείκτης, i και j οι δείκτες της κάθε διάστασης και sx το μέγεθος της i διάστασης.

Μετά δεν έχουμε παρά να γεμίσουμε το texture με τα χρώματα που υπολογίσαμε μέσω της συνάρτησης SetPixels32 και να εφαρμόσουμε τις αλλαγές, και τέλος να ανανεώσουμε το texture του material με το νέο που μόλις παράξαμε A.127.

Το μόνο που μένει πλέον είναι να δούμε πως γίνεται ακριβώς ο χρωματισμός μέσα στον σκιαστή (shader) και το τελικό αποτέλεσμα που προκύπτει.

5.2.2 Οπτικοποίηση με shaders

Έχουμε ήδη αναφέρει την έννοια των σκιαστών (shaders) που πρακτικώς είναι προγράμματα που τρέχουν στον επεξεργαστή γραφικών αντί για τον κύριο επεξεργαστή. Τα προγράμματα αυτά γράφονται σε άλλες γλώσσες που λέγονται shading languages. Στη Unity η γλώσσα λέγεται HLSL (High Level Shading Language) [45] και συντακτικά ανήκει στην οικογένεια της C. Πολλά από τα στοιχεία σύνταξης της γλώσσας που χρησιμοποιείται συγκεκριμένα στη Unity παραπέμπουν στην Cg, μια άλλη γλώσσα που χρησιμοποιούσε παλιότερα ως γλώσσα shading.

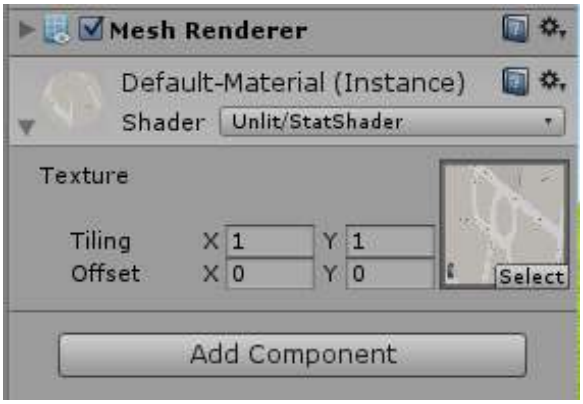
Η ακριβής λειτουργία των shaders και ο ρόλος τους στην επιστήμη των γραφικών είναι μεγάλο κεφάλαιο που δε θα αναλύσουμε στα πλαίσια αυτής της εργασίας, ωστόσο σημαντικό είναι το γεγονός πως αποτελούν βασικό κομμάτι της σύγχρονης σωλήνωσης γραφικών (graphics pipeline) [46]. Τα δυο βασικά σημεία που εξυπηρετούνται από shaders είναι η επεξεργασία σημείων και η επεξεργασία εικονοστοιχείων (vertex και pixel/fragment processing). Το πρώτο, το οποίο δε θα μας απασχολήσει, αφορά στην επεξεργασία και μετασχηματισμό δεδομένων που προέρχονται από γεωμετρίες αντικειμένων, με την είσοδο να είναι οι συντεταγμένες των σημείων στον διδιάστατο ή τριδιάστατο χώρο. Η έξοδος, δηλαδή μετά από όλους τους μετασχηματισμούς, είναι οι συντεταγμένες του σημείου αυτού σε συντεταγμένες οθόνης. Για παράδειγμα, ένα αντικείμενο που μέσα στο χώρο βρίσκεται στο $[23, 45, 2]$, μετά την προβολή του στην οθόνη μέσω της κάμερας μπορεί να καταλήξει στο pixel με συντεταγμένες $[960, 540]$, δηλαδη στο κέντρο της οθόνης που έχει ανάλυση Full HD ($1920x1080$). Αυτή η πληροφορία αρκεί για να δούμε το αντικείμενο στη σωστή θέση του, ωστόσο δεν αρκεί για να δούμε το σωστό χρώμα του.

Αυτό είναι δουλειά του επόμενου βήματος του graphics pipeline που λέγεται pixel/fragment processing και το αναλαμβάνει ο pixel/fragment shader. Η είσοδός του είναι οι συντεταγμένες της οθόνης που παρήγαγε πριν ο vertex shader, και με τη βοήθεια παράπλευρων πληροφοριών που του παρέχονται (όπως την κατεύθυνση και τη θέση του φωτός) χρωματίζει το συγκεκριμένο pixel αναλόγως.

Εδώ θα χρησιμοποιήσουμε αυτήν την ικανότητα που έχει ο fragment shader στο να χρωματίζει αντικείμενα, ώστε να χρωματίσουμε το κύριο texture του επιπέδου που αποτελεί τον δρόμο/δάπεδο.

Οι παράπλευρες πληροφορίες που αναφέραμε, στην προκειμένη θα είναι το επιπλέον texture που δημιουργήσαμε και χρωματίσαμε στην προηγούμενη παράγραφο.

Για αρχή δημιουργούμε μέσω της Unity ένα νέο shader StatShader.shader που είναι αντίγραφο του αρχικού shader που έχει η Unity (”Unlit/Texture”) για να ζωγραφίζει σκέτα texture χωρίς πληροφορίες φωτισμού, και τον βάζουμε στο μοντέλο του επιπέδου:



Σχήμα 48: Το material του δαπέδου με τον νέο shader της οπτικοποίσης

Προσθέτουμε μια νέα ιδιότητα στο shader [47] που είναι το νέο texture που θα συμψηφίζεται με το βασικό texture του material ώστε να προκύπτει το τελικό χρώμα A.128 όπου βλέπουμε και το κύριο texture. Οι ιδιότητες αυτές είναι τύπου sampler2D που είναι το είδος μεταβλητής που χρησιμοποιείται για δειγματοληψία (sampling) από texture. Μετά βλέπουμε τον vertex shader να κάνει τους μετασχηματισμούς που πρέπει για να έρθει το αντικείμενο σε συντεταγμένες οθόνης A.129.

Έπειτα προχωρούμε στον fragment shader. Ξεκινάμε δειγματοληπτώντας και από τα δυο texture A.130. Το fixed4 είναι τύπος διανύσματος τεσσάρων διαστάσεων που περιέχει αριθμούς σταθερής υποδιαστολής (fixed point) [48] και χρησιμοποιείται για το χρώμα που προκύπτει από την δειγματοληψία του texture. Αυτό διότι το χρώμα έχει τέσσερα στοιχεία: το κόκκινο, το πράσινο, το μπλε και τη διαφάνεια.

Θα δημιουργήσουμε μια κλίμακα χρωμάτων από το πράσινο έως το κόκκινο. Για τον υπολογισμό του τελικού χρώματος, ορίζουμε μια μεταβλητή col_rg τύπου fixed4. Στο στοιχείο

της διαφάνειας βάζουμε απλά την αρχική του δευτερεύοντος texture. Στο στοιχείο του μπλε βάζουμε 0 καθώς η κλίμακά μας δε θα περιέχει μπλε χρωματισμό.

Στο πράσινο θα βάλουμε το κόκκινο στοιχείο του δευτερεύοντος texture. Να σημειωθεί πως το κόκκινο στοιχείο του δευτερεύοντος texture δεν αφορά σε χρώμα αλλά στην μέτρηση της ταχύτητας που είχαμε πριν υπολογίσει και περάσει στο texture. Δηλαδή απλώς έτυχε να μεταφέρουμε αυτό το νούμερο μέσω του κόκκινου στοιχείου, ενώ θα μπορούσαμε με οποιοδήποτε από τα άλλα τρία. Το πράσινο άρα θα είναι 0 για μικρές τιμές της μέσης ταχύτητας και 1 για μεγάλες τιμές.

Αντίθετα, στο κόκκινο στοιχείο του υπολογιζόμενου χρώματος βάζουμε την συμπληρωματική, ως προς το 1.0, τιμή σε σχέση με το πράσινο. Αυτό σημαίνει ότι για το κόκκινο θα είναι 1 για μικρές τιμές της ταχύτητας και 0 για μεγάλες τιμές. Έτσι για τιμές ταχύτητας κοντά στο 0.5 το αποτέλεσμα θα είναι κίτρινο, αφού και το κόκκινο και το πράσινο θα είναι στο 0.5. Υπενθυμίζεται ότι η ταχύτητα διαιρείται με το όριο ταχύτητας πριν περαστεί στο texture ώστε να είναι στο εύρος $[0, 1]$.

Οι πράξεις αυτές αποτυπώνονται στον κώδικα A.131. Για να επιστρέψουμε το τελικό χρώμα, αναμειγνύουμε το αρχικό χρώμα του texture του δρόμου, ώστε να μη χαθεί αυτή η πληροφορία, με το χρώμα που μόλις υπολογίσαμε A.132.

Για να δούμε αυτήν την οπτικοποίηση κατά τη διάρκεια της προσομοίωσης, πατάμε στο "Switch View" της γραφικής διεπαφής. Τότε καλείται η συνάρτηση SwitchViewButtonClick A.133. Αυτή θα αλλάξει την κάμερα σε ορθογραφική λειτουργία[49], και θα την τοποθετήσει κατακόρυφα ώστε να βλέπει εποπτικά το χάρτη της προσομοίωσης. Αν αυτή η ιδιότητα είναι false σημαίνει πως εισαγόμαστε τώρα σε αυτήν τη λειτουργία.

Σε μια μεταβλητή preservedCameraPosition φροντίζουμε να κρατήσουμε τη θέση της κάμερας με την οποία πλοηγούμαστε μέσα στο χώρο. Αντίστοιχα στην μεταβλητή preservedCameraRotation κρατάμε την περιστροφή/κατεύθυνση. Κάνουμε έπειτα την κάμερα ορθογραφική και ορίζουμε άλλες παραμέτρους της κάμερας και τέλος βάζουμε το αντίστοιχο texture και material στο

μοντέλο του επιπέδου για να χρησιμοποιήσουμε όλα τα προηγούμενα που αναλύσαμε περί χρωμάτων Α.134.

Στην αντίθετη περίπτωση, δηλαδή που η κάμερα ήταν ήδη σε ορθογραφική λειτουργία, επαναφέρουμε τη θέση και την περιστροφή της ανακτώντας τις τιμές από τις αντίστοιχες μεταβλητές, απενεργοποιούμε την ορθογραφική λειτουργία, και επαναφέρουμε το material του επιπέδου στο να χρησιμοποιεί τον Standard shader της Unity ώστε να μην υφίσταται τους χρωματισμούς που υπολογίσαμε Α.135.

Αν αφήσουμε λοιπόν την προσομοίωση να τρέξει λίγη ώρα και πατήσουμε το κουμπί εναλλαγής σε λειτουργία οπτικοποίησης, παίρνουμε το παρακάτω:

Σχήμα 49: Παράδειγμα οπτικοποίησης των αποτελεσμάτων της προσομοίωσης

Μπορούμε να βγάλουμε κάποια συμπεράσματα μέσω των χρωμάτων που είναι αρκετά λογικά. Για παράδειγμα, βλέπουμε κόκκινες τις θέσεις στις οποίες δημιουργούνται τα αυτοκίνητα πάνω δεξιά, και σταδιακά όσο επιταχύνουν αυτό γίνεται κίτρινο και τελικά πράσινο. Επίσης, τα οχήματα που βρίσκονται μέσα στον κόμβο φαίνεται πως θα έχουν μια μέτρια ταχύτητα, ενώ αυτό που πάει να βγει από τον κόμβο στα δεξιά έχει δημιουργήσει μια πιο κόκκινη περιοχή καθώς επιβραδύνει για να παραχωρήσει προτεραιότητα στα οχήματα που κινούνται στον κυρίως δρόμο. Τέλος, όπως φαίνεται από το όχημα κάτω αριστερά, τα οχήματα που θα βγουν από τον

κόμβο σε εκείνο το σημείο αναμένουμε να έχουν μια αυξανόμενη ταχύτητα, όπως φαίνεται από τη μετάβαση από κίτρινο σε πράσινο χρώμα που δημιουργεί μια ζώνη επιτάχυνσης.

Η ακριβής ερμηνεία των αποτελεσμάτων αφήνεται στο χρήστη, ώστε να προσαρμόσει το σχεδιασμό του σύμφωνα με τις δικές του απαιτήσεις.

6 Συμπεράσματα

Το παρόν λοιπόν αποτελεί ένα λογισμικό για σχεδιασμό και προσομοίωση οδικών συστημάτων με πλήρη ελευθερία στις γεωμετρίες των δρόμων, των λωρίδων και των καμπυλών που τις ορίζουν, πάντα λαμβάνοντας υπ' όψη τη φυσική των οχημάτων. Οι χρήστες μπορούν να αποθηκεύουν τα οδικά συστήματα που σχεδιάζουν σε αρχεία χάρτη, τα οποία μπορούν να διαμοιράζονται με άλλους χρήστες του λογισμικού. Η προσομοίωση του οδικού συστήματος μπορεί να εκτελεσθεί σε ένα οποιοδήποτε συμβατό αρχείο χάρτη που έχει παραχθεί από το λογισμικό ανεξαρτήτως πλατφόρμας. Χρησιμοποιώντας τα εργαλεία της οπτικοποίησης αποτελεσμάτων, ο χρήστης τελικά αποτιμά το οδικό σύστημα που σχεδίασε και προσαρμόζει το σχέδιο αναλόγως.

Προφανώς υπάρχουν πολλές δυνατότητες για βελτίωση υπαρχόντων λειτουργιών σε θέματα σφαλμάτων ή αποδοτικής λειτουργίας, σε θέματα εμπειρίας του χρήστη αλλά και περιθώρια για προσθήκη νέων λειτουργιών και χαρακτηριστικών που κάνουν το λογισμικό πιο χρήσιμο και πιο χρηστικό, και τα αποτελέσματα που παράγει μεγαλύτερης αξίας.

6.1 Ελλείψεις και μελλοντικές βελτιώσεις

6.1.1 Βελτιστοποίηση αναζήτησης

Αυτή τη στιγμή, όπως είδαμε ο έλεγχος για άλλα οχήματα γίνεται σε μια επανάληψη που ελέγχει κάθε όχημα με κάθε άλλο του κόσμου. Αυτό είναι υποχρεωτικό, καθώς ο έλεγχος της προτεραιότητας του A στον B δε δίνει πληροφορία για το ανάποδο, συνεπώς πρέπει να γίνει

κι ο αντίστροφος έλεγχος. Αν θεωρήσουμε πως τα οχήματα είναι n στο πλήθος, τότε οι έλεγχοι που θα γίνουν ειναι $n(n-1)$ ή $n^2 - n$.

Να εισάγουμε εδώ εν συντομία την έννοια της ασυμπτωτικής πολυπλοκότητας (asymptotic complexity) που χαρακτηρίζει αλγορίθμους με τη σχέση των απαιτήσεων σε επεξεργασία και μνήμη σε σχέση με το μέγεθος της εισόδου. Αυτό γίνεται μέσω του Big O notation [50] που δίνει μαθηματικά αυτή τη σχέση.

Στην περίπτωσή μας, όπως είπαμε οι έλεγχοι που θα γίνουν θα είναι στο πλήθος n^2-n. Στο Big O notation σημασία έχει ο κυρίαρχος όρος μιας έκφρασης, άρα ο αλγόριθμος θα είναι $O(n^2)$. Η τετραγωνική πολυπλοκότητα είναι καλό στη γενική περίπτωση να αποφεύγεται, καθώς ακόμα και για μόνο 100 οχήματα, θα είχαμε ως αποτέλεσμα 10,000 ελέγχους, ενώ όσο μεγαλώνει το πλήθος των οχημάτων ο αριθμός των ελέγχων θα μεγάλωνε ακόμα περισσότερο.

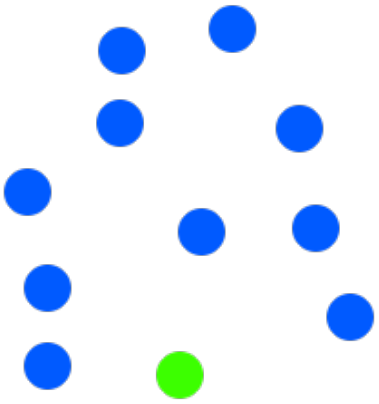
Ο έλεγχος γειτονικών αντικειμένων είναι κλασικό πρόβλημα της επιστήμης των υπολογιστών[51], με πολλές τεχνικές, όπως την χωρική διάτμηση (spatial partitioning)[52] να βελτιστοποιούν την αναζήτηση αναλόγως με τις εκάστοτε απαιτήσεις και το είδος του προβλήματος

Μια προσέγγιση που θα μπορούσαμε να έχουμε εδώ για μείωση της ασυμπτωτικής πολυπλοκότητας του προβλήματος, είναι ο χωρισμός του επιπέδου σε ορθογώνια κελιά, δηλαδή δημιουργία ενός πλέγματος (grid). Κάθε όχημα που βρίσκεται στο χάρτη ανά πάσα στιγμή θα βρίσκεται σε ένα συγκεκριμένο κελί. Ο έλεγχος θα περιορίζεται μόνο στα οχήματα που περιέχονται στο ίδιο κελί ή και στα γειτονικά. Εφόσον πρακτικά από φυσικής άποψης ο αριθμός των οχημάτων που θα χωρέσουν σε ένα κελί είναι περιορισμένος (έστω k) τότε ελέγχοντας για γειτονικά αυτοκίνητα θα ελέγξουμε το πολύ k φορές. Αυτό δεν αλλάζει όσο μεγάλος κι αν είναι ο αριθμός των οχημάτων, το οποίο ήδη μας προϊδεάζει πως ο έλεγχος θα πάψει να είναι $O(n^2)$. Συγκεκριμένα, εφόσον οι έλεγχοι γίνονται για κάθε αυτοκίνητο από τα n, και για κάθε αυτοκίνητο θα κάνουμε στη χειρότερη περίπτωση k ελέγχους, τότε τελικά ο αλγόριθμος θα είναι $O(kn)$ στην χειρότερη περίπτωση.

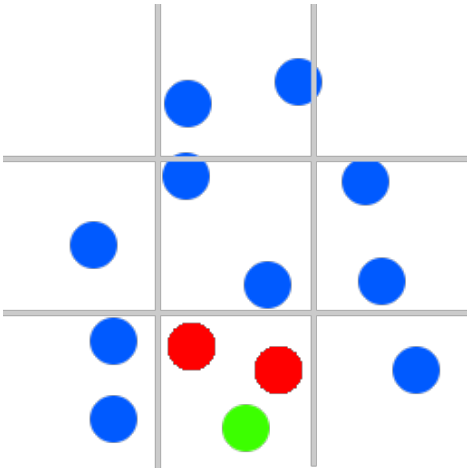
Ο αριθμός k των μέγιστων αυτοκινήτων ανά κελί είναι σταθερά, άρα ο αλγόριθμος από τετραγωνικός έγινε γραμμικός, όσο μεγάλη κι αν είναι αυτή η σταθερά. Το Big O δε σημαίνει πως αλγόριθμος μεγαλύτερης βαθμίδας θα είναι σίγουρα χειρότερος από αλγόριθμο χαμηλότερης,

αλλά μας περιγράφει τη συμπεριφορά σε όλο και αυξανόμενη είσοδο. Αυτό σημαίνει πως για μικρό αριθμό αυτοκινήτων, θα μπορούσε ίσως ο τετραγωνικός αλγόριθμος να είναι πιο γρήγορος εξαιτίας άλλων παραμέτρων.

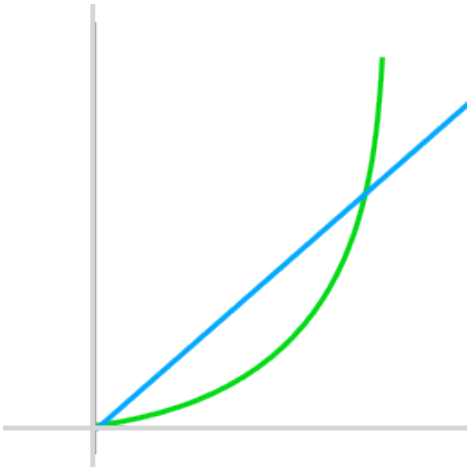
Εποπτικά τα παραπάνω φαίνονται στις παρακάτω εικόνες:



Σχήμα 50: Απλός (brute-force) έλεγχος γειτόνων με κάθε αντικείμενο



Σχήμα 51: Βελτιστοποίηση αναζήτησης με χρήση πλέγματος (grid)



Σχήμα 52: Σύγκριση τετραγωνικής και γραμμικής αλγοριθμικής πολυπλοκότητας

6.1.2 Προσθήκη υψομέτρου στη σχεδίαση

Όπως είδαμε, στην εργασία αυτή δεν χρησιμοποιούμε την y συντεταγμένη, κι όλες οι πράξεις γίνονται στο επίπεδο κατασκευάζοντας διδιάστατα διανύσματα μέσω των x και z.

Στα πλαίσια μιας προσομοίωσης σε έναν επίπεδο κόμβο ή σε μια διασταύρωση αυτό δεν έχει καμία επίπτωση, ωστόσο μας εμποδίζει από το να σχεδιάσουμε ανισόπεδους κόμβους, ράμπες εισόδου σε αυτοκινητόδρομους, ανηφόρες και άλλες μορφολογικές λεπτομέρειες που επηρεάζουν την φυσική συμπεριφορά των οχημάτων αλλά και τον χειρισμό τους από τους οδηγούς.

Ο σχεδιασμός λαμβάνοντας υπ' όψη το ύψος δεν έχει υλοποιηθεί σε κανένα επίπεδο στο εργαλείο σχεδιασμού, αφού θεωρούμε πως ο κόσμος είναι επίπεδος και η όψη σχεδιασμού είναι κάτοψη χωρίς δυνατότητα αλλαγής. Επίσης δεν μπορούμε να σχεδιάσουμε επικαλυπτόμενους δρόμους όπου ο ένας περνά πάνω από τον άλλον. Ακόμα, οι έλεγχοι για προτεραιότητα και επιβράδυνση γίνονται μόνο στο επίπεδο $y = 0$, άρα ακόμα κι αν ο σχεδιασμός ήταν δυνατός, δεν θα λαμβανόταν υπ' όψη από την προσομοίωση.

Ωστόσο, λόγω της αναλυτικής προσομοίωσης της φυσικής των οχημάτων, το κομμάτι του χειρισμού του οχήματος για επίτευξη συγκεκριμένης ταχύτητας και ακολούθησης τροχιάς θα δούλευε με ελάχιστες αλλαγές, αφού κρίνονται από τα φυσικά αποτελέσματα. Για παράδειγμα, αν το όχημα οδηγούνταν σε μια ανηφόρα ή κατηφόρα, δε θα χρειαζόταν κάποια αλλαγή για τη διατήρηση της ταχύτητάς του αφού οι συναρτήσεις χειρισμού των πεταλιών απλά θα εφάρμοζαν τιμές μεγαλύτερου γκαζιού ή και φρένου αντίστοιχα. Επίσης στις περιπτώσεις των στροφών, ο ελεγκτής που υλοποιήθηκε απλά θα διόρθωνε πιο επιθετικά την σύγκλιση προς την τροχιά λόγω των μεγαλύτερων σφαλμάτων.

6.1.3 Προσθήκη σήμανσης

Αυτή τη στιγμή το μόνο κομμάτι του Κώδικα Οδικής Κυκλοφορίας που μοντελοποιείται είναι η παραχώρηση προτεραιότητας.

Θα μπορούσαμε λοιπόν να βάλουμε επιπλέον παραμέτρους στην προσομοίωση που αφορούν στη σήμανση, όπως για παράδειγμα σήματα STOP, σήματα παραχώρησης προτεραιότητας, σηματοδότες και πιο περίπλοκα σημεία του Κ.Ο.Κ.

Κάποιες από αυτές τις αλλαγές θα ήταν απλές, όπως τα STOP και οι σηματοδότες, αφού τελικά θα καταλήγουν στο Behaviour.stop που ήδη έχει υλοποιηθεί, αλλά θα έπρεπε να μπουν επιπλέον συνθήκες για την κάλυψη αυτών των περιπτώσεων. Για παράδειγμα στην περίπτωση του STOP, θα αγνοούμε οτιδήποτε έχει να κάνει με προτεραιότητα, και θα δίνουμε Behaviour.stop όταν το όχημα πλησιάζει στο τέλος λωρίδας που έχει σημανθεί με την ιδιότητα του STOP.

6.1.4 Προσθήκη επιπλέον στατιστικών

Το μόνο στατιστικό που υπολογίζεται σε αυτήν την εργασία είναι η μέση ταχύτητα σε διάφορα σημεία του χάρτη. Ωστόσο υπάρχουν δεκάδες άλλα στατιστικά που θα μπορούσαμε να συλλέξουμε, με κάποια από αυτά να είναι ήδη έτοιμα.

Για παράδειγμα, όπως περνάμε στις συναρτήσεις της οπτικοποίησης τις πληροφορίες της ταχύτητας, θα μπορούσαμε πολύ απλά να τις αντικαταστήσουμε με τις πληροφορίες της κατανάλωσης που ήδη έχουμε υπολογισμένες. Έτσι, ο χρήστης θα αποτιμούσε το σχέδιό του με βάση την ενεργειακή του απόδοση, δηλαδή αν προκαλεί σε συγκεκριμένα μέρη του οδικού συστήματος μεγάλη κατανάλωση καυσίμου λόγω κακού σχεδιασμού που έχει ως αποτέλεσμα μη αποδοτική ροή κίνησης. Άλλο παράδειγμα είναι η μέτρηση της ροής των αυτοκινήτων σε απόλυτα νούμερα εισαχθέντων/εξαχθέντων οχημάτων από το χάρτη. Έχουμε πλήρη πρόσβαση στην πληροφορία δημιουργίας και καταστροφής οχημάτων, συνεπώς κάτι τέτοιο θα ήταν διαδικαστικό να υλοποιηθεί.

Βέβαια, άλλα εν δυνάμει στατιστικά δεν έχουν ήδη υπολογιστεί (είτε δεν έχουν υπολογιστεί άμεσα) και θα ήθελαν προσθήκη νέων ρουτινών σύλλεξης μετρήσεων και υπολογισμού. Το ποια είναι αυτά όμως θα το έκριναν οι απαιτήσεις του εκάστοτε χρήστη.

Σχήματα

Βιβλιογραφία

[1] https://gamefromscratch.com/gamedev-glossary-library-vs-framework-vs-engine/ accessed 2021-01-18

[2] https://unity.com/ accessed 2021-01-18

[3] https://developer.nvidia.com/gameworks-physx-overview accessed 2021-01-18

[4] https://en.wikipedia.org/wiki/C_Sharp_(programming_language) accessed 2021-01-18

[5] https://visualstudio.microsoft.com/ accessed 2021-01-18

[6] https://www.audacityteam.org accessed 2021-01-18

[7] https://www.reaper.fm accessed 2021-01-18

[8] https://www.gimp.org accessed 2021-01-18

[9] https://www.blender.org accessed 2021-01-18

[10] https://www.sciencedirect.com/science/article/abs/pii/S0096300309000137 accessed 2021-01-18

[11] http://traffic-flow-dynamics.org/ accessed 2021-01-18

[12] Traffic Flow Dynamics - Data, Models and Simulation - Treiber, Kesting (2013)

[13] https://www.traffic-simulation.de/ accessed 2021-01-18

[14] https://www.eclipse.org/sumo/ accessed 2021-01-18

[15] https://carla.org/ accessed 2021-01-18

[16] https://books.google.com/books?id=YmQy799flPkC&pg=PA264 accessed 2021-01-18

[17] https://www.theengineerspost.com/jet-propulsion/ accessed 2021-01-18

[18] https://www.egr.msu.edu/ lira/supp/steam/ accessed 2021-01-18

[19] https://www.britannica.com/biography/James-Watt#ref31592 accessed 2021-01-18

[20] https://www.grc.nasa.gov/WWW/K-12/airplane/powert.html accessed 2021-01-18

[21] Wapcaplet at English Wikipedia, CC BY-SA 3.0 <http://creativecommons.org/licenses/by-sa/3.0/>, via Wikimedia Commons

[22] https://commons.wikimedia.org/wiki/User:Borowski commonswiki accessed 2021-01-18

[23] http://users.tem.uoc.gr/ nchristakis/dissert_Zografakis.pdf accessed 2021-01-18

[24] http://old.kvm.tul.cz/studenti/texty/Engine_modeling_A_Keromnes.pdf accessed 2021-01-18

[25] https://vtechdyno.eu/theory-chassis-dynamometer.html accessed 2021-01-18

[26] John Dinkel, "Chassis Dynamometer", Road and Track Illustrated Automotive Dictionary, (Bentley Publishers, 2000) page 46

[27] https://www.caranddriver.com/news/a15347872/horsepower-vs-torque-whats-the-difference/ accessed 2021-01-18

[28] https://itstillruns.com/purpose-flywheel-car-6674453.html accessed 2021-01-18

[29] https://www.theengineerspost.com/what-is-clutch/ accessed 2021-01-18

[30] http://farside.ph.utexas.edu/teaching/336k/Newtonhtml/node64.html accessed 2021-01-18

[31] https://www.brainkart.com/article/Gear-Box–Principle-of-Gearing-and-Types-of-Gear-Boxes_5054/ accessed 2021-01-18

[32] https://web.mit.edu/2.972/www/reports/differential/differential.html accessed 2021-01-18

[33] https://www.matfoundrygroup.com/News%20and%20Blog/Types_of_Differential_and_How_They_Work accessed 2021-01-18

[34] https://docs.unity3d.com/Manual/class-WheelCollider.html accessed 2021-01-18

[35] https://www.grc.nasa.gov/WWW/k-12/airplane/drageq.html accessed 2021-01-18

[36] https://www.learncpp.com/cpp-tutorial/relational-operators-and-floating-point-comparisons/ accessed 2021-01-18

[37] https://docs.unity3d.com/ScriptReference/Vector2.Angle.html accessed 2021-01-18

[38] https://www.mathopenref.com/arctan.html accessed 2021-01-18

[39] https://fooplot.com/ accessed 2021-01-18

[40] https://www.ioas.gr/uploads/docs/2016/05/397.pdf accessed 2021-01-18

[41] https://cs.nyu.edu/exact/doc/subdiv1.pdf accessed 2021-01-18

[42] https://docs.unity3d.com/ScriptReference/Physics.Raycast.html accessed 2021-01-18

[43] https://developer.nvidia.com/gpugems/gpugems/part-v-performance-and-practicalities/chapter-28-graphics-pipeline-performance accessed 2021-01-18

[44] https://docs.unity3d.com/ScriptReference/RaycastHit-textureCoord.html accessed 2021-01-18

[45] https://docs.unity3d.com/Manual/SL-ShadingLanguage.html accessed 2021-01-18

[46] https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview accessed 2021-01-18

[47] https://docs.unity3d.com/Manual/SL-PropertiesInPrograms.html accessed 2021-01-18

[48] https://developer.download.nvidia.com/cg/Cg_language.html accessed 2021-01-18

[49] https://www.britannica.com/technology/orthographic-projection-engineering accessed 2021-01-18

[50] http://web.mit.edu/16.070/www/lecture/big_o.pdf accessed 2021-01-18

[51] Christer Ericson, Real-Time Collision Detection (2004)

[52] https://gameprogrammingpatterns.com/spatial-partition.html accessed 2021-01-18

A Κώδικας

A.1 Διάλογοι διαχείρισης αρχείων του .NET

```
using System;
using System.Windows.Forms;

namespace NETWindowsDialogs
{
    public static class Dialog
    {
        public static string OpenFileDialog(string filter)
        {
            OpenFileDialog myOpenDialog = new OpenFileDialog();
            myOpenDialog.Filter = filter;
```

```
            myOpenDialog.ShowDialog();
            return myOpenDialog.FileName;
        }
        public static string SaveFileDialog(string filter)
        {
            SaveFileDialog mySaveDialog = new SaveFileDialog();
            mySaveDialog.Filter = filter;
            mySaveDialog.ShowDialog();
            return mySaveDialog.FileName;
        }
    }
}
```

A.2 Χρήση του .NET DLL από τη Unity

```
using NETWindowsDialogs;
...
        public void onSimButtonClick(){
                mapFilename = Dialog.OpenFileDialog("MOTORS Map Files
                    (*.motors)|*.motors");
                if (mapFilename.Length==0) {
                        return;
                }
        }
```

A.3 Ορισμός καμπύλης Bezier

```
public class CubicBezier{
        public Vector3 p0;
        public Vector3 p1;
        public Vector3 p2;
        public Vector3 p3;
```

```
        public CubicBezier(Vector3 pt0, Vector3 pt1, Vector3 pt2,
            Vector3 pt3) {
                p0 = pt0;
                p1 = pt1;
                p2 = pt2;
                p3 = pt3;
        }
}
```

A.4 Συνάρτηση παραγωγής σημείων Bezier

```
public static List<Vector3> CubicBezierFromPoints(Vector3 p0, Vector3
    p1, Vector3 p2, Vector3 p3, float res)
{
    List<Vector3> output = new List<Vector3>();

    for (int i = 0; i < res; ++i)
    {
        Vector3 point = new Vector3(0, 0, 0);
        float t = i / res;
        point.x =
            (1-t)*(1-t)*(1-t)*p0.x+3*(1-t)*(1-t)*t*p1.x+3*(1-t)*t*t*p2.x+t*t*t*p3.x;
        point.y = p0.y;
        point.z =
            (1-t)*(1-t)*(1-t)*p0.z+3*(1-t)*(1-t)*t*p1.z+3*(1-t)*t*t*p2.z+t*t*t*p3.z;
        output.Add(point);
    }
    output.Add(p3);

    return output;
}
```

A.5 Επιλογή σημείου Bezier για συγκεκριμένη παράμετρο t

```
public static Vector3 BezierPointAtParameterVal(Vector3 p0, Vector3 p1,
    Vector3 p2, Vector3 p3, float t)
{
    Vector3 o = new Vector3();
    o.x =
        (1-t)*(1-t)*(1-t)*p0.x+3*(1-t)*(1-t)*t*p1.x+3*(1-t)*t*t*p2.x+t*t*t*p3.x;
    o.y = p0.y;
    o.z =
        (1-t)*(1-t)*(1-t)*p0.z+3*(1-t)*(1-t)*t*p1.z+3*(1-t)*t*t*p2.z+t*t*t*p3.z;

    return o;
}
```

A.6 Ορισμός ορίου λωρίδας

```
public class LaneBorder
{
    public GameObject laneBorderObject;
    public int laneBorderID = -1;

    public Vector3 pt0 = new Vector3(0, 0, 0);
    public Vector3 pt1 = new Vector3(0, 0, 0);
    public Vector3 pt2 = new Vector3(0, 0, 0);
    public Vector3 pt3 = new Vector3(0, 0, 0);

    public LaneBorder(int id)
    {
        this.laneBorderID = id;
    }
```

```
    public LaneBorder(GameObject go)
    {
        this.laneBorderObject = go;
    }
}
```

A.7 Ορισμός λωρίδας

```
public class Lane
{
    public int laneID = -1;

    public int leftBorderID = 0;
    public bool invertLeftBorder = false;
    public LaneBorderType leftBorderType = LaneBorderType.None;
    public int rightBorderID = 0;
    public bool invertRightBorder = false;
    public LaneBorderType rightBorderType = LaneBorderType.None;

    public LaneMode mode = 0;

    public LaneIndicator indicator;
    public Vector3 indicPos;

    public bool correctlyBound = true;

    //indices to the borders array instead of IDs. Used only in the sim
    public int leftBorderIdx;
    public int rightBorderIdx;
    //list of indices to the lanes array that the lane connects to.
       Used only in sim
    public List<int> connsTo = new List<int>();
```

```
}
```

A.8 Ορισμός τύπου ορίου λωρίδας

```
public enum LaneBorderType
{
    None, // no border
    Boundary, // edge of driveable road
    Single, // single or double line between opposite directions
    Dashed, // single dashed line allowing overtake
    DashedSame, // single dashed line for same-direction lanes
    DashedSpeedCtrl, // dense dashed line for a lane entering a
        slower/faster road
    DashedWarn, // dense dashed line warning for a potential danger
    DoubleDashed // for same-direction lanes about to split or reverse
        directions
};
```

A.9 Ορισμός είδους λειτουργίας λωρίδας

```
public enum LaneMode
{
    None,
    Start,
    End
};
```

A.10 Ορισμός σύνδεσης λωρίδων

```
public class LaneConnection
{
    public int connectsFrom = 0;
    public int connectsTo = 0;
    public bool stopExists = false;
```

```
    public bool correctlyBound = true;

    public LaneConnection(int fro, int to, bool stop)
    {
        this.connectsFrom = fro;
        this.connectsTo = to;
        this.stopExists = stop;
    }
}
```

A.11 Σκηνή κυρίως μενού

```
public class MenuState : MonoBehaviour {
        public static int choice = -1;
        public static string mapFilename;

        public void onExitButtonClick(){
                Application.Quit();
        }
        public void onSimButtonClick(){
                mapFilename = Dialog.OpenFileDialog("MOTORS Map Files
                    (*.motors)|*.motors");
                if (mapFilename.Length==0) {return;}
                choice = 0;
                SceneManager.LoadScene("loading");
        }
        public void onEditorButtonClick(){
                choice = 1;
                SceneManager.LoadScene("loading");
        }
}
```

A.12 Φόρτωση σκηνής

```
public class LoadingAnimation : MonoBehaviour {
        public GameObject disc;

        void Start(){
                switch(MenuState.choice){
                case 0:
                        SceneManager.LoadSceneAsync("simScene");
                        break;
                case 1:
                        SceneManager.LoadSceneAsync("pathEditor");
                        break;
                }
        }

        void Update(){
                disc.transform.Rotate(0,0,1f);
        }
}
```

A.13 Εισαγωγή και δημιουργία φόντου

```
if (File.Exists(backgroundPath)){
        backImgData = File.ReadAllBytes(backgroundPath);
        background =  GameObject.Find("BackgroundPlane");
        if (background==null){
                background =
                    GameObject.CreatePrimitive(PrimitiveType.Plane);
        }
        tex = new Texture2D(2, 2, TextureFormat.BGRA32, false);
        tex.LoadImage(backImgData);
```

```
        background.GetComponent<Renderer>().material.mainTexture = tex;
        background.GetComponent<Renderer>().material.shader =
            Shader.Find("Unlit/Texture");
        setBackgroundScale(10.0f);
}
```

A.14 Έλεγχος αρτιότητας χάρτη

```
bool CheckMapIntegrity(){
        bool lanesAndConnsOk = true;
        bool startExists = false;
        bool endExists = false;

        for (int i=0; i<lanes.Count; ++i){
                if (!lanes[i].correctlyBound){lanesAndConnsOk = false;}
                if (lanes[i].mode == 1){startExists = true;}
                if (lanes[i].mode == 2){endExists = true;}
        }
        for (int i=0; i<connections.Count; ++i){
                if (!connections[i].correctlyBound){
                        lanesAndConnsOk = false;
                        break;
                }
        }

        if (!lanesAndConnsOk){
                errors.text = "<color=red>Could not save map because of
                    lane/connection errors. " +
                        "Check lane/connection management</color>";
        }
        if (!(startExists&&endExists)){
                errors.text = "<color=red>Could not save map. You need
```

```
                at least one Start and one End lane.</color>";
        }

        return (lanesAndConnsOk&&startExists&&endExists);
}
```

A.15 Αποθήκευση χάρτη

```
public void OnSaveMapButtonClick(){
        if (!CheckMapIntegrity()) return;

        String so=""; //string for output data
        String pathAndFilename = Dialog.SaveFileDialog("MOTORS Map
            Files (*.motors)|*.motors");
        if (pathAndFilename.Length==0) return;

        for (int i=0; i<curves.Count; ++i){
                GameObject go = curves[i].laneBorderObject;
                if (go!=null){
                        drawPath scr = go.GetComponent<drawPath>();
                        so+="p,";
                        so+=scr.currentID+",";
                        so+=scr.p0.x+","+scr.p0.y+","+scr.p0.z+",";
                        so+=scr.p1.x+","+scr.p1.y+","+scr.p1.z+",";
                        so+=scr.p2.x+","+scr.p2.y+","+scr.p2.z+",";
                        so+=scr.p3.x+","+scr.p3.y+","+scr.p3.z;
                        so+="\n";
                }
        }

        for (int i=0; i<lanes.Count; ++i){
                so+="l,";
```

```
            so+=lanes[i].laneID+",";
            so+=lanes[i].leftBorderID+",";
            so+=lanes[i].invertLeftBorder+",";
            so+=(int)lanes[i].leftBorderType+",";
            so+=lanes[i].rightBorderID+",";
            so+=lanes[i].invertRightBorder+",";
            so+=(int)lanes[i].rightBorderType+",";
            so+=lanes[i].mode;
            so+="\n";
}

for (int i=0; i<connections.Count; ++i){
            so+="c,";
            so+=connections[i].connectsFrom+",";
            so+=connections[i].connectsTo+",";
            so+=connections[i].stopExists;
            so+="\n";
}
so+="s,"+backScale+"\n";
so+="i\n";
File.WriteAllText(pathAndFilename, so);

//write background image data
BinaryWriter bw;
FileStream fsw = new FileStream(pathAndFilename,
    FileMode.Append);
bw = new BinaryWriter(fsw);
if (backImgData!=null){
            for (int i=0; i<backImgData.Length; ++i){
                        bw.Write(backImgData[i]);
```

```
                }
            }
            bw.Close();

            errors.text = "<color=#00ff00ff>Map saved</color>";
}
```

A.16 Ανάκτηση χάρτη

```
while((si=rd.ReadLine()) != null){
            string[] attribs = si.Split(',');
            if (attribs[0]=="i") {break;}
            switch (attribs[0]) {
            case "p":
                    LaneBorder newCurve = new LaneBorder(newLaneBorderObj)
                    curves.Add(newCurve);
                    break;
            case "l":
                    Lane newLane = new Lane(Int32.Parse(attribs[1]) ,...);
                    lanes.Add(newLane);
                    break;
            case "c":
                    LaneConnection newConnection = new LaneConnection(...);
                    connections.Add(newConnection);
                    break;
            case "s":
                    backScale=Single.Parse(attribs[1]);
                    break;
            default:
                    errors.text = "<color=red>Loading map failed. Corrupt
                        file</color>";
                    onResetYes();
```

```
                return;
        }
}
```

A.17 Ανάκτηση δεδομένων εικόνας

```
//read background image data
byte[] tempWholeFileData = File.ReadAllBytes(pathAndFilename);
backImgData = null;
BinaryReader br;
FileStream fsr = new FileStream(pathAndFilename, FileMode.Open);
br = new BinaryReader(fsr);
int mapDataByteCounter = 0;
byte currByteRead = br.ReadByte();
while(currByteRead!=0x69){
        mapDataByteCounter++;
        currByteRead = br.ReadByte();
}
br.ReadByte();
backImgData = br.ReadBytes(tempWholeFileData.Length -
    mapDataByteCounter);
tex = new Texture2D(2, 2, TextureFormat.BGRA32, false);
tex.LoadImage(backImgData);
if (background==null){
        background = GameObject.CreatePrimitive(PrimitiveType.Plane);
}
background.GetComponent<Renderer>().material.mainTexture = tex;
background.GetComponent<Renderer>().material.shader =
    Shader.Find("Unlit/Texture");
setBackgroundScale(backScale);
br.Close();
```

A.18 Δημιουργία καμπύλης ροπής

```
public Engine() {

        torqueCurveSamples();

        // interpolate remaining torque curve values
        int prevIdx = 0;
        for (int i = 0; i < (int)revUpLimit; ++i) {
                if (torqueCurve[i] > 0f) {
                        for (int j = prevIdx; j < i; ++j) {
                                torqueCurve[j] = torqueCurve[i]
                                    - (torqueCurve[i] -
                                    torqueCurve[prevIdx]) *
                                    (i-j) / (i-prevIdx);
                        }
                        prevIdx = i;
                }
        }
}
```

A.19 Δείγματα πραγματικής καμπύλης ροπής

```
void torqueCurveSamples(){
        torqueCurve = new float[(int)revUpLimit];
        // samples
        torqueCurve[0]    = 400f; // hack for engine startup
        torqueCurve[1000] = 87.5f;
        torqueCurve[1500] = 95.0f;
        torqueCurve[2000] = 104.0f;
        torqueCurve[2500] = 108.0f;
        torqueCurve[3000] = 110.0f;
```

```
            torqueCurve[3500] = 115.5f;
            torqueCurve[3750] = 114.5f;
            torqueCurve[4000] = 115.0f;
            torqueCurve[4500] = 114.0f;
            torqueCurve[4750] = 111.0f;
            torqueCurve[5000] = 108.0f;
            torqueCurve[5250] = 100.0f;
            torqueCurve[5500] = 88.5f;
            torqueCurve[6000] = 75.0f;
            torqueCurve[6499] = 62.5f;
        }
```

A.20 Υπολογισμός ροπής από τις στροφές ανά λεπτό

```
public float GetTorqueFromCurrentRpm() {
        float rpm = GetRPM();
        float limitedRpm = (rpm > revUpLimit) ? (revUpLimit - 1f) : rpm;
        limitedRpm = (limitedRpm < 0f) ? 0f : limitedRpm;

        return torqueCurve[(int)limitedRpm];
}
```

A.21 Η μεταβλητή για την ποσότητα σύμπλεξης

```
        public float clutch; // [0,1]
```

A.22 Ορισμός φυσικών παραμέτρων του δίσκου

```
engn.rb = engineObject.GetComponent<Rigidbody>();
engn.rb.constraints = RigidbodyConstraints.FreezePosition |
    RigidbodyConstraints.FreezeRotationX |
    RigidbodyConstraints.FreezeRotationZ;
engn.rb.mass = 20f;
```

```
engn.rb.moi = 0.2f; // flywheel's moment of inertia in kgm^2
// needs non-zero XYZ but we care only about Y
engn.rb.inertiaTensor = new Vector3(0.1234f, engn.moi, 0.1234f);
engn.rb.isKinematic = false;
engn.rb.useGravity = false;
engn.rb.angularDrag = 0.3f; // experimental good value
```

A.23 Οι μεταβλητές για τις σχέσεις μετάδοσης

```
public float[] gearRatios = {0f, 3.46f, 1.96f, 1.39f, 1.03f, 0.85f };
public float fd = 4.19f; // final drive
```

A.24 Ορισμός φυσικών παραμέτρων του κιβωτίου

```
grbx.rb = gearboxObject.GetComponent<Rigidbody>();
grbx.rb.constraints = RigidbodyConstraints.FreezePosition |
    RigidbodyConstraints.FreezeRotationX |
    RigidbodyConstraints.FreezeRotationZ;
grbx.rb.mass = 2.5f;
grbx.rb.inertiaTensor = new Vector3(0.1234f, grbx.freeMoi, 0.1234f);
grbx.rb.isKinematic = false;
grbx.rb.useGravity = false;
grbx.rb.angularDrag = 0.3f;
```

A.25 Υλοποίηση ανοικτού διαφορικού

```
axleInfo.leftWheel.motorTorque = grbx.wheelTorque;
axleInfo.rightWheel.motorTorque = grbx.wheelTorque;
```

A.26 Υπολογισμός τρέχουσας εφαρμοζόμενης ροπής

```
float producedTorque = engn.GetTorqueFromCurrentRpm() * hndl.throttle;
```

A.27 Εφαρμογή της ροπής στους τροχούς

```
if (grbx.gear == 0) {
        // in neutral, the flywheel rotates freely
        engn.rb.AddTorque(transform.up * producedTorque);
        // no torque delivered to the wheels as there is no mechanical
            connection
        grbx.wheelTorque = 0f;
} else {
        if (engnRpm < (engn.stallRpm - 50f)) {
                grbx.clutch = 1f;
        }

        float ratio = grbx.gearRatios[grbx.gear] * grbx.fd;
        float engageTorque = ratio * (ratio * wheelAngVel -
            engn.rb.angularVelocity.y);
        engn.rb.AddTorque(transform.up * (producedTorque + engageTorque
            * (1f - grbx.clutch)));
        grbx.wheelTorque = ratio * (producedTorque - engageTorque) *
            (1f - grbx.clutch);
}
```

A.28 Η κλάση με τις πληροφορίες του άξονα

```
public class AxleInfo {
        public WheelCollider leftWheel, rightWheel;
        public bool motor;
        public bool steering;
}
```

A.29 Εφαρμογή της ροπής και του στριψίματος στους άξονες

```
void ApplyToAxes() {
        foreach (AxleInfo axleInfo in vhcl.axleInfos) {
                if (axleInfo.steering) {
```

```
                axleInfo.leftWheel.steerAngle = hndl.steer;
                axleInfo.rightWheel.steerAngle = hndl.steer;
                axleInfo.leftWheel.brakeTorque = hndl.brake;
                axleInfo.rightWheel.brakeTorque = hndl.brake;
            }
            if (axleInfo.motor) {
                axleInfo.leftWheel.motorTorque =
                    grbx.wheelTorque;
                axleInfo.rightWheel.motorTorque =
                    grbx.wheelTorque;
                axleInfo.leftWheel.brakeTorque = hndl.brake;
                axleInfo.rightWheel.brakeTorque = hndl.brake;
            }
        }
}
```

A.30 Περιστροφή των ροδών

```
void RotateWheels() {
        // visual only - get rotation from wheelCollider and apply it
            to the model
        Vector3 pos;
        Quaternion rot;
        vhcl.axleInfos[0].leftWheel.GetWorldPose(out pos, out rot);
        vhcl.wheelBL.transform.rotation = rot;

        vhcl.axleInfos[0].rightWheel.GetWorldPose(out pos, out rot);
        vhcl.wheelBR.transform.rotation = rot;

        vhcl.axleInfos[1].leftWheel.GetWorldPose(out pos, out rot);
        vhcl.wheelFL.transform.rotation = rot;
```

```
        vhcl.axleInfos[1].rightWheel.GetWorldPose(out pos, out rot);
        vhcl.wheelFR.transform.rotation = rot;
}
```

A.31 Υπολογισμός αεροδυναμικής αντίστασης

```
void ApplyAero() {
        Rigidbody rb = GetComponent<Rigidbody>();
        float speed = rb.velocity.magnitude;
        rb.AddForce(-0.0005f * transform.forward * speed * speed,
            ForceMode.Acceleration);
}
```

A.32 Υπολογισμός ήχου κινητήρα

```
void UpdateEngineSound(float rpm) {
        float freq = rpm / 30.0f;
        vhcl.engineSound.pitch = freq / 1000.0f;
}
```

A.33 Υπολογισμός ήχου κινητήρα

```
public float maxFuelConsumption = 20; // [l/100km] the engine is only
    able of a max energy conversion rate
```

A.34 Υπολογισμός κατανάλωσης καυσίμου

```
// accumulate measurements and compute the mean value
engn.consumptionAccum += engn.maxFuelConsumption * hndl.throttle;
engn.numConsumptionMeasurements++;
engn.consumption = engn.consumptionAccum /
    engn.numConsumptionMeasurements;
```

A.35 Η κύρια κλάση του οδηγού

```
public class Driver{
        public Trajectory trajectory;
        public Handling handling;
        public Behaviour behaviour;
        public Vehicle vehicle;
        public Engine engine;
        public Gearbox gearbox;
}
```

A.36 Η κλάση του χειρισμού

```
public class Handling{
        public bool letGas;
        public bool letBrake;
        public float throttle;
        public float brake;
        public float steer;
        public bool allowShift = true;
        public int checkTurnAt = 0;
        public List<Vector2> speedLUT;
        public float shiftUpRev = 1500f;
        public float shiftDownRev = 1100f;
}
```

A.37 Η κλάση της συμπεριφοράς

```
public class Behaviour{
        //multiplies the target angle. No responsiveness = 0, full
           responsiveness = 1
        public float responsiveness;
        //degrees per frame
        public float steeringAggression;
        public bool stop = false;
```

```
        public Behaviour(){
                responsiveness = 1.0f;
                steeringAggression = 2.0f;
        }

        public void SetResponsiveness(float r){
                if (r < 0f) r = -r;
                if (r > 1f) r = 1f;
                responsiveness = r;
        }

        public void Focus(){
                if (responsiveness < 1f) {
                        responsiveness += 0.01f;
                }
        }
}
```

A.38 Η συνάρτηση Start του οχήματος

```
        void Start() {
                InitializeParams();

                SetColor();

                state.dirs.Add(new Directions());

                vhcl.Lineup(traj.currPath);
        }
```

A.39 Αρχικοποίηση αντικειμένων και παραμέτρων του οχήματος

```
        traj = new Trajectory();
        bhvr = new Agent.Behaviour();
        hndl = new Handling();
        engn = new Engine();
        grbx = new Gearbox();
        bhvr.responsiveness = Random.Range(0.5f, 1.0f);
        bhvr.steeringAggression = Random.Range(3.0f, 4.0f);
        traj.currLane = 0;
        traj.currCheckpoint = 0;
        vhcl.letDestroy = false;
        hndl.letGas = true;
        hndl.throttle = 0f;
        hndl.brake = 0f;
        hndl.steer = 0f;
        grbx.clutch = 0f;
```

A.40 Αρχικοποίηση παραμέτρων του powertrain

```
void SetupPowertrain() {
        engn.rb = engineObject.GetComponent<Rigidbody>();
        engn.rb.detectCollisions = false;
        engn.rb.constraints = RigidbodyConstraints.FreezePosition |
                                                    RigidbodyConstraints.FreezeRotationX
                                                        |
                                                    RigidbodyConstraints.FreezeRotationZ;
        engn.rb.mass = 20f;
        ...
```

A.41 Αρχικοποίηση του πίνακα αντιστοιχίας γωνίας-ταχύτητας

```
hndl.speedLUT = new List<Vector2>(){
        //angle, desired speed in m/s
        new Vector2(0f,state.speedLimit),
```

```
        new Vector2(20f,12f),
        ...
};
```

A.42 Επιλογή αρχικής λωρίδας

```
        //start at a random start lane and determine next lane
        int randLane = Random.Range(0, state.startLanesIdx.Count);
        traj.GoToLane(state.startLanesIdx[randLane]);
        List<int> conns = state.lanes[traj.currLane].connsTo;
        traj.nextLane = Random.Range(0, conns.Count);
```

A.43 Αρχικοποίηση χρώματος

```
        void setColor() {
                vhcl.color = new Color(Random.Range(0f, 1f),
                    Random.Range(0f, 1f), Random.Range(0f, 1f));
                vhcl.car.transform.Find("Classic_16_Body").GetComponent<Renderer>().mat
                    = vhcl.color;
                ...
```

A.44 Η κλάση directions με τις διάφορες κατευθύνσεις

```
public class Directions {
        public Vector3 carPos;
        public Vector3 carDir;
        public Vector3 pathDir;
        public Vector3 carToTarget;
}
```

A.45 Η συνάρτηση ευθυγράμμισης του οχήματος κατά την τοποθέτηση

```
public void Lineup(List<Vector3> path) {
        //check for other cars there
```

```
        for (int i = 0; i < state.cars.Count; ++i) {
                Vehicle veh =
                    state.cars[i].GetComponent<simpleMove>().vhcl;
                if (Vector2.SqrMagnitude(veh.pos.xz() - path[0].xz()) <
                    25) {
                        letDestroy = true;
                }
        }

        dirCompute();

        if (letDestroy == false) {
                car.transform.position = path[0];
        }

        Vector3 pathDir = path[1] - path[0];
        float angleToPathDir = Vector2.Angle(dirVec.xz(), pathDir.xz());
        if (Vector3.Cross(dirVec.xz(), pathDir.xz()).z > 0) {
                angleToPathDir = -angleToPathDir;
        }
        car.transform.localRotation = Quaternion.Euler(0,
            angleToPathDir, 0);
}
```

A.46 Υπολογισμός κατεύθυνσης οχήματος

```
public void dirCompute() {
        dirVec = wheelFL.transform.position -
            wheelBL.transform.position;
}
```

A.47 Η συνάρτηση GoToLane

```
public void GoToLane(int lane){
        currLane = lane;
        Generate (false);
        currCheckpoint = 0;
}
```

A.48 Δημιουργία σημείων τροχιάς 1

```
public void Generate(bool smoothConnection) {

        LaneBorder left =
            state.borders[state.lanes[currLane].leftBorderIdx];
        LaneBorder right =
            state.borders[state.lanes[currLane].rightBorderIdx];

        lBorder = left;
        rBorder = right;
...
```

A.49 Δημιουργία σημείων τροχιάς 2

```
...
        Vector3 midPoint =
            CheckpointGenerator.BezierPointAtParameterVal(avgP0, avgP1,
            avgP2, avgP3, 0.5f);
...
```

A.50 Προσέγγιση μήκους καμπύλης

```
        //approximate arc length - 2 piece approximation
        float steps = (Vector3.Magnitude(midPoint - avgP0) +
            Vector3.Magnitude(avgP3 - midPoint))/meterRes;
```

A.51 Δημιουργία ομαλής σύνδεσης

```
        if (smoothConnection) {
                avgP0 = 0.5f*currTarget + 0.5f*avgP0;
        }
```

A.52 Χρήση της συνάρτησης για δημιουργία τροχιάς

```
        currPath = new List<Vector3>();
        currPath = CheckpointGenerator.CubicBezierFromPoints(avgP0,
            avgP1, avgP2, avgP3, steps);
```

A.53 Συνθήκες για επόμενο checkpoint

```
        if (CurrentTargetReached() || EndOfLaneReached()) return;
```

A.54 Χειρισμός για άφιξη σε checkpoint

```
bool CurrentTargetReached() {
        // if current target reached, go to next checkpoint
        if (Vector3.Distance(traj.currTarget, vhcl.pos) < 0.1f) {
                traj.currCheckpoint++;
                return true;
        } else {
                return false;
        }
}
```

A.55 Έλεγχος τέλους λωρίδας

```
bool EndOfLaneReached(){
        bool endOfLane = false;

        //end of lane, go to next if it exists
        if (traj.currCheckpoint >= (traj.currPath.Count - 1)) {
                if (state.lanes[traj.currLane].mode == 2){
```

```
                vhcl.letDestroy = true;
            } else {
                List<int> conns =
                    state.lanes[traj.currLane].connsTo;
                traj.NextLane(conns[traj.nextLane], true);
                traj.nextLane = Random.Range(0, conns.Count);
            }

            endOfLane = true;
        }

        return endOfLane;
}
```

A.56 Προετοιμασία τρέχοντος στόχου

```
void SetupCurrentTarget(){
        traj.currTarget = traj.currPath[traj.currCheckpoint + 1];
        carPosToTarget = traj.currTarget - vhcl.pos;

        vhcl.dirCompute();

        float angleToTarget = Vector2.Angle(vhcl.dirVec.xz(),
            carPosToTarget.xz());
        Vector3 crossProd = Vector3.Cross(vhcl.dirVec.xz(),
            carPosToTarget.xz());
        if (crossProd.z > 0.0f) { angleToTarget = -angleToTarget; }

        traj.distanceToCurrent =
            Vector3.Distance(traj.currPath[traj.currCheckpoint],
            vhcl.pos);
        float distanceToNext =
```

```
            Vector3.Distance(traj.currPath[traj.currCheckpoint + 1],
            vhcl.pos);

        // Skip passed checkpoints
        if (traj.distanceToCurrent >= distanceToNext) {
                traj.currCheckpoint++;
                return;
        }
}
```

A.57 Υπολογισμός γωνίας προς την τροχιά

```
        traj.angleToPathDir = traj.GetAngleToPathDir(vhcl.dirVec.xz());
```

A.58 Υπολογισμός προσήμου γωνίας μέσω εξωτερικού γινομένου

```
        Vector3 pathDirVec = traj.GetPathDirectionVec();
        Vector3 cross = Vector3.Cross(vhcl.dirVec.xz(),
            pathDirVec.xz());
        if (cross.z > 0.0f) { traj.angleToPathDir =
            -traj.angleToPathDir; }
```

A.59 Υπολογισμός παράγοντα s

```
        float s = 2f * Mathf.Atan(2.0f * Mathf.Pow(0.5f *
            traj.distanceToCurrent, 2f)) / Mathf.PI;
        float steerAngle = (traj.angleToPathDir + s * angleToTarget) /
            (1.0f + s);
```

A.60 Περιορισμός γωνίας στριψίματος

```
        // clamp
        if (steerAngle > vhcl.maxSteeringAngle) { steerAngle =
            vhcl.maxSteeringAngle; }
```

```
        else if (steerAngle < -vhcl.maxSteeringAngle) { steerAngle =
            -vhcl.maxSteeringAngle; }
```

A.61 Χρήση του responsiveness

```
        // apply responsiveness
        float finalSteerAngle = steerAngle * bhvr.responsiveness;
```

A.62 Χρήση του steeringAggression

```
        // steer slowly
        if (finalSteerAngle > hndl.steer) { hndl.steer +=
            bhvr.steeringAggression; }
        else if (finalSteerAngle < hndl.steer) { hndl.steer -=
            bhvr.steeringAggression; }
```

A.63 Αρχικοποίηση των πεταλιών

```
void CalculatePedals() {
        hndl.throttle = 0f;
        hndl.brake = 0f;

        vhcl.vel =
            vhcl.car.GetComponent<Rigidbody>().velocity.magnitude;
```

A.64 Περιορισμός δείκτη τροχιάς

```
        hndl.checkTurnAt = traj.currCheckpoint + (int)(2f * vhcl.vel /
            traj.meterRes);
        if (hndl.checkTurnAt >= (traj.currPath.Count - 1)) {
                // clamp
                hndl.checkTurnAt = traj.currPath.Count - 2;
        }
```

A.65 Διάνυσμα οχήματος-checkpoint

```
Vector2 turnVector = traj.currPath[hndl.checkTurnAt].xz() -
    vhcl.pos.xz();
float turnAngle = Mathf.Abs(Vector2.Angle(vhcl.dirVec.xz(),
    turnVector));
```

A.66 Υπολογισμός ταχύτητας από τη γωνία

```
//look up the LUT to find the suitable speed
for (int i = 0; i < hndl.speedLUT.Count; ++i) {
        if (turnAngle < hndl.speedLUT[i].x) {
                targetSpeed = hndl.speedLUT[i - 1].y +
                        ((hndl.speedLUT[i].x - turnAngle) /
                            (hndl.speedLUT[i].x -
                            hndl.speedLUT[i - 1].x)) *
                        (hndl.speedLUT[i].y - hndl.speedLUT[i -
                            1].y);
                i = hndl.speedLUT.Count;
        }
}
```

A.67 Εφαρμογή ορίου ταχύτητας

```
if (targetSpeed > state.speedLimit) {
        targetSpeed = state.speedLimit;
}
```

A.68 Ακινητοποίηση

```
if (bhvr.stop == true) {
        targetSpeed = 0f;
}
```

A.69 Υπολογισμός γκαζιού για επίτευξη ταχύτητας-στόχου 1

```
if (vhcl.vel < targetSpeed) {
        hndl.brake = 0f;
        hndl.throttle = 0.5f * (Mathf.Pow(2.0f, -0.0006f *
            Mathf.Pow(traj.angleToPathDir, 2.0f)));
```

A.70 Υπολογισμός γκαζιού για επίτευξη ταχύτητας-στόχου 2

```
} else {
        hndl.throttle = 0f;
        float vel_diff = vhcl.vel - targetSpeed;
        hndl.brake = (vhcl.maxBrakeTorque * (2f *
            Mathf.Atan(vel_diff) / Mathf.PI));
        if (bhvr.stop == true) {
                hndl.brake += 0.1f * vhcl.maxBrakeTorque;
        }
}
```

A.71 Υπολογισμός γκαζιού για επίτευξη ρελαντί

```
// keep engine idle
if (engnRpm < engn.stallRpm) {
        hndl.throttle += 0.1f;
}
```

A.72 Όρια περιοριστή στροφών

```
public float revUpLimit = 6500f; // rev limiter
public float revLowLimit = 6300f; // let throttle again when
    below these rpm
```

A.73 Χειρισμός γκαζιού στον περιοριστή στροφών

```
// rev limiter
if (engnRpm > engn.revUpLimit) {
```

```
            engn.letThrottle = false;
        }
        else if (engnRpm < engn.revLowLimit) {
            engn.letThrottle = true;
        }
```

A.74 Η μεταβλητή letThrottle

```
if (engn.letThrottle == false) { hndl.throttle = 0f; }
```

A.75 Έλεγχος τρέχουσας αλλαγής

```
if (grbx.currentlyOnGearChange) {
        grbx.gearChangeTime += 0.08f;
        grbx.clutch = 1f - grbx.gearChangeTime;
        engn.letThrottle = false;
        if (grbx.gearChangeTime > 1f) {
            grbx.currentlyOnGearChange = false;
            grbx.gearChangeTime = 0f;
        }
}
```

A.76 Ανανέωση χρόνου αλλαγής σχέσης

```
if (grbx.allowGearChangeTimeout > 0)
    grbx.allowGearChangeTimeout--;
```

A.77 Εύρος στροφών αλλαγής σχέσης

```
public float shiftUpRev = 1500f;
public float shiftDownRev = 1100f;
```

A.78 Εκτέλεση αλλαγής σχέσης

```
if (engnRpm > hndl.shiftUpRev) grbx.shiftUp();
if (engnRpm < hndl.shiftDownRev) grbx.shiftDown();
```

A.79 Αλλαγή σε επόμενη σχέση

```
public void shiftUp() {
        if (gear < 5 && allowGearChangeTimeout == 0) {
                setGear(gear + 1);
                allowGearChangeTimeout = 100;
        }
}
```

A.80 Η συνάρτηση setGear

```
public void setGear(int n) {
        currentlyOnGearChange = true;
        gear = n;
}
```

A.81 Αλλαγή σε προηγούμενη σχέση

```
public void shiftDown() {
        if (gear > 1 && allowGearChangeTimeout == 0) {
                setGear(gear - 1);
                allowGearChangeTimeout = 100;
        }
}
```

A.82 Υπολογισμός μεριάς σε σχέση με διάνυσμα

```
public static bool IsPointLeftOfVector(Vector2 vp1, Vector2 vp2,
    Vector2 p) {
        return (vp2.x - vp1.x) * (p.y - vp1.y) - (vp2.y - vp1.y) * (p.x
            - vp1.x) > 0;
}
```

A.83 Εξωτερικό γινόμενο

```
public static float CrossProduct(Vector2 vp1, Vector2 vp2, Vector2 p) {
        return (vp2.x - vp1.x) * (p.y - vp1.y) - (vp2.y - vp1.y) * (p.x
            - vp1.x);
}
```

A.84 Έλεγχος τομής ευθυγράμμων τμημάτων

```
public static bool VectorsIntersect(Vector2 v1p1, Vector2 v1p2, Vector2
    v2p1, Vector2 v2p2) {
        return ((CrossProduct(v1p1, v1p2, v2p1) * CrossProduct(v1p1,
            v1p2, v2p2) < 0f) &&
                (CrossProduct(v2p1, v2p2, v1p1) * CrossProduct(v2p1,
                    v2p2, v1p2) < 0f));
}
```

A.85 Αρχικοποίηση προτεραιότητας

```
void Awareness() {
        bhvr.stop = false;
```

A.86 Επιλογή μετέπειτα σημείου στην τροχιά

```
        int pointForwMeIdx = traj.currCheckpoint + (int)(2f * vhcl.vel
            / traj.meterRes);
        if (pointForwMeIdx >= (traj.currPath.Count - 1)) {
                pointForwMeIdx = traj.currPath.Count - 2;
        }
```

A.87 Ευθύγραμμο τμήμα τρέχοντος οχήματος

```
        Vector2 carPosMe = vhcl.pos.xz();
        Vector2 pointForwMe = traj.currPath[pointForwMeIdx].xz();
```

A.88 Έλεγχος για το αν πρόκειται για το τρέχον όχημα

```
for (int i = 0; i < state.cars.Count; ++i) {
        Move scr = state.cars[i].GetComponent<Move>();
        if (scr.id == id) {
                continue;// same car, skip
        }

        Vector2 carPosOther = scr.vhcl.pos.xz();
```

A.89 Ευθύγραμμο τμήμα για τα υπόλοιπα οχήματα

```
        int pointForwOtherIdx = scr.traj.currCheckpoint +
            (int)(2f * scr.vhcl.vel / scr.traj.meterRes);
        if (pointForwOtherIdx >= (scr.traj.currPath.Count - 1))
            {
                pointForwOtherIdx = scr.traj.currPath.Count - 2;
        }

        Vector2 pointForwOther =
            scr.traj.currPath[pointForwOtherIdx].xz();
```

A.90 Έλεγχος προτεραιότητας

```
        if (Utils.VectorsIntersect(carPosMe, pointForwMe,
            carPosOther, pointForwOther)) {
                // priority to the right
                bool isRight =
                    !Utils.IsPointLeftOfVector(carPosMe,
                    pointForwMe, carPosOther);
                bhvr.stop = isRight;
        }
```

A.91 Κλάση state 1

```
public class state : MonoBehaviour {
```

```
        public bool running;
```

A.92 Κλάση state 2

```
        public int instancesNum;
```

A.93 Κλάση state 3

```
        public static List<GameObject> cars;
```

A.94 Κλάση state 4

```
        public static float speedLimit = 17f; // 50 kmh;
```

A.95 Κλάση state 5

```
        public Text consumptionText;
...
```

A.96 Η start του state

```
void Start() {
        SetupVisualsLayer();
        SetupMeasuringPoints();
        LoadAndRenderGround();
        PopulateWorld();
        SetupLaneIndices();
        FindStartEndLanesIdx();
        SetupConnections();
}
```

A.97 Η φόρτωση του δαπέδου

```
        mapLength = 10.0f * s * tex.width / tex.height;
        mapWidth = 10.0f * s;
```

```
        //finally create ground
        ground = GameObject.CreatePrimitive(PrimitiveType.Plane);
        ground.GetComponent<Renderer>().material.mainTexture = tex;
        ground.GetComponent<Renderer>().material.shader =
            Shader.Find("Standard");
        ground.GetComponent<Renderer>().material.color = new
            Color(0.7f, 0.7f, 0.7f);
        ground.GetComponent<Renderer>().material.SetFloat("_Glossiness",
            0.0f);
        // the plane primitive is already 10x10 units
        ground.transform.localScale = new Vector3(-mapLength/10f, 1.0f,
            -mapWidth/10f);
```

A.98 Η συνάρτηση PopulateWorld

```
void PopulateWorld(){
        borders = new List<LaneBorder>();
        lanes = new List<Lane>();
        connections = new List<LaneConnection>();
        measuringPoints = new List<MeasuringPoint>();

        string mapFilename = menuState.mapFilename;
        mapFilename = "agia_sofia.motors";
        String si=""; //string for input line data
        StreamReader rd = new StreamReader(@mapFilename);

        while((si=rd.ReadLine()) != null){
                ...
```

A.99 Η συνάρτηση SetupLaneIndices

```
void SetupLaneIndices() {
        for (int i=0; i<lanes.Count; ++i) {
```

```
                lanes[i].leftBorderIdx = -3;
                lanes[i].rightBorderIdx = -4;
                for (int j=0; j<borders.Count; ++j) {
                        if (borders[j].laneBorderID ==
                            lanes[i].leftBorderID) {
                                lanes[i].leftBorderIdx = j;
                        }
                        if (borders[j].laneBorderID ==
                            lanes[i].rightBorderID) {
                                lanes[i].rightBorderIdx = j;
                        }
                }
        }
}
```

A.100 Έλεγχος λωρίδων έναρξης και τερματισμού

```
void FindStartEndLanesIdx() {
        for (int i = 0; i < lanes.Count; i++) {
                if (lanes[i].mode == 1) {
                        startLanesIdx.Add(i);
                }
                else if (lanes[i].mode == 2) {
                        endLanesIdx.Add(i);
                }
        }
}
```

A.101 Εύρεση συνδέσεων λωρίδων

```
void SetupConnections() {
        for (int i = 0; i < lanes.Count; i++) {
                for (int j = 0; j < connections.Count; ++j) {
```

```
                        if (lanes[i].laneID ==
                            connections[j].connectsFrom) {
                                int to_ID = connections[j].connectsTo;
                                lanes[i].connsTo.Add(Find.LaneIdxByID(to_ID,
                                    lanes));
                        }
                    }
            }
}
```

A.102 Έναρξη προσομοίωσης με το πρώτο αυτοκίνητο

```
void Update () {
        if (Input.GetKeyUp(KeyCode.Space)){
                DispatchCar();
```

A.103 Δημιουργία οχήματος

```
void DispatchCar() {
        if (running == false) return;

        Vector3 pos = new Vector3(100, 100, 100);

        GameObject g = Instantiate(car, pos, Quaternion.identity) as
            GameObject;
        g.SetActive(true);
        ++instancesNum;
        cars.Add(g);

        Invoke("DispatchCar", UnityEngine.Random.Range(3f, 8f));
}
```

A.104 Raycast από τις συντεταγμένες του ποντικιού

```
if (Input.GetMouseButtonDown(0)) {
        RaycastHit hitInfo = new RaycastHit();
        bool hit =
            Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition),
            out hitInfo);
```

A.105 Εύρεση δείκτη οχήματος σε περίπτωση επιτυχούς raycast

```
        if (hit) {
                simpleMove scr;
                Renderer carRenderer;

                int carIndex;
                try {
                        carIndex =
                            Int32.Parse(hitInfo.transform.gameObject.name);
                } catch (FormatException e) {
                        carIndex = -1;
                }
```

A.106 Εφαρμογή shader χρωματισμού στο επιλεγμένο όχημα

```
if (carIndex > -1) {
        scr = cars[carIndex].GetComponent<simpleMove>();
        carRenderer =
            scr.vhcl.car.transform.Find("Classic_16_Roof").GetComponent<Renderer>();
        carRenderer.material.shader =
            Shader.Find("Unlit/CarMarkShader");
}
```

A.107 Ο shader χρωματισμού στο επιλεγμένο όχημα

```
fixed4 frag (v2f i) : SV_Target
{
        // sample the texture
        fixed4 col = tex2D(_MainTex, i.uv);

        col.r = 0.0;
        col.g = sin(8.0*_Time.y)*sin(8.0*_Time.y);
        col.b = 0.0;
        col.a = 0.0;

        return col;
}
```

A.108 Έλεγχος για οχήματα που ολοκλήρωσαν τη διαδρομή τους

```
        for (int i = 0; i < cars.Count; ++i){
                Move script = cars[i].GetComponent<Move>();
                if (script.vhcl.letDestroy){
                        Destroy(cars[i]);
                        cars.RemoveAt(i);
                        --instancesNum;
                        dirs.RemoveAt(i);
                }
        }
```

A.109 Αρχικοποίηση της οπτικοποίησης

```
void Start() {
        SetupVisualsLayer();
        SetupMeasuringPoints();
```

A.110 Αρχικοποίηση σημείων μέτρησης

```
public class MeasuringPoint {
```

```
        public Vector3 pos;
        public float radius;
        public float valuesSum;
        public int measurementsNum;
```

A.111 Δημιουργία σημείων μέτρησης 1

```
void SetupMeasuringPoints() {
        // For each lane, we take measuring points along its average
           path
        // (the same path the cars follow) and check for cars in a
           circle

        for (int i = 0; i < lanes.Count; ++i) {
...
```

A.112 Δημιουργία σημείων μέτρησης 2

```
            Vector3 midPoint =
                CheckpointGenerator.BezierPointAtParameterVal(avgP0,
                avgP1, avgP2, avgP3, 0.5f);
```

A.113 Δημιουργία σημείων μέτρησης 3

```
            float steps = (Vector3.Magnitude(midPoint - avgP0) +
                Vector3.Magnitude(avgP3 - midPoint)) / 1f;
```

A.114 Δημιουργία σημείων μέτρησης 4

```
            List<Vector3> positions =
                CheckpointGenerator.CubicBezierFromPoints(avgP0,
                avgP1, avgP2, avgP3, steps);
```

A.115 Δημιουργία σημείων μέτρησης 5

```
            float laneWidth = Vector2.Distance(lb.pt0, rb.pt0);
```

A.116 Δημιουργία σημείων μέτρησης 6

```
            for (int k = 0; k < positions.Count; ++k) {
                    MeasuringPoint mp = new MeasuringPoint();
                    mp.pos = positions[k];
                    mp.radius = 0.5f * laneWidth;

                    measuringPoints.Add(mp);
            }
```

A.117 Η συνάρτηση CheckForMeasuringPoints

```
void CheckForMeasuringPoints() {
        for (int i = 0; i < state.measuringPoints.Count; ++i) {
                MeasuringPoint mp = state.measuringPoints[i];
                if (Vector2.Distance(vhcl.pos.xz() - mp.pos.xz()) <
                   mp.radius) {
                        state.measuringPoints[i].valuesSum += vhcl.vel;
                        state.measuringPoints[i].measurementsNum++;
                        continue;
                }
        }
}
```

A.118 Η συνάρτηση SetupVisualsLayer

```
void SetupVisualsLayer() {
        statVisualsTex = new Texture2D(statTexSizeX, statTexSizeY,
           TextureFormat.BGRA32, false);
```

A.119 Αρχικοποίηση του texture οπτικοποίησης 1

```
        for (int i = 0; i < statTexSizeX; ++i) {
                for (int j = 0; j < statTexSizeY; ++j) {
                        statVisualsTex.SetPixel(i, j, new Color(0.0f,
                            0.0f, 0.0f, 0.0f));
                }
        }
```

A.120 Αρχικοποίηση του texture οπτικοποίησης 2

```
        statVisualsTex.Apply();
```

A.121 Δημιουργία του texture οπτικοποίησης 1

```
        ground.GetComponent<Renderer>().material.SetTexture("_StatTex",
            statVisualsTex);
```

A.122 Δημιουργία του texture οπτικοποίησης 2

```
        statFillColors = new Color32[statTexSizeX * statTexSizeY];
```

A.123 Δημιουργία του texture οπτικοποίησης 3

```
void UpdateStatsTex() {
        for (int i = 0; i < measuringPoints.Count; ++i) {
                Vector3 p = measuringPoints[i].pos;
                int idxX = (int)(statTexSizeX * (p.x+0.5*mapLength) /
                    mapLength);
                int idxY = (int)(statTexSizeY * (p.z+0.5*mapWidth) /
                    mapWidth);
```

A.124 Δημιουργία του texture οπτικοποίησης 4

```
                float avgPointSpeed = measuringPoints[i].valuesSum /
                    measuringPoints[i].measurementsNum;
```

A.125 Δημιουργία του texture οπτικοποίησης 5

```
Color c = new Color(avgPointSpeed / speedLimit, 0.0f,
    0.0f, 1.0f);
```

A.126 Δημιουργία του texture οπτικοποίησης 6

```
statFillColors[(idxX-1)+(idxY-1)*statTexSizeX] = c;
statFillColors[(idxX-1)+(idxY+0)*statTexSizeX] = c;
statFillColors[(idxX-1)+(idxY+1)*statTexSizeX] = c;
statFillColors[(idxX+0)+(idxY-1)*statTexSizeX] = c;
statFillColors[(idxX+0)+(idxY+0)*statTexSizeX] = c;
statFillColors[(idxX+0)+(idxY+1)*statTexSizeX] = c;
statFillColors[(idxX+1)+(idxY-1)*statTexSizeX] = c;
statFillColors[(idxX+1)+(idxY+0)*statTexSizeX] = c;
statFillColors[(idxX+1)+(idxY+1)*statTexSizeX] = c;
```

A.127 Δημιουργία του texture οπτικοποίησης 7

```
statVisualsTex.SetPixels32(statFillColors);
statVisualsTex.Apply();
ground.GetComponent<Renderer>().material.SetTexture("_StatTex",
    statVisualsTex);
```

A.128 Ο shader οπτικοποίησης 1

```
sampler2D _MainTex;
sampler2D _StatTex;
```

A.129 Ο shader οπτικοποίησης 2

```
v2f vert (appdata v)
{
        v2f o;
        o.vertex = mul(UNITY_MATRIX_MVP, v.vertex);
        o.uv = TRANSFORM_TEX(v.uv, _MainTex);
```

```
        UNITY_TRANSFER_FOG(o,o.vertex);
        return o;
}
```

A.130 Ο shader οπτικοποίησης 3

```
        // sample the textures
        fixed4 col = tex2D(_MainTex, i.uv);
        fixed4 col2 = tex2D(_StatTex, i.uv); // this comes as red only
```

A.131 Ο shader οπτικοποίησης 4

```
        // make it green to red gradient
        fixed4 col_rg;
        col_rg.r = 1.0 - col2.r;
        col_rg.g = col2.r;
        col_rg.b = 0.0;
        col_rg.a = col2.a;
```

A.132 Ο shader οπτικοποίησης 4

```
        return 0.3*col + 0.7*col_rg;
```

A.133 Η συνάρτηση SwitchViewButtonClick 1

```
public void SwitchViewButtonClick() {
        Camera cameraComponent = cam.GetComponent<Camera>();
```

A.134 Η συνάρτηση SwitchViewButtonClick 2

```
        if (cameraComponent.orthographic == false) {
                preservedCameraPosition = cam.transform.position;
                preservedCameraRotation = cam.transform.rotation;
```

A.135 Η συνάρτηση SwitchViewButtonClick 3

```
} else {
        cameraComponent.orthographic = false;
        cam.transform.position = preservedCameraPosition;
        cam.transform.rotation = preservedCameraRotation;
        ground.GetComponent<Renderer>().material.shader =
            Shader.Find("Standard");
}
```