Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Εφαρμοσμένα Νευρωνικά Δίκτυα σε Ηλεκτρονικά Παίγνια: Μια μελέτη του εργαλείου ML – Agents στο περιβάλλον Unity** **Applied Neural Networks in Video Games: A case study of ML-Agents in Unity**
Ονοματεπώνυμο Φοιτητή	**Ματσιόζεκ Ιζαμπέλ - Βικτωρία**
Πατρώνυμο	**Κλάους Πέτερ**
Αριθμός Μητρώου	**ΜΠΣΠ/ 16018**
Επιβλέπων	**Θεμιστοκλής Παναγιωτόπουλος, Καθηγητής**

Ημερομηνία Παράδοσης **Δεκέμβριος 2018**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Θεμιστοκλής Παναγιωτόπουλος Καθηγητής	Δημήτρης Αποστόλου Καθηγητής	Άγγελος Πικράκης Καθηγητής

Περίληψη

Η μηχανική μάθηση είναι ένας από τους ταχύτερα αναπτυσσόμενος κλάδους της τεχνητής νοημοσύνης, που αποτελεί με τη σειρά της μέρος της επιστήμης των υπολογιστών. Ασχολείται με την μάθηση προτύπων από δεδομένα και την μελέτη αλγορίθμων που βελτιώνουν τη συμπεριφορά του υπολογιστή σε κάποια εργασία που του έχει ανατεθεί χρησιμοποιώντας την εμπειρία του, με απώτερο στόχο να προσφέρει μια μαθησιακή λύση ή απάντηση.

Το αντικείμενο της παρούσας διπλωματικής εργασίας είναι σύνθετο και έχει ως σκοπό να παρουσιάσει ένα μέρος αυτού του νέου ερευνητικού πεδίου, μέσα από τις εφαρμογές που αυτό βρίσκει στον χώρο των παιγνίων.

Αρχικά θα πρέπει να κατανοηθούν οι έννοιες και η ορολογία που διέπουν τον τομέα της μηχανικής μάθησης και των παιγνίων μέσα από μία βιβλιογραφική ανασκόπηση στη θεωρία τους (Κεφάλαια 1 & 2).

Στη συνέχεια παρουσιάζεται το εργαλείο ML-Agents της Unity, πλατφόρμα η οποία θα χρησιμοποιηθεί σαν το σημείο αναφοράς της παρούσας εργασίας (Κεφάλαιο 3), καθώς και ο τρόπος εγκατάστασης και χρήσης τους (Κεφάλαιο 4) και στη συνέχεια αναλύεται ένα παράδειγμα χρήσης των εργαλείων για τη δημιουργία ενός παιγνίου με πολλαπλούς πράκτορες (Κεφάλαιο 5).

Τέλος, γίνεται ανάλυση των συμπερασμάτων που προκύπτουν από το παράδειγμα και προτείνονται μελλοντικές βελτιώσεις (Κεφάλαιο 6).

Abstract

Machine learning is one of the most rapidly growing branches of artificial intelligence, which in turn is part of computer science. Machine learning deals with the extraction of models and algorithms from data, in order to improve the computer's behaviour in some task based on experience, with the ultimate goal of providing a learning solution or answer.

The subject of the current dissertation is complex and aims to present a part of this new scientific field through its applications in the field of video games.

Initially, the concepts and terminology governing the mechanics of machine learning and video games should be understood through a bibliographic review of their theory (Chapters 1 & 2).

Following, we introduce the ML-Agents toolkit for Unity, a platform that will be used as the reference point of this work (Chapter 3), present installation and usage instructions (Chapter 4), and then analyze a usage example of the toolkit for the creation of a multi-agent game (Chapter 5).

Finally, we analyze the conclusions drawn from the implemented example and propose future improvements (Chapter 6).

Περιεχόμενα

1. Εισαγωγή

Καθημερινά η εμφάνιση νέων και αναδυόμενων τεχνολογιών μας προκαλούν να αναθεωρήσουμε την αντίληψή μας για την μάθηση και την ευφυΐα. Από τα αυτόνομα αυτοκίνητα, υπολογιστές που παίζουν το παιχνίδι Go και σκάκι και είναι ικανοί να νικήσουν και τους ανθρώπινους αντιπάλους του, στους υπολογιστές που μπορούν να προσομοιώσουν ανθρώπινη συμπεριφορά και νοημοσύνη. Αυτή η έλευση νέων τεχνολογιών, που αποκαλούμε Μηχανική Μάθηση, κυριαρχεί στη σημερινή εποχή της ανάπτυξης τεχνολογιών. Μια νέα εποχή τεχνολογικής ανάπτυξης που έχει συγκριθεί ότι κατέχει την ίδια σημασία με την εφεύρεση της ηλεκτρικής ενέργειας και έχει ήδη κατηγοριοποιηθεί ως η επόμενη τεχνολογία που θα αλλάξει τη ζωή των ανθρώπων. Παρόλο που η μηχανική μάθηση είναι ένα πολυσυζητημένο θέμα, μπορεί να πάρει χρόνια έως ότου κυριαρχήσει στο τεχνολογικό παρασκήνιο.

1.1. ΣΚΟΠΌΣ ΚΑΙ ΣΤΌΧΟΙ ΤΗΣ ΕΡΓΑΣΊΑΣ

Αυτή η διπλωματική έχει ως σκοπό να παρουσιάσει ένα μέρος αυτού του νέου ερευνητικού πεδίου χρησιμοποιώντας την πλατφόρμα Machine Learning Agents που ονομάζεται ML-Agents από την Unity.

2. Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό σκιαγραφείται η επιστημονική περιοχή της θεωρίας παιχνιδιών και ιδιαίτερα αυτή της τεχνητής νοημοσύνης στα παιχνίδια και της μηχανικής μάθησης. Στο πλαίσιο αυτό, παρουσιάζονται αλγόριθμοι που υποστηρίζουν την ενισχυτική μάθηση καθώς και διάφορα άλλα μοντέλα εκμάθησης.

2.1. ΘΕΩΡΊΑ ΠΑΙΓΝΊΩΝ

Η θεωρία παιγνίων αποτελεί εδώ και δεκαετίες ένα τομέα με εξαιρετικό ενδιαφέρον για μελέτη και έρευνα. Οι επιστήμονες εστιάζουν σε παιχνίδια στρατηγικής και προσπαθούν με χρήση Τεχνητής Νοημοσύνης (Artificial Intelligence) να δημιουργήσουν ‘’έξυπνα’’ προγράμματα που θα συναγωνίζονται ως προς την απόδοσή τους πραγματικούς παίκτες. Τέτοιου είδους παιχνίδια ενδείκνυνται για μελέτη λόγω της πολυπλοκότητάς τους και της έξυπνης στρατηγικής που χρειάζεται να ακολουθήσει κάποιος για να κερδίσει. Επιπλέον, οι είσοδοι του παιχνιδιού και τα κριτήρια αξιολόγησης είναι γνωστά, ενώ το περιβάλλον του παιχνιδιού, οι έγκυρες κινήσεις και οι κινήσεις τερματισμού μπορούν εύκολα να προσομοιωθούν.

Το 1995 ο Samuel [Samuel 1959] έφτιαξε ένα πρόγραμμα ντάμας (checkers play), ενώ από το ’60 ξεκίνησε η δημιουργία προγραμμάτων σκακιού. Στη δεκαετία του ’90, η IBM, πρώτα με το Deep Thought και στη συνέχεια με το Deep Blue, κατέβαλε σφοδρές προσπάθειες για τη δημιουργία ενός προγράμματος σκακιού ικανού να ανταγωνίζεται και τους καλύτερους σκακιστές του κόσμου. Ένα από τα πιο σημαντικά παιχνίδια στις μέρες μας είναι το TD-Gammon του Tesauro [Tesauro 1992], [Tesauro 1995] για το τάβλι. Το σημαντικότερο και το πιο κρίσιμο σημείο ενός στρατηγικού παιχνιδιού είναι η επιλογή και η υλοποίηση της στρατηγικής που θα ακολουθήσει ο υπολογιστής κατά τη διάρκεια του παιχνιδιού. Με τον όρο στρατηγική εννοούμε την επιλογή της επόμενης κίνησής του υπολογιστή λαμβάνοντας υπόψη την παρούσα κατάστασή του, την κατάσταση του αντιπάλου, τις επιπτώσεις της κίνησής του και την ενδεχόμενη επόμενη κίνηση του αντιπάλου.

2.2. ΤΕΧΝΗΤΉ ΝΟΗΜΟΣΎΝΗ

Η τεχνητή νοημοσύνη (Artificial Intelligence - AI) έχει σχετικά μεγάλη ιστορία και αναφορές σε εκείνη μπορεί κάποιος να βρει μέχρι και στην αρχαιότητα. Η ιστορία της ξεκίνησε με μύθους και φήμες για τεχνητά όντα προικισμένα με νοημοσύνη ή και συνείδηση. Με άλλα λόγια η τεχνητή νοημοσύνη δημιουργήθηκε για να εκπληρώσει την επιθυμία των ανθρώπων να σφυρηλατήσουν

τους θεούς (McCorduck, 2004). Η τεχνητή νοημοσύνη όπως είναι γνωστή στην πληροφορική τυπικά ξεκίνησε το 1956 σε μια συνάντηση μερικών διακεκριμένων επιστημόνων όπως ο John McCarthy, ο Claude Shannon, κ.α.

Ο ορισμός την τεχνητής νοημοσύνης είναι αρκετά πολύπλοκος χωρίς σαφώς καθορισμένη εφαρμογή. Αυτό έγκειται στο ότι η ίδια η λέξη νοημοσύνη έχει ποικίλους ορισμούς και ότι με τα χρόνια οι μηχανές γίνονται ολοένα και πιο ικανές και κατά συνέπεια τα καθήκοντα και οι δυνατότητες τους που κάποτε θεωρούνταν ευφυΐα τώρα χαρακτηρίζονται ως τεχνολογίες ρουτίνας (Schank, 1991).

Ο Douglas Hofstadter, προτείνει ότι νοημοσύνη είναι:

- Να ανταποκρίνεσαι σε καταστάσεις με ελαστικότητα (όχι μηχανική συμπεριφορά)
- Να κατανοείς τα ασαφή ή αντιφατικά μηνύματα από τα συμφραζόμενα
- Να αναγνωρίζεις και να ιεραρχείς τα διάφορα δεδομένα με βάση τη σπουδαιότητα τους
- Να βρίσκεις ομοιότητες μεταξύ καταστάσεων οι οποίες μοιάζουν διαφορετικές
- Να βρίσκεις διαφορές μεταξύ καταστάσεων οι οποίες μοιάζουν παρόμοιες

Στο γενικό ορισμό που θα μπορούσε κανείς να πει ότι συμφωνούν όλοι οι εμπλεκόμενοι τομείς είναι ότι η τεχνητή νοημοσύνη ορίζεται ως μελέτη συστημάτων που δρουν με τρόπο που σε οποιονδήποτε παρατηρητή να φαίνονται έξυπνα. Επίσης, μια τεχνητή νοημοσύνη περιλαμβάνει τεχνικές που χρησιμοποιούνται για την επίλυση απλών προβλημάτων και ως εκ τούτου κάνοντας χρήση μεθόδων που βασίζονται στην ευφυή συμπεριφορά ανθρώπων και άλλων ζώων για την επίλυση πολύπλοκων προβλημάτων. (Coppin, 2004)

Η τεχνητή νοημοσύνη έχει αρκετές περιοχές εφαρμογής όπως για παράδειγμα, στην επίλυση προβλημάτων και απόδειξη θεωρημάτων, στην μηχανική μάθηση, στους ευφυείς πράκτορες (agents), στις ευφυείς υπηρεσίες διαδικτύου και το σημασιολογικό διαδίκτυο (sematic web), κι άλλες.

Η τεχνητή νοημοσύνη μπορεί να χωριστεί σε δυο προσεγγίσεις, την κλασική ή συμβολική (symbolic AI) και την υπολογιστική νοημοσύνη (computational intelligence) ή μη-συμβολική. Η κλασική προσέγγιση βασίζεται στην κατανόηση των νοητικών διεργασιών και ασχολείται με την προσομοίωση της ανθρώπινης νοημοσύνης προσεγγίζοντάς την με αλγορίθμους και συστήματα που βασίζονται στη γνώση χρησιμοποιώντας ως δομικές μονάδες τα σύμβολα (πχ. συστήματα κανόνων). Η δεύτερη βασίζεται στη μίμηση της βιολογικής λειτουργίας του εγκεφάλου όπως η διαδικασία της εξέλιξης των ειδών ή η λειτουργία του εγκεφάλου (π.χ. νευρωνικά δίκτυα, γενετική αλγόριθμοι).

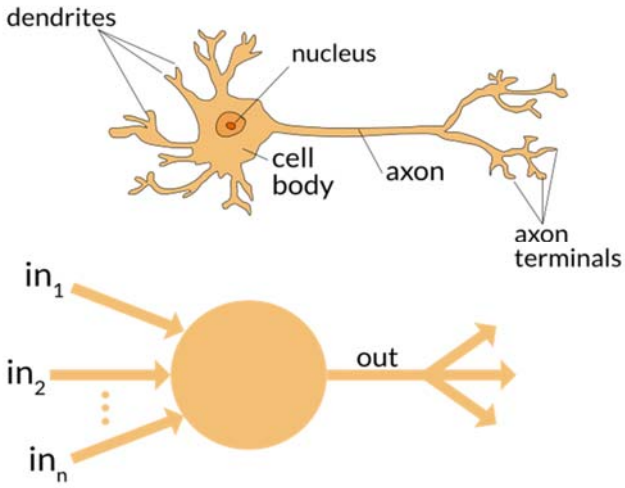
2.2.1. Τεχνητή νοημοσύνη στα παίγνια

Η τεχνητή νοημοσύνη στα παιχνίδια (Game AI) μπορεί να οριστεί ως μια δομή κώδικα που δίνει στις ελεγχόμενες από τον υπολογιστή οντότητες τη δυνατότητα να παίρνουν έξυπνες αποφάσεις. Η τεχνητή νοημοσύνη στα παίγνια μπορεί να χαρακτηριστεί ως συμπεριφορά του πράκτορα, δηλαδή ένα σύνολο ενεργειών χαμηλού επιπέδου όπως η επιλογή συγκεκριμένης προγραμματισμένης από πριν συμπεριφοράς ανάλογα με την δεδομένη κατάσταση του παιχνιδιού. Επιπλέον, οι αλγόριθμοι κίνησης και σύγκρουσης μπορούν επίσης να συμπεριληφθούν στην τεχνητή νοημοσύνη στα παίγνια.

2.3. ΝΕΥΡΩΝΙΚΆ ΔΊΚΤΥΑ

Τα νευρωνικά δίκτυα παρέχουν τα θεμέλια για μερικούς από τους πιο εντυπωσιακούς αλγορίθμους τόσο στην τεχνητή νοημοσύνη όσο και στη μηχανική μάθηση. Αλγόριθμοι που χαρακτηρίζονται ως οι πιο εντυπωσιακοί πού έχουμε δει τα τελευταία χρόνια. Έχουν επίσης γίνει το θεμέλιο ή το πρότυπο για αρκετές περιοχές της τεχνητής νοημοσύνης, από την αναγνώριση εικόνας και ομιλίας έως την αναπαραγωγή παιχνιδιών Atari. Παρόλο που μπορεί να ακούγεται ως κάτι περίπλοκο και τρομακτικό στην πραγματικότητα, ένα νευρωνικό δίκτυο είναι μια αρκετά απλή και κομψή δομή που προσομοιώνει την δομή των νευρώνων του ανθρώπινου εγκεφάλου.

Το θεμέλιο του ανθρώπινου εγκεφάλου και του ανθρώπινου νευρικού συστήματος είναι ένας νευρώνας, ο οποίος παρουσιάζεται στην παρακάτω εικόνα δίπλα σε έναν τεχνικό νευρώνα:



Εικόνα 1 - Νευρώνας

Κάθε νευρώνας αποτελείται από 3 κύρια τμήματα:

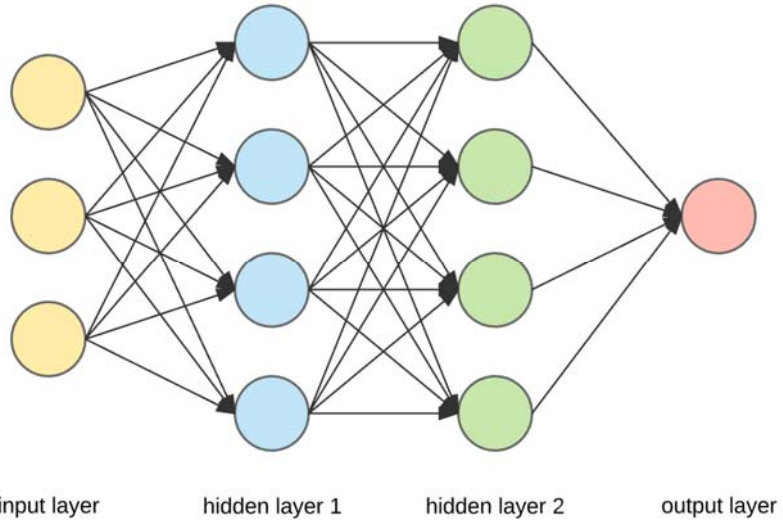
- τους δενδρίτες (dendrites), οι οποίοι λειτουργούν ως κανάλια εισόδου για το νευρώνα
- το κυρίως κυτταρικό σώμα (cell body)
- τον άξονα του κυττάρου-νευροάξονα (axon), που συνδέει ένα νευρώνα με άλλους νευρώνες

Ο άξονας του ενός νευρώνα μεταφέρει σήματα στους δενδρίτες γειτονικών νευρώνων μέσω του σημείου ένωσης που ονομάζεται νευροαξονική απόληξη ή σύναψη (synapse). Ένας νευρώνας μπορεί να λάβει σήματα από ένα σύνολο γειτονικών νευρώνων μέσω των δενδριτών, να τα επεξεργαστεί και να τροφοδοτήσει την έξοδό του μέσω του άξονα προς ένα άλλο σύνολο γειτονικών νευρώνων. Τα σήματα που έρχονται μέσω των δενδριτών “ζυγίζονται” και τα αποτελέσματα αθροίζονται. Όταν το άθροισμα ξεπεράσει το οριακό επίπεδο (τιμή κατωφλίου), ο νευρώνας δημιουργεί μια έξοδο (με τη μορφή νευρικής ώσης ή ηλεκτρικού σήματος) στον άξονά του, η οποία εν συνεχεία μέσω των συνάψεων θα μεταφερθεί στους γειτονικούς νευρώνες.

Τα μαθηματικά μοντέλα των τεχνητών νευρωνικών δικτύων, σε πλήρη αντιστοιχία με τα βιολογικά, αποτελούνται από έναν αριθμό απλών και με υψηλό βαθμό εσωτερικής διασύνδεσης επεξεργαστικών μονάδων, οργανωμένων σε στρώματα.

2.3.1. Ένας νευρώνας

Ο προσομοιωμένος νευρώνας στο προηγούμενο διάγραμμα αντιπροσωπεύει τη δομή ενός μόνο νευρώνα. Οι είσοδοι ή τα σήματα στον νευρώνα συνήθως αθροίζονται και στη συνέχεια αξιολογούνται εναντίον κάποιας λειτουργίας ενεργοποίησης. Παρακάτω παρατίθεται ένα διάγραμμα ως παράδειγμα μιας λειτουργίας ενεργοποίησης. Όταν ένας νευρώνας ενεργοποιείται στέλνει μια έξοδο που μπορεί να τροφοδοτηθεί σε περισσότερους νευρώνες ή να παραχθεί ως τελική έξοδος. Η παρακάτω εικόνα εμφανίζει ένα νευρωνικό δίκτυο πολλαπλών επιπέδων:



Εικόνα 2 - Νευρωνικό δίκτυο πολλαπλών επιπέδων

2.3.2. Ένα νευρωνικό δίκτυο πολλαπλών επιπέδων

Το προηγούμενο διάγραμμα αντιπροσωπεύει ένα τυπικό απλό νευρωνικό δίκτυο με είσοδο και έξοδο και δύο κρυμμένα στρώματα νευρώνων. Τα στρώματα μεταξύ της εισόδου και της εξόδου ονομάζονται κρυφά, διότι συνήθως δεν αλληλεπιδράμε με αυτά τα στρώματα. Το τελικό στρώμα, το στρώμα εξόδου, είναι εκεί όπου το δίκτυο παράγει το αποτέλεσμα. Αρχικά, αυτό το αποτέλεσμα δεν είναι βέλτιστο, επειδή το δίκτυο είναι ανεκπαίδευτο και δεν έχει αποκτήσει ακόμα γνώση για το τι χαρακτηρίζεται ως βέλτιστο αποτέλεσμα. Προκειμένου να είναι χρήσιμο, ένα νευρωνικό δίκτυο πρέπει να εκπαιδεύεται ως προς τη σωστή ή ίσως λάθος απάντηση. Υπάρχουν πολλοί τρόποι για να επιτευχθεί αυτό, αλλά ο συνηθέστερος τρόπος είναι να μεταφέρονται τα σφάλματα από την έξοδο πίσω στο δίκτυο και να εξετάζονται πάλι.

Η οπισθοδιάδοση του σφάλματος είναι μια θεμελιώδης ιδέα στα νευρωνικά δίκτυα και της βαθιάς μάθησης. Πρόκειται για μια μέθοδο με την οποία η λειτουργία ενεργοποίησης κάθε νευρώνα σταθμίζεται με βάση τη συνεισφορά του στο συνολικό ποσό σφάλματος. Γενικά αυτό πραγματοποιείται με αργό και επαναληπτικό τρόπο για να βρεθεί το καλύτερο δυνατό αποτέλεσμα. Αυτό σημαίνει ότι ένα τυπικό νευρωνικό δίκτυο πρέπει να εκτεθεί σε χιλιάδες δείγματα ή επαναλήψεις για να αναπτυχθεί αποτελεσματικά.

Συνοπτική περιγραφή ενός ΤΝΔ:

- Τα ΤΝΔ συνήθως οργανώνονται σε επίπεδα (layers) τα οποία καλούνται και στρώματα. Τα ενδιάμεσα επίπεδα καλούνται κρυμμένα επίπεδα (hidden layers) και δεν είναι απαραίτητο να υπάρχουν.
- Τα επίπεδα αποτελούνται από έναν αριθμό μονάδων (units) ή κόμβων (nodes) που είναι έτσι συνδεδεμένες μεταξύ τους, ώστε μία μονάδα να έχει συνδέσμους με πολλές άλλες μονάδες του ίδιου ή άλλου επιπέδου.
- Οι μονάδες επιδρούν σε άλλες μονάδες με το να τις διεγείρουν ή να αναστείλουν την ενεργοποίησή τους. Για να επιτευχθεί αυτό, η μονάδα λαμβάνει το σταθμισμένο άθροισμα όλων των εισόδων μέσω των συνδέσμων που καταλήγουν σε αυτήν και παράγει μέσω της συνάρτησης μετάβασης μία μοναδική έξοδο, εάν το άθροισμα υπερβαίνει μία τιμή κατωφλίου.
- Οι είσοδοι παρουσιάζονται στο δίκτυο μέσω του επιπέδου εισόδου (input layer) το οποίο επικοινωνεί με ένα ή περισσότερα κρυμμένα επίπεδα. Τα κρυμμένα επίπεδα συνδέονται με το επίπεδο εξόδου (output layer) από το οποίο εξάγεται η απάντηση. Βασικά στοιχεία της αρχιτεκτονικής των ΤΝΔ που πρέπει να καθοριστούν κατά τη δημιουργία τους είναι:
    - Ο αριθμός των ενδιάμεσων κρυφών επιπέδων

- Ο αριθμός των μονάδων (ή κόμβων) ανά επίπεδο
- Ο τρόπος σύνδεσης των μονάδων μεταξύ τους
- Η τιμή ενεργοποίησης (τιμή κατωφλίου)
- Η μορφή της συνάρτησης μετάβασης
- Οι τιμές των αρχικών βαρών μεταξύ των μονάδων
- Οι αλγόριθμοι (κανόνες εκπαίδευσης) που χρησιμοποιούνται, για να ενισχυθούν οι σύνδεσμοι μεταξύ των μονάδων κατά τη διαδικασία της εκπαίδευσης

2.3. ΜΗΧΑΝΙΚΉ ΜΆΘΗΣΗ

Η Μηχανική Μάθηση (Machine Learning), ως κλάδος της Τεχνητής Νοημοσύνης, ασχολείται με την μάθηση προτύπων από δεδομένα και την μελέτη αλγορίθμων που βελτιώνουν τη συμπεριφορά του υπολογιστή σε κάποια εργασία που του έχει ανατεθεί χρησιμοποιώντας την εμπειρία του. Είναι ένας τρόπος για έναν υπολογιστή να αφομοιώσει τα δεδομένα και να προσφέρει μια μαθησιακή λύση ή απάντηση.

Η μηχανική μάθηση έχει περιγράφει ως το επόμενο μεγάλο τεχνολογικό επίτευγμα για την ανθρωπότητα παρόμοια με την ηλεκτρική ενέργεια. Ενώ αυτό είναι ένας μεγάλος ισχυρισμός, μπορούμε να κάνουμε ορισμένες αναλογίες μεταξύ των δυο τεχνολογιών. Καταρχάς, δεν είναι απαραίτητο για τον χρήστη να καταλάβει την εσωτερική λειτουργία της ηλεκτρικής ενέργειας για να την χρησιμοποιήσει, και κατά κάποιον τρόπο το ίδιο ισχύει και για την μηχανική μάθηση και πολλές από τις πιο προηγμένες έννοιες.

Συχνά θεωρούμε την τεχνητή νοημοσύνη ως έναν ευρύτερο όρο για να αντικατοπτρίζει ένα "έξυπνο" σύστημα. Η μηχανική μάθηση είναι η εφαρμογή της τεχνητής νοημοσύνης που δίνει την ικανότητα σε ένα σύστημα να μαθαίνει από δεδομένα αυτόματα πάρα με ένα μηχανικό τρόπο. Αυτό επιτυγχάνεται με το να τροφοδοτείται το σύστημα με πληροφορίες και παρατηρήσεις οι οποίες μπορούν να χρησιμοποιηθούν για να βρεθούν πρότυπα και να γίνουν προβλέψεις για μελλοντικές εκβάσεις.

Για κάθε πρόβλημα προς επίλυση στο χώρο της μηχανικής μάθησης υπάρχει ένας κατάλληλος τρόπος μάθησης και για κάθε τρόπο μάθησης υπάρχει τουλάχιστον ένας κατάλληλος αλγόριθμος που μπορεί να χρησιμοποιηθεί. Όλοι οι αλγόριθμοι μηχανικής μάθησης διαχειρίζονται τη γνώση αναπαριστώντας την με κάποιον τρόπο αναπαράστασης της γνώσης ή με άλλους πιο μαθηματικοποιημένους τρόπους που θεωρούνται από τον συγκεκριμένο αλγόριθμο καταλληλότεροι να την εκφράσουν. Ορισμένοι αλγόριθμοι δέχονται ως είσοδο μόνο παρατηρήσεις και άλλοι λαμβάνουν υπόψη τους λίγο ή περισσότερο την προϋπάρχουσα γνώση.

Η βασικότερη φάση κάθε αλγόριθμου είναι η εκπαίδευση, όπου ο αλγόριθμος χρησιμοποιεί ως είσοδο ένα σύνολο δεδομένων εκπαίδευσης (training set) προς επίτευξη του σκοπού του για τη δημιουργία νέας γνώσης. Επιπλέον, μπορεί είτε να χρησιμοποιήσει λιγότερο ή περισσότερο την υπάρχουσα γνώση είτε να μην τη χρησιμοποιήσει καθόλου. Την εκπαίδευση ακολουθεί η φάση της πιστοποίησης της παραγόμενης νέας γνώσης. Συνήθως, η πιστοποίηση πραγματοποιείται καταρχάς από τον ίδιο τον αλγόριθμο μέσω διαδικασιών ανάκλησης (recall) με τη βοήθεια δεδομένων ελέγχου (test data) και, στη συνέχεια, μέσω κριτικής που κάνει ο χρήστης βάσει των γνώσεων που διαθέτει για το πρόβλημα που επιχειρεί να λύσει ο αλγόριθμος. Τέλος, η νέα γνώση δίνεται προς χρήση σε εφαρμογές στις οποίες είναι απαραίτητη, για να λυθούν πραγματικά προβλήματα.

2.3.1. Μοντέλα εκμάθησης

Εν γένει, ο τομέας της Μηχανικής Μάθησης αναπτύσσει τρεις τρόπους μάθησης. Πιο αναλυτικά:

- Μη Επιβλεπόμενη Μάθηση (Unsupervised Learning), όπου ο αλγόριθμος κατασκευάζει ένα μοντέλο για κάποιο σύνολο εισόδων υπό μορφή παρατηρήσεων χωρίς να γνωρίζει τις επιθυμητές εξόδους. Χρησιμοποιείται σε προβλήματα:
  - Ανάλυσης Συσχετισμών (Association Analysis)
  - Ομαδοποίησης (Clustering)
- Επιβλεπόμενη Μάθηση (Supervised Learning), η διαδικασία όπου ο αλγόριθμος κατασκευάζει μια συνάρτηση που απεικονίζει δεδομένες εισόδους (σύνολο εκπαίδευσης)

σε γνωστές επιθυμητές εξόδους, με απώτερο στόχο τη γενίκευση της συνάρτησης αυτής και για εισόδους με άγνωστη έξοδο. Χρησιμοποιείται σε προβλήματα:
    - Ταξινόμησης (Classification)
    - Πρόγνωσης (Prediction)
    - Διερμηνείας (Interpretation)
- Ενισχυτική Μάθηση (Reinforcement Learning), όπου ο αλγόριθμος μαθαίνει μια στρατηγική ενεργειών μέσα από άμεση αλληλεπίδραση με το περιβάλλον. Χρησιμοποιείται κυρίως σε προβλήματα Σχεδιασμού (Planning), όπως για παράδειγμα ο έλεγχος κίνησης ρομπότ και η βελτιστοποίηση εργασιών σε εργοστασιακούς χώρους.
- Μάθηση μέσω Μίμησης (Imitation Learning), μια τεχνική όπου οι πράκτορες εκπαιδεύονται βλέποντας μια επίδειξη των επιθυμητών ενεργειών και στη συνέχεια τους μιμούνται.
- Μάθηση με τεχνικές διαδοχικών μαθημάτων (Curriculum Learning), μια προηγμένη μορφή μάθησης που λειτουργεί με την διαχωρισμό ενός προβλήματος σε επίπεδα πολυπλοκότητας, που επιτρέπει στον πράκτορα να μάθει κάθε επίπεδο πολυπλοκότητας προτού προχωρήσει σε πιο προηγμένες δραστηριότητες. Για παράδειγμα, ένας σερβιτόρος μπορεί να πρέπει πρώτα να μάθει να ισορροπεί ένα δίσκο, μετά το δίσκο με ένα πιάτο φαγητού, στη συνέχεια να περπατάει με το δίσκο και το φαγητό και τελικά να παραδίδει το φαγητό σε ένα τραπέζι.
- Βαθιά μάθηση (Deep Learning), τεχνική στην οποία χρησιμοποιούνται διάφορες μορφές εσωτερικών μηχανισμών μάθησης για την εκπαίδευση ενός νευρωνικού δικτύου πολλαπλών επιπέδων.

2.3.2. Παραδείγματα ανά μοντέλο εκμάθησης

Προκειμένου να παρουσιάσουμε κάποιες από αυτές τις έννοιες με έναν πρακτικό τρόπο, ας δούμε κάποια παραδείγματα σεναρίων εφαρμόζοντας κάποιες από τις τεχνικές μάθησης που αναφέρθηκαν παραπάνω.

2.3.2.1. Μη Επιβλεπόμενη Μάθηση (Unsupervised Learning)

Ο στόχος της μη επιβλεπόμενης μάθησης είναι η ομαδοποίηση ή η σύμπλεξη παρόμοιων στοιχείων σε ένα σύνολο δεδομένων. Για παράδειγμα, μπορεί να θέλουμε να ομαδοποιήσουμε τους παίκτες ενός παιχνιδιού ανάλογα με το επίπεδο συμμετοχής τους. Η ταξινόμηση αυτή μπορεί να επιτρέψει την δημιουργία στοχευμένου κοινού. Για παράδειγμα, παίκτες με υψηλό επίπεδο συμμετοχής, θα μπορούσαν να οριστούν ως beta δοκιμαστές για νέες λειτουργίες, ενώ για τους παίκτες που δεν έχουν συμμετάσχει, θα μπορούσε να τους αποσταλεί χρήσιμα ηλεκτρονικά μαθήματα.

Έστω ότι θέλουμε να χωρίσουμε τους παίκτες μας σε δύο ομάδες. Θα πρέπει πρώτα να ορίσουμε τα βασικά χαρακτηριστικά των παικτών, όπως ο αριθμός των ωρών που παίζουν, το σύνολο των χρημάτων που δαπανούν για τις αγορές εντός της εφαρμογής και ο αριθμός των επιπέδων που έχουν ολοκληρώσει. Στη συνέχεια, μπορούμε να τροφοδοτήσουμε αυτό το σύνολο δεδομένων (τρία χαρακτηριστικά για κάθε παίκτη) σε έναν αλγόριθμο μη επιβλεπόμενης μάθησης, όπου θα καθοριστεί ο αριθμό των ομάδων. Στο αναφερόμενο παράδειγμα οι ομάδες είναι δύο. Λαμβάνοντας υπόψη τα χαρακτηριστικά που χρησιμοποιήθηκαν για την περιγραφή του κάθε παίκτη, το αποτέλεσμα, σε αυτή την περίπτωση, θα ήταν ένας διαχωρισμός όλων των παικτών σε δύο ομάδες, όπου μία ομάδα θα εκπροσωπεί σημασιολογικά τους παίκτες με υψηλή συμμετοχή και η δεύτερη ομάδα θα αντιπροσωπεύει σημασιολογικά τους παίκτες που δεν συμμετέχουν ενεργά.

Στο παράδειγμα αυτό της μη επιβλεπόμενης μάθησης δεν προσφέρθηκαν συγκεκριμένα παραδείγματα για τους παίκτες που θεωρούνται ότι έχουν υψηλή συμμετοχή και που δεν συμμετέχουν ενεργά. Το μόνο που ορίστηκε είναι τα κατάλληλα χαρακτηριστικά και χρησιμοποιήθηκε ο αλγόριθμος για να οριστούν οι δύο ομάδες. Αυτός ο τύπος συνόλου δεδομένων καλείται ένα σύνολο μη επισημασμένων δεδομένων καθώς δεν χαρακτηρίζονται

άμεσα από ετικέτες. Κατά συνέπεια, η μάθηση χωρίς επίβλεψη μπορεί να είναι χρήσιμη σε περιπτώσεις όπου αυτές οι ετικέτες μπορεί να είναι δύσκολο να παραχθούν.

2.3.2.2. Επιβλεπόμενη Μάθηση (Supervised Learning)

Στην επιβλεπόμενη μάθηση δεν θέλουμε μόνο να ομαδοποιήσουμε παρόμοια αντικείμενα αλλά να δημιουργήσουμε μια χαρτογράφηση κάθε αντικειμένου στην ομάδα (ή κλάση) στην οποία ανήκει. Χρησιμοποιώντας το παραπάνω παράδειγμα της ομαδοποίησης τον παικτών, ας υποθέσουμε ότι τώρα θέλουμε να προβλέψουμε ποιοι από τους παίκτες μας είναι έτοιμοι να σταματήσουν να παίζουν το παιχνίδι για τις επόμενες 30 ημέρες. Μπορούμε να εξετάσουμε τα ιστορικά αρχεία και να δημιουργήσουμε ένα σύνολο δεδομένων που περιέχει ιδιότητες των παικτών, εκτός από μια ετικέτα που δηλώνει εάν έχουν σταματήσει να παίζουν ή όχι. Επισημαίνεται ότι οι ιδιότητες του παίκτη που χρησιμοποιούνται για την συγκεκριμένη ταξινόμηση μπορεί να διαφέρουν από αυτές που χρησιμοποιήθηκαν στην προηγούμενη ομαδοποίηση.

Στη συνέχεια, μπορούμε να τροφοδοτήσουμε αυτό το σύνολο δεδομένων (χαρακτηριστικά και ετικέτα για κάθε παίκτη) σε έναν αλγόριθμο επιβλεπόμενης μάθησης, ο οποίος θα αντιστοιχίσει τα χαρακτηριστικά των παικτών σε μια ετικέτα η οποία θα δηλώνει εάν ο παίκτης αυτός θα σταματήσει να παίζει ή όχι. Η γενική ιδέα είναι ότι ο αλγόριθμος επιβλεπόμενης μάθησης θα μάθει ποιες τιμές αυτών των χαρακτηριστικών αντιστοιχούν συνήθως σε παίκτες που τα έχουν παρατήσει και σε παίχτες που δεν τα έχουν παρατήσει. Για παράδειγμα, μπορεί να μάθει ότι οι παίκτες που ξοδεύουν πολύ λίγο και παίζουν για πολύ σύντομες περιόδους, πιθανότατα θα τα παρατήσουν.

Λαμβάνοντας υπόψη αυτό το μαθηματικό μοντέλο, μπορούν να προστεθούν τα χαρακτηριστικά ενός νέου παίκτη (που πρόσφατα άρχισε να παίζει το παιχνίδι) και το μοντέλο θα μπορέσει να εξάγει μια προβλεπόμενη ετικέτα για αυτόν τον παίκτη. Αυτή η πρόβλεψη είναι η προσδοκία του αλγορίθμου για το αν ο παίκτης θα τα παρατήσει εύκολα ή όχι. Μπορούμε τώρα να χρησιμοποιήσουμε αυτές τις προβλέψεις για να βρούμε τους παίκτες που αναμένεται να σταματήσουν να παίζουν και να τους δελεάσουμε για να συνεχίσουν να παίζουν το παιχνίδι.

2.3.2.3. Ενισχυτική Μάθηση (Reinforcement Learning)

Η ενισχυτική μάθηση μπορεί να θεωρηθεί ως μια μορφή μάθησης για τη λήψη διαδοχικών αποφάσεων που σχετίζονται συνήθως με τον έλεγχο ρομπότ (αλλά στην πραγματικότητα είναι πολύ πιο γενική). Έστω ότι έχουμε κατασκευάσει ένα αυτόνομο ρομπότ πυρόσβεσης που έχει την εντολή να πλοηγηθεί σε μια περιοχή, να βρει την πυρκαγιά και να την εξουδετερώσει. Σε κάθε δεδομένη στιγμή, το ρομπότ αντιλαμβάνεται το περιβάλλον μέσω των αισθητήρων του (π.χ. κάμερα, θερμότητα), επεξεργάζεται αυτές τις πληροφορίες και παράγει μια ενέργεια (π.χ. μετακίνηση προς τα αριστερά, περιστροφή του σωλήνα νερού, ενεργοποίηση του νερού). Με άλλα λόγια, παίρνει συνεχώς αποφάσεις για το πώς να αλληλεπιδράσει σε αυτό το περιβάλλον παίρνοντας ως δεδομένα την δική του αντίληψη του κόσμου (δηλ. της εισόδου των αισθητήρων) και του σκοπού (δηλαδή εξουδετέρωση της φωτιάς). Η διδασκαλία ενός ρομπότ για να είναι μια επιτυχημένη μηχανή πυρόσβεσης είναι ακριβώς αυτό που η ενισχυτική μάθηση έχει σχεδιαστεί να κάνει.

Πιο συγκεκριμένα, ο στόχος της ενισχυτικής μάθησης είναι να μάθει μια τακτική/πολιτική, η οποία ουσιαστικά είναι μια χαρτογράφηση των παρατηρήσεων στις αντίστοιχες ενέργειες. Μια παρατήρηση είναι αυτό που το ρομπότ μπορεί να μετρήσει από το περιβάλλον του (στην περίπτωση αυτή, όλους τους αισθητήρες εισόδου του) και μια ενέργεια, στην πιο απλοποιημένη μορφή του, είναι μια αλλαγή στη διάρθρωση του ρομπότ (π.χ. θέση του, θέση του σωλήνα νερού και εάν ο σωλήνας νερού είναι ενεργοποιημένος ή όχι).

Το τελευταίο μέρος του αλγορίθμου της ενισχυτικής μάθησης είναι το σήμα ανταμοιβής. Όταν εκπαιδεύουμε ένα ρομπότ ως μια μηχανή πυρόσβεσης, παρέχουμε ανταμοιβές (θετικές και αρνητικές) που δείχνουν πόσο καλά τα πάει στην ολοκλήρωση της εργασίας. Σημειώνεται ότι το ρομπότ δεν ξέρει πώς να σβήσει φωτιές προτού εκπαιδευτεί. Μαθαίνει τον σκοπό επειδή λαμβάνει μια μεγάλη θετική ανταμοιβή όταν σβήνει τη φωτιά και μια μικρή αρνητική ανταμοιβή για κάθε δευτερόλεπτο που περνάει χωρίς να έχει ολοκληρώσει τον στόχο. Το γεγονός ότι οι

ανταμοιβές είναι αραιές (δηλ. μπορεί να μην παρέχονται σε κάθε βήμα, αλλά μόνο όταν ένα ρομπότ φτάνει σε μια επιτυχημένη ή αποτυχημένη κατάσταση) είναι ένα καθοριστικό χαρακτηριστικό της ενισχυτικής μάθησης και ο λόγος που η μάθηση καλών πολιτικών μπορεί να είναι δύσκολη και χρονοβόρα για σύνθετα περιβάλλοντα.

Η εκμάθηση μιας πολιτικής συνήθως απαιτεί πολλές δοκιμές και επαναληπτικές ενημερώσεις πολιτικής. Συγκεκριμένα, το ρομπότ τοποθετείται σε διάφορες καταστάσεις πυρκαγιάς και με την πάροδο του χρόνου μαθαίνει μια βέλτιστη πολιτική που του επιτρέπει να σβήνει τις πυρκαγιές πιο αποτελεσματικά. Προφανώς, δεν μπορούμε να περιμένουμε να εκπαιδεύσουμε επανειλημμένα ένα ρομπότ στον πραγματικό κόσμο, ιδιαίτερα όταν πρόκειται για πυρκαγιές. Αυτός είναι ακριβώς ο λόγος για τον οποιο η εκπαίδευση πραγματοποιείται σε ένα περιβάλλον προσομοίωσης και εκμάθησης τέτοιων συμπεριφορών.

Πέρα από την εκπαίδευση ρομπότ η ενισχυτική μάθηση μπορεί να χρησιμοποιηθεί και σε χαρακτήρες σε παιχνίδια. Στην ουσία ένας non-playable character (NPC) μπορει να χαρακτηριστεί ως ένα εικονικό ρομπότ, με τις δικές του παρατηρήσεις για το περιβάλλον, το δικό του σύνολο ενεργειών και έναν συγκεκριμένο στόχο.

2.3.2.4. Εξειδικευμένα είδη Μάθησης (Deep Learning)

Υπάρχει πληθώρα ειδών μάθησης, άλλα λιγότερο εξειδικευμένα και άλλα περισσότερο. Μια οικογένεια αλγορίθμων, η λεγομένη βαθιά μάθηση, μπορούν να χρησιμοποιηθούν για την αντιμετώπιση οποιωνδήποτε προβλημάτων που παρουσιάστηκαν παραπάνω. Πιο συγκεκριμένα, μπορούν να χρησιμοποιηθούν για την επίλυση τόσο της επιλογής χαρακτηριστικών όσο και μοντέλων. Η βαθιά εκμάθηση έχει κερδίσει δημοτικότητα τα τελευταία χρόνια, λόγω της εξαιρετικής απόδοσης της σε πολλά περίπλοκα προβλήματα μηχανικής μάθησης. Ένα παράδειγμα είναι το AlphaGo, ένα υπολογιστικό πρόγραμμα που παίζει το παιχνίδι Go, το οποίο εκμεταλλευόμενο τη βαθιά εκμάθηση μπόρεσε να νικήσει τον παγκόσμιο πρωταθλητή Go, Lee Sedol.

Ένα βασικό χαρακτηριστικό των αλγορίθμων βαθιάς μάθησης είναι η ικανότητά τους να μαθαίνουν πολύπλοκες λειτουργίες από μεγάλες ποσότητες δεδομένων. Αυτό τους καθιστά κατάλληλους για σενάρια ενισχυτική μάθηση όπου υπάρχει διαθέσιμος μεγάλος όγκος δεδομένων, για παράδειγμα μέσω της χρήσης ενός προσομοιωτή ή μιας μηχανής προσομοίωσης. Δημιουργώντας εκατοντάδες χιλιάδες προσομοιώσεις του περιβάλλοντος, μπορούν οι αλγόριθμοι αυτοί να μάθουν πολιτικές για πολύ περίπλοκα περιβάλλοντα.

2.3.2.5. Σύγκριση μοντέλων εκπαίδευσης

Μια κοινή πτυχή και των τριών κλάδων της μηχανικής μάθησης είναι ότι όλες περιλαμβάνουν μια φάση εκπαίδευσης και μια φάση συμπερασμάτων. Παρόλο που οι λεπτομέρειες των φάσεων είναι διαφορετικές για καθένα από τα τρία μοντέλα, η φάση της εκπαίδευσης και στα τρία μοντέλα περιλαμβάνει την οικοδόμηση ενός μοντέλου χρησιμοποιώντας τα παρεχόμενα δεδομένα, ενώ η φάση των συμπερασμάτων περιλαμβάνει την εφαρμογή αυτού του μοντέλου σε νέα και έως πρότινος άγνωστα δεδομένα.

Τόσο για την επιβλεπόμενη όσο και για τη μη επιβλεπόμενη μάθηση, υπάρχουν δύο εργασίες που πρέπει να εκτελεστούν: επιλογή χαρακτηριστικών και επιλογή μοντέλου. Η επιλογή χαρακτηριστικών (που ονομάζεται επίσης επιλογή ιδιοτήτων) αφορά την επιλογή του τρόπου με τον οποίο επιθυμούμε να εκπροσωπούμε την ενδιαφέρουσα οντότητα, στην περίπτωση αυτή, τον παίκτη. Η επιλογή μοντέλου, από την άλλη πλευρά, αφορά την επιλογή του αλγορίθμου (και των παραμέτρων του) που θα εκτελέσει την εργασία με τον βέλτιστο τρόπο. Και οι δύο αυτές διεργασίες είναι αναγκαία πεδία στην μηχανική μάθηση και, στην πράξη, απαιτούν αρκετές επαναλήψεις για την επίτευξη της καλύτερης απόδοσης.

Παρόμοια με τη επιβλεπόμενη και μη επιβλεπόμενη μάθηση, η ενισχυτική μάθηση περιλαμβάνει δύο κύριες εργασίες: την επιλογή χαρακτηριστικών και επιλογή μοντέλου. Η επιλογή χαρακτηριστικών ορίζει το σύνολο των παρατηρήσεων για το ρομπότ που το βοηθάνε να ολοκληρώσει τον στόχο του, ενώ η επιλογή μοντέλου καθορίζει τη μορφή της πολιτικής (χαρτογράφηση παρατηρήσεις στις αντίστοιχες ενέργειες) και τις παραμέτρους της. Στην πράξη,

η εκπαίδευση συμπεριφορών είναι μια επαναληπτική διαδικασία που μπορεί να απαιτεί την αλλαγή των χαρακτηριστικών και των επιλογών μοντέλου.

Πιο συγκεκριμένα:

- Για το παράδειγμα της μη επιβλεπόμενης μάθησης, η φάση της εκπαίδευσης προσδιορίζει τις δύο βέλτιστες ομάδες με βάση τα δεδομένα που περιγράφουν τους υπάρχοντες παίκτες, ενώ η φάση των συμπερασμάτων αναθέτει έναν νέο παίκτη σε μια από τις δύο ομάδες.
- Για το παράδειγμα της επιβλεπόμενης μάθησης, η φάση της εκπαίδευσης προσδιορίζει τη χαρτογράφηση των χαρακτηριστικών των παικτών στις αντίστοιχες ετικέτες του παίκτη (ανεξάρτητα από το εάν τα έχουν παρατήσει ή όχι) και η φάση των συμπερασμάτων προβλέπει αν ένας νέος παίκτης θα τα παρατήσει ή όχι με βάση αυτής της μαθημένης χαρτογράφησης.
- Για το παράδειγμα της ενισχυτικής μάθησης, η φάση της εκπαίδευσης προσδιορίζει τη βέλτιστη πολιτική μέσω καθοδηγούμενων δοκιμών και στη φάση των συμπερασμάτων ο πράκτορας παρατηρεί και αναπαράγει ενέργειες στο φυσικό περιβάλλον χρησιμοποιώντας τη μαθησιακή πολιτική του.

Συνοπτικά, πέρα από την επιλογή χαρακτηριστικών και μοντέλων και οι τρεις κατηγορίες αλγορίθμων περιλαμβάνουν φάσεις εκπαίδευσης και συμπερασμάτων. Αυτό που τελικά τους διαχωρίζει είναι το είδος των διαθέσιμων δεδομένων προς μάθηση. Στην μη επιβλεπόμενη μάθηση το σύνολο των δεδομένων ήταν μια συλλογή χαρακτηριστικών, στην επιβλεπόμενη μάθηση το σύνολο δεδομένων ήταν μια συλλογή ζευγών χαρακτηριστικών-ετικετών και, τέλος, στην ενισχυτική μάθηση το σύνολο δεδομένων ήταν μια συλλογή πλειάδων παρατηρήσεων-ενεργειών-ανταμοιβών.

2.4. ΜΗΧΑΝΙΚΉ ΜΆΘΗΣΗ ΚΑΙ ΤΕΧΝΗΤΉ ΝΟΗΜΟΣΎΝΗ ΣΤΑ ΠΑΙΧΝΊΔΙΑ

Εδώ και χρόνια τα παιχνίδια και οι τεχνικές προσομοίωσης δεν είναι ξένες έννοιες στις τεχνολογίες τεχνητής νοημοσύνης. Αυτές οι τεχνολογίες περιλαμβάνουν εφαρμογές όπως Δέντρα Συμπεριφοράς, Μηχανή Πεπερασμένων Καταστάσεων, πλέγματα πλοήγησης, A* και άλλους ευρετικούς τρόπους που χρησιμοποιούν οι προγραμματιστές παιχνιδιών για την προσομοίωση της νοημοσύνης. Ενώ για αρκετά χρόνια γινόταν χρήση βασικών τεχνικών της τεχνικής ευφυίας σε παιχνίδια, τα τελευταία χρόνια παρατηρείται μια μαζική μετακίνηση των προγραμματιστών παιχνιδιών προς την μηχανική μάθηση. Αυτό οφείλεται σε μεγάλο βαθμό στην πρωτοβουλία OpenAI, μια πρωτοβουλία που ενθαρρύνει την έρευνα σε όλο τον ακαδημαϊκό κόσμο και τη βιομηχανία να μοιραστούν ιδέες και έρευνες για τα τεχνητή νοημοσύνη και τη μηχανική μάθηση. Αυτό έχει ως αποτέλεσμα την έκρηξη της ανάπτυξης νέων ιδεών, μεθόδων και τομέων έρευνας. Αυτό σημαίνει για παιχνίδια και προσομοιώσεις ότι δεν χρειάζεται πλέον να προσομοιώνεται η νοημοσύνη αλλά μπορούν να δημιουργούνται πράκτορες που μαθαίνουν από το περιβάλλον τους και μπορούν να μάθουν να κερδίζουν τους ανθρώπινους αντιπάλους τους.

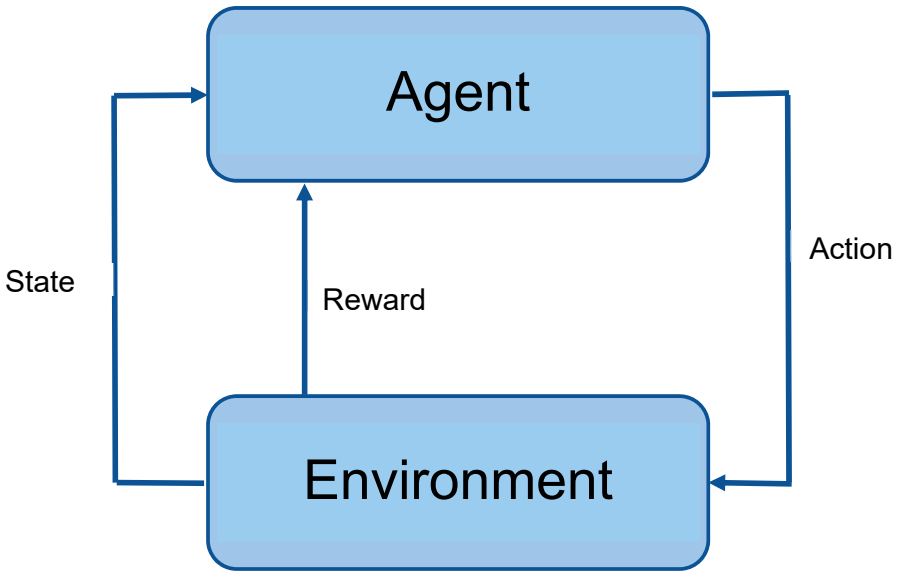
Η πλειοψηφία της τεχνητής νοημοσύνης στα παιχνίδια που υπάρχει έως σήμερα είναι ενσωματωμένη στον κώδικα (hardcoded) και αποτελείται από δέντρα αποφάσεων με μερικές φορές έως και χιλιάδες κανόνες. Όλα αυτά πρέπει να συντηρούνται χειροκίνητα και να ελέγχονται διεξοδικά. Αντίθετα, η μηχανική μάθηση βασίζεται σε αλγόριθμους που μπορούν να κατανοήσουν τα ανεπεξέργαστα δεδομένα, χωρίς να χρειάζεται κάποιον ειδικό να καθορίσει τον τρόπο ερμηνείας αυτών των δεδομένων. Για παράδειγμα, στην μηχανική όραση το πρόβλημα ταξινόμησης του περιεχομένου μιας εικόνας. Μέχρι πριν από λίγα χρόνια, οι ειδικοί θα έγραφαν ενσωματωμένα φίλτρα, τα οποία θα εξήγαγαν χρήσιμα χαρακτηριστικά για την ταξινόμηση μιας εικόνας που εμπεριείχε μια γάτα ή εάν σκύλο. Αντίθετα, η μηχανική μάθηση και ειδικότερα οι νεότερες προσεγγίσεις της βαθιάς μάθησης (Deep Learning), χρειάζονται μόνο τις εικόνες και τις ετικέτες των κλάσεων και μαθαίνουν αυτόματα τις χρήσιμες λειτουργίες. Θεωρείται ότι αυτή η αυτοματοποιημένη μάθηση μπορεί να βοηθήσει στην απλοποίηση και την επιτάχυνση της διαδικασίας παραγωγής παιχνιδιών και κατά συνέπεια να ενσωματώσει στον τομέα του game development ένα εύρος προσομοιώσεων σεναρίων της μηχανικής μάθησης. Υπάρχουν μέθοδοι οι οποίες είναι κατάλληλες για πιο περίπλοκες συμπεριφορές πρακτόρων σε παιχνίδια με πλούσιο γραφικό περιβάλλον κάνοντας χρήση νευρωνικών δικτύων (deep neural networks).

Συγκεκριμένα, αυτή η αυτοματοποιημένη εκμάθηση μπορεί να εφαρμοστεί στη συμπεριφορά των πρακτόρων, μεταξύ άλλων και των NPC (Non-Player Character). Μπορεί να χρησιμοποιηθεί η Ενισχυτική Μάθηση για να εκπαιδευτούν πράκτορες να υπολογίζουν την αξία μιας ενέργειας που μπορούν να λάβουν μέσα σε ένα περιβάλλον. Αφού εκπαιδευτούν, αυτοί οι πράκτορες μπορούν να αναλάβουν δράση για να λάβουν τη μεγαλύτερη αξία, χωρίς να χρειάζεται να προγραμματιστεί ρητά πώς να ενεργήσουν.

Με πιο προχωρημένες μεθόδους μηχανικής μάθησης είναι δυνατόν να εκπαιδευτούν πράκτορες οι οποίοι θα λειτουργούν ως συμπαίκτες ή αντίπαλοι σε ένα εύρος τύπου παιχνιδιών το οποίο μπορεί να είναι παιχνίδια τύπου μάχης και οδήγησης μέχρι και first person shooter και παιχνίδια real time στρατηγικής. Και όλα αυτά μόνο με την υλοποίηση του τι πρέπει ο πράκτορας να καταφέρει και όχι πως, δηλαδή χωρίς να χρειάζεται να γραφτεί ούτε ένας κανόνας.

2.4.1. Ενισχυτική Μάθηση σε παιχνίδια

Όπως αναφέρθηκε παραπάνω, μια βασική ιδέα πίσω από την ενισχυτική μάθηση είναι η εκτίμηση της αξίας κάθε ενέργειας και η ανάλογη δράση βάσει της εκτιμώμενης τιμής. Στην ενισχυτική μάθηση, αυτός που εκτελεί την ενέργεια ονομάζεται πράκτορας και αυτό που χρησιμοποιεί για τη λήψη αποφάσεων σχετικά με τις ενέργειές του ονομάζεται πολιτική. Ένας πράκτορας είναι πάντα ενσωματωμένος σε ένα περιβάλλον και σε οποιαδήποτε δεδομένη στιγμή ο πράκτορας είναι σε μια ορισμένη κατάσταση. Από αυτή την κατάσταση, μπορεί να εκτελέσει μια ενέργεια από ένα σύνολο ενεργειών. Η αξία μιας δεδομένης κατάστασης αναφέρεται στην τελική αξία της ανταμοιβής που θα λάβει ο πράκτορας από την κατάσταση. Η πραγματοποίηση κάποιας ενέργειας σε μια κατάσταση μπορεί να θέσει έναν πράκτορα σε μια νέα κατάσταση, να προσφέρει ανταμοιβή ή και τα δύο. Η συνολική αθροιστική ανταμοιβή είναι αυτό που όλοι οι πράκτορας ενισχυτικής μάθηση προσπαθούν να μεγιστοποιήσει με την πάροδο του χρόνου.



Εικόνα 3 - Η αλληλεπίδραση πράκτορα - περιβάλλοντος

2.4.1.1. Ενισχυτική Μάθηση σε προβλήματα κουλοχέρηδων (bandit problem)

Η πιο απλή εκδοχή ενός προβλήματος ενισχυτικής μάθησης ονομάζεται πρόβλημα των πολλαπλών κουλοχέρηδων (multi-armed bandit). Το πρόβλημα των κουλοχέρηδων ονομάζεται έτσι λόγω της ομοιότητας του με τους κουλοχέρηδες στα καζίνο. Αυτό το όνομα προέρχεται από το πρόβλημα βελτιστοποίησης των κερδών σε πολλαπλές μηχανές κουλοχέρηδων, οι οποίες επίσης αναφέρονται και ως "single-arm bandits", δεδομένης της τάσης τους να κλέβουν κέρματα

από τους χρήστες τους. Σε αυτο το παράδειγμα, το περιβάλλον αποτελείται από μία μόνο κατάσταση και ο πράκτορας μπορεί να λάβει μία από τις ν ενέργειες. Κάθε ενέργεια παρέχει άμεση ανταμοιβή στον πράκτορα. Ο στόχος του πράκτορα είναι να μάθει να επιλέγει τη δράση που προσφέρει τη μεγαλύτερη ανταμοιβή.

Πιο συγκεκριμένα, έστω ότι το σενάριο του παιχνιδιού είναι τύπου Dungeon crawl. Ο πράκτορας εισέρχεται σε ένα δωμάτιο στο οποίο και βρίσκει μια σειρά από μπαούλα που παρατάσσονται κατά μήκος του τείχους. Κάθε ένα από αυτά τα μπαούλα έχει μια ορισμένη πιθανότητα να περιέχει είτε ένα διαμάντι (ανταμοιβή +1) είτε ένα εχθρικό φάντασμα (ανταμοιβή -1).

Ο στόχος του πράκτορα είναι να μάθει ποιο μπαούλα είναι πιο πιθανό να έχει το διαμάντι (για παράδειγμα, το τρίτο από τα δεξιά). Ο λογικός τρόπος για να ανακαλύψει ποιο μπαούλο είναι αυτό με την επιβράβευση είναι να ανοίξει καθένα από τα μπαούλα. Πράγματι, ένα μεγάλο μέρος της ενισχυτικής μάθησης αποτελείται από δοκιμή και λάθος έως ότου ο πράκτορας έχει μάθει αρκετά για τον κόσμο έτσι ώστε να ενεργήσει βέλτιστα. Με βάση της ορολογίας της ενισχυτικής μάθησης, το άνοιγμα κάθε μπαούλου αντιστοιχεί στη λήψη μιας σειράς ενεργειών (ανοίγοντας κάθε μπαούλο πολλές φορές) και η εκμάθηση αντιστοιχεί στην ενημέρωση της εκτίμησης της αξίας κάθε ενέργειας. Μόλις ο πράκτορας έχει εκπαιδευτεί αρκετά, τότε μπορεί να επιλέγει πάντοτε το μπαούλο με την υψηλότερη εκτιμώμενη αξία.

Αυτές οι εκτιμώμενες αξίες μπορούν να εξαχθούν χρησιμοποιώντας μια επαναληπτική διαδικασία την οποία αρχικοποιούμε με μια αρχική σειρά εκτιμήσεων $V(a)$ και στη συνέχεια την προσαρμόζουμε κάθε φορά που κάνουμε μια ενέργεια και παρατηρούμε το αποτέλεσμα της. Η εξίσωση αυτή γράφεται ως εξής:

$$V(a) = V(a) + a * (r - V(a))$$

Όπου a αντιστοιχεί στο ρυθμό μάθησης, το $V(a)$ είναι η εκτιμώμενη αξία μιας δεδομένης ενέργειας και r είναι η άμεση ανταμοιβή που λαμβάνεται για τη λήψη αυτής της ενέργειας.

Εκ πρώτης όψεως, η παραπάνω εξίσωση δηλώνει ότι η τρέχουσα εκτιμώμενη αξία προσαρμόζεται ανάλογα με την αξία της ανταμοιβής που ελήφθη. Με αυτό τον τρόπο διασφαλίζεται ότι πάντα αλλάζουν οι εκτιμήσεις έτσι ώστε να αντικατοπτρίζουν καλύτερα την πραγματική δυναμική του περιβάλλοντος. Επίσης, διασφαλίζεται ότι οι εκτιμήσεις δεν θα γίνουν αδικαιολόγητα μεγάλες, όπως θα συνέβαινε αν απλά υπολογίζονταν τα θετικά αποτελέσματα. Σε κώδικα αυτό μπορεί να επιτευχθεί διατηρώντας ένα διάνυσμα εκτιμήσεων και αναφέροντάς τες με ένα δείκτη που αντιστοιχεί στην ενέργεια που επέλεξε ο πράκτορας.

2.4.1.2. Το πρόβλημα κουλοχέρηδων με συνυπολογισμό περιβάλλοντος (Contextual Bandits)

Το σενάριο που περιγράφεται παραπάνω στερείται μιας σημαντικής πτυχής οποιουδήποτε ρεαλιστικού περιβάλλοντος: έχει μόνο μια κατάσταση. Στην πραγματικότητα (και σε οποιοδήποτε περιβάλλον παιχνιδιού), ένα οποιοδήποτε περιβάλλον μπορεί να έχει από δεκάδες, πχ. δωμάτια σε ένα σπίτι, έως δισεκατομμύρια πιθανές καταστάσεις. Κάθε μια από αυτές τις καταστάσεις μπορεί να έχει τη δική της μοναδική δυναμική από την άποψη του τρόπου με τον οποίο οι ενέργειες παρέχουν νέες ανταμοιβές ή επιτρέπουν την κίνηση μεταξύ των καταστάσεων. Ως εκ τούτου, πρέπει οι ενέργειες να περιορίζονται για την κατάσταση και κατ' επέκταση και η εκτιμώμενη αξία τους.

Άρα, συμβολικά, θα πρέπει να χρησιμοποιείται το $Q(s, a)$ αντί για το $V(a)$. Αυτό σημαίνει ότι η ανταμοιβή που αναμένεται να ληφθεί είναι πλέον συνάρτηση τόσο της ενέργειας που αναλαμβάνεται όσο και της κατάστασης στην οποία βρίσκεται ο πράκτορας κατά τη λήψη αυτής της ενέργειας.

Στο παιχνίδι τύπου Dungeon crawl, η έννοια της κατάστασης μπορεί να μας επιτρέπει να έχουμε διαφορετικά σετ μπαούλων σε διαφορετικά δωμάτια. Κάθε ένα από αυτά τα δωμάτια μπορεί να έχει διαφορετικό μπαούλο με την υψηλότερη εκτιμώμενη αξία και ως εκ τούτου ο πράκτορας πρέπει να μάθει να επιλέγει διαφορετικές ενέργειες σε διαφορετικά δωμάτια. Αυτό

μπορεί να επιτευχθεί μέσω κώδικα διατηρώντας μια μήτρα εκτιμώμενων αξιών, αντί απλώς ενός διανύσματος. Αυτή η μήτρα μπορεί να έχει ως ευρετήριο την κατάσταση και την ενέργεια.

2.4.1.3. Εξερεύνηση και αξιοποίηση

Υπάρχει ένα ακόμη σημαντικό κομμάτι του παζλ για να μπορέσει η Ενισχυτική Μάθηση να δουλέψει. Πριν ο πράκτορας μάθει μια πολιτική για να επιλέγει την βέλτιστη ενέργεια με την μεγαλύτερη επιβράβευση, χρειάζεται μια πολιτική που θα του επιτρέψει να μάθει αρκετές πληροφορίες για τον κόσμο, για να γνωρίζει τι εστί βέλτιστο.

Αυτό δημιουργεί το κλασικό δίλημμα για τον τρόπο εξισορρόπησης της εξερεύνησης (μάθηση σχετικά με τη δομή των αξιών του περιβάλλοντος μέσω δοκιμών και σφαλμάτων) και την αξιοποίηση (ενεργώντας με βάση της μαθημένης δομής των αξιών του περιβάλλοντος). Μερικές φορές αυτοί οι δύο στόχοι ευθυγραμμίζονται, αλλά συχνά δεν το κάνουν. Υπάρχουν ορισμένες στρατηγικές που πρέπει να ληφθούν για την εξισορρόπηση αυτών των δύο στόχων. Παρακάτω περιγράφονται μερικές προσεγγίσεις:

- Μια απλή αλλά ισχυρή στρατηγική ακολουθεί την αρχή της «αισιοδοξίας ενόψει της αβεβαιότητας». Η ιδέα εδώ είναι ότι ο πράκτορας ξεκινά με υψηλές εκτιμώμενες αξίες $V(a)$ για κάθε ενέργεια, έτσι ώστε ενεργώντας άπληστα (να πράττει την ενέργεια με την μέγιστη αξία) να οδηγεί τον πράκτορα να διερευνήσει κάθε μία από τις ενέργειες τουλάχιστον μία φορά. Εάν η ενέργεια δεν οδηγήσει σε καλή ανταμοιβή, η εκτιμώμενη αξία θα μειωθεί αναλόγως, ειδάλλως η εκτιμώμενη αξία θα παραμείνει υψηλή, καθώς αυτή η ενέργεια μπορεί να είναι καλή υποψήφια για να την δοκιμάσει και πάλι στο μέλλον. Από μόνο του, όμως, αυτό το ευρετικό δεν είναι αρκετό, αφού ίσως χρειαστεί να συνεχίσουμε να εξερευνούμε μια δεδομένη κατάσταση για να βρούμε μια σπάνια, αλλά υψηλή ανταμοιβή.
- Μια άλλη στρατηγική είναι να προστεθεί τυχαίος θόρυβος στις εκτιμώμενες αξίες για κάθε ενέργεια και στη συνέχεια να ενεργήσει ο πράκτορας με άπληστους όρους με βάση αυτές τις νέες εκτιμήσεις με θόρυβο. Με αυτή την προσέγγιση, όσο ο θόρυβος είναι μικρότερος από τη διαφορά μεταξύ της πραγματικά βέλτιστης ενέργειας και των άλλων ενεργειών, θα πρέπει να συγκλίνουν στην βέλτιστη εκτίμηση αξιών.
- Προχωρώντας ένα ακόμα βήμα παραπέρα, θα μπορούσε κάποιος να επωφεληθεί από τη φύση των ίδιων των εκτιμώμενων αξιών, κανονικοποιώντας τις και πιθανολογώντας λαμβάνοντας ενέργειας. Στην περίπτωση αυτή, εάν οι εκτιμήσεις για κάθε ενέργεια ήταν σχεδόν ίσες, τότε θα επιλέγονταν ενέργειες με ίσες πιθανότητες. Από την άλλη πλευρά, εάν μια ενέργεια είχε πολύ μεγαλύτερη εκτιμώμενη αξία, τότε θα επιλεγόταν πιο συχνά. Με τον τρόπο αυτό σταδιακά θα εξαλείφονταν οι ενέργειες χωρίς ανταμοιβή επιλέγοντας τις όλο και πιο σπάνια.

2.4.2. Ενισχυτική Μάθηση με Q-Learning

Ο στόχος όταν χρησιμοποιείται η ενισχυτική μάθηση είναι να εκπαιδευτεί ένας πράκτορας με τέτοιο τρόπο ώστε να μεγιστοποίηση τις μελλοντικές αναμενόμενες ανταμοιβές μέσα σε ένα συγκεκριμένο περιβάλλον. Στα προβλήματα του κουλοχέρη (Contextual Bandits) που παρουσιάστηκε παραπάνω το περιβάλλον ήταν σχετικά στατικό και με αρκετά απλές καταστάσεις. Οι καταστάσεις του ήταν σε ποιο από τα τρία δωμάτια βρισκόταν ο πράκτορας και ποιο από τα τρία μπαούλα μέσα στο δωμάτιο άνοιγε. Η έλλειψη αραιών ανταμοιβών και οι μεταβολές καταστάσεων καθιστούν το συγκεκριμένο πρόβλημα απλοϊκό, αρκετά αρχάριο και στην ουσία δεν μπορεί να χαρακτηριστεί και ως ένα πραγματικό πρόβλημα ενισχυτικής μάθησης.

Με αραιές ανταμοιβές αναφερόμαστε στην τακτική να μην λαμβάνεται ανταμοιβή για κάθε ενέργεια που πράττει ο πράκτορας αλλά να λαμβάνεται η ανταμοιβή για κάποια ενέργεια, παρόλο που μπορεί στην πραγματικότητα να είναι η βέλτιστη, με μια καθυστέρηση ή έπειτα από μια σειρά βέλτιστων ενεργειών. Πιο συγκεκριμένα, για παράδειγμα ενώ ο πράκτορας μπορεί να έχει βρει και να ακολουθεί το σωστό μονοπάτι, την τελική ανταμοιβή θα την λάβει στο τέλος της διαδρομής και όχι για κάθε βήμα που κάνει προς την σωστή κατεύθυνση. Κάθε ένα από αυτά τα βήματα – ενέργειες παρόλο που μπορεί να μην απέδωσαν ανταμοιβή ήταν απαραίτητα για την επίτευξη της

τελικής ανταμοιβής. Για να είναι εφικτό αυτό χρειαζόμαστε ένα τρόπο ανταμοιβής με πίστωση, δηλαδή να επιτρέπεται έστω έμμεσα στον πράκτορα να μάθει ότι οι προηγούμενες ενέργειες ήταν πολύτιμες.

Το δεύτερο στοιχείο που λείπει είναι ότι στην ενισχυτική μάθηση συνήθως υπάρχει μια μετάβαση μεταξύ των καταστάσεων. Με αυτό τον τρόπο οι ενέργειες του πράκτορα όχι μόνο παράγουν ανταμοιβές βάσει μιας συνάρτησης ανταμοιβών $R(s,a) \rightarrow r$ αλλά παράγουν νέες καταστάσεις σύμφωνα με μια συνάρτησης μετάβασης καταστάσεων $P(s,a)$ →s'. Για παράδειγμα, κάθε βήμα που κάνει ο πράκτορας στο μονοπάτι τον πηγαίνει σε ένα νέο μέρος επομένως σε μια νέα κατάσταση. Ως εκ τούτου, θέλουμε ο πράκτορας μας όχι μόνο να μάθει να επιλέγει ενέργειες που θα βελτιστοποιήσουν την τρέχουσα δυνατή ανταμοιβή, αλλά να επιλέγει ενέργειες που θα τον οδηγήσουν σε νέες καταστάσεις μέσα στο περιβάλλον που ξέρουμε ότι θα προσφέρουν ακόμα μεγαλύτερη επιβράβευση.

2.4.2.1.Εξίσωση Bellman

Ενώ αυτά τα δυο πρόσθετα στοιχεία πολυπλοκότητας μπορεί αρχικά να φαίνεται ότι δεν σχετίζονται μεταξύ τους στην πραγματικότητα συνδέονται άμεσα. Και τα δυο υποδηλώνουν μια σχέση μεταξύ των μελλοντικών καταστάσεων στις οποίες μπορεί να βρεθεί ο πράκτορας και των μελλοντικών ανταμοιβών που μπορεί να λάβει. Μπορούμε να επωφεληθούμε από αυτή την σχέση για να μάθουμε στον πράκτορα να επιλέγει την βέλτιστη ενέργεια αποκτώντας την ικανότητα να προβλέπει την έκβαση των πραγμάτων. Δηλαδή, ότι κάτω από μια αληθινά βέλτιστη συνάρτηση Q (μια θεωρητική συνάρτηση που μπορεί ή μπορεί και ποτέ να μην υπάρξει) η αξία μιας τρέχουσας κατάστασης και ενέργειας μπορεί να αναλυθεί στην άμεση ανταμοιβή r συν την προεξοφλημένη μέγιστη μελλοντική αναμενόμενη ανταμοιβή από την επόμενη κατάσταση που ο πράκτορας θα βρεθεί για την επιλογή της συγκεκριμένης ενέργειας:

$$\gamma max_{a'} Q^*(s', a')$$

Αυτό ονομάζεται εξίσωση Bellman και μπορεί να γραφτεί ως εξής:

$$Q^*(s,a) = r + \gamma max_{a'} Q^*(s', a')$$

Εδώ το γ είναι ένας όρος έκπτωσης, ο οποίος σχετίζεται με το πόσο θέλουμε ο πράκτορας μας να νοιάζεται για μελλοντικές πιθανές ανταμοιβές. Εάν ορίσαμε το $\gamma = 1{,}0$ ο πράκτορας μας θα αξιολογήσει με την ίδια βαρύτητα όλες τις πιθανές μελλοντικές ανταμοιβές και σε εκπαιδευτικές περιόδους που δεν τελειώνουν ποτέ, η εκτιμώμενη αξία μπορεί να φτάσει στο άπειρο. Για το λόγο αυτό, θέτουμε το γ σε κάτι μεγαλύτερο από 0 και μικρότερο από 1. Οι τυπικές τιμές είναι μεταξύ 0,7 και 0,99.

Η εξίσωση Bellman είναι χρήσιμη επειδή μας παρέχει έναν τρόπο να σκεφτούμε την ενημέρωση της συνάρτησης Q, ξεκινώντας από την ίδια την συνάρτηση Q. Το $Q^*(s,a)$ αναφέρεται σε μια βέλτιστη συνάρτηση Q, αλλά ακόμη και οι τρέχουσες, υπο-βέλτιστες εκτιμήσεις τιμής Q της επόμενης κατάστασης μπορεί να βοηθήσουν να ωθήσουμε την εκτίμηση της τρέχουσας κατάστασης προς μια πιο ακριβή κατεύθυνση.

Δεδομένου ότι στηριζόμαστε κυρίως στις πραγματικές ανταμοιβές σε κάθε βήμα, μπορούμε να εμπιστευτούμε ότι η εκτίμηση της τιμής Q θα βελτιωθεί σταδιακά. Μπορούμε να χρησιμοποιήσουμε την εξίσωση Bellman για να ενημερώσουμε την ακόλουθη νέα ενημερωμένη έκδοση της συνάρτησης Q-learning:

$$Q(s,a) = Q(s,a) + a(r + \gamma max_{a'} Q(s', a') - Q(s,a))$$

Αυτό μοιάζει με τον προηγούμενο αλγόριθμο κουλοχέρη (Contextual Bandits), εκτός από το ότι ο στόχος μας Q περιλαμβάνει πλέον την προεξοφλημένη μελλοντική αναμενόμενη ανταμοιβή στο επόμενο βήμα.

3. Περιβάλλον και εργαλεια

Στο κεφάλαιο αυτό περιγράφεται η πλατφόρμα υλοποίησης παιγνίων Unity3D, η βιβλιοθήκη εργαλείων ML-Agents, καθώς και η διασύνδεση μεταξύ τους για την υλοποίηση παιγνίων με μηχανική μάθηση.

3.1. Η ΠΛΑΤΦΟΡΜΑ UNITY3D

Η Unity3D είναι μια πλατφόρμα εργαλείων καθώς και μηχανή παιχνιδιών που προσφέρει υψηλής ποιότητας χαρακτηριστικά και εργαλεία διαχείρισης περιεχομένου, τα οποία την καθιστούν ιδανική για χρήση ως πλατφόρμα ταχείας ανάπτυξης βιντεοπαιχνιδιών καθώς και προσομοιώσεων. Ο προγραμματισμός σε αυτό το περιβάλλον βασίζεται σε scripts κώδικα γραμμένα σε γλώσσες C#, Javascript ή Boo, βασιζόμενος στην πλατφόρμα ανοιχτού κώδικα Mono.

Παρέχεται δωρεάν πακέτο χρήσης που την καθιστά προσιτή σε πληθώρα προγραμματιστών. Επιτρέπει στους ερευνητές και τους προγραμματιστές να μετατρέψουν τα παιχνίδια και τις προσομοιώσεις που δημιουργούν χρησιμοποιώντας τον επεξεργαστή Unity σε περιβάλλοντα όπου οι ευφυείς πράκτορες μπορούν να εκπαιδευτούν χρησιμοποιώντας μεθόδους εκμάθησης βαθιάς ενίσχυσης, εξελικτικές στρατηγικές ή άλλες μεθόδους μηχανικής μάθησης μέσω ενός απλού Python API.

3.1.1. Μηχανικη μαθηση Unity

Η Unity έχει συμπεριλάβει την ιδέα της ενσωμάτωσης της μηχανικής μάθησης σε όλες τις πτυχές του προϊόντος της και όχι μόνο για χρήση ως ευφυείς πράκτορες σε παιχνίδια. Ενώ οι περισσότεροι προγραμματιστές μπορεί να προσπαθήσουν να χρησιμοποιήσουν την μηχανική μάθηση για τυχερά παιχνίδια, σίγουρα μπορεί να βοηθήσει την ανάπτυξη παιχνιδιών και στους ακόλουθους τομείς:

- **Παραγωγή χαρτών και επιπέδων (Map / Level Generation):** Υπάρχουν ήδη πολλά παραδείγματα όπου οι προγραμματιστές χρησιμοποιούν την μηχανική μάθηση για να δημιουργήσουν τα πάντα από μπουντρούμια έως και ρεαλιστικά εδάφη. Αυτή η δυνατότητα προσφέρει την δημιουργία παιχνιδιών που κάθε επίπεδο ή πίστα να είναι διαφορετικό με ατελείωτη επαναληψιμότητα. Η δυνατότητα αυτή μπορεί να είναι ένα από τα πιο περίπλοκα παραδείγματα μηχανικής μάθησης ως προς την ανάπτυξη τους.
- **Υφή / Shader Generation:** Ένας άλλος τομέας όπου μπορεί να χρησιμοποιηθεί η μηχανική μάθηση είναι η υφή και η παραγωγή shader. Αυτές οι τεχνολογίες παίρνουν μια ώθηση που προήλθε από την προσοχή των προηγμένων γενετικών αντίθετων δικτύων, ή GAN. Υπάρχουν πολλά μεγάλα και διασκεδαστικά παραδείγματα αυτής της τεχνολογίας στην πράξη τα λεγόμενα Deepfake.
- **Παραγωγή μοντέλου:** Υπάρχουν μερικά έργα που υλοποιούνται σε αυτόν τον τομέα, τα οποία θα μπορούσαν να απλοποιήσουν σημαντικά την κατασκευή τρισδιάστατων αντικειμένων μέσω ενισχυμένης σάρωσης ή / και αυτόματης δημιουργίας. Η λογική στην παραγωγή ενός μοντέλου είναι ότι ο χρήστης περιγράφει με κείμενο ένα απλό μοντέλο και η μηχανική μάθηση θα αναλάβει να το χτίσει, σε πραγματικό χρόνο, για παράδειγμα σε ένα παιχνίδι ή σε άλλη εφαρμογή AR / VR / MR
- **Δημιουργία ήχου:** Η μηχανική μάθηση είναι σε θέση να παράγει ηχητικά εφέ ή μουσική και ήδη ερευνάται η δυνατότητα αυτή και για άλλους τομείς πέρα από τον τομέα των παιχνιδιών. Η δυνατότητα αυτή προσφέρει στον προγραμματιστή να έχει ένα ειδικά σχεδιασμένο soundtrack για το παιχνίδι του το οποίο αναπτύχθηκε με μηχανική μάθηση.
- **Τεχνητοί παίκτες:** Οι τεχνητοί παίκτες δημιουργημένοι με μηχανική μάθηση περιλαμβάνουν πολλές συμπεριφορές από τους ίδιους τους παίκτες για να παίξουν ένα παιχνίδι. Δηλαδή είναι σε θέση να παίζουν τα παιχνίδια εξ ονόματός του προγραμματιστή ως βελτιωμένοι πράκτορες δοκιμών ή ως έναν τρόπο εμπλοκής των παικτών κατά τη διάρκεια χαμηλής δραστηριότητας. Εάν το εκάστοτε παιχνίδι είναι αρκετά απλό, θα μπορούσαν επίσης χρησιμοποιηθούν ως ένας τρόπος για τον αυτοματοποιημένο ελέγχου επιπέδων.

- **NPC ή ευφυείς πράκτορες:** Επί του παρόντος, υπάρχουν αρκετά μοντέλα πέρα από την μηχανική μάθηση για να μοντελοποιηθεί νοημοσύνη με βασικές συμπεριφορές για παράδειγμα με τη μορφή δένδρων συμπεριφοράς. Παρόλο που είναι απίθανο τα δέντρα συμπεριφοράς ή άλλα παρόμοια μοντέλα να εξαφανιστούν σύντομα, η μοντελοποίηση με μηχανική μάθηση θα μπορούσε να είναι σε θέση να μοντελοποιήσει έναν πράκτορα ο οποίος μπορεί να εμφανίσει απρόβλεπτες και πρωτότυπες συμπεριφορές.

3.2. TENSORFLOW

Όπως θα περιγραφεί και στις επόμενες ενότητες, για τη μηχανική μάθηση πολλοί αλγόριθμοι που παρέχονται στο πακέτο εργαλείων ML-Agents αξιοποιούν κάποια μορφή βαθιάς μάθησης. Οι υλοποιήσεις που παρέχονται κάνουν χρήση της βιβλιοθήκης ανοιχτού κώδικα TensorFlow. Αυτό σημαίνει ότι τα μοντέλα που παράγονται από το πακέτο εργαλείων ML-Agents είναι (επί του παρόντος) σε μορφή που κατανοείται μόνο από το TensorFlow. Παρακάτω παρέχεται μια σύντομη επισκόπηση της βιβλιοθήκης TensorFlow και των εργαλείων που σχετίζονται με αυτή, τα οποία χρησιμοποιούνται στο πλαίσιο του πακέτου εργαλείων ML-Agents.

3.2.1. TensorFlow

Το TensorFlow είναι μια βιβλιοθήκη ανοιχτού κώδικα για την εκτέλεση υπολογισμών χρησιμοποιώντας γραφήματα ροής δεδομένων, την υποκείμενη αναπαράσταση μοντέλων βαθιάς μάθησης. Διευκολύνει την εκπαίδευση και την εξαγωγή συμπερασμάτων σε CPU και GPU σε επιτραπέζιο υπολογιστή, server ή κινητή συσκευή. Μέσα στο πακέτο εργαλείων ML-Agents, όταν εκπαιδεύεται η συμπεριφορά ενός πράκτορα, η έξοδος είναι ένα αρχείο TensorFlow (.bytes) το οποίο μπορεί στη συνέχεια να ενσωματωθεί σε έναν εγκέφαλο.

3.2.2. TensorBoard

Ένα στοιχείο των μοντέλων εκπαίδευσης με το TensorFlow καθορίζει τις τιμές ορισμένων χαρακτηριστικών μοντέλων (που ονομάζονται υπερπαραμετρικά). Η εύρεση των σωστών τιμών αυτών των υπερπαραμέτρων μπορεί να απαιτήσει λίγες επαναλήψεις. Συνεπώς, χρησιμοποιούμε ένα εργαλείο απεικόνισης μέσα στο TensorFlow που ονομάζεται TensorBoard. Επιτρέπει την απεικόνιση ορισμένων ιδιοτήτων παράγοντα (π.χ. επιβράβευση) κατά τη διάρκεια της κατάρτισης, η οποία μπορεί να είναι χρήσιμη τόσο για την οικοδόμηση διαίσθησης για τους διαφορετικούς υπερπαραμετρικούς παράγοντες όσο και για τη ρύθμιση των βέλτιστων τιμών για το περιβάλλον Unity.

3.2.3. TensorFlowSharp

Ένα από τα μειονεκτήματα του TensorFlow είναι ότι δεν παρέχει ένα εγγενές C# API. Αυτό σημαίνει ότι ο εσωτερικός εγκέφαλος δεν υποστηρίζεται εγγενώς, καθώς τα σενάρια Unity είναι γραμμένα σε C#. Κατά συνέπεια, για να ενεργοποιήσουμε τον εσωτερικό εγκέφαλο, εκμεταλλευόμαστε τη βιβλιοθήκη TensorFlowSharp που παρέχει συνδέσεις .NET στο TensorFlow. Έτσι, όταν υπάρχει ένα περιβάλλον Unity που περιέχει έναν εσωτερικό εγκέφαλο, η εξαγωγή γίνεται μέσω του TensorFlowSharp. Δεδομένης της εξάρτησης από το TensorFlowSharp, ο εσωτερικός εγκέφαλος χαρακτηρίζεται επί του παρόντος ως πειραματικός.

3.3. ΤΟ ΕΡΓΑΛΕΙΟ ML-AGENTS ΤΗΣ UNITY

Το εργαλείο Unity Machine Learning Agents Toolkit (ML-Agents Toolkit) είναι ένα plugin ανοιχτού κώδικα της Unity που επιτρέπει σε παιχνίδια και προσομοιώσεις να χρησιμεύσουν ως περιβάλλοντα για την εκπαίδευση ευφυών πρακτόρων. Το πακέτο εργαλείων ML-Agents παρέχει όλα τα απαραίτητα εργαλεία για τη χρήση της Unity ως μηχανισμό προσομοίωσης για την εκμάθηση των πολιτικών διαφορετικών αντικειμένων στο περιβάλλον της Unity.

Οι πράκτορες μπορούν να εκπαιδευτούν χρησιμοποιώντας ενισχυτική μάθηση, μάθηση μέσω μίμησης, νευροεξέλιξη (neuroevolution) ή άλλες μεθόδους μάθησης μέσω ενός απλού Python API. Παρέχει επίσης εφαρμογές (βασισμένες στη βιβλιοθήκη TensorFlow) από υπερσύγχρονους αλγόριθμους που επιτρέπουν στους προγραμματιστές παιχνιδιών και τους χομπίστες να εκπαιδεύουν ευφυείς πράκτορες για παιχνίδια 2D, 3D και VR / AR. Αυτοί οι εκπαιδευμένοι πράκτορες μπορούν να χρησιμοποιηθούν για πολλαπλούς σκοπούς, συμπεριλαμβανομένου του ελέγχου της συμπεριφοράς NPC (σε μια ποικιλία ρυθμίσεων όπως multi-agent και adversarial), την αυτοματοποιημένη δοκιμή των παιχνιδιών και την αξιολόγηση των διαφορετικών αποφάσεων σχεδιασμού παιχνιδιών πριν από την διάθεση τους στο κοινό.

Το πακέτο εργαλείων ML-Agents είναι αμοιβαία επωφελές τόσο για τους προγραμματιστές παιχνιδιών όσο και για τους ερευνητές της τεχνητής νοημοσύνης, καθώς παρέχει μια κεντρική πλατφόρμα όπου η πρόοδος στην τεχνητή νοημοσύνη μπορεί να αξιολογηθεί στα πλούσια περιβάλλοντα της Unity και στη συνέχεια να έχουν πρόσβαση στις ευρύτερες κοινότητες ερευνητών και προγραμματιστών παιχνιδιών.

3.3.1. Παράδειγμα λειτουργίας: Εκπαίδευση συμπεριφορών NPC

Για να εξηγήσουμε το υλικό και την ορολογία που θα χρησιμοποιηθεί στη συνέχεια, θα χρησιμοποιήσουμε ένα υποθετικό παράδειγμα σε όλη την έκταση του υποκεφαλαίου. Θα διερευνήσουμε το πρόβλημα της εκπαίδευσης της συμπεριφοράς ενός μη χειριζόμενου από τον παίκτη χαρακτήρα (NPC) σε ένα παιχνίδι. (Το NPC είναι ένας χαρακτήρας παιχνιδιού που δεν ελέγχεται ποτέ από έναν πραγματικό παίκτη και η συμπεριφορά του είναι προκαθορισμένη από τον προγραμματιστή παιχνιδιών.) Πιο συγκεκριμένα, ας υποθέσουμε ότι οικοδομούμε ένα παιχνίδι πολλών παικτών με θέμα τον πόλεμο στο οποίο οι παίκτες ελέγχουν τους στρατιώτες. Σε αυτό το παιχνίδι, έχουμε ένα NPC που λειτουργεί ως γιατρός, βρίσκοντας και αναζωογονώντας τραυματίες παίκτες. Τέλος, ας υποθέσουμε ότι υπάρχουν δύο ομάδες, καθεμιά με πέντε παίκτες και έναν γιατρό NPC.

Η συμπεριφορά ενός γιατρού είναι πολύ περίπλοκη. Πρώτα χρειάζεται να αποφύγει να τραυματιστεί, πράγμα που απαιτεί ανίχνευση πότε κινδυνεύει και μετακίνηση σε ασφαλή τοποθεσία. Δεύτερον, πρέπει να γνωρίζει ποια από τα μέλη της ομάδας του είναι τραυματίες και χρειάζονται βοήθεια. Σε περίπτωση πολλαπλών τραυματισμών, πρέπει να αξιολογήσει το βαθμό τραυματισμού και να αποφασίσει ποιον θα βοηθήσει πρώτα. Τέλος, ένας καλός γιατρός θα επιλέξει να τοποθετήσει τον εαυτό του πάντα σε μια θέση όπου μπορεί να βοηθήσει γρήγορα τα μέλη της ομάδας του. Ο παράγοντας σε όλα αυτά τα χαρακτηριστικά σημαίνει ότι σε κάθε περίπτωση ο γιατρός πρέπει να συνυπολογίσει πολλά χαρακτηριστικά του περιβάλλοντος (π.χ. θέση των μελών της ομάδας, θέση των εχθρών, ποια μέλη της ομάδας τους τραυματίζονται και σε ποιο βαθμό) και στη συνέχεια να αποφασίσει (π.χ. να κρυφτεί από τους εχθρού, να μετακινηθείτε για να βοηθήσετε ένα από τα μέλη του). Δεδομένου του μεγάλου αριθμού ρυθμίσεων του περιβάλλοντος και του μεγάλου αριθμού ενεργειών που μπορεί να κάνει ο γιατρός, ο καθορισμός και η εφαρμογή τέτοιων πολύπλοκων συμπεριφορών με το χέρι είναι δύσκολο και επιρρεπής σε σφάλματα.

Με το εργαλείο ML-Agents, είναι δυνατόν να εκπαιδευτούν τέτοιες περίπλοκες συμπεριφορές πρακτόρων NPC (που ονομάζονται agents) χρησιμοποιώντας μια ποικιλία μεθόδων. Η βασική ιδέα είναι πολύ απλή. Πρέπει να οριστούν τρεις οντότητες σε κάθε στιγμή του παιχνιδιού (που ονομάζεται περιβάλλον):

- **Παρατηρήσεις (Observations)** - τι αντιλαμβάνεται ο γιατρός σχετικά με το περιβάλλον του. Οι παρατηρήσεις μπορούν να είναι αριθμητικές ή / και οπτικές. Οι αριθμητικές παρατηρήσεις μετρούν τις ιδιότητες του περιβάλλοντος από την άποψη του πράκτορα. Για τον γιατρό αυτό θα ήταν χαρακτηριστικά του πεδίου μάχης που είναι ορατά σε αυτόν. Για τα πιο περίπλοκα περιβάλλοντα, ένας πράκτορας θα απαιτήσει αρκετές συνεχείς αριθμητικές παρατηρήσεις. Οι οπτικές παρατηρήσεις (Visual observations), από την άλλη πλευρά, είναι εικόνες που παράγονται από τις κάμερες που συνδέονται με τον πράκτορα και αντιπροσωπεύουν αυτό που ο πράκτορας βλέπει εκείνη τη στιγμή. Είθισται να συγχέονται οι παρατηρήσεις ενός πράκτορα με την κατάσταση του περιβάλλοντος (ή παιχνιδιού). Η κατάσταση του περιβάλλοντος αντιπροσωπεύει πληροφορίες για

ολόκληρη τη σκηνή που περιέχει όλους τους χαρακτήρες του παιχνιδιού. Η παρατήρηση του πράκτορα, ωστόσο, περιέχει μόνο πληροφορίες τις οποίες γνωρίζει ο πράκτορας και είναι συνήθως ένα υποσύνολο της περιβαλλοντικής κατάστασης. Για παράδειγμα, η παρατήρηση του γιατρού δεν μπορεί να περιλαμβάνει πληροφορίες σχετικά με έναν κρυφό εχθρό που δεν γνωρίζει ο γιατρός.

- **Ενέργειες (Actions)** - ποιες ενέργειες μπορεί να κάνει ο γιατρός. Παρόμοια με τις παρατηρήσεις, οι ενέργειες μπορούν είτε να είναι συνεχείς είτε διακριτές ανάλογα με την πολυπλοκότητα του περιβάλλοντος και τους πράκτορες. Στην περίπτωση του γιατρού, αν το περιβάλλον είναι ένας απλός κόσμος πλέγματος όπου μόνο η θέση του έχει σημασία, τότε αρκεί μια ξεχωριστή ενέργεια που λαμβάνει μία από τις τέσσερις αξίες (βόρεια, νότια, ανατολικά, δυτικά). Ωστόσο, εάν το περιβάλλον είναι πιο σύνθετο και ο γιατρός μπορεί να κινηθεί ελεύθερα, τότε είναι πιο κατάλληλο να χρησιμοποιηθούν δύο συνεχείς ενέργειες (μία για κατεύθυνση και άλλη για ταχύτητα).
- **Σήματα ανταμοιβής (Reward signals)** - μια τιμή με βαθμωτό μέγεθος που δείχνει πόσο καλά κάνει ο γιατρός την εργασία του. Σημειώνεται ότι το σήμα ανταμοιβής δεν χρειάζεται να παρέχεται σε κάθε στιγμή, αλλά μόνο όταν ο γιατρός εκτελεί μια ενέργεια που είναι καλή ή κακή. Για παράδειγμα, μπορεί να πάρει μια μεγάλη αρνητική ανταμοιβή αν πεθάνει, μια μέτρια θετική ανταμοιβή όποτε αναβιώνει ένα τραυματισμένο μέλος της ομάδας και μια μέτρια αρνητική ανταμοιβή όταν ένα πληγωμένο μέλος της ομάδας πεθαίνει λόγω έλλειψης βοήθειας. Σημειώνεται επίσης ότι το σήμα ανταμοιβής είναι ο τρόπος με τον οποίο οι στόχοι της εργασίας μεταβιβάζονται στον πράκτορα, έτσι πρέπει να ρυθμιστούν με τέτοιο τρόπο ώστε η μέγιστη ανταμοιβή να δημιουργεί την επιθυμητή βέλτιστη συμπεριφορά.

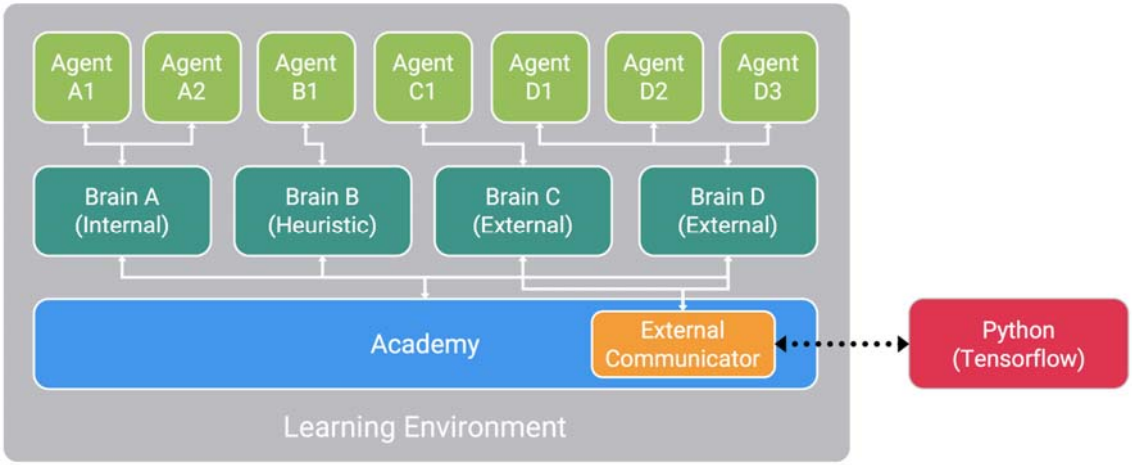
Αφού έχουν οριστεί αυτές οι τρεις οντότητες (τα δομικά στοιχεία ενός έργου ενισχυτικής μάθησης), μπορεί τώρα να εκπαιδευτεί ο γιατρός στην επιθυμητή συμπεριφορά. Αυτό επιτυγχάνεται με την προσομοίωση του περιβάλλοντος για πολλές δοκιμές όπου ο γιατρός, με την πάροδο του χρόνου, μαθαίνει ποια είναι η βέλτιστη δράση για κάθε παρατήρηση μεγιστοποιώντας της μελλοντικής ανταμοιβής του.

Το κλειδί είναι ότι με την εκμάθηση των ενεργειών που μεγιστοποιούν την ανταμοιβή του, ο γιατρός μαθαίνει τις συμπεριφορές που τον καθιστούν καλό γιατρό (δηλαδή αυτός που σώζει τον μεγαλύτερο αριθμό ζωών). Στην ορολογία της ενισχυτικής μάθησης, η συμπεριφορά που μαθαίνεται ονομάζεται πολιτική, η οποία είναι ουσιαστικά μια (βέλτιστη) χαρτογράφηση από παρατηρήσεις σε δράσεις. Σημειώνεται ότι η διαδικασία μάθησης μιας πολιτικής μέσω των προσομοιώσεων που τρέχουν ονομάζεται φάση εκπαίδευσης ενώ παίζοντας το παιχνίδι με ένα NPC που χρησιμοποιεί τη μαθησιακή του πολιτική ονομάζεται φάση συμπερασμάτων.

3.3.2. Βασικά συστατικά

Το πακέτο εργαλείων ML-Agents είναι ένα plugin για την Unity που περιλαμβάνει τρία συστατικά υψηλού επιπέδου:

- **Περιβάλλον εκπαίδευσης (Learning Environment)** - περιέχει τη σκηνή της Unity και όλους τους χαρακτήρες του παιχνιδιού.
- **Python API** - περιέχει όλους τους αλγόριθμους μηχανικής μάθησης που χρησιμοποιούνται για την εκπαίδευση (εκμάθηση συμπεριφοράς ή πολιτικής). Σημειώνεται ότι, αντίθετα από το περιβάλλον εκμάθησης, το Python API δεν αποτελεί μέρος της Unity, αλλά επικοινωνεί με την Unity μέσω εξωτερικού συστήματος διασύνδεσης (Communicator).
- **Εξωτερικός Communicator** - συνδέει το περιβάλλον εκμάθησης με το Python API.



Εικόνα 4 - Περιβάλλον εκμάθησης της Unity

Το περιβάλλον εκμάθησης περιέχει τρία πρόσθετα στοιχεία που βοηθούν στην οργάνωση της σκηνής της Unity:

- **Πράκτορες (Agents)** - συνδέονται με το Unity μέσω ενός GameObject (οποιοσδήποτε χαρακτήρας μέσα σε μια σκηνή) και χειρίζονται τις παρατηρήσεις τους, εκτελώντας τις ενέργειες που δέχονται και αναθέτοντας μια ανταμοιβή (θετική / αρνητική) όταν είναι απαραίτητο. Κάθε πράκτορας συνδέεται με ακριβώς έναν εγκέφαλο.
- **Εγκέφαλοι (Brains)** - ενσωματώνουν τη λογική για τη λήψη αποφάσεων για τον πράκτορα. Στην ουσία, ο εγκέφαλος είναι αυτός που ακολουθεί την πολιτική για κάθε πράκτορα και καθορίζει ποιες ενέργειες πρέπει να κάνει ο πράκτορας σε κάθε περίπτωση. Πιο συγκεκριμένα, είναι το στοιχείο που λαμβάνει τις παρατηρήσεις και τις ανταμοιβές από τον πράκτορα και επιστρέφει μια ενέργεια.
- **Ακαδημία (Academy)** - φροντίζει τη διαδικασία παρατήρησης και λήψης αποφάσεων. Μέσα στην ακαδημία μπορούν να οριστούν διάφορες περιβαλλοντικοί παράμετροι όπως η ποιότητα απόδοσης και η ταχύτητα με την οποία εκτελείται το περιβάλλον.

Κάθε περιβάλλον εκμάθησης πρέπει να έχει πάντα μία κεντρική (global) ακαδημία και έναν πράκτορα για κάθε χαρακτήρα στη σκηνή. Παρόλο που κάθε πράκτορας πρέπει να συνδεθεί με έναν εγκέφαλο, είναι δυνατό για τους πράκτορες που έχουν παρόμοιες παρατηρήσεις και ενέργειες να συνδεθούν με τον ίδιο εγκέφαλο. Στο παράδειγμα του παιχνιδιού μας, έχουμε δύο ομάδες με το δικό τους γιατρό. Έτσι θα έχουμε δύο πράκτορες στο περιβάλλον μάθησης, ένα για κάθε γιατρό, αλλά και οι δύο αυτοί γιατροί μπορούν να συνδεθούν με τον ίδιο εγκέφαλο. Σημειώνεται ότι αυτοί οι δύο γιατροί συνδέονται με τον ίδιο εγκέφαλο επειδή ο χώρος των παρατηρήσεων και των ενεργειών τους είναι παρόμοιος. Αυτό δεν σημαίνει ότι σε κάθε περίπτωση θα έχουν ίδιες παρατηρήσεις και δράσεις. Με άλλα λόγια, ο εγκέφαλος καθορίζει το χώρο όλων των πιθανών παρατηρήσεων και ενεργειών, ενώ οι πράκτορες που συνδέονται με αυτόν (στην περίπτωση αυτή οι γιατροί) μπορούν να έχουν τις δικές τους, μοναδικές παρατηρήσεις και δράσεις. Εάν επεκτείνουμε το παιχνίδι να περιέχει και NPC οι οποίοι θα λειτουργούν σαν οδηγοί τανκς, τότε ο πράκτορας που συνδέεται με αυτούς τους χαρακτήρες δεν μπορεί να μοιραστεί έναν εγκέφαλο με τον πράκτορα που λειτουργεί ως γιατρός διότι οι γιατροί και οι οδηγοί έχουν διαφορετικές ενέργειες.

Στην πράξη, έχουμε τέσσερις διαφορετικούς τύπους εγκεφάλων, οι οποίοι επιτρέπουν ένα ευρύ φάσμα σεναρίων εκπαίδευσης και συμπερασμάτων:

- **Εξωτερικός (External)** - οι αποφάσεις λαμβάνονται χρησιμοποιώντας το Python API. Εδώ οι παρατηρήσεις και οι ανταμοιβές που συλλέγει ο εγκέφαλος διαβιβάζονται στο Python API μέσω του External Communicator. Το Python API επιστρέφει τότε την αντίστοιχη ενέργεια που πρέπει να ληφθεί από τον πράκτορα.
- **Εσωτερικός (Internal)** - οι αποφάσεις λαμβάνονται χρησιμοποιώντας ένα ενσωματωμένο μοντέλο TensorFlow. Το ενσωματωμένο μοντέλο TensorFlow

αντιπροσωπεύει μια γνωστή πολιτική και ο εγκέφαλος χρησιμοποιεί αυτό το μοντέλο απευθείας για να καθορίσει τη δράση για κάθε πράκτορα.

- **Παίκτης (Player)** - οι αποφάσεις λαμβάνονται χρησιμοποιώντας πραγματική είσοδο από το πληκτρολόγιο. Εδώ, ένας πραγματικός παίκτης ελέγχει τον πράκτορα και οι παρατηρήσεις και οι ανταμοιβές που συλλέγονται από τον εγκέφαλο δεν χρησιμοποιούνται για τον έλεγχο του πράκτορα.
- **Ευρετικός (Heuristic)** - οι αποφάσεις λαμβάνονται χρησιμοποιώντας κωδικοποιημένη συμπεριφορά. Αυτό μοιάζει με τον τρόπο με τον οποίο οι περισσότερες συμπεριφορές χαρακτήρων ορίζονται αυτή τη στιγμή και μπορούν να βοηθήσουν στη σύγκριση του τρόπου με τον οποίο ένας πράκτορας με κωδικοποιημένη συμπεριφορά λαμβάνει αποφάσεις σε σχέση με έναν πράκτορα του οποίου η συμπεριφορά έχει εκπαιδευτεί. Στο παράδειγμά μας, αφού έχει εκπαιδευτεί ένας εγκέφαλος για τους γιατρούς, μπορούμε να αναθέσουμε στον ένα γιατρό της μίας ομάδας τον εκπαιδευμένο εγκέφαλο και να αναθέσουμε στον γιατρό της άλλης ομάδας έναν εγκέφαλο με κωδικοποιημένη συμπεριφορά. Στη συνέχεια, μπορούμε να αξιολογήσουμε ποιος γιατρός είναι πιο αποτελεσματικός.

Όπως περιγράφεται επί του παρόντος, μπορεί να φαίνεται ότι ο εξωτερικός Communicator και το Python API αξιοποιούνται μόνο από τον εξωτερικό εγκέφαλο. Στην πραγματικότητα είναι δυνατόν να ρυθμιστούν κατάλληλα οι παράμετροι και των υπολοίπων τύπων εγκεφάλου ώστε να στέλνουν παρατηρήσεις, ανταμοιβές και ενέργειες στο Python API μέσω του External Communicator (μια λειτουργία που ονομάζεται broadcasting). Όπως θα δούμε σύντομα, αυτό επιτρέπει επιπλέον τρόπους εκπαίδευσης.

3.3.3. Τρόποι Εκπαίδευσης

Δεδομένης της ευελιξίας του εργαλείου ML-Agents, υπάρχουν μερικοί τρόποι με τους οποίους μπορεί να πραγματοποιηθεί η εκπαίδευση και τα συμπεράσματα.

3.3.3.1. Ενσωματωμένη εκπαίδευση και συμπεράσματα

Όπως αναφέρθηκε προηγουμένως, το πακέτο εργαλείων ML-Agents συνοδεύεται από πολλές υλοποιήσεις αλγορίθμων τελευταίας τεχνολογίας για την εκπαίδευση ευφυών πρακτόρων. Σε αυτό το παράδειγμα, ο τύπος εγκεφάλου έχει οριστεί σε εξωτερικό κατά τη διάρκεια της εκπαίδευσης και εσωτερικό κατά τη διάρκεια της εξαγωγής. Ειδικότερα, κατά τη διάρκεια της εκπαίδευσης, όλοι οι γιατροί στη σκηνή στέλνουν τις παρατηρήσεις τους στο Python API μέσω του External Communicator. Το Python API επεξεργάζεται αυτές τις παρατηρήσεις και στέλνει πίσω τις ενέργειες για κάθε γιατρό. Κατά τη διάρκεια της εκπαίδευσης αυτές οι ενέργειες είναι κυρίως διερευνητικές για να βοηθήσουν το Python API να μάθει την καλύτερη πολιτική για κάθε γιατρό. Μόλις τελειώσει η εκπαίδευση, η εξαγόμενη πολιτική για κάθε γιατρό μπορεί να εξαχθεί.

Δεδομένου ότι όλες οι υλοποιήσεις στο εργαλείο ML-Agents βασίζονται στο TensorFlow, η γνωστή πολιτική είναι απλώς ένα αρχείο μοντέλου TensorFlow. Στη συνέχεια, κατά τη διάρκεια της φάσης των συμπερασμάτων, αλλάζεται ο τύπος του εγκεφάλου σε εσωτερικό και συμπεριλαμβάνεται το μοντέλο TensorFlow που παράγεται από τη φάση της εκπαίδευσης. Τώρα κατά τη διάρκεια της φάσης των συμπερασμάτων, οι γιατροί συνεχίζουν να παράγουν τις παρατηρήσεις τους, αλλά αντί να αποστέλλονται στο Python API, θα τροφοδοτούν το εσωτερικό τους (ενσωματωμένο) μοντέλο για να δημιουργήσουν τη βέλτιστη δράση για κάθε γιατρό σε κάθε χρονική στιγμή.

3.3.3.2.Προσαρμοσμένη εκπαίδευση και εξαγωγή

Στην προηγούμενη λειτουργία, ο εξωτερικός τύπος εγκεφάλου χρησιμοποιήθηκε για την εκπαίδευση και για τη δημιουργία ενός μοντέλου TensorFlow που ο εσωτερικός τύπος εγκεφάλου μπορεί να καταλάβει και να χρησιμοποιήσει. Ωστόσο, οποιοσδήποτε χρήστης του πακέτου εργαλείων ML-Agents μπορεί να αξιοποιήσει τους δικούς του αλγόριθμους τόσο για την εκπαίδευση όσο και για την εξαγωγή συμπερασμάτων. Σε αυτή την περίπτωση, ο τύπος του

εγκεφάλου θα είναι ρυθμισμένος στο εξωτερικό τόσο για τις φάσεις κατάρτισης όσο και για τα συμπεράσματα και οι συμπεριφορές όλων των πρακτόρων στη σκηνή θα ελέγχονται εντός της Python.

3.3.3.3. Μάθηση με τεχνικών διαδοχών μαθημάτων (Curriculum Learning)

Αυτή η λειτουργία είναι μια επέκταση της ενσωματωμένης εκπαίδευσης που αναφέρθηκε παραπάνω και είναι ιδιαίτερα χρήσιμη όταν εκπαιδεύονται πολύπλοκες συμπεριφορές για σύνθετα περιβάλλοντα. Η μάθηση με τεχνικών διαδοχών μαθημάτων είναι ένας τρόπος κατάρτισης ενός μοντέλου μηχανικής μάθησης όπου εισάγονται σταδιακά πιο δύσκολες πτυχές ενός προβλήματος με τέτοιο τρόπο ώστε το μοντέλο να δυσκολεύει διαδοχικά. Αυτή η ιδέα έχει δημιουργηθεί εδώ και αρκετά χρόνια και στην ουσία την ίδια τακτική συνήθως χρησιμοποιούν και οι άνθρωποι. Για παράδειγμα στην πρωτοβάθμια εκπαίδευση, υπάρχουν συγκεκριμένα μαθήματα και θέματα τα οποία διδάσκονται στα παιδιά. Η αριθμητική διδάσκεται πριν από την άλγεβρα, για παράδειγμα. Ομοίως, η άλγεβρα διδάσκεται πριν από τα πιο σύνθετα μαθηματικά προβλήματα. Οι δεξιότητες και οι γνώσεις που αποκτήθηκαν σε προηγούμενα μαθήματα παρέχουν ένα υπόβαθρο για μεταγενέστερα μαθήματα. Η ίδια αρχή μπορεί να εφαρμοστεί στη μηχανική μάθηση, όπου η εκπαίδευση σε ευκολότερα προβλήματα μπορεί να προσφέρει ένα υπόβαθρο για δυσκολότερα προβλήματα στο μέλλον.

Όταν σκεφτόμαστε πώς λειτουργεί η ενισχυτική μάθηση, το κριτήριο εκπαίδευσης είναι η αμοιβή που λαμβάνεται περιστασιακά καθ 'όλη τη διάρκεια της εκπαίδευσης. Το σημείο εκκίνησης κατά την εκπαίδευση ενός πράκτορα για την εκπλήρωση αυτού του καθήκοντος θα είναι μια τυχαία πολιτική. Αυτή η πολιτική εκκίνησης θα έχει τον πράκτορα που λειτουργεί σε κύκλους και πιθανότατα ποτέ, ή πολύ σπάνια, δεν θα επιτύχει την ανταμοιβή για σύνθετα περιβάλλοντα.

Έτσι, με την απλούστευση του περιβάλλοντος κατά την έναρξη της εκπαίδευσης, επιτρέπουμε στον πράκτορα να ενημερώνει γρήγορα την τυχαία πολιτική σε μια πιο ουσιαστική, η οποία βελτιώνεται διαδοχικά καθώς το περιβάλλον αυξάνεται βαθμιαία σε πολυπλοκότητα. Στο παράδειγμά μας, μπορούμε να φανταστούμε την πρώτη εκπαίδευση του γιατρού όταν κάθε ομάδα περιέχει μόνο έναν παίκτη, και στη συνέχεια αυξάνοντας κατ 'επανάληψη τον αριθμό των παικτών (δηλαδή την πολυπλοκότητα του περιβάλλοντος). Το πακέτο εργαλείων ML-Agents υποστηρίζει τη ρύθμιση παραμέτρων προσαρμοσμένου περιβάλλοντος μέσα στην ακαδημία. Αυτό επιτρέπει τη δυναμική προσαρμογή στοιχείων του περιβάλλοντος που σχετίζονται με τη δυσκολία ή την πολυπλοκότητα με βάση την πρόοδο της εκπαίδευσης.

3.3.3.4. Μάθηση μέσω μίμησης (Imitation Learning)

Είναι συχνά πιο σκόπιμο να δείξουμε απλώς τη συμπεριφορά που θέλουμε ένας πράκτορας να εκτελέσει, παρά να επιχειρήσουμε να του τη μάθουμε μέσω μεθόδων δοκιμής και σφάλματος. Για παράδειγμα, αντί να εκπαιδευτεί ο γιατρός με τη ρύθμιση της ανταμοιβής, η μάθηση με μίμηση επιτρέπει την παροχή πραγματικών παραδειγμάτων από έναν ελεγκτή παιχνιδιών σχετικά με τον τρόπο συμπεριφοράς του γιατρού. Πιο συγκεκριμένα, σε αυτόν τον τρόπο, ο τύπος εγκεφάλου κατά τη διάρκεια της προπόνησης θα έχει οριστεί ως Player και όλες οι ενέργειες που εκτελούνται με τον ελεγκτή (εκτός από τις παρατηρήσεις του πράκτορα) θα καταγράφονται και θα αποστέλλονται στο Python API. Ο αλγόριθμος μάθησης με μίμηση θα χρησιμοποιήσει έπειτα αυτά τα ζεύγη παρατηρήσεων και ενεργειών από τον ανθρώπινο παίκτη για να μάθει μια πολιτική στον πράκτορα.

3.3.4. Ευέλικτα σενάρια εκπαίδευσης

Πέρα από τον παραδοσιακό τρόπο εκπαίδευσης που αναφέρθηκε έως τώρα, δηλαδή την σύνδεση ενός εγκεφάλου με έναν πράκτορα και ένα σήμα ανταμοιβής, το εργαλείο ML-Agents προσφέρει την δυνατότητα διαφόρων σεναρίων εκπαίδευσης. Παρακάτω είναι μερικά παραδείγματα :

- Πολλαπλοί ανεξάρτητοι πράκτορες - Πολλαπλοί ανεξάρτητοι πράκτορες με ανεξάρτητα σήματα ανταμοιβής που συνδέονται με έναν μόνο εγκέφαλο. Μια παραλληλοποιημένη

έκδοση του παραδοσιακού σεναρίου εκπαίδευσης, που μπορεί να επιταχύνει και να σταθεροποιήσει τη διαδικασία εκπαίδευσης. Χρήσιμη όταν υπάρχουν πολλές εκδόσεις του ίδιου χαρακτήρα σε ένα περιβάλλον που θα πρέπει να μάθει παρόμοιες συμπεριφορές.

- Ανταγωνιστικοί πράκτορες - Δύο πράκτορες που αλληλεπιδρουν με σήματα αντίστροφης ανταμοιβής και που συνδέονται με ένα μόνο εγκέφαλο. Σε παιχνίδια δύο παικτών, η αυτο-εκπαίδευση με αντιπάλους μπορεί να επιτρέψει σε έναν πράκτορα να γίνει όλο και πιο εξειδικευμένος, έχοντας πάντοτε τον ιδανικό αντίπαλο: τον εαυτό του. Αυτή ήταν η στρατηγική που χρησιμοποιήθηκε κατά την εκπαίδευση του AlphaGo, και πιο πρόσφατα χρησιμοποιήθηκε από το OpenAI.
- Πράκτορες συνεργασίας - Πολλαπλοί πράκτορες που αλληλεπιδρούν με κοινό σήμα επιβράβευσης που συνδέεται είτε με ένα μόνο είτε με πολλαπλούς διαφορετικούς εγκεφάλους. Σε αυτό το σενάριο, όλοι οι πράκτορες πρέπει να συνεργαστούν για να ολοκληρώσουν μια εργασία που δεν θα μπορούσε να ολοκληρωθεί από έναν μόνο πράκτορα. Παραδείγματα περιλαμβάνουν περιβάλλοντα όπου κάθε πράκτορας έχει μόνο πρόσβαση σε μερικές πληροφορίες, τις οποίες πρέπει να μοιραστεί για να ολοκληρώσουν την εργασία ή να λυθεί ένα παζλ.
- Πολλαπλοί ανταγωνιστικοί πράκτορες - Πολλαπλοί πράκτορες που συνδέονται είτε με ένα είτε με πολλαπλούς διαφορετικούς εγκεφάλους και που αλληλεπιδρούν και διαθέτουν αντίστροφα σήματα ανταμοιβής. Σε αυτό το σενάριο, πρέπει να ανταγωνίζονται μεταξύ τους είτε να κερδίσουν έναν διαγωνισμό είτε να αποκτήσουν κάτι το οποίο θα τους βοηθήσει να κερδίσουν τον αντίπαλο τους. Όλα τα ομαδικά αθλήματα εμπίπτουν σε αυτό το σενάριο.
- Οικοσύστημα - Πολλαπλοί πράκτορες με ανεξάρτητα σήματα ανταμοιβής που συνδέονται είτε με έναν είτε με πολλαπλούς διαφορετικούς εγκεφάλους. Αυτό το σενάριο μπορεί να θεωρηθεί ότι δημιουργεί έναν μικρό κόσμο στον οποίο αλληλεπιδρούν ζώα με διαφορετικούς στόχους, όπως μια σαβάνα στην οποία μπορεί να υπάρχουν ζέβρες, ελέφαντες και καμηλοπαρδάλεις ή μια αυτόνομη προσομοίωση οδήγησης σε ένα αστικό περιβάλλον.

3.3.5. Επιπρόσθετα χαρακτηριστικά

Εκτός από τα ευέλικτα σενάρια κατάρτισης που διατίθενται, το πακέτο εργαλείων ML-Agents περιλαμβάνει πρόσθετα χαρακτηριστικά που βελτιώνουν την ευελιξία και την ερμηνεία της διαδικασίας εκπαίδευσης.

- Λήψη αποφάσεων κατ' απαίτηση (on-demand decision making) - Με το ML-Agents είναι δυνατό να ρυθμιστούν οι πράκτορες για να ζητούν αποφάσεις μόνο όταν χρειάζεται αντί να ζητούν αποφάσεις σε κάθε βήμα του περιβάλλοντος. Αυτό επιτρέπει την εκπαίδευση σε παιχνίδια στρατηγικής, μη πραγματικού χρόνου (turn-based strategy), παιχνίδια όπου οι πράκτορες πρέπει να αντιδρούν σε γεγονότα ή παιχνίδια όπου οι πράκτορες μπορούν να αναλάβουν δράσεις μεταβλητής διάρκειας.
- Ενίσχυση μνήμης - Σε μερικά σενάρια, οι πράκτορες πρέπει να μάθουν να θυμούνται το παρελθόν για να είναι ικανοί να λάβουν την καλύτερη απόφαση. Όταν ένας πράκτορας έχει περιορισμένη παρατηρητικότητα του περιβάλλοντος, η παρακολούθηση των παρελθουσών παρατηρήσεων μπορεί να βοηθήσει τον πράκτορα να μάθει. Συγκεκριμένα επιτρέπει στον πράκτορα να αποθηκεύει μνήμες που θα χρησιμοποιηθούν σε μελλοντικά βήματα.
- Παρακολούθηση λήψης αποφάσεων του πράκτορα - Η Unity παρέχει μια κλάση παρακολούθησης την λεγόμενη Agent Monitor, η οποία μπορεί να προσφέρει πληροφορίες στο περιβάλλον της Unity για τον εκπαιδευμένο πράκτορα, όπως για παράδειγμα η αντίληψη του πράκτορα σχετικά με το πόσο καλά τα πάει. Με την αξιοποίηση της Unity ως εργαλείο απεικόνισης και την παροχή αυτών των αποτελεσμάτων σε πραγματικό χρόνο, οι ερευνητές και οι προγραμματιστές μπορούν να εντοπίσουν λάθη στην εκπαίδευση και να διορθώνουν την συμπεριφορά ενός πράκτορα.

- Περίπλοκες οπτικές παρατηρήσεις - Σε αντίθεση με άλλες πλατφόρμες, όπου η παρατήρηση του πράκτορα μπορεί να περιορίζεται σε ένα μόνο φορέα ή εικόνα, το εργαλείο ML-Agents επιτρέπει τη χρήση πολλαπλών καμερών για παρατηρήσεις ανά πράκτορα. Αυτό επιτρέπει στους πράκτορες να μάθουν να ενσωματώνουν πληροφορίες από πολλαπλές οπτικές ροές. Αυτό μπορεί να είναι χρήσιμο σε διάφορα σενάρια, όπως εκπαίδευση ενός αυτο-οδηγούμενου αυτοκινήτου το οποίο απαιτεί πολλαπλές κάμερες με διαφορετικές οπτικές γωνίες.
- Μετάδοση (Broadcasting) - Όπως αναφέρθηκε παραπάνω, ένας εξωτερικός εγκέφαλος στέλνει τις παρατηρήσεις για όλους τους πράκτορες του στο Python API. Το broadcasting είναι ένα χαρακτηριστικό που μπορεί να ενεργοποιηθεί για τις άλλες τρεις λειτουργίες (Player, Internal, Heuristic) όπου οι παρατηρήσεις και οι πράξεις των πρακτόρων στέλνονται επίσης στο Python API (παρά το γεγονός ότι ο πράκτορας δεν ελέγχεται από το Python API).

4. Εγχειρίδιο εγκατάστασης και χρήσης Unity3D & ML-Agents

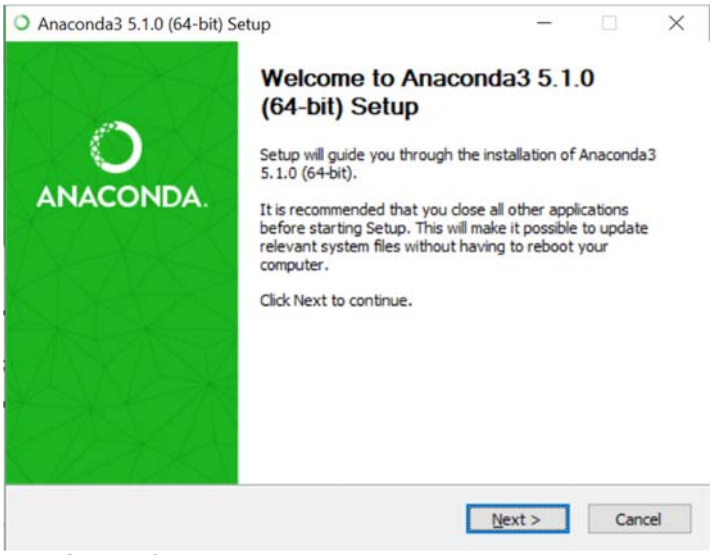
Η ενότητα αυτή λειτουργεί ως οδηγός εγκατάστασης και χρήσης της βιβλιοθήκης ML-Agents της Unity συμπεριλαμβανομένου του τρόπου ρύθμισης του υπολογιστή και του περιβάλλοντος Unity. Πέρα από τις οδηγίες εγκατάστασης του εργαλείου περιγράφεται και ένα παράδειγμα εκπαίδευσης και χρήσης ενός μοντέλου.

4.1. ΕΓΚΑΤΆΣΤΑΣΗ ΤΟΥ ΕΡΓΑΛΕΊΟΥ ML-AGENTS ΓΙΑ WINDOWS

Η εγκατάσταση του εργαλείου ML-Agents έγινε σε λειτουργικό σύστημα Windows 10 στο οποίο έχει δοκιμαστεί και υποστηρίζεται. Για να χρησιμοποιηθεί το πακέτο εργαλείων, πρέπει να εγκατασταθεί η γλώσσα προγραμματισμού Python και τα απαιτούμενα πακέτα της όπως περιγράφονται παρακάτω. Αυτός ο οδηγός καλύπτει επίσης τον τρόπο ρύθμισης εκπαίδευσης κάνοντας χρήση την κάρτα γραφικών το οποίο είναι αρκετά χρήσιμο για προχωρημένους χρήστες. Η εκπαίδευση μέσω κάρτα γραφικών δεν είναι απαιτούμενο για την έκδοση v0.4 του εργαλείου. Ωστόσο, η εκπαίδευση σε μια κάρτα γραφικών μπορεί να απαιτείται σε μελλοντικές εκδόσεις και λειτουργίες.

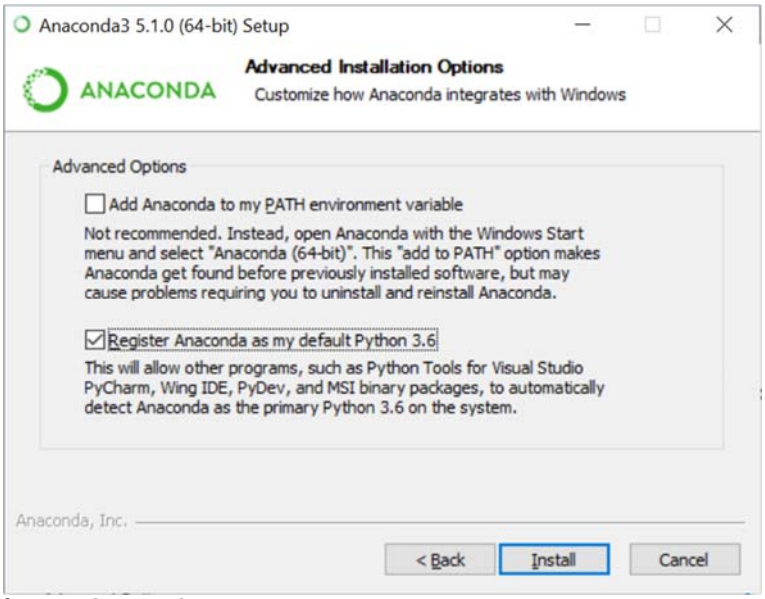
4.1.1. Εγκατάσταση Python μέσω Anaconda

Κατεβάζουμε και εγκαθιστούμε το Anaconda για Windows. Με τη χρήση του Anaconda, δίνεται η δυνατότητα ο χρήστης να διαχειριστεί χωριστά περιβάλλοντα για διαφορετικές εκδόσεις της Python. Απαιτείται η Python 3.5 ή 3.6 καθώς δεν υποστηρίζεται πλέον η Python 2. Σε αυτόν τον οδηγό χρησιμοποιήθηκαν Python 3.6 και Anaconda 5.1.



Εικόνα 5 - Εγκατάσταση Anaconda

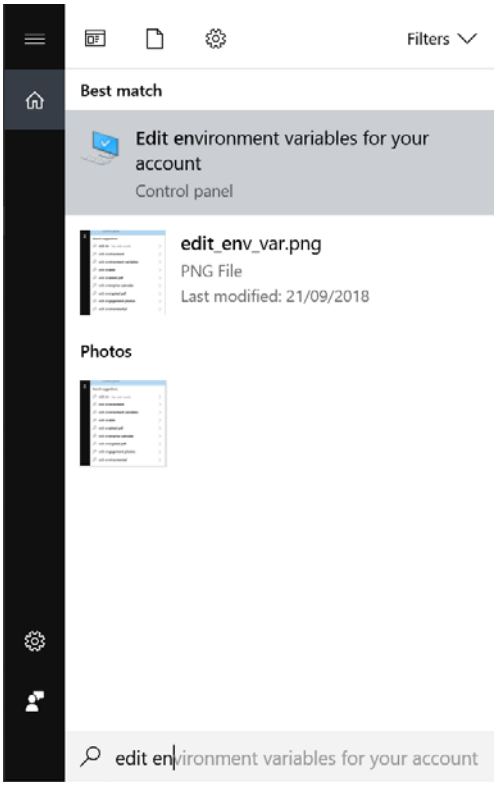
Προτείνεται να αφήσουμε τις επιλογές στις προκαθορισμένες.



Εικόνα 6 - Εγκατάσταση Anaconda

Αφού ολοκληρωθεί η εγκατάσταση, ανοίγουμε την εφαρμογή Anaconda για να ολοκληρωθεί η ρύθμιση των παραμέτρων του συστήματος. Από την γραμμή αναζήτησης των Windows πληκτρολογούμε anaconda navigator. Εφόσον δεν εμφανιστούν κάποια μηνύματα σφάλματος μπορούμε να κλείσουμε την εφαρμογή Anaconda navigator.

Στην συνέχεια πρέπει να καταχωρήσουμε τις παραμέτρους του συστήματος αν δεν έχουν δημιουργηθεί αυτόματα.

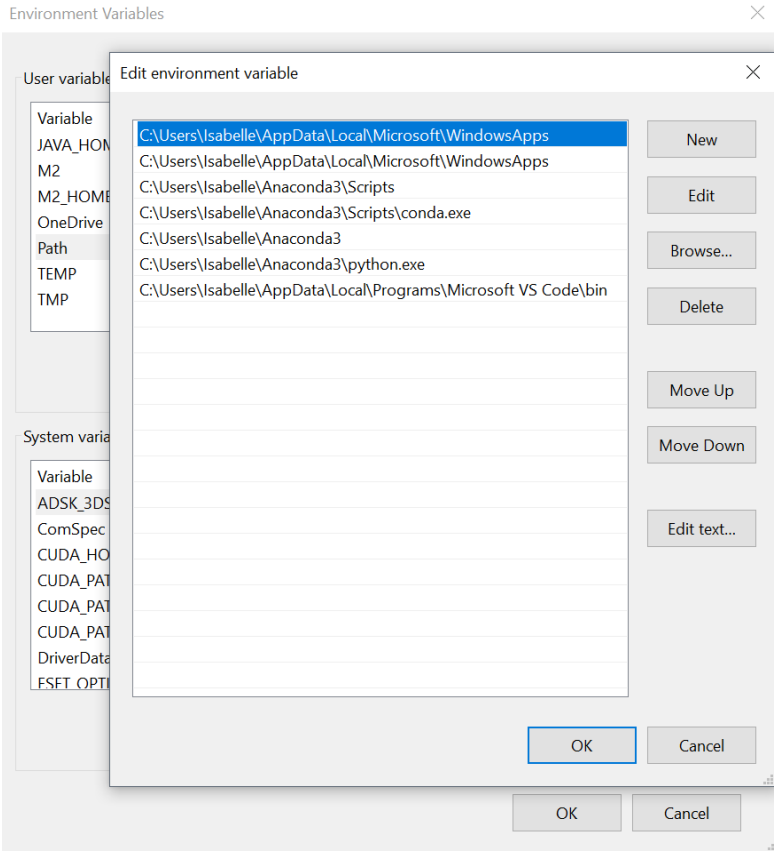


Εικόνα 7 - Καταχωρήσουμε παραμέτρων συστήματος

Από την μπάρα αναζήτησης τον Windows πληκτρολογούμε environment variables, επιλέγουμε το edit environment variable και πατάμε το κουμπί environment variable από το tab advance.

Στο πλαίσιο User variables επιλέγουμε με διπλό κλικ το PATH και προσθέτουμε τα παρακάτω αν δεν υπάρχουν.

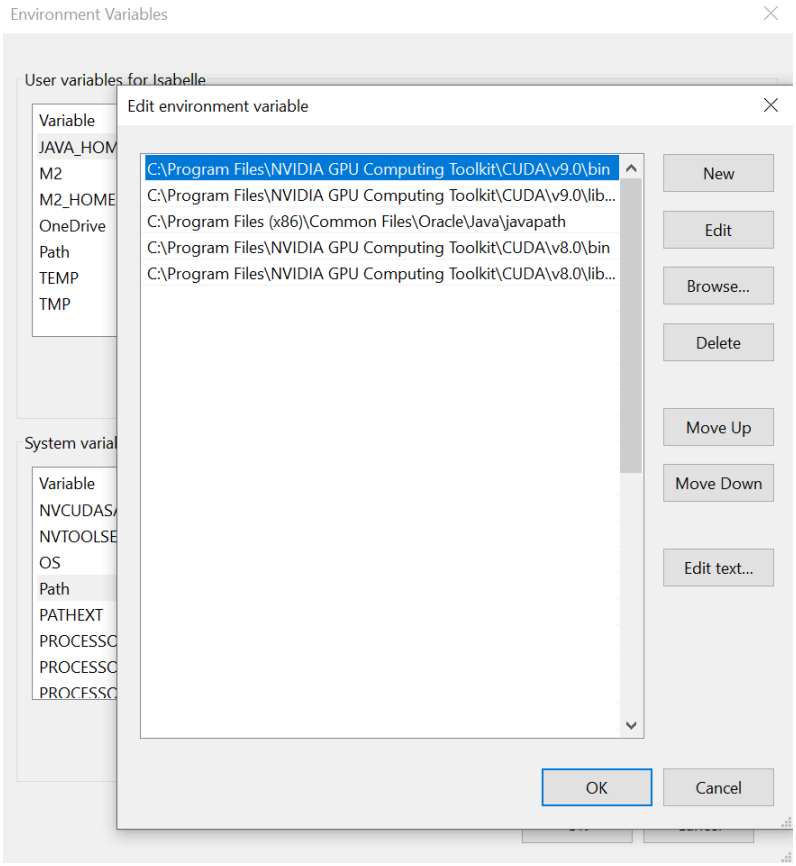
```
%UserProfile%\Anaconda3\Scripts
%UserProfile%\Anaconda3\Scripts\conda.exe
%UserProfile%\Anaconda3
%UserProfile%\Anaconda3\python.exe
```



Εικόνα 8 - Καταχωρήσουμε παραμέτρων συστήματος

Στο πλαίσιο System variables επιλέγουμε με διπλό κλικ το PATH και προσθέτουμε τα παρακάτω αν δεν υπάρχουν.

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\lib\x64
C:\Program Files\NVIDIA GPU Computing
Toolkit\CUDA\v9.0\extras\CUPTI\libx64
```



Εικόνα 9 - Καταχωρήσουμε παραμέτρων συστήματος

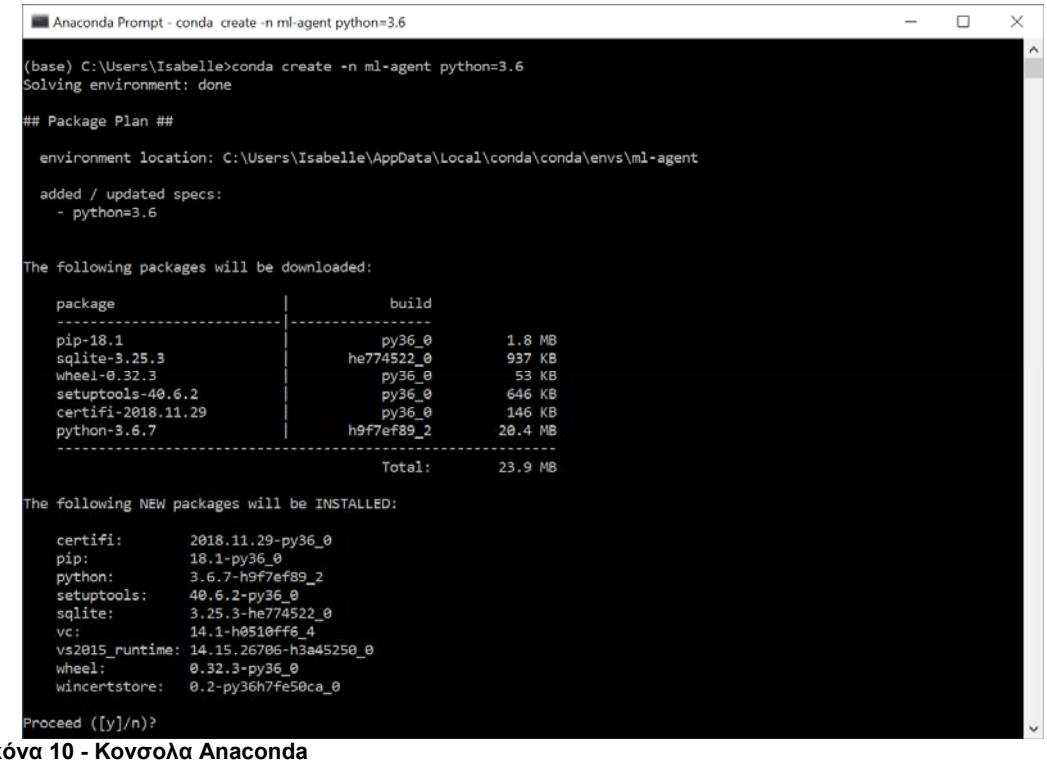
4.1.2. Εγκατάσταση και ενεργοποίηση νέου περιβάλλοντος conda

Στην ενότητα αυτή θα δημιουργηθεί ένα περιβάλλον conda το οποίο θα χρησιμοποιηθεί με το εργαλείο ML-Agents. Πιο συγκεκριμένα, οποιοδήποτε πακέτο εγκατασταθεί θα είναι περιορισμένο στο περιβάλλον αυτό και δεν θα επηρεάσει κάποιο άλλο εγκατεστημένο περιβάλλον conda. Κάθε φορά που θα θέλουμε να τρέξουμε το περιβάλλον αυτό θα πρέπει να το ενεργοποιήσουμε.

Για να δημιουργήσουμε ένα νέο περιβάλλον ανοίγουμε την κονσόλα του anaconda (Anaconda prompt) και πληκτρολογούμε την εντολή:

```
conda create -n ml-agents python=3.6
```

Αν ερωτηθούμε εάν θέλουμε να εγκαταστήσουμε νέα πακέτα πληκτρολογούμε Y και πατάμε enter. Η εγκατάσταση τον πακέτων αυτόν απαιτείται από το εργαλείο. Το νέο περιβάλλον ονομάζεται ml-agents και χρησιμοποιεί την έκδοση 3.6 της Python.



Εικόνα 10 - Κονσολα Anaconda

Για να κάνουμε χρήση του περιβάλλοντος πρέπει πρώτα να το ενεργοποιήσουμε. Το ίδιο πράγμα πρέπει να γίνεται κάθε φορά που θέλουμε να χρησιμοποιήσουμε το περιβάλλον αυτό. Η ενεργοποίηση γίνεται με την παρακάτω εντολή:

```
activate ml-agents
```

Θα πρέπει να φαίνεται ότι έχει προστεθεί το λεκτικό ml-agents στην τελευταία γραμμή.

Στη συνέχεια, πρέπει να εγκατασταθεί το tensorflow. Η εγκατάσταση του πακέτου πραγματοποιείται με την εντολή pip - το οποίο είναι ένα σύστημα διαχείρισης πακέτων που χρησιμοποιείται για την εγκατάσταση πακέτων Python. Οι τελευταίες εκδόσεις του TensorFlow δεν είναι συμβατές με το εργαλείο, οπότε θα πρέπει να εγκατασταθεί την έκδοση 1.7.1. Στο ίδιο παράθυρο κονσόλας, πληκτρολογούμε την ακόλουθη εντολή (βεβαιωθείτε ότι είστε συνδεδεμένοι στο διαδίκτυο):

```
pip install tensorflow == 1.7.1
```

4.1.3. Εγκατάσταση απαιτούμενων πακέτων της Python

Το εργαλείο ML-Agents εξαρτάται από μια σειρά πακέτων της Python τα οποία θα εγκαταστήσουμε με την εντολή `pip`.
Αρχικά θα πρέπει να κατεβάσουμε το project ML-Agents Toolkit από το Github στον υπολογιστή μας. Αυτό μπορεί να γίνει με δυο τρόπου:

- Χρησιμοποιώντας το Git και εκτελώντας την παρακάτω εντολή την κονσόλα Anaconda Prompt:
  `git clone https://github.com/Unity-Technologies/ml-agents.git`
- Κατεβάζοντας απευθείας όλο τα αρχεία από την σελίδα του project στο Github

Αφού έχουμε κατεβάσει τα αρχεία τοπικά στον υπολογιστή μας, από το Anaconda Prompt, μεταβαίνουμε στον φάκελο python μέσα στον φάκελο ml-agents με τη εντολή (στο παράδειγμά μας, τα αρχεία βρίσκονται στην τοποθεσία C: \Downloads):

```
cd C:\Downloads\ml-agents\ml-agents
```

Βεβαιωθείτε ότι είστε συνδεδεμένοι στο διαδίκτυο και στη συνέχεια πληκτρολογήστε τη εντολή:

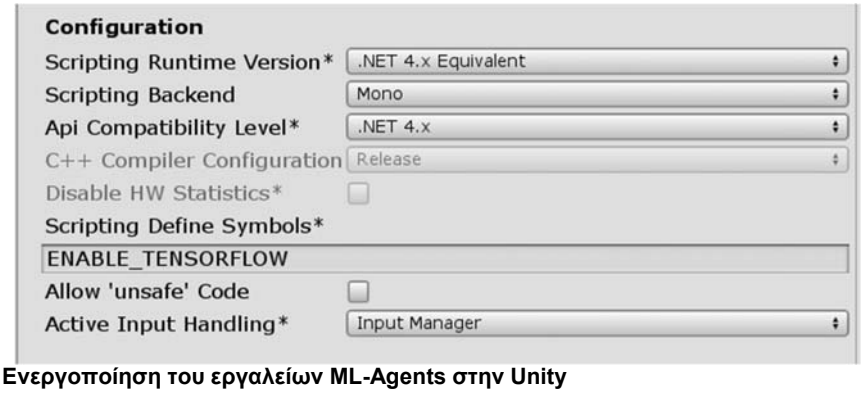
```
pip install
```

Αυτό θα ολοκληρώσει την εγκατάσταση όλων των απαιτούμενων πακέτων Python για να μπορούμε να χρησιμοποιήσουμε το εργαλείων ML-Agents.

4.2. ΕΝΕΡΓΟΠΟΊΗΣΗ ΤΟΥ ΕΡΓΑΛΕΊΩΝ ML-AGENTS ΣΤΗΝ UNITY

Προκειμένου να χρησιμοποιηθεί το εργαλείο ML-Agents στην Unity, πρέπει πρώτα να αλλαχτούν κάποιες ρυθμίσεις στο περιβάλλον της Unity. Για να υπάρχει η δυνατότητα χρήσης προ-εκπαιδευμένων μοντέλων θα πρέπει ενσωματωθεί το plugin TensorFlowSharp στην Unity.

1. Εκκίνηση της Unity
2. Στο παράθυρο "Projects", επιλέξτε την επιλογή "Open" στο επάνω μέρος του παραθύρου.
3. Χρησιμοποιώντας το παράθυρο διαλόγου, εντοπίζουμε τον φάκελο UnitySDK και επιλέγουμε τον ML-Agents και κάνουμε κλικ στο κουμπί "Open" .
4. Απο την μπάρα εργαλείων της Unity επιλέγουμε Edit > Project Settings > Player
5. Για καθεμία από τις πλατφόρμες που θελουμε να στοχεύσουμε (PC, Mac και Linux Standalone, iOS ή Android):
    a. Επιλεγουμε την ενότητα "Other Settings".
    b. Θέτουμε Scripting Runtime Version σε Experimental (.NET 4.6 Equivalent or .NET 4.x Equivalent))
    c. Στα Scripting Defined Symbols, προσθέτουμε ENABLE_TENSORFLOW. Αφού το πληκτρολογήσουμε πατάμε Enter.
6. Επιλέγουμε File > Save Project



Εικόνα 11 - Ενεργοποίηση του εργαλείων ML-Agents στην Unity

Κατεβάζουμε το plugin TensorFlowSharp. Στη συνέχεια το προσθέτουμε στην Unity κάνοντας διπλό κλικ στο ληφθέν αρχείο. Για να ελέγξουμε ότι προστέθηκε με επιτυχία πάμε από το Project κάτω από το στοιχείο Assets > ML-Agents > Plugins > Computer.

4.3. ΠΑΡΆΔΕΙΓΜΑ ΧΡΉΣΗΣ (BALL3D)

Παρακάτω παρουσιάζεται ένα ήδη υλοποιημένο παράδειγμα της Unity που χρησιμοποιεί το πακέτο ML-Agents. Στο εν λόγω παράδειγμα το μοντέλο πρέπει να εκπαιδευτεί να ισορροπεί μπάλες πάνω σε πλατφόρμες.

4.3.1. Κάνοντας χρήση ενός εκπαιδευμένου μοντέλου στο περιβάλλον της Unity

Παρακάτω περιγράφεται η διαδικασία χρήσης ενός εκπαιδευμένου μοντέλου. Για το παράδειγμα αυτό χρησιμοποιήθηκε το μοντέλο 3DBall.

1. Στο παράθυρο Project, μεταβαίνουμε στο φάκελο Assets / ML-Agents / Examples / 3DBall και ανοίγουμε το αρχείο σκηνής 3DBall.
2. Στο παράθυρο Hierarchy, επιλέγουμε το παιδί Ball3DBrain κάτω από το Ball3DAcademy GameObject για να εμφανιστούν οι ιδιότητές του στο παράθυρο Inspector.
3. Στο στοιχείο Brain του Ball3DBrain , αλλάζουμε το Brain Type σε Internal.
4. Στο παράθυρο "Project", εντοπίζουμε το φάκελο "Assets/ML-Agents/Examples/3DBall/TFModels".
5. Προσθέτουμε το αρχείο μοντέλου 3DBall από το φάκελο TFModels στο πεδίο Graph Model του στοιχείου Brain του αντικειμένου Ball3DBrain.
6. Κάνουμε κλικ στο κουμπί Play και θα βλέπουμε ότι οι πλατφόρμες εξισορροπούν τις μπάλες χρησιμοποιώντας το προ ρυθμισμένο μοντέλο.

4.3.2. Εκπαίδευση εξωτερικού εγκεφάλου με ενισχυτική μάθηση

Δεδομένου ότι πρόκειται να χρησιμοποιήσουμε το περιβάλλον για τη διεξαγωγή της εκπαίδευσης θα πρέπει να θέσουμε τον εγκέφαλο που θα χρησιμοποιηθεί από τους πράκτορες σε εξωτερικό. Αυτό επιτρέπει στους πράκτορες να επικοινωνούν με την εξωτερική διαδικασία εκπαίδευσης κατά τη λήψη των αποφάσεών τους.

1. Στο παράθυρο Scene, κάνουμε κλικ στο εικονίδιο τριγώνου δίπλα στο αντικείμενο Ball3DAcademy.
2. Επιλέξτε το παιδικό αντικείμενο Ball3DBrain.
3. Στο παράθυρο "Inspector", ορίζουμε τον Brain Type σε External.

Στη συνέχεια περιγράφεται η εκπαίδευση του περιβάλλοντος:

1. Ανοίγουμε ένα παράθυρο εντολών, κονσόλας ή τερματικού.
2. Ορίζουμε την διαδρομή στο φάκελο όπου έχουμε αποθηκεύσει το εργαλείων ML-Agents.
3. Εκτελούμε την εντολή `mlagents-learn <trainer-config-path> --run-id=<run-identifier> --train` όπου:
    a. `<trainer-config-path>` είναι η σχετική ή απόλυτη διαδρομή του αρχείου της διαμόρφωσης της εκπαιδευτής. Οι προεπιλογές που χρησιμοποιούνται από τα παραδείγματα που περιλαμβάνονται στο MLAgentsSDK υπάρχουν ορισμένα στο αρχειο config / trainer_config.yaml.
    b. `<run-identifier>` είναι μια συμβολοσειρά που χρησιμοποιείται για τον διαχωρισμό των αποτελεσμάτων διαφορετικών δοκιμών εκπαίδευσης.
    c. `-train` είναι το όρισμα που λέει στο εργαλείο mlagents-learn να εκτελέσει εκπαίδευση και όχι εξαγωγή συμπερασμάτων.
4. Εάν έχουμε κλωνοποιήσει το εργαλείο ML-Agents, τότε μπορούμε απλά να τρέξετε την εντολη `mlagents-learn config/trainer_config.yaml --run-id=firstRun --train`
5. Όταν εμφανιστεί στην οθόνη το μήνυμα "Start training by pressing the Play button in the Unity Editor", πατάμε το πλήκτρο ► (Play) στο περιβάλλον της Unity για να ξεκινήσει η εκπαίδευση στο Editor.

Σημείωση: Εάν χρησιμοποιείτε το Anaconda, μην ξεχάσετε να ενεργοποιήσετε πρώτα το περιβάλλον ml-agents.

Για να σταματήσουμε την εκπαίδευση μπορουμε να πατήσετε Ctrl + C και το εκπαιδευμένο μοντέλο θα είναι στο φακελο

models/<*run-identifier*>/editor_<*academy_name*>_<*run-identifier*>.bytes

όπου <*academy_name*> είναι το όνομα του Academy GameObject την τρέχουσα σκηνή. Αυτό το αρχείο αντιστοιχεί στο τελευταίο σημείο ελέγχου του μοντέλου. Τώρα μπορούμε να ενσωματώσουμε αυτό το εκπαιδευμένο μοντέλο στον εγκέφαλο ακολουθώντας τα παρακάτω βήματα, τα οποία είναι παρόμοια με τα βήματα που περιγράφονται παραπάνω.

1. Μετακινούμε το αρχείο μοντέλου στο φάκελο UnitySDK/Assets/ML-Agents/Examples/3DBall/TFModels/.
2. Ανοίγουμε τον Επεξεργαστή Unity και επιλέγουμε τη σκηνή 3DBall όπως περιγράφεται παραπάνω.
3. Επιλέγουμε το αντικείμενο Ball3DBrain από την ιεραρχία σκηνών.
4. Αλλάζουμε το Type of Brain σε Internal.
5. Σέρνουμε το αρχείο <env_name> _ <run-identifier> .bytes από το παράθυρο Project του Editor στο παράθυρο διαλόγου Graph Model στο παράθυρο επιθεώρησης Ball3DBrain.
6. Πατάμε το κουμπί ► (Play) στο επάνω μέρος του Editor

4.4. ΒΈΛΤΙΣΤΕΣ ΠΡΑΚΤΙΚΈΣ ΣΧΕΔΙΑΣΜΟΎ ΠΕΡΙΒΆΛΛΟΝΤΟΣ

- Είναι συχνά χρήσιμο να ξεκινάμε με την πιο απλή μορφή του προβλήματος, για να βεβαιωθούμε ότι ο πράκτορας μπορεί να εκπαιδευτεί σε αυτό. Από εκεί αυξάνουμε την πολυπλοκότητα με την πάροδο του χρόνου. Αυτό μπορεί να γίνει είτε με το χέρι είτε μέσω Curriculum Learning, όπου παρουσιάζεται στον πράκτορα μια σειρά μαθημάτων που αυξάνονται προοδευτικά σε δυσκολία.
- Όπου είναι δυνατόν, είναι συχνά χρήσιμο να διασφαλίζεται ότι μπορεί να ολοκληρωθεί η εργασία χρησιμοποιώντας ως εγκέφαλο τον τύπο Player με τον οποίον θα ελεγχθεί ο πράκτορας.
- Είναι συχνά χρήσιμο να υπάρχουν πολλά αντίγραφα του πράκτορα και να επισυναφθεί ο εγκέφαλος που θα εκπαιδευτεί σε όλους αυτούς τους πράκτορες. Με αυτό τον τρόπο ο εγκέφαλος μπορεί να πάρει περισσότερες πληροφορίες ανατροφοδότησης (feedback) από όλους αυτούς τους πράκτορες, γεγονός που βοηθάει στην εκπαίδευση πιο γρήγορα.

4.4.1. Ανταμοιβές

- Το μέγεθος οποιασδήποτε ανταμοιβής δεν πρέπει, τυπικά, να είναι μεγαλύτερο από 1,0 προκειμένου να εξασφαλιστεί μια πιο σταθερή μαθησιακή διαδικασία.
- Οι θετικές ανταμοιβές είναι συχνά πιο χρήσιμες στη διαμόρφωση της επιθυμητής συμπεριφοράς ενός πράκτορα από τις αρνητικές ανταμοιβές.
- Για τις εργασίες μετακίνησης χρησιμοποιείται συνήθως μια μικρή θετική ανταμοιβή (+0,1) για την προς τα εμπρός ταχύτητα.
- Εάν επιθυμείτε ο πράκτορας να ολοκληρώσει γρήγορα μια εργασία, είναι συχνά χρήσιμο να δοθεί μια μικρή ποινή σε κάθε βήμα (-0,05) οπου ο πράκτορας δεν έχει ολοκληρώσει την εργασία. Σε αυτή την περίπτωση η ολοκλήρωση της εργασίας θα πρέπει να συμπίπτει με το τέλος του επεισοδίου.
- Οι υπερβολικά μεγάλες αρνητικές ανταμοιβές μπορούν να προκαλέσουν ανεπιθύμητη συμπεριφορά, όταν ένας πράκτορας μαθαίνει να αποφεύγει οποιαδήποτε συμπεριφορά μπορεί να προκαλέσει αρνητική ανταμοιβή, ακόμη και αν πρόκειται για συμπεριφορά που μπορεί τελικά να οδηγήσει σε θετική ανταμοιβή.

4.4.2. Διάνυσμα Παρατηρήσεων (Vector Observations)

- Οι διανυσματικές παρατηρήσεις πρέπει να περιλαμβάνουν όλες τις μεταβλητές που απαιτούνται έτσι ώστε να επιτρέπεται στον πράκτορα να λαμβάνει την βέλτιστη τεκμηριωμένη απόφαση.

- Στις περιπτώσεις όπου οι διανυσματικές παρατηρήσεις πρέπει να διατηρηθούν στη μνήμη ή να συγκριθούν με το χρόνο, αυξάνουμε την τιμή Stacked Vectors για να επιτρέψουμε στον πράκτορα να παρακολουθεί τις πολλαπλές παρατηρήσεις στο παρελθόν.
- Οι μεταβλητές κατηγοριών όπως ο τύπος του αντικειμένου (Sword, Shield, Bow) θα πρέπει να κωδικοποιούνται με τροπο 1 απο ν (δηλ. 3 > 0, 0, 1).
- Εκτός από την κωδικοποίηση μη-αριθμητικών τιμών, όλες οι είσοδοι θα πρέπει να κανονικοποιηθούν ώστε να είναι στην περιοχή 0 έως +1 (ή -1 έως 1). Για παράδειγμα, οι πληροφορίες θέσης x ενός πράκτορα όπου η μέγιστη δυνατή τιμή είναι maxValue θα πρέπει να καταγράφονται ως AddVectorObs (transform.position.x / maxValue) αντί του AddVectorObs (transform.position.x).
- Οι πληροφορίες θέσης των σχετικών GameObjects θα πρέπει να κωδικοποιούνται σε σχετικές συντεταγμένες όποτε είναι δυνατόν. Αυτό συχνά σχετίζεται με τη σχετική θέση του πράκτορα.

4.4.3. Διανύσματα Δράσεις

- Όταν χρησιμοποιείται συνεχής έλεγχος, οι τιμές δράσης πρέπει να περικοπούν σε ένα κατάλληλο εύρος. Το μοντέλο PPO κλιμακώνει αυτόματα αυτές τις τιμές μεταξύ -1 και 1, αλλά τα συστήματα εκπαίδευσης τρίτων ενδέχεται να μην το κάνουν.
- Βεβαιωθείτε ότι έχετε ρυθμίσει το Μέγεθος Διαστήματος του διανύσματος δράσης στον αριθμό των χρησιμοποιούμενων διανυσμάτων δράσης, και όχι μεγαλύτερης διάρκειας, καθώς το τελευταίο μπορεί να επηρεάσει την αποτελεσματικότητα της διαδικασίας εκπαίδευσης.

5. Παράδειγμα εφαρμογής ενισχυτικής μάθησης σε περιβάλλον Unity

Στην ενότητα αυτή περιγράφεται σε βάθος η εκπαίδευση πρακτόρων με ενισχυτική μάθηση σε περιβάλλον Unity με χρήση του εργαλείου ML-Agents, ενώ επιπλέον παρουσιάζονται παραδείγματα εφαρμογής της εν λόγω μορφής μηχανικής μάθησης.

5.1. ΕΝΙΣΧΥΤΙΚΉ ΜΆΘΗΣΗ ΣΤΗΝ UNITY

Η ενισχυτική μάθηση είναι μια μορφή τεχνητής νοημοσύνης που εκπαιδεύει τους πράκτορες να εκτελούν τα καθήκοντά τους ανταμείβοντας την επιθυμητή συμπεριφορά. Κατά την ενισχυτική μάθηση, ένας πράκτορας διερευνά το περιβάλλον του, παρατηρεί την κατάσταση των πραγμάτων και με βάση αυτές τις παρατηρήσεις, αναλαμβάνει δράση. Εάν η ενέργεια οδηγεί σε καλύτερη κατάσταση, ο πράκτορας λαμβάνει θετική ανταμοιβή. Εάν οδηγεί σε μια λιγότερο επιθυμητή κατάσταση, τότε ο πράκτορας δεν λαμβάνει ανταμοιβή ή αρνητική ανταμοιβή (τιμωρία). Καθώς ο πράκτορας μαθαίνει κατά τη διάρκεια της εκπαίδευσης, βελτιστοποιεί τη λήψη αποφάσεων έτσι ώστε να λαμβάνει τη μέγιστη ανταμοιβή με την πάροδο του χρόνου.

Το πακέτο εργαλείων ML-Agents χρησιμοποιεί μια μορφή ενισχυτικής μάθησης που ονομάζεται Proximal Policy Optimization (PPO). Το PPO χρησιμοποιεί ένα νευρωνικό δίκτυο για να προσεγγίσει την ιδανική πολιτική που θα χαρτογραφεί τις παρατηρήσεις ενός πράκτορα στις αντίστοιχες βέλτιστες δυνατές ενέργειες τις οποίες μπορεί ένας πράκτορας να πράξει σε μια δεδομένη κατάσταση. Ο αλγόριθμος PPO του εργαλείου ML-Agents υλοποιείται στο TensorFlow και εκτελείται σε ξεχωριστή διαδικασία Python (επικοινωνώντας με την τρέχουσα εφαρμογή Unity μέσω μιας υποδοχής - socket).

Υπάρχουν μερικές παράμετροι που σχετίζονται με την εκπαίδευση και την κατάλληλη προσαρμογή τόσο την Unity όσο και την πλευρά της εκπαίδευσης μέσω Python. Παρόλα αυτά δεν απαιτείται βαθιά γνώση του ίδιου του αλγορίθμου για να δημιουργηθεί και να εκπαιδευτεί ένας πράκτορας με επιτυχία.

5.1.1. Διαδικασία προσομοίωσης και εκπαίδευσης

Η εκπαίδευση και η προσομοίωση προχωρούν σε διαδοχικά βήματα από την κλάση Academy του εργαλείου ML-Agents. Η κλάση Academy συνεργάζεται με τα αντικείμενα Agent και Brain στη σκηνή για να προχωρήσει στην προσομοίωση. Όταν η Academy έχει φτάσει το μέγιστο καθορισμένο αριθμό βημάτων ή όλοι οι πράκτορες στη σκηνή έχουν ολοκληρώσει τη δράση τους, ένας κύκλος εκπαίδευσης έχει ολοκληρωθεί.

Κατά τη διάρκεια της εκπαίδευσης, η εξωτερική διαδικασία εκπαίδευσης της Python επικοινωνεί με την Academy για να εκτελέσει μια σειρά κύκλων ενώ συλλέγει δεδομένα και βελτιστοποιεί το μοντέλο του νευρωνικού δικτύου. Ο τύπος του εγκεφάλου που δηλώνεται σε έναν πράκτορα καθορίζει εάν συμμετέχει σε εκπαίδευση ή όχι. Ο εξωτερικός εγκέφαλος επικοινωνεί με την εξωτερική διαδικασία για την εκπαίδευση του μοντέλου TensorFlow. Όταν ολοκληρωθεί η εκπαίδευση με επιτυχία, παράγεται ένα εκπαιδευμένο μοντέλο το οποίο μπορεί να προστεθεί στο έργο Unity για χρήση με έναν εσωτερικό εγκέφαλο.

Η κλάση Academy του εργαλείου ML-Agents ενορχηστρώνει τον βρόχο προσομοίωσης του πράκτορα ως εξής:

1. Καλεί τη συνάρτηση AcademyReset() του αντικειμένου Academy.
2. Καλεί τη συνάρτηση AgentReset() για κάθε πράκτορα στην σκηνή.
3. Καλεί τη συνάρτηση CollectObservations() για κάθε πράκτορα στη σκηνή.
4. Χρησιμοποιεί την κατηγορία Brain του κάθε πράκτορα για να αποφασίσει ποια θα είναι η επόμενη ενέργεια του πράκτορα.
5. Καλεί τη συνάρτηση AcademyStep() του αντικειμένου Academy.
6. Καλεί τη συνάρτηση AgentAction() για κάθε πράκτορα στη σκηνή, περνώντας ως όρισμα την ενέργεια που έχει επιλέγει από τον εγκέφαλο του πράκτορα. (Αυτή η λειτουργία δεν καλείται εάν ο πράκτορας έχει ολοκληρώσει την ενέργεια του.)
7. Καλεί τη λειτουργία AgentOnDone() του πράκτορα εάν ο πράκτορας έχει φτάσει τον μέγιστο αριθμό βημάτων του ή έχει επισημανθεί ότι έχει ολοκληρώσει την ενέργεια του. Προαιρετικά, μπορεί να οριστεί ένας πράκτορας ως ολοκληρωμένος εάν έχει ολοκληρώσει την ενέργεια του πριν το τέλος του κύκλου έτσι ώστε να πραγματοποιηθεί επανεκκίνηση του κύκλου εκπαίδευσης. Σε αυτή την περίπτωση, η Ακαδημία καλεί τη λειτουργία AgentReset().
8. Όταν η Academy φτάσει στα μέγιστα βήματα που έχουν ορισθεί στο max step count, ξεκινά ένας νέος κύκλος εκπαίδευσης καλώντας τη συνάρτηση AcademyReset() του αντικειμένου Ακαδημίας.

Για να δημιουργηθεί ένα περιβάλλον εκπαίδευσης, επεκτείνονται τα αντικείμενα της Academy και των Agent για να εφαρμοστούν οι παραπάνω μέθοδοι. Το ελάχιστο που απαιτείται είναι οι συναρτήσεις Agent.CollectObservations() και Agent.AgentAction(). Οι υπόλοιπες συναρτήσεις είναι προαιρετικές και η χρήση τους εξαρτάται από το συγκεκριμένο σενάριο.

Σημείωση: Το API που χρησιμοποιείται από τη διαδικασία εκπαίδευσης Python PPO για να επικοινωνεί και να ελέγχει την Academy κατά τη διάρκεια της εκπαίδευσης μπορεί να χρησιμοποιηθεί και για άλλους σκοπούς. Για παράδειγμα, θα μπορούσε να χρησιμοποιηθεί το API για αλγόριθμους σχεδιασμένους από τον εκάστοτε προγραμματιστή κάνοντας χρήση την Unity ως μηχανή προσομοίωσης.

5.1.2. Οργάνωση της σκηνής στην Unity

Για να υλοποιηθεί μια εκπαίδευση μέσω του πακέτου εργαλείων ML-Agents σε μια σκηνή Unity, η σκηνή θα πρέπει να περιέχει μια υποκλάση της Academy και τα απαιτούμενα αντικείμενα Brain για καθε υποτάξεις Agent. Οποιοσδήποτε πράκτορας της σκηνής πρέπει να είναι συνδεδεμένος με το GameObjects που είναι παιδί της Academy στην ιεραρχία της σκηνής. Οι παράμετροι του πράκτορα πρέπει να επισυνάπτονται στο GameObject που αντιπροσωπεύει τον κάθε πράκτορα.

Πρέπει ένας εγκέφαλος να αντιστοιχεί σε κάθε πράκτορα ή μπορεί ένας εγκέφαλος να μοιράζεται μεταξύ πολλών πρακτόρων. Κάθε πράκτορας θα συλλέγει τις δικές του παρατηρήσεις και θα ενεργεί ανεξάρτητα χρησιμοποιώντας την ίδια πολιτική λήψης αποφάσεων και, για τους εσωτερικούς εγκεφάλους, το ίδιο εκπαιδευμένο μοντέλο TensorFlow.

5.1.3. Ακαδημία (Academy)

Το αντικείμενο της Academy χειρίζεται τους Agents και τις διαδικασίες λήψης αποφάσεων. Κάθε σκηνή μπορεί να εμπεριέχει μόνο ένα αντικείμενο της Academy.

Πρέπει να δημιουργηθεί μια υποκατηγορία της κλάσης Academy αφού η βασική κλάση είναι αφηρημένη. Όταν δημιουργηθεί η υποκλάση της Academy, μπορούν να εφαρμοστούν οι ακόλουθες προαιρετικές μέθοδοι:

- **InitializeAcademy()** - Προετοιμάζει το περιβάλλον την πρώτη φορά που εκκινείται.
- **AcademyReset()** - Ετοιμάζει το περιβάλλον και τους πράκτορες για το επόμενο κύκλο εκπαίδευσης. Με την συνάρτηση αυτή μπορούν να τοποθετηθούν και να αρχικοποιηθούν οντότητες στη σκηνή.
- **AcademyStep()** - Προετοιμάζει το περιβάλλον για το επόμενο βήμα προσομοίωσης. Η κλάση Academy καλεί αυτή τη μέθοδο πριν καλέσει τις μεθόδους του AgentAction() για το τρέχον βήμα. Η μέθοδος αυτή μπορεί να χρησιμοποιηθεί για να ενημερωθούν αντικείμενα στην σκηνή πριν οι πράκτορες λάβουν τις ενέργειες τους. Σημειώνεται ότι οι Πράκτορες έχουν ήδη συλλέξει τις παρατηρήσεις τους και επιλέξει μια ενέργεια πριν η Academy καλέσει αυτή τη μέθοδο. Συνεπώς δεν θα επηρεαστούν από την ενδεχόμενη ενημέρωση που θα πραγματοποιηθεί στην μέθοδο αυτή.

Οι βασικές μέθοδοι της Academy ορίζουν επίσης διάφορες σημαντικές ιδιότητες που μπορούν να οριστούν στον Unity Editor Inspector.

Για την εκπαίδευση, η πιο σημαντική από αυτές τις ιδιότητες είναι το Max Steps, το οποίο καθορίζει πόσο διαρκεί κάθε κύκλος εκπαίδευσης. Μόλις ο μετρητής βημάτων της Academy φτάσει σε αυτή την τιμή, καλείται η συνάρτηση AcademyReset() για να ξεκινήσει ο επόμενος κύκλος εκπαίδευσης.

5.1.4. Εγκέφαλος (Brain)

Ο εγκέφαλος εμπεριέχει τη διαδικασία λήψης αποφάσεων. Τα αντικείμενα του εγκεφάλου πρέπει να είναι παιδιά της Academy στην ιεραρχία της σκηνής στην Unity. Κάθε πράκτορας πρέπει να έχει έναν εγκέφαλο, αλλά μπορεί να χρησιμοποιηθεί ο ίδιος εγκέφαλος σε περισσότερο από έναν πράκτορα.

Η συμπεριφορά του εγκεφάλου καθορίζεται από τον τύπο του εγκεφάλου. Κατά τη διάρκεια της εκπαίδευσης, ορίζεται ο τύπος του εγκεφάλου για κάθε πράκτορα ως εξωτερικός. Για να χρησιμοποιηθεί το εκπαιδευμένο μοντέλο, εισαγάγεται το αρχείο μοντέλου στο έργο Unity και ορίζεται ο τύπος του εγκεφάλου σε εσωτερικό. Οι διάφοροι τύποι εγκεφάλων περιγράφονται παρακάτω. Μπορεί να επεκταθεί η κλάση CoreBrain για να δημιουργηθούν διαφορετικοί τύποι Brain εάν οι τέσσερις ενσωματωμένοι τύποι δεν προσφέρουν αυτό που απαιτείται για την εκάστοτε εκπαίδευση.

Η κατηγορία Brain έχει πολλές σημαντικές ιδιότητες που μπορούν να οριστούν χρησιμοποιώντας το παράθυρο Inspector της Unity. Αυτές οι ιδιότητες πρέπει να είναι κατάλληλες για τους πράκτορες χρησιμοποιώντας τον εγκέφαλο. Για παράδειγμα, η ιδιότητα Vector Observation Space Size πρέπει να ταιριάζει με το μήκος του διανύσματος χαρακτηριστικών που δημιουργείται από έναν πράκτορα.

5.1.5. Πράκτορας (Agent)

Η κατηγορία Agent αντιπροσωπεύει έναν ηθοποιό στη σκηνή που συλλέγει παρατηρήσεις και εκτελεί ενέργειες. Η κατηγορία Agent είναι συνήθως προσαρτημένη στο GameObject στη σκηνή που διαφορετικά αντιπροσωπεύει τον ηθοποιό - για παράδειγμα, σε ένα αντικείμενο παίκτη σε ένα παιχνίδι ποδοσφαίρου ή σε ένα αντικείμενο αυτοκινήτου σε προσομοίωση οχήματος. Σε κάθε πράκτορα πρέπει να ανατεθεί ένας εγκέφαλος.

Για να δημιουργηθεί ένας πράκτορας, επεκτείνεται η κλάση Agent και εφαρμόζονται οι βασικές μέθοδοι CollectObservations () και AgentAction ():

- **CollectObservations()** - Συλλέγει τις παρατηρήσεις του πράκτορα για το περιβάλλον του.

- **AgentAction()** - Εκτελεί τη δράση που επιλέγεται από τον εγκέφαλο του πράκτορα και αναθέτει μια ανταμοιβή στην τρέχουσα κατάσταση.

Οι υλοποιήσεις αυτών των μεθόδων καθορίζουν τον τρόπο με τον οποίο θα πρέπει να οριστούν οι ιδιότητες του Brain που έχουν εκχωρηθεί σε αυτόν τον πράκτορα.

Πρέπει επίσης να καθοριστεί ο τρόπος με τον οποίο ο πράκτορας ολοκληρώνει την ενέργεια του ή ο επιτρεπόμενος χρόνος προσπάθειας. Είναι δυνατόν να οριστεί και χειροκίνητα πότε ο πράκτορας έχει ολοκληρώσει ή έχει αποτύχει στην αποστολή του από την μέθοδο AgentAction(). Επίσης, θα μπορούσε να καταχωρηθεί μια θετική τιμή στην ιδιότητα Max Steps του πράκτορα έτσι ώστε ο πράκτορας να χαρακτηρίσει τον εαυτό του ως ολοκληρωμένο (done) όταν έχει φτάσει την τιμή των βημάτων. Όταν η Academy φτάσει στα δικά της Max Steps, αρχίζει ο επόμενος κύκλος εκπαίδευσης. Εάν επιλεγεί η ιδιότητα ResetOnDone ενός πράκτορα σε true, τότε ο πράκτορας μπορεί να επιχειρήσει την δράση του αρκετές φορές σε ένα κύκλο εκπαίδευσης.

Για να αρχικοποιηθεί ο πράκτορας ξανά και να ξεκινήσει ένας νέος κύκλος εκπαίδευσης χρησιμοποιείται η συνάρτηση Agent.AgentReset().

5.1.6. Περιβάλλοντα

Ένα περιβάλλον στο εργαλείο ML-Agents μπορεί να είναι οποιαδήποτε σκηνή ενσωματωμένη στην Unity. Η σκηνή στην Unity παρέχει το περιβάλλον στο οποίο οι πράκτορες παρατηρούν, ενεργούν και μαθαίνουν.

Ο τρόπος με τον οποίο ρυθμίζεται η σκηνή για να χρησιμεύσει ως μαθησιακό περιβάλλον εξαρτάται αποκλειστικά από τον εκάστοτε στόχο του προγραμματιστή. Μπορεί για παράδειγμα να γίνεται προσπάθεια λύσης ενός συγκεκριμένου προβλήματος ενισχυτικής μάθησης περιορισμένου εύρους, στην περίπτωση αυτή μπορεί να χρησιμοποιηθεί η ίδια σκηνή τόσο για εκπαίδευση όσο και για δοκιμή εκπαιδευμένων πρακτόρων. Σε περιπτώσεις πολύπλοκων παιχνιδιών ή προσομοιώσεων, μπορεί να είναι πιο αποτελεσματικό και πρακτικό να δημιουργηθεί μία σκηνή κατάρτισης με σκοπό τη χρήση.

Οι σκηνές εκπαίδευσης και δοκιμής (ή κανονικού παιχνιδιού) πρέπει να περιέχουν το αντικείμενο της Academy για τον έλεγχο της διαδικασίας λήψης αποφάσεων από τους πράκτορες. Η Academy ορίζει πολλές ιδιότητες που μπορούν να ρυθμιστούν διαφορετικά για μια σκηνή εκπαίδευσης σε σχέση με μια κανονική σκηνή. Οι ιδιότητες Configuration της Academy ελέγχουν την απόδοση και την κλίμακα χρόνου. Είναι δυνατόν για παράδειγμα να επιταχυνθεί ο χρόνος εκπαίδευσης ελαττώνοντας την γραφική απόδοση της Unity ρυθμίζοντας κατάλληλα το Learning Configuration. Επίσης θα πρέπει να ρυθμιστούν κατάλληλα και οι άλλες λειτουργίες της ακαδημίας, όπως για παράδειγμα τα μέγιστα βήματα της ακαδημίας. Πιο συγκεκριμένα, τα μέγιστα βήματα πρέπει να είναι όσο το δυνατόν συντομότερα για την εκπαίδευση - μόνο για να μπορεί ο πράκτορας να εκπληρώσει το καθήκον του, με κάποιο επιπλέον χρόνο για "περιπλανηθεί" ενώ μαθαίνει. Σε κανονικές σκηνές, συχνά δεν είναι επιθυμητό η Academy να επαναφέρει τη σκηνή καθόλου. Σε αυτή την περίπτωση, τα μέγιστα βήματα πρέπει να τεθούν στο μηδέν.

Όταν δημιουργείται ένα περιβάλλον εκπαίδευσης στο Unity, πρέπει να ρυθμίζεται η σκηνή έτσι ώστε να μπορεί να ελεγχθεί από την εξωτερική διαδικασία εκπαίδευσης. Οι προϋποθέσεις περιλαμβάνουν:

- Η σκηνή εκπαίδευσης πρέπει να ξεκινήσει αυτόματα όταν η εφαρμογή Unity ξεκινάει από τη διαδικασία εκπαίδευσης.
- Η σκηνή πρέπει να περιλαμβάνει τουλάχιστον έναν εξωτερικό εγκέφαλο.
- Η Academy πρέπει να επαναφέρει τη σκηνή σε ένα έγκυρο σημείο εκκίνησης για κάθε επεισόδιο εκπαίδευσης.
- Ένας κύκλος εκπαίδευσης πρέπει να έχει ένα ορισμένο τέλος - είτε χρησιμοποιώντας το Max Steps είτε ορίζοντας τον κάθε πράκτορα ως ολοκληρωμένο.

5.2. ΧΡΉΣΗ ΤΟΥ TENSORBOARD ΓΙΑ ΤΗΝ ΠΑΡΑΤΉΡΗΣΗ ΤΗΣ

ΕΚΠΑΊΔΕΥΣΗΣ

Το πακέτο εργαλείων ML-Agents αποθηκεύει στατιστικά στοιχεία κατά τη διάρκεια της συνεδρίας μάθησης που μπορούν να προβληθούν με ένα βοηθητικό πρόγραμμα του TensorFlow που ονομάζεται TensorBoard.

Η εντολή `mlagents-learn` αποθηκεύει τα στατιστικά της κατάρτισης σε ένα φάκελο που ονομάζεται περίληψη, που οργανώνεται από την τιμή `run-id` που αντιστοιχίζεται σε μια εκπαιδευτική συνεδρία.

Για να παρακολουθήσετε τη διαδικασία εκπαίδευσης, είτε κατά τη διάρκεια της κατάρτισης είτε μετά, εκκινήστε το TensorBoard:

1. Ανοίξτε ένα παράθυρο τερματικού ή κονσόλας:
2. Μεταβείτε στον κατάλογο όπου είναι εγκατεστημένο το ML-Agents Toolkit.
3. Από την εκτέλεση της γραμμής εντολών: `tensorboard --logdir=summaries`
4. Ανοίξτε ένα παράθυρο του προγράμματος περιήγησης και μεταβείτε στο *localhost: 6006.*

Σημείωση: Εάν δεν εκχωρήσετε ένα αναγνωριστικό `run-id`, το `mlagents-learn` χρησιμοποιεί την προεπιλεγμένη συμβολοσειρά "ppo". Όλα τα στατιστικά στοιχεία θα αποθηκευτούν στον ίδιο υπο-φάκελο και θα εμφανιστούν ως μία συνεδρία στο TensorBoard. Μετά από λίγες διαδρομές, οι οθόνες μπορεί να είναι δύσκολο να ερμηνευτούν σε αυτή την κατάσταση. Μπορείτε να διαγράψετε τους φακέλους κάτω από τον κατάλογο περιλήψεων για να καταργήσετε τα παλιά στατιστικά στοιχεία.

Στην αριστερή πλευρά του παραθύρου TensorBoard, μπορείτε να επιλέξετε ποια από τις διαδρομές εκπαίδευσης θέλετε να εμφανίσετε. Μπορείτε να επιλέξετε πολλαπλά κριτήρια εκτέλεσης για να συγκρίνετε τα στατιστικά στοιχεία. Το παράθυρο TensorBoard παρέχει επίσης επιλογές για την εμφάνιση και την ομαλοποίηση των γραφημάτων. Όταν εκτελείται το πρόγραμμα εκπαίδευσης, μπορείτε να χρησιμοποιήσετε την επιλογή `-save-freq` για να καθορίσετε πόσο συχνά θα αποθηκεύονται τα στατιστικά στοιχεία.

- **Μάθημα** - Σχεδιάζει την πρόοδο από το μάθημα στο μάθημα. Ενδιαφέρουσα μόνο όταν η εκπαίδευση αφορά προγράμματα σπουδών.
- **Αθροιστική ανταμοιβή** - Η μέση αθροιστική ανταμοιβή επεισοδίου πάνω σε όλους τους παράγοντες. Θα πρέπει να αυξηθεί κατά τη διάρκεια μιας επιτυχούς συνεδρίας κατάρτισης.
- **Εντροπία** - Τυχαιότητα στις αποφάσεις του μοντέλου. Πρέπει να μειώνεται αργά κατά τη διάρκεια μιας επιτυχημένης διαδικασίας κατάρτισης. Εάν μειώνεται πολύ γρήγορα, θα πρέπει να αυξηθεί το υπερπαραμετρικό βήτα.
- **Μήκος επεισοδίου** - Το μέσο μήκος κάθε επεισοδίου στο περιβάλλον για όλους τους παράγοντες.
- Ποσοστό εκμάθησης - Πόσο μεγάλο είναι το βήμα που λαμβάνει ο αλγόριθμος εκπαίδευσης καθώς αναζητά τη βέλτιστη πολιτική. Πρέπει να μειωθεί με την πάροδο του χρόνου.
- **Απώλεια πολιτικής** - Το μέσο μέγεθος της συνάρτησης απώλειας πολιτικής. Αντιστοιχεί στο πόσο αλλάζει η πολιτική (διαδικασία λήψης αποφάσεων). Το μέγεθος αυτού πρέπει να μειωθεί κατά τη διάρκεια μιας επιτυχημένης συνεδρίας κατάρτισης.
- **Εκτίμηση αξίας** - Η μέση εκτίμηση αξίας για όλες τις καταστάσεις που επισκέφθηκε ο πράκτορας. Θα πρέπει να αυξηθεί κατά τη διάρκεια μιας επιτυχούς συνεδρίας κατάρτισης.
- **Αξία απώλειας** - Η μέση απώλεια της ενημέρωσης λειτουργία αξία. Αντιστοιχεί στο βαθμό στον οποίο το μοντέλο είναι σε θέση να προβλέψει την αξία κάθε κατάστασης. Αυτό θα πρέπει να αυξηθεί ενώ ο πράκτορας μαθαίνει και στη συνέχεια να μειώνεται όταν η ανταμοιβή σταθεροποιηθεί.
- (Ειδική περίπτωση) **Εσωτερική ανταμοιβή** - Αυτό αντιστοιχεί στη μέση αθροιστική εγγενή ανταμοιβή που δημιουργείται ανά επεισόδιο.
- (Ειδική περίπτωση) **Απώλεια προς τα εμπρός** - Το μέσο μέγεθος της συνάρτησης απώλειας αντίστροφου μοντέλου. Αντιστοιχεί στο πόσο καλά το μοντέλο είναι σε θέση να προβλέψει τη νέα κωδικοποίηση παρατήρησης.

- (Ειδική περίπτωση) **Αντίστροφη απώλεια** - Το μέσο μέγεθος της λειτουργίας απώλειας πρότυπου μοντέλου. Αντιστοιχεί στο πόσο καλά το μοντέλο είναι σε θέση να προβλέψει τη δράση που έχει γίνει μεταξύ δύο παρατηρήσεων.

5.3. ΕΚΠΑΊΔΕΥΣΗ ΜΕ ΤΗΝ ΒΕΛΤΙΣΤΟΠΟΊΗΣΗ ΒΈΛΤΙΣΤΗΣ ΠΟΛΙΤΙΚΉΣ

Οι ML-Agents χρησιμοποιούν μια τεχνική ενισχυτικής μάθησης που ονομάζεται Proximal Policy Optimization (PPO). Το PPO χρησιμοποιεί ένα νευρωνικό δίκτυο για να προσεγγίσει την ιδανική λειτουργία που χαρτογραφεί τις παρατηρήσεις ενός πράκτορα στην καλύτερη δυνατή ενέργεια ενός πράκτορα μπορεί να λάβει σε μια δεδομένη κατάσταση. Ο αλγόριθμος PPO του εργαλείου ML-Agents υλοποιείται στο TensorFlow και εκτελείται σε ξεχωριστή διαδικασία της Python (επικοινωνώντας με την τρέχουσα εφαρμογή Unity μέσω μιας υποδοχής).

5.3.1. Βέλτιστες πρακτικές κατά την εκπαίδευση με PPO

Η επιτυχημένη εκπαίδευσης ενός μοντέλου ενισχυτικής μάθησης συχνά περιλαμβάνει την προσαρμογή των υπερπαραμέτρων (hyperparameters) της εκπαίδευσης, ειδικά όταν οι προεπιλεγμένες παράμετροι δεν φαίνεται να δίνουν το επίπεδο απόδοσης που θα θέλαμε.

5.3.1.1. gamma

Η παράμετρος gamma αντιστοιχεί στον συντελεστή προεξόφλησης για μελλοντικές ανταμοιβές. Με αυτό μπορεί να εκτιμηθεί πόσο μακριά στο μέλλον ο πράκτορας πρέπει να νοιάζεται για πιθανές ανταμοιβές. Σε περιπτώσεις όπου ο πράκτορας πρέπει να ενεργεί στο παρόν για να προετοιμαστεί για ανταμοιβές στο μακρινό μέλλον, αυτή η τιμή θα πρέπει να είναι μεγάλη. Σε περιπτώσεις όπου οι ανταμοιβές είναι πιο άμεσες, μπορεί να είναι μικρότερες.

Τυπικό εύρος τιμών: 0,8 - 0,995

5.3.1.2. lambda

Η παράμετρος lambda αντιστοιχεί στην παράμετρο λάμδα που χρησιμοποιείται κατά τον υπολογισμό της εκτίμησης γενικευμένου πλεονεκτήματος (GAE). Αυτό μπορεί να θεωρηθεί ως το πόσο ο πράκτορας βασίζεται στην τρέχουσα εκτίμηση αξίας του κατά τον υπολογισμό μιας εκτίμησης επικαιροποιημένης τιμής. Οι χαμηλές τιμές αντιστοιχούν στο ότι δίνεται περισσότερη βαρύτητα στην εκτίμηση της τρέχουσας τιμής (η οποία μπορεί να είναι υψηλή μεροληψία) και οι υψηλές τιμές αντιστοιχούν στο ότι δίνεται περισσότερη βαρύτητα στις πραγματικές ανταμοιβές που εισπράττονται από το περιβάλλον (που μπορεί να είναι μεγάλη διακύμανση). Η παράμετρος παρέχει μια ανταλλαγή μεταξύ των δύο, και η σωστή τιμή μπορεί να οδηγήσει σε μια πιο σταθερή διαδικασία κατάρτισης.

Τυπική περιοχή: 0,9 - 0,95

5.3.1.3. buffer_size

Η παράμετρος buffer_size αντιστοιχεί σε πόσες εμπειρίες (παρατηρήσεις πράκτορα, ενέργειες και ανταμοιβές που αποκτήθηκαν) πρέπει να συλλεχθούν πριν γίνει οποιαδήποτε εκμάθηση ή ενημέρωση του μοντέλου. Αυτό θα πρέπει να είναι πολλαπλάσιο του μεγέθους batch_size. Συνήθως ένα μεγαλύτερο buffer_size αντιστοιχεί σε πιο σταθερές ενημερώσεις εκπαίδευσης.

Τυπική περιοχή: 2048 - 409600

5.3.1.4. batch_size

Η παράμετρος batch_size είναι ο αριθμός των εμπειριών που χρησιμοποιούνται για μια επανάληψη μιας ενημέρωσης καθόδου κλίμακας. Αυτό θα πρέπει πάντα να είναι ένα κλάσμα του

buffer_size. Αν χρησιμοποιείτε χώρο συνεχούς δράσης, αυτή η τιμή πρέπει να είναι μεγάλη (με τη σειρά 1000s). Αν χρησιμοποιείτε ένα διακριτό χώρο δράσης, αυτή η τιμή πρέπει να είναι μικρότερη (κατά σειρά 10 δευτερολέπτων).

Τυπικό εύρος (συνεχής): 512 - 5120
Τυπική περιοχή (διακριτή): 32 - 512

5.3.1.5. num_epoch

Η παράμετρος num_epoch είναι ο αριθμός των περασμάτων μέσα από την προσωρινή μνήμη εμπειρίας κατά τη διάρκεια της καθόδου κλίμακας. Όσο μεγαλύτερος είναι ο αριθμός batch_size, τόσο μεγαλύτερη είναι η αποδεκτή τιμή αυτής της παραμέτρου. Η μείωση αυτή θα εξασφαλίσει πιο σταθερές ενημερώσεις, με κόστος βραδύτερης μάθησης.

Τυπική περιοχή: 3 - 10

5.3.1.6. learning_rate

Η παράμετρος learning_rate αντιστοιχεί στη βαρύτητα του κάθε βήματος ενημέρωσης καθόδου κλίμακας. Αυτό συνήθως μειώνεται εάν η εκπαίδευση είναι ασταθής και η ανταμοιβή δεν αυξάνεται σταθερά.

Τυπική περιοχή: 1e-5 - 1e-3

5.3.1.7. time_horizon

Η παράμετρος time_horizon αντιστοιχεί στον αριθμό βαθμίδων εμπειρίας για τη συλλογή ανά παράγοντα πριν προστεθεί στο buffer εμπειρίας. Όταν επιτευχθεί αυτό το όριο πριν από το τέλος ενός επεισοδίου, χρησιμοποιείται μια εκτίμηση αξίας για την πρόβλεψη της συνολικής αναμενόμενης ανταμοιβής από την τρέχουσα κατάσταση του παράγοντα. Ως εκ τούτου, αυτή η παράμετρος μεταβάλλεται μεταξύ μιας εκτίμησης με λιγότερο προκατειλημμένη αλλά μεγαλύτερη διακύμανση (μακροπρόθεσμος χρονικός ορίζοντας) και μιας πιο μεροληπτικής αλλά λιγότερο ποικίλης εκτίμησης (βραχυχρόνιος χρονικός ορίζοντας). Σε περιπτώσεις όπου υπάρχει συχνή ανταμοιβή σε ένα επεισόδιο ή τα επεισόδια είναι απαγορευτικά μεγάλα, ένας μικρότερος αριθμός μπορεί να είναι πιο κατάλληλος. Αυτός ο αριθμός πρέπει να είναι αρκετά μεγάλος για να συλλάβει όλη τη σημαντική συμπεριφορά μέσα σε μια ακολουθία των πράξεων ενός πράκτορα.

Τυπική περιοχή: 32 - 2048

5.3.1.8. max_steps

Η παράμετρος max_steps αντιστοιχεί σε πόσες βαθμίδες της προσομοίωσης (πολλαπλασιασμένες με το συντελεστή frame-skip) εκτελούνται κατά τη διάρκεια της διαδικασίας κατάρτισης. Αυτή η τιμή πρέπει να αυξηθεί για πιο περίπλοκα προβλήματα.

Τυπική περιοχή: 5e5 - 1e7

5.3.1.9. beta

Η παράμετρος beta αντιστοιχεί στη βαρύτητα της κανονικοποίησης της εντροπίας, γεγονός που καθιστά την πολιτική "πιο τυχαία". Αυτό εξασφαλίζει ότι οι πράκτορες διερευνούν σωστά το χώρο δράσης κατά τη διάρκεια της εκπαίδευσης. Με την αύξηση αυτή θα διασφαλιστεί η λήψη περισσότερων τυχαίων ενεργειών. Αυτό πρέπει να ρυθμιστεί έτσι ώστε η εντροπία (μετρήσιμη από το TensorBoard) να μειώνεται αργά παράλληλα με την αύξηση της ανταμοιβής. Αν η εντροπία πέσει πολύ γρήγορα, αυξήστε το βήτα. Αν η εντροπία πέσει πολύ αργά, μειώστε την τιμή της παραμέτρου βήτα.

Τυπική περιοχή: 1e-4 - 1e-2

5.3.1.10. epsilon

Η παράμετρος epsilon αντιστοιχεί στο αποδεκτό όριο απόκλισης μεταξύ της παλαιάς και της νέας πολιτικής κατά την ενημέρωση καθόδου κλίμακας. Ο καθορισμός αυτής της μεταβλητής σε μικρές τιμές θα οδηγήσει σε πιο σταθερές ενημερώσεις, αλλά θα επιβραδύνει επίσης τη διαδικασία εκπαίδευσης.

Τυπική περιοχή: 0,1 - 0,3

5.3.1.11. normalization

Η παράμετρος normalization αντιστοιχεί στο εάν η κανονικοποίηση εφαρμόζεται στις εισόδους παρατήρησης διανύσματος. Αυτή η κανονικοποίηση βασίζεται στον μέσο όρο και τη διακύμανση της παρατήρησης του φορέα. Η κανονικοποίηση μπορεί να είναι χρήσιμη σε περιπτώσεις σύνθετων προβλημάτων συνεχούς ελέγχου, αλλά μπορεί να είναι επιβλαβής με απλούστερα διακριτά προβλήματα ελέγχου.

5.3.1.12. num_layers

Η παράμετρος num_layers αντιστοιχεί στον αριθμό κρυφών στρωμάτων μετά την είσοδο παρατήρησης ή μετά την κωδικοποίηση συνελικτικού νευρωνικού δικτύου (CNN) της οπτικής παρατήρησης. Για απλά προβλήματα, λιγότερα στρώματα είναι πιθανό να εκπαιδευτούν γρηγορότερα και πιο αποτελεσματικά. Μπορεί να απαιτούνται περισσότερα στρώματα για πιο περίπλοκα προβλήματα ελέγχου.

Τυπική περιοχή: 1 - 3

5.3.1.13. hidden_units

Η παράμετρος hidden_units αντιστοιχεί στον αριθμό μονάδων σε κάθε πλήρως συνδεδεμένο στρώμα του νευρικού δικτύου. Για απλά προβλήματα όπου η σωστή ενέργεια είναι ένας απλός συνδυασμός των παραμέτρων παρατήρησης, αυτό πρέπει να είναι μικρό. Για προβλήματα όπου η δράση είναι πιο πολύπλοκη αλληλεπίδραση μεταξύ των μεταβλητών παρατήρησης, αυτό θα πρέπει να είναι μεγαλύτερο.

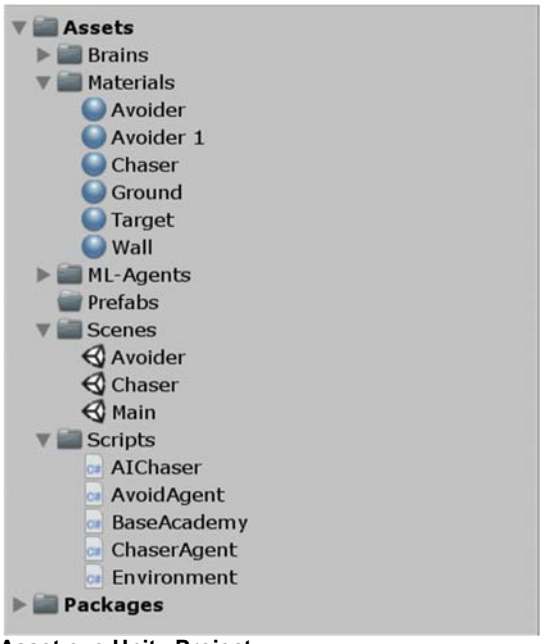
Τυπική περιοχή: 32 - 512

5.4. Ρ΄ΥΘΜΙΣΗ ΤΟΥ ΠΕΡΙΒΆΛΛΟΝΤΟΣ ΤΗΣ UNITY - MONODEVELOP

Σε αυτή την ενότητα περιγράφονται οι ρυθμίσεις του περιβάλλοντος της Unity, οι σχετικές σκηνές και αντικείμενα, καθώς και ο πηγαίος κώδικας C# που σχετίζεται με το παράδειγμα. Επιπλέον, περιγράφεται η διαδικασία εκπαίδευσης με το εργαλείο Anaconda, καθώς και οι ρυθμίσεις του εργαλείου Tensorflow.

5.4.1. Περιγραφή σκηνών και αντικειμένων περιβάλλοντος Unity

Για το παράδειγμα που θα υλοποιηθεί, θα πρέπει αρχικά να οριστούν τα εξής στοιχεία και φάκελοι μέσα στα Assets του Unity project:

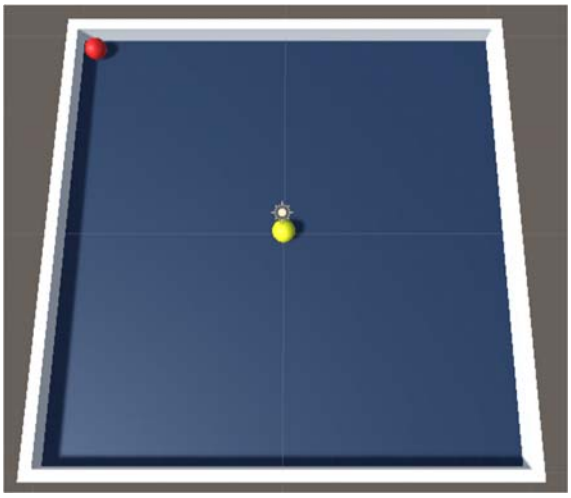


Εικόνα 12 - Δομή φακέλου Asset στο Unity Project

- Ο φάκελος Scenes που περιλαμβάνει τις σκηνές Unity που θα χρησιμοποιηθούν.
- Ο φάκελος Materials που περιλαμβάνει τα materials των αντικειμένων.
- Ο φάκελος Scripts που περιλαμβάνει τα αρχεία πηγαίου κώδικα C# των αντικειμένων.
- Ο φάκελος Brains που θα περιλαμβάνει τα εκπαιδευμένα μοντέλα των εγκεφάλων μετά την εκπαίδευση.
- Ο φάκελος ML-Agents ο οποίος θα πρέπει να έχει αντιγραφεί από το Github repository του εργαλείου.

Συγκεκριμένα, όσον αφορά τις σκηνές του περιβάλλοντος, θα υλοποιηθούν τρεις σκηνές, Avoider, Chaser και Main.

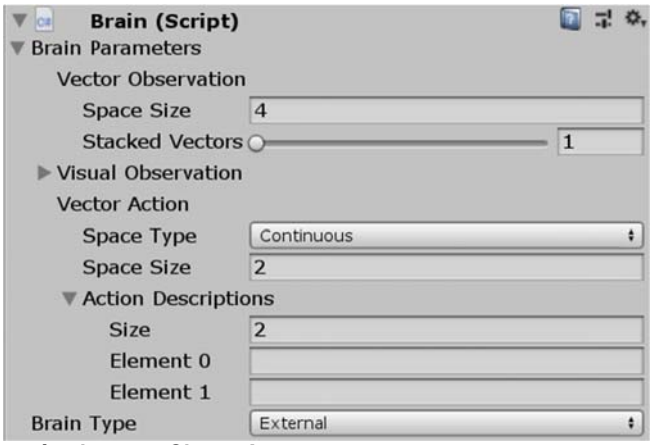
5.4.1.1. Η σκηνή Chaser



Εικόνα 13 - Σκηνή Chaser

Η σκηνή Chaser αφορά την εκπαίδευση του πράκτορα ChaserAgent και αποτελείται από τα εξής μέρη:

- Main Camera, Directional Light και EventSystem - Προεπιλεγμένα αντικείμενα μιας οποιασδήποτε σκηνής Unity.
- Academy - Η Academy του παραδείγματος που θα υλοποιηθεί. Περιέχει τους εγκεφάλους (στη συγκεκριμένη περίπτωση τον εγκέφαλο Chase Brain).
- Chase Brain - Ο εγκέφαλος του πράκτορα ChaserAgent. Θα εκπαιδευτεί μέσω του περιβάλλοντος Anaconda. Σύμφωνα με τον κώδικα που περιγράφεται παρακάτω, ορίζουμε τις παραμέτρους το εγκεφάλου στις εξής:

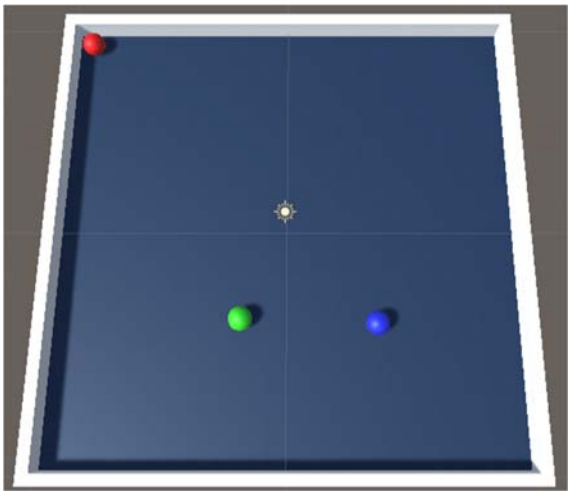


Εικόνα 14 - Ρυθμίσεις εγκέφαλου του ChaserAgent

- Environment - Περιλαμβάνει όλα τα αντικείμενα που σχετίζονται με το περιβάλλον της σκηνής:
    - Ground - Ένα αντικείμενο Unity που χρησιμεύει σαν την επιφάνεια της πίστας στην οποία θα εκπαιδευτεί ο εγκέφαλος του πράκτορα.
    - Walls - Τέσσερα αντικείμενα που χρησιμεύουν ως περίφραξη της πίστας.
    - Target - Το αντικείμενο που κυνηγά ο πράκτορας ChaserAgent. Το συγκεκριμένο αντικείμενο δεν κινείται και αλλάζει θέση όταν ο πράκτορας το αγγίξει.
    - Chaser - Το αντικείμενο που σχετίζεται με τον πράκτορα ChaserAgent. Το αντικείμενο αυτό ελέγχεται από τον κώδικα που περιγράφεται παρακάτω στην κλάση ChaserAgent.

Εικόνα 15 - Δομή σκηνής Chaser

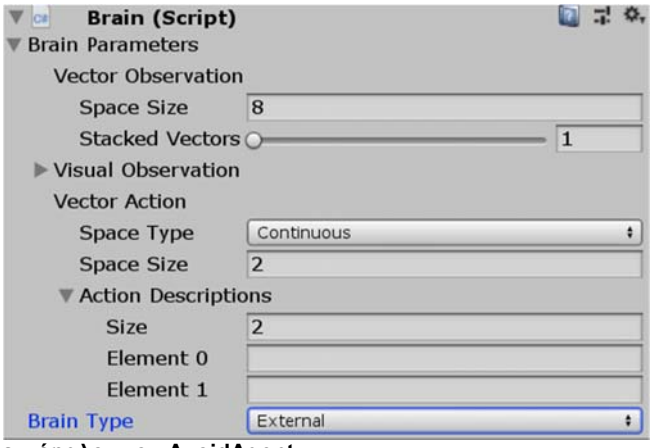
5.4.1.2. Η σκηνή Avoider



Εικόνα 16 - Σκηνή Avoider

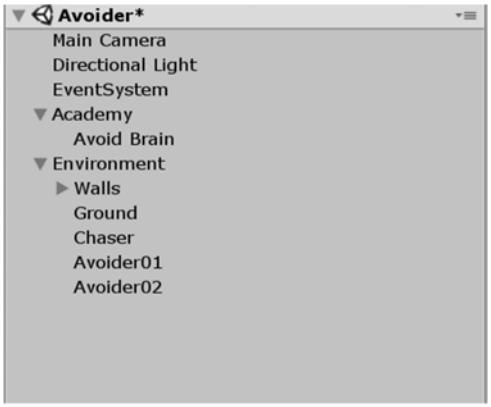
Η σκηνή Avoider αφορά την εκπαίδευση του πράκτορα AvoidAgent και αποτελείται από τα εξής μέρη:

- Main Camera, Directional Light και EventSystem - Προεπιλεγμένα αντικείμενα μιας οποιασδήποτε σκηνής Unity.
- Academy - Η Academy του παραδείγματος που θα υλοποιηθεί. Περιέχει τους εγκεφάλους (στη συγκεκριμένη περίπτωση τον εγκέφαλο Avoid Brain).
- Avoid Brain - Ο εγκέφαλος του πράκτορα AvoidAgent. Θα εκπαιδευτεί μέσω του περιβάλλοντος Anaconda. Σύμφωνα με τον κώδικα που περιγράφεται παρακάτω, ορίζουμε τις παραμέτρους το εγκεφάλου στις εξής:



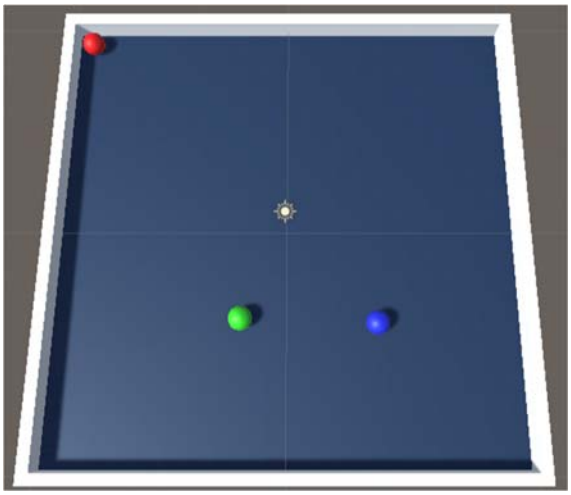
Εικόνα 17 - Ρυθμίσεις εγκέφαλου του AvoidAgent

- Environment - Περιλαμβάνει όλα τα αντικείμενα που σχετίζονται με το περιβάλλον της σκηνής:
    - Ground - Ένα αντικείμενο Unity που χρησιμεύει σαν την επιφάνεια της πίστας στην οποία θα εκπαιδευτεί ο εγκέφαλος του πράκτορα.
    - Walls - Τέσσερα αντικείμενα που χρησιμεύουν ως περίφραξη της πίστας.
    - Chaser - Το αντικείμενο που κυνηγά τους πράκτορες AvoidAgent. Το συγκεκριμένο αντικείμενο κινείται με προκαθορισμένες συναρτήσεις και ο κώδικάς του περιγράφεται παρακάτω στην κλάση AIChaser.
    - Avoider01 και Avoider02 - Τα αντικείμενα που σχετίζονται με τους πράκτορες AvoidAgent. Τα αντικείμενα αυτό ελέγχονται από τον κώδικα που περιγράφεται παρακάτω στην κλάση AvoidAgent.



Εικόνα 18 - Δομή σκηνής Avoider

5.4.1.2. Η σκηνή Main



Εικόνα 19 - Σκηνή Main

Η σκηνή Main αφορά την αξιολόγηση των εκπαιδευμένων εγκεφάλων των πρακτόρων και περιλαμβάνει τα εξής μέρη:

- Main Camera, Directional Light και EventSystem - Προεπιλεγμένα αντικείμενα μιας οποιασδήποτε σκηνής Unity.
- Academy - Η Academy του παραδείγματος που θα υλοποιηθεί. Περιέχει τους εγκεφάλους (στη συγκεκριμένη περίπτωση τους εγκεφάλους Avoid Brain και Chase Brain).
- Avoid Brain - Ο εγκέφαλος του πράκτορα AvoidAgent. Ο εγκέφαλος θα πρέπει να έχει εκπαιδευτεί μέσω του περιβάλλοντος Anaconda και να οριστεί η παράμετρος Brain Type ως Internal με Graph Model το .bytes αρχείο που έχει προκύψει από την εκπαίδευση.
- Chase Brain - Ο εγκέφαλος του πράκτορα ChaserAgent. Ο εγκέφαλος θα πρέπει να έχει εκπαιδευτεί μέσω του περιβάλλοντος Anaconda και να οριστεί η παράμετρος Brain Type ως Internal με Graph Model το .bytes αρχείο που έχει προκύψει από την εκπαίδευση.
- Environment - Περιλαμβάνει όλα τα αντικείμενα που σχετίζονται με το περιβάλλον της σκηνής:
    - Ground - Ένα αντικείμενο Unity που χρησιμεύει σαν την επιφάνεια της πίστας στην οποία θα εκπαιδευτεί ο εγκέφαλος του πράκτορα.
    - Walls - Τέσσερα αντικείμενα που χρησιμεύουν ως περίφραξη της πίστας.
    - Chaser - Το αντικείμενο που σχετίζεται με τον πράκτορα ChaserAgent. Το αντικείμενο αυτό ελέγχεται από τον κώδικα που περιγράφεται παρακάτω στην κλάση ChaserAgent.
    - Avoider01 και Avoider02 - Τα αντικείμενα που σχετίζονται με τους πράκτορες AvoidAgent. Τα αντικείμενα αυτό ελέγχονται από τον κώδικα που περιγράφεται παρακάτω στην κλάση AvoidAgent.

Εικόνα 20 - Δομή σκηνής Main

5.4.2. Περιγραφή πηγαίου κώδικα

Σε αυτή την ενότητα περιγράφονται τα σημαντικά σημεία του πηγαίου κώδικα που σχετίζεται με το παράδειγμα.

5.4.2.1. BaseAcademy

Η κλάση BaseAcademy που υλοποιήθηκε κληρονομεί την κλάση Academy του πακέτου MLAgents και αποτελεί την Academy του παραδείγματος που υλοποιήθηκε. Ενδιαφέρον παρουσιάζει η μέθοδος InitializeAcademy().

```
    public        override        void        InitializeAcademy()        {
  Monitor.SetActive                                                (true);
}
```

Η κλάση Monitor της βιβλιοθήκης ML-Agents επιτρέπει την απεικόνιση πληροφοριών σχετικά με τους πράκτορες ή τη διαδικασία εκπαίδευσης μέσα σε μια σκηνή της Unity. Από προεπιλογή, η κλάση Monitor είναι ενεργοποιημένη μόνο στη φάση συμπερασμάτων και οχι κατα την διάρκεια της εκπαιδευσης. Χρησιμοποιείται η εντολή SetActive(boolean) μεσα στην μεθοδο InitializeAcademy() για να είναι δυνατή η παρακολούθηση και η άντληση πληροφοριών κατά την εκπαίδευση.

5.4.2.2. ChaserAgent

Η κλάση ChaserAgent που υλοποιήθηκε κληρονομεί την κλάση Agent του πακέτου MLAgents. Η συγκεκριμένη κλάση αφορά ένα πράκτορα που θα πρέπει να μάθει να πλοηγείται και να πιάνει ένα στόχο. Ενδιαφέρον παρουσιάζουν οι μέθοδοι Update(), AgentReset(), CollectObservations() και AgentAction().

```
    void Update () {
      Monitor.Log ("TargetId", environment.GetComponent<Environment>
().targetIndex.ToString (), transform);
      Monitor.Log ("CumulativeReward",GetCumulativeReward (),
transform);
      Monitor.Log ("Reward",GetReward (), transform);
    }
```

Για να απεικονίσουμε συγκεκριμένες πληροφορίες σχετικές με τον πράκτορα χρησιμοποιείται η συνάρτηση Log της κλάσης Monitor. Η συνάρτηση Log παίρνει ως όρισμα τις τιμές key (το όνομα της πληροφορίας), value (η τιμή της πληροφορίας) και target (ένα αντικείμενο Transform στο οποίο θα εμφανιστεί η πληροφορία).

```
    public override void AgentReset() {
      // Move the target to a new spot
      Target.position = new Vector3 (Random.value * 18 - 9, 0.5f,
Random.value * 18 - 9);
    }
```

Στη συνάρτηση AgentReset() εκτελείται αρχικοποίηση του πράκτορα, είτε λόγω εκκίνησης της σκηνής είτε λόγω του ότι έχει κληθεί η μέθοδος Done(). Κάθε φορά που εκτελείται η συνάρτηση μετακινείται ο στόχος του πράκτορα σε μια νέα τυχαία θέση στο περιβάλλον.

```
    public override void CollectObservations() {
      // Get target
      int targetIndex =
environment.GetComponent<Environment>().targetIndex;
      Target = targets[targetIndex];
      // Calculate relative position
      Vector3 relativePositon = Target.position -
this.transform.position;
      // Relative position
      AddVectorObs (relativePositon.x/10);
      AddVectorObs (relativePositon.z/10);
      // Agent velocity
      AddVectorObs (rBody.velocity.x / 10);
      AddVectorObs (rBody.velocity.z / 10);
    }
```

Με τη μέθοδο CollectObservations() συλλέγονται οι πληροφορίες που τροφοδοτούν τον εγκέφαλο του πράκτορα. Για τον εν λόγω πράκτορα οι πληροφορίες που συλλέγονται είναι η σχετική θέση του στόχου ως προς τον πράκτορα, καθώς και η τιμή της ταχύτητας του πράκτορα.

```
    public override void AgentAction(float[] vectorAction, string
textAction) {
      // Calculate distance to target
      float distanceToTarget = Vector3.Distance
(this.transform.position, Target.position);
      // Reward for catching the target
      if(distanceToTarget < 1.42f) {
        AddReward (10.0f);
        Done ();
      }
      // Reward for getting closer to the target
      if (distanceToTarget < previousDistance) {
        AddReward (0.05f);
      }
      // Time penalty
      AddReward (-0.01f);
      // Action, size = 2
```

```
      Vector3 controlSignal = Vector3.zero;
      controlSignal.x = vectorAction [0];
      controlSignal.z = vectorAction [1];
      rBody.AddForce (controlSignal * speed);
      // Store the current distance to target
      previousDistance = distanceToTarget;
    }
```

Η μέθοδος AgentAction() τροφοδοτείται από τον εγκέφαλο και χρησιμοποιείται για την επιλογή της επόμενης ενέργειας του πράκτορα. Ο πράκτορας επιβραβεύεται με θετική ή αρνητική επιβράβευση βάσει της απόστασης από το στόχο του, καθώς και από τη σχέση αυτής της απόστασης με την προηγούμενη τιμή της απόστασης. Οι τιμές των σημάτων ελέγχου που λαμβάνονται από τον εγκέφαλο σε μορφή Array χρησιμοποιούνται για να κινήσουν το RigidBody που σχετίζεται με τον πράκτορα.

5.4.2.3. AvoidAgent

Η κλάση AvoidAgent που υλοποιήθηκε κληρονομεί την κλάση Agent του πακέτου MLAgents. Η συγκεκριμένη κλάση αφορά ένα πράκτορα που θα πρέπει να μάθει να πλοηγείται και να μην επιτρέπει σε ένα άλλο πράκτορα να τον πλησιάσει για όσο το δυνατόν μεγαλύτερο χρονικό διάστημα. Ενδιαφέρον παρουσιάζουν οι μέθοδοι Update(), AgentReset(), CollectObservations() και AgentAction().

```
    void Update () {
      Monitor.Log ("AgentId",agentId.ToString (), transform);
      Monitor.Log      ("CumulativeReward",GetCumulativeReward      (),
transform);
      Monitor.Log ("Reward",GetReward (), transform);
    }
```

Για να απεικονίσουμε συγκεκριμένες πληροφορίες σχετικές με τον πράκτορα χρησιμοποιείται η συνάρτηση Log της κλάσης Monitor. Η συνάρτηση Log παίρνει ως όρισμα τις τιμές key (το όνομα της πληροφορίας), value (η τιμή της πληροφορίας) και target (ένα αντικείμενο Transform στο οποίο θα εμφανιστεί η πληροφορία).

```
    public override void AgentReset() {
      // Increment index when reset
      environment.GetComponent<Environment>().targetIndex++;
      if(environment.GetComponent<Environment>().targetIndex ==
environment.GetComponent<Environment>().maxTargetIndex)
environment.GetComponent<Environment>().targetIndex = 0;
      // Move the chaser to a new spot
      this.transform.position = new Vector3 (Random.value * 18 - 9 +
environment.transform.position.x, 0.5f +
environment.transform.position.y, Random.value * 18 - 9 +
environment.transform.position.z);
    }
```

Στη συνάρτηση AgentReset() εκτελείται αρχικοποίηση του πράκτορα, είτε λόγω εκκίνησης της σκηνής είτε λόγω του ότι έχει κληθεί η μέθοδος Done(). Κάθε φορά που εκτελείται η συνάρτηση μετακινείται ο πράκτορας σε μια νέα τυχαία θέση στο περιβάλλον, ενώ ενημερώνεται η τιμή targetIndex στο περιβάλλον ούτως ώστε να αλλάξει ο στόχος.

```
    public override void CollectObservations() {
      // Agent positions
      AddVectorObs (Chaser.transform.position.x / 10);
      AddVectorObs (Chaser.transform.position.z / 10);
      AddVectorObs (this.transform.position.x / 10);
      AddVectorObs (this.transform.position.z / 10);
      // Agent velocity
      AddVectorObs (rBody.velocity.x / 10);
      AddVectorObs (rBody.velocity.z / 10);
      AddVectorObs (Chaser.GetComponent<Rigidbody>().velocity.x / 10);
      AddVectorObs (Chaser.GetComponent<Rigidbody>().velocity.z / 10);
    }
```

Με τη μέθοδο CollectObservations() συλλέγονται οι πληροφορίες που τροφοδοτούν τον εγκέφαλο του πράκτορα. Για τον εν λόγω πράκτορα οι πληροφορίες που συλλέγονται είναι η θέση του πράκτορα που τον κυνηγάει, η θέση του ίδιου του πράκτορα, καθώς και οι ταχύτητες των δύο πρακτόρων.

```
    public override void AgentAction(float[] vectorAction, string
textAction) {
      // Calculate distance from Chaser
      float distanceToTarget = Vector3.Distance
(this.transform.position, Chaser.transform.position);
      // Penalty for being caught
      if(distanceToTarget < 1.42f) {
        AddReward (-10.0f);
        Done ();
      }
      // Time reward
      AddReward (0.01f);
      // Action, size = 2
      Vector3 controlSignal = Vector3.zero;
      controlSignal.x = vectorAction [0];
      controlSignal.z = vectorAction [1];
      rBody.AddForce (controlSignal * speed);
    }
```

Η μέθοδος AgentAction() τροφοδοτείται από τον εγκέφαλο και χρησιμοποιείται για την επιλογή της επόμενης ενέργειας του πράκτορα. Ο πράκτορας επιβραβεύεται με θετική ή αρνητική επιβράβευση βάσει της απόστασης από τον πράκτορα που τον κυνηγάει, καθώς και βάσει του χρόνου που βρίσκεται στο περιβάλλον. Οι τιμές των σημάτων ελέγχου που λαμβάνονται από τον εγκέφαλο σε μορφή Array χρησιμοποιούνται για να κινήσουν το RigidBody που σχετίζεται με τον πράκτορα.

5.4.2.4. AIChaser

Η κλάση AIChaser που υλοποιήθηκε χρησιμοποιείται μόνο κατά τη διάρκεια της εκπαίδευσης του πράκτορα AvoidAgent. Η συγκεκριμένη κλάση αφορά ένα αντικείμενο χωρίς εγκέφαλο που κινείται προς τη θέσης ενός πράκτορα. Ενδιαφέρον παρουσιάζει η μέθοδος Update().

```
    void Update () {
      int targetIndex = environment.targetIndex;
      target = targets[targetIndex];
      // The step size is equal to speed times frame time
      float step = speed * Time.deltaTime;
      // Move our porsition towards target
      transform.position = Vector3.MoveTowards (transform.position,
target.position, step);
    }
```

Στη μέθοδο Update() χρησιμοποιούνται μέθοδοι της Unity για να κινηθεί το αντικείμενο προς τη θέση του πράκτορα. Χρησιμοποιείται η συνάρτηση MoveTowards() που δέχεται σαν ορίσματα τη θέση του πράκτορα, τη θέση του στόχου και την ταχύτητα και κινεί το αντικείμενο προς το στόχο του. Επιπλέον στη μέθοδο Update() ενημερώνεται η τιμή targetIndex που υποδηλώνει το δείκτη του στόχου του αντικειμένου.

5.4.3. Ρύθμιση παραμέτρων

Σύμφωνα με δοκιμαστικές εκπαιδεύσεις που εκτελέστηκαν για τα δύο μοντέλα, παρατηρήθηκε ότι ο προεπιλεγμένος αριθμός βημάτων (`max_steps`) είναι αρκετά μικρός. Εμπειρικά προκύπτει ότι επαρκής αριθμός βημάτων για τον εγκέφαλο Chase Brain είναι 2.0e5, ενώ για τον εγκέφαλο Avoid Brain είναι 3.0e5.

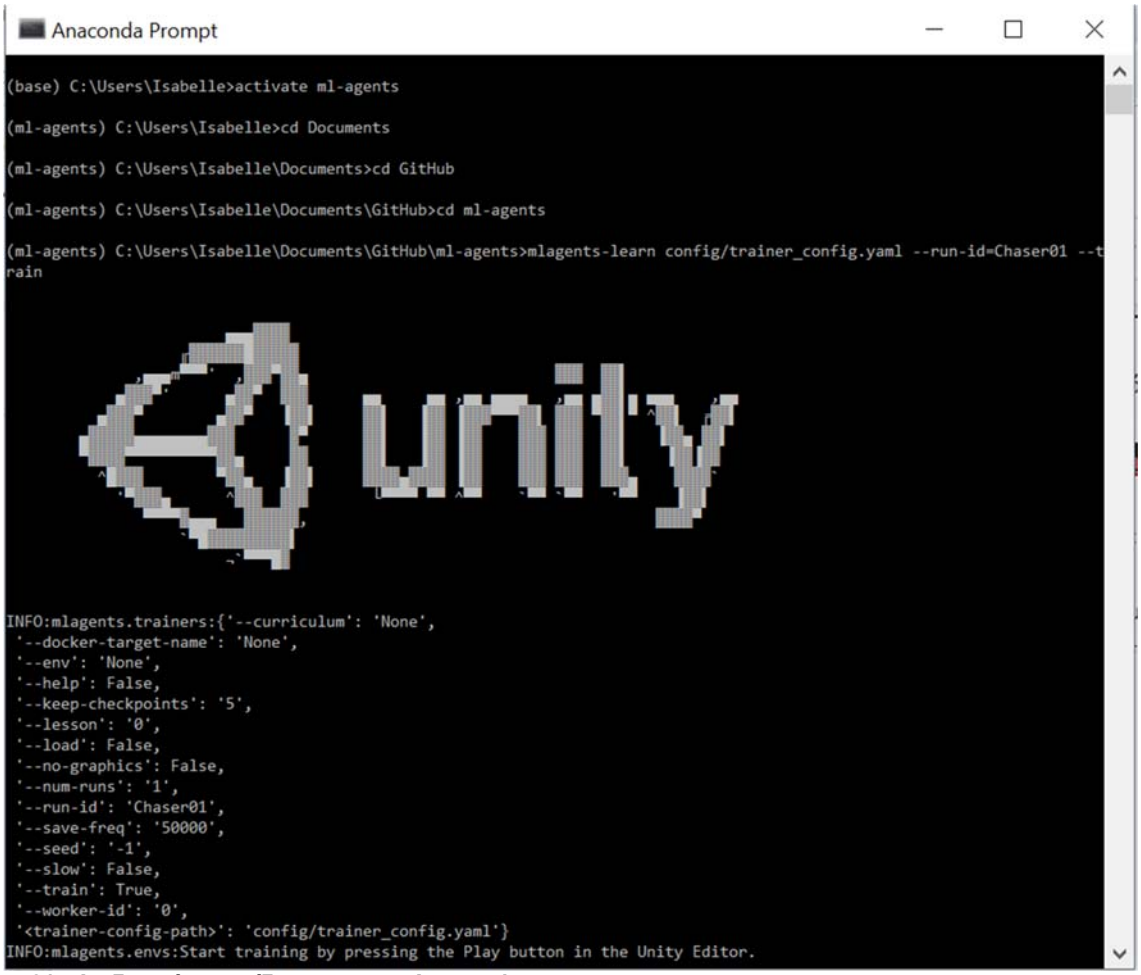
```
    Chase Brain:
     max_steps: 2.0e5

    Avoid Brain:
     max_steps: 3.0e5
```

Για τις υπόλοιπες παραμέτρους δεν απαιτείται κάποια επιπλέον ρύθμιση, καθώς οι προεπιλεγμένες τιμές είναι επαρκείς.

5.4.4. Εκπαίδευση και αποτελέσματα

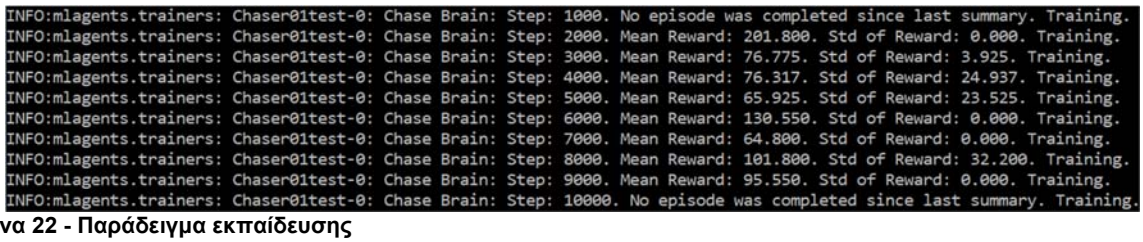
Η διαδικασία εκπαίδευσης για τον εγκέφαλο Chase Brain είναι η ακόλουθη:

1. Εκκίνηση περιβάλλοντος Unity και άνοιγμα της σκηνής Chaser. Καθορισμός του εγκεφάλου Chase Brain σε external.
2. Εκκίνηση του εργαλείου Anaconda.
3. Εκτέλεση της εντολής `activate ml-agents` από το εργαλείο Anaconda. Η εντολή αυτή ενεργοποιεί το πακέτο ML-Agents.
4. Πλοήγηση μέσω της γραμμής εντολών του εργαλείου Anaconda στο φάκελο που βρίσκεται το Unity project.
5. Εκτέλεση της εντολής εκπαίδευσης `mlagents-learn config/trainer_config.yaml --run-id=Chaser01 --train`.
6. Πάτημα του κουμπιού Play στο περιβάλλον της Unity μόλις ζητηθεί από το Anaconda.
7. Αναμονή για ολοκλήρωση της εκπαίδευσης. Ο χρόνος εκπαίδευσης μπορεί να διαφέρει ανά υπολογιστή και η εκπαίδευση μπορεί συχνά να διαρκέσει αρκετά λεπτά.



Εικόνα 21 - Διαδικασία εκπαίδευσης μεσω Anaconda

Η παραπάνω διαδικασία μπορεί να εκτελεστεί και για τον εγκέφαλο Avoid Brain με μόνη διαφοροποίηση ότι θα πρέπει να χρησιμοποιηθεί η σκηνή Avoider και να οριστεί το `run-id` σε `Avoider01`.



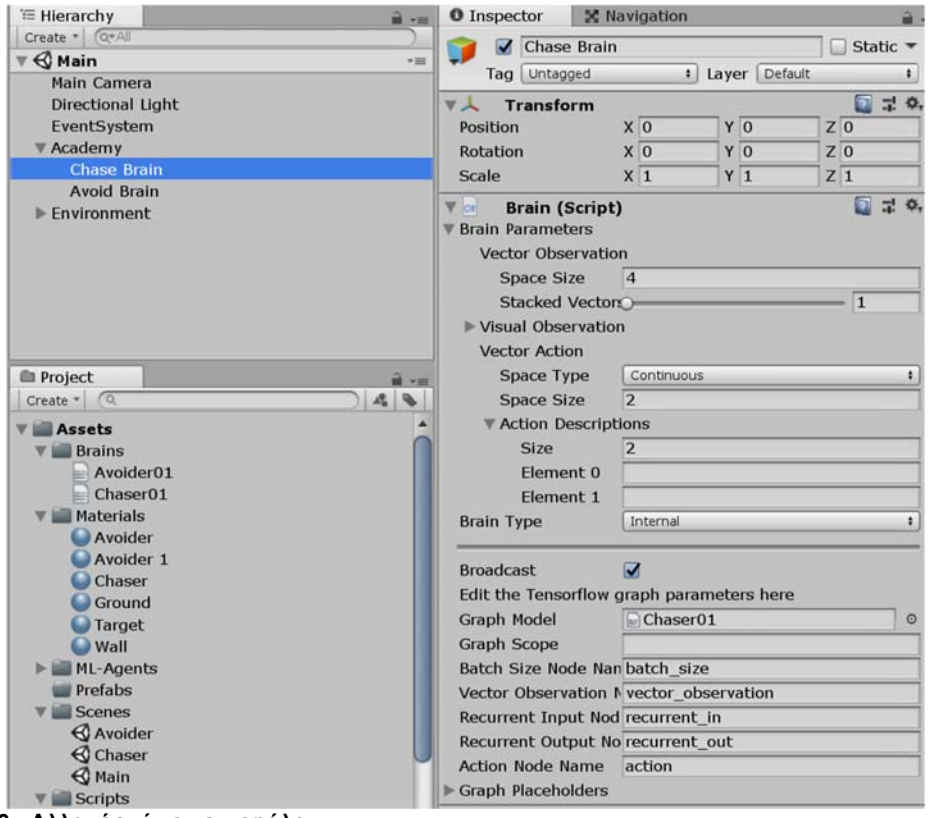
Εικόνα 22 - Παράδειγμα εκπαίδευσης

Παραπάνω φαίνεται ένα παράδειγμα εκπαίδευσης του πράκτορα Chase Brain.

5.4.5. Αξιολόγηση

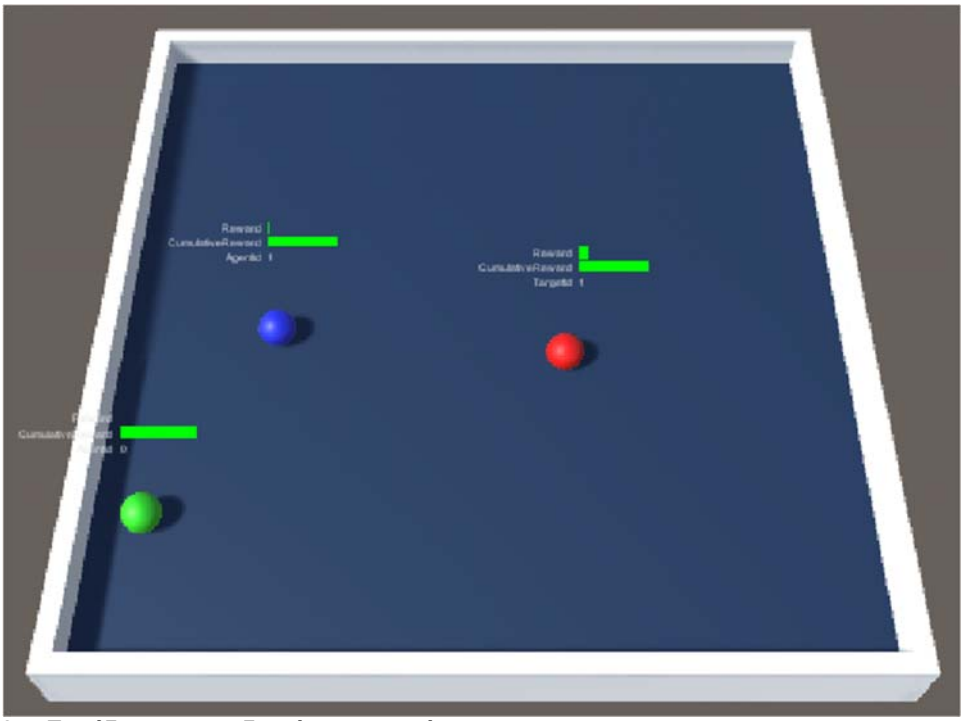
Έχοντας ολοκληρώσει τις διαδικασίες εκπαίδευσης των δύο πρακτόρων, μπορούμε πλέον να αντιγράψουμε τα δύο αρχεία .bytes στο φάκελο Brains του Unity project και στη συνέχεια να

αλλάξουμε το Brain Type των πρακτόρων της σκηνής Main σε Internal, θέτοντας το Graph Model του κάθε πράκτορα στο εκάστοτε αρχείο.



Εικόνα 23 - Αλλαγής τύπου εγκεφάλου

Στη συνέχεια μπορούμε να πατήσουμε Play στη σκηνή Main και να δούμε το αποτέλεσμα της εκπαίδευσης των δύο εγκεφάλων και τον τρόπο που αλληλεπιδρούν μεταξύ τους.

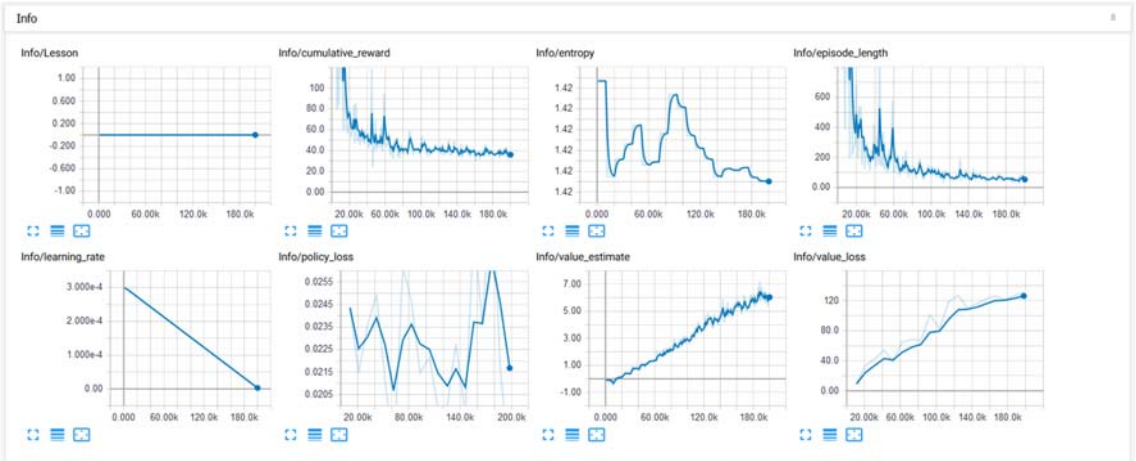


Εικόνα 24 - Παράδειγμα εκπαιδευμένων πρακτόρων

Επιπλέον μπορούμε να ελέγξουμε τις πληροφορίες που προκύπτουν από την εκπαίδευση, εκκινώντας το Tensorboard από το εργαλείο Anaconda με την παρακάτω εντολή:

```
tensorboard --logdir=summaries
```

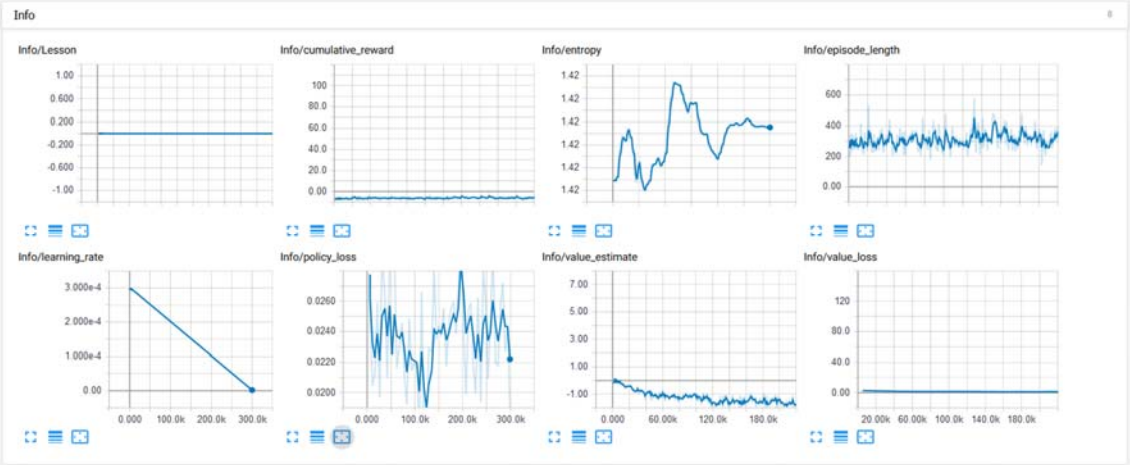
Στη συνέχεια, πλοηγούμαστε στη διεύθυνση localhost:6006 από οποιοδήποτε φυλλομετρητή (browser) και μπορούμε να επιλέξουμε τα μοντέλα που μας ενδιαφέρουν. Παρακάτω φαίνονται τα αποτελέσματα εκπαίδευσης για τα δύο μοντέλα του παραδείγματος μας:



Εικόνα 25 - Στατιστικά στοιχεία Tensorboard του πράκτορα Chaser

Για τον πράκτορα Chaser, παρατηρούμε ότι τα περισσότερα μεγέθη σχετικά με την εκπαίδευση παρουσιάζουν τα αναμενόμενα αποτελέσματα, υπάρχουν όμως αυξομειώσεις και

διαβαθμίσεις που μπορεί να σχετίζονται με το συγκεκριμένο παράδειγμα ή την τυχαιότητα των επεισοδίων.



Εικόνα 26 - Στατιστικά στοιχεία Tensorboard του πράκτορα Avoider

Για τον πράκτορα Avoider, παρατηρούμε ότι τα περισσότερα μεγέθη σχετικά με την εκπαίδευση πλησιάζουν στα αναμενόμενα αποτελέσματα, υπάρχουν όμως αρκετές διαφοροποιήσεις οι οποίες προκύπτουν από την τυχαιότητα των επεισοδίων και το περιβάλλον του παραδείγματος.

6. Αξιολόγηση

Έχοντας περιγράψει και ολοκληρώσει τη διαδικασία εκπαίδευσης και αξιολόγησης του παραπάνω παραδείγματος, προκύπτουν τα εξής συμπεράσματα:

- Και οι δύο πράκτορες εκπαιδεύονται καλύτερα με σχετικά απλά μοντέλα όσον αφορά τις παρατηρήσεις.
- Συγκεκριμένα ο πράκτορας Chaser αρκεί να λαμβάνει ποινή συνεχόμενα βάσει του χρόνου, ούτως ώστε να κινείται και να επιβραβεύεται εφόσον πλησιάζει το στόχο του, ούτως ώστε να εκπαιδευτεί να κυνηγάει ένα στόχο. Επιπλέον, ο πράκτορας επιβραβεύεται με μία μεγάλη βαθμολογία όταν πιάσει το στόχο του.
- Αντίστοιχα, ο πράκτορας Avoider αρκεί να επιβραβεύεται συνεχώς καθώς ο πράκτορας που τον κυνηγά δεν τον έχει πιάσει και να λαμβάνει μια αρκετά μεγάλη ποινή όταν τον πιάσει ο αντίπαλο πράκτορας.
- Παρατηρήθηκε ότι ο πράκτορας Chaser αρκεί να λαμβάνει παρατηρήσεις για τη σχετική απόσταση από το στόχο του και για την ταχύτητά του, ενώ ο πράκτορας Avoider χρησιμοποιεί σαν παρατηρήσεις τις θέσεις των δύο πρακτόρων και τις ταχύτητές τους.
- Σημειώνεται ότι η αντίστροφη επιβράβευση του πράκτορα Chaser δεν λειτουργεί επαρκώς καλά για τον πράκτορα Avoider καθώς τείνει να εγκλωβίζεται στις γωνίες, δεδομένου ότι προσπαθεί να αποκτήσει τη μέγιστη επιβράβευση απομακρυνόμενος από τον αντίπαλο πράκτορα και στη συνέχεια δε μπορεί να μάθει να βγαίνει από αυτή τη θέση.
- Σαν λύση στο παραπάνω πρόβλημα, δοκιμάστηκε και η προσθήκη παρατηρήσεων στον πράκτορα Avoider που αφορούν τη σχέση του με τα περιθώρια του χώρου, όπως επίσης και ποινές που σκοπό είχαν να τον αναγκάσουν να απεγκωλβιστεί, χωρίς όμως αποτέλεσμα, λόγω του ότι το μοντέλο αποκτούσε μεγαλύτερη πολυπλοκότητα. Σημειώνεται επίσης ότι η λύση αυτή, ακόμα και αν λειτουργούσε, δε θα μπορούσε να γενικευθεί για άλλους χώρους, επομένως δε θα ήταν ιδανική. Δοκιμάστηκε επιπλέον η επιβράβευση/ποινή και των δύο πρακτόρων συναρτήσει της απόστασης μεταξύ τους, η οποία όμως δημιούργησε προβλήματα στην εκπαίδευση, λόγω αστάθειας του μοντέλου, καθώς αυτό μεταβάλλεται συνεχώς με αυτό τον τρόπο.

- Παρατηρήθηκε ότι και οι δύο πράκτορες εκπαιδεύονται επαρκώς σε περίπου 200.000 επαναλήψεις. Πιο συγκεκριμένα, ο πράκτορας Avoider χρειάζεται περίπου 250.000 - 300.000 επαναλήψεις για να εκπαιδευτεί αρκετά.
- Αυξάνοντας τον αριθμό επαναλήψεων στην εκπαίδευση, παρατηρούμε ότι οι πράκτορες από ένα σημείο και μετά αλλάζουν συμπεριφορά και δεν λειτουργούν όπως πριν, δίνοντας λιγότερο βέλτιστα αποτελέσματα.
- Οι πράκτορες Avoider, εφόσον υπάρχουν δύο στο χώρο, δε συνεργάζονται μεταξύ τους. Αυτό οφείλεται στο γεγονός ότι δεν έχουν κάποιο κίνητρο συνεργασίας (πχ δεν λαμβάνουν ποινή μόνο εφόσον πιάσει ο αντίπαλος πράκτορας το σωστό στόχο).

7. Συμπέρασμα

Αυτό το κεφάλαιο συνοψίζει τα βήματα που ακολουθήθηκαν σε αυτό το έργο, αναλύοντας την προσφορά του έργου καθώς και μελλοντικές βελτιώσεις που μπορούν να προστεθούν.

7.1. ΣΎΝΟΨΗ ΈΡΓΟΥ

Ο στόχος του έργου ήταν να παρουσιαστεί η βιβλιοθήκη εργαλείων ML-Agents της Unity και να αναλυθεί μια υλοποίηση ενός περιβάλλοντος εκπαίδευσης πρακτόρων μέσω της ενισχυτικής μάθησης. Μετά από μια ενδελεχή θεωρητική μελέτη και ανάλυση των βέλτιστων τακτικών εκπαίδευσης ο στόχος αυτός εκπληρώθηκε αφού το εκπαιδευτικό μοντέλο το οποίο υλοποιήθηκε ικανοποιεί πλήρως τις απαιτήσεις που ορίστηκαν. Ειδικότερα τα μοντέλα είναι ικανά να εκπληρώσουν με επιτυχία τον στόχο ο οποίος τους έχει ανατεθεί και να επιδείξουν τεχνητή ευφυΐα.

7.2. ΜΕΛΛΟΝΤΙΚΈΣ ΒΕΛΤΙΏΣΕΙΣ

Στην ενότητα αυτή προτείνονται κάποιες μελλοντικές βελτιώσεις που μπορούν να υλοποιηθούν έτσι ώστε να βελτιώνονται σταδιακά οι πράκτορες ή να εκπαιδεύονται πιο γρήγορα.

Περίπλοκα περιβάλλοντα

Η προσθήκη εμποδίων ή αφαίρεση της περίφραξης (τοίχων) στις σκηνές εκπαίδευσης θα καθιστούσε την εκπαίδευση πιο περίπλοκη και πιο ενδιαφέρουσα. Επιπλέων θα μπορούσε το περιβάλλον να γίνει ακόμα πιο περίπλοκο προσθέτοντας διαφορετικά σχήματα πίστας ή υψώματα και εμπόδια.

Περισσότεροι πράκτορες

Προσθέτοντας περισσότερους πράκτορες θα μπορούσε να βελτιωθεί η εκπαίδευση του εκάστοτε εγκεφάλου καθώς θα είχε περισσότερα δεδομένα για να τροφοδοτήσει το νευρωνικό δίκτυο.

Περισσότερες εκδοχές της ίδιας σκηνής

Δημιουργώντας αντίγραφα του εκπαιδευτικού περιβάλλοντος με τον ίδιο εγκέφαλο θα μπορούσε να προσφέρει την δυνατότητα στον εγκέφαλο να εκπαιδεύεται πιο γρήγορα, επομένως να χρειάζεται λιγότερο χρόνο για να φτάσει στην βέλτιστη λύση.

Βιβλιογραφία

Coppin, B. (2004). *Artificial Intelligence Illuminated*. Ανάκτηση 05 2018

Lanham, M. (2018). *Learn Unity ML-Agents - Fundamentals of Unity Machine Learning*. Ανάκτηση 07 2018

McCorduck, P. (2004). *Machines who think : a personal inquiry into the history and prospects of artificial intelligence.* Ανάκτηση 05 2018

Schank, R. C. (1991). *Where's The AI. AI Magazine*, 12(4), 38. Ανάκτηση 04 2018

Schwab, B. (2009). AI Game Engine Programming. Ανάκτηση 04 2018

Unity. (2017, 08 22). *Unity AI – Reinforcement Learning with Q-Learning*. Ανάκτηση 05 2018, από Unity Blog: https://blogs.unity3d.com/2017/08/22/unity-ai-reinforcement-learning-with-q-learning/

Unity. (2017, 06 26). *Unity AI-themed Blog Entries*. Ανάκτηση 05 2018, από Unity Blog: https://blogs.unity3d.com/2017/06/26/unity-ai-themed-blog-entries/

Unity-Technologies. (2018). *Background: Machine Learning*. Ανάκτηση 06 2018, από GitHub: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Background-Machine-Learning.md

Unity-Technologies. (2018). *Background: TensorFlow*. Ανάκτηση 06 2018, από GitHub: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Background-TensorFlow.md

Unity-Technologies. (2018). *Basic Guide*. Ανάκτηση 07 2018, από GitHub: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Basic-Guide.md

Unity-Technologies. (2018). *Getting Started with the 3D Balance Ball Environment*. Ανάκτηση 07 2018, από GitHub: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Getting-Started-with-Balance-Ball.md

Unity-Technologies. (2018). *Installing ML-Agents Toolkit for Windows*. Ανάκτηση 07 2018, από GitHub: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Installation-Windows.md

Unity-Technologies. (2018). *Making a New Learning Environment*. Ανάκτηση 08 2018, από GitHub: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Create-New.md

Unity-Technologies. (2018). *Training with Proximal Policy Optimization*. Ανάκτηση 08 2018, από GitHub: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-PPO.md

Γεωργούλη, Α. (2015). *ΚΕΦΑΛΑΙΟ 4 - Μηχανική Μάθηση*. Ανάκτηση 05 2018, από Τεχνητή Νοημοσύνη» – Μια εισαγωγική προσέγγιση: http://repfiles.kallipos.gr/html_books/93/04a-main.html

Παράρτημα

ΠΗΓΑΊΟΣ ΚΏΔΙΚΑΣ: ΑΡΧΕΙΟΥ BASEACADEMY

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;
    using MLAgents;

    public class BaseAcademy : Academy {

        public override void InitializeAcademy() {

            Monitor.SetActive (true);

        }

    }
```

ΠΗΓΑΊΟΣ ΚΏΔΙΚΑΣ: ΑΡΧΕΙΟΥ CHASERAGENT

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;
    using MLAgents;
    using static BaseAcademy;

    public class ChaserAgent : Agent {

        public float speed = 10;
        public List<Transform> targets;
        private float previousDistance = float.MaxValue;
        public GameObject environment;
        public Transform Target;

        Rigidbody rBody;
        void Start () {
            rBody = GetComponent<Rigidbody> ();
        }

        void Update () {
            Monitor.Log
("TargetId",environment.GetComponent<Environment>().targetIndex.ToString
(), transform);
            Monitor.Log     ("CumulativeReward",GetCumulativeReward     (),
transform);
            Monitor.Log ("Reward",GetReward (), transform);
        }

        public override void AgentReset() {

            // Move the target to a new spot
```

```
            Target.position = new Vector3 (Random.value * 18 - 9, 0.5f,
Random.value * 18 - 9);
        }

        public override void CollectObservations() {

            // Get target
            int                         targetIndex                         =
environment.GetComponent<Environment>().targetIndex;
            Target = targets[targetIndex];

            // Calculate relative position
            Vector3      relativePositon      =      Target.position      -
this.transform.position;

            // Relative position
            AddVectorObs (relativePositon.x/10);
            AddVectorObs (relativePositon.z/10);

            // Agent velocity
            AddVectorObs (rBody.velocity.x / 10);
            AddVectorObs (rBody.velocity.z / 10);

        }

        public  override  void  AgentAction(float[]  vectorAction,  string
textAction) {

            // Calculate distance to target
            float        distanceToTarget        =        Vector3.Distance
(this.transform.position, Target.position);

            // Reward for catching the target
            if(distanceToTarget < 1.42f) {
                AddReward (10.0f);
                Done ();
            }

            // Reward for getting closer to the target
            if (distanceToTarget < previousDistance) {
                AddReward (0.05f);
            }

            // Time penalty
            AddReward (-0.01f);

            // Action, size = 2
            Vector3 controlSignal = Vector3.zero;
            controlSignal.x = vectorAction [0];
            controlSignal.z = vectorAction [1];
            rBody.AddForce (controlSignal * speed);
```

```
            // Store the current distance to target
            previousDistance = distanceToTarget;
        }
    }
```

ΠΗΓΑΊΟΣ ΚΏΔΙΚΑΣ: ΑΡΧΕΙΟΥ AVOIDAGENT

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;
    using MLAgents;
    using static BaseAcademy;

    public class AvoidAgent : Agent {

        public float speed = 10;
        public int agentId;
        public GameObject environment;
        public GameObject Chaser;
        Rigidbody rBody;

        void Start () {
            rBody = GetComponent<Rigidbody> ();
        }

        void Update () {
            Monitor.Log ("AgentId",agentId.ToString (), transform);
            Monitor.Log     ("CumulativeReward",GetCumulativeReward     (),
transform);
            Monitor.Log ("Reward",GetReward (), transform);
        }

        public override void AgentReset() {

            // Increment index when reset
            environment.GetComponent<Environment>().targetIndex++;
            if(environment.GetComponent<Environment>().targetIndex      ==
environment.GetComponent<Environment>().maxTargetIndex)
environment.GetComponent<Environment>().targetIndex = 0;

            // Move the chaser to a new spot
            this.transform.position = new Vector3 (Random.value * 18 - 9 +
environment.transform.position.x, 0.5f + environment.transform.position.y,
Random.value * 18 - 9 + environment.transform.position.z);
        }

        // Observation size = 8
        public override void CollectObservations() {

            // Agent positions
            AddVectorObs (Chaser.transform.position.x / 10);
            AddVectorObs (Chaser.transform.position.z / 10);
```

```
            AddVectorObs (this.transform.position.x / 10);
            AddVectorObs (this.transform.position.z / 10);

            // Agent velocity
            AddVectorObs (rBody.velocity.x / 10);
            AddVectorObs (rBody.velocity.z / 10);
            AddVectorObs  (Chaser.GetComponent<Rigidbody>().velocity.x  /
10);
            AddVectorObs  (Chaser.GetComponent<Rigidbody>().velocity.z  /
10);
        }

        public  override  void  AgentAction(float[]  vectorAction,  string
textAction) {

            // Calculate distance from Chaser
            float       distanceToTarget       =       Vector3.Distance
(this.transform.position, Chaser.transform.position);

            // Penalty for being caught
            if(distanceToTarget < 1.42f) {
                AddReward (-10.0f);
                Done ();
            }

            // Time reward
            AddReward (0.01f);

            // Action, size = 2
            Vector3 controlSignal = Vector3.zero;
            controlSignal.x = vectorAction [0];
            controlSignal.z = vectorAction [1];
            rBody.AddForce (controlSignal * speed);
        }
    }
```

ΠΗΓΑΊΟΣ ΚΏΔΙΚΑΣ: ΑΡΧΕΙΟΥ AICHASER

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;

    public class AIChaser : MonoBehaviour {

        public List<Transform> targets;
        private Transform target;
        public Environment environment;

        //Speed in units per second
        public float speed = 10;

        void Update () {
```

```
            int targetIndex = environment.targetIndex;
            target = targets[targetIndex];

            // The step size is equal to speed times frame time
            float step = speed * Time.deltaTime;

            // Move our porsition towards target
            transform.position = Vector3.MoveTowards (transform.position,
target.position, step);
        }

    }
```

ΠΗΓΑΊΟΣ ΚΏΔΙΚΑΣ: ΑΡΧΕΙΟΥ ENVIRONMENT

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;

    public class Environment : MonoBehaviour {

        public int targetIndex = 0;
        public int maxTargetIndex;

        // Use this for initialization - `else` condition only applies to
Chaser scene
        void Start () {
            if (this.GetComponentsInChildren<AvoidAgent> ().Length > 0)
                maxTargetIndex = this.GetComponentsInChildren<AvoidAgent>
().Length;
            else
                maxTargetIndex = 1;
        }

        // Update is called once per frame
        void Update () {

        }
    }
```