ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΙΑ ΜΑΘΗΜΑΤΟΣ

«ΕΥΦΥΕΙΣ ΠΡΑΚΤΟΤΡΕΣ»

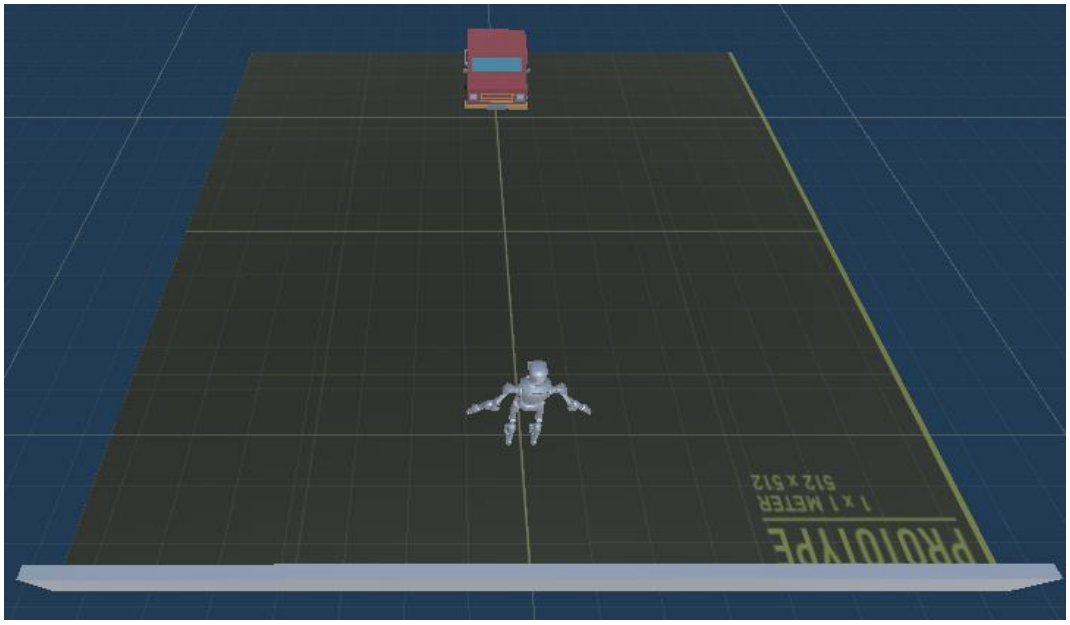
ΠΕΙΡΑΙΑΣ 2023

ΣΤΟΙΧΕΙΑ ΟΜΑΔΑΣ	
ΑΥΓΕΡΙΝΟΣ ΧΡΗΣΤΟΣ	Π19020
ΠΑΝΑΓΙΩΤΟΠΟΥΛΟΣ ΔΗΜΗΤΡΙΟΣ	Π19130
ΑΘΑΝΑΣΙΟΣ ΒΙΤΑΚΗΣ	Π19247

Πίνακας περιεχομένων

1. Περιγραφή του προβλήματος

Το αντικείμενο της εργασίας είναι η δημιουργία ενός mini 2D ή 3D παιχνιδιού εκπαίδευσης ευφυών Πρακτόρων μέσα στο περιβάλλον της Unity. Εμείς δημιουργήσαμε ένα 3D παιχνίδι ελέγχοντας πώς λειτουργεί η τεχνητή νοημοσύνη αποφυγής παικτών με τους ML-Agents. Πιο συγκεκριμένα φτιάξαμε έναν πράκτορα(ένα ρομπότ) που πρέπει να αποφύγει τη σύγκρουση με ένα αυτοκίνητο (στόχος-target). Επίσης, εφαρμόζονται αλλαγές στην ταχύτητα στόχου για να σας δείξουν πόσο έξυπνος είναι ο πράκτορας ώστε να εφαρμόζει αλλαγές στη συχνότητα κίνησης.



2. Περιγραφή της θεωρητικής βάσης της εφαρμογής (αλγόριθμοι και μεθοδολογίες)

Η θεωρητική βάση της εφαρμογής του παιχνιδιού εκπαίδευσης ευφυών πρακτόρων στο περιβάλλον της Unity περιλαμβάνει τη χρήση αλγορίθμων και μεθοδολογιών που επιτρέπουν στον πράκτορα να λειτουργεί έξυπνα και να αποφεύγει τη σύγκρουση με το αυτοκίνητο στόχο.

Έτσι όσον αφορά τους αλγορίθμους και τις μεθοδολογίες, μια δημοφιλής προσέγγιση είναι η χρήση αλγορίθμων ενισχυτικής μάθησης, όπως ο αλγόριθμος PPO που χρησιμοποιήθηκε στη συγκεκριμένη περίπτωση.

```yaml
behaviors:
  PlayerAvoidance:
    trainer_type: ppo
    hyperparameters:
      batch_size: 10
      buffer_size: 100
      learning_rate: 3.0
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_sche
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 500000
    time_horizon: 64
    summary_freq: 10000
```

Yaml file (config)

Ο Proximal Policy Optimization (PPO) είναι ένας σύγχρονος και ο πιο κύριος αλγόριθμος ενισχυτικής μάθησης που χρησιμοποιείται για την εκπαίδευση των πρακτόρων στην εργαλειοθήκη ML Agents της Unity. Είναι ένας αλγόριθμος βελτιστοποίησης πολιτικής που ανήκει στην κατηγορία των μεθόδων "on-policy". Έχει σχεδιαστεί για να βελτιστοποιεί την πολιτική ενός πράκτορα συλλέγοντας επαναληπτικά εμπειρίες στο περιβάλλον και ενημερώνοντας την πολιτική με βάση τα δεδομένα που συλλέγονται. Ο αλγόριθμος PPO των ML-Agents υλοποιείται σε TensorFlow και εκτελείται σε ξεχωριστή διεργασία Python (επικοινωνώντας με την εκτελούμενη εφαρμογή Unity μέσω μιας υποδοχής).

Για να εκπαιδευτεί ένας πράκτορας, θα πρέπει να του δώσουμε ένα ή περισσότερα σήματα ανταμοιβής τα οποία ο πράκτορας θα πρέπει να προσπαθήσει να μεγιστοποιήσει. Στη συνέχεια θα μιλήσουμε πιο αναλυτικά για τα σήματα ανταμοιβής που δίνουμε εμείς στους πράκτορες μας.

Ο PPO αλγόριθμος έχει γίνει δημοφιλής λόγω της σταθερότητάς του, της αποδοτικότητας του δείγματος και της δυνατότητας χειρισμού συνεχών χώρων δράσης. Επιτυγχάνει μια ισορροπία μεταξύ εξερεύνησης και εκμετάλλευσης, επιτρέποντας στον πράκτορα να βελτιώνει την πολιτική του με την πάροδο του χρόνου.

Συνολικά, η εφαρμογή χρησιμοποιεί μια συνδυασμένη προσέγγιση μεταξύ της αναπαράστασης γνώσης, των αλγορίθμων ενισχυτικής μάθησης και των προσαρμογών στους αλγορίθμους για να επιτρέψει στον πράκτορα να μάθει και να εφαρμόζει έξυπνες στρατηγικές για την αποφυγή της σύγκρουσης με τον στόχο και την αντιμετώπιση των αλλαγών στην ταχύτητα του στόχου.

3. Περιγραφή σημαντικών σχεδιαστικών αποφάσεων και στοιχείων υλοποίησης

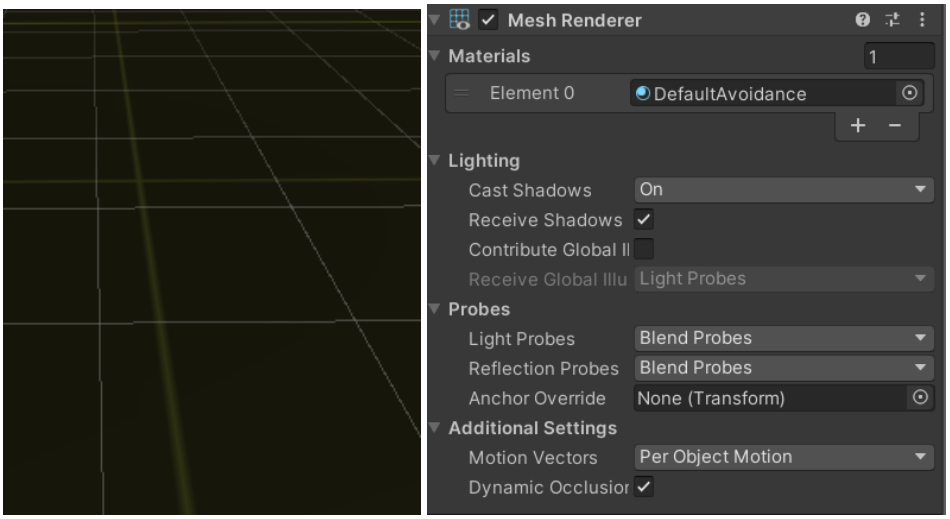
Στο πλαίσιο της εργασίας μας, έχουν παρθεί σημαντικές σχεδιαστικές αποφάσεις και στοιχείων υλοποίησης. Ορισμένες από τις κύριες αποφάσεις και στοιχεία υλοποίησης περιλαμβάνουν:

i. <u>Αναπαράσταση Εδάφους</u>

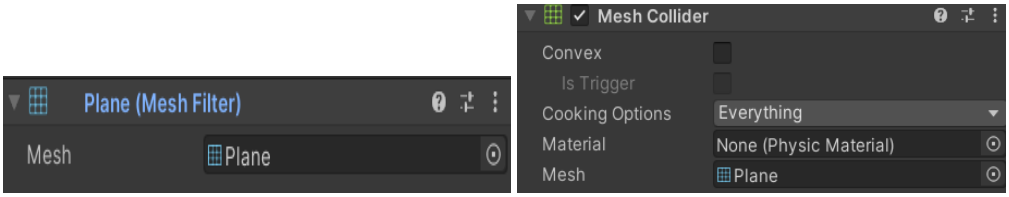
Το έδαφος που θα διαδραματίζεται το παιχνίδι και θα περιλαμβάνει τον πράκτορα, τον στόχο και ότι άλλο χρειάζεται έχει αναπαρασταθεί στο περιβάλλον της Unity. Αυτό περιλαμβάνει τον σχεδιασμό και την υλοποίηση του μοντέλου του εδάφους.

Τα components του εδάφους μας είναι:

- Mesh Renderer για την εμφάνιση της οπτικής αναπαράστασης του εδάφους μέσα στον κόσμο του παιχνιδιού με mesh filter ένα πλέγμα (Plane).
- Mesh Collider για να μπορεί αν κάνει collide με αλλά αντικείμενα στη σκηνή.



Μοντέλο Πλέγματος



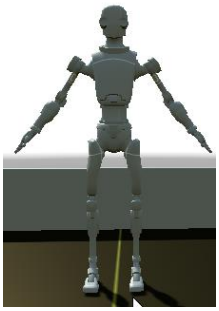
ii. <u>Αναπαράσταση Πράκτορα</u>

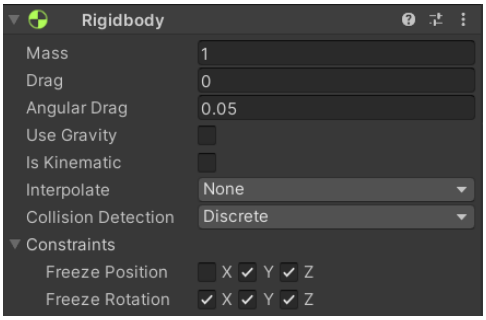
Έχει γίνει απόφαση για την αναπαράσταση του πράκτορα ως ένα ρομπότ εντός του περιβάλλοντος της Unity. Αυτή η αναπαράσταση περιλαμβάνει το σχεδιασμό και την υλοποίηση του μοντέλου του ρομπότ, των αισθητήρων του, και των δυνατοτήτων κίνησής του. Το ρομπότ θεωρείται ο παίκτης-player του παιχνιδιού.

Τα components του player-agent μας είναι:

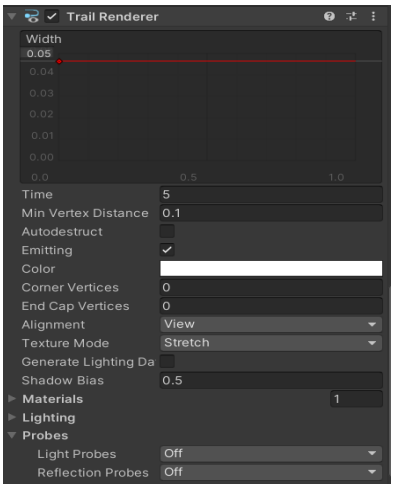
- Sphere Collider ώστε να μπορεί να κάνει collide με τον στόχο-target (που σημαίνει τιμωρία του πράκτορα).
- Mesh Renderer για την εμφάνιση της οπτικής αναπαράστασης ενός πράκτορα μέσα στον κόσμο του παιχνιδιού.
- Rigidbody για να προσομοιώνουμε ρεαλιστικές κινήσεις και αλληλεπιδράσεις των πρακτόρων του παιχνιδιού με βάση τη φυσική. Έτσι ανταποκρίνονται σε δυνάμεις όπως η βαρύτητα, οι συγκρούσεις και οι εφαρμοζόμενες δυνάμεις με στόχο να κινούμαστε στο χώρο μόνο στον X άξονα. Για αυτό και στους περιορισμούς έχουμε κάνει freeze τον Y και Z άξονα της θέσης του πράκτορας καθώς και όλους τους άξονες της περιστροφής.
- Επιπλέον έχουμε ένα Trail Renderer component για να μπορούμε να ορίσουμε κινήσεις τους πράκτορα δεξιά και αριστερά μέσω του script Agent Avoidance που έχουμε προσθέσει.
- Σε αυτό το script έχουμε δηλώσει το material που θα χρησιμοποιείται ως το έδαφος(Ground), το success,failure και default material για κάθε κατάσταση, την ταχύτητα που θα κινείται ο πράκτορας(50) και τις 3 θέσεις που θα κινείται στο χώρο(μια στην προκαθορισμένη θέση στη μέση, μια με offset 3.2 στα δεξιά και μια με offset -3.2 δηλαδή στα αριστερά).
- Ακόμα έχουμε το DicisionRequester component με τις default τιμές(5 περίοδοι απόφασης) και τέλος
- τα behavior parameters που έχουμε επιλέξει το όνομα PlayerAvoidance για όλους τους πράκτορες ώστε να ακολουθούν όλοι την ιδιά συμπεριφορά με 6 observations που σημαίνει ότι έχουμε 3 για τους άξονες x,y,z για τον πράκτορα-agent και άλλους 3 για τους άξονες του στόχου-target. Ακόμη χρησιμοποιούμε Discrete actions καθώς δεν θέλουμε συνεχομένη κίνηση αλλά θα έχουμε τις 3 θέσεις που θα ανιχνεύουμε γιατρό και το branch size είναι 3. Το μοντέλο που έχουμε εισάγει (PlayerAvoidance NNModel) είναι το τελευταίο εκπαιδευμένο μοντέλο μας με δοκιμές για διάφορες ταχύτητες του στόχου μας ώστε ακόμα και όταν έχουμε πολύ γρήγορη κίνηση οι πράκτορες μας να είναι αρκετά έξυπνοι για να αποφεύγουν τους στόχους. Τέλος στο behavior type έχουμε επιλέξει το Default που σημαίνει ότι κάνουμε training έναντι των mlagnets της python και όχι

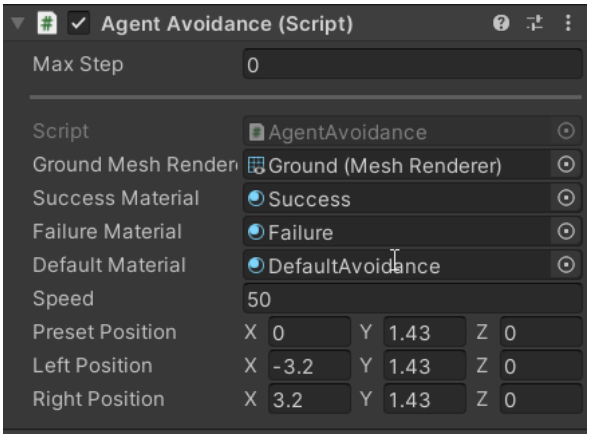
με το heuristic μοντέλο που ελέγχουμε εμείς τα actions που θα οδηγούν την συμπεριφορά.

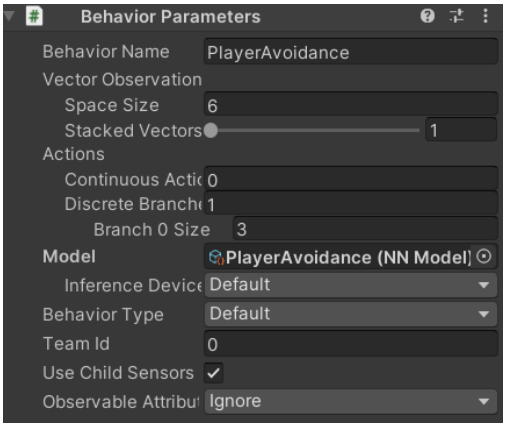


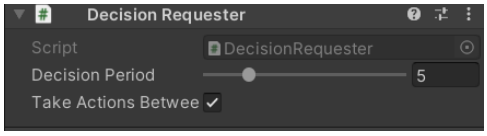


Μοντέλο Ρομπότ







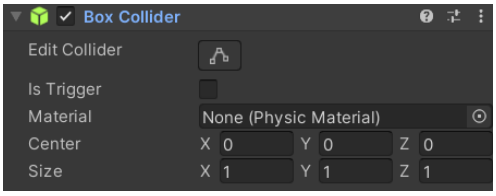


iii. Αναπαράσταση Στόχου

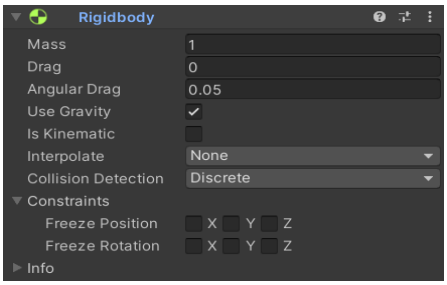
Ο στόχος του πράκτορα, που είναι ένα αυτοκίνητο, έχει επίσης αναπαρασταθεί στο περιβάλλον της Unity. Αυτό περιλαμβάνει τον σχεδιασμό και την υλοποίηση του μοντέλου του αυτοκινήτου, την ανίχνευση σύγκρουσης με τον πράκτορα και τον υπολογισμό της αντίστασης για τον πράκτορα να αποφύγει τη σύγκρουση.

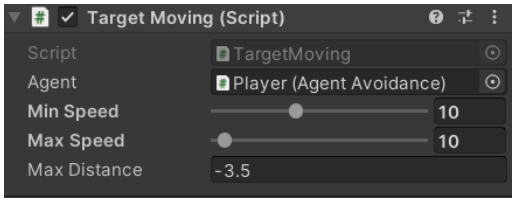
Τα components του στόχου-target μας είναι:

- Box Collider ώστε να μπορεί να κάνει collide με τον πράκτορα (που σημαίνει τιμωρία του πράκτορα) και με τον τοίχο (που σημαίνει επιβράβευση του πράκτορα καθώς θα έχει αποφύγει το στόχο).
- Mesh Renderer για την εμφάνιση της οπτικής αναπαράστασης ενός πράκτορα μέσα στον κόσμο του παιχνιδιού.
- Rigidbody για να προσομοιώνουμε ρεαλιστικές κινήσεις και αλληλεπιδράσεις των στόχων του παιχνιδιού με βάση τη φυσική. Έτσι ανταποκρίνονται σε δυνάμεις όπως η βαρύτητα, οι συγκρούσεις και οι εφαρμοζόμενες δυνάμεις με στόχο να κινούμαστε στο χώρο μόνο στον X άξονα. Δεν έχουμε περιορισμούς στους άξονες.
- Τέλος έχουμε ένα script το Target Moving που έχουμε δηλώσει τον πράκτορα(agent) ώστε να μπορούμε να δίνουμε κατεύθυνση του στόχου προς τον πράκτορα, την ελάχιστη ταχύτητα του στόχου (που είναι 10 αλλά μπορεί να κυμανθεί από 0.5 μέχρι 25), την μέγιστη ταχύτητα του πράκτορα (που είναι 10 αλλά μπορεί να κυμανθεί από 5 μέχρι 150), και την μέγιστη απόσταση που μπορεί να φτάσει ο στόχος που είναι λίγο μετα τον τοίχο ώστε να κάνει collide και μετα να ξανά τοποθετηθεί στην αρχική του θέση και ο στόχος και ο πράκτορας.



Μοντέλο Αμάξι



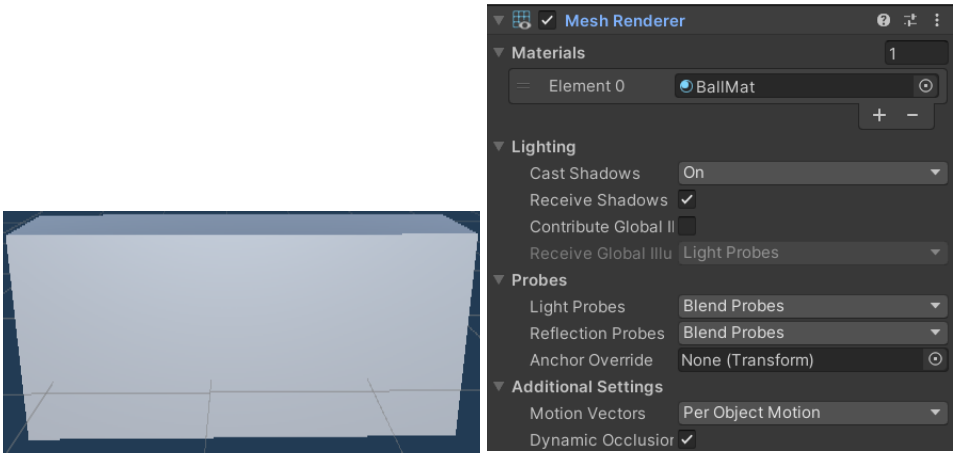


iv. Αναπαράσταση Τοίχου

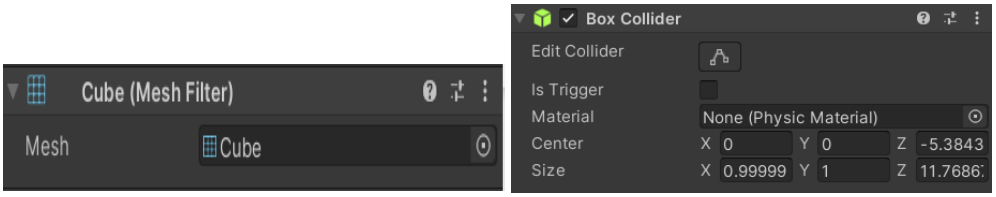
Ο τοίχος που βρίσκεται πίσω από τον παίκτη ώστε αν ο στόχος δεν συγκρουστεί με τον πράκτορα να συγκρουστεί με αυτόν έχει επίσης αναπαρασταθεί στο περιβάλλον της Unity. Αυτό περιλαμβάνει τον σχεδιασμό και την υλοποίηση του μοντέλου του τοίχου.

Τα components του τοίχου μας είναι:

- Mesh Renderer για την εμφάνιση της οπτικής αναπαράστασης του τοίχου μέσα στον κόσμο του παιχνιδιού με mesh filter έναν κύβο (Cube).
- Box Collider ώστε να μπορεί να κάνει collide με τον στόχο (που σημαίνει επιβράβευση του πράκτορα καθώς θα έχει αποφύγει το στόχο) και να γίνουν reset στις θέσεις τους ο πράκτορας και ο στόχος.



Μοντέλο Τοίχου



v. **Επιλογή Αλγορίθμου Ενισχυτικής Μάθησης**

Έχει γίνει επιλογή του αλγορίθμου ενίσχυσης μάθησης για την εκπαίδευση του πράκτορα. Αυτός είναι ο αλγόριθμος Proximal Policy Optimization (PPO), που είναι ένας δημοφιλής αλγόριθμος που χρησιμοποιείται στο πλαίσιο των ML Agents της Unity, όπως αναφέρθηκε προηγουμένως.

vi. **Ρύθμιση Περιβάλλοντος και Παραμέτρων**

Έχει γίνει ρύθμιση του περιβάλλοντος του παιχνιδιού και των παραμέτρων του οι οποίες αλλάζουν δυναμικά κατά τη διάρκεια του παιχνιδιού, για να ελέγξουμε την ικανότητα του πράκτορα να προσαρμόζεται στην κίνηση του στόχου και να επιτύχουμε μια πιο έξυπνη συμπεριφορά του πράκτορα. Αυτό περιλαμβάνει τη ρύθμιση της ταχύτητας κίνησης του στόχου, της συχνότητας εμφάνισης του στόχου, και των περιορισμών κίνησης για τον πράκτορα που αναλύσαμε παραπάνω.

Οι παραπάνω αποφάσεις και στοιχεία υλοποίησης παίζουν σημαντικό ρόλο στη δημιουργία και λειτουργία του παιχνιδιού με τους ευφυείς Πράκτορες.

4. Ανάλυση των Scripts

AgentAvoidance

Το script που έχει κάθε πράκτορας για να αποφεύγει τους στόχους.

```
using TMPro;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using UnityEngine;

public class AgentAvoidance : BaseAgent
{
    //Μεταβλητή που αναπαριστά την ταχύτητα που θα κινείται δεξιά-αριστερά ο agent-player
    [SerializeField]
    private float speed = 50.0f;

    //Η προκαθορισμένη θέση που πρέπει αν εχει ο player κάθε φορά που επαναφέρεται
    [SerializeField]
    private Vector3 presetPosition = Vector3.zero;

    //Η μέγιστη θέση που μπορεί να φτάσει στα αριστερά
    [SerializeField]
    private Vector3 leftPosition = Vector3.zero;

    //Η μέγιστη θέση που μπορεί να φτάσει στα δεξιά
    [SerializeField]
    private Vector3 rightPosition = Vector3.zero;

    private TargetMoving targetMoving = null;


    //Η μεταβλητή αυτή αναπαριστά την επόμενη κατεύθυνση που θα κινηθούν
    private Vector3 moveTo = Vector3.zero;

    /*Πρέπει να γνωρίζω τη προηγούμενη θέση του agent κάθε φορά γιατί οι agents είναι έξυπνοι και αν
    συνηδητοποιήσουν οτι αν κάτσουν σε μια συγκεκριμένη θέση δεξιά η αριστερά θα έχουν παντα επιτυχία τότε
    θα κάθονται πάντα εκεί.Γιαυτο χρειάζομαστε αυτη τη πληροφορία ώστε αν κάτσουν στην ίδια θέση 2-3 φορές
    συνεχόμενα να τους τιμωρήσωγια να καταλάβουν οτι δεν τους επιτρέπεται αυτο.*/
    private Vector3 prevPosition = Vector3.zero;
```

```
void Awake()
{
    //Αρχικοποιήση της μεταβλητής targetMoving
    targetMoving = transform.parent.GetComponentInChildren<TargetMoving>();
}

public override void OnEpisodeBegin()
{
    //Η μέθοδος που ενεργοποιείται στην αρχή καποιου episode και πρέπει να τοποθετήσει τον agent στην προκαθορισμένη θέση
    //του και αρχικοποιεί την προηγούμενη θέση του και την κατεύθυνση που θα πρεπει να κινηθεί με την προκαθορισμένη θέση
    transform.localPosition = presetPosition;
    moveTo = prevPosition = presetPosition;
}

public override void CollectObservations(VectorSensor sensor)
{
    // 3 observations - x, y, z
    sensor.AddObservation(transform.localPosition);

    // 3 observations - x, y, z
    sensor.AddObservation(targetMoving.transform.localPosition);
}
```

```
public override void OnActionReceived(ActionBuffers actions)
{
    var vectorAction = actions.DiscreteActions.Array;

    prevPosition = moveTo; //ενημερώνω ποια ήταν η προηγούμενη θέση του agent
    int direction = Mathf.FloorToInt(vectorAction[0]); //βρίσκω ποια είναι η κατεύθυνση του agent αυτη τη στιγμή δηλ αν ειναι στη μέση,
                                                       //δεξιά η αριστερά γιαυτό κρατάω και το ακέραιο μέρος του.
    moveTo = presetPosition;

    switch (direction)
    {
        //ενημερώνω την κετεύθυνση που θα κινηθεί αναλογα με το direction
        case 0:
            moveTo = presetPosition;
            break;
        case 1:
            moveTo = leftPosition;
            break;
        case 2:
            moveTo = rightPosition;
            break;
    }
    //κινώ τον agent προς την κατεύθυνση που εχει το moveTo με την ταχυτητα που έχουμε επιλέξει
    transform.localPosition = Vector3.MoveTowards(transform.localPosition, moveTo, Time.fixedDeltaTime * speed);

    if (prevPosition == moveTo)
    {
        //το κανω για να βρώ ποσες φορές κάθεται συνεχόμενα στην ίδια θέση ωστε να τον τιμωρήσω
        punishCounter++;
    }

    if (punishCounter > 3.0f)
    {
        //Αν έχει κάτσει 3 φορές στην ίδια θέση συνεχόμενα τότε τιμωρώ τον agent
        AddReward(-0.01f);
        punishCounter = 0;
    }
}
```

```
public void TakeAwayPoints()
{
    //Η μέθοδος που τιμωρεί τον agent όποτε κριθεί οτι ειναι αναγκαίο
    AddReward(-0.01f);
    targetMoving.ResetTarget();

    EndEpisode();
    StartCoroutine(SwapGroundMaterial(failureMaterial, 0.5f));//ενημερώνουμε το material του εδάφους με κοκκινο καθως θα εχουμε ανεπιτυχία
}



public void GivePoints()
{
    //Η μέθοδος που επιβραβεύσει τον agent όποτε κριθεί οτι ειναι αναγκαίο
    AddReward(1.0f);
    targetMoving.ResetTarget();

    EndEpisode();
    StartCoroutine(SwapGroundMaterial(successMaterial, 0.5f));//ενημερώνουμε το material του εδάφους με πράσινο καθως θα εχουμε επιτυχία
}
```

```
public override void Heuristic(in ActionBuffers actionsOut)
{
    //Αν επιλέξω behavior type = heuristic only στον player τοτε με αυτή τη μέθοδο  θα έχω πλήρη
    //έλεγχο με βάση τα βελάκια που θα πατάω εγώ για να κινηθώ
    var descreteActions = actionsOut.DiscreteActions.Array;

    //Προκαθορισμένη θέση
    if (Input.GetKeyDown(KeyCode.DownArrow))
    {
        descreteActions[0] = 0;
    }

    //κίνηση αριστερά
    if (Input.GetKeyDown(KeyCode.LeftArrow))
    {
        descreteActions[0] = 1;
    }

    //κίνηση δεξιά
    if (Input.GetKeyDown(KeyCode.RightArrow))
    {
        descreteActions[0] = 2;
    }
}
```

TargetMoving

Το script που έχει ο κάθε στόχος για να μετακινείται προς τον πράκτορα.

```
using UnityEngine;
using UnityEngine.SocialPlatforms;

public class TargetMoving : MonoBehaviour
{
    [SerializeField]
    private AgentAvoidance agent = null;

    //Η ελάχιστη ταχύτητα του target θα κυμαίνεται μεταξύ 0.5 και 25.0
    [SerializeField]
    [Range(0.5f, 25.0f)]
    private float minSpeed = 5.0f;

    //Η μέγιστη ταχύτητα του target θα κυμαίνεται  μεταξύ 5.0 και 150.0
    [SerializeField]
    [Range(5.0f, 150.0f)]
    private float maxSpeed = 150.0f;

    //θα επιλέγεται τυχαία μεταξύ της ελάχιστης και μέγιστης ταχύτητας που μπορεί να έχει
    private float speed = 0;

    //η μέγιστη απόσταση που μπορεί να φτάσει μακρυά στον z άξονα το target
    [SerializeField]
    private float maxDistance = -3.5f;

    // Η default θέση που πρέπει να έχει το target κάθε φορά που επαναφέρεται
    private Vector3 originalPosition;

    private Quaternion originalRotation;
```

```
void Awake()
{
    //Η μέθοδος που ενεργοποιείται στην αρχή η οποία πρέπει να τοποθετήσει το target στην default
    //θέση του και του επιλέγει μια τυχαία ταχύτητα μεταξύ της μέγιστης και της ελάχιστης
    originalPosition = transform.localPosition;
    originalRotation = transform.localRotation;
    speed = Random.Range(minSpeed, maxSpeed);
}

void Update()
{
    // αν βρισκόμαστε πέρα από τη μέγιστη απόσταση επανεκκινούμε τη θέση του
    if (transform.localPosition.z <= maxDistance)
    {
        transform.localPosition = originalPosition;
        transform.localRotation = originalRotation;
    }
    else
    {
        // Το μετακινούμε προς τη μέγιστη απόσταση με την συγκεκριμενη ταχυτητα που εχει επιλεγεί
        transform.localPosition = new Vector3(transform.localPosition.x, transform.localPosition.y,
            transform.localPosition.z - (Time.deltaTime * speed));
    }
}

public void ResetTarget()
{
    //Η μεθοδος που επαναφέρει το target στην αρχική του θέση και ξανά επιλέγει τυχαία ταχύτητα
    speed = Random.Range(minSpeed, maxSpeed);
    transform.localPosition = originalPosition;
    transform.localRotation = originalRotation;
}
```

```
void OnCollisionEnter(Collision collision)
{
    if (collision.transform.tag.ToLower() == "player")
    {
        //Αν ο target κάνει collide με τον player τότε πρεπει να τιμωρήσουμε τον agent
        //οπότε καλούμε την μέθοδο του TakeAwayPoints
        Debug.Log("Points taken away");
        agent.TakeAwayPoints();
    }
    else if (collision.transform.tag.ToLower() == "wall")
    {
        //Αν ο target κάνει collide με τον τοίχο που είναι και το σωστό τότε πρεπει να
        //επιβραβεύσουμε τον agent οπότε καλούμε την μέθοδο του GivePoints
        Debug.Log("Points gained");
        agent.GivePoints();
    }
}
```

BaseAgent

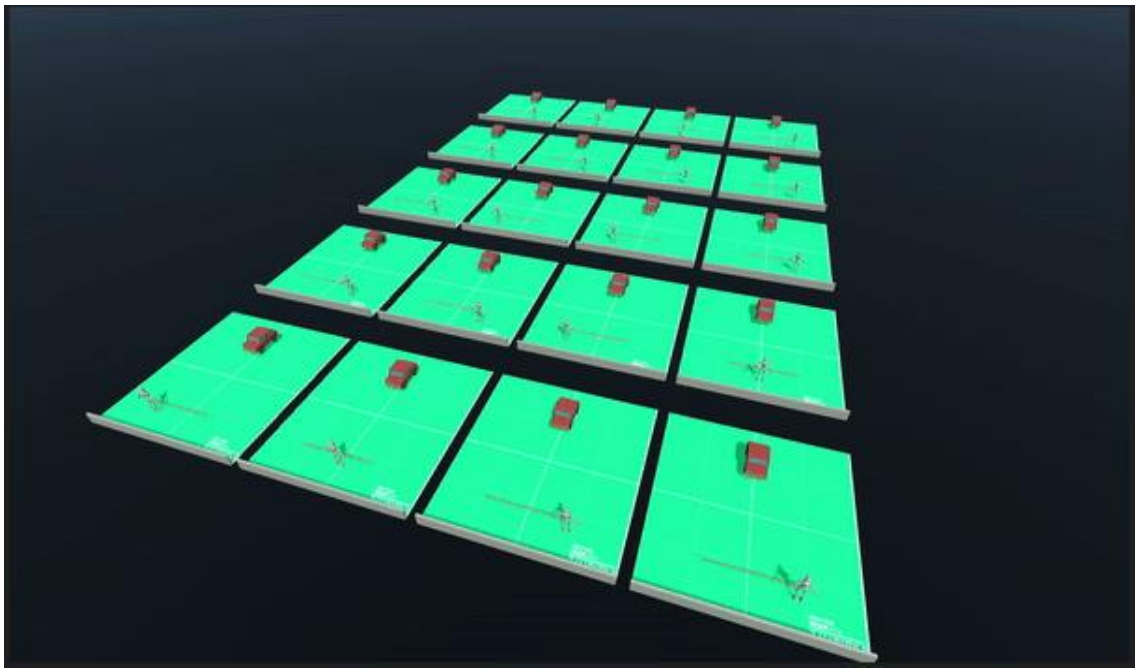
Κλάση που έχει φτιαχτεί για να κληρονομεί η AgentAvoidance και να ρυθμίζει τις αλλαγές στο material του εδάφους κάθε φορά που έχουμε επιτυχία η αποτυχία.

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


public class BaseAgent : Agent
{
    //Κλάση που εχει φτιαχτεί για να κληρονομεί η AgentAvoidance και να ρυθμίζει τις αλλαγές
    //στο material του εδάφους κάθε φορά που έχουμε επιτυχία η αποτύχια.
    [SerializeField]
    protected MeshRenderer groundMeshRenderer;

    [SerializeField]
    protected Material successMaterial;

    [SerializeField]
    protected Material failureMaterial;

    [SerializeField]
    protected Material defaultMaterial;

    protected IEnumerator SwapGroundMaterial(Material mat, float time)
    {
        //Η μέθοδος που αλλάζει το material του εδάφους και περιμένει μερικά δευτερόλεπτα
        //για να επαναφέρει μετά στο αρχικό default material
        groundMeshRenderer.material = mat;
        yield return new WaitForSeconds(time);
        groundMeshRenderer.material = defaultMaterial;
    }
}
```

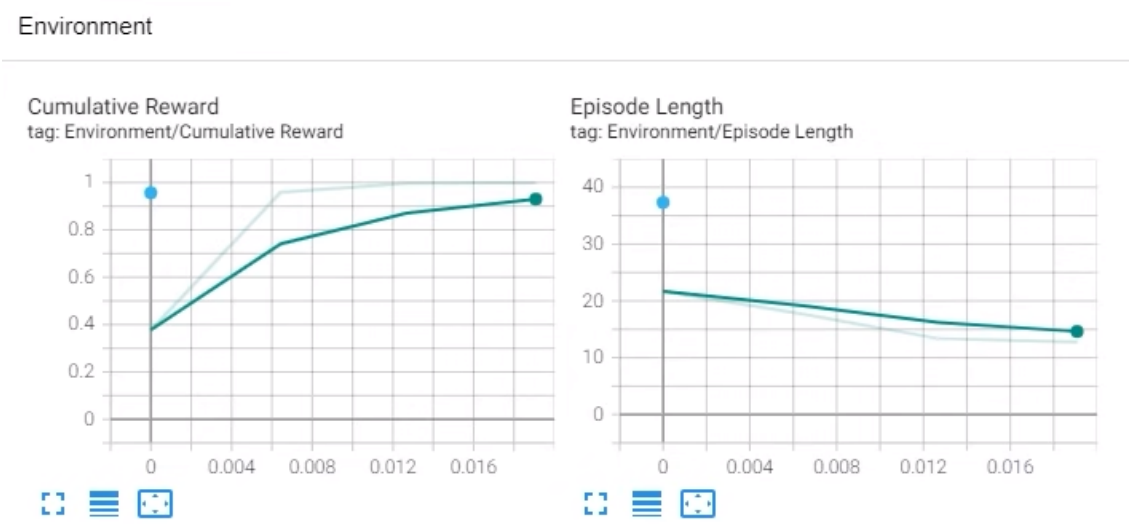
5. Ολοκληρωμένη περιγραφή μίας παραδειγματικής εκτέλεσης και των αποτελεσμάτων της

Παράδειγμα 1

Εδώ βλέπουμε ένα παράδειγμα εκτέλεσης του παιχνιδιού μας με το εκπαιδευμένο μοντέλο που έχουν οι πράκτορες και με χαμηλή και σταθερή ταχύτητα σε 10 για όλους τους στόχους καθώς έχουμε επιλέξει της ελάχιστη και μέγιστη ταχύτητα τους να είναι και στα δυο 10 ώστε να δούμε πιο ξεκάθαρα τα αποτελέσματα και ότι έχουν εκπαιδευτεί τόσο καλά οπού με μια χαμηλή ταχύτητα αποφεύγουν κατά 99% τους στόχους.

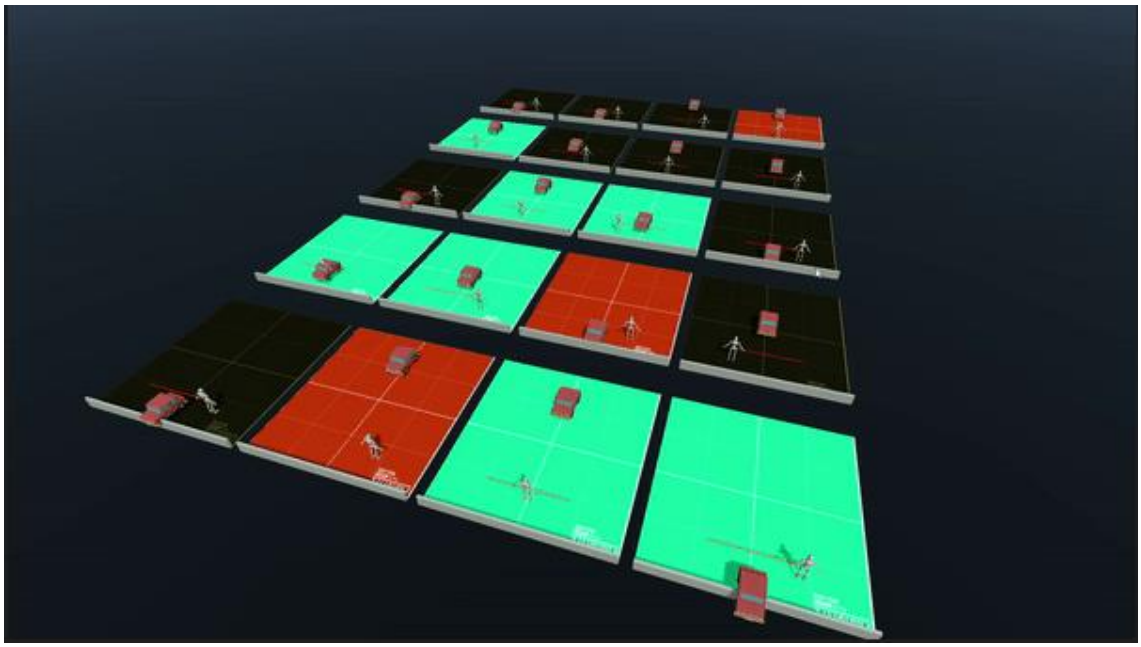


Παρακάτω βλέπουμε και τα στατιστικά από την διαδικασία εκπαίδευσης του παιχνιδιού μας δηλαδή την αύξηση του cumulative reward ανά episode καθώς και το episode length που μειώνεται που σημαίνει ότι οι πράκτορες αποφεύγουν όλο και ταχύτερα τους στόχους.



Παράδειγμα 2

Εδώ βλέπουμε ένα ακόμα παράδειγμα εκτέλεσης του παιχνιδιού μας με το εκπαιδευμένο μοντέλο που έχουν οι πράκτορες και με ταχύτητα να επιλέγεται τυχαία μεταξύ 12 (που είναι η ελάχιστη ταχύτητα) και 110 (που είναι η μέγιστη) όλους τους στόχους. Παρατηρούμε ότι ακόμα και τώρα που αρκετοί στόχοι έχουν υψηλή ταχύτητα, οι πράκτορες μας αποφεύγουν σε ικανοποιητικό επίπεδο τους στόχους.



6. Περιγραφή πρόσθετων δυνατοτήτων της εφαρμογής, εάν υπάρχουν

Μερικές ιδέες για πρόσθετες δυνατότητες στην εφαρμογή του παιχνιδιού εκπαίδευσης ευφυών πρακτόρων στο περιβάλλον της Unity, είναι οι εξής:

- **Πολυπλοκότητα περιβάλλοντος**: Μπορούμε να εισάγουμε περισσότερα στοιχεία και εμπόδια στο περιβάλλον, ώστε να αυξήσετε την πολυπλοκότητα του παιχνιδιού. Για παράδειγμα, μπορείτε να προσθέσουμε εμπόδια που κινούνται ή περιβάλλοντα με πολλές διαφορετικές επιφάνειες.

- **Ανανεώσιμες πηγές ενέργειας**: Μπορούμε να εισάγουμε ανανεώσιμες πηγές ενέργειας για τον πράκτορα μας (το ρομπότ). Αυτό θα μπορούσε να περιλαμβάνει την ανάκτηση ενέργειας από διάφορα στοιχεία στο περιβάλλον. Ο πράκτορας θα χρειαστεί να προσαρμόσει τη στρατηγική του για την αποφυγή της έλλειψης ενέργειας και την αποδοτική χρήση των πηγών ενέργειας.

- **Ποικιλία στόχων**: Αντί για ένα μόνο αυτοκίνητο-στόχο, μπορούμε να προσθέσουμε πολλαπλούς στόχους με διάφορες ιδιότητες και αναμενόμενη συμπεριφορά. Αυτό θα προκαλέσει μια πιο πολύπλοκη συμπεριφορά από τον πράκτορα καθώς θα πρέπει να αποφασίζει ποιον στόχο να επιλέξει και πώς να αντιδράσει σε κάθε έναν από αυτούς.

- **Πολλαπλοί πράκτορες**: Μπορούμε να εξετάσουμε την προσθήκη πολλαπλών πρακτόρων στο παιχνίδι. Κάθε πράκτορας θα είναι υπεύθυνος για την αποφυγή της σύγκρουσης με το αυτοκίνητο-στόχο και θα μπορεί να αλληλεπιδρά με τους άλλους πράκτορες στο περιβάλλον.

Αυτές είναι μερικές ιδέες που θα μπορούσαμε να εξερευνήσουμε για πρόσθετες δυνατότητες στην εφαρμογή μας.

7. Αναλυτική περιγραφή της συμβολής του κάθε μέλους της ομάδας

Κάθε μέλος της ομάδας απέδωσε τον ίδιο χρόνο, την ίδια προσπάθεια και την ίδια αφοσίωση για την εκτέλεση και την δημιουργία του τελικού αποτελέσματος της εργασίας. Πιο αναλυτικά:

- Ο **Παναγιωτόπουλος Δημήτριος** βρήκε τα assets που χρειάστηκαν για να κατασκευάσουμε την ιδέα (που σκεφτήκαμε όλοι μαζί) και δημιούργησε όλο το περιβάλλον τοποθετώντας όλα τα αντικείμενα στο χώρο δηλαδή το έδαφος(ground), τον τοίχο(wall) , το ρομπότ που χρησιμοποιήσαμε για πράκτορα καθώς και τον στόχο (το αυτοκίνητο).
- Ο **Αυγερινος Χρηστος** ανέλαβε ότι έχει να κάνει με τον πράκτορα, έγραψε το script *AgentAvoidance* που χρειάστηκε για τον παίκτη για να ρυθμίσουμε την συμπεριφορά του (τιμωρώντας η επιβραβραβευοντας τον οπότε χρειαζόταν) ώστε να αποφεύγει τους στόχους κινώντας τον δεξια-αριστερα στον χώρο καθώς και ρύθμισε όλες τις παραμέτρους ώστε να λειτουργεί άρτια με όσο το δυνατόν υψηλό ποσοστό μέσης ανταμοιβής(mean reward).
- Ο **Βιτάκης Αθανάσιος** ανέλαβε ότι έχει να κάνει με τον στόχο(αυτοκίνητο), έγραψε το script *TargetMoving* που χρειάστηκε για τον στόχο ώστε να κινείται προς τον πράκτορα με σκοπό να τον πέτυχει ρυθμίζοντας όλες τις κατάλληλες παραμέτρους που χρειάστηκαν για τον στόχο καθώς και το script *BaseAgent* ώστε να ρυθμίζουμε τις αλλαγές στο material του εδάφους στο τέλος κάθε episode ανάλογα αν έχουμε επιτυχία η αποτυχία αποφυγής του πράκτορα.

Η διαδικασία την εκπαίδευσης και ελέγχου για το αν το παιχνίδι λειτουργεί με σωστή εκπαίδευση των πρακτόρων και με υψηλό mean reward για οποιαδήποτε ταχύτητα των στόχων και των πρακτόρων στο χώρο καθώς και την διόρθωση τυχών σφαλμάτων η bugs έγινε από όλα τα μέλη της ομάδας.

8. Αναλυτική περιγραφή ανοικτών θεμάτων, ανεπίλυτων προβλημάτων και πιθανοτήτων εμφάνισης σφαλμάτων κατά την εκτέλεση

Σε γενικές γραμμές δεν έχουν ανιχνευθεί θέματα ή προβλήματα που παραμένουν ανοικτά και δεν έχουν επιλυθεί πλήρως στην εφαρμογή μας. Το μόνο πρόβλημα που παρατηρήθηκε κατά την διαδικασία εκπαίδευσης των πρακτόρων ήταν ότι αν βάζαμε τους στόχους να έχουν πολύ υψηλή ταχύτητα για παράδειγμα 500 δεν προλαβαίναμε να δούμε αν ο στόχος πέτυχε τον πράκτορα η τον τοίχο ούτε τις αλλαγές στο material του εδάφους (αν δηλαδή είχαμε επιτυχία η αποτυχία) και είχαμε αρνητικές τιμές στο mean reward γιατρό ορίσαμε μια ελάχιστη και μέγιστη ταχύτητα στον στόχο με την τελική ταχύτητα του κάθε στόχου να επιλέγεται τυχαία μεταξύ της ελάχιστης και μέγιστης.