НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет Программной Инженерии и Компьютерной Техники

Нейроинтерфейсы информационных систем

«Парковка с использованием ИИ в Unity: автоматическая парковка использует набор инструментов Unity ML-Agents, реализующий алгоритмы нейронной сети»

Выполнил студент

Чан Дык Зюи

Группа № P33202

Преподаватель: Штенников Д. Г.

г. Санкт-Петербург

2024

Содержание

Цель работы: Использование набор инструментов Unity ML-Agents для развертывания алгоритмов нейронных сетей.

Программа работы:

Шаг 1: Настройка среды разработки

Установка Unity и ML-Agents Toolkit:
• Загрузим и установим Unity Hub и Unity Editor с официального сайта Unity.
• Установим Unity ML-Agents Toolkit в соответствии с инструкциями из официальной документации ML-Agents.
Создание нового проекта:
• Откроем Unity Hub, создадим новый проект и выберем шаблон 3D.
Установка необходимых пакетов:
• Откроем в Unity "Window" -> "Package Manager" и установим такие пакеты, как ML-Agents, Input System и другие необходимые пакеты.

Шаг 2: Создание симуляционной среды

Построение среды:
• Создадим новую сцену и спроектируем парковочную зону, включая стены, препятствия и парковочные места.
Добавление автомобиля:
• Создадим или загрузим модель автомобиля из Unity Asset Store или другого источника.
• Добавим автомобиль в сцену и настроим такие компоненты, как Rigidbody и Collider.

Шаг 3: Создание агента и академии

Создание агента:
• Создадим GameObject для агента (автомобиля) и добавим компонент Agent из ML-Agents.
• Создадим C# скрипт, наследующий от класса Agent, и реализуем методы, такие как CollectObservations, OnActionReceived и Heuristic.
Создание академии:
• Создадим GameObject для академии и добавим компонент Academy.

Шаг 4: Определение целей и вознаграждений

Определение целей:
• Определим парковочные места на сцене и сделаем их целями для агента.
Определение вознаграждений:
• Определим вознаграждения за действия, такие как приближение к парковочному месту, правильная парковка, и штрафы за столкновение с препятствиями или неправильную парковку.

Шаг 5: Обучение модели

Настройка обучения:
• Создадим файл конфигурации .yaml для настройки параметров обучения, таких как learning rate, максимальное количество шагов и другие параметры.

Запуск обучения:
• Используем команду mlagents-learn в терминале для запуска процесса обучения.
• Будем наблюдать за процессом обучения и при необходимости корректировать параметры.

Шаг 6: Тестирование и доработка

Тестирование модели:
• После завершения обучения, протестируем модель, запустив симуляцию в Unity.
• Понаблюдаем за поведением агента и оценим его эффективность.
Доработка модели:
• На основе результатов тестирования, скорректируем параметры обучения, вознаграждения и компоненты агента для улучшения эффективности.

Шаг 7: Развертывание и публикация

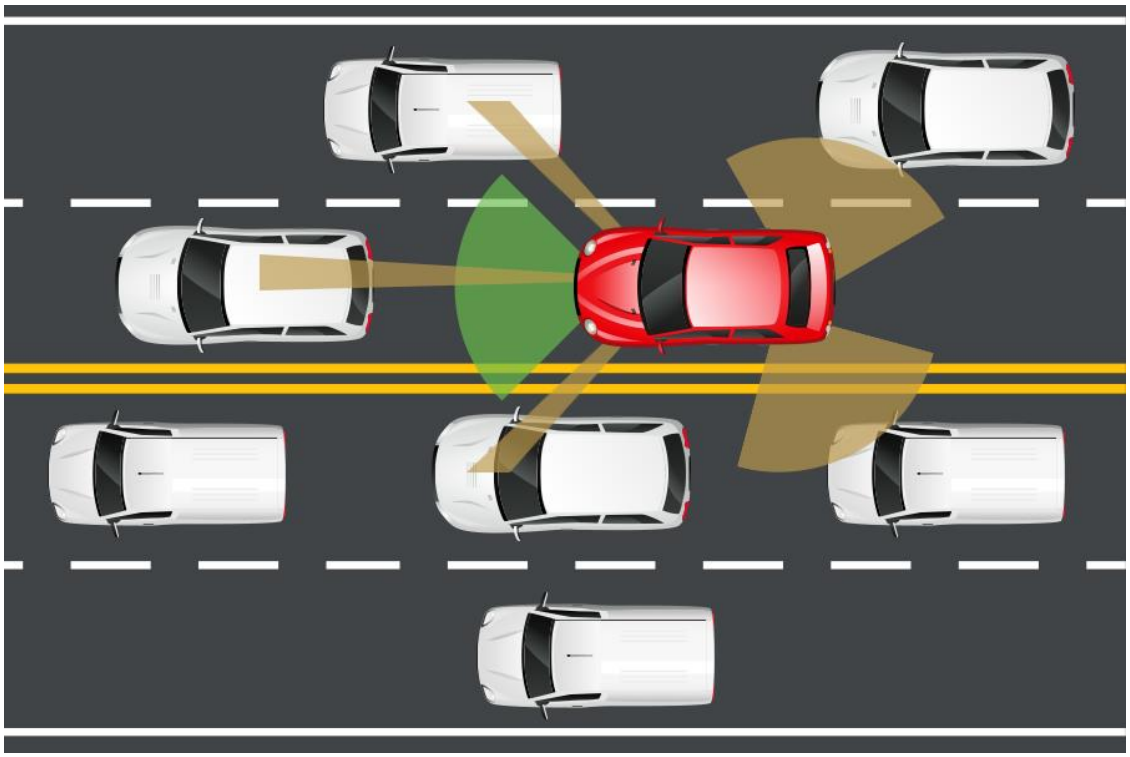
Развертывание проекта:
• Упакуем проект в отдельное приложение или опубликуем на таких платформах, как Unity Asset Store или GitHub.
Написание документации и руководства:
• Напишем документацию по использованию и установке проекта для других пользователей.

Отчет:

I. Исследование

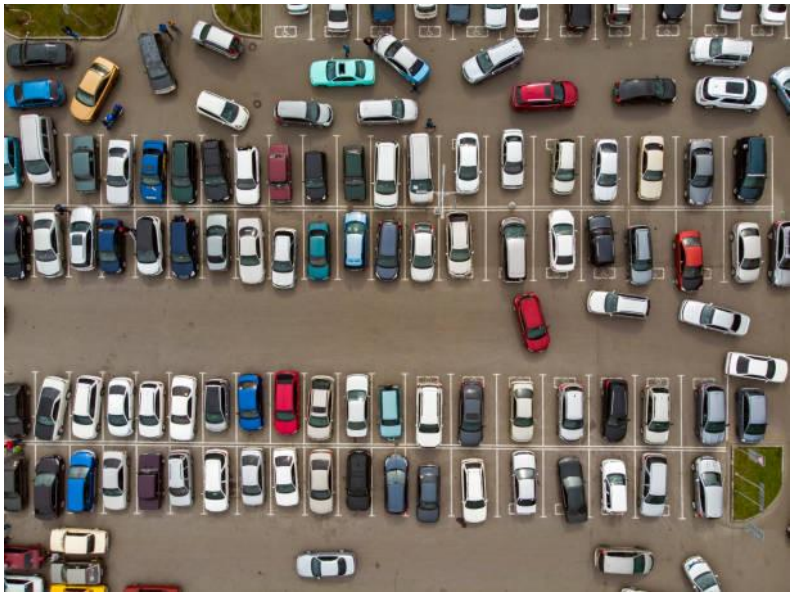
1. Проблема **– предотвращение столкновений**

- Существующие системы беспилотного вождения используют датчики для предотвращения столкновений.
- В будущем все автомобили могут быть автономными и общаться напрямую.
- Можем ли мы использовать такую связь для повышения безопасности парковки?



2. Проблема **– парковка**

- Автостоянки – это плотная, опасная среда с множеством сценариев.
- Естественно, мы можем смоделировать автостоянку как мультиагентную систему и использовать RL.
- Исследовать влияние общения, конкуренции и сотрудничества.



3. Задний план

- Автономное вождение — активная область исследований и приложений.
- «Deep RL позволяет решать мультиагентные задачи реальной сложности» [1]
- Недавние работы по теме (21 год):
- Моделирование самостоятельной парковки автомобиля с использованием RL и Unity [2]
- Динамическое предотвращение препятствий и планирование пути на основе RL [3]
- Удивительная эффективность PPO в кооперативных многоагентных играх [4]
- Мультиагентный RL: опрос [1]
- Современная работа, ориентированная на будущее
- Предполагает группы автономных автомобилей.
- Предполагает технологические возможности

4. Мотивация

Благодаря этой работе мы можем:

- Лучше понять производительность и ограничения систем самостоятельной парковки на основе RL и их технологические требования.
- Определение поведенческих моделей в системах самостоятельной парковки на основе RL.
- Оценка социальной совместимости систем самостоятельной парковки на основе RL.
- Повышение безопасности и эффективности парковки
- Улучшенный дизайн систем самостоятельной парковки

5. Метод

- Моделирование агентов с помощью MDP
- Реализация MDP с использованием Unity для моделирования
- Обучение и оценка алгоритмов RL с помощью Python и Unity
- Использование алгоритма PPO для обучения RL
- Использование системы пакетных вычислений Warwick для масштабного обучения

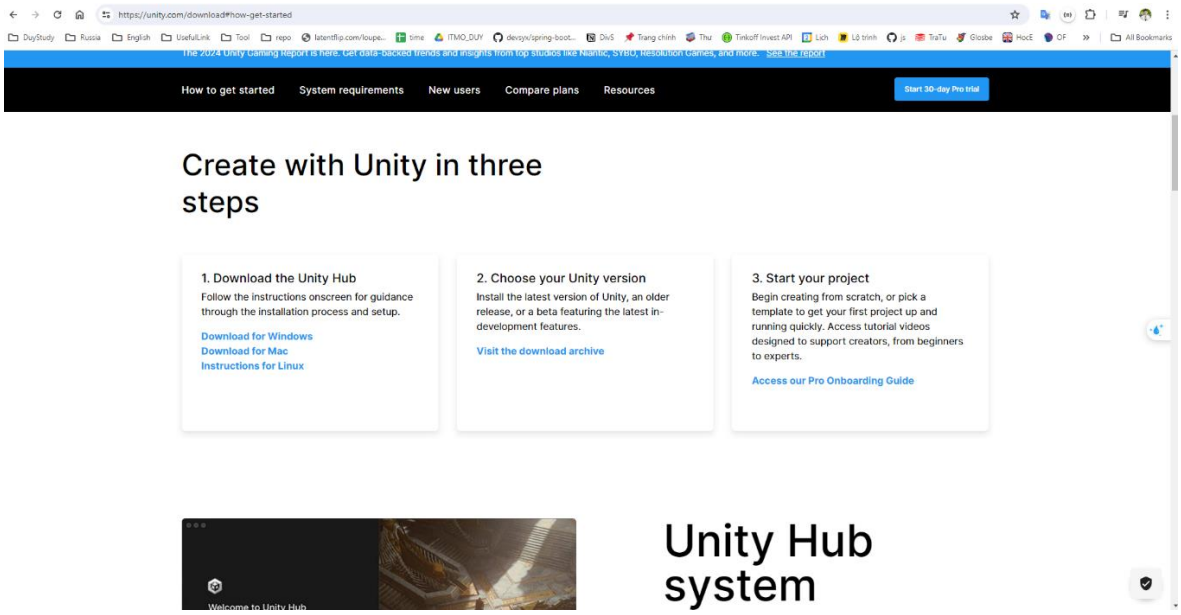
6. МДП

- Довольно стандартный базовый MDP, используемый для робототехники и беспилотного вождения.
- Расширение нашей проблемы:
    - Обмен/отслеживание состояния: знайте состояние фиксированного количества близлежащих автомобилей для предотвращения столкновений и планирования.
    - Выбор и сообщение целей – например, указание
    - Награды за участие в конкурсах и сотрудничестве.
    - Различные функции безопасности…

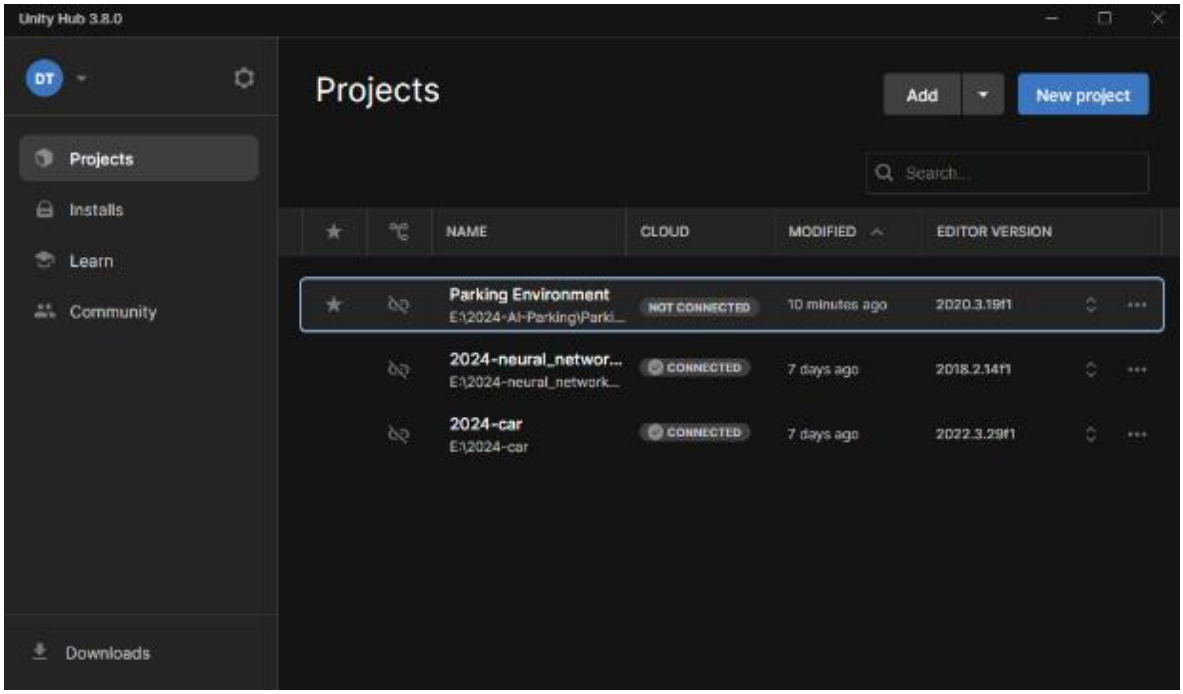
II. Выполнение

Шаг 1: Настройка среды разработки

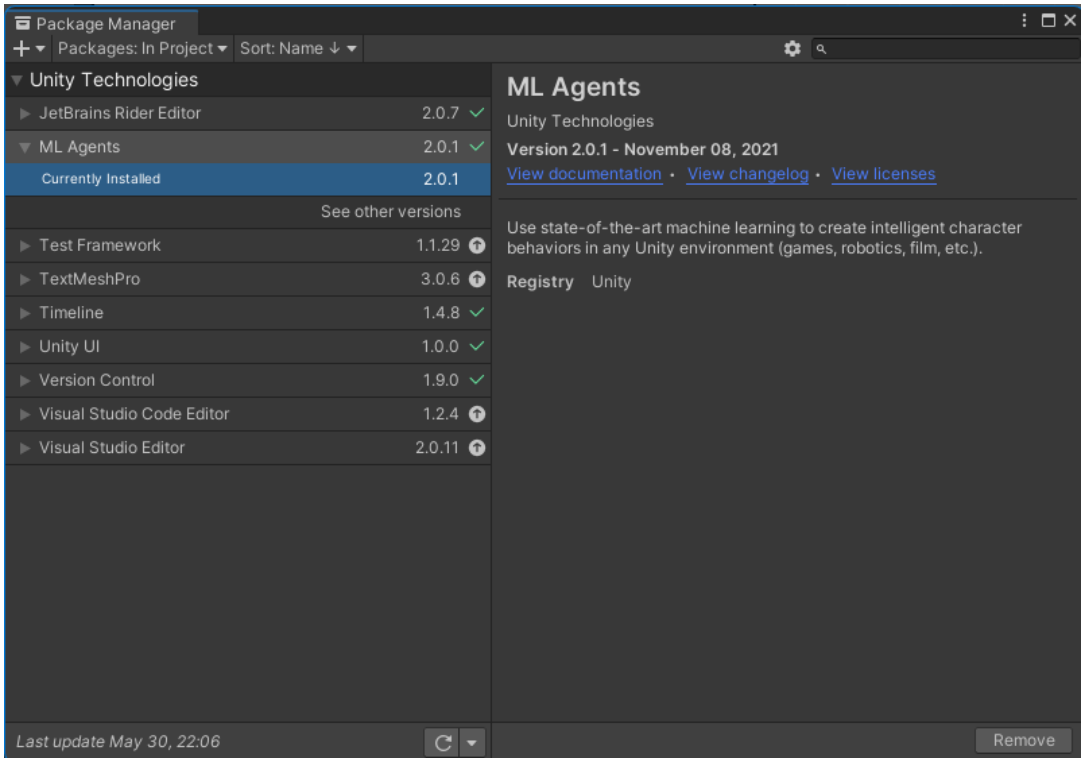
1. **Установка Unity и ML-Agents Toolkit**:



2. **Создание нового проекта:**

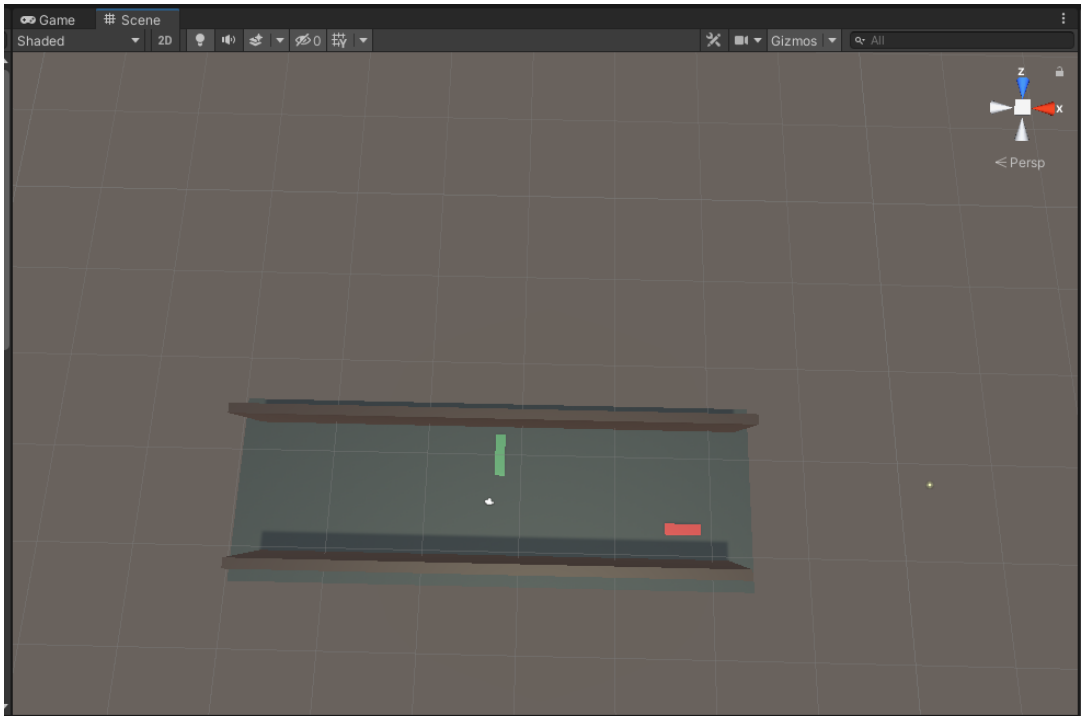


3. **Установка необходимых пакетов**:

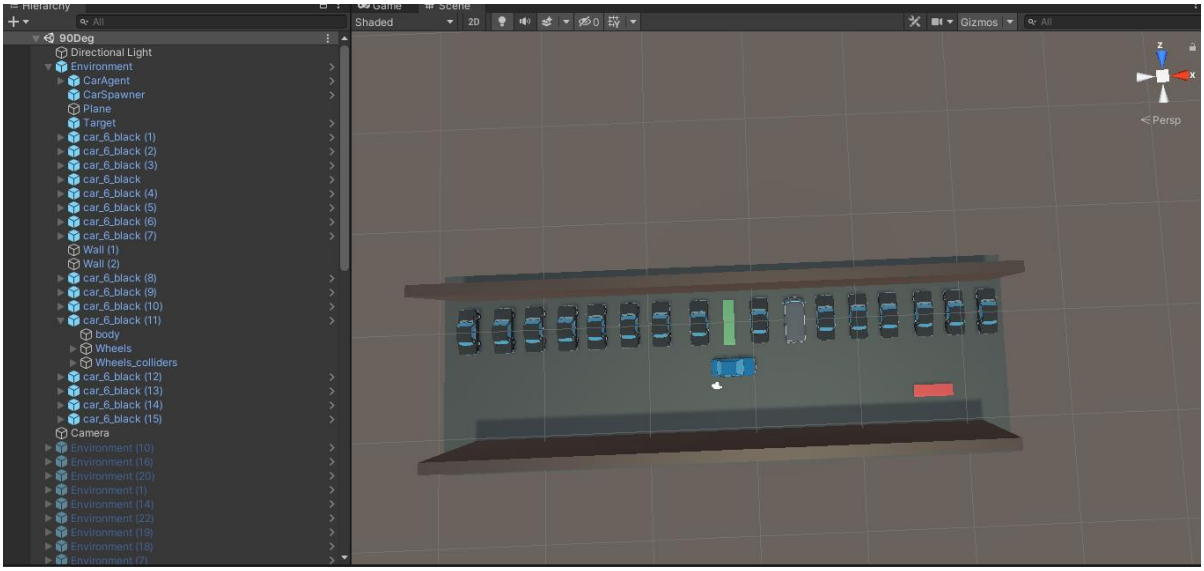


Шаг 2: Создание симуляционной среды

1. **Построение среды**:



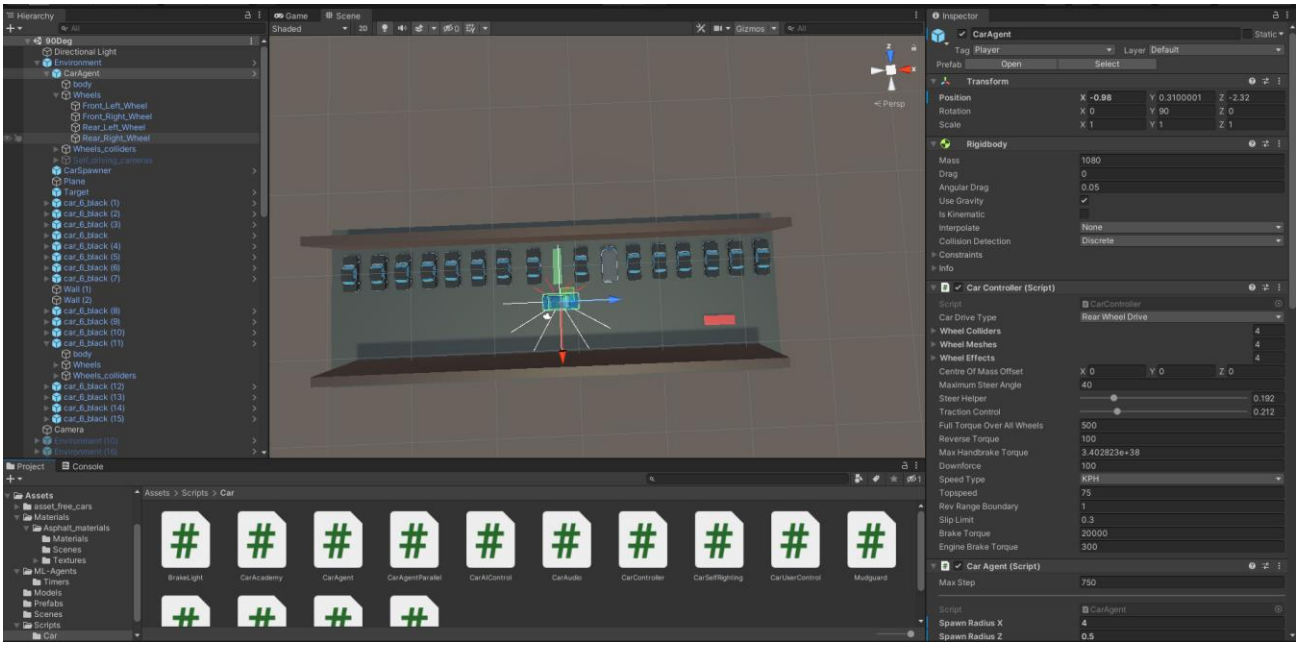
2. **Добавление автомобиля**:



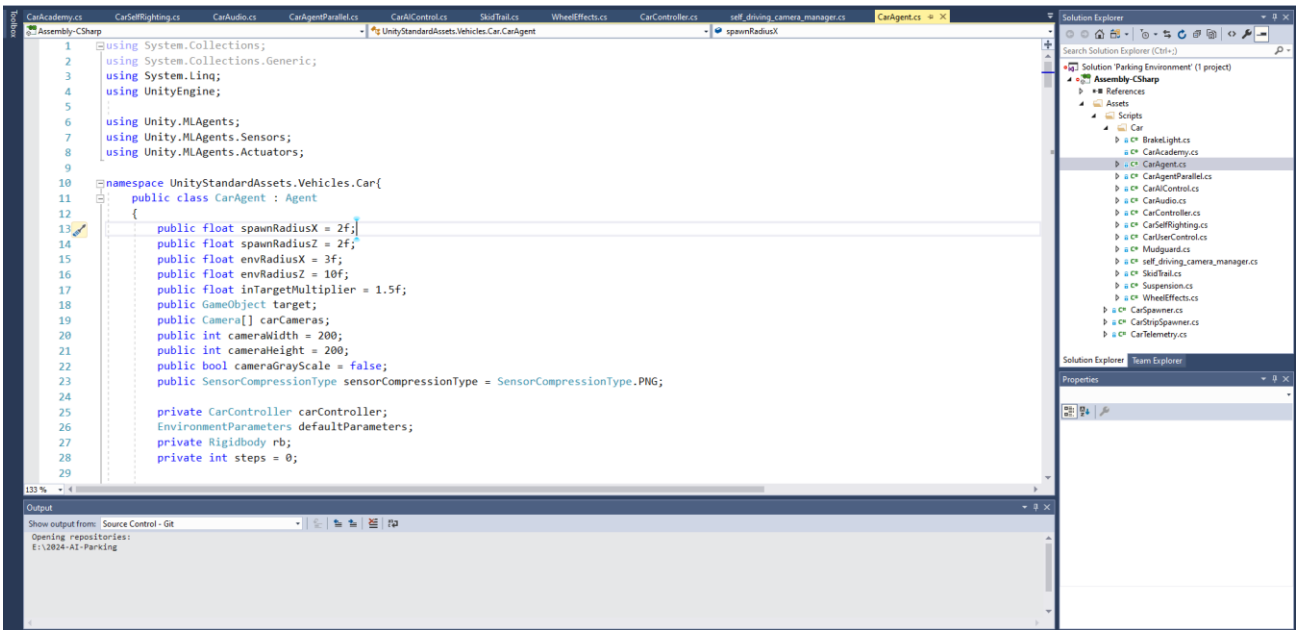
Шаг 3: Создание агента и академии

1. **Создание агента**:

    - Создадим GameObject для агента (автомобиля) и добавим компонент Agent из ML-Agents.



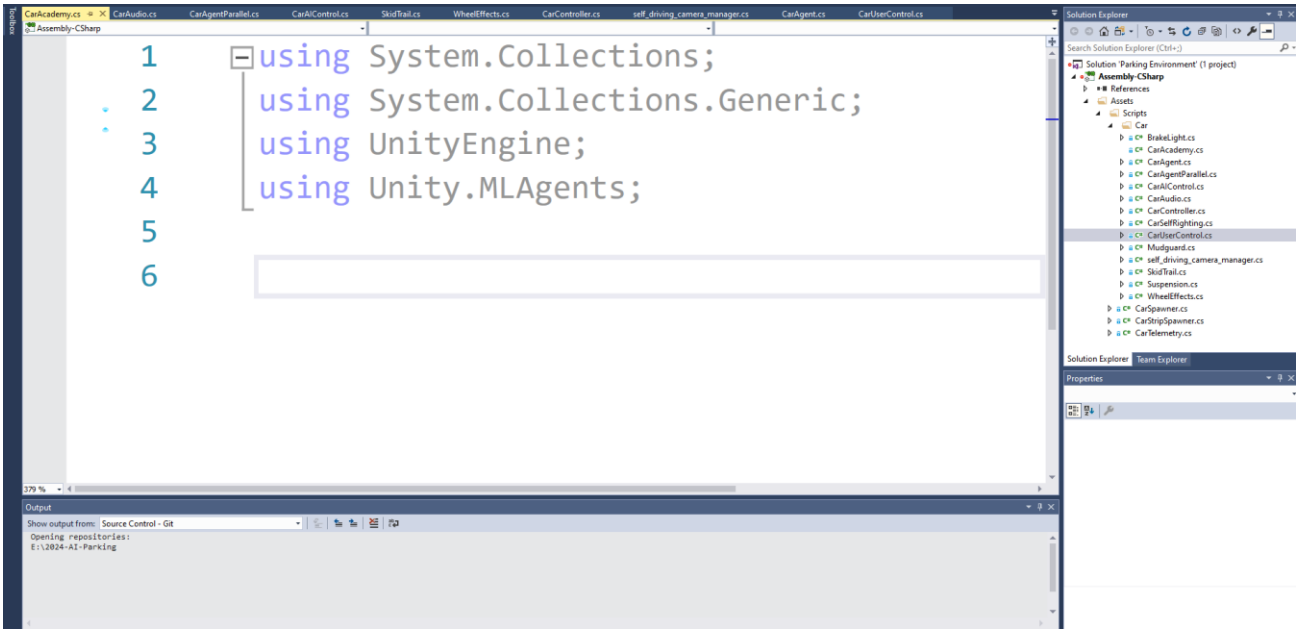
- Создадим C# скрипт, наследующий от класса **Agent**, и реализуем методы, такие как **CollectObservations**, **OnActionReceived** и **Heuristic**.



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

using Unity.MLAgents;
using Unity.MLAgents.Sensors;
using Unity.MLAgents.Actuators;

namespace UnityStandardAssets.Vehicles.Car{
    public class CarAgent : Agent
    {
        public float spawnRadiusX = 2f;
        public float spawnRadiusZ = 2f;
        public float envRadiusX = 3f;
        public float envRadiusZ = 10f;
        public float inTargetMultiplier = 1.5f;
        public GameObject target;
        public Camera[] carCameras;
        public int cameraWidth = 200;
        public int cameraHeight = 200;
        public bool cameraGrayScale = false;
        public SensorCompressionType sensorCompressionType = SensorCompressionType.PNG;

        private CarController carController;
        EnvironmentParameters defaultParameters;
        private Rigidbody rb;
        private int steps = 0;
```

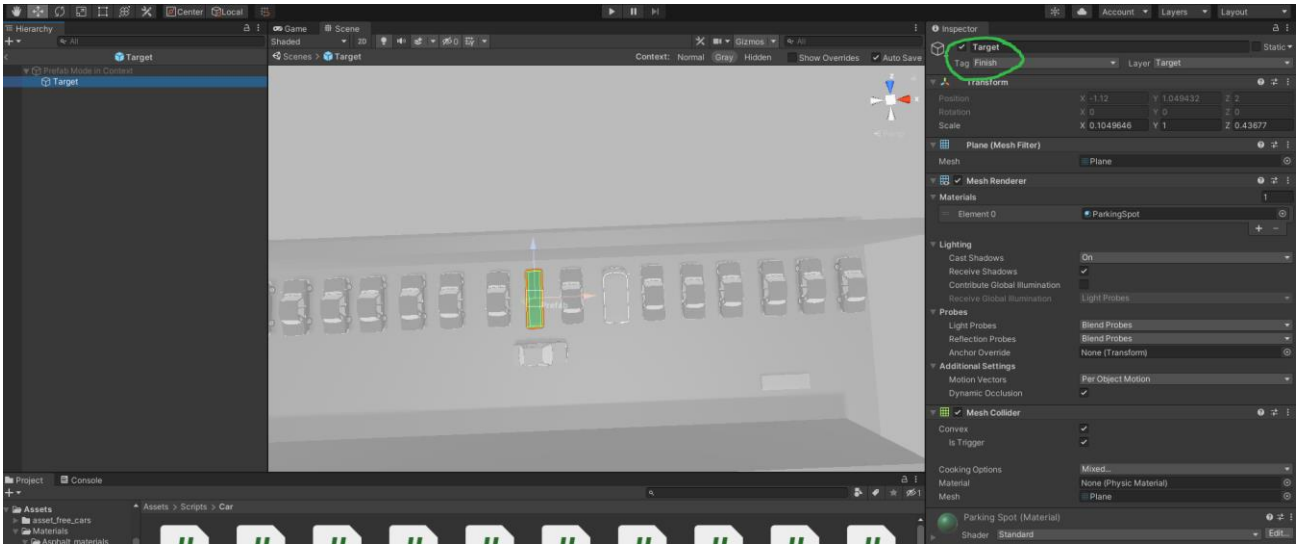
2. **Создание академии**:

- Создадим GameObject для академии и добавим компонент **Academy**.



Шаг 4: Определение целей и вознаграждений

1. **Определение целей**:
    - Определим парковочные места на сцене и сделаем их целями для агента.



2. **Определение вознаграждений**:
    - Определим вознаграждения за действия, такие как приближение к парковочному месту, правильная парковка, и штрафы за столкновение с препятствиями или неправильную парковк

```
    private void FindParkingSpot(){
        var RpMeasurements = RayPerceptionMeasurements();

        int LeftLikelihoodScore = 0;
        int RightLikelihoodScore = 0;

        // First check if the left and right perpendicular sensors are detecting a long range
        if(RpMeasurements.RDistL[2] > 0.9f){
            LeftLikelihoodScore += 1;
        }
        if(RpMeasurements.RDistR[2] > 0.9f){
            RightLikelihoodScore += 1;
        }

        // Sum the distances of the sensors for each side to check which side is just seeing space
        if(RpMeasurements.RDistL.Sum() < RpMeasurements.RDistR.Sum()){
            LeftLikelihoodScore += 1;
        }
        else{
            RightLikelihoodScore += 1;
        }

        // Check if sensor observations are symmetrical. This indicates the agent is in the middle of the parking space
        float RayDiff1 = Mathf.Abs(RpMeasurements.RDistL[0] - RpMeasurements.RDistL[4]);
        float RayDiff2 = Mathf.Abs(RpMeasurements.RDistL[1] - RpMeasurements.RDistL[3]);
        float TotalRayDiff = RayDiff1 + RayDiff2;

        if(TotalRayDiff < 0.1f){
            LeftLikelihoodScore += 1;
        }

        RayDiff1 = Mathf.Abs(RpMeasurements.RDistR[0] - RpMeasurements.RDistR[4]);
        RayDiff2 = Mathf.Abs(RpMeasurements.RDistR[1] - RpMeasurements.RDistR[3]);
        TotalRayDiff = RayDiff1 + RayDiff2;

        if(TotalRayDiff < 0.1f){
            RightLikelihoodScore += 1;
        }
```

```
    }

    // Can be refactored
    // If one of the sides met all the requirements, the agent is in the middle of a space
    if(LeftLikelihoodScore == 3){
        // Validate the spot is lage enough
        float PredictedSpace = (RpMeasurements.RDistL[1] * Mathf.Cos(60*Mathf.Deg2Rad)) + (RpMeasurements.RDistL[3] * Mathf.Cos(60*Mathf.Deg2Rad));
        // Distances are in normalised units, so multiply by the ray length to get the actual distance
        PredictedSpace *= 7;


        if(PredictedSpace > 3f){
            isLookingForSpot = false;
            isPositioning = true;
            predictedSpotSize = PredictedSpace;
            detectedSpotLocation = new Vector3(transform.position.x, transform.position.y, transform.position.z);
            Debug.Log("Found spot left");
        }
    }
    else if(RightLikelihoodScore == 3){
        // Validate the spot is lage enough
        float PredictedSpace = (RpMeasurements.RDistR[1] * Mathf.Cos(60*Mathf.Deg2Rad)) + (RpMeasurements.RDistR[3] * Mathf.Cos(60*Mathf.Deg2Rad));
        // Distances are in normalised units, so multiply by the ray length to get the actual distance
        PredictedSpace *= 7;

        if(PredictedSpace > 3f){
            isLookingForSpot = false;
            isPositioning = true;
            predictedSpotSize = PredictedSpace;
            detectedSpotLocation = new Vector3(transform.position.x, transform.position.y, transform.position.z);
            Debug.Log("Found spot right");
        }
    }

    else{
        isLookingForSpot = true;
    }

}
```

Шаг 5: Обучение модели

1. **Настройка обучения**:

- Создадим файл конфигурации **.yaml** для настройки параметров обучения, таких как learning rate, максимальное количество шагов и другие параметры.

```yaml
behaviors:
  CarBehaviour:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 5120
      learning_rate: 0.00035
      beta: 0.0025
      epsilon: 0.3
      lambd: 0.95
      num_epoch: 5
      learning_rate_schedule: linear

    network_settings:
      normalize: true
      hidden_units: 264
      num_layers: 3

    reward_signals:
      extrinsic:
        gamma: 0.95
        strength: 0.99
      gail:
        strength: 0.3
        demo_path: Demos/90dgRS3s.demo
        use_actions: false

    behavioral_cloning:
      demo_path: Demos/90dgRS3s.demo
      steps: 750000
      strength: 0.4

    keep_checkpoints: 15
    checkpoint_interval: 1000000
    time_horizon: 264
    max_steps: 50000000
    summary_freq: 100000
    threaded: true

  CarBehaviourSAC:
    trainer_type: sac
    hyperparameters:
      batch_size: 128
      buffer_size: 50000
```

```
    learning_rate: 0.00035
    buffer_init_steps: 5000
    init_entcoef: 0.5
    steps_per_update: 1
    learning_rate_schedule: constant

  network_settings:
    normalize: true
    hidden_units: 128
    num_layers: 3

  reward_signals:
    extrinsic:
      gamma: 0.75
      strength: 0.99
    gail:
      strength: 0.3
      demo_path: Demos/90dgRev3s.demo
      use_actions: false

  behavioral_cloning:
    demo_path: Demos/90dgRev3s.demo
    steps: 300000
    strength: 0.3

  keep_checkpoints: 15
  time_horizon: 264
  max_steps: 30000000
  summary_freq: 20000
  threaded: false
```

CarBehaviour (PPO)

- **trainer_type: ppo**
    - Выбор алгоритма обучения. PPO выбран за его стабильность и эффективность при обучении сложных политик.
- **hyperparameters**
    - **batch_size**: Размер batch определяет количество образцов данных, используемых при каждом обновлении весов. Большой размер batch (1024) помогает лучше усреднять градиенты и уменьшать шум в процессе обучения.
    - **buffer_size**: Размер буфера определяет количество сохраняемых опытов перед обновлением. Большой буфер (5120) позволяет хранить больше опытов, помогая модели учиться в различных ситуациях.

- **learning_rate**: Скорость обучения определяет степень изменения весов при каждом обновлении. Низкое значение (0.00035) делает процесс обучения медленным и стабильным, избегая быстрого и нестабильного обучения.
- **beta**: Коэффициент энтропии помогает поддерживать разнообразие в действиях агента, избегая излишней уверенности в своих действиях и потери способности к исследованию.
- **epsilon**: Этот параметр определяет степень допустимых изменений политики в PPO. Большие значения позволяют больше изменений, меньшие поддерживают стабильность.
- **lambd**: Этот параметр влияет на точность Generalized Advantage Estimation (GAE). Значение 0.95 часто выбирается для баланса между точностью и стабильностью.
- **num_epoch**: Количество проходов по всем данным при каждом обновлении помогает эффективно оптимизировать веса.
- **learning_rate_schedule**: График изменения скорости обучения. Линейный график позволяет скорости обучения постепенно снижаться со временем, что делает процесс обучения медленнее при приближении к оптимуму.

- **network_settings**
  - **normalize**: Нормализация входных данных помогает стабилизировать процесс обучения, уменьшая изменчивость входных данных.
  - **hidden_units**: Количество нейронов в каждом скрытом слое определяет сложность нейронной сети. 264 скрытых нейрона обеспечивают достаточную способность для обучения сложным шаблонам.
  - **num_layers**: Количество скрытых слоев определяет глубину нейронной сети. 3 скрытых слоя достаточно для моделирования сложных зависимостей, не усложняя при этом модель.
- **reward_signals**
  - **extrinsic**:
    - **gamma**: Коэффициент дисконтирования помогает модели фокусироваться на краткосрочных или долгосрочных наградах. Gamma 0.95 балансирует между краткосрочными и долгосрочными наградами.
    - **strength**: Сила внешнего сигнала награды. 0.99 делает внешние награды значимыми для процесса обучения.
  - **gail**:
    - **strength**: Сила сигнала награды GAIL помогает агенту учиться на предоставленных демо.

        - **demo_path**: Путь к демо данным, предоставляющим примеры желаемого поведения.
        - **use_actions**: Определяет, использовать ли действия из демо в процессе обучения.
    - **behavioral_cloning**
        - **demo_path**: Путь к демо данным помогает агенту учиться на примерах поведения.
        - **steps**: Количество шагов обучения для Behavioral Cloning. Большое количество шагов (750000) помогает агенту детально изучить демо данные.
        - **strength**: Сила сигнала Behavioral Cloning. 0.4 делает награды от клонирования значимыми.
- **Другие параметры**
    - **keep_checkpoints**: Сохранение 15 контрольных точек помогает гарантировать возможность восстановления на различных этапах процесса обучения.
    - **checkpoint_interval**: Интервал между сохранениями контрольных точек помогает управлять объемом памяти и частотой восстановления.
    - **time_horizon**: Максимальная длина каждого эпизода взаимодействия агента помогает контролировать длину сохраняемых последовательностей действий.
    - **max_steps**: Максимальное количество шагов обучения помогает контролировать время и ресурсы, затрачиваемые на обучение.
    - **summary_freq**: Частота записи данных обучения помогает детально отслеживать прогресс.
    - **threaded**: Использование многопоточности помогает увеличить производительность обучения.

CarBehaviourSAC (SAC)

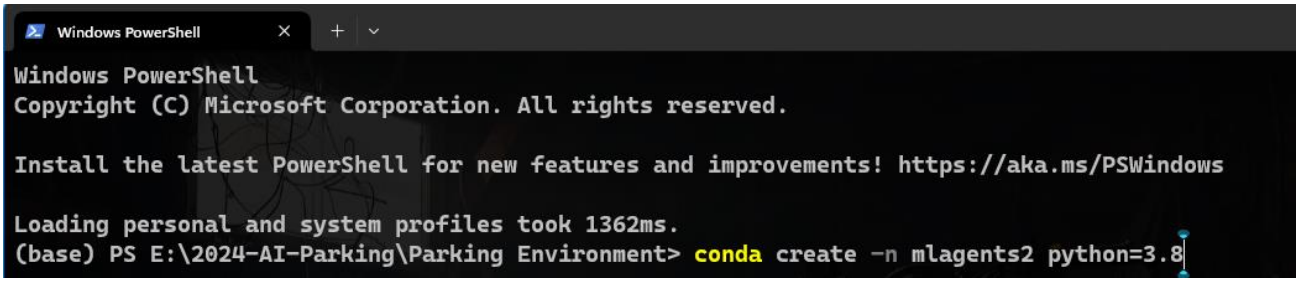
- **trainer_type: sac**
    - Выбор алгоритма обучения. SAC выбран за его стабильность и способность оптимизировать сложные поведения.
- **hyperparameters**
    - **batch_size**: Меньший размер batch (128) для более быстрой реакции на изменения в среде.
    - **buffer_size**: Больший размер буфера (50000) для хранения большего количества опытов и обучения в различных ситуациях.

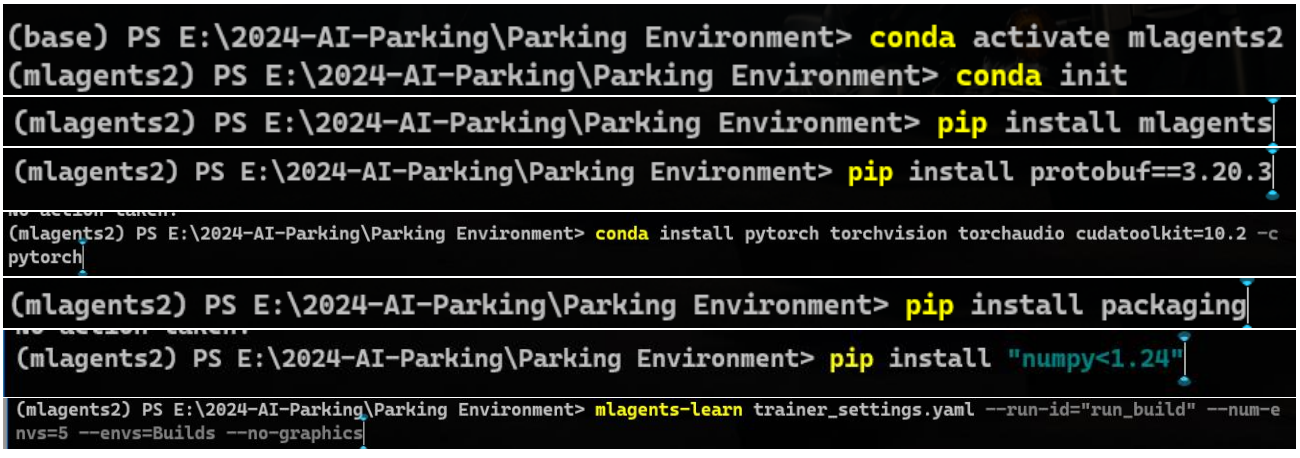
- **learning_rate**: Скорость обучения помогает регулировать степень изменения весов при каждом обновлении.
- **buffer_init_steps**: Количество шагов инициализации для буфера, чтобы обеспечить достаточное количество данных перед началом обучения.
- **init_entcoef**: Начальный коэффициент энтропии помогает контролировать разнообразие в действиях агента.
- **steps_per_update**: Количество шагов между обновлениями помогает контролировать частоту обучения.
- **learning_rate_schedule**: График скорости обучения constant удерживает скорость обучения неизменной на протяжении всего процесса обучения.

- **network_settings**
  - **normalize**: Нормализация входных данных помогает стабилизировать процесс обучения.
  - **hidden_units**: Количество нейронов в каждом скрытом слое определяет сложность нейронной сети.
  - **num_layers**: Количество скрытых слоев определяет глубину нейронной сети.
- **reward_signals**
  - **extrinsic**:
    - **gamma**: Коэффициент дисконтирования помогает фокусироваться на краткосрочных или долгосрочных наградах.
    - **strength**: Сила внешнего сигнала награды.
  - **gail**:
    - **strength**: Сила сигнала награды GAIL помогает учиться на предоставленных демо.
    - **demo_path**: Путь к демо данным, предоставляющим примеры желаемого поведения.
    - **use_actions**: Определяет, использовать ли действия из демо в процессе обучения.
  - **behavioral_cloning**
    - **demo_path**: Путь к демо данным помогает агенту учиться на примерах поведения.
    - **steps**: Количество шагов обучения для Behavioral Cloning.
    - **strength**: Сила сигнала Behavioral Cloning.

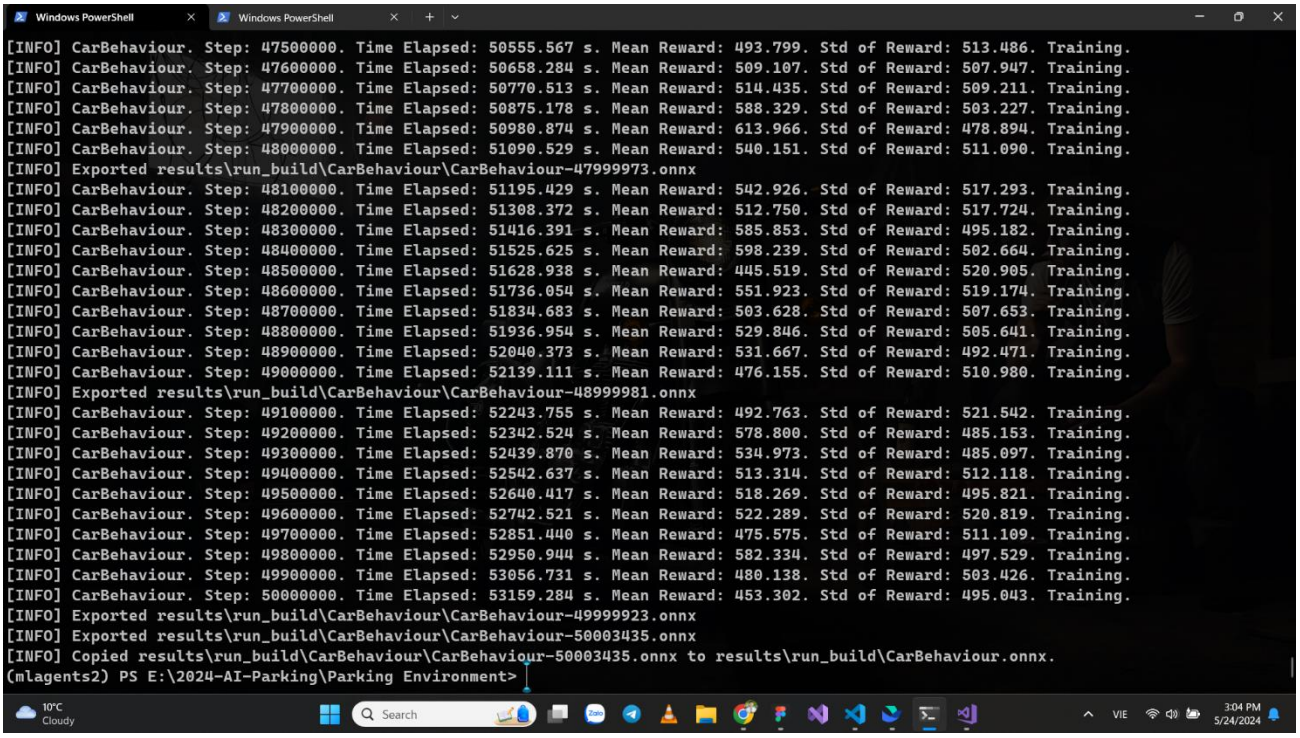
- **Другие параметры**
    - **keep_checkpoints**: Сохранение 15 контрольных точек помогает гарантировать возможность восстановления на различных этапах процесса обучения.
    - **time_horizon**: Максимальная длина каждого эпизода взаимодействия агента помогает контролировать длину сохраняемых последовательностей действий.
    - **max_steps**: Максимальное количество шагов обучения помогает контролировать время и ресурсы, затрачиваемые на обучение.
    - **summary_freq**: Частота записи данных обучения помогает детально отслеживать прогресс.
    - **threaded**: Решает, использовать ли многопоточность для увеличения производительности обучения.

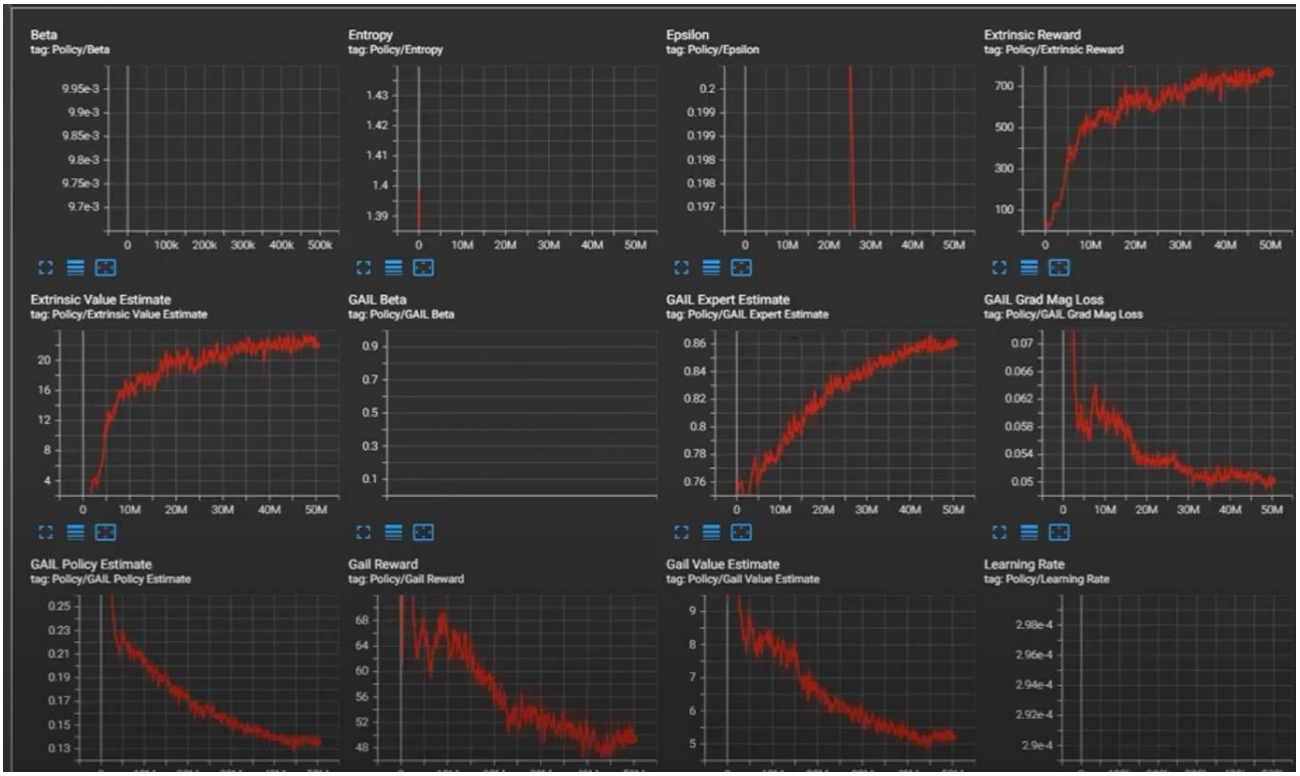
2. **Запуск обучения**:
    - Используем команду **mlagents-learn** в терминале для запуска процесса обучения.
    - Будем наблюдать за процессом обучения и при необходимости корректировать параметры.






D:\miniconda\envs\mlagents2\lib\site-packages\torch__init__.py:747: UserWarning: torch.set_default_tensor_type() is deprecated as of PyTorch 2.1, please use torch.set_default_dtype() and torch.set_default_device() as alternatives. (Triggered internally at C:\cb\pytorch_1000000000000\work\torch\csrc\tensor\python_tensor.cpp:433.)
_C._set_default_tensor_type(t)
Version information:
ml-agents: 0.30.0,
ml-agents-envs: 0.30.0,
Communicator API: 1.5.0,
PyTorch: 2.3.0


Windows PowerShell
Windows PowerShell
[INFO] CarBehaviour. Step: 47500000. Time Elapsed: 50555.567 s. Mean Reward: 493.799. Std of Reward: 513.486. Training.
[INFO] CarBehaviour. Step: 47600000. Time Elapsed: 50658.284 s. Mean Reward: 509.107. Std of Reward: 507.947. Training.
[INFO] CarBehaviour. Step: 47700000. Time Elapsed: 50770.513 s. Mean Reward: 514.435. Std of Reward: 509.211. Training.
[INFO] CarBehaviour. Step: 47800000. Time Elapsed: 50875.178 s. Mean Reward: 588.329. Std of Reward: 503.227. Training.
[INFO] CarBehaviour. Step: 47900000. Time Elapsed: 50980.874 s. Mean Reward: 613.966. Std of Reward: 478.894. Training.
[INFO] CarBehaviour. Step: 48000000. Time Elapsed: 51090.529 s. Mean Reward: 540.151. Std of Reward: 511.090. Training.
[INFO] Exported results\run_build\CarBehaviour\CarBehaviour-47999973.onnx
[INFO] CarBehaviour. Step: 48100000. Time Elapsed: 51195.429 s. Mean Reward: 542.926. Std of Reward: 517.293. Training.
[INFO] CarBehaviour. Step: 48200000. Time Elapsed: 51308.372 s. Mean Reward: 512.750. Std of Reward: 517.724. Training.
[INFO] CarBehaviour. Step: 48300000. Time Elapsed: 51416.391 s. Mean Reward: 585.853. Std of Reward: 495.182. Training.
[INFO] CarBehaviour. Step: 48400000. Time Elapsed: 51525.625 s. Mean Reward: 598.239. Std of Reward: 502.664. Training.
[INFO] CarBehaviour. Step: 48500000. Time Elapsed: 51628.938 s. Mean Reward: 445.519. Std of Reward: 520.905. Training.
[INFO] CarBehaviour. Step: 48600000. Time Elapsed: 51736.054 s. Mean Reward: 551.923. Std of Reward: 519.174. Training.
[INFO] CarBehaviour. Step: 48700000. Time Elapsed: 51834.683 s. Mean Reward: 503.628. Std of Reward: 507.653. Training.
[INFO] CarBehaviour. Step: 48800000. Time Elapsed: 51936.954 s. Mean Reward: 529.846. Std of Reward: 505.641. Training.
[INFO] CarBehaviour. Step: 48900000. Time Elapsed: 52040.373 s. Mean Reward: 531.667. Std of Reward: 492.471. Training.
[INFO] CarBehaviour. Step: 49000000. Time Elapsed: 52139.111 s. Mean Reward: 476.155. Std of Reward: 510.980. Training.
[INFO] Exported results\run_build\CarBehaviour\CarBehaviour-48999981.onnx
[INFO] CarBehaviour. Step: 49100000. Time Elapsed: 52243.755 s. Mean Reward: 492.763. Std of Reward: 521.542. Training.
[INFO] CarBehaviour. Step: 49200000. Time Elapsed: 52342.524 s. Mean Reward: 578.800. Std of Reward: 485.153. Training.
[INFO] CarBehaviour. Step: 49300000. Time Elapsed: 52439.870 s. Mean Reward: 534.973. Std of Reward: 485.097. Training.
[INFO] CarBehaviour. Step: 49400000. Time Elapsed: 52542.637 s. Mean Reward: 513.314. Std of Reward: 512.118. Training.
[INFO] CarBehaviour. Step: 49500000. Time Elapsed: 52640.417 s. Mean Reward: 518.269. Std of Reward: 495.821. Training.
[INFO] CarBehaviour. Step: 49600000. Time Elapsed: 52742.521 s. Mean Reward: 522.289. Std of Reward: 520.819. Training.
[INFO] CarBehaviour. Step: 49700000. Time Elapsed: 52851.440 s. Mean Reward: 475.575. Std of Reward: 511.109. Training.
[INFO] CarBehaviour. Step: 49800000. Time Elapsed: 52950.944 s. Mean Reward: 582.334. Std of Reward: 497.529. Training.
[INFO] CarBehaviour. Step: 49900000. Time Elapsed: 53056.731 s. Mean Reward: 480.138. Std of Reward: 503.426. Training.
[INFO] CarBehaviour. Step: 50000000. Time Elapsed: 53159.284 s. Mean Reward: 453.302. Std of Reward: 495.043. Training.
[INFO] Exported results\run_build\CarBehaviour\CarBehaviour-49999923.onnx
[INFO] Exported results\run_build\CarBehaviour\CarBehaviour-50003435.onnx
[INFO] Copied results\run_build\CarBehaviour\CarBehaviour-50003435.onnx to results\run_build\CarBehaviour.onnx.
(mlagents2) PS E:\2024-AI-Parking\Parking Environment>
10°C
Cloudy
Search
VIE
3:04 PM
5/24/2024



Шаг 6: Тестирование и доработка

1. **Тестирование модели**:
    - После завершения обучения, протестируем модель, запустив симуляцию в Unity.
    - Понаблюдаем за поведением агента и оценим его эффективность.
2. **Доработка модели**:
    - На основе результатов тестирования, скорректируем параметры обучения, вознаграждения и компоненты агента для улучшения эффективности.

Шаг 7: Развертывание и публикация

1. **Развертывание проекта**:
    - Упакуем проект в отдельное приложение или поделитесь им на таких платформах, как Unity Asset Store или GitHub.

Демо: https://youtu.be/n5qa4nFWkqE

Рекомендации

- AI Learns to Park - Deep Reinforcement Learning (https://www.youtube.com/watch?v=VMp6pq6_QjI)

- Multi-Agent Car Parking using Reinforcement Learning Demo (https://www.youtube.com/watch?v=xMCpVxDpogA)
- Parallel Parking using Reinforcement Learning - Unity3D & ML-Agents (https://www.youtube.com/watch?v=QDVwSAgY6cA)
- Multi-Agent Car Parking using Reinforcement Learning (https://arxiv.org/pdf/2206.13338)

Вывод

В процессе работы над проектом я успешно создал разнообразную и сложную среду моделирования, которая включает в себя зоны парковки, препятствия и автомобили, что позволило мне достоверно воссоздать пространство реальной парковки.

Я углубился во множество аспектов разработки автономных систем искусственного интеллекта. Я изучил принципы работы с ML-Agents Toolkit и его интеграцию с Unity, а также освоил методы создания и обучения агентов для выполнения конкретных задач, таких как автоматическая парковка.

Мой опыт включает в себя работу с различными типами данных, определение целей и наград, разработку и оптимизацию алгоритмов обучения с поддержкой глубокого обучения. Я также приобрел практические навыки в настройке параметров обучения и анализе результатов обучения для достижения оптимальной производительности системы.

Кроме того, я углубился в аспекты тестирования и оценки моделей, проводя эксперименты в различных сценариях и корректируя настройки для повышения эффективности и надежности моих решений. Этот проект стал для меня не только возможностью применить теоретические знания, но и научиться решать реальные проблемы в области искусственного интеллекта и автоматизации.