ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΥΤΟΜΑΤΗ ΕΚΠΑΙΔΕΥΣΗ ΑΥΤΟΚΙΝΗΤΟΥ ΜΕ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΧΑΡΑΛΑΜΠΙΔΗ ΘΕΟΔΩΡΟΥ

(ΑΕΜ: 2220)

Επιβλέπων : Ιωάννης Σινάτκας

Καθηγητής

Καστοριά Απρίλιος - 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΥΤΟΜΑΤΗ ΕΚΠΑΙΔΕΥΣΗ ΑΥΤΟΚΙΝΗΤΟΥ ΜΕ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΧΑΡΑΛΑΜΠΙΔΗ ΘΕΟΔΩΡΟΥ

(ΑΕΜ: 2220)

Επιβλέπων : Ιωάννης Σινάτκας

Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15/4/2024

.....	.....	.....
Ιωάννης Σινάτκας	Δημήτριος Φωτιάδης	Μιχαήλ Σταμπουλτζής
Καθηγητής	Επίκουρος Καθηγητής	Λέκτορας

Καστοριά Απρίλιος - 2024

Ευχαριστίες

Εκφράζω την εγκάρδια ευγνωμοσύνη μου στο τμήμα Πληροφορικής του Πανεπιστημίου Δυτικής Μακεδονίας, που μου έδωσε την ευκαιρία να ξεκινήσω το ακαδημαϊκό μου ταξίδι και για την προώθηση ενός περιβάλλοντος μάθησης και ανάπτυξης. Αυτή η διατριβή αντιπροσωπεύει το αποκορύφωμα πολυετούς αφοσίωσης και γνωσιακής υποστήριξης από τους καθηγητές. Πιο συγκεκριμένα, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Ιωάννη Σινάτκα, που συντέλεσε όχι μόνο στην ολοκλήρωση αλλά και στην βελτιστοποίηση της παρούσας πτυχιακής εργασίας.

Περίληψη

Η τεχνολογία αυτόνομης οδήγησης έχει αναδειχθεί ως ένα πολλά υποσχόμενο θέμα στα μέσα μεταφοράς, έχοντας δώσει την υπόσχεση επαναπροσδιορισμού του μοντέλου μετακίνησης και της ενίσχυσης της οδικής ασφάλειας σε όλο το οδικό δίκτυο του κόσμου. Αυτή η πτυχιακή διατριβή παρουσιάζει μια έρευνα για την ανάπτυξη ενός συστήματος αυτοοδηγούμενων αυτοκινήτων χρησιμοποιώντας τεχνολογίες όπως Python, Pygame και τον αλγόριθμο NEAT (NeuroEvolution of Augmenting Topologies). Η έρευνα στοχεύει στη διερεύνηση των δυνατοτήτων των νευρωνικών δικτύων και των εξελικτικών τεχνικών υπολογισμού στην εκπαίδευση αυτόνομων οχημάτων και της αξιολόγησης της απόδοσή τους.

Στη συγκεκριμένη εργασία, μετά την υλοποίηση του προγράμματος θα διεξαχθούν τρία πειράματα. Σε αυτά τα τρία πειράματα θα δημιουργηθούν κυκλικά περιβάλλοντα έτσι ώστε να μπορούν τα αυτοκίνητα να κινηθούν καθ’ όλη τη διάρκεια της προσομοίωσης. Επίσης σε κάθε ένα πείραμα θα παραμετροποιηθεί ο αριθμός των κρυφών επιπέδων του νευρωνικού δικτύου, έτσι ώστε να γίνει σύγκριση αποδοτικότητας του μοντέλου που θα εκπαιδευτεί, μεταξύ των τριών πειραμάτων.

Πιο συγκεκριμένα, τα τρία αυτά πειράματα θα γίνουν με μηδέν, δέκα και εκατό κρυφά επίπεδα στο νευρωνικό δίκτυο αντιστοίχως. Κατά αυτό τον τρόπο, τα μοντέλα που θα εκπαιδευτούν θα επιφέρουν και τα αντίστοιχα δεδομένα αναφοράς, που με την βοήθεια τους θα μπορέσει να διεξαχθεί το συμπέρασμα για το πιο από τα τρία μοντέλα έχει καλύτερο βαθμό καταλληλότητας στο να κινείται αυτόβουλα στα κυκλικά περιβάλλοντα.

Λέξεις Κλειδιά: **Τεχνητή Νοημοσύνη, Νευρωνικά Δίκτυα, Εξελικτικοί Αλγόριθμοι**

Abstract

Autonomous driving technology has emerged as a promising frontier in transportation, with the potential to redefine mobility and enhance safety on roads worldwide. This thesis presents an investigation into the development of a self-driving car system using technologies such as Python, Pygame, and the NEAT (NeuroEvolution of Augmenting Topologies) algorithm. The research aims to explore the capabilities of neural networks and evolutionary computation techniques in training autonomous vehicles and to evaluate their performance.

In this work, after the implementation of the program, three experiments will be conducted. In these three experiments, circular environments will be created so that the cars can move throughout the simulation. Also, in each experiment, the number of hidden layers of the neural network will be parameterized, so that the efficiency of the model to be trained can be compared between the three experiments.

More specifically, these three experiments will be done with zero, ten and one hundred hidden layers in the neural network respectively. In this way, the models that will be trained will also bring the corresponding reference data, with the help of which it will be possible to conclude which of the three models has a better degree of suitability to move autonomously in circular environments.

Key Words: **Artificial Intelligence, Neural Networks, Evolution Algorithms**

Πίνακας Περιεχομένων

Λίστα Εικόνων

Εισαγωγή

Τα τελευταία χρόνια, η πρόοδος των τεχνολογιών αυτόνομης οδήγησης έχει συγκεντρώσει σημαντική προσοχή λόγω των δυνατοτήτων της να φέρει επαναστατικό επαναπροσδιορισμό στα μέσα μεταφορών παγκοσμίως. Η ενσωμάτωση των τεχνικών τεχνητής νοημοσύνης (AI) και μηχανικής μάθησης (ML) έχει παίξει καθοριστικό ρόλο στην ανάπτυξη αυτόνομων οχημάτων, επιτρέποντάς τους να αντιλαμβάνονται και να πλοηγούνται σε πολύπλοκα περιβάλλοντα χωρίς ανθρώπινη παρέμβαση. Σε αυτό το πλαίσιο, η παρούσα πτυχιακή διατριβή, διερευνά την εφαρμογή ενός συστήματος αυτοοδηγούμενων αυτοκινήτων χρησιμοποιώντας τεχνολογίες όπως Python, Pygame και τον αλγόριθμο NEAT (NeuroEvolution of Augmenting Topologies).

Ο στόχος αυτής της έρευνας είναι η σχεδίαση και ανάπτυξη ενός μοντέλου αυτοοδηγούμενου αυτοκινήτου ικανό να μάθει και να προσαρμόσει τη συμπεριφορά του μέσω διαδικασιών εξέλιξης. Οι παραδοσιακές προσεγγίσεις στον αυτόνομο έλεγχο οχημάτων βασίζονται συχνά σε προκαθορισμένους κανόνες ή χειροποίητους αλγόριθμους, οι οποίοι μπορεί να δυσκολεύονται ως προς τη γενίκευση σε διαφορετικά και δυναμικά περιβάλλοντα. Αντίθετα, οι εξελικτικοί αλγόριθμοι προσφέρουν μια πολλά υποσχόμενη εναλλακτική προσέγγιση, επιτρέποντας στο σύστημα να εξελίσσεται και να βελτιστοποιεί τις διαδικασίες λήψης αποφάσεων με την πάροδο του χρόνου, μιμούμενοι τις αρχές της φυσικής επιλογής.

Όσον αφορά τις τεχνολογίες που θα χρησιμοποιηθούν, η επιλογή της Python ως κύριας γλώσσας προγραμματισμού προσφέρει πολλά πλεονεκτήματα, συμπεριλαμβανομένης της ευκολίας χρήσης, των εκτεταμένων βιβλιοθηκών της στο οικοσύστημα της καθώς και μια ζωντανή κοινότητα προγραμματιστών που μπορεί να συμβουλευτεί κάποιος σε περίπτωση λειτουργικής ανάγκης. Η βιβλιοθήκη Pygame, είναι ένα σύνολο λειτουργικών μονάδων στη γλώσσα Python, που έχουν σχεδιαστεί για την ανάπτυξη παιχνιδιών, καθώς παρέχει ένα βολικό πλαίσιο για τη δημιουργία δια δραστικών προσομοιώσεων του περιβάλλοντος του αυτοκινήτου αυτόνομης οδήγησης, επιτρέποντας οπτικοποίηση και πειραματισμό σε πραγματικό χρόνο. Επιπλέον, η ενσωμάτωση του NEAT, μιας ισχυρής βιβλιοθήκης εξελικτικών αλγορίθμων, εξουσιοδοτεί το σύστημα αυτοοδηγούμενων αυτοκινήτων να εξελίσσει αυτόνομα την δομή αλλά και τα βάρη των νευρωνικών δικτύων, επιτρέποντας προσαρμοστικές και ισχυρές στρατηγικές ελέγχου.

Μέσω της παρούσας διπλωματικής εργασίας, έχει τεθεί ως στόχος, η απάντηση των ακόλουθων ερευνητικών ερωτημάτων. Πώς μπορούν οι τεχνολογίες Python, Pygame και NEAT να χρησιμοποιηθούν αποτελεσματικά για την ανάπτυξη μιας πλατφόρμας προσομοίωσης αυτοοδηγούμενου αυτοκινήτου, καθώς και η εύρεση του βαθμού απόδοσης ενός τέτοιου μοντέλου.

1. Σκοπός Της Εργασίας

Ο στόχος αυτής της έρευνας είναι η σχεδίαση και ανάπτυξη ενός μοντέλου αυτοοδηγούμενου αυτοκινήτου ικανό να μάθει και να προσαρμόσει τη συμπεριφορά του μέσω διαδικασιών εξέλιξης. Οι παραδοσιακές προσεγγίσεις στον αυτόνομο έλεγχο οχημάτων βασίζονται συχνά σε προκαθορισμένους κανόνες ή χειροποίητους αλγόριθμους, οι οποίοι μπορεί να δυσκολεύονται ως προς τη γενίκευση σε διαφορετικά και δυναμικά περιβάλλοντα. Αντίθετα, οι εξελικτικοί αλγόριθμοι προσφέρουν μια πολλά υποσχόμενη εναλλακτική προσέγγιση, επιτρέποντας στο σύστημα να εξελίσσεται και να βελτιστοποιεί τις διαδικασίες λήψης αποφάσεων με την πάροδο του χρόνου, μιμούμενοι τις αρχές της φυσικής επιλογής.

Σε αυτό το κεφάλαιο θα αναλυθεί η δομή και η ακολουθιακή αρχιτεκτονική της εργασίας, θα γίνει αναφορά για τη λήψη και την εγκατάσταση των εργαλείων που θα χρησιμοποιηθούν, καθώς θα αναπτυχθούν πρότυπα για το περιβάλλον και το αυτοκίνητο.

1.1 Αρχιτεκτονική και δομή εργασίας

Σε αυτό το κεφάλαιο θα γίνει αναφορά της αρχιτεκτονικής της εργασίας και πως έχει δομηθεί. Αναφορικά η αρχιτεκτονική εφαρμογών αναφέρεται στον δομικό σχεδιασμό εφαρμογών λογισμικού. Περιλαμβάνει τη δομή υψηλού επιπέδου, τα στοιχεία, τις σχέσεις και τις αλληλεπιδράσεις μέσα σε ένα σύστημα εφαρμογών. Ο στόχος της αρχιτεκτονικής εφαρμογών είναι να διασφαλίσει ότι το σύστημα λογισμικού πληροί τις λειτουργικές και μη λειτουργικές του απαιτήσεις, όπως η επεκτασιμότητα, η αξιοπιστία, η δυνατότητα συντήρησης και η απόδοση.

Η αρχιτεκτονική εφαρμογών τυπικά τεκμηριώνεται χρησιμοποιώντας διάφορα διαγράμματα, όπως διαγράμματα κλάσεων, διαγράμματα ανάπτυξης και διαγράμματα ακολουθίας, για να βοηθήσει τους ενδιαφερόμενους, είτε είναι μία ομάδα προγραμματιστών είτε είναι κάποιος μη σχετικός επι του θέματος, να κατανοήσουν τη δομή και τη συμπεριφορά του συστήματος ή του λογισμικού. Χρησιμεύει ως σχέδιο για τις ομάδες ανάπτυξης που πρέπει να ακολουθήσουν κατά τη φάση υλοποίησης και παρέχει τη βάση για συνεχή συντήρηση και εξέλιξη της εφαρμογής.

1.1.1 Δομή εργασίας

Είναι πολύ σημαντικό για τον αναγνώστη της εργασίας να γνωρίζει τη δομή της παρούσας εργασίας πριν ακολουθήσει η ανάπτυξη της για λόγους καλύτερης κατανόησης.

Αρχικά έχει δημιουργηθεί ένας φάκελος ο οποίος στεγάζει όλη την εργασία και την επιμέρους λογική της. Ο φάκελος αυτός μπορεί να έχει οποιοδήποτε όνομα όπως για παράδειγμα test, αλλά θα είναι καλύτερο να αντιπροσωπεύει το σκοπό της εργασίας. Για αυτό τον λόγο ο φάκελος έχει ονομαστεί AI_CARS και μέσα σε αυτό το φάκελο βρίσκονται τα επιμέρους κομμάτια του, που το κάθε ένα από αυτά συμπεριλαμβάνει τη λογική που δίνει λειτουργικότητα στην εργασία.

Μέσα στον φάκελο AI_CARS, δημιουργείται ένας φάκελος με όνομα assets. Η σκοπιμότητα αυτού του φακέλου είναι η αποθήκευση της εικόνας του αυτοκινήτου σε αυτόν, έτσι ώστε να μπορεί να γίνει εύκολα αναφορά σε αυτή την εικόνα καθώς και το που βρίσκεται.

Στη συνέχεια δημιουργείται ένας φάκελος με όνομα car, ο οποίος χαρακτηριστικά περικλείει όλα τα αρχεία που δίνουν λειτουργικότητα στο αυτοκίνητο της εργασίας. Σε αυτό το φάκελο δημιουργούνται τα αρχεία με όνομα actions.py, car_ai.py, car.py και config.txt. Αντιστοίχως, στο πρώτο αρχείο υπάρχει η λογική των δράσεων ή των κινήσεων που μπορεί να εκτελέσει το αυτοκίνητο, στο δεύτερο υπάρχει η λογική της τεχνητής νοημοσύνης και της λήψης αποφάσεων για το πως θα δράσει το αυτοκίνητο και στο τρίτο περιγράφεται η λογική και οι λειτουργίες που περιγράφουν αυτό το αυτοκίνητο. Το τελευταίο αρχείο είναι το αρχείο διαμόρφωσης που περιέχει τις σχετικές πληροφορίες που χρειάζεται η τεχνητή νοημοσύνη.

Αμέσως μετά δημιουργείται ένας φάκελος με όνομα environment, ο οποίος χαρακτηριστικά περικλείει όλα τα αρχεία που δίνουν λειτουργικότητα στο περιβάλλον της εργασίας. Σε αυτό το φάκελο δημιουργούνται τα αρχεία colors.py και environment.py. Αντιστοίχως, στο πρώτο αρχείο υπάρχουν συγκεντρωμένα τα χρώματα που θα χρησιμοποιηθούν, καθώς στο δεύτερο υπάρχει το σύνολο των λειτουργιών για την σχεδίαση του περιβάλλοντος.

Επίσης δημιουργείται ο φάκελος με όνομα NN, που είναι συντομογραφία για Neural Network. Σε αυτό το φάκελο θα αποθηκεύονται τα εκπαιδευμένα νευρωνικά δίκτυα κατά τη διάρκεια της προσομοίωσης, μετά την πάροδο ενός αριθμού γενεών.

Τέλος δημιουργείται το αρχείο main.py που θα καλείται για την εκκίνηση της εργασίας. Στο παρακάτω σχέδιο αναπαρίσταται η δομή των φακέλων και των αρχείων που αναφέρθηκαν.

AI_CARS

├── assets/

├── car/

│ ├── actions.py

│ ├── car_ai.py

| ├── car.py

| └── config.txt

├── environment/

| └── colors.py

| └── environment.py

├── main.py

├── NN/

└──

1.1.2 Διάγραμμα ροής

Τα διαγράμματα ροής απεικονίζουν τις αλληλεπιδράσεις μεταξύ διαφορετικών στοιχείων ή αντικειμένων μέσα στο σύστημα με την πάροδο του χρόνου. Δείχνουν τη ροή της λογικής και των ακολουθιακών λειτουργιών, μαζί με τη σειρά εκτέλεσης. Τα διαγράμματα ακολουθίας είναι ιδιαίτερα χρήσιμα για την κατανόηση της συμπεριφοράς του συστήματος κατά τη διάρκεια της σχεδίασης και ανάπτυξης λογισμικών, βοηθώντας στον εντοπισμό πιθανών λαθών που μπορούν να προκύψουν κατά τη διάρκεια της ανάπτυξης αλλά και της εκτέλεσης ενός έργου. Έχοντας αναφέρει τα παραπάνω, θα ακολουθήσει αναλυτικά η ακολουθιακή λογική που θα χρησιμοποιηθεί ως γνώμονας για την ολοκλήρωση της εργασίας.

Σκοπός της συγκεκριμένης εργασίας, είναι η εκπαίδευση ενός αυτοκινήτου να κινείται αυτόνομα μέσα σε μία πίστα. Από τα παραπάνω μπορεί να γίνει η διαπίστωση ότι χρειάζεται ένα όχημα, ένα ικανό περιβάλλον, καθώς και μια τεχνητή νοημοσύνη που εκπληρώνει την αυτονομία κινήσεων.

Με τα παραπάνω ως στοιχεία, αρχικά πρέπει το περιβάλλον να αρχικοποιηθεί καθώς είναι το πρώτο στάδιο της ακολουθίας. Χωρίς αυτό, το αυτοκίνητο δεν έχει λόγο να βρίσκεται σε ενεργή κατάσταση. Έτσι σε δεύτερο στάδιο αρχικοποιείται το αυτοκίνητο μέσα στο περιβάλλον.

Αφού έχει τελειώσει αυτή η διαδικασία μπορεί να ακολουθήσει η εκκίνηση της τεχνητής νοημοσύνης. Αυτή η διαδικασία πρέπει να ενεργοποιηθεί μετά την ολοκλήρωση ενεργοποίησης του αυτοκινήτου, αλλά ακριβώς πριν ακολουθήσει η ενεργοποίηση της προσομοίωσης έτσι ώστε να αποφευχθεί η λήψη αποφάσεων από την τεχνητή νοημοσύνη πριν την προσομοίωση. Η λειτουργία αυτή θα πρέπει να διαρκέσει έως την στιγμή που θα ξεπεραστεί ο χρόνος διάρκειας της προσομοίωσης.

Αμέσως μετά την εκκίνηση της προσομοίωσης, το αυτοκίνητο βρίσκεται πλέον σε κάποια ενεργή κατάσταση στο περιβάλλον. Το επόμενο στάδιο της ακολουθίας είναι η συλλογή δεδομένων από το αυτοκίνητο για να τροφοδοτήσει την τεχνητή νοημοσύνη. Η

τεχνητή νοημοσύνη επεξεργάζεται τα δεδομένα και στην συνέχεια λαμβάνει την απόφαση επόμενης κίνησης του αυτοκινήτου.

Αφού έχει ληφθεί η απόφαση κίνησης, το αυτοκίνητο ανταποκρίνεται σε αυτή. Κατά αυτό τον τρόπο το αυτοκίνητο βρίσκεται σε κάποια διαφορετική ενεργή κατάσταση και έτσι επαναλαμβάνονται τα προηγούμενα βήματα έως την λήξη λειτουργίας της προσομοίωσης.

Στην παρακάτω εικόνα, φαίνεται το ακολουθιακό διάγραμμα ροής που περιγράφηκε.

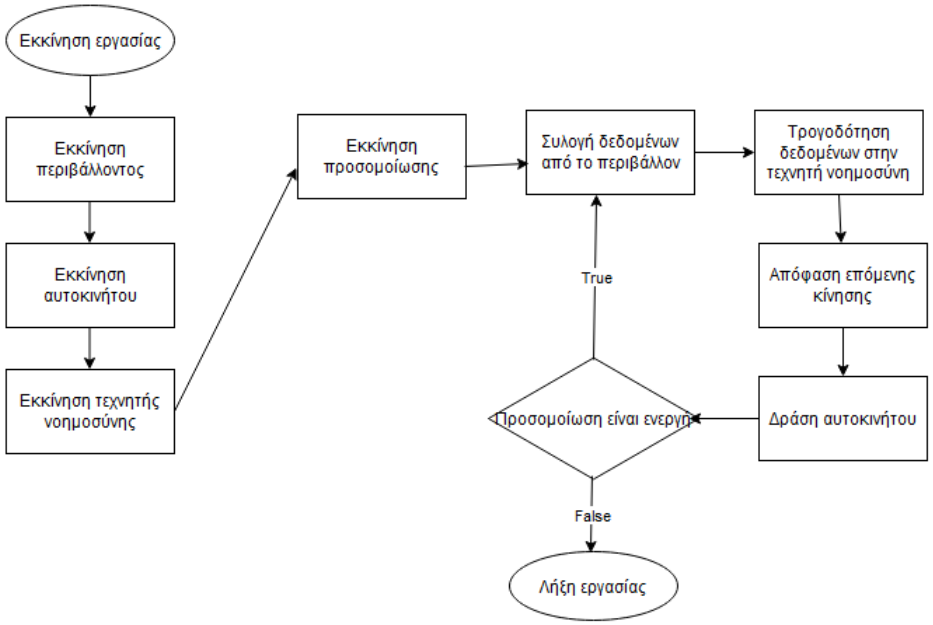

Figure 1 Διάγραμμα ροής

1.2 Λήψη και εγκατάσταση εργαλείων

Μετά από έρευνα για το πως θα υλοποιηθεί ένα τέτοιο σενάριο σε ένα γραφικό περιβάλλον για την καλύτερη κατανόηση του, λήφθηκε η απόφαση να χρησιμοποιηθούν τα παρακάτω εργαλεία. Αρχικά το πρώτο εργαλείο που θα χρησιμοποιηθεί ονομάζεται Visual Studio Code και είναι αυτό που θα χρησιμοποιηθεί για την ανάπτυξη της λογικής σε μορφή κώδικα. Το Visual Studio Code (VS Code) είναι ένα δωρεάν πρόγραμμα επεξεργασίας πηγαίου κώδικα που αναπτύχθηκε από τη Microsoft για λειτουργικά συστήματα όπως Windows, Linux και macOS. Το VS Code προσφέρει λειτουργίες όπως επισήμανση σύνταξης, συμπλήρωση κώδικα, δυνατότητες εντοπισμού σφαλμάτων, ενσωμάτωση ελέγχου έκδοσης (με Git) και μια εκτεταμένη αγορά για επεκτάσεις (extensions) που μπορούν να προσαρμόσουν και να βελτιώσουν τη λειτουργικότητά του σύμφωνα με τις ανάγκες του χρήστη. Ένα από τα βασικά χαρακτηριστικά του είναι ο ελαφρύς του χαρακτήρας σε συνδυασμό με ισχυρές δυνατότητες, γεγονός που το καθιστά δημοφιλή επιλογή μεταξύ των προγραμματιστών

για διάφορες εργασίες προγραμματισμού, από ένα απλό σενάριο έως έργα ανάπτυξης λογισμικού μεγάλης κλίμακας. Επίσης ένα σημαντικό χαρακτηριστικό του είναι πως, διανέμεται δωρεάν χωρίς καμία μεταγενέστερη χρηματική επιβάρυνση.

Το δεύτερο εργαλείο που θα χρησιμοποιηθεί είναι η γλώσσα προγραμματισμού Python. Η επιλογή της γλώσσας δεν είναι τυχαία, καθώς είναι μια υψηλού επιπέδου, ευέλικτη γλώσσα προγραμματισμού γνωστή για την απλότητα και την αναγνωρισιμότητά της. Έχει αποκτήσει τεράστια δημοτικότητα λόγω της ευκολίας εκμάθησης, το εκτεταμένο εύρος βιβλιοθηκών και της ζωντανής υποστήριξης της κοινότητας της.

Τα τελευταία εργαλεία που θα χρησιμοποιηθούν είναι δύο βιβλιοθήκες σχεδιασμένες για την Python, η Pygame και η NEAT. Η βιβλιοθήκε Pygame είναι ένα σύνολο λειτουργικών μονάδων Python που έχουν σχεδιαστεί για τη σύνταξη βιντεοπαιχνιδιών. Παρέχει λειτουργικότητα για τον χειρισμό γραφικών, ήχου, συσκευών εισόδου όπως πληκτρολόγια, ποντίκια. Ένα από τα κύρια χαρακτηριστικά της βιβλιοθήκης Pygame είναι ότι, επιτρέπει την εύκολη δημιουργία γραφικών, μέσω της σχεδίασης σχημάτων και εικόνων στην οθόνη. Επίσης παρέχει την δυνατότητα χειρισμού συμβάντων που μπορεί να δεχτεί ως είσοδος από έναν χρήστη.

Η βιβλιοθήκη NEAT (NeuroEvolution of Augmenting Topologies) που θα χρησιμοποιηθεί, είναι ένας γενετικός αλγόριθμος ειδικά σχεδιασμένος για εξελικτικά νευρωνικά δίκτυα. Ο αλγόριθμος NEAT λειτουργεί ξεκινώντας με έναν πληθυσμό απλών νευρωνικών δικτύων, συνήθως με πολύ λίγους νευρώνες και συνδέσεις. Αυτά τα δίκτυα στη συνέχεια εξελίσσονται με την πάροδο των γενεών μέσω διαδικασιών όπως η μετάλλαξη, η διασταύρωση και η επιλογή. Ο NEAT εισάγει καινοτόμους μηχανισμούς για την εξέλιξη τόσο της δομής όσο και των βαρών των νευρωνικών δικτύων, επιτρέποντας την εμφάνιση πολύπλοκων δικτύων που μπορούν να επιλύσουν διάφορες εργασίες ή προβλήματα. Η βιβλιοθήκη Python NEAT περιλαμβάνει λειτουργικότητα για τη δημιουργία γονιδιωμάτων νευρωνικών δικτύων, τη διαχείριση πληθυσμών δικτύων, την εφαρμογή γενετικών λειτουργιών όπως η μετάλλαξη και η διασταύρωση και η διευκόλυνση της διαδικασίας εξέλιξης [1].

1.2.1 Λήψη και εγκατάσταση Visual Studio Code

Το Visual Studio Code είναι ένας κειμενογράφος στον οποίο μπορούμε να γράψουμε προγράμματα χρησιμοποιώντας γλώσσες προγραμματισμού όπως η Python και διανέμεται εντελώς δωρεάν από την Microsoft. Ο λόγος για τον οποίο επιλέχθηκε ο παρών κειμενογράφος σε αντίθεση με ένα ολοκληρωμένο περιβάλλον ανάπτυξης προγραμμάτων όπως το Visual Studio, είναι ότι η χρήση του και η ρύθμιση του είναι αρκετά απλή αλλά και πολύ μικρό σε μέγεθος. Το προς λήψη μπορεί να βρεθεί στον παρακάτω ιστότοπο (https://code.visualstudio.com). Με την ολοκλήρωση της λήψης μπορεί να ακολουθήσει η εγκατάσταση του εργαλείου ακολουθώντας τα απλά βήματα.

1.2.2 Λήψη Και εγκατάσταση Paint 3D

Το συγκεκριμένο εργαλείο βρίσκεται προεγκατεστημένο στους υπολογιστές με λειτουργικό σύστημα Windows 10. ‘Ομως σε περίπτωση που δεν υπάρχει, μπορεί να γίνει λήψη δωρεάν από τη επίσημη σελίδα της Microsoft

(https://apps.microsoft.com/store/detail/paint-3d/9NBLGGH5FV99). Μετά την λήψη του εκτελέσιμου αρχείου μπορεί να γίνει η εγκατάση του ακολουθώντας τα σχετικά βήματα.

1.2.3 Λήψη και εγκατάσταση Python και βιβλιοθηκών

Η Python είναι μια αντικειμενοστραφής γλώσσα προγραμματισμου και είναι αυτή που θα χρησιμοποιηθεί για την επίτευξη αυτής της εργασίας. Για να γίνει λήψη της θα πρέπει να γίνει μετάβαση στον παρακάτω ιστότοπο και να επιλεχθεί μία από τις εκδόσεις. Προτεινόμενες εκδόσεις είναι από την έκδοση 3.6 έως την έκδοση 3.11 που είναι και η τελευταία κατά την περίοδο συγγραφής της παρούσας εργασίας. Μετά την ολοκλήρωση της λήψης μπορεί να ακολουθήσει η εγκατάσταση της Python ανοίγοντας το εκτελέσιμο αρχείο. Πολύ σημαντικό κατά την εγκατάσταση είναι η επιλογή των πεδίων που φαίνονται στην παρακάτω εικόνα.

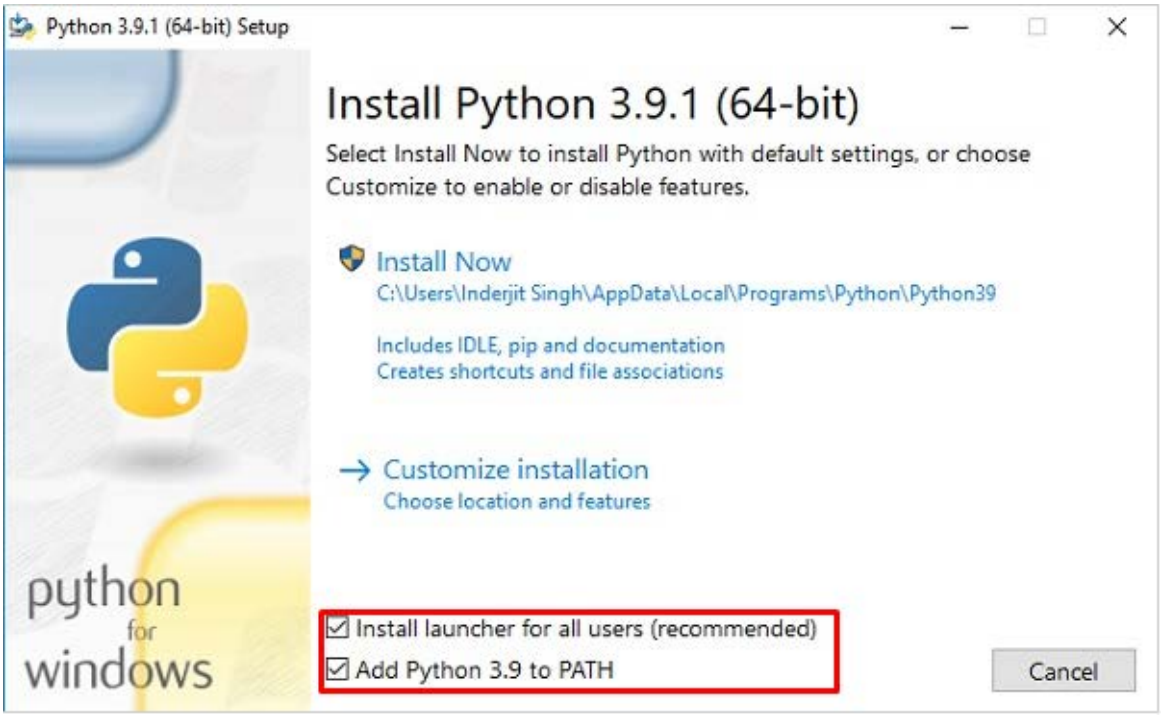

Figure 2 Εγκατάσταση Python

Με την ολοκλήρωση αυτής της διαδικασίας θα έχει εγκατασταθεί επιτυχώς η γλώσσα προγραμματισμού Python στον υπολογιστή, καθώς και το σύστημα διαχείρισης πακέτων της Python που ονομάζεται pip. Με το pip δίνεται η δυνατότητα εγκατάστασης περισσότερων πακέτων επέκτασης για την Python στον χρήστη και θα χρησιμοποιηθεί για την εγκατάσταση των βιβλιοθηκών Pygame και NEAT. Σε αυτό το σημείο θα πρέπει να δημιουργηθεί ο φάκελος στον οποίο θα υλοποιηθεί το τελικό πρόγραμμα της

εργασίας έτσι ώστε να μπορέσει να γίνει λήψη και εγκατάσταση των παραπάνω βιβλιοθηκών. Οι συγκεκριμένες βιβλιοθήκες δεν υπάρχουν σε μορφή απλού εκτελέσιμου αρχείου και για αυτό τον λόγο θα πρέπει να γίνει χρήση της γραμμής εντολών του Windows για ανακατεύθυνση στον φάκελο που δημιουργήθηκε, για την λήψη και την εγκατάσταση τους. Για να γίνει αυτό θα πρέπει να χρησιμοποιηθεί η εντολή "CD" μαζί με την διαδρομή του φακέλου όπως για παράδειγμα "cd C:\Users\Theodore\Desktop\test".

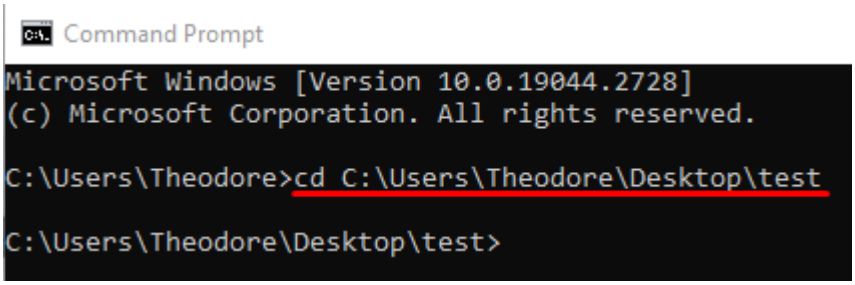

Figure 3 Αλαγή φακέλου

Αφού βρισκόμαστε στο φάκελο που θα υλοποιηθεί το τελικό πρόγραμμα της εργασίας, θα πρέπει να δημιουργηθεί ένα αρχείο τύπου Python έτσι ώστε να μπορέσει να γίνει λήψη και εγκατάσταση των λοιπών βιβλιοθηκών. Για να δημιουργηθεί ένα τέτοιο αρχείο θα πρέπει να χρησιμοποιηθεί η εντολή "type NUL > main.py" όπου δημιουργεί ένα κενό αρχείο με όνομα main και είναι τύπου Python. Στη συνέχεια για τη βιβλιοθήκη Pygame θα χρησιμοποιηθεί η εντολή "py -m pip install -U pygame --user" και σύντομα μετά την εκτέλεση της θα έχει εγκατασταθεί η βιβλιοθήκη όπως φαίνεται στην παρακάτω εικόνα.

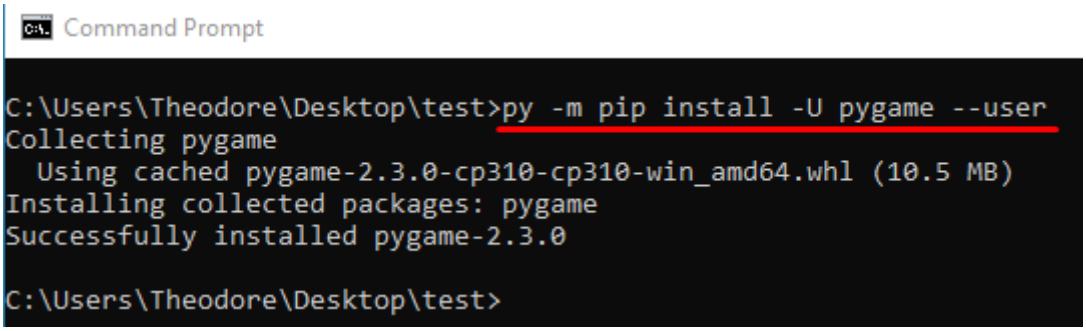

Figure 4 Εγκατάσταση Pygame

Ομοίως για τη βιβλιοθήκη NEAT θα χρησιμοποιηθεί η εντολή "pip install neat-python" και σύντομα μετά την εκτέλεση της θα έχει εγκατασταθεί η βιβλιοθήκη όπως φαίνεται στην παρακάτω εικόνα.

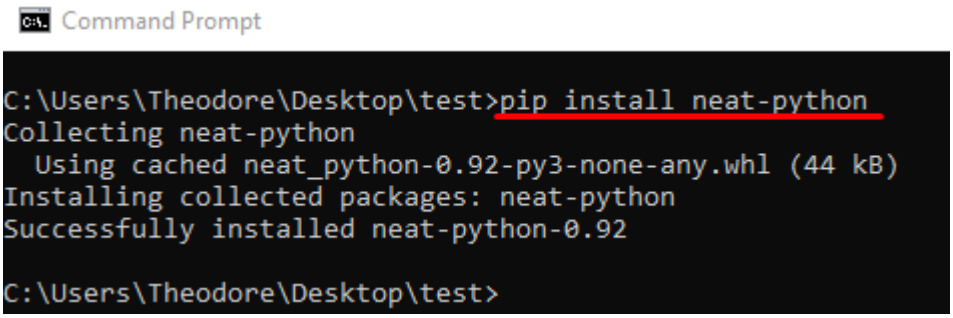

Figure 5 Εγκατάσταση NEAT

Μετά από τα παραπάνω βήματα έχουν εγκατασταθεί επιτυχώς όλα τα εργαλεία που θα χρησιμοποιηθούν για την παρούσα εργασία.

1.3 Πρωτότυπο περιβάλλοντος και οχήματος

1.3.1 Πρωτότυπο περιβάλλοντος

Η δημιουργία ενός σωστού περιβάλλοντος είναι το πιό σημαντικό στάδιο της συγκεκριμένης εργασίας. Το συγκεκριμένο περιβάλλον που πρέπει να δημιουργηθεί, θα πρέπει να μπορεί να στεγάσει μια τεχνητή νοημοσύνη που θα εκπαιδεύει ένα όχημα να κινείται σε δρόμο. Από αυτή την εκφώνηση δημιουργούνται κάποιες απαιτήσεις και σύμφωνα με αυτές θα πρέπει να δομηθεί το περιβάλλον. Η πρώτη και πιο ξεκάθαρη απαίτηση είναι να υπάρχει δρόμος για να κινηθεί ένα όχημα. Αμέσως μετά, μια σημαντική απαίτηση είναι το μέγεθος του δρόμου στη διάσταση του πλάτους, καθώς σε έναν πολύ στενό δρόμο το όχημα μπορεί να βγαίνει εκτός των ορίων πολύ εύκολα ή το όχημα να κινείται εξαρχής εκτός ορίων. Επίσης μια σημαντική απαίτηση είναι το σχήμα του δρόμου, καθώς αυτό είναι που θα καθορίσει την ορθότητα αλλα και την αποτελεσματικότητα του περιβάλλοντος. Ένας ευθύς δρόμος μπορεί να θεωρείται δρόμος, αλλά στην προκειμένη περίπτωση δεν θα δώσει τη δυνατότητα στη τεχνητή νοημοσύνη να εκπαιδευτεί στο να στρίβει, καθιστώντας το περιβάλλον λανθασμένο. Για αυτό το λόγο ένα σωστό και αποδοτικό περιβάλλον θα πρέπει να έχει αρκετές στροφές με φορά στα αριστερά αλλά και προς τα δεξιά. Έτσι σύμφωνα με τις παραπάνω παρατηρήσεις μπορεί να δημιουργηθεί ένα περιβάλλον ώστε να υπάρχει ένα οπτικό προσχέδιο για αναφορά.

Στην παρακάτω εικόνα φαίνεται πως μπορεί να δείχνει ένα περιβάλλον.

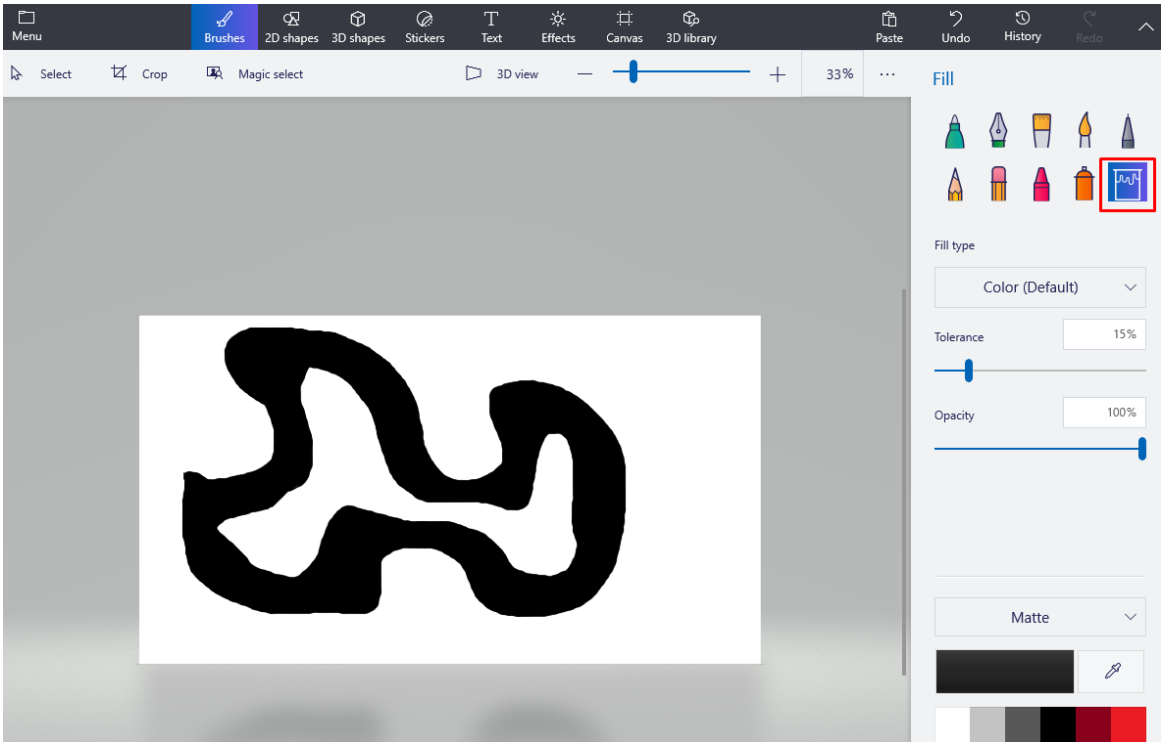

Figure 6 Πρωτότυπο περιβάλλοντος

1.3.2 Πρωτότυπο οχήματος

Μετά την ολοκλήρωση του σχεδιασμού του περιβάλλοντος, θα πρέπει να γίνει ο σχεδιασμός ενός οχήματος έτσι ώστε να μπορεί να γίνει η γραφική του αναπαράσταση πάνω στο περιβάλλον. Επίσης το όχημα θα αναπαριστά γραφικά έναν πράκτορα ο οποίος θα αλληλοεπιδρά με το περιβάλλον. Έχοντας αυτά υπόψη, πρέπει να δημιουργηθεί ένα όχημα που να καλύπτει τις ανάγκες της εργασίας.

Το βασικότερο κριτήριο για τον σχεδιασμό του οχήματος είναι το μέγεθος του οχήματος στη διάσταση του μήκους, καθώς θα είναι πιο εύκολο στο σχεδιασμό του, τόσο στη γραφική του αναπαράσταση αλλά και στην ανάπτυξη της λογικής που θα χρειαστεί. Έτσι λήφθηκε η απόφαση ώστε το όχημα να αναπαριστά ένα απλό επιβατικό όχημα τεσσάρων τροχών, όσον αφορά τις διαστάσεις του.

Για να επιτευχθεί το ζητούμενο θα χρησιμοποιηθεί το εργαλείο Paint 3D ομοίως όπως έγινε και για το περιβάλλον. Ανοίγοντας το εργαλείο και επιλέγοντας την καρτέλα “2D shapes” μπορεί να χρησιμοποιηθεί το τετράγωνο σχήμα ώστε να σχεδιαστεί ένα ορθογώνιο όπως στην παρακάτω εικόνα.

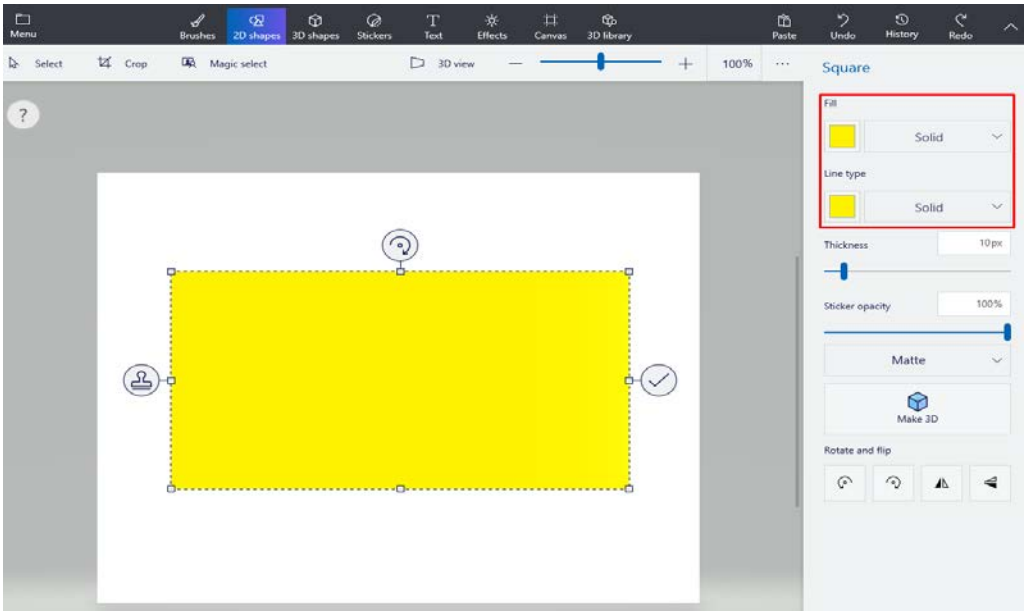

Figure 7 Αρχικό σχέδιο αυτοκινήτου

Στη συνέχεια, για να εμφανίζεται αποκλειστικά και μόνο το ορθογώνιο σχήμα, χωρίς τον καμβά στο παρασκήνιο, θα πρέπει να ενεργοποιηθεί η επιλογή “transparent canvas” στην καρτέλα “Canvas” όπως φαίνεται στην παρακάτω εικόνα.

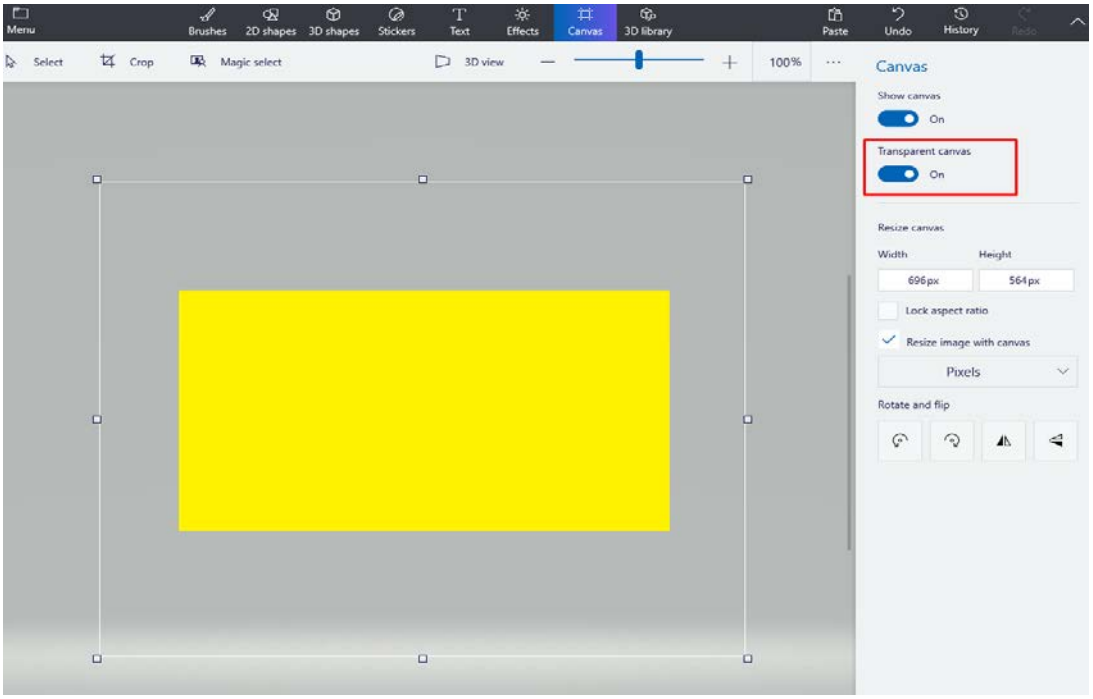

Figure 8 Τελικό σχέδιο αυτοκινήτου

Αφού έχει τελειώσει η σχεδίαση του οχήματος, θα πρέπει να αποθηκευτεί το συγκεκριμένο σχήμα σε ένα αρχείο με ένα όνομα και είδους αρχείου “png” όπως στην παρακάτω εικόνα.

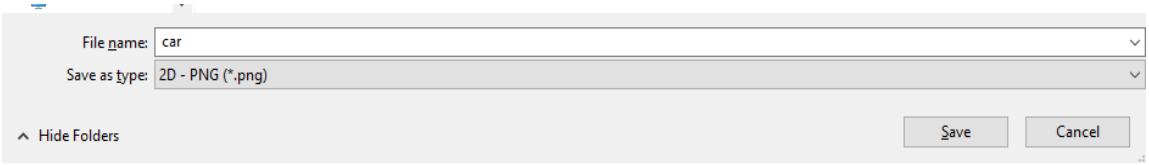

Figure 9 Σώσιμο αυτοκινήτου

Κατά αυτόν τον τρόπο έχει ολοκληρωθεί η διαδικασία σχεδιασμού οχήματος με επιτυχία και σε αυτό το σημείο είναι επιθυμητό να μεταφερθεί η εικόνα του οχήματος στον εκάστοτε φάκελο που θα στεγάσει τη λογική του προγράμματος και συγκεκριμένα στον φάκελο assets που έχει δημιουργηθεί που βρίσκεται στη διαδρομή “C:\Users\Theodore\Desktop\AI_Cars\assets”

2. Ανάπτυξη Εργασίας

2.1 Χαρακτηριστικά Εργασίας

2.1.1 Χαρακτηριστικά οχήματος

Τα χαρακτηριστικά του οχήματος που έχουν διακριθεί για τον συγκεκριμένο σκοπό είναι τα εξής τρία.

Το πρώτο χαρακτηριστικό είναι η ένταξη αισθητήρων έτσι ώστε το όχημα να έχει τη δυνατότητα αντίληψης του περιβάλλοντος. Οι αισθητήρες θα λειτουργούν ως ραντάρ για το όχημα τροφοδοτώντας το με πληροφορίες σχετικά με την κατάσταση του περιβάλλοντος κάθε στιγμή που περνά.

Το δεύτερο χαρακτηριστικό είναι συλλογή δεδομένων. Η συλλογή δεδομένων θα γίνεται μέσω των αισθητήρων και θα χρησιμοποιείται από την τεχνητή νοημοσύνη για να αντιλαμβάνεται τα πλαίσια του δρόμου και τα πλαίσια έξω από αυτόν.

Το τελευταίο χαρακτηριστικό είναι η επεξεργασία των δεδομένων. Κατά αυτόν τον τρόπο το όχημα θα μπορεί να λάβει μια απόφαση κίνησης που θα μπορεί να εκτελέσει ανάλογα με τη σχετική του θέση στο περιβάλλον.

Έτσι με αυτά τα χαρακτηριστικά ως γνώμονα μπορεί να υλοποιηθεί η λογική του οχήματος με μεγαλύτερη σαφήνεια.

2.1.2 Χαρακτηριστικά τεχνητής νοημοσύνης

Στη συγκεκριμένη εργασία, ο σκοπός της τεχνητής νοημοσύνης είναι να μπορεί να εκπαιδεύσει το όχημα να κινείται αυτόβουλα μέσα στα πλαίσια ενός δρόμου. Αυτό όμως δεν θα μπορεί να το επιτύχει εξ αρχής καθώς δεν θα υπάρχει προηγούμενη εμπειρία στην οποία να έχει πρόσβαση. Για αυτό τον λόγο, το κυρίως χαρακτηριστικό της τεχνητής νοημοσύνης θα είναι η δυνατότητα της να εξελίσσεται με σκοπό τη αποδοτικότερη επιτυχία του στόχου της.

Ως αλγόριθμος εξέλιξης θα χρησιμοποιηθεί η βιβλιοθήκη της Python NEAT που έχει ήδη εγκατασταθεί στο φάκελο της εργασίας. Η βιβλιοθήκη NEAT είναι μια δημοφιλής βιβλιοθήκη για την εφαρμογή αλγορίθμων νευροεξέλιξης, οι οποίοι είναι μια τεχνική μηχανικής μάθησης που περιλαμβάνει την εξέλιξη τεχνητών νευρωνικών δικτύων μέσω μιας διαδικασίας μετάλλαξης και επιλογής, προκειμένου να βελτιστοποιηθούν για μια δεδομένη εργασία.

2.1.3 Χαρακτηριστικά γραφικής αναπαράστασης

Η γραφική αναπαράσταση είναι ένα πολύ σημαντικό μέρος αυτής της εργασίας καθώς είναι ο τρόπος που θα μπορούν να ληφθούν οπτικά αποτελέσματα της προόδου του προγράμματος. Το βασικό χαρακτηριστικό είναι ότι θα πρέπει να υπάρχει εικόνα

του τι συμβαίνει κάθε στιγμή που το πρόγραμμα θα βρίσκεται σε λειτουργία και για την υλοποίηση αυτή θα χρησιμοποιηθεί η βιβλιοθήκη της Python Pygame που έχει εγκατασταθεί. Η βιβλιοθήκη Pygame είναι ένα σύνολο λειτουργικών μονάδων της γλώσσας Python που έχουν σχεδιαστεί για τη σύνταξη βιντεοπαιχνιδιών. Παρέχει στους προγραμματιστές τη δυνατότητα δημιουργίας δισδιάστατων παιχνιδιών και εφαρμογών πολυμέσων καθώς περιλαμβάνει λειτουργικότητα για χειρισμό γραφικών και κινούμενων εικόνων.

Για την συγκεκριμένη εργασία, το περιβάλλον που θα σχεδιαστεί, πρέπει να πληροί μερικές προϋποθέσεις έτσι ώστε αυτό να καθίσταται κατάλληλο. Το βασικό χαρακτηριστικό είναι ότι ο δρόμος που θα σχεδιαστεί, πρέπει να είναι ατέρμονος δηλαδή, η αρχή του δρόμου να ενώνεται με το τέλος του. Αυτό σημαίνει ότι ο δρόμος πρέπει να είναι κυκλικός. Το δεύτερο σημαντικό χαρακτηριστικό είναι ότι ο δρόμος και τα όρια του, πρέπει να είναι ευδιάκριτα, τόσο στο ανθρώπινο μάτι όσο και στην τεχνητή νοημοσύνη που θα δρα σε αυτό. Για αυτό τον λόγο έχει αποφασιστεί ότι, ο δρόμος θα έχει μαύρο χρώμα καθώς το υπόλοιπο περιβάλλον θα έχει το χρώμα άσπρο. Κατά αυτό τον τρόπο μπορεί με ευκολία να διακριθεί το πλαίσιο του δρόμου, καθώς και τα όρια του.

2.2 Ανάπτυξη λογικής οχήματος

Σύμφωνα με τα παραπάνω χαρακτηριστικά που έχουν αναλυθεί, θα αναπτυχθεί η λογική του οχήματος με τη γλώσσα προγραμματισμού Python. Για να ξεκινήσει η διαδικασία θα πρέπει να εκκινηθεί το εργαλείο Visual Studio Code στον φάκελο που έχει δημιουργηθεί για την συγκεκριμένη εργασία, που θα χρησιμοποιηθεί ώς κειμενογράφος. Στην καρτέλα Explorer του Visual Studio Code πρέπει να δημιουργηθεί ένα καινούργιο αρχείο με όνομα “car.py”, στο οποίο θα αναπτυχθεί μια κλάση Car που θα χαρακτηρίζει το όχημα.

Στην κλάση Car θα εισαχθούν δύο βιβλιοθήκες, θα αρχικοποιηθούν μερικές σταθερές και θα αναπτυχθούν μερικοί μέθοδοι που θα δώσουν τις παρακάτω λειτουργίες στο όχημα της εργασίας.

- init(): Αυτή η μέθοδος αρχικοποιεί την κλάση Car με κάποιες μεταβλητές που το χαρακτηρίζουν.
- draw_radars(): Αυτή η μέθοδος σχεδιάζει τους αισθητήρες στην οθόνη.
- draw(): Αυτή η μέθοδος εμφανίζει την εικόνα του οχήματος και τα ραντάρ στην οθόνη.
- out_of_bounds(): Αυτή η μέθοδος ελέγχει εάν το όχημα έχει βγει εκτός των ορίων της πίστας.
- distance_to_border(): Αυτή η μέθοδος ελέγχει ποια είναι η απόσταση του οχήματος απο τα όρια της πίστας.
- calculate_corner(): Αυτή η μέθοδος υπολογίζει τα σημεία των γωνιών του οχήματος.
- refresh_corners_positions(): Αυτή η μέθοδος υπολογίζει και ανανεώνει τα σημεία των γωνιών του οχήματος, όσο αυτό κινείται.
- check_collision(): Αυτή η μέθοδος ελέγχει εάν το όχημα έχει συγκρουστεί με τα πλαίσια εκτός του δρόμου στον εκάστοτε χάρτη.
- check_radar(): Αυτή η μέθοδος ελέγχει την απόσταση των αισθητήρων του αυτοκινήτου από το πλησιέστερο όριο του δρόμου στον εκάστοτε χάρτη.

- get_data(): Αυτή η μέθοδος αποθηκεύει τα δεδομένα που δέχονται οι αισθητήρες, υπολογίζει την απόσταση οχήματος σε σχέση με τα όρια του δρόμου και επιστρέφει την τιμή.
- get_reward(): Αυτή η μέθοδος υπολογίζει και επιστρέφει μια σχετική επιβράβευση για το όχημα.
- update_car_center(): Αυτή η μέθοδος περιστρέφει την εικόνα του οχήματος και επιστρέφει την περιστρεφόμενη εικόνα. Η περιστροφή γίνεται γύρω από το κέντρο της εικόνας.
- accelerate(): Αυτή η μέθοδος αυξάνει την ταχύτητα του οχήματος.
- brake(): Αυτή η μέθοδος μειώνει την ταχύτητα του οχήματος.
- turn_left(): Αυτή η μέθοδος αλλάζει την διεύθυνση του οχήματος προς τα αριστερά.
- turn_right(): Αυτή η μέθοδος αλλάζει την διεύθυνση του οχήματος προς τα δεξιά.
- update_car_state(): Αυτή η μέθοδος ενημερώνει τη θέση του αυτοκινήτου, ελέγχει για συγκρούσεις και ενημερώνει τις ενδείξεις των ραντάρ.

2.2.1 Δήλωση βιβλιοθηκών, σταθερών και κλάσης Car

Ξεκινώντας την ανάπτυξη του οχήματος θα εισαχθούν πρώτα δύο βιβλιοθήκες που θα χρειαστούν στην πορεία. Οι δύο αυτές είναι η βιβλιοθήκη Math που περιέχει μεθόδους για μαθηματικές πράξεις και η βιβλιοθήκη Pygame που εμπεριέχει μεθόδους για την γραφική αναπαράσταση του οχήματος. Επίσης εισάγουμε το αρχείο με τα χρώματα τα οποία θα χρησιμοποιηθούν αργότερα

Στη συνέχεια θα αρχικοποιηθεί η σταθερά που θα χρησιμοποιηθεί για την εικόνα του οχήματος, με όνομα CAR_SPRITE_PATH. Αυτή η σταθερά αρχικοποιείται με τη σχετική διαδρομή προς το φάκελο που έχει αποθηκευτεί η εικόνα του οχήματος.

Αμέσως μετά θα δημιουργηθεί η κλάση Car στην οποία θα αρχικοποιηθούν οι σταθερές που το προσδιορίζουν, καθώς θα αναπτυχθούν οι μέθοδοι που θα δώσουν λειτουργικότητα στο όχημα.

Αρχικά, δηλώνονται δύο σταθερές που προσδιορίζουν τις διαστάσεις του οχήματος στον άξονα X και Y. Αυτές οι σταθερές ονομάζονται CAR_SIZE_X και CAR_SIZE_Y, οι οποίες αρχικοποιούνται με τον ακέραιο αριθμό εξήντα (60), που προσδιορίζει τα pixel που καταλαμβάνει, έτσι ώστε αυτό να έχει μια τετράγωνη μορφή.

Στη συνέχεια, δηλώνονται τρείς σταθερές που προσδιορίζουν την ταχύτητα του οχήματος. Αυτές ονομάζονται DEFAULT_SPEED, MINIMUM_SPEED και SPEED_INCREMENT, καθώς αρχικοποιούνται αντιστοίχως με ακέραιες τιμές, οι δύο πρώτες με την τιμή δέκα (10) και η τρίτη με την τιμή ένα (1).

Στη συνέχεια, δηλώνεται μια σταθερά που προσδιορίζει την ταχύτητα περιστροφής του οχήματος όταν αυτό στρίβει προς μία κατεύθυνση. Αυτή ονομάζεται ANGLE_INCREMENT και αρχικοποιείται με την τιμή δέκα (10).

Αμέσως μετά, δηλώνεται μια σταθερά που προσδιορίζει τον συνολικό αριθμό μοιρών σε έναν κύκλο και ονομάζεται FULL_CIRCLE_DEGREES. Αυτή η σταθερά αρχικοποιείται με την ακέραια τιμή τριακόσια εξήντα (360).

Τέλος, θα δηλωθούν μερικές σταθερές που θα προσδιορίζουν τα ραντάρ του οχήματος που θα λειτουργούν ως αισθητήρες για την συλλογή δεδομένων. Η σταθερά RADAR_DRAW_DISTANCE προσδιορίζει τη μέγιστη απόσταση που μπορεί να δει ένα ραντάρ και αρχικοποιείται με την τιμή χίλια (1000). Οι σταθερές RADAR_START_ANGLE και RADAR_END_ANGLE που προσδιορίζουν τις γωνίες έναρξης και λήξης του οπτικού πεδίου του ραντάρ, αρχικοποιούνται αντιστοίχως τις τιμές μείον ενενήντα (-90) και ενενήντα (90). Η σταθερά RADAR_STEP_ANGLE που προσδιορίζει τη γωνία μεταξύ κάθε ραντάρ στο οπτικό πεδίο και αρχικοποιείται με την τιμή σαράντα πέντε (45). Τέλος οι σταθερές RADAR_LINE και RADAR_CIRCLE προσδιορίζουν αντίστοιχα το πάχος της γραμμής του ραντάρ που θα σχεδιάζεται από το όχημα, μέχρι εκεί που θα καταλήγει καθώς και η διάμετρος του κύκλου που θα σχεδιάζεται στο σημείο που καταλήγει η γραμμή του ραντάρ, για την καλύτερη απεικόνιση. Αυτές οι σταθερές θα αρχικοποιηθούν αντιστοίχως με τις ακέραιες τιμές δύο (2) και τέσσερα (4).

Ο τρόπος σύνταξης φαίνεται στη παρακάτω εικόνα.

```
1   import pygame
2   import math
3   from environment.colors import Color
4
5   CAR_SPRITE_PATH = "assets/car.png"
6   DEAD_CAR_SPRITE_PATH = "assets/dead_car.png"
7
8
    Theodore Charalampidis, 2 weeks ago | 2 authors (You and others)
9   class Car:
10
11      CAR_SIZE_X = 60
12      CAR_SIZE_Y = 60
13      DEFAULT_SPEED = 10
14      MINIMUM_SPEED = 10
15      SPEED_INCREMENT = 1
16      ANGLE_INCREMENT = 10
17      FULL_CIRCLE_DEGREES = 360
18      RADAR_DRAW_DISTANCE = 1000
19      RADAR_START_ANGLE = -90
20      RADAR_END_ANGLE = 90
21      RADAR_STEP_ANGLE = 45
22      RADAR_LINE = 2
23      RADAR_CIRCLE = 4
```

Figure 10 Κλάση Car

2.2.2 Δήλωση μεθόδου init()

Η πρώτη μέθοδος που θα αναλυθεί είναι η μέθοδος init(). Αυτή είναι μια μέθοδος καταστεκαυστής και είναι υπεύθυνη για την αρχικοποίηση τιμών στις μεταβλητές που προσδιορίζουν το όχημα. Η δεσμευμένη λέξη self που χρησιμοποιείται είναι μια παράμετρος που κάνει αναφορά στην τρέχουσα παρουσία της κλάσης και χρησιμοποιείται για την πρόσβαση σε μεταβλητές που ανήκουν στην κλάση. Η δεύτερη παράμετρος που χρησιμοποιείται είναι η start_position, η οποία είναι μια λίστα που αντιπροσωπεύει την αρχική θέση του οχήματος. Η αρχική θέση είναι μια λίστα δύο στοιχείων, όπου το πρώτο στοιχείο είναι η συντεταγμένη X και το δεύτερο στοιχείο η συντεταγμένη Y. Χρησιμοποιώντας λοιπόν αυτές τις παραμέτρους μπορεί να ξεκινήσει η αρχικοποίηση των μεταβλητών.

Η πρώτη μεταβλητή της κλάσης Car ονομάζεται _sprite και ορίζεται με τη φόρτωση της εικόνας του οχήματος από την σταθερά CAR_SPRITE_PATH που περιέχει την σχετική διαδρομή προς τον φάκελο που έχει δηλωθεί. Αυτή η εικόνα μετατρέπεται σε μια μορφή με την οποία είναι πιο εύκολο να επεξεργαστεί η Pygame χρησιμοποιώντας τη μέθοδο convert_alpha(). Στη συνέχεια, η συνάρτηση scale() χρησιμοποιείται για να αλλάξει το μέγεθος της εικόνας στις διαστάσεις που καθορίζονται από τις σταθερές CAR_SIZE_X και CAR_SIZE_Y.

Η μεταβλητή sprite ορίζεται στη συνέχεια με την μεταβλητή _sprite. Αυτό θα φανεί πολύ χρήσιμο στη συνέχεια όταν θα χρειαστεί να περιστραφεί η εικόνα του οχήματος, καθώς θα εξακολουθεί να υπάρχει μια αναφορά στο αρχικό _sprite [2] [3].

Αμέσως μετά η μεταβλητή position ορίζεται με ένα αντίγραφο της λίστας start_position. Η μέθοδος copy() χρησιμοποιείται για να διασφαλιστεί ότι οι αλλαγές στη λίστα position δεν επηρεάζουν τη λίστα start_position.

Η μεταβλητή facing_angle έχει οριστεί με την τιμή μηδέν (0), καθώς αυτό σημαίνει ότι η κατεύθυνση που βλέπει αρχικά το όχημα, είναι η δεξιά.

Η μεταβλητή speed αρχικοποιείται με την τιμή DEFAULT_SPEED, έτσι το όχημα έχει την αρχική του ταχύτητα.

Στη συνέχεια η μεταβλητή car_center είναι μια λίστα που αντιπροσωπεύει το κέντρο του αυτοκινήτου ως προς τον άξονα X και Y. Υπολογίζεται προσθέτοντας το μισό μέγεθος του αυτοκινήτου με την αρχική θέση που έχει πάρει στη μεταβλητή position.

Η επόμενη μεταβλητή ονομάζεται radars και αρχικοποιείται με μια κενή λίστα. Αυτό θα χρησιμοποιηθεί στη συνέχεια για την αποθήκευση των ραντάρ που χρησιμοποιεί το όχημα για να αντιληφθεί το περιβάλλον του.

Η μεταβλητή alive είναι τύπο boolean και έχει οριστεί με την τιμή True. Αυτό θα χρησιμοποιηθεί για να παρακολουθείτε εάν το αυτοκίνητο είναι ακόμα "ζωντανό" ή όχι, ανάλογα με το αν έχει χτυπήσει στα όρια της πίστας.

Στη συνέχεια η μεταβλητή distance_driven έχει οριστεί με την τιμή μηδέν (0). Αυτό θα χρησιμοποιηθεί με σκοπό την παρακολούθηση της απόστασης που έχει διανύσει το όχημα μέχρι τι στιγμή που θα σταματήσει η προσομοίωση ή όταν αυτό “πεθάνει”.

Τέλος η μεταβλητή speed_penalty ορίζεται επίσης με την τιμή μηδέν (0). Αυτό θα χρησιμοποιηθεί ως ποινή για ορισμένες ενέργειες του οχήματος κατά την επιβράβευση του.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
25      def __init__(self, start_position: list):
26          self._sprite = pygame.transform.scale(
27              pygame.image.load(CAR_SPRITE_PATH).convert_alpha(),
28              (self.CAR_SIZE_X, self.CAR_SIZE_Y)
29          )
30          self.sprite = self._sprite
31          self.position = start_position.copy()
32          self.facing_angle = 0
33          self.speed = self.DEFAULT_SPEED
34          self.car_center = [
35              self.position[0] + self.CAR_SIZE_X / 2,
36              self.position[1] + self.CAR_SIZE_Y / 2
37          ]
38          self.radars = []
39          self.alive = True
40          self.distance_driven = 0
41          self.speed_penalty = 0
```

Figure 11 Μέθοδος init

2.2.3 Δήλωση μεθόδου draw_radars()

Η δεύτερη μέθοδος που θα αναπτυχθεί είναι η draw_radars(self, environment: pygame.Surface). Η συγκεκριμένη μέθοδος χρησιμοποιείται για τη σχεδίαση ραντάρ στο δεδομένο περιβάλλον. Το περιβάλλον αναμένεται να είναι ένα αντικείμενο pygame.Surface, στο οποίο θα γίνει αναφορά σε επόμενο κεφάλαιο, το οποίο είναι ουσιαστικά ένας κενός καμβάς όπου μπορούν να σχεδιαστούν σχήματα, γραμμές, καθώς και να εμφανιστούν εικόνες.

Όσο για την λειτουργικότητα αυτή της μεθόδου, η μέθοδος draw_radars τρέχει σε έναν βρόχο επανάληψης στη λίστα self.radars της κλάσης Car, όπου έχει αρχικοποιηθεί

ως κενή. Κάθε ραντάρ σε αυτή τη λίστα αναμένεται να είναι ένα σύνολο αριθμών, όπου το πρώτο στοιχείο είναι η θέση του ραντάρ, δηλαδή προς την διεύθυνση που θα κοιτάζει. Αυτή η θέση χρησιμοποιείται για να χαραχθεί μια γραμμή και ένας κύκλος που θα αντιπροσωπεύει το ραντάρ.

Για να μπορέσει να συμβεί αυτό, θα χρησιμοποιηθούν δύο μέθοδοι απο την βιβλιοθήκη Pygame. Η μέθοδος pygame.draw.line χρησιμοποιείται για να εμφανίσει μια γραμμή στο περιβάλλον. Η γραμμή έχει ώς αφετηρία το κέντρο του αυτοκινήτου, self.car_center, τη θέση ή διεύθυνση την οποία το ραντάρ θα κοιτάζει, radar_position, το χρώμα της γραμμής με το οποίο θα αντιπροσωπεύει το μήκος του ραντάρ το οποίο είναι πράσινο, όπως υποδεικνύεται από το Color.GREEN και το πάχος της γραμμής που ορίζεται από το self.RADAR_LINE [4].

Η μέθοδος pygame.draw.circle χρησιμοποιείται για να εμφανίσει έναν κύκλο στο περιβάλλον στη θέση radar. Το χρώμα του κύκλου είναι κόκκινο, όπως υποδεικνύεται από το Color.RED, και η ακτίνα του κύκλου ορίζεται από το self.RADAR_CIRCLE [5].

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
48      def draw_radars(self, environment: pygame.Surface) -> None:
49          for radar in self.radars:
50              radar_position = radar[0]
51              pygame.draw.line(environment, Color.GREEN,
52                               self.car_center, radar_position, self.RADAR_LINE)
53              pygame.draw.circle(
54                  environment, Color.RED, radar_position, self.RADAR_CIRCLE)
```

Figure 12 Μέθοδος draw_radar

2.2.4 Δήλωση μεθόδου draw()

Η τρίτη μέθοδος που θα αναπτυχθεί είναι η draw(). Αυτή η μέθοδος παίρνει δύο ορίσματα ως παραμέτρους, τις self και environment. Η παράμετρος self υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Car και καλείται για να χρησιμοποιείται για τη σχεδίαση του αυτοκινήτου στο δεδομένο περιβάλλον εάν το αυτοκίνητο είναι ζωντανό. Το περιβάλλον αναμένεται να είναι ένα αντικείμενο pygame.Surface, στο οποίο θα γίνει αναφορά σε επόμενο κεφάλαιο, το οποίο είναι ουσιαστικά ένας κενός καμβάς όπου μπορούν να σχεδιαστούν σχήματα, γραμμές, καθώς και να εμφανιστούν εικόνες.

Αρχικά αυτή η μέθοδος πρέπει να ελέγχει εάν το αυτοκίνητο είναι ζωντανό, έτσι ώστε να μπορέσει να σχεδιαστεί στην οθόνη. Ο έλεγχος if self.alive: ελέγχει αν το αυτοκίνητο είναι ζωντανό. Το χαρακτηριστικό alive είναι τύπου boolean που αντιπροσωπεύει εάν το αυτοκίνητο είναι ζωντανό ή όχι. Εάν το αυτοκίνητο δεν είναι ζωντανό, η λειτουργικότητα της μεθόδου δεν εκτελείται.

Εάν το αυτοκίνητο είναι ζωντανό, η μέθοδος προχωρά στο να σχεδιάσει το αυτοκίνητο στο περιβάλλον. Η εντολή environment.blit(self.sprite, self.position) είναι υπεύθυνη για αυτό. Η μέθοδος blit είναι μια μέθοδος της βιβλιοθήκης pygame που σχεδιάζει μια εικόνα (σε αυτήν την περίπτωση, self.sprite) στο αντικείμενο Surface στην καθορισμένη θέση (σε αυτήν την περίπτωση, self.position) [6].

Μετά την ολοκλήρωση της σχεδίασης του αυτοκινήτου, η μέθοδος καλεί την μέθοδο draw_radars. Αυτή η μέθοδος είναι υπεύθυνη για τη σχεδίαση των ραντάρ του αυτοκινήτου στο περιβάλλον, όπως έχει γίνει αναφορά στο προηγούμενο υποκεφάλαιο.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
43      def draw(self, environment: pygame.Surface) -> None:
44          if self.alive:
45              environment.blit(self.sprite, self.position)
46              self.draw_radars(environment)
```

Figure 13 Μέθοδος draw

2.2.5 Δήλωση Μεθόδου out_of_bounds()

Η τέταρτη μέθοδος που θα αναπτυχθεί είναι η out_of_bounds. Αυτή η μέθοδος παίρνει πέντε ορίσματα ώς παραμέτρους, τα self, point_x, point_y, width και height. Αυτή η μέθοδος έχει σχεδιαστεί για να ελέγχει εάν ένα δεδομένο σημείο, που καθορίζεται από τις συντεταγμένες x και y του (point_x και point_y), είναι εκτός των ορίων μιας συγκεκριμένης περιοχής. Η περιοχή ορίζεται από το πλάτος και το ύψος της (width και height).

Για να μπορέσει να συμβεί αυτό, η μέθοδος χρησιμοποιεί το λογικό OR (ή) για να συνδυάσει τέσσερις συνθήκες. Εάν κάποια από αυτές τις συνθήκες είναι αληθείς (True), η μέθοδος θα επιστρέψει την τιμή True, υποδεικνύοντας ότι το σημείο είναι εκτός ορίων, άρα και το όχημα επίσης. Εάν καμία από αυτές τις συνθήκες δεν είναι True, η μέθοδος θα επιστρέψει False, υποδεικνύοντας ότι το σημείο βρίσκεται εντός των ορίων της περιοχής.

Πιο συγκεκριμένα, οι τέσσερις συνθήκες είναι οι παρακάτω :

- point_x < 0: Ελέγχει εάν η συντεταγμένη x του σημείου είναι μικρότερη από 0. Εάν είναι, το σημείο είναι εκτός ορίων στην αριστερή πλευρά.
- point_x >=width: Ελέγχει εάν η συντεταγμένη x του σημείου είναι μεγαλύτερη ή ίση με το πλάτος της περιοχής. Αν είναι, το σημείο είναι εκτός ορίων στη δεξιά πλευρά.

- point_y < 0: Ελέγχει εάν η συντεταγμένη y του σημείου είναι μικρότερη από 0. Εάν είναι, το σημείο είναι εκτός ορίων στην επάνω πλευρά.
- point_y >= ύψος: Ελέγχει εάν η συντεταγμένη y του σημείου είναι μεγαλύτερη ή ίση με το ύψος της περιοχής. Εάν είναι, το σημείο είναι εκτός ορίων στην κάτω πλευρά.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
56      def out_of_bounds(self, point_x: int, point_y: int, width: int, height: int) -> bool:
57          return point_x < 0 or point_x >= width or point_y < 0 or point_y >= height
```

Figure 14 Μέθοδος out_of_bounds

2.2.6 Δήλωση μεθόδου distance_to_border()

Η πέμπτη μέθοδος που θα αναπτυχθεί είναι η distance_to_border(). Αυτή η μέθοδος παίρνει τρία ορίσματα ως παραμέτρους, τα sefl, point_x και point_y. Η παράμετρος self υποδηλώνει ότι αυτή η συνάρτηση είναι μέρος της κλάσης Car. Οι δύο τελευταίοι παράμετροι, αντιπροσωπεύουν τις συντεταγμένες x και y ενός σημείου σε ένα δισδιάστατο χώρο. Αυτή η μέθοδος έχει σχεδιαστεί για τον υπολογισμό της απόστασης του αυτοκινήτου από τα όρια της πίστας.

Πιο συγκεκριμένα αυτή η μέθοδος χρησιμοποιεί τη συνάρτηση math.hypot για να υπολογίσει την Ευκλείδεια απόσταση. Η συνάρτηση math.hypot παίρνει μια ακολουθία αριθμών (σε αυτή την περίπτωση, τις διαφορές μεταξύ των συντεταγμένων x και y του κέντρου του αυτοκινήτου και του δεδομένου σημείου στα όρια της πίστας) και επιστρέφει την τετραγωνική ρίζα του αθροίσματος των τετραγώνων αυτών των αριθμών. Αυτό ισοδυναμεί με τον υπολογισμό του μήκους της υποτείνουσας σε ένα ορθογώνιο τρίγωνο.

Στη συνέχεια, το αποτέλεσμα της συνάρτησης math.hypot μετατρέπεται σε ακέραιο χρησιμοποιώντας τη συνάρτηση int. Αυτό συμβαίνει επειδή η συνάρτηση math.hypot επιστρέφει ένα float αριθμό, αλλά η μέθοδος καθορίζεται να επιστρέφει έναν ακέραιο αριθμό (όπως υποδεικνύεται από το -> int στον ορισμό της μεθόδου).

Η μέθοδος append στην Python προσθέτει ένα στοιχείο στο τέλος μιας λίστας. Σε αυτήν την περίπτωση, προσθέτουμε μια πλειάδα, η οποία είναι μια αμετάβλητη ακολουθία στοιχείων. Η πλειάδα περιέχει τις συντεταγμένες του σημείου και την απόσταση αυτού του σημείου από το κέντρο του αυτοκινήτου.

Τέλος, αυτή η μέθοδος δεν επιστρέφει κάποια τιμή. Σκοπός της είναι να υπολογίζει μια απόσταση και να ενημερώσει τη λίστα self.radars με αυτές τις πληροφορίες.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
59      def distance_to_border(self, point_x: int, point_y: int) -> int:
60          distance = int(math.hypot(
61              point_x - self.car_center[0], point_y - self.car_center[1]))
62          self.radars.append([(point_x, point_y), distance])
```

Figure 15 Μέθοδος distance_to_border

2.2.7 Δήλωση μεθόδου check_collision()

Η έκτη μέθοδος που θα αναπτυχθεί είναι η check_collision(). Αυτή η μέθοδος παίρνει δύο ορίσματα ως παραμέτρους, τις self και environment. Η παράμετρος self υποδηλώνει ότι αυτή η συνάρτηση είναι μέρος της κλάσης Car, καθώς το όρισμα environment υποδηλώνει ότι αυτή η μέθοδος δέχεται τον χάρτη της πίστας που θα κινείται το όχημα, έτσι ώστε να κάνει τους απαραίτητους ελέγχους. Αυτή η μέθοδος έχει σχεδιαστεί για να ελέγχει εάν το αυτοκίνητο έχει συγκρουστεί με τα όρια της πίστας στο περιβάλλον του.

Αρχικά αυτή η μέθοδος ανακτά το μέγεθος του περιβάλλοντος χρησιμοποιώντας τη μέθοδο get_size, η οποία επιστρέφει μια πλειάδα δύο ακεραίων που αντιπροσωπεύουν το πλάτος και το ύψος του περιβάλλοντος.

Στη συνέχεια, η μέθοδος επαναλαμβάνεται με έναν βρόχο επανάληψης σε κάθε από τις γωνίες self.corners, όπου είναι μια λίστα σημείων που αντιπροσωπεύουν τις γωνίες του αυτοκινήτου. Για κάθε γωνία, ελέγχει δύο συνθήκες:

- Αν η γωνία είναι εκτός των ορίων του περιβάλλοντος. Αυτό γίνεται χρησιμοποιώντας τη μέθοδο out_of_bounds, η οποία αναπτύχθηκε σε προηγούμενο υποκεφάλαιο.
- Εάν το χρώμα του pixel στις συντεταγμένες της γωνίας στο περιβάλλον είναι λευκό. Αυτό γίνεται χρησιμοποιώντας τη μέθοδο get_at του αντικειμένου enviroment, η οποία επιστρέφει το χρώμα του pixel στις δεδομένες συντεταγμένες και συγκρίνοντάς το με το Color.WHITE [7].

Τέλος, εάν οποιαδήποτε από αυτές τις συνθήκες ισχύει, η μέθοδος ορίζει το self.alive σε False που υποδηλώνει ότι το αυτοκίνητο δεν είναι πλέον "ζωντανό" λόγω της σύγκρουσης και επιστρέφει True, υποδεικνύοντας ότι έχει συμβεί σύγκρουση. Εάν καμία συνθήκη δεν ισχύει για οποιαδήποτε γωνία, η μέθοδος επιστρέφει False, υποδεικνύοντας ότι δεν έχει συμβεί σύγκρουση.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
64      def check_collision(self, environment: pygame.Surface) -> bool:
65          environment_width, environment_height = environment.get_size()
66          for point in self.corners:
67              point_x, point_y = point
68              if self.out_of_bounds(point_x, point_y, environment_width, environment_height) or \
69                 environment.get_at((int(point_x), int(point_y))) == Color.WHITE:
70                  self.alive = False
71                  return True
72          return False
```

Figure 16 Μέθοδος check_collision

2.2.8 Δήλωση μεθόδου check_radar()

Η έβδομη μέθοδος που θα αναπτυχθεί ονομάζεται check_radar(). Αυτή η μέθοδος παίρνει τρία ορίσματα ως παραμέτρους, τις self, degree και environment. Η παράμετρος self υποδηλώνει ότι αυτή η μεθοδος είναι μέρος της κλάσης Car. Το όρισμα degree υποδικνίει απο ποια γωνία σε μοίρες θα γίνει έλεγχος του ραντάρ. Το όρισμα environment υποδηλώνει ότι αυτή η μέθοδος δέχεται τον χάρτη της πίστας που θα κινείται το όχημα, έτσι ώστε να κάνει τους απαραίτητους ελέγχους. Αυτή η μέθοδος έχεις σχεδιαστεί για την προσομοίωση ενός αισθητήρα ραντάρ που ανιχνεύει την απόσταση από το πλησιέστερο όριο της πίστας σε μια δεδομένη κατεύθυνση.

Αρχικά η μέθοδος μετατρέπει πρώτα την μεταβλητή degree σε ακτίνια επειδή οι συναρτήσεις math.cos και math.sin στην Python λειτουργούν με ακτίνια και όχι με μοίρες. Η σταθερά FULL_CIRCLE_DEGREES αφαιρείται από το άθροισμα της γωνίας πρόσοψης (self.facing_angle) του αυτοκινήτου και τη δεδομένη μοίρα.

Στη συνέχεια, η μέθοδος εισέρχεται σε έναν βρόχο επανάληψης που υπολογίζει τις συντεταγμένες x και y ενός σημείου που απέχει κάποιες μονάδες μήκους από το κέντρο του αυτοκινήτου προς την κατεύθυνση του ραντάρ. Η μεταβλητή μήκους (length) ξεκινά από το 1 και αυξάνεται σε κάθε επανάληψη του βρόχου.

Ο βρόχος συνεχίζεται μέχρι να ικανοποιηθεί μία από τις τρεις προϋποθέσεις:

- Το υπολογιζόμενο σημείο είναι εκτός των ορίων της πίστας. Αυτό ελέγχεται από τη συνάρτηση out_of_bounds, η οποία αναπτύχθηκε σε προηγούμενο υποκεφάλαιο.
- Το χρώμα του pixel στο υπολογιζόμενο σημείο του περιβάλλοντος είναι λευκό. Αυτό ελέγχεται με τη μέθοδο get_at του αντικειμένου pygame.Surface, το οποίο επιστρέφει το χρώμα του pixel στις δεδομένες συντεταγμένες.
- Η μεταβλητή length υπερβαίνει την τιμή της σταθεράς RADAR_DRAW_DISTANCE. Αυτή η σταθερά αντιπροσωπεύει τη μέγιστη απόσταση που μπορεί να ανιχνεύσει το ραντάρ.

Σε περίπτωση που ικανοποιηθεί μία από τις παραπάνω προϋποθέσεις, ο βρόχος επανάληψης θα σπάσει.

Μόλις τελειώσει ο βρόχος, η συνάρτηση καλεί τη μέθοδο distance_to_border με το τελευταίο υπολογισμένο σημείο. Έτσι μπορεί να υπολογιστεί η καινούρια απόσταση από τα όρια της πίστας.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
74      def check_radar(self, degree: int, environment: pygame.Surface) -> None:
75          radians = math.radians(self.FULL_CIRCLE_DEGREES -
76                                 (self.facing_angle + degree))
77          length = 1
78          environment_width, environment_height = environment.get_size()
79
80          while True:
81              length += 1
82              point_x = int(self.car_center[0] + math.cos(radians) * length)
83              point_y = int(self.car_center[1] + math.sin(radians) * length)
84
85              if self.out_of_bounds(point_x, point_y, environment_width, environment_height):
86                  break
87              if environment.get_at((point_x, point_y)) == Color.WHITE:
88                  break
89              if length > Car.RADAR_DRAW_DISTANCE:
90                  break
91
92          self.distance_to_border(point_x, point_y)
```

Figure 17 Μέθοδος check_radar

2.2.9 Δήλωση μεθόδου calculate_corner()

Η όγδοη μέθοδος που θα αναλυθεί ονομάζεται calculate_corner(). Αυτή η μέθοδος παίρνει τέσσερα ορίσματα ως παραμέτρους, τις self, angle, car_x, car_y. Η παράμετρος self υποδηλώνει ότι αυτή η συνάρτηση είναι μέρος της κλάσης Car. Angle είναι η γωνία του αυτοκινήτου που θα υπολογισθεί. Οι παράμετροι car_x και car_y αναπαριστούν τις πλευρές του αυτοκινήτου στον άξονα x και y. Αυτή η μέθοδος έχει σχεδιαστεί για τον υπολογισμό της θέσης της κάθε γωνίας ενός αυτοκινήτου.

Αρχικά η μέθοδος ξεκινά με τον υπολογισμό της γωνίας corner σε ακτίνια. Η σταθερά FULL_CIRCLE_DEGREES αφαιρείται από το άθροισμα της γωνίας πρόσοψης (self.facing_angle) του αυτοκινήτου και τη δεδομένη μοίρα και στη συνέχεια μετατρέπει αυτό το αποτέλεσμα σε ακτίνια χρησιμοποιώντας τη συνάρτηση math.radians. Αυτό συμβαίνει επειδή οι τριγωνομετρικές συναρτήσεις στη μαθηματική ενότητα της Python αναμένουν την εισαγωγή των παραμέτρων τους σε ακτίνια και όχι σε μοίρες.

Στη συνέχεια, υπολογίζει τις συντεταγμένες x και y του γωνιακού σημείου. Αυτό συμβαίνει προσθέτοντας τις συνιστώσες x και y των αποστάσεων car_x και car_y (υπολογίζονται χρησιμοποιώντας math.cos(corner) και math.sin(corner) αντίστοιχα) στις συντεταγμένες x και y του κέντρου του αυτοκινήτου (self.car_center[0] και self.car_center[1] αντίστοιχα).

Τέλος η μέθοδος επιστρέφει αυτές τις συντεταγμένες x και y ως λίστα ακεραίων έτσι ώστε να χρησιμοποιηθούν σε μελλοντικούς υπολογισμούς.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
94      def calculate_corner(self, angle, car_x, car_y) -> list[int]:
95          corner = math.radians(self.FULL_CIRCLE_DEGREES -
96                                (self.facing_angle + angle))
97          return [
98              self.car_center[0] + math.cos(corner) * car_x,
99              self.car_center[1] + math.sin(corner) * car_y
100         ]
```

Figure 18 Μέθοδος calculate_corner

2.2.10 Δήλωση μεθόδου refresh_corners_positions()

Η ένατη μέθοδος που θα αναπτυχθεί ονομάζεται refresh_corners_position(). Αυτή η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Car. Ο σκοπός αυτής της μεθόδου είναι να ανανεώνει ή να ενημερώνει τις θέσεις των γωνιών του αυτοκινήτου. Αυτό συμβαίνει υπολογίζοντας τις θέσεις των γωνιών σε συγκεκριμένες γωνίες και αποθηκεύοντάς τες στο χαρακτηριστικό self.corners του αυτοκινήτου.

Η μέθοδος ξεκινά ορίζοντας δύο μεταβλητές, τις car_x και car_y, οι οποίες ορίζονται στο μισό του μεγέθους του αυτοκινήτου στις διαστάσεις x και y αντίστοιχα.

Στη συνέχεια, ορίζει μια λίστα γωνιών, [30, 150, 210, 330]. Αυτές οι γωνίες αντιπροσωπεύουν τις θέσεις των γωνιών του αυτοκινήτου σε μοίρες, καθώς το αυτοκίνητο είναι ορθογώνιο αλλά και τετράγωνο ως γεωμετρικό σχήμα.

Τέλος, χρησιμοποιεί list comprehension για να υπολογίσει τις θέσεις των γωνιών και τις αποθηκεύει στο χαρακτηριστικό self.corners. Η μέθοδος calculate_corner() καλείται για κάθε γωνία στη λίστα των γωνιών, για τον υπολογισμό κάθε γωνίας, όπως προαναφέρθηκε στο προηγούμενο υποκεφάλαιο. Έτσι αυτή η μέθοδος θα χρησιμοποιείται κάθε φορά που η κατάσταση του αυτοκινήτου θα αλλάζει.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
102     def refresh_corners_positions(self) -> None:
103         car_x = Car.CAR_SIZE_X / 2
104         car_y = Car.CAR_SIZE_Y / 2
105         angles = [30, 150, 210, 330]
106 
107         self.corners = [self.calculate_corner(
108             angle, car_x, car_y) for angle in angles]
109 
```

Figure 19 Μέθοδος refresh_corners_positions

2.2.11 Δήλωση μεθόδου update_car_center()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται update_car_center(). Αυτή η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Car. Αυτή η μέθοδος χρησιμοποιείται για την ενημέρωση του κέντρου του αυτοκινήτου, καθώς και για την περιστροφή της εικόνας του όσο αυτό κινείται στην πίστα.

Η μέθοδος ξεκινά παίρνοντας το ορθογώνιο που περικλείει την εικόνα του αυτοκινήτου (self._sprite) χρησιμοποιώντας τη μέθοδο get_rect() [8]. Κατά αυτό τον τρόπο, ο αλγόριθμος θα έχει πάντα το σωστό ορθογώνιο του αυτοκινήτου όσο αυτό αλλάζει θέση στην πίστα.

Στη συνέχεια, η εικόνα του αυτοκινήτου περιστρέφεται με την γωνία που καθορίζεται από το self.facing_angle χρησιμοποιώντας τη συνάρτηση pygame.transform.rotate(). Αυτή η λειτουργία δημιουργεί μια νέα επιφάνεια που συγκρατεί την περιστραμμένη εικόνα [9].

Για να διασφαλιστεί ότι η εικόνα θα περιστρέφεται γύρω από το κέντρο του αυτοκινήτου και όχι στην επάνω αριστερή γωνία του, που είναι το προεπιλεγμένο σημείο στην βιβλιοθήκη του Pygame, θα πρέπει να τεθεί πιο είναι το κέντρο της εικόνας. Έτσι, για να συμβεί αυτό, αποθηκεύεται στο κέντρο του αρχικού ορθογωνίου, το κέντρο του περιστραμένου ορθογωνίου.

Στη συνέχεια, χρησιμοποιείται η μέθοδος subsurface() για τη δημιουργία μιας νέας επιφάνειας που αναφέρεται στην περιοχή του περιστραμένου ορθογωνίου που ταιριάζει στο αρχικό ορθογώνιο. Αυτή η νέα επιφάνεια έχει εκχωρηθεί στο self.sprite.

Τέλος, το κέντρο του αυτοκινήτου ενημερώνεται, υπολογίζοντας τις συντεταγμένες x και y του αντιστοίχως, στο χαρακτηριστικό self.car_center.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
110     def update_car_center(self) -> None:
111         sprite_as_rect = self._sprite.get_rect()
112         rotated_sprite = pygame.transform.rotate(
113             self._sprite, self.facing_angle)
114         sprite_as_rect.center = rotated_sprite.get_rect().center
115         self.sprite = rotated_sprite.subsurface(sprite_as_rect)
116         self.car_center = [
117             int(self.position[0]) + Car.CAR_SIZE_X / 2,
118             int(self.position[1]) + Car.CAR_SIZE_Y / 2
119         ]
```

Figure 20 Μέθοδος updata_car_center

2.2.12 Δήλωση μεθόδου update_car_state()

Στη συνέχεια θα αναλυθεί η μέθοδος update_car_state(). Αυτή η μέθοδος παίρνει δύο ορίσματα ως παραμέτρους, τα self και environment. Η παράμετρος self υποδηλώνει ότι αυτή η συνάρτηση είναι μέρος της κλάσης Car. Το όρισμα environment υποδηλώνει ότι αυτή η μέθοδος δέχεται τον χάρτη της πίστας που θα κινείται το αυτοκίνητο, έτσι ώστε να κάνει τους απαραίτητους ελέγχους. Αυτή η μέθοδος έχει σχεδιαστεί για να ενημερώνει την κατάσταση του αυτοκινήτου, όσο αυτό κινείται μέσα στην πίστα.

Αρχικά η μέθοδος καλεί πρώτα την μέθοδος update_car_center() που αναλύθηκε στο προηγούμενο υποκεφάλαιο, έτσι ώστε το κέντρο του αυτοκινήτου να είναι ενημερωμένο.

Στη συνέχεια, η μέθοδος υπολογίζει τη νέα θέση του αυτοκινήτου με βάση την τρέχουσα ταχύτητα του (self.speed) και τη γωνία πρόσοψής του (self.facing_angle). Η γωνία πρόσοψής μετατρέπεται από μοίρες σε ακτίνια και στη συνέχεια το συνημίτονο και το ημίτονο αυτής της γωνίας χρησιμοποιούνται για τον υπολογισμό των συντεταγμένων x και y, αντίστοιχα. Μετά από αυτούς τους υπολογισμούς, ενημερώνεται η συνολική απόσταση που το αυτοκίνητο έχει διανύσει.

Σε επόμενο βήμα, καλείται η μέθοδος refresh_corners_positions(), η οποία υπολογίζει τις θέσεις των γωνιών του αυτοκινήτου και αμέσως μετά η μέθοδος check_collision() για να διαπιστωθεί εάν το αυτοκίνητο έχει συγκρουστεί.

Τέλος, τα δεδομένα που βρίσκονται στα ραντάρ (self.radars) του αυτοκινήτου καθαρίζονται και η μέθοδος check_radar() καλείται για ένα εύρος γωνιών. Αυτό σημαίνει ότι, για κάθε μία γωνία, θα ελεγχθεί το αντίστοιχο ραντάρ και θα αποθηκεύσει τα νέα δεδομένα που θα χρειαστεί για τους επόμενους ελέγχου.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
121     def update_car_state(self, environment: pygame.Surface) -> None:
122         self.update_car_center()
123         radians = math.radians(self.FULL_CIRCLE_DEGREES - self.facing_angle)
124         self.position[0] += math.cos(radians) * self.speed
125         self.position[1] += math.sin(radians) * self.speed
126         self.distance_driven += self.speed
127         self.refresh_corners_positions()
128         self.check_collision(environment)
129         self.radars.clear()
130         for radar_angle in range(self.RADAR_START_ANGLE, self.RADAR_END_ANGLE + 1, self.RADAR_STEP_ANGLE):
131             self.check_radar(radar_angle, environment)
```

Figure 21 Μέθοδος update_car_state

2.2.13 Δήλωση μεθόδου get_data()

Η επόμενη μέθοδος που θα αναλυθεί ονομάζεται get_data(). Αυτή η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Car. Αυτή η μέθοδος έχει σχεδιαστεί, έτσι ώστε να παίρνει τις αποστάσεις των ραντάρ ως δεδομένα για τους απαραίτητους υπολογισμούς.

Αρχικά η μέθοδος ξεκινά ορίζοντας μια list comprehension που επαναλαμβάνεται μέσα στο χαρακτηριστικό self.radars. Για κάθε ραντάρ στο self.radars, παίρνει το δεύτερο στοιχείο (δείκτης 1) και το μετατρέπει σε ακέραιο. Το αποτέλεσμα είναι μια λίστα ακεραίων, η οποία αποθηκεύεται στη μεταβλητή distances.

Στη συνέχεια, η μέθοδος ελέγχει εάν το μήκος των αποστάσεων είναι μικρότερο από 5. Εάν είναι, προσθέτει μηδενικά στις αποστάσεις έως ότου το μήκος της είναι 5. Αυτό γίνεται χρησιμοποιώντας τον τελεστή +=, ο οποίος επεκτείνει τις αποστάσεις της λίστας με μια άλλη λίστα. Η δεύτερη λίστα δημιουργείται πολλαπλασιάζοντας μια λίστα που περιέχει ένα μόνο μηδέν με τη διαφορά μεταξύ του 5 και του τρέχοντος μήκους των αποστάσεων. Αυτό έχει ως αποτέλεσμα μια λίστα με μηδενικά του απαιτούμενου μήκους.

Τέλος, η μέθοδος επιστρέφει τη λίστα αποστάσεων. Εάν το χαρακτηριστικό self.radar έχει λιγότερα από 5 στοιχεία, ορισμένα από τα στοιχεία στη λίστα που επιστρέφεται θα έχουν την τιμή μηδέν.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
133     def get_data(self) -> list[int]:
134         distances = [int(radar[1]) for radar in self.radars]
135         distances += [0] * (5 - len(distances))
136         return distances
```

Figure 22 Μέθοδος get_data

2.2.14 Δήλωση μεθόδου get_reward()

Στη συνέχεια θα αναπτυχθεί η μέθοδος get_reward(). Αυτή η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Car. Η συγκεκριμένη μέθοδος έχει σχεδιαστεί έτσι ώστε το αυτοκίνητο να επιβραβεύεται όσο αυτό κινείται στην πίστα. Αυτή η λειτουργία έχει ιδιαίτερη σημασία σε αυτή την εργασία καθώς η επιβράβευσή μιας τεχνητής νοημοσύνης είναι πολύ σημαντική. Επίσης σε αυτό το σημείο θα ήταν καλό να αναφερθεί ότι, μπορούν να βρεθούν διάφοροι τρόποι επιβράβευσης. Στη συγκεκριμένη εργασία ο υπολογισμός της παρακάτω επιβράβευσης αποφασίστηκε λόγο αποδοτικότητας.

Αρχικά η μέθοδος ξεκινά με τον υπολογισμό της ανταμοιβής αποστάσεως (distance_reward) που είναι η απόσταση που διανύει το αυτοκίνητο διαιρούμενο με το 10000. Αυτό σημαίνει ότι για κάθε 10000 μονάδες απόστασης που αυτό διανύει, το αυτοκίνητο λαμβάνει ανταμοιβή 1 μονάδας.

Στη συνέχεια, υπολογίζει την ανταμοιβή ταχύτητας (speed_reward) που είναι η τετραγωνική ρίζα της ταχύτητας του αυτοκινήτου. Αυτό σημαίνει ότι η ανταμοιβή αυξάνεται μη γραμμικά με την ταχύτητα, αλλά με φθίνουσα ταχύτητα. Για παράδειγμα, η αύξηση της ταχύτητας από 1 σε 4 τετραπλασιάζει την ανταμοιβή, αλλά η αύξηση της ταχύτητας από 4 σε 9 αυξάνει μόνο την ανταμοιβή κατά 1,5.

Στη συνέχεια, υπολογίζει μια ποινή (speed_penalty) η οποία είναι η ποινή ταχύτητας του αυτοκινήτου διαιρούμενη με το 100. Αυτό αντιπροσωπεύει μια ποινή για υπέρβαση ενός συγκεκριμένου ορίου ταχύτητας ή για ακανόνιστη συμπεριφορά οδήγησης που έχει ως αποτέλεσμα γρήγορες αλλαγές στην ταχύτητα.

Τέλος, η μέθοδος υπολογίζει την τελική ανταμοιβή (finale_reward) προσθέτοντας την ανταμοιβή αποστάσεως (distance_reward) και την ανταμοιβή ταχύτητας (speed_reward), καθώς στη συνέχεια αφαιρείται η ποινή ταχύτητας (speed_penalty). Αυτή η τελική ανταμοιβή αντιπροσωπεύει μια ισορροπία μεταξύ της ενθάρρυνσης του αυτοκινήτου να οδηγεί μεγάλες αποστάσεις και σε υψηλές ταχύτητες, αλλά και την τιμωρία του για ανεπιθύμητη οδηγική συμπεριφορά.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
138     def get_reward(self) -> float:
139         distance_reward = self.distance_driven / 10000
140         speed_reward = self.speed ** 0.5
141         penalty = self.speed_penalty / 100
142         final_reward = distance_reward + speed_reward - penalty
143         return final_reward
```

Figure 23 Μέθοδος get_reward

2.2.15 Δήλωση μεθόδου accelerate()

Η συγκεκριμένη μέθοδος έχει σχεδιαστεί για να αυξάνει την ταχύτητα του αυτοκινήτου. Η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Car.

Όπως αναφέρθηκε, η λειτουργία αυτής της μεθόδου είναι να αυξάνει την ταχύτητα του αυτοκινήτου. Αυτό σημαίνει πως όταν αυτό θα παίρνει την απόφαση να επιταχύνει, θα πρέπει να γίνονται και οι απαραίτητοι υπολογισμοί. Αυτό μπορεί να συμβεί απλά, αυξάνοντας την ήδη υπάρχουσα ταχύτητα κατά ένα ποσοστό. Έτσι μέσα στη μέθοδο, το self.speed += Car.SPEED_INCREMENT αυξάνει την τιμή του self.speed κατά μια σταθερή τιμή Car.SPEED_INCREMENT, οι οποία έχει δηλωθεί στην αρχή της κλάσης Car και έχει την τιμή ένα (1).

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
145     def accelerate(self) -> None:
146         self.speed += Car.SPEED_INCREMENT
```

Figure 24 Μέθοδος accelerate

2.2.16 Δήλωση μεθόδου brake()

Αμέσως μετά την μέθοδο επιτάχυνσης ακολουθεί η μέθοδος επιβράδυνσης, brake(). Η συγκεκριμένη μέθοδος έχει σχεδιαστεί για να μπορεί το αυτοκίνητο να επιβραδύνει όταν αυτό πληροί κάποια συνθήκη. Η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Car.

Για να μπορέσει να συμβεί αυτό, πρέπει πρώτα να δημιουργηθεί αυτή η συνθήκη. Έτσι η μέθοδος ελέγχει πρώτα εάν η τρέχουσα ταχύτητα του αυτοκινήτου (self.speed) είναι μεγαλύτερη από μια προκαθορισμένη ελάχιστη ταχύτητα (Car.MINIMUM_SPEED). Εάν είναι, η μέθοδος μειώνει την ταχύτητα κατά ένα σταθερό ποσοστό (Car.SPEED_INCREMENT). Αλλιώς, εάν η τρέχουσα ταχύτητα είναι μεγαλύτερη από την ελάχιστη ταχύτητα (Car.MINIMUM_SPEED), η μέθοδος ορίζει την ταχύτητα (self.speed) στην ελάχιστη ταχύτητα (Car.MINIMUM_SPEED). Αυξάνει επίσης το χαρακτηριστικό speed_penalty κατά 1, που όπως προαναφέρθηκε χρησιμοποιείται για την εκπόνηση ποινής στο αυτοκίνητο.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
148     def brake(self) -> None:
149         if self.speed > Car.MINIMUM_SPEED:
150             self.speed -= Car.SPEED_INCREMENT
151         else:
152             self.speed = Car.MINIMUM_SPEED
153             self.speed_penalty += 1
```

Figure 25 Μέθοδος brake

2.2.17 Δήλωση μεθόδου turn_left()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται turn_left().Η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Car. Αυτή η μέθοδος έχει σχεδιαστεί έτσι ώστε το αυτοκίνητο να μπορεί να στρίψει αριστερόστροφα.

Για να μπορέσει να συμβεί αυτό, πρέπει πρώτα να γίνει η εξής παρατήρηση. Το αυτοκίνητο μπορεί να στρίψει γύρω από το κέντρο του, καθώς αυτό σημαίνει ότι για να μπορέσει να συμβεί αυτό θα πρέπει να αυξηθεί η γωνία που αυτό κοιτάει σε μοίρες. Για να γίνει αυτό αντιληπτό θα δοθεί το παρακάτω παράδειγμα.

Όταν το αυτοκίνητο βρίσκεται στις μηδέν (0) μοίρες, αυτό κοιτάζει και έχει διεύθυνση πρός τα δεξιά της οθόνης. Προσθέτοντας τριάντα (30) μοίρες με θετικό πρόσημο, η διεύθυνση του θα αλλάξει προς τα αριστερά, καθώς θα περιστραφεί γύρω από το κέντρο του.

Έτσι για να υπολογισθεί αυτή αλλαγή στην διεύθυνσή του, θα πρέπει να προσθέσουμε στην ήδη υπάρχουσα διεύθυνση (self.facing_angle), το ποσοστό περιστροφής (Car.ANGLE_INCREMENT). Αυτό το ποσοστό έχει αρχικοποιηθεί στην αρχή της κλάσης Car και έχει την τιμή δέκα (10).

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
155     def turn_left(self) -> None:
156         self.facing_angle += Car.ANGLE_INCREMENT
```

Figure 26 Μέθοδος turn_left

2.2.18 Δήλωση μεθόδου turn_right()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται turn_right().Η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Car. Αυτή η μέθοδος έχει σχεδιαστεί έτσι ώστε το αυτοκίνητο να μπορεί να στρίψει δεξιόστροφα.

Αυτή η μέθοδος μοιάζει πολύ με την προηγούμενη, καθώς η συμπεριφορά της είναι αντιστρόφως ανάλογη με την μέθοδο turn_left().

Έτσι για να υπολογιστεί η αλλαγή της διεύθυνσης προς τα δεξιά, αρκεί να αφαιρέσουμε από την ήδη υπάρχουσα διεύθυνση (self.facing_angle), το ποσοστό περιστροφής (Car.ANGLE_INCREMENT). Αυτό το ποσοστό έχει αρχικοποιηθεί στην αρχή της κλάσης Car και έχει την τιμή δέκα (10).

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
158     def turn_right(self) -> None:
159         self.facing_angle -= Car.ANGLE_INCREMENT
```

Figure 27 Μέθοδος turn_right

2.3 Δήλωση κλάσης Actions

Καθώς έχει τελειώσει η ανάπτυξη της λογικής και των μεθόδων του αυτοκινήτου, θα ακολουθήσει η ανάπτυξη και η επεξήγηση των δράσεων ή κινήσεων που αυτό μπορεί να εκτελέσει. Αυτές οι κινήσεις πηγάζουν και δεν διαφέρουν από αυτές που μπορεί να εκτελέσει ένα πραγματικό αυτοκίνητο.

Έχοντας αναφέρει αυτά, οι κινήσεις που μπορεί να εκτελέσει ένα αυτοκίνητο είναι οι εξής:

- Επιτάχυνση: Ένα αυτοκίνητο πρέπει να μπορεί να αυξάνει την ταχύτητα του, όταν ο οδηγός του το επιθυμεί. Έτσι με το ίδιο τρόπο το αυτοκίνητο αυτής της εργασίας θα πρέπει να έχει την δυνατότητα να μπορεί να αυξήσει την ταχύτητα του όταν αυτό θα πάρει αυτή την απόφαση μέσω της τεχνητής νοημοσύνης.
- Επιβράδυνση: Ένα αυτοκίνητο πρέπει να μπορεί να επιβραδύνει την ταχύτητα του, όταν ο οδηγός του το επιθυμεί. Έτσι με τον ίδιο τρόπο το αυτοκίνητο αυτής της εργασίας θα πρέπει να έχει την δυνατότητα να μπορεί να επιβραδύνει την ταχύτητα του όταν αυτό θα πάρει αυτή την απόφαση μέσω της τεχνητής νοημοσύνης.
- Περιστροφή: Ένα αυτοκίνητο πρέπει να έχει την δυνατότητα να αλλάξει την πορεία του προς τα αριστερά ή προς τα δεξιά, όταν ο οδηγός του το επιθυμεί. Έτσι με τον ίδιο τρόπο το αυτοκίνητο αυτής της εργασίας θα πρέπει να έχει την λειτουργία της περιστροφής όταν αυτό θα πάρει αυτή την απόφαση μέσω της τεχνητής νοημοσύνης.

Μέσα σε αυτό το αρχείο θα δημιουργηθεί μια νέα κλάση με όνομα Actions, έτσι ώστε να μπορέσουν να αρχικοποιηθούν τα παρακάτω χαρακτηριστικά.

Η κλάση Actions έχει τέσσερα χαρακτηριστικά κλάσης, τα TURN_LEFT, TURN_RIGHT, ACCELERATE και BRAKE. Σε καθένα από αυτά τα χαρακτηριστικά εκχωρείται μια μοναδική ακέραια τιμή, που κυμαίνεται από 0 έως 3.

Στο χαρακτηριστικό TURN_LEFT εκχωρείται η τιμή μηδέν (0). Αυτό αντιπροσωπεύει μια ενέργεια που το αυτοκίνητο πρέπει να στρίψει αριστερά.

Στο χαρακτηριστικό TURN_RIGHT εκχωρείται η τιμή ένα (1). Αυτό αντιπροσωπεύει μια ενέργεια που το αυτοκίνητο πρέπει να στρίψει δεξιά.

Στο χαρακτηριστικό ACCELEREATE εκχωρείται η τιμή δύο (2).Αυτό αντιπροσωπεύει μια ενέργεια που το αυτοκίνητο πρέπει να αυξήσει την ταχύτητά του.

Στο χαρακτηριστικό BRAKE εκχωρείται η τιμή τρία (3). Αυτό αντιπροσωπεύει μια ενέργεια που το αυτοκίνητο πρέπει να μειώσει την ταχύτητά του ή να σταματήσει.

Έτσι κατά αυτόν τον τρόπο αυτές οι σταθερές που έχουν δηλωθεί στην κλάση Actions, μπορούν να χρησιμοποιηθούν με ευκολία σε άλλα μέρη του κώδικα και πιο συγκεκριμένα, όταν θα αναπτυχθεί η λογική της τεχνητής νοημοσύνης.

Συνολικά ο τρόπος σύνταξης της παραπάνω κλάσης και των σταθερών της, φαίνεται στη παρακάτω εικόνα.

```
1  class Actions:
2      TURN_LEFT = 0
3      TURN_RIGHT = 1
4      ACCELERATE = 2
5      BRAKE = 3
6
```

Figure 28 Κλάση Actions

2.4 Δήλωση αρχείου Car_AI

Το επόμενο βήμα για έρθουμε πιο κοντά στην ολοκλήρωση αυτής της εργασίας, είναι η υλοποίηση της τεχνητής νοημοσύνης που θα λαμβάνει τις απαραίτητες αποφάσεις, έτσι ώστε ένα αυτοκίνητο να μπορεί να εκπαιδευτεί στο να κινείται αυτόνομα σε μια πίστα.

Έχοντας λάβει υπόψη τα προαναφερθέντα χαρακτηριστικά προηγούμενου κεφαλαίου, σχετικά με αυτά που πρέπει να έχει μια τεχνητή νοημοσύνη για την συγκεκριμένη εργασία, θα ακολουθήσει επεξηγηματικά η ανάπτυξη της.

Σε αυτό το σημείο θα δημιουργηθεί ένα αρχείο με όνομα car_ai.py στον φάκελο car που έχουμε δημιουργήσει στον φάκελο αυτής της εργασίας. Μέσα σε αυτό το αρχείο

θα δημιουργηθεί μια νέα κλάση με όνομα CarAi, έτσι ώστε να μπορέσουν να αρχικοποιηθούν τα παρακάτω χαρακτηριστικά καθώς και η λειτουργικότητα της.

2.4.1 Δήλωση βιβλιοθηκών, σταθερών και κλάσης CarAI

Ξεκινώντας την διαδικασία της υλοποίησης, θα πρέπει να πρώτα να δηλωθούν οι βιβλιοθήκες και τα αρχεία που θα χρησιμοποιηθούν. Αυτές οι δηλώσεις εισαγωγής τοποθετούνται συνήθως στην κορυφή ενός αρχείου Python και καθιστούν τη λειτουργικότητα των εισαγόμενων λειτουργικών μονάδων και κλάσεων διαθέσιμη για χρήση στον υπόλοιπο κώδικα. Έτσι, τα τέσσερα αρχεία και βιβλιοθήκες που θα εισαχθούν είναι τα παρακάτω.

- import neat: Αυτή η γραμμή εισάγει τη βιβλιοθήκη NEAT (NeuroEvolution of Augmenting Topologies). και είναι η μέθοδος για την εξέλιξη των τεχνητών νευρωνικών δικτύων.
- from car.car import Car: Αυτή η γραμμή εισάγει την κλάση Car από το αρχείο που δημιουργήθηκε και αναπτύχθηκε σε προηγούμενο κεφάλαιο.
- from car.actions import Actions: Αυτή η γραμμή εισάγει την κλάση Actions από το αρχείο που δημιουργήθηκε και αναπτύχθηκε στο προηγούμενο κεφάλαιο.
- import pygame: Αυτή η γραμμή εισάγει τη βιβλιοθήκη Pygame. Το Pygame είναι ένα σύνολο λειτουργικών μονάδων Python που έχουν σχεδιαστεί για τη σύνταξη βιντεοπαιχνιδιών και στην παρούσα εργασία το περιβάλλον που θα εκπαιδευτεί το όχημα να κινείται.

Στη συνέχεια θα δηλωθεί η κλάση CarAI καθώς και τα δύο χαρακτηριστικά που θα έχει ώς μεταβλητές, τις TOTAL_GENERATIONS και TIME_TO_LIVE.

Το χαρακτηριστικό TOTAL_GENERATIONS, αρχικοποιείται με την τιμή μηδέν (0). Αυτή η μεταβλητή μπορεί να χρησιμοποιηθεί για να παρακολουθεί πόσες γενιές τεχνητής νοημοσύνης έχουν δημιουργηθεί. Στο πλαίσιο της τεχνητής νοημοσύνης, μια γενιά αναφέρεται συνήθως σε μια ομάδα περιπτώσεων τεχνητής νοημοσύνης που δημιουργούνται, αξιολογούνται και στη συνέχεια χρησιμοποιούνται για τη δημιουργία της επόμενης γενιάς. Αυτή είναι μια κοινή πρακτική στους γενετικούς αλγόριθμους, όπου κάθε γενιά αναμένεται να έχει καλύτερες επιδόσεις από την προηγούμενη.

Το χαρακτηριστικό TIME_TO_LIVE έχει οριστεί με την τιμή δεκαπέντε (15). Αυτό αντιπροσωπεύει το χρονικό διάστημα σε δευτερόλεπτα που επιτρέπεται στον αλγόριθμο να ζήσει ή να λειτουργήσει. Μετά την πάροδο αυτού του χρόνου, ο αλγόριθμος μπορεί να τερματιστεί. Αυτή είναι μια κοινή πρακτική σε προσομοιώσεις όπου οι περιπτώσεις τεχνητής νοημοσύνης έχουν περιορισμένη διάρκεια ζωής για να ενθαρρύνουν τη γρήγορη μάθηση και προσαρμογή.

Συνολικά ο τρόπος σύνταξης της παραπάνω κλάσης και των σταθερών της, φαίνεται στη παρακάτω εικόνα.

```
1   import neat        You, 3 months ago •
2   from car.car import Car
3   from car.actions import Actions
4   import pygame
5
6
    You, 3 months ago | 1 author (You)
7   class CarAI:
8
9       TOTAL_GENERATIONS = 0
10      TIME_TO_LIVE = 15
```

Figure 29 Κλάση CarAI

2.4.2 Δήλωση μεθόδου init()

Η πρώτη μέθοδος που θα αναλυθεί είναι η μέθοδος init(). Αυτή είναι μια μέθοδος καταστεκαυστής και είναι υπεύθυνη για την αρχικοποίηση τιμών στις μεταβλητές που προσδιορίζουν την κλάση CarAI.

Αρχικά η μέθοδος αυξάνει τη μεταβλητή κλάσης TOTAL_GENERATIONS, η οποία παρακολουθεί τον συνολικό αριθμό των γενεών που δημιουργήθηκαν. Στη συνέχεια αρχικοποιεί την κλάση με τα γονιδιώματα (genomes) και το περιεχόμενο του αρχείου διαμόρφωσης (config), που παρέχονται από τον αλγόριθμο NEAT και μια θέση εκκίνησης για τα αυτοκίνητα (start_position). Τα γονιδιώματα χρησιμοποιούνται για τη δημιουργία νευρωνικών δικτύων και για κάθε γονιδίωμα δημιουργείται ένα αντικείμενο Car [10].

Η μέθοδος, δημιουργεί επίσης μια λίστα νευρωνικών δικτύων (self.nets) και μια λίστα αυτοκινήτων (self.cars). Κάθε νευρωνικό δίκτυο που δημιουργείται από ένα γονιδίωμα χρησιμοποιώντας τη μέθοδο neat.nn.FeedForwardNetwork.create() και κάθε αυτοκίνητο δημιουργείται με τον κατασκευαστή της κλάσης Car. Η παράμετρος start_position μεταβιβάζεται στον κατασκευαστή του αυτοκινήτου για να ορίσει την αρχική θέση κάθε αυτοκινήτου [11].

Στη συνέχεια, η μέθοδος αρχικοποιεί την φυσική κατάσταση ή βαθμολογία (fitness) κάθε γονιδιώματος με την τιμή μηδέν (0). Αυτή η φυσική κατάσταση είναι ένα μέτρο απόδοσης για ένα γονιδίωμα (και επομένως το αντίστοιχο αυτοκίνητο). Χρησιμοποιείται από τον αλγόριθμο NEAT για να προσδιορίσει ποια γονιδιώματα (και τα σχετικά νευρωνικά δίκτυα τους) θα πρέπει να διατηρηθούν και να επιτραπεί να αναπαραχθούν ή εξελιχθούν στην επόμενη γενιά.

Τέλος, η μέθοδος ορίζει το χαρακτηριστικό self.remaining_cars, με τον συνολικό αριθμό των αυτοκινήτων. Αυτή η μεταβλητή παρακολουθεί πόσα αυτοκίνητα είναι ακόμα ζωντανά (δηλαδή, δεν έχουν τρακάρει ή έχουν αφαιρεθεί με άλλο τρόπο από την προσομοίωση).

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
12      def __init__(self, genomes: neat.DefaultGenome, config: neat.Config, start_position: list):
13          CarAI.TOTAL_GENERATIONS += 1
14          self.genomes = genomes
15          self.nets = [neat.nn.FeedForwardNetwork.create(
16              genome, config) for _, genome in genomes]
17          self.cars = [Car(start_position) for _ in genomes]
18          self.best_fitness = 0
19
20          for _, genome in genomes:
21              genome.fitness = 0
22
23          self.remaining_cars = len(self.cars)
```

Figure 30 Μέθοδος init

2.4.3 Δήλωση μεθόδου decisions()

Η επόμενη μέθοδος ονομάζεται decisions(). Αυτή η μέθοδος δέχεται δύο ορίσματα στη λίστα παραμέτρων, τα self και environment. Η παράμετρος self υποδηλώνει ότι αυτή η συνάρτηση είναι μέρος της κλάσης Car. Η παράμετρος environment υποδηλώνει ότι αυτή η μέθοδος δέχεται τον χάρτη της πίστας που θα κινείται το αυτοκίνητο, έτσι ώστε να κάνει τους απαραίτητους ελέγχους. Αυτή η μέθοδος έχει σχεδιαστεί έτσι ώστε, η τεχνητή νοημοσύνη να μπορεί να λαμβάνει αποφάσεις για την επόμενη κίνηση του κάθε αυτοκινήτου, αφού πρώτα τις έχουν παραχωρηθεί τα απαραίτητα δεδομένα που λαμβάνει το κάθε αυτοκίνητο από το περιβάλλον στο οποίο υπάρχει.

Έτσι η μέθοδος ξεκινά μπαίνοντας σε έναν βρόχο επανάληψης στο χαρακτηριστικό self.genomes χρησιμοποιώντας τη συνάρτηση απαρίθμησης της Python (enumerate). Το self.genomes είναι μια λίστα όπου κάθε στοιχείο είναι μια πλειάδα που περιέχει ένα αναγνωριστικό γονιδιώματος και ένα γονιδίωμα. Η συνάρτηση απαρίθμησης χρησιμοποιείται για τη λήψη τόσο του δείκτη τιμής των δεδομένων όσο και της τιμής κάθε στοιχείου στη λίστα. Ο δείκτης αποθηκεύεται στο i και η πλειάδα στο (genome_id, genome).

Κατά αυτόν τον τρόπο για κάθε γονιδίωμα, η συνάρτηση ανακτά το αντίστοιχο αυτοκίνητο και νευρωνικό δίκτυο από τα χαρακτηριστικά self.cars και self.nets αντίστοιχα. Στη συνέχεια λαμβάνει ορισμένα δεδομένα από το αυτοκίνητο χρησιμοποιώντας την μέθοδο car.get_data(), που έχει αναλυθεί σε προηγούμενο κεφάλαιο, και τα τροφοδοτεί στο νευρωνικό δίκτυο χρησιμοποιώντας την μέθοδο net.activate(car_data), που παρέχεται από την βιβλιοθήκη της Python NEAT. Η έξοδος του νευρωνικού δικτύου είναι μια λίστα τιμών και ο δείκτης της μέγιστης τιμής

αποθηκεύεται σε μία μεταβλητή που θα καθορίσει την επιλογή της κίνησης (choice) του αυτοκινήτου [11].

Στη συνέχεια, η επιλογή της κίνησης (choice) αντιστοιχίζεται με ένα σύνολο προκαθορισμένων ενεργειών, δηλαδή οι ενέργειες που αναπτύχθηκαν στην κλάση Actions, χρησιμοποιώντας τη δήλωση αντιστοίχισης της Python (match case), η οποία είναι παρόμοια με μια δήλωση διακόπτη (switch statement) σε άλλες γλώσσες προγραμματισμού. Ανάλογα με την τιμή της επιλογής, το αυτοκίνητο είτε θα στρίψει αριστερά, θα στρίψει δεξιά, θα επιταχύνει ή θα φρενάρει.

Αμέσως μετά την επιλογή κίνησης, η μέθοδος ελέγχει εάν το αυτοκίνητο είναι ακόμα ζωντανό μετά τη λήψη της απόφασής του, η κατάσταση του αυτοκινήτου ενημερώνεται μέσω της μεθόδου update_car_state() της κλάσης του αυτοκινήτου, με βάση το περιβάλλον. Στη συνέχεια η φυσική κατάσταση ή βαθμολογία του γονιδιώματος (fitness) αυξάνεται με την ανταμοιβή που λαμβάνεται από την μέθοδο get_reward() της κλάσης του αυτοκινήτου.

Τέλος, η συνάρτηση ενημερώνει το χαρακτηριστικό self.remaining_cars με τον αριθμό των αυτοκινήτων που είναι ακόμα ζωντανά και το χαρακτηριστικό self.best_fitness με την καταλληλότητα του γονιδιώματος που έχει την υψηλότερη φυσική κατάσταση ή βαθμολογία.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```python
    def decisions(self, environment: pygame.Surface) -> None:
        for i, (genome_id, genome) in enumerate(self.genomes):
            car = self.cars[i]
            net = self.nets[i]
            car_data = car.get_data()
            output = net.activate(car_data)
            choice = output.index(max(output))

            match choice:
                case Actions.TURN_LEFT:
                    car.turn_left()
                case Actions.TURN_RIGHT:
                    car.turn_right()
                case Actions.ACCELERATE:
                    car.accelerate()
                case Actions.BRAKE:
                    car.brake()

            if car.alive:
                car.update_car_state(environment)
                genome.fitness += car.get_reward()

        self.remaining_cars = len([car for car in self.cars if car.alive])
        self.best_fitness = max(
            self.genomes, key=lambda genome: genome[1].fitness)[1].fitness

```

Figure 31 Μέθοδος decisions

2.5 Αρχείο διαμόρφωσης

Σε αυτό το κεφάλαιο θα πρέπει να μελετηθεί το αρχείο διαμόρφωσης που θα χρειαστεί η βιβλιοθήκη της Python NEAT έτσι ώστε να μπορεί να λειτουργήσει. Ένα αρχείο διαμόρφωσης NEAT στην Python είναι ένα αρχείο κειμένου που καθορίζει τις παραμέτρους για την εκτέλεση ενός αλγορίθμου NEAT. Το NEAT είναι μια μέθοδος για την εξέλιξη τεχνητών νευρωνικών δικτύων μέσω μιας διαδικασίας μετάλλαξης και επιλογής δειγμάτων (specimens). Μεταβάλλει τόσο τις παραμέτρους στάθμισης όσο και τις δομές των δικτύων, προσπαθώντας να βρει μια ισορροπία μεταξύ της καταλληλότητας και της πολυπλοκότητας κατά τη διάρκεια της εξέλιξης. Το αρχείο διαμόρφωσης περιέχει διάφορες παραμέτρους που ελέγχουν τη διαδικασία εξέλιξης και τις ιδιότητες των δημιουργούμενων νευρωνικών δικτύων. Αυτές οι παράμετροι περιλαμβάνουν το μέγεθος του πληθυσμού, τα ποσοστά διαφορετικών τύπων μεταλλάξεων, τα κατώτατα όρια συμβατότητας για τα είδη (species) και πολλά άλλα. Πιο συγκεκριμένα ένα τέτοιο αρχείο διαμόρφωσης χωρίζεται σε πέντε ενότητες. Αυτές οι ενότητες είναι οι [NEAT], [DefaultGenome], [DefaultSpeciesSet], [DefaultStagnation] και [DefaultReproduction].

Σε αυτό το σημείο θα γίνει ανάλυση των παραπάνω ενοτήτων, με τις τιμές που έχουν οριστεί για την συγκεκριμένη εργασία [10].

1. [NEAT]
    1. fitness_criterion: Αυτή η παράμετρος καθορίζει τη μέθοδο που χρησιμοποιείται για την αξιολόγηση της καταλληλότητας ενός γονιδιώματος. Σε αυτήν την περίπτωση, έχει οριστεί στο μέγιστο, πράγμα που σημαίνει ότι το γονιδίωμα με την υψηλότερη βαθμολογία φυσικής κατάστασης θεωρείται το καλύτερο.
    2. fitness_threshold: Αυτή είναι η βαθμολογία φυσικής κατάστασης που πρέπει να φτάσει ένα γονιδίωμα για να σταματήσει τη διαδικασία εξέλιξης. Εδώ, έχει οριστεί σε 100000000. Εάν ένα γονιδίωμα φτάσει ή υπερβεί αυτό το σκορ, η διαδικασία εξέλιξης θα σταματήσει.
    3. pop_size: Αυτή η παράμετρος καθορίζει το μέγεθος του πληθυσμού για κάθε γενιά. Εδώ, έχει οριστεί σε 50, που σημαίνει ότι θα υπάρχουν 50 γονιδιώματα σε κάθε γενιά.
    4. reset_on_extinction: Εάν αυτή η παράμετρος οριστεί σε True, ο πληθυσμός θα μηδενιστεί εάν όλα τα είδη εξαφανιστούν και θα δημιουργηθεί πληθυσμός εκ νέου. Σε αυτήν τη διαμόρφωση, έχει οριστεί σε True.
2. [DefaultGenome]
    1. node activation options: Αυτές οι παράμετροι ελέγχουν τη λειτουργία ενεργοποίησης για τους κόμβους του νευρωνικού δικτύου. Η προεπιλεγμένη συνάρτηση ενεργοποίησης είναι tanh και έχει ρυθμό μετάλλαξης 0,01.
    2. node aggregation options: Αυτές οι παράμετροι ελέγχουν τον τρόπο με τον οποίο συγκεντρώνονται οι είσοδοι σε έναν κόμβο. Η προεπιλεγμένη συνάρτηση συνάθροισης είναι το άθροισμα και έχει ρυθμό μετάλλαξης 0,01.
    3. node bias options: Αυτές οι παράμετροι ελέγχουν τις τιμές πόλωσης των κόμβων. Η αρχική μεροληψία είναι μια κατανομή Gauss με μέσο όρο 0,0 και τυπική απόκλιση 1,0. Η μέγιστη και η ελάχιστη τιμή πόλωσης είναι 30,0 και -30,0 αντίστοιχα. Η ισχύς μετάλλαξης είναι 0,5, ο ρυθμός μετάλλαξης είναι 0,7 και ο ρυθμός αντικατάστασης είναι 0,1.
    4. genome compatibility options: Αυτές οι παράμετροι χρησιμοποιούνται για τον υπολογισμό της απόστασης συμβατότητας μεταξύ των γονιδιωμάτων. Οι συντελεστές για ασυνεχή γονίδια και διαφορές βάρους είναι 1,0 και 0,5 αντίστοιχα.

5. connection add/remove rates: Αυτές οι παράμετροι ελέγχουν τις πιθανότητες προσθήκης ή διαγραφής μιας σύνδεσης. Και οι δύο πιθανότητες είναι 0,5.

6. connection enable options: Αυτές οι παράμετροι ελέγχουν εάν μια σύνδεση είναι ενεργοποιημένη και το ποσοστό μετάλλαξης. Από την προεπιλογή, μια σύνδεση είναι ενεργοποιημένη και το ποσοστό μετάλλαξης είναι 0,01.

7. feed_forward: Αυτή η παράμετρος ελέγχει εάν το δίκτυο είναι δίκτυο τροφοδοσίας. Εδώ, έχει οριστεί σε True.

8. initial_connection: Αυτή η παράμετρος καθορίζει τον τρόπο δημιουργίας των αρχικών συνδέσεων του δικτύου. Εδώ, έχει οριστεί σε πλήρη, πράγμα που σημαίνει ότι κάθε κόμβος εισόδου συνδέεται με κάθε κόμβο εξόδου.

9. node add/remove rates: Αυτές οι παράμετροι ελέγχουν τις πιθανότητες προσθήκης ή διαγραφής ενός κόμβου. Και οι δύο πιθανότητες είναι 0,2.

10. network parameters: Αυτές οι παράμετροι καθορίζουν τον αριθμό των κρυφών κόμβων, κόμβων εισόδου και κόμβων εξόδου στο δίκτυο. Εδώ, δεν υπάρχουν κρυφοί κόμβοι, 5 κόμβοι εισόδου και 4 κόμβοι εξόδου.

11. node response options: Αυτές οι παράμετροι ελέγχουν τις τιμές απόκρισης των κόμβων. Η αρχική απόκριση είναι μια κατανομή Gauss με μέσο όρο 1,0 και η τυπική απόκλιση 0,0. Οι μέγιστες και ελάχιστες τιμές απόκρισης είναι 30,0 και -30,0 αντίστοιχα. Η ισχύς μετάλλαξης, ο ρυθμός μετάλλαξης και ο ρυθμός αντικατάστασης είναι 0,0.

12. connection weight options: Αυτές οι παράμετροι ελέγχουν τις αρχικές τιμές των συντελεστών βαρύτητας των συνδέσεων. Ο μέσος όρος της κατανομής Gauss που χρησιμοποιείται για την προετοιμασία των βαρών σύνδεσης είναι 0 και η τυπική απόκλιση 1,0. Οι μέγιστες και ελάχιστες τιμές απόκρισης είναι 30,0 και -30,0 αντίστοιχα. Η ισχύς μετάλλαξης είναι 0.5, ο ρυθμός μετάλλαξης είναι 0.8 και ο ρυθμός αντικατάστασης είναι 0,1.

3. [DefaultSpeciesSet]

    1. compatibility_threshold: Αυτή η παράμετρος χρησιμοποιείται για να αποφασίσει πότε δύο γονιδιώματα ανήκουν στο ίδιο είδος. Είναι ένα μέτρο του πόσο όμοια πρέπει να είναι δύο γονιδιώματα για να θεωρούνται μέρος του ίδιου είδους. Εάν η απόσταση συμβατότητας μεταξύ δύο γονιδιωμάτων είναι μικρότερη από αυτό το όριο,

θεωρούνται ότι ανήκουν στο ίδιο είδος. Σε αυτή τη διαμόρφωση, το compatibility_threshold έχει οριστεί σε 2.0.

4. [DefaultStagnation]

    1. species_fitness_func: Αυτή η παράμετρος καθορίζει τον τρόπο υπολογισμού της καταλληλότητας ενός είδους. Οι επιλογές είναι συνήθως «max» (η καταλληλότητα του καλύτερου μέλους του είδους) ή «μέση» (η μέση καταλληλότητα όλων των μελών του είδους).

    2. max_stagnation: Αυτή η παράμετρος καθορίζει τον μέγιστο αριθμό γενεών που μπορεί να περάσει ένα είδος χωρίς να βελτιωθεί πριν θεωρηθεί στάσιμο και αφαιρεθεί από τον πληθυσμό. Εάν ένα είδος δεν παρουσιάζει καμία βελτίωση στην καταλληλότητά του για γενιές max_stagnation, θεωρείται στάσιμο.

    3. species_elitism: Αυτή η παράμετρος είναι μια μορφή ελιτισμού, μια έννοια στους γενετικούς αλγόριθμους όπου τα πιο κατάλληλα μέλη του πληθυσμού είναι εγγυημένα ότι θα μεταφερθούν στην επόμενη γενιά, διασφαλίζοντας ότι η ποιότητα του πληθυσμού δεν υποβαθμίζεται. Σε αυτήν την περίπτωση σημαίνει ότι, τα δύο κορυφαία είδη θα διατηρηθούν για την επόμενη γενιά, ακόμα κι αν δεν βελτιωθούν.

5. [DefaultReproduction]

    1. elitism: Αυτή η παράμετρος καθορίζει πόσα από τα πιο κατάλληλα γονιδιώματα από κάθε είδος μεταφέρονται στην επόμενη γενιά χωρίς μετάλλαξη. Αυτό διασφαλίζει ότι τα πιο επιτυχημένα γονιδιώματα θα συνεχίσουν να αποτελούν μέρος του πληθυσμού. Σε αυτή την περίπτωση τρία από αυτά τα γονιδιώματα θα μεταφερθούν ακέραια στην επόμενη γενιά.

    2. survival_threshold: Αυτή η παράμετρος εκφράζει το ποσοστό ενός είδους που του επιτρέπεται να αναπαραχθεί για την επόμενη γενιά. Επιλέγονται για αναπαραγωγή μόνο τα πιο κατάλληλα γονιδιώματα που ανήκουν πάνω από το 20% των αποδόσεων όλων των γονιδίων.

Ο τρόπος σύνταξης ενός τέτοιου αρχείου με τις παραπάνω ενότητες, φαίνεται στις παρακάτω εικόνες.

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 100000000
pop_size              = 50
reset_on_extinction   = True

[DefaultGenome]
# node activation options
activation_default      = tanh
activation_mutate_rate  = 0.1
activation_options      = tanh

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.1
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5
```

Figure 32 Αρχείο διαμόρφωσης 1

```
# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.1

feed_forward            = True
initial_connection      = full_direct

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2

# network parameters
num_hidden              = 100
num_inputs              = 5
num_outputs             = 4

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 30
weight_min_value        = -30
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 2.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 10
species_elitism      = 5

[DefaultReproduction]
elitism            = 3
survival_threshold = 0.2
```

Figure 33 Αρχείο διαμόρφωσης 2

2.6 Ανάπτυξη περιβάλλοντος

Με βάση τα χαρακτηριστικά που έχουν δοθεί σχετικά με το περιβάλλον και την γραφική του αναπαράσταση, θα πρέπει να σχεδιαστεί και να αναπτυχθεί του περιβάλλοντος καθώς και της ενσωμάτωσης της τεχνητής νοημοσύνης, έτσι ώστε να μπορέσει να ολοκληρωθεί η εργασία. Η λογική για να επιτευχθεί αυτό, θα συμπεριληφθεί στο αρχείο που θα δημιουργηθεί με όνομα environmet.py, καθώς και ένα αρχείο με όνομα colors.py στο οποίο θα δηλωθούν τα χρώματα που θα χρησιμοποιηθούν.

Στο αρχείο environmet.py, θα εισαχθούν οι απαραίτητες βιβλιοθήκες, θα αρχικοποιθούν οι σταθερές που θα χρησιμοποιηθούν και θα αναπτυχθούν οι παρακάτω μέθοδοι που δίνουν τη λειτουργικότητα της συγκεκριμένης εργασίας.

- init(): Αυτή η μέθοδος αρχικοποιεί την κλάση Environment με κάποιες μεταβλητές που χαρακτηρίζουν το περιβάλλον.
- _drawing_environment(): Αυτή η μέθοδος είναι υπεύθυνη για την λειτουργικότητα της χάραξης του δρόμου της πίστας.
- _placing_start_point(): Αυτή η μέθοδος είναι υπεύθυνη για την τοποθέτηση του σημείου αφετηρίας για το αυτοκίνητο.
- initialize_car_state(): Αυτή η μέθοδος είναι υπεύθυνη για την τοποθέτηση του αυτοκινήτου στο σημείο αφετηρίας.
- handle_quit_event(): Αυτή η μέθοδος είναι υπεύθυνη για τον τερματισμό της λειτουργίας της προσομοίωσης.
- handle_keyup_event(): Αυτή η μέθοδος είναι υπεύθυνη για να χειρίζεται την περίπτωση που πιεστεί πλήκτρο (spacebar) από τον χρήστη, μετά την χάραξη πορείας.
- handle_mousebutton_event(): Αυτή η μέθοδος είναι υπεύθυνη για να χειρίζεται την περίπτωση που πιεστεί κάποιο πλήκτρο από το ποντίκι του χρήστη και ρυθμίζει το μέγεθος του εργαλείου πινέλου κατά την χάραξη πορείας.
- initialize_environment(): Αυτή η μέθοδος είναι υπεύθυνη για να ρυθμίσει ή να προετοιμάσει τις διάφορες λειτουργίες και χαρακτηριστικά του περιβάλλοντος.
- run(): Αυτή η μέθοδος είναι υπεύθυνη για την εκκίνηση του προγράμματος όταν αυτό κληθεί.
- initiate_ai(): Αυτή η μέθοδος είναι υπεύθυνη για την εκκίνηση και αρχικοποίηση της τεχνητής νοημοσύνης όταν ξεκινήσει η προσομοίωση.
- run_simulation(): Αυτή η μέθοδος είναι υπεύθυνη για την εκκίνηση της προσομοίωσης.

2.6.1 Δήλωση κλάσης Colors

Στο αρχείο colors.py, θα δημιουργηθεί η κλάση Colors, καθώς είναι ένας απλός τρόπος οργάνωσης και αποθήκευσης μιας συλλογής τιμών που χαρακτηρίζουν χρώματα. Κάθε χρώμα αντιπροσωπεύεται ως μια πλειάδα τριών ακεραίων, που αντιστοιχούν στις τιμές Κόκκινο, Πράσινο και Μπλε (RGB) του χρωματικού μοντέλου.

Στο χρωματικό μοντέλο RGB, τα χρώματα δημιουργούνται συνδυάζοντας κόκκινο, πράσινο και μπλε φως σε διάφορες εντάσεις. Κάθε ένταση αντιπροσωπεύεται από μια ακέραια τιμή μεταξύ του μηδενός (0) και διακόσια πενήντα πέντε (255). Για παράδειγμα, το (255, 255, 255) αντιπροσωπεύει το λευκό, όπου όλα τα χρώματα είναι στη μέγιστη έντασή τους και το (0, 0, 0) αντιπροσωπεύει το μαύρο, όπου όλα τα χρώματα είναι στην ελάχιστη έντασή τους.

Για αυτό τον λόγο, στην κλάση Color θα οριστούν τα εξής πέντε χρώματα ως χαρακτηριστικά: WHITE, BLACK, GREEN, RED και BLUE. Σε αυτά τα χαρακτηριστικά θα αρχικοποιηθούν οι κατάλληλες τιμές έτσι ώστε το κάθε ένα να αντιπροσωπεύει το αντίστοιχο χρώμα με το χρωματικό μοντέλο RGB.

Συνολικά ο τρόπος σύνταξης της παραπάνω κλάσης και των σταθερών της, φαίνεται στη παρακάτω εικόνα.

```
1   class Color:        You, 3 month
2       WHITE = (255, 255, 255)
3       BLACK = (0, 0, 0)
4       GREEN = (0, 255, 0)
5       RED = (255, 0, 0)
6       BLUE = (0, 0, 255)
```

Figure 34 Κλάση Color

2.6.2 Δήλωση βιβλιοθηκών, σταθερών και κλάσης Environment

Ξεκινώντας την διαδικασία της ανάπτυξης του περιβάλλοντος, θα δηλωθούν πρώτα οι απαραίτητες βιβλιοθήκες, η κλάση Environment καθώς και οι σταθερές που θα χρησιμοποιηθούν έτσι ώστε να επιτευχθεί ο στόχος.

Στην κορυφή του αρχείου, εισάγονται οι εξής βιβλιοθήκες. Το pygame, ένα σύνολο λειτουργικών μονάδων Python που έχουν σχεδιαστεί για τη σύνταξη βιντεοπαιχνιδιών. neat, η βιβλιοθήκη για τη μέθοδο NeuroEvolution of Augmenting Topologies (NEAT), η οποία είναι μια μέθοδος για την εξέλιξη τεχνητών νευρωνικών δικτύων. Η βιβλιοθήκη χρόνος (Time), μια ενότητα για το χειρισμό εργασιών που σχετίζονται με το χρόνο. Στη συνέχεια εισάγονται και οι κλάσεις Car και CarAI που έχουν ήδη δημιουργηθεί και αναπτυχθεί.

Αμέσως μετά, η κλάση Environment ορίζεται με τα χαρακτηριστικά που την καθορίζουν. Αυτά τα χαρακτηριστικά περιλαμβάνουν το WIDTH και το HEIGHT, που αντιπροσωπεύουν τις διαστάσεις του περιβάλλοντος. FPS, το οποίο σημαίνει καρέ ανά δευτερόλεπτο και είναι ένα κοινό μέτρο του χρόνου που χρειάζεται μια οθόνη γραφικών για να ενημερωθεί. BRUSH_SIZE, MAX_BRUSH_SIZE και MIN_BRUSH_SIZE, τα οποία σχετίζονται με την λειτουργικότητα σχεδίασης ή χάραξης της πορείας του δρόμου στο περιβάλλον. Τα χαρακτηριστικά SCROLL_UP_BUTTON και SCROLL_DOWN_BUTTON, τα οποία αντιπροσωπεύουν τα κουμπιά του ποντικιού που χρησιμοποιούνται για την αύξηση και μείωση του μεγέθους του εργαλείου πινέλο, που θα χρησιμοποιηθεί για την χάραξη πορείας.

Συνολικά ο τρόπος σύνταξης της παραπάνω κλάσης και των σταθερών της, φαίνεται στη παρακάτω εικόνα.

```
1   import pygame
2   import neat
3   from car.car_ai import CarAI
4   from car.car import Car
5   import time
6   from environment.colors import Color
7   import os
8   import glob
9
10
    You, 7 days ago | 2 authors (You and others)
11  class Environment:
12
13      WIDTH = 1900
14      HEIGHT = 1000
15      FPS = 60
16      BRUSH_SIZE = 50
17      MAX_BRUSH_SIZE = 100
18      MIN_BRUSH_SIZE = 10
19      SCROLL_UP_BUTTON = 4
20      SCROLL_DOWN_BUTTON = 5
```

Figure 35 Κλάση Environment

2.6.3 Δήλωση μεθόδου init()

Η πρώτη μέθοδος που θα αναλυθεί είναι η μέθοδος init(). Αυτή είναι μια μέθοδος καταστεκαυστής και είναι υπεύθυνη για την αρχικοποίηση τιμών στις μεταβλητές που προσδιορίζουν το περιβάλλον. Το όρισμα self που χρησιμοποιείται είναι μια παράμετρος που κάνει αναφορά στην τρέχουσα παρουσία της κλάσης και

χρησιμοποιείται για την πρόσβαση σε μεταβλητές που ανήκουν στην ίδια κλάση. Το όρισμα NEAT_CONFIG είναι η παράμετρος που δέχεται από το αρχείο διαμόρφωσης που έχει αναλυθεί σε προηγούμενο κεφάλαιο. Το τελευταίο όρισμα είναι το MAX_SIMULATIONS, το οποίο δηλώνει τον μέγιστο αριθμό των προσομοιώσεων που θα εκτελεστούν.

Ξεκινώντας, αρχικοποιούνται πολλές μεταβλητές που χαρακτηρίζουν το περιβάλλον, συμπεριλαμβανομένων των NEAT_CONFIG, MAX_SIMULATIONS και τίτλου εργασίας. Ο τίτλος έχει οριστεί σε AI_Cars και χρησιμοποιείται ως λεζάντα κατά την ώρα της προσομοίωσης [12]. Αμέσως μετά αρχικοποιήται το μέγεθος της οθόνης που θα χρησιμοποιηθεί για την προσομοίωση, καθώς ορίζεται και το χρώμα της οθόνης σε άσπρο. Στη συνέχεια αρχικοποιούνται διάφορες μεταβλητές τύπου boolean (is_running, placing_start_point, ai_initialization) με την τιμή False. Αυτές οι μεταβλητές χρησιμοποιούνται ως σημαίες (flags) για τον έλεγχο της κατάστασης και της συμπεριφοράς της προσομοίωσης. Η μεταβλητή drawing_environment, που είναι και αυτή τύπου boolean, αρχικοποιήται με την τιμή True, έτσι ώστε όταν ξεκινήσει η προσομοίωση, να μπορεί ο χρήστης να χαράξει την πορεία που επιθυμεί να ακολουθήσει το αυτοκίνητο. Μια πολύ σημαντική μεταβλητή που ορίζεται, είναι η clock. Αυτή η μεταβλητή αρχικοποιήται με ένα αντικείμενο pygame.time.Clock, το οποίο μπορεί να χρησιμοποιηθεί για τον έλεγχο του ρυθμού καρέ (frame rate) της οθόνης pygame που θα δημιουργηθεί. Τέλος ορίζονται οι μεταβλητές, track (πίστα), το environment και το final_car_pos σε None. Αυτές οι μεταβλητές θα χρησιμοποιηθούν αργότερα στην προσομοίωση [13] [14].

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
22      def __init__(self, NEAT_CONFIG, MAX_SIMULATIONS):
23          self.NEAT_CONFIG = NEAT_CONFIG
24          self.MAX_SIMULATIONS = MAX_SIMULATIONS
25          self.title = "AI_Cars"
26          pygame.display.set_caption(self.title)
27          self.screen = pygame.display.set_mode(
28              (self.WIDTH, self.HEIGHT))
29          self.screen.fill(Color.WHITE)
30          self.is_running = False
31          self.clock = pygame.time.Clock()
32          self.drawing_environment = True
33          self.placing_start_point = False
34          self.ai_initialization = False
35          self.track = None
36          self.environment = None
37          self.car = Car([0, 0])
38          self.final_car_pos = None
```

Figure 36 Μέθοδος init

2.6.4 Δήλωση μεθόδου _drawing_environment()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται _drawing_environment(). Η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Environment. Αυτή η μέθοδος έχει σχεδιαστεί έτσι ώστε ο χρήστης να έχει την δυνατότητα σχεδίασης στο παράθυρο που δημιουργείται μέσω του Pygame, με βάση τις εισόδους του ποντικιού.

Η μέθοδος ξεκινά καλώντας την μέθοδος pygame.mouse.get_pressed(), η οποία επιστρέφει μια πλειάδα δυαδικών τιμών που αντιπροσωπεύουν την κατάσταση του πλήκτρο του ποντικιού που πιέστηκε. Η κατάσταση των πλήκτρων αποθηκεύεται στη μεταβλητή mousebutton_pressed [15].

Στη συνέχεια, η μέθοδος καλεί την μέθοδο pygame.mouse.get_pos(), το οποίο επιστρέφει μια πλειάδα που αντιπροσωπεύει την τρέχουσα θέση του δρομέα (cursor) του ποντικιού στην οθόνη. Αυτή η θέση αποθηκεύεται στη μεταβλητή mouse_pos [16].

Στη συνέχεια, η μέθοδος ελέγχει την κατάσταση του αριστερού κουμπιού του ποντικιού (που αντιπροσωπεύεται από το mouse_pressed[0]). Εάν πιεστεί το αριστερό κουμπί του ποντικιού (δηλαδή, το mouse_pressed[0] είναι True), η μέθοδος σχεδιάζει έναν μαύρο κύκλο στην οθόνη στην τρέχουσα θέση του ποντικιού. Το χρώμα του κύκλου έχει οριστεί σε μαύρο μέσω τη κλάσης Colors που έχει δημιουργηθεί (Color.BLACK), το κέντρο του ορίζεται στην τρέχουσα θέση του ποντικιού (mouse_pos)

και το μέγεθος του εργαλείου πινέλου (η ακτίνα του) που έχει οριστεί με το χαρακτηριστικό BRUSH_SIZE [5].

Εάν δεν πιεστεί το αριστερό πλήκτρο του ποντικιού, η μέθοδος ελέγχει την κατάσταση του δεξιού πλήκτρο του ποντικιού (που αντιπροσωπεύεται από το mouse_pressed[2]). Εάν πιεστεί το δεξί πλήκτρο του ποντικιού, η μέθοδος σχεδιάζει έναν λευκό κύκλο στην οθόνη στην τρέχουσα θέση του ποντικιού. Αυτή η λειτουργικότητα μας δίνει την δυνατότητα διόρθωσης τυχόν λάθη που μπορούν να συμβούν κατά την σχεδίαση. Το χρώμα του κύκλου έχει οριστεί σε άσπρο μέσω τη κλάσης Colors που έχει δημιουργηθεί (Color.WHITE), το κέντρο του ορίζεται στην τρέχουσα θέση του ποντικιού (mouse_pos) και το μέγεθος του εργαλείου πινέλου (η ακτίνα του) που έχει οριστεί με το χαρακτηριστικό BRUSH_SIZE [5].

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
40      def _drawing_environment(self):
41          mousebutton_pressed = pygame.mouse.get_pressed()
42          mouse_pos = pygame.mouse.get_pos()
43          if mousebutton_pressed[0]:
44              pygame.draw.circle(self.screen, Color.BLACK,
45                                 mouse_pos, self.BRUSH_SIZE)
46          elif mousebutton_pressed[2]:
47              pygame.draw.circle(self.screen, Color.WHITE,
48                                 mouse_pos, self.BRUSH_SIZE)
```

Figure 37 Μέθοδος _drawing_environment

2.6.5 Δήλωση μεθόδου _placing_start_point()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται _placing_start_point(). Η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Environment. Αυτή η μέθοδος έχει σχεδιαστεί έτσι ώστε ο χρήστης να έχει την δυνατότητα να τοποθετεί το αυτοκίνητο καθώς και την εικόνα του σε ένα σημείο της οθόνης που αυτός έχει επιλέξει.

Η συνάρτηση ξεκινά λαμβάνοντας την τρέχουσα θέση του ποντικιού χρησιμοποιώντας τη μέθοδο pygame.mouse.get_pos(), η οποία επιστρέφει μια πλειάδα δύο ακεραίων που αντιπροσωπεύουν τις συντεταγμένες x και y του δρομέα του ποντικιού (cursor). Στη συνέχεια ανακτά την εικόνα του αυτοκινήτου, την θέση του καθώς και το κέντρο του αυτοκινήτου από την κλάση Car και αρχικοποιεί αυτά τα δεδομένα στις μεταβλητές car_sprite, car_position και car_center αντιστοίχως [16].

Στη συνέχεια, η μέθοδος ελέγχει εάν το αριστερό πλήκτρο του ποντικιού δεν είναι πιεστεί καλώντας την μέθοδο pygame.mouse.get_pressed()[0]. Εάν το πλήκτρο δεν

πιεστεί, ενημερώνει τη θέση του αυτοκινήτου ώστε να είναι κεντραρισμένη γύρω από την τρέχουσα θέση του ποντικιού. Στη συνέχεια, το sprite του αυτοκινήτου σχεδιάζεται σε αυτή τη νέα θέση στην οθόνη χρησιμοποιώντας τη μέθοδο blit του αντικειμένου self.screen [6].

Εάν έχει πιεστεί το αριστερό πλήκτρο του ποντικιού, η συνάρτηση ορίζει το χαρακτηριστικό ai_initialization σε True, το χαρακτηριστικό placing_start_point σε False και αντιγράφει την τρέχουσα οθόνη, δηλαδή ότι έχει σχεδιαστεί για την πίστα καθώς και για το αυτοκίνητο, στο χαρακτηριστικό environment. Στη συνέχεια, σχεδιάζει το sprite του αυτοκινήτου στην κεντρική θέση του αυτοκινήτου και ενημερώνει το χαρακτηριστικό final_car_pos με την τρέχουσα θέση του αυτοκινήτου ώς την αφετηρία του.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
50      def _placing_start_point(self):
51          mouse_x, mouse_y = pygame.mouse.get_pos()
52          car_sprite = self.car.sprite
53          car_position = self.car.position
54          car_center = self.car.car_center
55
56          if not pygame.mouse.get_pressed()[0]:
57              car_position[0] = mouse_x - Car.CAR_SIZE_X / 2
58              car_position[1] = mouse_y - Car.CAR_SIZE_Y / 2
59              self.screen.blit(car_sprite, car_position)
60          else:
61              self.ai_initialization = True
62              self.placing_start_point = False
63              self.environment = self.screen.copy()
64              self.screen.blit(car_sprite, car_center)
65              self.final_car_pos = list(car_position)
```

Figure 38 Μέθοδος _placing_start_point

2.6.6 Δήλωση μεθόδου initialize_car_state()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται initialize_car(). Η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Environment. Αυτή η μέθοδος έχει σχεδιαστεί έτσι ώστε να αρχικοποιείται ή να ενημερώνεται η κατάσταση του αυτοκινήτου γραφικά, κατά την πάροδο της προσομοίωσης.

Αρχικά καλείται η μέθοδος pygame.display.update() για να ενημερώσει την οθόνη σύμφωνα με την δεδομένη κατάσταση του περιβάλλοντος. Εάν έχουν γίνει αλλαγές

στην οθόνη της προσομοίωσης, αυτή η λειτουργία θα κάνει αυτές τις αλλαγές ορατές στον χρήστη. Είναι μια κοινή λειτουργία να καλείτε τον κύριο βρόχο του παιχνιδιού, μετά από όλες τις εντολές σχεδίασης [17].

Στη συνέχεια γίνεται ένας έλεγχος για να διαπιστωθεί εάν η προσομοίωση βρίσκεται στην κατάσταση τοποθέτησης αρχικού σημείου για το αυτοκίνητο με το αντίστοιχο χαρακτηριστικό (placing_start_point). Εάν το χαρακτηριστικό placing_start_point έχει την τιμή True, εκτελείται η μέθοδος screen.blit(). Η συνάρτηση blit που παρέχεται από την βιβλιοθήκη Pygame χρησιμοποιείται για να σχεδιάσει μια εικόνα του αυτοκινήτου πάνω στην εικόνα του περιβάλλοντος [6].

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
67      def initialize_car_state(self):
68          pygame.display.update()
69          if self.placing_start_point:
70              self.screen.blit(self.track, (0, 0))
71
```

Figure 39 Μέθοδος initialize_car_state

2.6.7 Δήλωση μεθόδου handle_quit_event()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται handle_quit_event(). Η μέθοδος δέχεται ως όρισμα το self και το event. Η παράμετρος self υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Environment. Η παράμετρος event αντιπροσωπεύει ένα συμβάν που μπορεί να συμβεί κατά την προσομοίωση. Αυτή η μέθοδος έχει σχεδιαστεί έτσι ώστε να μπορεί να τερματιστεί ομαλά η προσομοίωση.

Αρχικά η μέθοδος, κάνει έναν έλεγχο για το αν ο τύπος του συμβάντος είναι pygame.QUIT. Το συμβάν pygame.QUIT ενεργοποιείται όταν ο χρήστης πιέσει το πλήκτρο κλεισίματος του παραθύρου του Pygame. Εάν ο τύπος του συμβάντος είναι όντως pygame.QUIT, καλείται η μέθοδος exit(). Αυτή η συνάρτηση είναι μια βασική μέθοδος Python που προκαλεί την έξοδο της τρέχουσας διαδικασίας με μια προαιρετική κατάσταση ή την επιστροφή στο σύστημα. Σε αυτήν την περίπτωση, δεν παρέχεται καμία κατάσταση, επομένως η διαδικασία θα εξέλθει με κατάσταση none.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
72      def handle_quit_event(self, event):
73          if event.type == pygame.QUIT:
74              exit()
```

Figure 40 Μέθοδος handle_quit_event

2.6.8 Δήλωση μεθόδου handle_keyup_event()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται handle_keyup_event(). Αυτή η μέθοδος δέχεται δύο ορίσματα στη λίστα παραμέτρων της, το self και event. Η παράμετρος self υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Environment. Η παράμετρος event αντιπροσωπεύει ένα συμβάν που μπορεί να συμβεί κατά την προσομοίωση. Αυτή η μέθοδος έχει σχεδιαστεί έτσι ώστε να διαπιστώνει πότε έχει πιεστεί ένα πλήκτρο από το πληκτρολόγιο.

Αρχικά η μέθοδος, κάνει έναν έλεγχο για το αν ο τύπος του συμβάντος είναι ένα συμβάν keyup και εάν το πλήκτρο που πιέστηκε είναι το πλήκτρο κενού διαστήματος (spacebar). Εάν και οι δύο περιπτώσεις ισχύουν, τότε η μέθοδος αλλάζει την κατάσταση των παρακάτω χαρακτηριστικών του περιβάλλοντος.

Πρώτα, η κατάσταση του χαρακτηριστικού drawing_environment αλλάζει σε False. Αυτό υποδεικνύει ότι, η χάραξη διαδρομής για το αυτοκίνητο έχει τελειώσει. Στη συνέχεια, η κατάσταση του χαρακτηριστικού placing_start_point αλλάζει σε True. Αυτό σημαίνει ότι η κατάσταση της προσομοίωσης είναι έτοιμη για να τοποθετηθεί το αρχικο σημείο αφετηρίας για το αυτοκίνητο. Τέλος ορίζεται το χαρακτηριστικό τη πίστας (track) ορίζεται με ένα αντίγραφο της οθόνης. Αυτό σημαίνει ότι η τρέχουσα κατάσταση της οθόνης αποθηκεύεται στο χαρακτηριστικό track όταν πιεστεί το πλήκτρο κενού διαστήματος. Η μέθοδος copy() δημιουργεί ένα νέο αντικείμενο που είναι αντίγραφο του υπάρχοντος αντικειμένου οθόνης, επομένως οι αλλαγές στο ένα δεν θα επηρεάσουν το άλλο.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
76      def handle_keyup_event(self, event):
77          if event.type == pygame.KEYUP and event.key == pygame.K_SPACE:
78              self.drawing_environment = False
79              self.placing_start_point = True
80              self.track = self.screen.copy()
```

Figure 41 Μέθοδος handle_keyup_event

2.6.9 Δήλωση μεθόδου handle_mousebuttondown_event()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται handle_mousebutton_event(). Αυτή η μέθοδος δέχεται δύο ορίσματα στη λίστα παραμέτρων της, το self και event. Η παράμετρος self υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Environment. Η παράμετρος event αντιπροσωπεύει ένα συμβάν που μπορεί να συμβεί κατά την προσομοίωση. Αυτή η μέθοδος έχει σχεδιαστεί έτσι ώστε να διαπιστώνει πότε έχει πιεστεί ένα πλήκτρο από το ποντίκι έτσι ώστε ο χρήστης να έχει την δυνατότητα αλλαγής του μεγέθους εργαλείου πινέλου.

Αρχικά η μέθοδος κάνει έναν έλεγχο για το αν ο τύπος συμβάντος είναι MOUSEBUTTONDOWN, καθώς αυτό αντιπροσωπεύει την πίεση πλήκτρου ποντικιού. Εάν ο τύπος συμβάντος είναι πράγματι ένα συμβάν με το πλήκτρο του ποντικιού, τότε ο κωδικός ελέγχει ποιο πλήκτρο πιέστηκε. Εάν πιέστηκε το πλήκτρο SCROLL_UP_BUTTON και το τρέχον μέγεθος εργαλείου πινέλου (BRUSH_SIZE) είναι μικρότερο από το χαρακτηριστικό μέγιστου μεγέθους (MAX_BRUSH_SIZE), τότε αυξάνεται το BRUSH_SIZE κατά μία (1) μονάδα. Αυτό αντιπροσωπεύει, την αύξηση του μεγέθους μιας βούρτσας σε ένα πρόγραμμα σχεδίασης όταν ο χρήστης κάνει κύλιση προς τα πάνω με το πλήκτρο κύλισης του ποντικιού.

Ομοίως, εάν πιεστεί το πλήκτρο SCROLL_DOWN_BUTTON και το τρέχον μέγεθος εργαλείου πινέλου BRUSH_SIZE είναι μεγαλύτερο από το χαρακτηριστικό ελάχιστου μεγέθους (MIN_BRUSH_SIZE), μειώνεται το BRUSH_SIZE κατά μία (1) μονάδα. Αυτό αντιπροσωπεύει τη μείωση του μεγέθους του πινέλου όταν ο χρήστης κάνει κύλιση προς τα κάτω με το πλήκτρο κύλισης του ποντικιού.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
82      def handle_mousebuttondown_event(self, event):
83          if event.type == pygame.MOUSEBUTTONDOWN:
84              if event.button == self.SCROLL_UP_BUTTON and self.BRUSH_SIZE < self.MAX_BRUSH_SIZE:
85                  self.BRUSH_SIZE += 1
86              elif event.button == self.SCROLL_DOWN_BUTTON and self.BRUSH_SIZE > self.MIN_BRUSH_SIZE:
87                  self.BRUSH_SIZE -= 1
```

Figure 42 Μέθοδος handle_mousebuttondown_event

2.6.10 Δήλωση μεθόδου initialize_environment()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται initialize_environment(). Η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Environment. Η συγκεκριμένη μέθοδος έχει σχεδιαστεί έτσι ώστε να αρχικοποιείται ή να ενημερώνεται το περιβάλλον ανάλογα με την κατάσταση που βρίσκεται.

Η μέθοδος initialize_environment() είναι η κύρια μέθοδος που ρυθμίζει το περιβάλλον για την προσομοίωση. Ελέγχει διάφορες συνθήκες και καλεί τις κατάλληλες μεθόδους με βάση αυτές τις συνθήκες. Εάν το χαρακτηριστικό drawing_environment είναι True, καλεί την μέθοδο _drawing_environment() που επιτρέπει την χάραξη πορείας. Εάν το χαρακτηριστικό placing_start_point είναι True, καλεί την μέθοδο _placing_start_point που επιτρέπει την αρχικοποίηση του σημείου αφετηρίας. Εάν το χαρακτηριστικό ai_initialization είναι True, καλεί την μέθοδο initiate_ai που εκκινεί την τεχνητή νοημοσύνη. Τέλος, καλεί την μέθοδο initialize_car_state() για την αρχικοποίηση ή ενημέρωση της γραφικής αναπαράστασης του αυτοκινήτου.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
89      def initialize_environment(self):
90          if self.drawing_environment:
91              self._drawing_environment()
92          if self.placing_start_point:
93              self._placing_start_point()
94          if self.ai_initialization:
95              self.initiate_ai()
96          self.initialize_car_state()
```

Figure 43 Μέθοδος initialize_environment

2.6.11 Δήλωση μεθόδου run()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται run(). Η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Environment. Η συγκεκριμένη μέθοδος έχει σχεδιαστεί έτσι ώστε να είναι ο κύριος βρόγχος επανάληψης που διατηρεί το περιβάλλον σε λειτουργία και χειρίζεται τα διάφορα συμβάντα που δέχεται από τον χρήστη.

Στην αρχή της μεθόδου, το χαρακτηριστικό running ορίζεται σε True. Αυτό το χαρακτηριστικό χρησιμοποιείται για τον έλεγχο του κύριου βρόγχου επανάληψης του περιβάλλοντος. Όσο η εκτέλεση είναι True, το περιβάλλον θα συνεχίσει να επεξεργάζεται τα πιθανά συμβάντα.

Στη συνέχεια, ορίζεται η μεταβλητή event_handlers, που είναι τύπου λεξικού (dictionary). Αυτό το λεξικό αντιστοιχίζει τους τύπους συμβάντων Pygame στις αντίστοιχες μεθόδους που διαχειρίζονται τα συμβάντα (events). Για παράδειγμα, εάν προκύψει ένα συμβάν pygame.QUIT, θα κληθεί η μέθοδος handle_quit_event().

Έτσι, όσο το χαρακτηριστικό running είναι αληθές, ο κύριος βρόγχος επανάληψης συνεχίζει να εκτελείται ελέγχοντας τα τυχόν συμβάντα. Μέσα στον βρόχο, η μέθοδος ανακτά όλα τα συμβάντα που έχουν συμβεί από την τελευταία επανάληψη χρησιμοποιώντας την μέθοδο pygame.event.get() [18]. Για κάθε συμβάν, ανακτά τον αντίστοιχο διακριτικό στοιχείο (key) από το λεξικό event_handlers. Εάν βρεθεί ότι, κάποιο από αυτά τα διακριτικά έχει ενεργοποιηθεί, καλείται η μέθοδος με το αντίστοιχο συμβάν.

Τέλος, η μέθοδος initialize_environment() καλείται στο τέλος κάθε επανάληψης του βρόχου, έτσι ώστε η γραφική αναπαράσταση του περιβάλλοντος να είναι ενημερωμένη.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
 98    def run(self) -> None:
 99        self.running = True
100        event_handlers = {
101            pygame.QUIT: self.handle_quit_event,
102            pygame.KEYUP: self.handle_keyup_event,
103            pygame.MOUSEBUTTONDOWN: self.handle_mousebuttondown_event,
104        }
105        while self.running:
106            for event in pygame.event.get():
107                handler = event_handlers.get(event.type)
108                if handler:
109                    handler(event)
110            self.initialize_environment()
```

Figure 44 Μέθοδος run

2.6.12 Δήλωση μεθόδου initiate_ai()

Η επόμενη μέθοδος που θα αναπτυχθεί ονομάζεται initiate_ai(). Η μέθοδος δέχεται ως όρισμα μόνο το self, το οποίο υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Environment. Η συγκεκριμένη μέθοδος έχει σχεδιαστεί για τη ρύθμιση και την εκκίνηση της τεχνητής νοημοσύνης, που βασίζεται στον αλγόριθμο NEAT.

Σε πρώτο βήμα, η μέθοδος ξεκινά δημιουργώντας ένα αντικείμενο διαμόρφωσης για το NEAT χρησιμοποιώντας πολλές προεπιλεγμένες ρυθμίσεις μαζί με το αρχείο διαμόρφωσης που έχει προσαρμοστεί σε προηγούμενο κεφάλαιο [10].

Στη συνέχεια, η μέθοδος ελέγχει την περίπτωση που υπάρχει κάποιο αποθηκευμένο νευρωνικό δίκτυο από προηγούμενη εκπαίδευση. Αυτό είναι πολύ χρήσιμο καθώς δίνει την λειτουργία μετεκπαίδευσης του νευρώνα ακόμα και σε ένα διαφορετικά σχεδιασμένο περιβάλλον, από αυτό που είχε εκπαιδευτεί αρχικά. Για να μπορέσει να συμβεί αυτό, καλείτε η μέθοδος glob.glob('NN/save-*') για την εύρεση όλων των αρχείων που υπάρχουν στον φάκελο με όνομα NN (συντομογραφία για Neural Network) που ξεκινούν με όνομα save-. Αυτά τα αρχεία ταξινομούνται κατά χρόνο και με αύξουσα φορά, δηλαδή το πιο πρόσφατο θα βρίσκεται στο τέλος αυτής της λίστας.

Σε περίπτωση που βρεθεί ένα τέτοιο αρχείο με επιτυχία, θα φορτωθεί το νευρωνικό δίκτυο καθώς και με το αρχείο διαμόρφωσης που περιέχει τις υπόλοιπες πληροφορίες που χρειάζεται ο αλγόριθμος. Εάν όχι (κάτι που θα προκαλούσε εξαίρεση), δημιουργείται ένας νέος πληθυσμός από την αρχή με την μέθοδο neat.Population(config) [19].

Στη συνέχεια, εντάσσονται αρκετοί ρεπόρτερ στον αλγόριθμο. Αυτοί οι ρεπόρτερ θα παρέχουν πληροφορίες σχετικά με την πρόοδο του αλγορίθμου όταν εκτελείται. Το neat.StdOutReporter εκτυπώνει στατιστικά στοιχεία στην κονσόλα, το neat.StatisticsReporter συλλέγει δεδομένα από γενιά σε γενιά και το neat.Checkpointer

αποθηκεύει την κατάσταση του αλγορίθμου κάθε 10 γενιές, ώστε να μπορεί να συνεχιστεί αργότερα [20].

Αφού ολοκληρωθεί αυτή η διαδικασία, εκκινείτε ή ενεργοποιείται η τεχνητή νοημοσύνη όταν κληθεί η συγκεκριμένη μέθοδος από το πρόγραμμα.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
112     def initiate_ai(self):
113         config = neat.config.Config(
114             neat.DefaultGenome,
115             neat.DefaultReproduction,
116             neat.DefaultSpeciesSet,
117             neat.DefaultStagnation,
118             self.NEAT_CONFIG
119         )
120 
121         try:
122             checkpoint_files = glob.glob('NN/save-*')
123             checkpoint_files.sort(key=os.path.getmtime)
124             latest_checkpoint = checkpoint_files[-1]
125             population = neat.Checkpointer.restore_checkpoint(
126                 latest_checkpoint)
127         except Exception:
128             population = neat.Population(config)
129 
130         population.add_reporter(neat.StdOutReporter(True))
131         population.add_reporter(neat.StatisticsReporter())
132 
133         population.add_reporter(neat.Checkpointer(
134             generation_interval=10, filename_prefix="NN/save-"))
135         population.run(self.run_simulation, self.MAX_SIMULATIONS)
```

Figure 45 Μέθοδος initiate_ai

2.6.13 Δήλωση μεθόδου run_simulation()

Η τελευταία μέθοδος που θα αναπτυχθεί ονομάζεται run_simulation(). Αυτή η μέθοδος έχει σχεδιαστεί έτσι ώστε να είναι ο κύριος βρόγχος επανάληψης της προσομοίωσης, όπου τα αυτοκίνητα αλληλεπιδρούν με το περιβάλλον. Η μέθοδος παίρνει τρία ορίσματα στην λίστα παραμέτρων της, το self, genomes και config. Η παράμετρος self υποδηλώνει ότι αυτή η μέθοδος είναι μέρος της κλάσης Environment. Η παράμετρος genomes είναι μια λίστα γονιδιωμάτων που αντιπροσωπεύουν την τεχνητή νοημοσύνη κάθε αυτοκινήτου. Το config είναι το αρχείο διαμόρφωσης για τον αλγόριθμο NEAT.

Έτσι η λειτουργία ξεκινά με τη δημιουργία ενός αντικειμένου του CarAI με τα παρεχόμενα γονιδιώματα, το αρχείο διαμόρφωσης και μια τελική θέση του

αυτοκινήτου. Στη συνέχεια εισέρχεται σε έναν βρόχο που συνεχίζεται μέχρι να σταματήσει η προσομοίωση.

Μέσα στον βρόχο, πρώτα ελέγχει για τυχόν συμβάντα που μπορούν να συμβούν από τον χρήστη κατά την διάρκεια της σχεδίασης της πίστας και στη συνέχεια η τοποθέτηση σημείου αφετηρίας για το αυτοκίνητο. Σε περίπτωση που εντοπιστεί το συμβάν QUIT, το πρόγραμμα τερματίζεται.

Στη συνέχεια, καλείται η μέθοδος decisions(), που έχει αναπτυχθεί στην κλάση CarAI. Αυτή η μέθοδος είναι υπεύθυνη για τη λήψη αποφάσεων για κάθε αυτοκίνητο με βάση την τρέχουσα κατάσταση του περιβάλλοντος. Όμως, εάν όλα τα αυτοκίνητα δεν είναι πλέον ενεργά, δηλαδή, έχουν συγκρουστεί και έχουν πεθάνει, ο βρόχος επανάληψης τερματίζει έτσι ώστε να τελειώσει η τρέχουσα γενιά προσομοίωσης.

Επίσης, η μέθοδος ελέγχει εάν ο χρόνος που έχει παρέλθει από την έναρξη της προσομοίωσης υπερβαίνει ένα ορισμένο όριο που έχει οριστεί. Εάν όντως συμβεί, ο βρόχος επανάληψης τερματίζει έτσι ώστε να τελειώσει η τρέχουσα γενιά προσομοίωσης, ακόμα και αν υπάρχουν ζωντανά ή ενεργά αυτοκίνητα στο περιβάλλον. Η χρησιμότητα αυτής της λειτουργίας είναι πολύ σημαντική για την εκπαίδευση της τεχνητής νοημοσύνης καθώς, ένα σύστημα τεχνητής νοημοσύνης που εκπαιδεύεται για πολύ μεγάλο χρονικό διάστημα, μπορεί ενδεχομένως να προσαρμοστεί υπερβολικά στα δεδομένα εκπαίδευσης. Αυτό σημαίνει ότι η τεχνητή νοημοσύνη μπορεί να έχει καλή απόδοση στα συγκεκριμένα δεδομένα στα οποία εκπαιδεύτηκε, αλλά μπορεί να μην γενικεύεται καλά σε νέα, μη ορατά δεδομένα. Ως αποτέλεσμα, η τεχνητή νοημοσύνη μπορεί να μην αποδίδει τόσο καλά σε περιπτώσεις πραγματικού κόσμου όσο κατά τη διάρκεια της εκπαίδευσης.

Αμέσως μετά, η μέθοδος σχεδιάζει στην οθόνη την κατάσταση του περιβάλλοντος και των αυτοκινήτων. Η λεζάντα του παραθύρου ενημερώνεται με πληροφορίες σχετικά με την τρέχουσα γενιά, τον αριθμό των αυτοκινήτων που είναι ακόμα ζωντανά, τον χρόνο που απομένει και την καλύτερη απόδοση φυσικής κατάστασης που έχει επιτευχθεί μέχρι στιγμής.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```python
    def run_simulation(self, genomes: neat.DefaultGenome, config: neat.Config) -> None:
        car_ai = CarAI(genomes, config, self.final_car_pos)
        timer = time.time()
        self.is_running = True
        while self.is_running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    exit()

            car_ai.decisions(self.environment)
            if car_ai.remaining_cars == 0:
                break

            time_remaining = time.time() - timer
            if time_remaining > CarAI.TIME_TO_LIVE:
                break

            self.screen.blit(self.environment, (0, 0))
            for car in car_ai.cars:
                car.draw(self.screen)

            generation_text = "Generation: " + str(car_ai.TOTAL_GENERATIONS)
            stil_alibe_text = "Still Alive: " + str(car_ai.remaining_cars)
            time_remaining_text = "Time Left: " + \
                str(round(CarAI.TIME_TO_LIVE - time_remaining, 2)) + "s"
            best_fitnes_text = "Best Fitness: " + \
                str(round(car_ai.best_fitness))
            pygame.display.set_caption(self.title + " - " + generation_text + " - " +
                                       stil_alibe_text + " - " + time_remaining_text + " - " + best_fitnes_text)

            self.initialize_car_state()
            self.clock.tick(self.FPS)
```

Figure 46 Μέθοδος run_simulation

2.7 Δήλωση αρχείου main

Η ανάπτυξη της λογικής του αυτοκινήτου, της τεχνητής νοημοσύνης και του περιβάλλοντος έχει τελειώσει και πλέων σε αυτό το σημείο θα αναπτυχθεί η κύρια λογική main, που ο μοναδικός σκοπός της είναι να λειτουργεί ως μία μέθοδος ενορχήστρωσης των λειτουργιών που έχουν αναπτυχθεί σε προηγούμενα κεφάλαια.

Η σύσταση της λογικής είναι πολύ απλή, καθώς αποτελείται από την εισαγωγή μιας κλάσης και της μεθόδου main. Αρχικά θα γίνει η εισαγωγή της κλάσης Environment, έτσι ώστε να μπορεί να εκινηθεί το περιβάλλον στο οποίο θα γίνει η προσομοίωση.

Στη συνέχεια ορίζονται δύο σταθερές, η NEAT_CONFIG, που είναι μια σταθερά τύπου χαρακτήρων, που δείχνει τη θέση του αρχείου διαμόρφωσης για τον αλγόριθμο NEAT. Η σταθερά MAX_SIMULATIONS, που είναι τύπου ακεραίου αριθμού αντιπροσωπεύει τον μέγιστο αριθμό προσομοιώσεων που μπορούν να εκτελεστούν πρίν τερματιστεί το πρόγραμμα της εργασίας.

Αμέσως μετά γίνεται δήλωση της μεθόδου main(). Αυτή η μέθοδος δημιουργεί ένα αντικείμενο τύπο Environment από την κλάση Environment, περνώντας ως παραμέτρους την θέση του αρχείο διαμόρφωσης και τον μέγιστο αριθμό των προσομοιώσεων. Αφού δημιουργήσει αυτό το αντικείμενο, καλή την μέθοδο run() που έχει αναπτυχθεί στην κλάση Environment, έτσι ώστε να μπορέσει να εκινηθεί το πρόγραμμα.

Τέλος, γίνεται ο έλεγχος if __name__ == "__main__", για να διασφαλιστεί ότι ορισμένα τμήματα του κώδικα εκτελούνται μόνο όταν το σενάριο εκτελείται απευθείας και όχι όταν εισάγεται ως λειτουργική μονάδα. Σε αυτήν την περίπτωση, η συνάρτηση main() καλείται μόνο όταν το σενάριο εκτελείται απευθείας. Εάν αυτό το σενάριο εισαχθεί ως λειτουργική μονάδα σε άλλο σενάριο, η συνάρτηση main() δεν θα κληθεί. Αυτός είναι ένας κοινός τρόπος της Python για τη δημιουργία ενός σεναρίου τόσο εισαγόμενο ως λειτουργικό στοιχείο όσο και εκτελέσιμο ως αυτόνομο σενάριο.

Συνολικά ο τρόπος σύνταξης της παραπάνω μεθόδου και των μεταβλητών της, φαίνεται στη παρακάτω εικόνα.

```
1   from environment.environment import Environment
2
3   NEAT_CONFIG = "car/config.txt"
4   MAX_SIMULATIONS = 1000      You, 3 months ago • changed repo
5
6
7   def main() -> None:
8       environment = Environment(NEAT_CONFIG, MAX_SIMULATIONS)
9       environment.run()
10
11
12  if __name__ == "__main__":
13      main()
14
```

Figure 47 Αρχείο main

3. Εκτέλεση Εργασίας

Η διαδικασία ανάπτυξης της λογικής της εργασίας έχει τελειώσει και σε αυτό το κεφάλαιο θα γίνει αναφορά για το πως θα εκτελεστεί η εργασία, με τρία διαφορετικά παραδείγματα. Αυτά τα τρία διαφορετικά παραδείγματα θα διαφέρουν στις τιμές που θα εισαχθούν στο αρχείο διαμόρφωσης που έχει αναπτυχθεί σε προηγούμενο κεφάλαιο. Πιο συγκεκριμένα, το χαρακτηριστικό που θα αλλάξει βρίσκεται στις παραμέτρους του δικτύου, στην ενότητα του DefaultGenome. Αλλάζοντας την συγκεκριμένη τιμή σε αυτό το χαρακτηριστικό, θα αλλάξει και ο αριθμός των ενδιάμεσων κρυφών επιπέδων στο νευρωνικό δίκτυο.

Στα νευρωνικά δίκτυα, το κρυφό επίπεδο αναφέρεται στα επίπεδα των νευρώνων που βρίσκονται μεταξύ του επιπέδου εισόδου και του επιπέδου εξόδου. Αυτά τα επίπεδα ονομάζονται κρυμμένα επειδή δεν εκτίθενται άμεσα στην είσοδο ή την έξοδο. Ως εκ τούτου, είναι ενδιάμεσα στρώματα που μετατρέπουν την είσοδο σε κάτι που μπορεί να χρησιμοποιήσει το επίπεδο εξόδου.

Κάθε νευρώνας σε ένα κρυφό επίπεδο λαμβάνει είσοδο από τους νευρώνες του προηγούμενου επιπέδου , εκτελεί κάποιους υπολογισμούς σε αυτόν, καθώς περιλαμβάνει βάρη και προκαταλήψεις (bias) και μετά περνάει το αποτέλεσμα στους νευρώνες στο επόμενο επίπεδο. Ο αριθμός των κρυφών επιπέδων και ο αριθμός των νευρώνων σε κάθε κρυφό επίπεδο είναι παράμετροι που μπορούν να προσαρμοστούν με βάση την πολυπλοκότητα του προβλήματος και την επιθυμητή απόδοση του νευρωνικού δικτύου.

Έχοντας αναφέρει τα παραπάνω, θα ακολουθήσουν τρία πειράματα με διαφορετικά κρυφά επίπεδα με τιμές μηδέν (0), δέκα (100) και εκατό (100).

Επιπρόσθετα θα πρέπει να γίνει αναφορά των χαρακτηριστικών του υπολογιστικού συστήματος που έχει χρησιμοποιηθεί για αυτή την εργασία. Τα κύρια χαρακτηριστικά που έχουν ιδιαίτερη σημασία στη εργασία, είναι η ισχύς του επεξεργαστή (CPU) και η μνήμη τυχαίας προσπέλασης (Ram). το σύστημα που έχει χρησιμοποιηθεί για την εκτέλεση αυτής της εργασίας διαθέτει οκτώ πυρήνες που χρονίζουν στα 3.6 GHz και μνήμη με χωρητικότητα 16 GB και χρονίζει στα 3.200 MHz.

3.1 Πρώτο πείραμα

Το πρώτο πείραμα που θα διεξαχθεί, θα παραμετροποιηθεί με μηδέν κρυφά επίπεδα. Η ύπαρξη μηδενικών κρυφών επιπέδων σε ένα νευρωνικό δίκτυο σημαίνει ότι υπάρχει μόνο ένα στρώμα νευρώνων, το οποίο συνδέει απευθείας την είσοδο με την έξοδο. Αυτός ο τύπος νευρωνικού δικτύου αναφέρεται συχνά ως perceptron μονής στρώσης ή ένα νευρωνικό δίκτυο τροφοδοσίας χωρίς κρυφά στρώματα.

Ωστόσο, τα δίκτυα με μηδενικά κρυφά επίπεδα έχουν περιορισμένη εκφραστικότητα και δεν είναι ικανά να μάθουν πολύπλοκα μοτίβα ή σχέσεις σε δεδομένα που δεν είναι γραμμικά διαχωριζόμενα. Είναι κυρίως χρήσιμα για απλές εργασίες ταξινόμησης ή προβλήματα παλινδρόμησης όπου η σχέση εισόδου-εξόδου είναι σχετικά απλή.

Ξεκινώντας το πρώτο πείραμα, δεν θα αλλάξει η τιμή της παραμέτρου num_hidden, διότι έχει αρχικά καθοριστεί μηδέν κατά την δημιουργία του αρχείου διαμόρφωσης config.txt. Για να εκκινηθεί η εργασία, πρέπει να τρέξει με εντολή από την κονσόλα. Έτσι, πρέπει να βρίσκεται κάποιος μέσα στον πηγαίο φάκελο της εργασίας με όνομα AI_Cars, στο ίδιο επίπεδο φακέλου που βρίσκεται και το αρχείο main.py. Για να συμβεί αυτό, μπορεί να χρησιμοποιηθεί το τερματικό CMD ή το ενσωματωμένο τερματικό το προγράμματος VS Code. Στην περίπτωση του τερματικού CMD, θα πρέπει να γίνει αλλαγή φακέλου με την εντολή αλλαγής φακέλου cd (change directory), προς τον φάκελο που στεγάζει την εργασία, όπως για παράδειγμα (cd C:\Users\Theodore\Desktop\workspace\AI_Cars). Στην περίπτωση του VS Code μπορεί να απλά να ανοίξει η εργασία στον πηγαίο φάκελο της, μέσω του ευρετηρίου που προσφέρει το ίδιο το πρόγραμμα.

Αφού πλέων κάποιος βρίσκεται στον πηγαίο φάκελο της εργασίας και στο ίδιο επίπεδο φακέλου με το αρχείο main.py, μένει να γίνει εκτέλεση του προγράμματος με την εντολή (python main.py). Η συγκεκριμένη εντολή θα εκτελέσει το πρόγραμμα και όταν ξεκινήσει προσομοίωση θα εμφανίζει στην κονσόλα τα μηνύματα των ρεπόρτερ που θα παρέχουν πληροφορίες σχετικά με την πρόοδο του αλγορίθμου όσο εκτελείται. Επίσης, η εργασία μπορεί να εκτελεστεί με την εντολή (python main.py -> report1.txt). Η μόνη διαφορά με αυτή την εντολή είναι στο ότι θα απενεργοποιήσει την ζωντανή διαδικασία παροχής πληροφοριών στην κονσόλα για όσο το πρόγραμμα εκτελείται, καθώς θα δημιουργεί ένα νέο αρχείο με όνομα report1.txt και αφού πρώτα τελειώσει η προσομοίωση ή τερματιστεί, θα γράψει αναλυτικά την αναφορά στο αρχείο. Αυτός ο τρόπος εκτέλεσης θεωρείται καλύτερος διότι κατά αυτόν τον τρόπο μπορούν να αποθηκευτούν η πληροφορίες από την αναφορά και να μελετηθούν σε δεύτερη φάση.

Μετά την εκτέλεση του προγράμματος, θα εμφανιστεί στην οθόνη του υπολογιστή ένα κενό παράθυρο από το Pygame που μέσα σε αυτό μπορεί να χαραχθεί η πίστα που θα κινείται το αυτοκίνητο, όπως έχει περιγραφεί σε προηγούμενο κεφάλαιο.

Στη συνέχεια, αφού ολοκληρωθεί η χάραξη της πίστας, θα πιεστεί το πλήκτρο κενού διαστήματος (spacebar) για να τοποθετηθεί το αυτοκίνητο στην αρχική του θέση. Με την τοποθέτηση του αυτοκινήτου στο περιβάλλον, ενεργοποιείται η τεχνητή νοημοσύνη και η προσομοίωση πλέων ξεκινά.

Για όση ώρα η προσομοίωση παραμένει ενεργή, το νευρωνικό δίκτυο εκπαιδεύεται ανάλογα με τις καταστάσεις που βρίσκεται το αυτοκίνητο. Για μια σωστή εκπαίδευση

χρειάζεται αρκετός χρόνος και έτσι προτείνεται ο αλγόριθμος να εκπαιδευτεί για τουλάχιστον πενήντα (50) γενιές. Αυτό θα έχει ως αποτέλεσμα όχι μόνο καλύτερη εκπαίδευση αλλά θα μπορούν να παρατηρηθούν σχετικές μεταλλάξεις στα νευρωνικά δίκτυα.

Στην παρακάτω εικόνα αναπαρίσταται μία προσομοίωση.

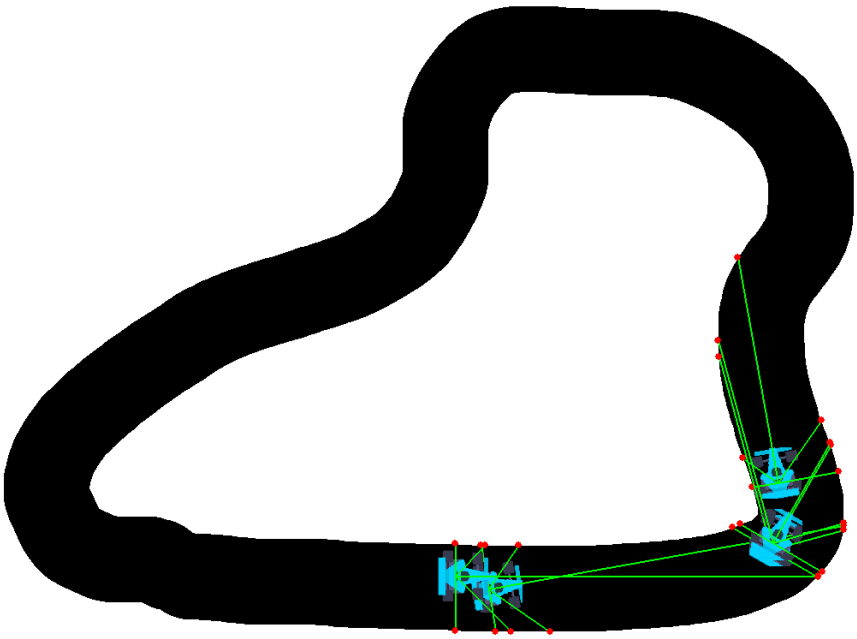

Figure 48 Πρώτο πείραμα

3.2 Δεύτερο πείραμα

Για το δεύτερο πείραμα θα πρέπει να παραμετροποιηθεί η τιμή του χαρακτηριστικού των κρυφών επιπέδων, num_hidden, στο αρχείο διαμόρφωσης config.txt, με την τιμή δέκα (10). Έχοντας δέκα κρυφά επίπεδα επιτρέπει στο νευρωνικό δίκτυο να μάθει περίπλοκες ιεραρχικές αναπαραστάσεις των δεδομένων εισόδου. Κάθε κρυφό επίπεδο μπορεί να μάθει να εξάγει διαφορετικά χαρακτηριστικά ή αναπαραστάσεις από την είσοδο και τα επόμενα επίπεδα μπορούν να μάθουν να συνδυάζουν αυτά τα χαρακτηριστικά για να κάνουν όλο και πιο αφηρημένες αναπαραστάσεις. Αυτή η ιεραρχική εκμάθηση αναπαράστασης επιτρέπει στα νευρωνικά δίκτυα με πολλαπλά κρυφά επίπεδα να μοντελοποιούν αποτελεσματικά σύνθετες σχέσεις και μοτίβα σε δεδομένα, καθιστώντας τα κατάλληλα για εργασίες όπως η αναγνώριση εικόνας, η επεξεργασία φυσικής γλώσσας και πολλά άλλα. Ωστόσο, τα βαθύτερα δίκτυα με περισσότερα κρυφά επίπεδα απαιτούν επίσης περισσότερους

υπολογιστικούς πόρους και μπορεί να είναι πιο απαιτητική για την αποτελεσματική εκπαίδευση.

Για να εκτελεστεί η εργασία θα χρησιμοποιηθεί η ίδια εντολή, όπως και στο πρώτο πείραμα, αλλάζοντας μόνο το όνομα του αρχείου που θα αποθηκευτούν τα δεδομένα της αναφοράς. Έτσι η εντολή που θα εκτελέσει την εργασία θα είναι η (python main.py -> report2.txt).

Ομοίως θα σχεδιαστεί το περιβάλλον και θα τοποθετηθεί το αυτοκίνητο ώστε να ξεκινήσει η προσομοίωση και αυτή θα τρέξει για τουλάχιστον πενήντα (50) γενιές.

Στην παρακάτω εικόνα αναπαρίσταται μία προσομοίωση.

Figure 49 Δεύτερο πείραμα

3.3 Τρίτο πείραμα

Για το δεύτερο πείραμα θα πρέπει να παραμετροποιηθεί η τιμή του χαρακτηριστικού των κρυφών επιπέδων, num_hidden, στο αρχείο διαμόρφωσης config.txt, με την τιμή εκατό (100). Όμως, όπως έχει αναφερθεί, τα περισσότερα κρυφά επίπεδα απαιτούν επίσης περισσότερους υπολογιστικούς πόρους και για αυτό το λόγο η γραφική αναπαράσταση της εργασίας κατά τη διάρκεια της προσομοίωσης, ενδεχομένως να είναι αργή. Στην περίπτωση που οι πόροι του συστήματος δεν επαρκούν, θα πρέπει να παραμετροποιηθεί το χαρακτηριστικό των κρυφών επιπέδων με μία μικρότερη τιμή, ώστε να καθιστά την προσομοίωση αποδοτική από άποψη χρόνου λειτουργίας.

Για να εκτελεστεί η εργασία θα χρησιμοποιηθεί η ίδια εντολή, όπως και στα προηγούμενα πειράματα, αλλάζοντας μόνο το όνομα του αρχείου που θα

αποθηκευτούν τα δεδομένα της αναφοράς. Έτσι η εντολή που θα εκτελέσει την εργασία θα είναι η (python main.py -> report3.txt).

Ομοίως θα σχεδιαστεί το περιβάλλον και θα τοποθετηθεί το αυτοκίνητο ώστε να ξεκινήσει η προσομοίωση και αυτή θα τρέξει για τουλάχιστον πενήντα (50) γενιές.

Στην παρακάτω εικόνα αναπαρίσταται μία προσομοίωση.

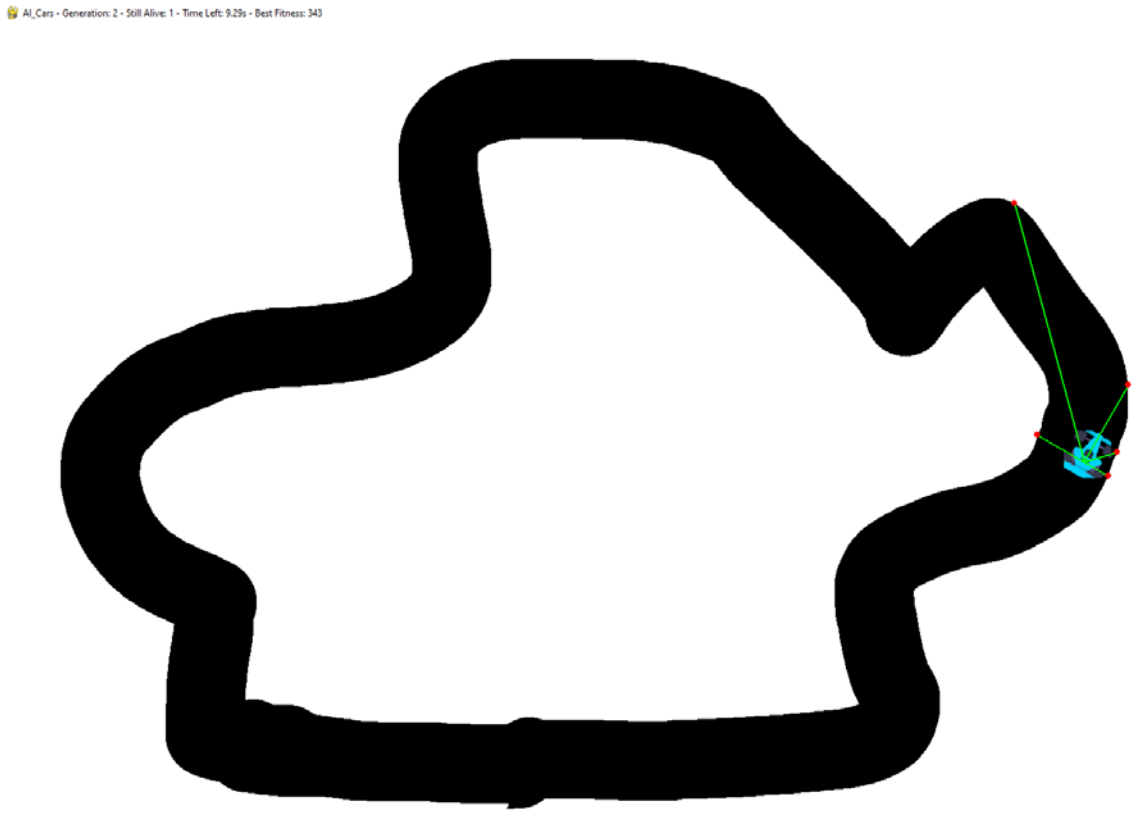

Figure 50 Τρίτο πείραμα

4. Αποτελέσματα

Μετά την εκτέλεση τριών πειραμάτων, υπάρχουν πλέων τρία αρχεία με τα δεδομένα της αναφοράς που έχουν προκύψει από τις προσομοιώσεις. Σε αυτό το κεφάλαιο θα γίνει η ανάλυση αυτών των δεδομένων και για τις τρεις περιπτώσεις. Πρώτα θα γίνει ανάλυση της σημασίας των δεδομένων που έχουν και στη συνέχεια, η ανάλυση των τριών πειραμάτων. Σε αυτό το σημείο θα ήταν καλό να αναφερθεί ότι, τα δεδομένα της αναφοράς κατά την διάρκεια των συγκεκριμένων πειραμάτων μπορεί να διαφέρουν από αυτά που μπορεί να λάβει ο αναγνώστης κατά τη διάρκεια της δικής του προσομοίωσης.

4.1 Ανάλυση δεδομένων αναφοράς

Αυτή η αναφορά είναι το αποτέλεσμα μιας προσομοίωσης γενετικού αλγορίθμου, ειδικά για έναν πληθυσμό αυτοκινήτων με τη χρήση τεχνητής νοημοσύνης. Τα στοιχεία της παρακάτω αναφοράς είναι τα εξης.

- Generation: Αυτή είναι η τρέχουσα επανάληψη της προσομοίωσης. Με κάθε γενιά, η τεχνητή νοημοσύνη προσαρμόζει τον πληθυσμό με βάση τα αποτελέσματα της προηγούμενης γενιάς.
- Population's average fitness: Αυτή είναι η μέση βαθμολογία καταλληλότητας όλων των μελών του πληθυσμού. ο βαθμός καταλληλότητας (Fitness) είναι ένα μέτρο της απόδοσης κάθε μέλους στην προσομοίωση.
- Best fitness: Αυτή είναι η υψηλότερη βαθμολογία καταλληλότητας που επιτυγχάνεται από ένα μέλος του πληθυσμού.
- Average adjusted fitness: Αυτή είναι η μέση βαθμολογία καταλληλότητας , προσαρμοσμένη ώστε να διατηρείται η σύγκριση μεταξύ των μελών.
- Mean genetic distance: Αυτή είναι η μέση διαφορά μεταξύ των γονιδίων όλων των μελών του πληθυσμού.
- Population of X members in Y species: Αυτό δείχνει τον αριθμό των διακριτών «ειδών» στον πληθυσμό. Κάθε είδος αντιπροσωπεύει μια διαφορετική προσέγγιση ή στρατηγική.
- ID, age, size, fitness, adj fit, stag: Αυτά είναι χαρακτηριστικά για κάθε είδος. ID είναι το αναγνωριστικό του είδους, age είναι ο αριθμός των γενεών που έχει υπάρξει το είδος, size είναι ο αριθμός των μελών του είδους, fitness είναι η υψηλότερη βαθμολογία καταλληλότητας εντός του είδους, adj fit είναι η προσαρμοσμένη βαθμολογία καταλληλότητας και stag είναι ο αριθμός των γενεών που το είδος έχει περάσει χωρίς βελτίωση ή στασιμότητα.
- Total extinctions: Αυτός είναι ο αριθμός των ειδών που έχουν εξαφανιστεί λόγω στασιμότητας.

- Generation time: Αυτός είναι ο χρόνος που χρειάστηκε για την εκτέλεση της προσομοίωσης για την τρέχουσα γενιά.

4.2 Ανάλυση πρώτου πειράματος

Όπως έχει αναφερθεί, το πρώτο πείραμα θα μηδέν (0) κρυφά επίπεδα και θα είναι ενεργό για πενήντα (50) γενιές. Έτσι αφού η διαδικασία έχει τελειώσει μπορούν να γίνουν οι παρακάτω παρατηρήσεις.

Κατά την εκκίνηση της προσομοίωσης δημιουργείται ένα είδος (species) με πληθυσμό πενήντα, πράγμα που σημαίνει ότι στο περιβάλλον θα υπάρχουν πενήντα αυτοκίνητα τα οποία θα εκπαιδεύονται να κινούνται στην πίστα. Μετά την πάροδο της πρώτης γενιάς εμφανίζονται τα εξής δεδομένα της πρώτης αναφοράς.

****** Running generation 0 ******

Population's average fitness: 63.13502 stdev: 53.02682

Best fitness: 205.68354 - size: (4, 20) - species 1 - id 16

Average adjusted fitness: 0.236

Mean genetic distance 1.126, standard deviation 0.274

Population of 50 members in 1 species:

ID	age	size	fitness	adj fit	stag
1	0	50	205.7	0.236	0

Total extinctions: 0

Generation time: 1.993 sec

Για τα συγκεκριμένα δεδομένα θα γίνουν οι εξής παρατηρήσεις.

- Running generation 0: Αυτό υποδηλώνει ότι εκτελείται η πρώτη γενιά (γενιά 0).
- Population's average fitness: 63.13502 stdev: 53.02682: Αυτή η γραμμή δείχνει τη μέση βαθμολογία καταλληλότητας του πληθυσμού και την τυπική απόκλιση του. Οι βαθμολογίες φυσικής κατάστασης χρησιμοποιούνται σε γενετικούς αλγόριθμους για να μετρηθεί πόσο καλά κάθε άτομο στον πληθυσμό λύνει το πρόβλημα.

- Best fitness: 205.68354 - size: (4, 20) - species 1 - id 16: Αυτή η γραμμή δείχνει τη βαθμολογία καταλληλότητας του καλύτερου ατόμου στον πληθυσμό, το μέγεθός του, το είδος στο οποίο ανήκει και το αναγνωριστικό του.
- Average adjusted fitness: 0.236: Αυτή είναι η μέση βαθμολογία καταλληλότητας του πληθυσμού, προσαρμοσμένη ώστε να αποτρέπεται η υπερβολικά γρήγορη σύγκλιση του αλγόριθμου σε μία κακή λύση.
- Mean genetic distance 1.126, standard deviation 0.274: Αυτή η γραμμή δείχνει τη μέση γενετική απόσταση, ένα μέτρο του πόσο διαφορετικά είναι τα άτομα του πληθυσμού μεταξύ τους και την τυπική τους απόκλιση.
- ID, age, size, fitness, adj fit, stag: Αυτή η γραμμή παρέχει λεπτομέρειες για το είδος 1. Είναι η πρώτη γενιά (age 0), έχει 50 μέλη, η καλύτερη βαθμολογία καταλληλότητας είναι 205,7, η μέση προσαρμοσμένη βαθμολογία φυσικής κατάστασης είναι 0,236 και δεν είναι στάσιμη.
- Total extinctions: 0: Αυτή η γραμμή δείχνει ότι ο αριθμός των ειδών που έχουν εξαφανιστεί είναι μηδέν.
- Generation time 1.993 sec: Αυτός είναι ο χρόνος που χρειάστηκε για την εκτέλεση της προσομοίωσης για την τρέχουσα γενιά. Ο προκαθορισμένος χρόνος προσομοίωσης είναι 15 δευτερόλεπτα. Η συγκεκριμένη προσομοίωση έτρεξε για περίπου 2 δευτερόλεπτα δίνοντας το συμπέρασμα ότι τα αυτοκίνητα συγκρούστηκαν με τα όρια της πίστας πριν ολοκληρωθεί η προσομοίωση.

Με την πάροδο των γενεών λαμβάνουμε όλα τα δεδομένα από τις αναφορές. Πιο συγκεκριμένα η τελευταία προσομοίωση έχει τα εξής δεδομένα

****** Running generation 49 ******

Population's average fitness: 1200.39814 stdev: 817.03518

Best fitness: 1780.62750 - size: (5, 10) - species 5 - id 1542

Average adjusted fitness: 0.659

Mean genetic distance 1.971, standard deviation 0.607

Population of 49 members in 6 species:

ID	age	size	fitness	adj fit	stag

1 49 7 1780.6 0.679 43

2 35 8 1780.6 0.667 9

3 35 11 1780.6 0.800 9

4 11 6 1780.6 0.441 9

5 10 8 1780.6 0.668 2

6 10 9 1780.6 0.698 9

Total extinctions: 0

Generation time: 15.487 sec (15.480 average)

Στη συγκεκριμένη προσομοίωση μπορεί πλέον εμφανώς να διακριθούν αρκετές διαφορές. Η πρώτη και πιο σημαντική παρατήρηση είναι ότι τα είδοι έχουν πολλαπλασιαστεί μέσω της λειτουργίας μετάλλαξης του αλγορίθμου. Η μέση βαθμολογία καταλληλότητας έχει αυξηθεί μαζί με την τυπική απόκλιση της. Αυτό σημαίνει ότι πλέων ένα αυτοκίνητο είναι πιο αποδοτικό στο να κινείται μέσα στο περιβάλλον του. Μια πολύ σημαντική παρατήρηση που μπορεί να γίνει, είναι ότι το πρώτο είδος έχει παραμείνει στάσιμο για 43 γενιές. Αυτό σημαίνει ότι η βαθμολογία καταλληλότητας του δεν έχει αλλάξει και ενδεχομένως εάν η προσομοίωση συνέχιζε για περισσότερες γενιές, αυτό το είδος θα αφανιζόταν έτσι ώστε ο αλγόριθμος να δημιουργήσει ένα νέο είδος που θα μπορούσε να παραστεί πιο κατάλληλο με την πάροδο των γενεών. Τέλος, μπορεί να παρατηρηθεί ότι ο χρόνος προσομοίωσης ειναι στα 15.487 δευτερόλεπτα καθώς ο μέσος όρος όλου του πληθυσμού είναι στα 15.480. Αυτό σημαίνει ότι πλέον τα αυτοκίνητα μπορούν να παραμείνουν ζωντανά καθ’ όλη τη διάρκεια της προσομοίωσης. κριμένα θα αναλυθεί η τελευταία γενιά έτσι ώστε να γίνει μια καλή σύγκριση.

4.3 Ανάλυση δεύτερου πειράματος

Το δεύτερο πείραμα έχει παραμετροποιηθεί έτσι ώστε να έχει δέκα (10) κρυφά επίπεδα και θα είναι ενεργό για πενήντα (50) γενιές. Έτσι αφού η διαδικασία έχει τελειώσει μπορούν να γίνουν οι παρακάτω παρατηρήσεις από το αρχείο αναφοράς των δεδομένων.

****** Running generation 0 ******

Population's average fitness: 48.69086 stdev: 44.85911

Best fitness: 168.51891 - size: (14, 110) - species 21 - id 21

Average adjusted fitness: 0.199

Mean genetic distance 3.260, standard deviation 0.420

Population of 150 members in 50 species:

ID	age	size	fitness	adj fit	stag
1	0	3	19.9	0.006	0
2	0	3	22.1	0.021	0
3	0	3	146.6	0.853	0
4	0	3	22.8	0.025	0
5	0	3	146.6	0.853	0
6	0	3	22.1	0.021	0
7	0	3	69.6	0.338	0
8	0	3	69.6	0.338	0
9	0	3	19.0	0.000	0
10	0	3	19.9	0.006	0
11	0	3	19.9	0.006	0
12	0	3	22.1	0.021	0
13	0	3	161.9	0.956	0
14	0	3	23.2	0.028	0
15	0	3	157.2	0.924	0
16	0	3	25.3	0.042	0
17	0	3	34.9	0.106	0
18	0	3	59.9	0.273	0
19	0	3	19.9	0.006	0

20	0	3	46.5	0.184	0
21	0	3	168.5	1.000	0
22	0	3	143.5	0.832	0
23	0	3	150.1	0.877	0
24	0	3	25.2	0.041	0
25	0	3	22.1	0.021	0
26	0	3	19.9	0.006	0
27	0	3	22.1	0.021	0
28	0	3	25.3	0.042	0
29	0	3	19.9	0.006	0
30	0	3	25.3	0.042	0
31	0	3	69.3	0.336	0
32	0	3	28.5	0.064	0
33	0	3	22.1	0.021	0
34	0	3	23.2	0.028	0
35	0	3	25.3	0.042	0
36	0	3	33.2	0.095	0
37	0	3	25.3	0.042	0
38	0	3	28.8	0.066	0
39	0	3	19.0	0.000	0
40	0	3	68.4	0.330	0
41	0	3	29.9	0.073	0
42	0	3	53.2	0.229	0
43	0	3	23.0	0.027	0
44	0	3	19.9	0.006	0
45	0	3	59.9	0.273	0

46	0	3	44.9	0.174	0
47	0	3	19.9	0.006	0
48	0	3	34.9	0.106	0
49	0	3	23.2	0.028	0
50	0	3	31.5	0.083	0

Total extinctions: 0

Generation time: 1.538 sec

Μπορεί πλέον άμεσα να παρατηρηθούν πολλές διαφορές στο πως έχει συμπεριφερθεί ο αλγόριθμος. Η βασική διαφορά που έχει είναι ότι ο πληθυσμός των ειδών είναι πλέων πενήντα και το σύνολο του πληθυσμού των αυτοκινήτων είναι εκατόν πενήντα. Ο πληθυσμός των αυτοκινήτων είναι αρκετά μεγάλος, ώστε να υπάρχει ποικιλομορφία στις πιθανές λύσεις. Επίσης μια σημαντική παρατήρηση είναι ότι ο βαθμός καταλληλότητας αυτής της προσομοίωσης είναι αρκετά χαμηλότερος από τον βαθμός καταλληλότητας της πρώτης προσομοίωσης του πρώτου πειράματος. Αυτό συμβαίνει διότι πλέων τα βάρη που υπολογίζονται μέσα στα κρυφά επίπεδα είναι πολύ περισσότερα καθώς και το εύρος κινήσεων είναι πολύ μεγαλύτερο κατά την λήψη αποφάσεων από την τεχνητή νοημοσύνη.

Μετά την πάροδο των γενεών έχουν ληφθεί τα απαραίτητα δεδομένα από το αρχείο αναφοράς και θα γίνει η σχετική ανάλυση και σύγκριση της πρώτης προσομοίωσης με της τελευταίας.

****** Running generation 49 ******

Population's average fitness: 1021.86058 stdev: 1283.05823

Best fitness: 3035.14701 - size: (13, 72) - species 50 - id 302

Average adjusted fitness: 0.181

Mean genetic distance 2.753, standard deviation 1.024

Population of 50 members in 5 species:

ID	age	size	fitness	adj fit	stag
====	===	====	=======	=======	====

3 49 4 230.7 0.064 48

30 49 22 3035.1 0.427 37

31 49 3 231.0 0.034 48

44 49 3 194.2 0.020 48

50 49 18 3035.1 0.360 27

Total extinctions: 0

Generation time: 15.461 sec (15.456 average)

Στην τελευταία προσομοίωση, τα είδοι και ο πληθυσμός τους έχουν αλλάξει κατά πολύ με την πάροδο των γενεών. Ο αλγόριθμος έχει αφαιρέσει τα είδοι που είχαν παραμείνει στάσιμα για πολλές γενιές και έχει κρατήσει πέντε από αυτά καθώς ο ελιτισμός στο αρχείο διαμόρφωσης έχει χαρακτηριστεί με την τιμή πέντε, δηλαδή ότι πρέπει να κρατήσει οπωσδήποτε πέντε είδοι. Η δεύτερη πιο σημαντική παρατήρηση που μπορεί να γίνει, είναι ότι ο καλύτερος βαθμός καταλληλότητας στο δεύτερο πείραμα είναι μεγαλύτερος από αυτόν του πρώτου πειράματος. Αυτό συμβαίνει διότι λόγο των κρυφών επιπέδων, τα αυτοκίνητα έχουν εκπαιδευτεί να είναι αρκετά αποδοτικότερα από ότι αυτά του πρώτου πειράματος το οποίο δεν είχε κρυφά επίπεδα.

4.4 Ανάλυση τρίτου πειράματος

Το τρίτο πείραμα έχει παραμετροποιηθεί έτσι ώστε να έχει εκατό (100) κρυφά επίπεδα και θα είναι ενεργό για πενήντα (50) γενιές. Έτσι αφού η διαδικασία έχει τελειώσει μπορούν να γίνουν οι παρακάτω παρατηρήσεις.

****** Running generation 0 ******

Population's average fitness: 54.17532 stdev: 43.66041

Best fitness: 184.94405 - size: (104, 920) - species 48 - id 48

Average adjusted fitness: 0.227

Mean genetic distance 3.844, standard deviation 0.499

Population of 150 members in 50 species:

ID age size fitness adj fit stag

==== === ==== ======= ======= ====

1 0 3 137.9 0.722 0

2	0	3	38.0	0.131	0
3	0	3	28.5	0.075	0
4	0	3	19.9	0.024	0
5	0	3	25.3	0.056	0
6	0	3	18.9	0.018	0
7	0	3	25.3	0.056	0
8	0	3	25.3	0.056	0
9	0	3	15.8	0.000	0
10	0	3	15.8	0.000	0
11	0	3	155.7	0.827	0
12	0	3	18.9	0.018	0
13	0	3	28.5	0.075	0
14	0	3	15.8	0.000	0
15	0	3	34.7	0.112	0
16	0	3	104.9	0.527	0
17	0	3	19.0	0.019	0
18	0	3	76.3	0.358	0
19	0	3	25.3	0.056	0
20	0	3	51.8	0.212	0
21	0	3	40.5	0.146	0
22	0	3	43.5	0.164	0
23	0	3	98.7	0.490	0
24	0	3	23.1	0.043	0
25	0	3	135.6	0.708	0
26	0	3	22.1	0.037	0
27	0	3	19.9	0.024	0

28	0	3	125.0	0.646	0
29	0	3	31.5	0.093	0
30	0	3	79.5	0.377	0
31	0	3	44.1	0.167	0
32	0	3	18.9	0.018	0
33	0	3	35.2	0.115	0
34	0	3	20.1	0.025	0
35	0	3	18.9	0.018	0
36	0	3	65.0	0.291	0
37	0	3	23.2	0.044	0
38	0	3	25.8	0.059	0
39	0	3	131.8	0.686	0
40	0	3	53.4	0.222	0
41	0	3	118.7	0.608	0
42	0	3	104.9	0.527	0
43	0	3	33.2	0.102	0
44	0	3	47.6	0.188	0
45	0	3	47.6	0.188	0
46	0	3	36.6	0.123	0
47	0	3	50.6	0.205	0
48	0	3	184.9	1.000	0
49	0	3	25.3	0.056	0
50	0	3	121.6	0.626	0

Total extinctions: 0

Generation time: 4.193 sec

Η πρώτη μεγάλη παρατήρηση που θα γίνει, είναι η οπτική παρατήρηση του ότι πλέων η εργασία εκτελείται πολύ πιο αργά. Ο λόγος που συμβαίνει αυτό είναι οι διαθέσιμοι πόροι του υπολογιστικού συστήματος. Από άλλες πλευρές η συμπεριφορά των αυτοκινήτων δεν έχει αλλάξει κατα πολύ σε σύγκριση με του δεύτερου πειράματος.

Έτσι μετά από αρκετές προσομοιώσεις, θα αναλυθούν τα παρακάτω δεδομένα από το αρχείο αναφοράς.

****** Running generation 49 ******

Population's average fitness: 797.33309 stdev: 659.15910

Best fitness: 1614.31783 - size: (103, 656) - species 36 - id 911

Average adjusted fitness: 0.429

Mean genetic distance 3.280, standard deviation 1.255

Population of 51 members in 5 species:

ID	age	size	fitness	adj fit	stag
8	49	12	1429.9	0.492	15
11	49	3	155.7	0.085	49
36	49	12	1614.3	0.495	37
37	49	12	1429.9	0.552	15
48	49	12	1429.9	0.524	7

Total extinctions: 0

Generation time: 16.066 sec (16.065 average)

Θα παρατηρηθεί η ίδια συμπεριφορά στην αφαίρεση των στάσιμων ειδών, έτσι ώστε να παραμείνουν πέντε από αυτά. Επίσης μπορεί να διακριθεί άνοδος στον καλύτερο βαθμό καταλληλότητας. Όμως, σε σύγκριση με το δεύτερο πείραμα αυτή η βαθμολογία είναι χαμηλότερη. Αυτό δεν σημαίνει ότι το πείραμα με τα δέκα κρυφά επίπεδα υπερτερεί σε σύγκριση με αυτό των εκατό. Πιο συγκεκριμένα σημαίνει ότι, δεν έχουν περάσει αρκετές γενιές προσομοιώσεων έτσι ώστε τα αυτοκίνητα του τρίτου πειράματος να είναι αποδοτικότερα.

Συμπεράσματα

Σύμφωνα με τα παραπάνω πειράματα, μπορεί πλέων κάποιος να αποκομίσει μερικά συμπεράσματα, συγκρίνοντας τα αντίστοιχα αποτελέσματα τους.

Με μηδενικά κρυφά επίπεδα, το νευρωνικό σας δίκτυο ουσιαστικά λειτουργεί ως απλό perceptron. Αυτό μπορεί να οδηγήσει σε περιορισμένη ικανότητα εκμάθησης σύνθετων συμπεριφορών, ειδικά για μια εργασία τόσο περίπλοκη όπως η αυτοοδήγηση. Το αυτοκίνητο μπορεί να δυσκολευτεί να πλοηγηθεί σε πιο σύνθετα περιβάλλοντα, οδηγώντας ενδεχομένως σε χαμηλότερες επιδόσεις και λιγότερες επιτυχημένες δοκιμές.

Η εισαγωγή δέκα κρυφών επιπέδων προσθέτει πολυπλοκότητα στο νευρωνικό δίκτυο. Επιτρέπει την εκμάθηση πιο πολύπλοκων μοτίβων κατά τη διάρκεια της εκπαίδευσης. Το αυτοκίνητο μπορεί να παρουσιάσει βελτιωμένη απόδοση σε σύγκριση με το πείραμα μηδενικών κρυφών στρωμάτων, καθώς μπορεί πλέον να καταγράψει πιο σύνθετα χαρακτηριστικά του περιβάλλοντος και να λειτουργήσει υπολογίζοντας τη βαρύτητα των αποφάσεων που λαμβάνει.

Στην περίπτωση των εκατό κρυφών επιπέδων αυξάνει σημαντικά την πολυπλοκότητα του νευρωνικού δικτύου, καθώς αυτό μπορεί ενδεχομένως να καταγράψει ακόμη πιο περίπλοκα μοτίβα στα δεδομένα που λαμβάνει από το περιβάλλον. Ωστόσο, η εκπαίδευση ενός νευρωνικού δικτύου με τόσο μεγάλο αριθμό κρυφών επιπέδων μπορεί να είναι προκλητική. Μπορεί να απαιτήσει μεγαλύτερους χρόνους εκπαίδευσης και προσεκτική ρύθμιση και παραμετροποίηση για την αποφυγή προβλημάτων όπως η εξαφάνιση λόγω στασιμότητας ή ο κίνδυνος υπερβολικής προσαρμογής λόγω της απόλυτης πολυπλοκότητας του μοντέλου, ειδικά εάν το σύνολο δεδομένων εκπαίδευσης δεν είναι αρκετά διαφορετικό.

Μελλοντική Επέκταση

Στον χώρο της ανάπτυξης λογισμικών, τίποτα δεν είναι τέλειο αλλά ούτε παραμένει στάσιμο. Για αυτό τον λόγο, θα διατυπωθούν μερικές ιδέες επέκτασης τη συγκεκριμένης εργασίας.

Στο συγκεκριμένο λογισμικό που δημιουργήθηκε έχει παρατηρηθεί ότι το αυτοκίνητο δεν μπορεί σχεδόν ποτέ να κινηθεί σε μια ευθεία γραμμή, ακόμα και όταν ο δρόμος της πίστας είναι ευθύς. Για να μπορέσει να συμβεί αυτό, θα πρέπει να σχεδιαστεί και να αναπτυχθεί η λογική βέλτιστης πορείας.

Επίσης η δειγματοληψία των δεδομένων γίνεται από τα πέντε ραντάρ. Αυτό θα μπορούσε να βελτιστοποιηθεί έτσι ώστε να προσομοιωθεί η λειτουργικότητα ενός Lidar Sensor. Ένας τέτοιος αισθητήρας έχει την δυνατότητα να χαρτογραφεί το περιβάλλον στο οποίο βρίσκεται, μέσω μιας κάμερας. Για να μπορέσει να προσομοιωθεί αυτό ως λειτουργία στην παρούσα εργασία, θα πρέπει να αυξηθεί ο αριθμός των ραντάρ. Πιο συγκεκριμένα θα πρέπει ο αριθμός τους να είναι αυτός που θα συλλέγει δεδομένα από σε όλο το εμπρόσθιο εύρος των εκατόν ογδόντα μοιρών του αυτοκινήτου.

Βιβλιογραφία

[1] P. NEAT, «NEAT Overview,» [Ηλεκτρονικό]. Available: https://neat-python.readthedocs.io/en/latest/neat_overview.html. [Πρόσβαση 10 March 2024].

[2] Pygame, «pygame.transform.scale(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/transform.html#pygame.transform.scale. [Πρόσβαση 10 March 2024].

[3] Pygame, «pygame.image.load(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/image.html#pygame.image.load. [Πρόσβαση 10 March 2024].

[4] Pygame, «pygame.draw.line(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/draw.html#pygame.draw.line. [Πρόσβαση 10 March 2024].

[5] Pygame, « pygame.draw.circle(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/draw.html#pygame.draw.circle. [Πρόσβαση 10 March 2024].

[6] Pygame, «blit(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/surface.html?highlight=blit#pygame.Surface.blit. [Πρόσβαση 10 March 2024].

[7] Pygame, «get_at(),» [Ηλεκτρονικό]. [Πρόσβαση 10 March 2024].

[8] Pygame, «get_rect(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/surface.html?highlight=get_rect#pygame.Surface.get_rect. [Πρόσβαση 10 March 2024].

[9] Pygame, «pygame.transform.rotate(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/transform.html#pygame.transform.rotate. [Πρόσβαση 10 March 2024].

[10] P. NEAT, «Configuration file description,» [Ηλεκτρονικό]. Available: https://neat-python.readthedocs.io/en/latest/config_file.html. [Πρόσβαση 10 March 2024].

[11] P. NEAT, «nn.feed_forward,» [Ηλεκτρονικό]. Available: https://neat-python.readthedocs.io/en/latest/module_summaries.html#nn-feed-forward.

[Πρόσβαση 10 March 2024].

[12] Pygame, «pygame.display.set_caption(),» [Ηλεκτρονικό]. [Πρόσβαση 10 March 2024].

[13] Pygame, «pygame.display.set_mode(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/display.html#pygame.display.set_mode. [Πρόσβαση 10 March 2024].

[14] Pygame, «pygame.time.Clock,» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/time.html#pygame.time.Clock. [Πρόσβαση 10 March 2024].

[15] Pygame, «pygame.mouse.get_pressed(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/mouse.html#pygame.mouse.get_pressed. [Πρόσβαση 10 March 2024].

[16] Pygame, «pygame.mouse.get_pos(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/mouse.html#pygame.mouse.get_pos. [Πρόσβαση 10 March 2024].

[17] Pygame, «pygame.display.update(),» [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/ref/display.html#pygame.display.update. [Πρόσβαση 10 March 2024].

[18] Pygame, «pygame.event.get(),» [Ηλεκτρονικό]. [Πρόσβαση 10 March 2024].

[19] P. NEAT, «Cechkpoint,» [Ηλεκτρονικό]. Available: https://neat-python.readthedocs.io/en/latest/module_summaries.html?highlight=checkpoint#checkpoint. [Πρόσβαση 10 March 2024].

[20] P. NEAT, «Reporting/logging,» [Ηλεκτρονικό]. Available: https://neat-python.readthedocs.io/en/latest/customization.html#reporting-logging. [Πρόσβαση 10 March 2024].

Παράρτημα Κώδικα

Αρχείο car.py

```
import pygame
import math
from environment.colors import Color

CAR_SPRITE_PATH = "assets/car.png"
DEAD_CAR_SPRITE_PATH = "assets/dead_car.png"


class Car:

    CAR_SIZE_X = 60
    CAR_SIZE_Y = 60
    DEFAULT_SPEED = 10
    MINIMUM_SPEED = 10
    SPEED_INCREMENT = 1
    ANGLE_INCREMENT = 10
    FULL_CIRCLE_DEGREES = 360
    RADAR_DRAW_DISTANCE = 1000
    RADAR_START_ANGLE = -90
    RADAR_END_ANGLE = 90
    RADAR_STEP_ANGLE = 45
    RADAR_LINE = 2
    RADAR_CIRCLE = 4

    def __init__(self, start_position: list):
        self._sprite = pygame.transform.scale(
            pygame.image.load(CAR_SPRITE_PATH).convert_alpha(),
            (self.CAR_SIZE_X, self.CAR_SIZE_Y)
        )
        self.sprite = self._sprite
        self.position = start_position.copy()
        self.facing_angle = 0
        self.speed = self.DEFAULT_SPEED
        self.car_center = [
            self.position[0] + self.CAR_SIZE_X / 2,
            self.position[1] + self.CAR_SIZE_Y / 2
        ]
        self.radars = []
        self.alive = True
        self.distance_driven = 0
        self.speed_penalty = 0
```

```
    def draw(self, environment: pygame.Surface) -> None:
        if self.alive:
            environment.blit(self.sprite, self.position)
            self.draw_radars(environment)

    def draw_radars(self, environment: pygame.Surface) -> None:
        for radar in self.radars:
            radar_position = radar[0]
            pygame.draw.line(environment, Color.GREEN,
                             self.car_center, radar_position,
self.RADAR_LINE)
            pygame.draw.circle(
                environment, Color.RED, radar_position,
self.RADAR_CIRCLE)

    def out_of_bounds(self, point_x: int, point_y: int, width:
int, height: int) -> bool:
        return point_x < 0 or point_x >= width or point_y < 0 or
point_y >= height

    def distance_to_border(self, point_x: int, point_y: int) ->
int:
        distance = int(math.hypot(
            point_x - self.car_center[0], point_y -
self.car_center[1]))
        self.radars.append([(point_x, point_y), distance])

    def check_collision(self, environment: pygame.Surface) ->
bool:
        environment_width, environment_height =
environment.get_size()
        for point in self.corners:
            point_x, point_y = point
            if self.out_of_bounds(point_x, point_y,
environment_width, environment_height) or \
                environment.get_at((int(point_x), int(point_y)))
== Color.WHITE:
                self.alive = False
                return True
        return False
```

```
    def check_radar(self, degree: int, environment:
pygame.Surface) -> None:
        radians = math.radians(self.FULL_CIRCLE_DEGREES -
                               (self.facing_angle + degree))
        length = 1
        environment_width, environment_height =
environment.get_size()

        while True:
            length += 1
            point_x = int(self.car_center[0] + math.cos(radians)
* length)
            point_y = int(self.car_center[1] + math.sin(radians)
* length)

            if self.out_of_bounds(point_x, point_y,
environment_width, environment_height):
                break
            if environment.get_at((point_x, point_y)) ==
Color.WHITE:
                break
            if length > Car.RADAR_DRAW_DISTANCE:
                break

        self.distance_to_border(point_x, point_y)

    def calculate_corner(self, angle, car_x, car_y) -> list[int]:
        corner = math.radians(self.FULL_CIRCLE_DEGREES -
                              (self.facing_angle + angle))
        return [
            self.car_center[0] + math.cos(corner) * car_x,
            self.car_center[1] + math.sin(corner) * car_y
        ]

    def refresh_corners_positions(self) -> None:
        car_x = Car.CAR_SIZE_X / 2
        car_y = Car.CAR_SIZE_Y / 2
        angles = [30, 150, 210, 330]

        self.corners = [self.calculate_corner(
            angle, car_x, car_y) for angle in angles]
```

```
    def update_car_center(self) -> None:
        sprite_as_rect = self._sprite.get_rect()
        rotated_sprite = pygame.transform.rotate(
            self._sprite, self.facing_angle)
        sprite_as_rect.center = rotated_sprite.get_rect().center
        self.sprite = rotated_sprite.subsurface(sprite_as_rect)
        self.car_center = [
            int(self.position[0]) + Car.CAR_SIZE_X / 2,
            int(self.position[1]) + Car.CAR_SIZE_Y / 2
        ]

    def update_car_state(self, environment: pygame.Surface) ->
None:
        self.update_car_center()
        radians = math.radians(self.FULL_CIRCLE_DEGREES -
self.facing_angle)
        self.position[0] += math.cos(radians) * self.speed
        self.position[1] += math.sin(radians) * self.speed
        self.distance_driven += self.speed
        self.refresh_corners_positions()
        self.check_collision(environment)
        self.radars.clear()
        for radar_angle in range(self.RADAR_START_ANGLE,
self.RADAR_END_ANGLE + 1, self.RADAR_STEP_ANGLE):
            self.check_radar(radar_angle, environment)

    def get_data(self) -> list[int]:
        distances = [int(radar[1]) for radar in self.radars]
        distances += [0] * (5 - len(distances))
        return distances

    def get_reward(self) -> float:
        distance_reward = self.distance_driven / 10000
        speed_reward = self.speed ** 0.5
        penalty = self.speed_penalty / 100
        final_reward = distance_reward + speed_reward - penalty
        return final_reward

    def accelerate(self) -> None:
        self.speed += Car.SPEED_INCREMENT

    def brake(self) -> None:
```

```
        if self.speed > Car.MINIMUM_SPEED:
            self.speed -= Car.SPEED_INCREMENT
        else:
            self.speed = Car.MINIMUM_SPEED
            self.speed_penalty += 1

    def turn_left(self) -> None:
        self.facing_angle += Car.ANGLE_INCREMENT

    def turn_right(self) -> None:
        self.facing_angle -= Car.ANGLE_INCREMENT
```

Αρχείο car_ai.py

```
import neat
from car.car import Car
from car.actions import Actions
import pygame


class CarAI:

    TOTAL_GENERATIONS = 0
    TIME_TO_LIVE = 15

    def __init__(self, genomes: neat.DefaultGenome, config:
neat.Config, start_position: list):
        CarAI.TOTAL_GENERATIONS += 1
        self.genomes = genomes
        self.nets = [neat.nn.FeedForwardNetwork.create(
            genome, config) for _, genome in genomes]
        self.cars = [Car(start_position) for _ in genomes]
        self.best_fitness = 0

        for _, genome in genomes:
            genome.fitness = 0

        self.remaining_cars = len(self.cars)

    def decisions(self, environment: pygame.Surface) -> None:
        for i, (genome_id, genome) in enumerate(self.genomes):
```

```
            car = self.cars[i]
            net = self.nets[i]
            car_data = car.get_data()
            output = net.activate(car_data)
            choice = output.index(max(output))

            match choice:
                case Actions.TURN_LEFT:
                    car.turn_left()
                case Actions.TURN_RIGHT:
                    car.turn_right()
                case Actions.ACCELERATE:
                    car.accelerate()
                case Actions.BRAKE:
                    car.brake()

            if car.alive:
                car.update_car_state(environment)
                genome.fitness += car.get_reward()

        self.remaining_cars = len([car for car in self.cars if
car.alive])
        self.best_fitness = max(
            self.genomes, key=lambda genome:
genome[1].fitness)[1].fitness
```

Αρχείο actions.py

```
class Actions:
    TURN_LEFT = 0
    TURN_RIGHT = 1
    ACCELERATE = 2
    BRAKE = 3
```

Αρχείο config.txt

```
[NEAT]

fitness_criterion     = max

fitness_threshold     = 100000000
```

```
pop_size            = 50
reset_on_extinction   = True

[DefaultGenome]
# node activation options
activation_default      = tanh
activation_mutate_rate  = 0.1
activation_options      = tanh

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.1
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5
```

```
# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.1

feed_forward            = True
initial_connection      = full_direct

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2

# network parameters
num_hidden              = 3
num_inputs              = 5
num_outputs             = 4

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
```

```
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean       = 0.0
weight_init_stdev      = 1.0
weight_max_value       = 30
weight_min_value       = -30
weight_mutate_power    = 0.5
weight_mutate_rate     = 0.8
weight_replace_rate    = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 2.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation      = 10
species_elitism     = 5

[DefaultReproduction]
elitism          = 3
survival_threshold = 0.2
```

Αρχείο environment.py

```
import pygame
```

```
import neat
from car.car_ai import CarAI
from car.car import Car
import time
from environment.colors import Color
import os
import glob


class Environment:

    WIDTH = 1900
    HEIGHT = 1000
    FPS = 60
    BRUSH_SIZE = 50
    MAX_BRUSH_SIZE = 100
    MIN_BRUSH_SIZE = 10
    SCROLL_UP_BUTTON = 4
    SCROLL_DOWN_BUTTON = 5

    def __init__(self, NEAT_CONFIG, MAX_SIMULATIONS):
        self.NEAT_CONFIG = NEAT_CONFIG
        self.MAX_SIMULATIONS = MAX_SIMULATIONS
        self.title = "AI_Cars"
        pygame.display.set_caption(self.title)
        self.screen = pygame.display.set_mode(
            (self.WIDTH, self.HEIGHT))
        self.screen.fill(Color.WHITE)
        self.is_running = False
        self.clock = pygame.time.Clock()
        self.drawing_environment = True
        self.placing_start_point = False
        self.ai_initialization = False
        self.track = None
        self.environment = None
        self.car = Car([0, 0])
        self.final_car_pos = None

    def _drawing_environment(self):
        mousebutton_pressed = pygame.mouse.get_pressed()
        mouse_pos = pygame.mouse.get_pos()
        if mousebutton_pressed[0]:
```

```
                pygame.draw.circle(self.screen, Color.BLACK,
                                   mouse_pos, self.BRUSH_SIZE)
            elif mousebutton_pressed[2]:
                pygame.draw.circle(self.screen, Color.WHITE,
                                   mouse_pos, self.BRUSH_SIZE)

    def _placing_start_point(self):
        mouse_x, mouse_y = pygame.mouse.get_pos()
        car_sprite = self.car.sprite
        car_position = self.car.position
        car_center = self.car.car_center

        if not pygame.mouse.get_pressed()[0]:
            car_position[0] = mouse_x - Car.CAR_SIZE_X / 2
            car_position[1] = mouse_y - Car.CAR_SIZE_Y / 2
            self.screen.blit(car_sprite, car_position)
        else:
            self.ai_initialization = True
            self.placing_start_point = False
            self.environment = self.screen.copy()
            self.screen.blit(car_sprite, car_center)
            self.final_car_pos = list(car_position)

    def initialize_car_state(self):
        pygame.display.update()
        if self.placing_start_point:
            self.screen.blit(self.track, (0, 0))

    def handle_quit_event(self, event):
        if event.type == pygame.QUIT:
            exit()

    def handle_keyup_event(self, event):
        if event.type == pygame.KEYUP and event.key ==
pygame.K_SPACE:
            self.drawing_environment = False
            self.placing_start_point = True
            self.track = self.screen.copy()

    def handle_mousebuttondown_event(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN:
```

```
            if event.button == self.SCROLL_UP_BUTTON and
self.BRUSH_SIZE < self.MAX_BRUSH_SIZE:
                self.BRUSH_SIZE += 1
            elif event.button == self.SCROLL_DOWN_BUTTON and
self.BRUSH_SIZE > self.MIN_BRUSH_SIZE:
                self.BRUSH_SIZE -= 1

    def initialize_environment(self):
        if self.drawing_environment:
            self._drawing_environment()
        if self.placing_start_point:
            self._placing_start_point()
        if self.ai_initialization:
            self.initiate_ai()
        self.initialize_car_state()

    def run(self) -> None:
        self.running = True
        event_handlers = {
            pygame.QUIT: self.handle_quit_event,
            pygame.KEYUP: self.handle_keyup_event,
            pygame.MOUSEBUTTONDOWN:
self.handle_mousebuttondown_event,
        }
        while self.running:
            for event in pygame.event.get():
                handler = event_handlers.get(event.type)
                if handler:
                    handler(event)
            self.initialize_environment()

    def initiate_ai(self):
        config = neat.config.Config(
            neat.DefaultGenome,
            neat.DefaultReproduction,
            neat.DefaultSpeciesSet,
            neat.DefaultStagnation,
            self.NEAT_CONFIG
        )

        try:
            checkpoint_files = glob.glob('NN/save-*')
```

```
            checkpoint_files.sort(key=os.path.getmtime)
            latest_checkpoint = checkpoint_files[-1]
            population = neat.Checkpointer.restore_checkpoint(
                latest_checkpoint)
        except Exception:
            population = neat.Population(config)

        population.add_reporter(neat.StdOutReporter(True))
        population.add_reporter(neat.StatisticsReporter())

        population.add_reporter(neat.Checkpointer(
            generation_interval=10, filename_prefix="NN/save-"))
        population.run(self.run_simulation, self.MAX_SIMULATIONS)

    def run_simulation(self, genomes: neat.DefaultGenome, config:
neat.Config) -> None:
        car_ai = CarAI(genomes, config, self.final_car_pos)
        timer = time.time()
        self.is_running = True
        while self.is_running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    exit()

            car_ai.decisions(self.environment)
            if car_ai.remaining_cars == 0:
                break

            time_remaining = time.time() - timer
            if time_remaining > CarAI.TIME_TO_LIVE:
                break

            self.screen.blit(self.environment, (0, 0))
            for car in car_ai.cars:
                car.draw(self.screen)

            generation_text = "Generation: " +
str(car_ai.TOTAL_GENERATIONS)
            stil_alibe_text = "Still Alive: " +
str(car_ai.remaining_cars)
            time_remaining_text = "Time Left: " + \
```

```
                str(round(CarAI.TIME_TO_LIVE - time_remaining,
2)) + "s"
            best_fitnes_text = "Best Fitness: " + \
                str(round(car_ai.best_fitness))
            pygame.display.set_caption(self.title + " - " +
generation_text + " - " +
                                       stil_alibe_text + " - " +
time_remaining_text + " - " + best_fitnes_text)

            self.initialize_car_state()
            self.clock.tick(Environment.FPS)
```

Αρχείο colors.py

```
class Color:
    WHITE = (255, 255, 255)
    BLACK = (0, 0, 0)
    GREEN = (0, 255, 0)
    RED = (255, 0, 0)
    BLUE = (0, 0, 255)
```