



**МИНОБРНАУКИ РОССИИ**  
федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«Национальный исследовательский университет «МЭИ»

**Институт** ИВТИ  
**Кафедра** ПМИИ

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**  
**(бакалаврская работа)**

**Направление** 01.03.02 Прикладная математика и информатика  
(код и наименование)

**Направленность (профиль)** Математическое и программное

**Обеспечение вычислительных машин и компьютерных сетей**

**Форма обучения** очная  
(очная/очно-заочная/заочная)

**Тема:** Разработка программного приложения для моделирования  
боевого робота.

**Студент** А-05-16 Кутепов А.О.  
группа подпись фамилия и инициалы

**Научный**  
**руководитель** К.т.н доцент Бартеньев О.В.  
уч. степень должность подпись фамилия и инициалы

**Консультант**  
уч. степень должность подпись фамилия и инициалы

**Консультант**  
уч. степень должность подпись фамилия и инициалы

**«Работа допущена к защите»**

**Зав. кафедрой** Д.т.н. профессор Еремеев А.П.  
уч. степень звание подпись фамилия и инициалы

**Дата** \_\_\_\_\_

**Москва, 2020**



**Институт  
Кафедра**

**АВТИ  
ПМ**

**ЗАДАНИЕ  
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ  
(бакалаврскую работу)**

**Направление** 01.03.02 Прикладная математика и информатика  
(код и наименование)

**Направленность (профиль)** Математическое и программное  
обеспечение вычислительных машин и сетей

**Форма обучения** очная  
(очная/очно-заочная/заочная)

**Тема:** Разработка программного приложения для моделирования  
боевого робота.

<b>Студент</b>	A-05-16			Кутепов А.О.
	группа		подпись	фамилия и инициалы

<b>Научный руководитель</b>	К.Т.Н	доцент		Бартеньев О.В.
	уч. степень	должность	подпись	фамилия и инициалы

<b>Консультант</b>				
	уч. степень	должность	подпись	фамилия и инициалы

<b>Консультант</b>				
	уч. степень	должность	подпись	фамилия и инициалы

<b>Зав. кафедрой</b>	Д.Т.Н	профессор		Еремеев А.П.
	уч. степень	звание	подпись	фамилия и инициалы

**Место выполнения работы** «Национальный исследовательский  
университет «МЭИ»

## СОДЕРЖАНИЕ РАЗДЕЛОВ ЗАДАНИЯ И ИСХОДНЫЕ ДАННЫЕ

1. Обзор боевых роботов (БР). Их классификация.
2. Описание выбранной модели БР.
3. Описание выбранной среды для моделирования БР.
4. Описание задач, решаемых при моделировании БР.
5. Создание модели БР в выбранной среде.
6. Анимация БР.
7. Автоматическое определение угроз и противодействие им.

## ПЕРЕЧЕНЬ ГРАФИЧЕСКОГО МАТЕРИАЛА

1. Графическое представление модели БР.
2. Структура созданного программного приложения.
3. Видеофайл, демонстрирующий различные аспекты поведения БР.

**Количество листов** \_\_\_\_\_

**Количество слайдов в презентации** \_\_\_\_\_

## РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Николенко С., Кадури А., Архангельская Е. Глубокое обучение. – СПб.: Питер, 2018. – 480 с.
2. Д. Хокинг. Unity в действии. Мультиплатформенная разработка на C#. 2-е межд. изд. – СПб.: Питер, 2019. – 352 с.
3. M. Gaig. Unity Game Development in 24 hours. – 800 East 96th Street , Indianapolis , Indiana 46240 USA : Sams, 2018. – 618 с.
4. Документация по машинному обучению в Unity. [Электронный ресурс] URL: <https://github.com/Unity-Technologies/mlagents/blob/master/docs/Readme.md>.

## АННОТАЦИЯ

выпускной работы бакалавра на тему «Разработка программного приложения для моделирования боевого робота».

Автор работы: Кутепов Артём Олегович— студент группы А-05-16.

Выпускная работа состоит из введения, трёх глав, заключения, списка использованных источников и приложения. Объём выпускной работы составляет 57 страниц без приложений. Иллюстративный материал включает 58 рисунков. Список использованных источников представлен 16 наименованиями.

В первой главе проводится обзор на боевых роботов, указывается их классификация и говорится об плюсах их автономной работы. Во второй главе рассматриваются теоретические основы обучения с подкреплением и некоторые алгоритмы. В третьей главе приведено создание моделей боевых роботов с помощью различных методов обучения с подкреплением. В заключении приведены основные результаты, полученные в выпускной работе бакалавра.

The graduation work consists of an introduction, three chapters, a conclusion, a list of used sources and supplement. The volume of the final work is 57 pages without supplement. The illustrative material includes 58 figures. The list of used sources consists of 16 items.

The first chapter reviews the combat robots, indicates their classification and talks about the advantages of their offline work. The second chapter discusses the theoretical foundations of reinforcement learning and some algorithms. The third chapter provides the creation of models of combat robots using various training methods with reinforcements. In conclusion, the main results obtained in the graduate work of the bachelor are presented.

## ОГЛАВЛЕНИЕ

АННОТАЦИЯ .....	4
СПИСОК СОКРАЩЕНИЙ .....	6
ВВЕДЕНИЕ.....	7
ГЛАВА 1.Обзор боевых роботов и их классификация .....	8
1.1 Обзор боевых роботов .....	8
1.2 Классификация боевых роботов .....	11
1.3 Управление боевыми роботами .....	12
1.4 Выводы.....	13
ГЛАВА 2.Теоретические основы обучения с подкреплением. ....	14
2.1 Постановка задачи.....	14
2.2 Алгоритмы обучения с подкреплением .....	16
2.3 Исследование среды.....	19
2.4 Обучение на основе демонстраций .....	21
2.5 Выводы.....	22
ГЛАВА 3.Применение методов обучения с подкреплением. ....	23
3.1 Unity3D – симулятор для боевого робота. ....	23
3.2 Боевой робот №1 .....	25
3.2.1 Модель №1. ....	29
3.2.2 Модель №2 .....	39
3.2.3 Модель №3 .....	40
3.2.4 Модель №4 .....	40
3.2.5 Модель №5. Curriculum learning .....	42
3.2.6 Статистика моделей .....	44
3.3 Боевой робот №2 .....	46
3.3.1 Модель №1 .....	50
3.3.2 Модель №2.....	57
3.4 Выводы.....	61
ЗАКЛЮЧЕНИЕ .....	62
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	63
ПРИЛОЖЕНИЕ А. Код боевого робота №1 .....	65
ПРИЛОЖЕНИЕ В. Код боевого робота №2.....	68

## **СПИСОК СОКРАЩЕНИЙ**

ИИ – Искусственный Интеллект – Artificial Intelligence – AI.

БР – Боевой Робот.

МО – Машинное Обучение – Machine Learning – ML.

Обучение с подкреплением – Reinforcement Learning – RL.

НС – Нейронная сеть – Neural Network – NN.

## ВВЕДЕНИЕ

Машинное обучение является ветвью искусственного интеллекта. Сегодня с помощью методов МО люди решают задачи в различных сферах деятельности. Например, с помощью МО можно распознать рукописный текст и изображение, выявить опасное заболевание у человека, спрогнозировать погоду и автомобильные пробки на несколько дней вперёд, выявить приходящий на электронную почту спам и многое другое.

Обучение с подкреплением – один из методов машинного обучения. С помощью RL удалось добиться потрясающих результатов в таких играх как шахматы, нарды, покер, го, Dota 2, StarCraft, Quake III Arena, классические игры для игровой консоли Atari 2600. Во всех этих играх НС достигла сравнимого с человеком уровня игры, а в некоторых даже превзошла его.

Также RL нашло своё применение в роботике, где оно используется при разработке различных роботов, беспилотных автомобилей.

Достижения RL могут быть использованы и в военных целях для повышения обороноспособности государства. Например, для создания боевого робота, который самостоятельно смог бы патрулировать заданную ему территорию, определять угрозы и противодействовать им.

Целью данной выпускной работы является создание программного обеспечения для моделирования боевого робота. Для достижения заданной цели необходимо решить следующие задачи:


1. Рассмотреть характеристики существующих боевых роботов.  
Выбрать модель БР.
2. Выбрать и создать среду работы БР.
3. Определить угрозы и методы борьбы с ними.
4. Применить и сравнить методы обучения БР.

# ГЛАВА 1. Обзор боевых роботов и их классификация




## 1.1 Обзор боевых роботов





В таблице 1 представлены некоторые общие характеристики лишь малой части БР, уже стоящих на вооружении различных стран мира либо проходящих испытания.


Таблица 1. Некоторые общие характеристики БР.

Название робототехнического комплекса	Снаряженная масса, т	Габариты, мм	Максимальная скорость движения, км/ч	Управление	Вооружение и оборудование	Предназначение и возможности
 <p>«Уран-6»</p>	6	4565x2010 x1470	5	Дистанционное, по радиоканалу не менее 800 м.	Бойковый, катковый и фрезерный траллы, бульдозерный отвал и механический схват.	Прodelывание проходов в минно-взрывных заграждениях; разминирование урбанизированных участков местности, а также горных и мелколесистых территорий.
 <p>«Уран-9»</p>	до 12	5600x2500 x3100	Не менее 35	Дистанционное, по радиоканалу не менее 4 км.	30-мм автоматическая пушка 2А72; 7,62-мм пулемет ПКТМ; Комплекс управляемого вооружения «Атака»; Реактивные пехотные огнемёты РПО ПДМ-А "Шмель-М".	Ведение разведки; огневая поддержка общевойсковых и разведывательных подразделений; поражение противотанковыми ракетами неподвижной и движущейся цели типа «танк» на дальностях до 5000 м днём и 3500 м ночью; поражение пушечно-стрелковым и огнемётным вооружением днём и ночью неподвижных и движущихся целей; движение по заданному маршруту с автоматическим определением и объездом препятствий; автоматическое сопровождение целей.
 <p>«Уран-14»</p>	14	3800x2180 x2019	10	Дистанционное, по радиоканалу не менее 1 км.	Водяной пожарный насос; цистерны с водой и пенообразователем; поворотный схват.	Дистанционное пожаротушение опасных и труднодоступных для личного состава объектов; разведка очагов пожара при помощи системы видеонаблюдения в оптическом и тепловизионном диапазонах; проделывание проходов в завалах и разрушениях; подъём и перемещение с помощью схвата опасных и взрывчатых грузов массой до



						1,5 тонн.
 <p>«Платформа-М»</p>	не более 0.8	1600x1200 x1200	12	Дистанционное по радиоканалу до 1500 м.	Тепловизоры; дальномеры; видеокамеры; спецсредства для обнаружения различных веществ; радиолокационная станция; пулемёты; гранатомёты; противотанковый ракетный комплекс; минный тралл; громкоговорители.	Разведка; обнаружение и поражение различных целей; огневая поддержка войсковых подразделений; патрулирование и охрана объектов; постановка дымовых завес; дистанционное минирование и разминирование; доставка грузов на небольшие расстояния; проведение аудиопропаганды.
 <p>«Капитан»</p>	0.035	620x465x 215	5,4	Управление по оптоволоконному или радиоканалу до 1200 м.	Многоступенной манипулятор с захватным устройством; система видеонаблюдения; набор инженерных инструментов.	Ведение оптической и акустической разведки; наблюдение за окружающей обстановкой; досмотр автотранспортных средств, самолетов; обследование помещений, подвалов, убежищ; выполнение манипуляций с лёгкими грузами; поиск и обезвреживание взрывоопасных предметов.
 <p>«Вихрь»</p>	14,7	6700x3300 x2450	по суше – 60 на плаву - 10	Ручное и дистанционное управление по защищённому каналу связи до 10 км.	30-мм пушка 2А72; 7,62-мм пулемет ПКТМ; противотанковый ракетный комплекс "Корнет-М".	Огневая поддержка в ходе ведения боевых действий; ведение разведки; поражение живой силы и легкобронированных целей противника; выполнение специальных задач; при движении к заданной точке может автоматически объезжать препятствия.

 <p>«Guardium»</p>	1.4	2950x2020 x1800	80	Ручное и дистанционное управление, автопилот.	40-мм автоматический гранатомет; крупнокалиберный 12,7-мм пулемёт; устройство для метания гранат со слезоточивым газом; шестиствольный пулемёт.	Патрулирование; сопровождение автоколонн; разведка; охрана; эвакуация раненных.
 <p>БПЛА «MQ-9 Reaper» (Predator B)</p>	4.760	Длина: 11 Размах крыла: 20	400	Управление по спутниковым каналам связи, автопилот.	Ракеты «воздух — земля» AGM-114 «Хеллфайр» ;  бомбы Mark 82 с лазерным и GPS наведением ;  телекамеры, работающие в видимом и инфракрасном диапазонах; радар.	Разведка; наблюдение; целеуказание; нанесение точечных ударов.
 <p>БПЛА «MQ-27A»</p>	0.022	Длина: 1.71 Размах крыла: 3.11	148	Управление по спутниковым каналам связи, автопилот.	ИК- камера , система связи.	Разведка.
 <p>Роботизированный комплекс малая подводная лодка «Суррогат»</p>	Водоизмещение 40 тонн.	17	44км/ч	Управление по спутниковым каналам связи, автопилот.	Буксируемые антенны различного назначения; аппаратура для картографирования и разведки.	Имитация подводных лодок различных классов и типов, дезинформация противника, разведка, картографирование местности.

 <p>многоцелевой самоходный подводный аппарат «Прометей»</p> <p>(«Суррогат-6», Капулон)</p>	100 тонн	Длина:  20 м  Радиус:  1.8 м	200	Автопилот, управление с подводных лодок.	Сверхмощная боеголовка до 100 Мт.	Создание искусственного цунами и массивного ядерного загрязнения побережья с целью невозможности ведения там хозяйственной деятельности и проживания.
--	----------	--	-----	--	-----------------------------------	---

## 1.2 Классификация боевых роботов

БР можно классифицировать по среде работы, способу управления, массе, выполняемой задаче, внешнему виду, двигательной установке, шасси и способу перемещения. Некоторые роботы могут функционировать в различных средах, выполнять несколько задач, управляться при помощи операторов и действовать самостоятельно.

Таб. 2. Классификация БР.

<b>I. По среде работы</b>
1) Суша
2) Воздух
3) Море
<b>II. По способу управления</b>
1) Операторы
2) Полностью автономные
<b>III. По массе</b>
1) Лёгкие (до 3 т.)
2) Средние (от 3 т. до 13 т.)
3) Тяжёлые (свыше 13 т.)
<b>IV. Выполняемые задачи</b>
1) Патрулирование
2) Разминирование
3) Охрана
4) Разведка

5) Дезинформация противника
6) Доставка грузов
<b>V. Внешний вид</b>
1) Аналоги боевым машинам
2) Паукоподобные
3) Кошкоподобные
4) Собакоподобные
5) Змееподобные
6) Андроиды
<b>VI. По двигательным установкам</b>
1) Двигатели внутреннего сгорания
2) Турбореактивные двигатели
3) Ядерные энергоустановки
4) Электродвигатели
<b>VII. По шасси и способам передвижения</b>
1) Колёсные
2) Гусеничные
3) Шаровидные
4) Плавающие
5) Самолётного типа
6) Вертолётного типа
7) Подводные
8) Шагоходы

### 1.3 Управление боевыми роботами

Дистанционное управление БР имеет следующие преимущества перед обычной боевой машиной, управляемой находящимися в ней людьми:

1. Безопасность личного состава. Операторы могут находиться в нескольких километрах от места ведения боевых действий.
2. Увеличение количества боевой техники. Один оператор – одна машина.
3. Снижение стоимости машины благодаря исчезновению места для размещения экипажа, брони для его защиты и комфортной подвески.

4. Повышение технико-эксплуатационных характеристик машины благодаря снижению массы, размеров. Возможности зависят от аппаратуры, а не от экипажа.

Автономная работа БР может дать следующие преимущества по сравнению с дистанционным управлением:

1. Безопасность операторов. Зачастую робот управляется находящимися поблизости от него сидящими в укрытии операторами, демаскировка которых способна привести к срыву операции, потерям среди личного состава, техники, в том числе и самого БР.
2. Исключит человеческий фактор. Во-первых, человеку свойственно ошибаться. Во-вторых, у него могут возникнуть по причинам личного характера нервные срывы, что негативно отразится на выполнении боевого задания.
3. Не потребуются мощные средства связи. БР сможет самостоятельно действовать за сотни и даже тысячи километров от начальной точки по всему Земному шару. Отсутствие принимаемых и передаваемых сигналов затруднит обнаружение БР, а вражеские средства глушения сигнала не смогут повлиять на его работу.
4. Повышение скорости и эффективности работы БР. Ведь оператор может эффективно одновременно управлять одним-двумя БР, а некоторые машины для управления требуют слаженной работы нескольких операторов.
5. Самостоятельное объединение множества БР в подразделения и их слаженная работа.

#### **1.4 Выводы**

Вооружённые силы различных стран мира заинтересованы в автономных боевых роботах. Создание ИИ для БР является актуальной задачей.

## ГЛАВА 2. Теоретические основы обучения с подкреплением.

### 2.1 Постановка задачи

На каждом шаге агент находится в состоянии  $s \in S$ , где  $S$  – множество всех состояний, выбирает некоторое действие  $a \in A$  из имеющегося набора действий  $A$ . Среда сообщает в каком новом состоянии  $s' \in S$  он оказался и какую награду  $R$  он получил после выполнения выбранного действия. Этот процесс изображён на рисунке 1.

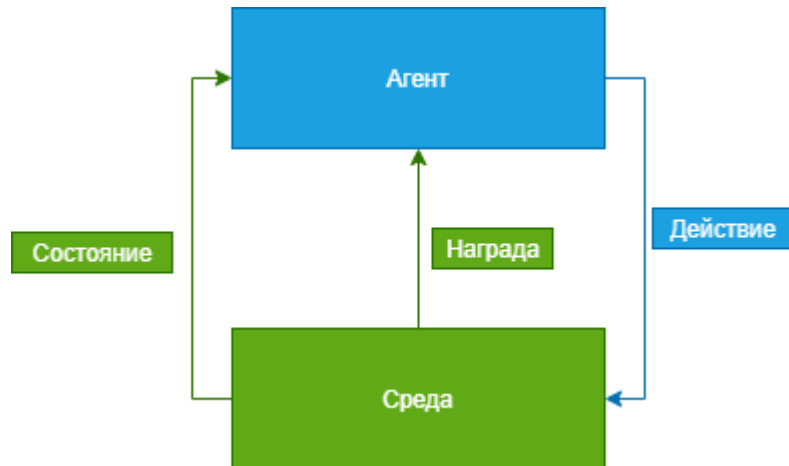


Рис. 1. Цикл обучения с подкреплением.

Цель агента — получить максимальную награду. Награда  $R$  может определяться:

- 1) либо за конечное время  $h$  («horizont» – горизонт)

$$R = E[r_0 + r_1 + r_2 + \dots + r_h] = E[\sum_{t=0}^h r_t] \quad (2.1),$$

где  $r_t$  - награда агента на шаге  $t$ ;

- 2) либо за бесконечное предстоящее время

$$R = E[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots + \gamma^k r_k + \dots] = E[\sum_{t=0}^{\infty} \gamma^t r_t] \quad (2.2),$$

где  $\gamma \in (0,1)$  – дисконтирующий множитель (discount factor). Причём в данном случае  $R$  оказывается ограниченной

$$R = \sum_{t=0}^{\infty} \gamma^t r(st) \leq \sum_{t=0}^{\infty} \gamma^t r_{max} = \frac{r_{max}}{1-\gamma} \quad (2.3) .$$

Но для достижения своей цели агент должен знать в каком состоянии какое выбрать действие. Агент выполняет действия и, если они приводят к требуемому результату, получает от среды награду и тем самым *подкрепляет*

выполнение своих правильных действий. Поэтому у агента должна быть стратегия  $\pi$  (policy, поведение) – функция, которая по каждому состоянию среды  $s$  выдает действие, которое необходимо совершить в этом состоянии  $\pi(s) : S \rightarrow A$ . Стратегия – это функция, которая для данного состояния  $s$  выдает распределение вероятностей на множестве действий  $A$ . Оптимальной стратегией является та стратегия  $\pi^*$ , которая позволяет агенту при следовании этой стратегии достичь максимальной ожидаемой полезности. Задача поиска оптимальной стратегии:

$$\pi^* = \arg \max_{\pi} E[\sum_{t=0}^{\infty} \gamma^t r(s_t) | \pi] \quad (2.4)$$

*Марковский процесс принятия решений* состоит из:

- множества состояний  $S$ ;
- множества действий  $A$ ;
- функции вознаграждения  $R: S \times A \rightarrow R$ ; ожидаемое вознаграждение при переходе из  $s$  в  $s'$  после действия  $a$  составляет  $R_{ss'}^a$ ;
- функции перехода между состояниями  $P_{ss'}^a: S \times A \rightarrow \Pi(S)$ , где  $\Pi(S)$  – множество распределений вероятностей над  $S$ ; это значит, что вероятность попасть из состояния  $s$  в состояние  $s'$  после действия  $a$  равна  $P_{ss'}^a$ .

Марковский переход – модель переходов зависит от текущего состояния, а не от истории предыдущих переходов.

*Функция значения состояния* (value function,  $V(s)$ ) – общее ожидаемое подкрепление, которое можно получить, если начать с этого состояния:

$$V^{\pi}(s) = E[R_t | s_t = s] = E_{\pi} [\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s] \quad (2.5)$$

*Функция полезности действия*  $Q$  выражает общее подкрепление, ожидаемое, если агент начнет в состоянии  $s$  и сделает там действие  $a$ :

$$Q^{\pi}(s, a) = E[R_t | s_t = s, a_t = a] = E_{\pi} [\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a] \quad (2.6)$$

Для известной стратегии  $\pi$  значения  $V$  и  $Q$  удовлетворяют уравнениям Беллмана:

$$\begin{aligned}
V^\pi(s) &= E[R_t | s_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s] = \\
&= E[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s] = \\
&= \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s']) = \\
&= \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma V^\pi(s')) \tag{2.7}
\end{aligned}$$

$$Q(s, a) = \sum_{s' \in S} P_{ss'}^a (R_{ss'}^a + \gamma \sum_{a'} \pi(s, a') Q(s, a')) \tag{2.8}$$

А для оптимальных значений  $V^*$  и  $Q^*$ :

$$V^*(s) = \max_a \sum_{s' \in S} P_{ss'}^a (R_{ss'}^a + \gamma V^*(s')) \tag{2.9}$$

$$Q^*(s, a) = \sum_{s' \in S} P_{ss'}^a (R_{ss'}^a + \gamma \max_{a'} Q^*(s, a')) \tag{2.10}$$

Причём:

$$V^*(s) = \max_{a'} Q^*(s, a') \tag{2.11}$$

$$\pi^* = \arg \max_a \sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma V^*(s')) \tag{2.12}$$

## 2.2 Алгоритмы обучения с подкреплением

Простейший алгоритм TD(0)-обучения (Temporal Difference) выглядит так. Сначала нужно инициализировать функцию  $V(s)$  и стратегию  $\pi$  произвольно, а затем на каждом эпизоде обучения:

- инициализировать  $s$ ;
- для каждого шага в эпизоде:
  - выбрать  $a$  по стратегии  $\pi$ ;
  - сделать  $a$ , пронаблюдать результат  $r$  и следующее состояние  $s$ ;
  - обновить функцию  $V$  на состоянии  $s$  по формуле:

$$V(s) := V(s) + \alpha(r + \gamma V(s') - V(s)) \tag{2.13},$$

где  $\alpha$  – параметр, характеризующий скорость обучения

- перейти к следующему шагу, присвоив  $s := s'$ .

Если реализовать этот алгоритм для функции  $Q$

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.14},$$

то получится алгоритм SARSA.



А для функции

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (2.15)$$

алгоритм Q-обучения.

Табличные методы, в которых для каждого состояния или пары состояние–действие определяется полезность и сохраняется в таблицу или в список не подходят в тех случаях, когда множество состояний и действий очень велико или даже бесконечно.

Решением этой проблемы является использование приближённого представления функции полезности, когда агент по небольшому количеству наблюдаемых состояний может провести обобщение и вычислить функцию полезности для оставшегося множества состояний и действий.

Нейронные сети отлично работают в качестве универсального чёрного ящика, который может приблизить любую функцию. При определении параметров аппроксиматора необходимо задать функцию потерь и способ обновления параметров по текущему значению этой функции. В качестве правила обновления параметров  $w$  в нейронной сети применяется *градиентный спуск* (gradient descent). Его идея заключается в том, что агент меняет параметры в направлении, обратном (при решении задачи минимизации) тому, которое задается градиентом функции потерь:

$$w_{t+1} = w_t - \alpha \nabla_w L(w_t) \quad (2.16),$$

где  $L(w)$  – оптимизируемая функция потерь,  $\alpha$  – параметр длины шага, задающий скорость обучения. Агент использует текущие оценки полезности:  $\hat{V}(s, w)$ , или  $\hat{Q}(s, a, w)$ , для выбора действия, после каждого шага или в конце эпизода агент обновляет веса аппроксиматора с помощью градиентного шага, получает новые оценки полезности и обновляет свою стратегию и т. д.

В задачах, в которых непрерывны не только состояния, но и действия применяется *градиентный спуск по стратегиям* (policy gradient).

Рассмотрим стратегию  $\pi$ , которая делает действие  $a$  в зависимости от текущего состояния  $s$  и собственных параметров  $w$ ,  $a \sim \pi(a | s, w)$ . Нужно максимизировать функцию  $J = E[\sum_{t=0}^{\infty} \gamma^t r_t]$ . Поскольку вознаграждения зависят от стратегии, а та задана параметрически, получается, что целевая функция — это функция от  $w$ . Допустим, что происходит марковский процесс принятия решений из одного шага, агент начинает в состоянии  $s \in S$  с вероятностью  $p_0(s)$ , и после одного действия  $a \in A$  получает награду  $R_s^a$ . Тогда:

$$J(w) = \sum_{s \in S} p_0(s) \sum_{a \in A} \pi(a|s; w) R_s^a \quad (2.17),$$

$$\nabla_w J(w) = \sum_{s \in S} p_0(s) \sum_{a \in A} R_s^a \nabla_w \pi(a|s; w) \quad (2.18),$$

Заметим что:

$$\nabla_w \pi(a|s; w) = \pi(a|s; w) \frac{\nabla_w \pi(a|s; w)}{\pi(a|s; w)} = \pi(a|s; w) \nabla_w \log \pi(a|s; w) \quad (2.19),$$

Значит:

$$\begin{aligned} \nabla_w J(w) &= \sum_{s \in S} p_0(s) \sum_{a \in A} R_s^a \pi(a|s; w) \nabla_w \log \pi(a|s; w) = \\ &= E_{\pi(w)} [R_s^a \nabla_w \log \pi(a|s; w)] \end{aligned} \quad (2.20),$$

и

$$\nabla_w J(w) = E_{\pi(w)} [\nabla_w \log \pi(a|s; w) Q^{\pi(w)}(s, a)] \quad (2.21),$$

Градиент по стратегиям используется в алгоритме REINFORCE:

- 1: function REINFORCE
- 2:     инициализируем  $w$ ;
- 3:     for каждого эпизода  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \approx \pi(w)$  do
- 4:         for  $t = 1$  и до  $T - 1$  do
- 5:              $w \leftarrow w + \alpha \nabla_w \log \pi(a, s, w) R_t$
- 6:     return  $w$

Но получить хорошие результаты с помощью методов градиентного спуска по стратегиям сложно, потому что они чувствительны к выбору размера шага: слишком маленький - прогресс движется очень медленно; слишком большой - сигнал перегружен шумом, или можно увидеть катастрофические падения производительности. Они также часто имеют очень низкую эффективность выборки, принимая миллионы шагов для изучения простых задач.

К счастью, алгоритм Proximal Policy Optimization(PPO) обеспечивает баланс между простотой реализации, сложностью выборки и простотой настройки, вычисляет обновление на каждом шаге, которое минимизирует функцию потерь, при этом обеспечивая относительно небольшое отклонение от предыдущей политики:

$$L^{CLIP}(w) = E_t[\min(r_t(w)A_t, \text{clip}(r_t(w), 1 - \epsilon, 1 + \epsilon)A_t)] \quad (2.22),$$

где

$r_t$  - отношение вероятностей при новой и старой политике соответственно,

$A_t$  - ожидаемое преимущество в момент времени  $t$ ,

$\epsilon$  - гиперпараметр (обычно 0.1 или 0.2)

1:Algorithm PPO

2:   for iteration=1, 2, . . . do

3:       for actor=1, 2, . . . , N do

4:           Run policy  $\pi_{w_{old}}$  in environment for T timesteps

5:           Compute advantage estimates  $A_1 \dots A_T$

8:       end for

9:       Optimize surrogate L wrt  $\theta$ , with K epochs and minibatch size  $M \leq NT$

10:        $w_{old} \leftarrow w$

11:   end for

Основными преимуществами методов, основанных на стратегии, являются лучшая сходимость процесса обучения, большая эффективность в задачах высокой размерности, возможность работы с непрерывным множеством действий и возможность обучаться стохастическим стратегиям, которые в некоторых задачах могут приводить к оптимальному решению в отличие от детерминированных стратегий.

## 2.3 Исследование среды

Классической проблемой обучения с подкреплением является проблема применения-исследования (exploitation vs. exploration problem) соотношения между процессом исследования среды, когда агент набирает необходимые для

принятия решений знания, и процессом применения найденной стратегии по достижению цели.

Чтобы получить какое-то представление об окружающей его среде боевой робот может собрать необходимую информацию о ней при помощи закрепленных на нём сенсоров и/или получить сведения от других объектов, в роли которых могут выступать спутники, радиопередатчики или другие роботы.

Для исследования среды применяется  $\epsilon$ -жадная стратегия ( $\epsilon$ -greedy), которая заключается в том, чтобы выбрать действие с наилучшей ожидаемой прибылью с вероятностью  $1 - \epsilon$ , а с вероятностью  $\epsilon$  выбрать случайное действие. Обычно начинают с больших  $\epsilon$ , а затем уменьшают. Выбор стратегии этого уменьшения — важный параметр алгоритма.

$$\pi_{\epsilon}(s) = \begin{cases} random, & \text{с вероятностью } p = \epsilon \\ argmax_a Q(s, a), & \text{с вероятностью } p = 1 - \epsilon \end{cases} \quad (2.23)$$

Агент может получать не только внешние(extrinsic) награды от среды, но ещё и так называемые внутренние(intrinsic) награды, благодаря которым агент может исследовать среду с редкими внешними наградами, когда агент получает вознаграждение только при успешном выполнении задачи или в конце эпизода обучения. Внешние и внутренние награды могут использоваться вместе.

Принцип генерации внутренних наград за любопытство(curiosity reward) следующий:

Обучаются две модели НС:

- Обратная модель, которая учится по текущему и следующему наблюдению агента предсказывать предпринятое между ними действие
- Прямая модель, которая учится по текущему наблюдению и действию предсказывать следующее наблюдение.

Разница между прогнозируемыми и фактическими наблюдениями и действиями используется в качестве внутренней награды агента. Чем больше модели будут удивлены, тем больше будет награда.

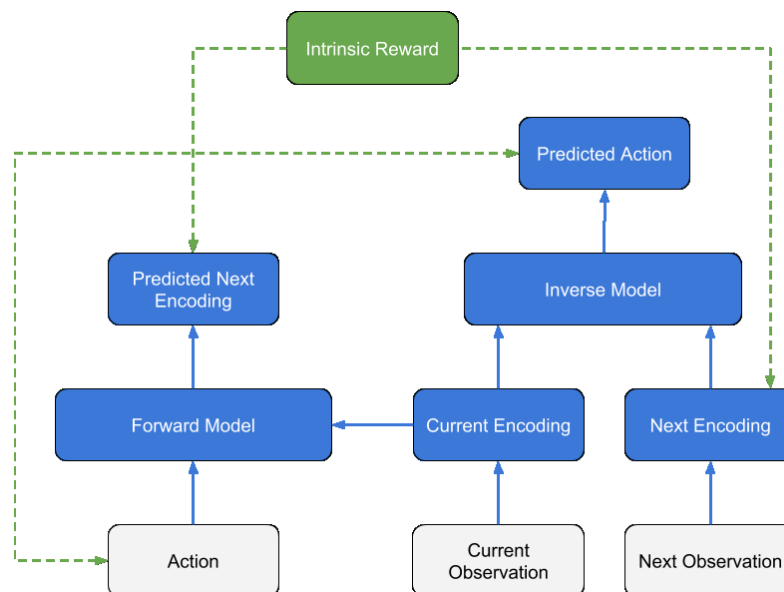


Рис. 2. Модуль генерации внутренних наград агента.

## 2.4 Обучение на основе демонстраций

Агент может получать внутренние награды за копирование действий эксперта. При имитационном обучении агент старается действовать так, как действовал в таких же ситуациях эксперт.

При обучении на основе демонстраций применяется генеративно-сопоставительная сеть (Generative Adversarial Imitation Learning, GAIL). Дискриминатор (вторая НС) учится определять является ли действие/наблюдение демонстрацией или произведено агентом. Чем ближе производимое агентом действие/наблюдение к демонстрируемому, тем выше награда. С каждым шагом агент стремится копировать движения эксперта, а дискриминатор всё строже различать действия/наблюдения эксперта и агента, так что агенту приходится стараться с каждым разом лучше, чтобы обмануть дискриминатор.

При имитационном обучении НС оптимизирует поведение агента путём достижения минимальных различий между действиями эксперта и агента в той же ситуации.

## **2.5 Выводы**

Агент стремиться выполнять действия, ведущие к максимальной награде. Со временем у агента формируется оптимальная стратегия обучения.

Существуют различные алгоритмы обучения с подкреплением, но для задачи создания ИИ для боевого робота подойдёт алгоритм основанный на градиентном спуске по стратегиям. Одним из таких алгоритмов является Proximal Policy Optimization.

Агент может получить не только награды от среды, но и внутренние награды благодаря исследовательскому и имитационному методам обучения.

## ГЛАВА 3. Применение методов обучения с подкреплением.

### 3.1 Unity3D – симулятор для боевого робота.

Создавать модель и среду БР будем в Unity3D – межплатформенной среде разработки компьютерных игр. Unity3D также используется для создания проектов в автоиндустрии, кинематографе и архитектуре. А с помощью Unity Machine Learning Agents Toolkit (ML-Agents) – пакете инструментов для машинного обучения, появилась возможность обучать NPC – находящихся не под управлением игрока персонажей. Возможности ML-Agents интересны не только разработчикам игр, но и исследователям в области ИИ, так как с помощью Unity3D можно смоделировать различные реалистичные среды, настроить физику и добавить визуальные эффекты. ML-Agents является проектом с открытым исходным кодом и бурно развивается, предоставляя в каждом обновлении новые возможности для обучения агентов.

Unity3D при помощи Python API взаимодействует с Tensorflow – библиотекой с открытым исходным кодом для МО. В Tensorflow находятся реализованные алгоритмы RL и с помощью неё выполняются необходимые вычисления. Взаимодействие с Python API показано на рисунке 2.

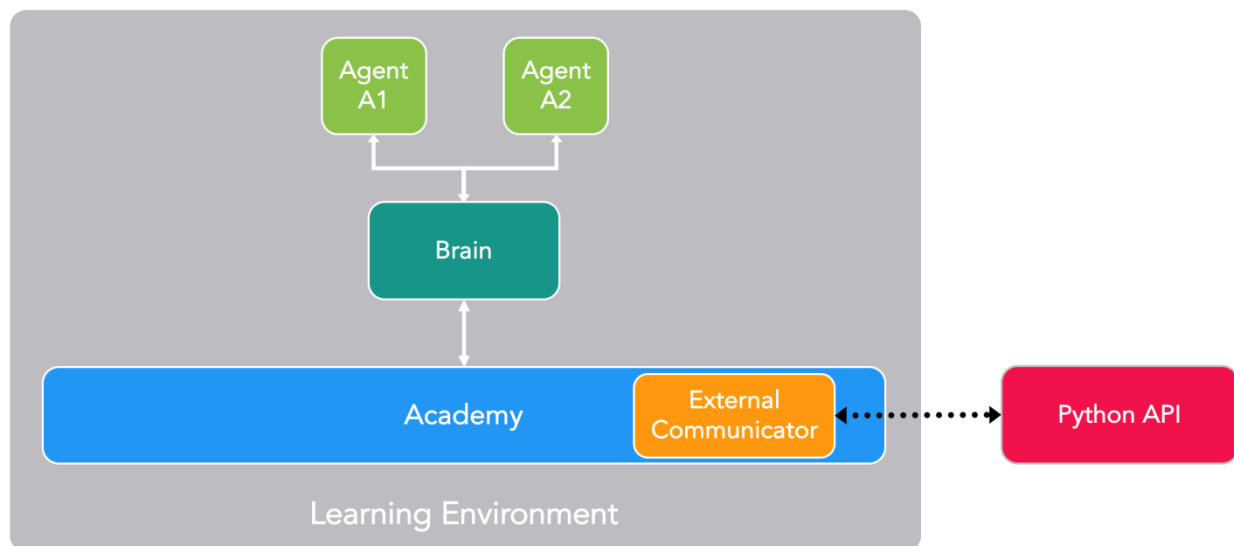


Рис. 2. Взаимодействие с Python API.

В одной среде могут действовать несколько одинаковых агентов, тогда они объединяются в одну группу и для их работы используется одна НС. На

рисунке 2 эти агенты обозначены как AgentA1 и AgentA2. Brain является НС, содержит настройки для обучения агентов, собирает от них информацию и передаёт её Academy. Academy управляет средой, вызывает функции для обучения агента, взаимодействует с Python API через External Communicator. Средой обучения для агента в Unity3D является сцена.

Если же в среде действуют различные агенты, то для каждой группы агентов должен существовать свой Brain(своя НС). На рисунке 3 группа агентов №1 использует BrainA, состоящие из одного агента группы №2 и №3 – BrainB и BrainC соответственно, и состоящая из трёх агентов группа №4 – BrainD.

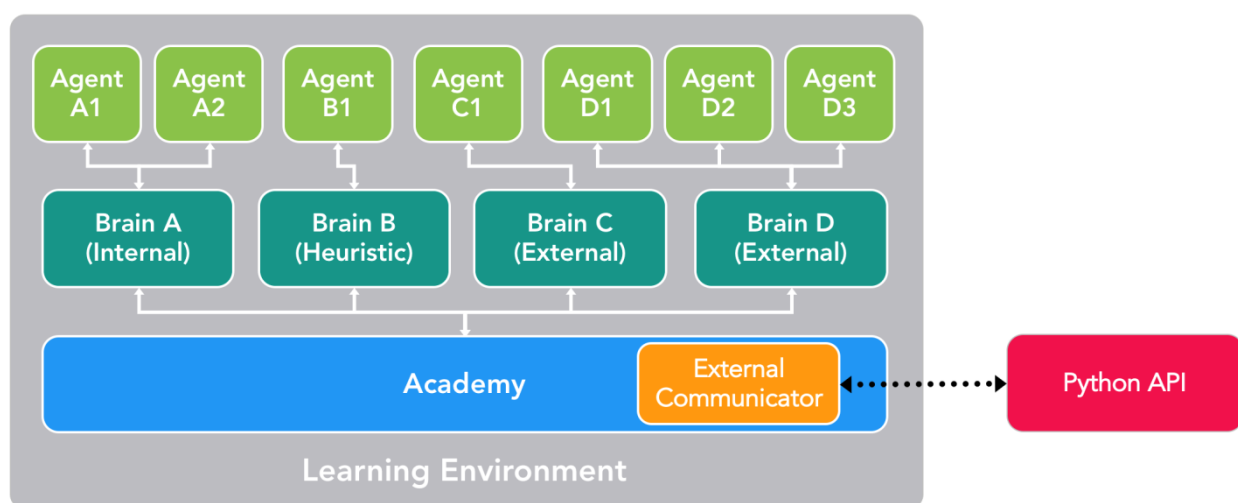


Рис. 3. Группы агентов.

Brain может находиться в 3-х режимах:

1. External - внешнем. Применяется во время обучения агентов для создания модели НС.
2. Internal – внутреннем. Используется для демонстрации работы обученных агентов, применение готовой модели НС.
3. Neuristic – эвристическом. Применяется для первоначального тестирования агентов, правильности проверки работы всей системы перед началом обучения. Тестирование проводится посредством получения входных сигналов от клавиатуры или других устройств ввода исследователя ИИ.



Для задания функционирования объектов в Unity3D используется система компонентов. Компонент представляет собой написанный на языке C# класс и после добавления к объекту начинает с ним взаимодействовать. Для описания собственных агентов и среды работы необходимо создать собственные классы, наследуемые от классов Agent и Academy соответственно.

### 3.2 Боевой робот №1

БР №1 представляет собой шарообразного робота, использующего для перемещения три ноги. Задача данного робота – находясь на ограниченной по площади платформе без ограждений достигнуть заданного объекта и не упасть вниз (выйти за пределы платформы). Объектом является сфера зелёного цвета. Робот может двигаться вперёд и назад, поворачиваться вокруг своей оси. На платформе расположен текст, отображающий накопленную агентом награду. Если БР выполняет задачу, то платформа меняет свой цвет на зелёный, иначе – на красный.

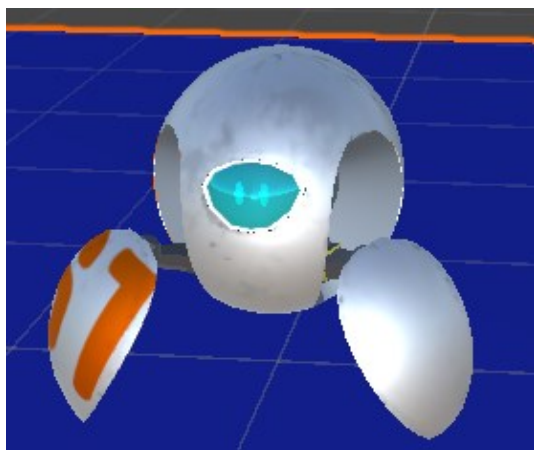


Рис. 5. БР№1 вид спереди.

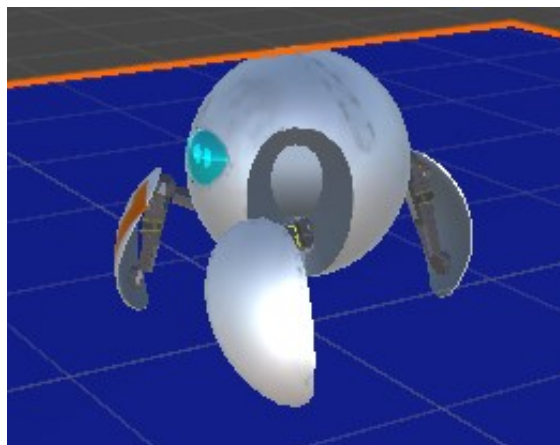


Рис. 6. БР№1 вид сбоку.

Для настройки анимации БР нужно подключить к нему показанный в окне Inspector на рисунке 7 компонент Animator, параметром которого выступает контроллер анимационных клипов, представляющий собой конечный автомат, состояниями которого являются анимационные клипы, а переходы осуществляются при выполнении условий. Условия задаются при помощи переменных четырёх типов, в данном случае все переменные имеют логический(bool) тип. Так как в действия робота не входит сворачивание,

разворачивание и перекачивание, то анимационные клипы для данных действий использоваться не будут. Но для демонстрации в первом эпизоде обучения робот разворачивается и переходит в бездействующее состояние Idle.

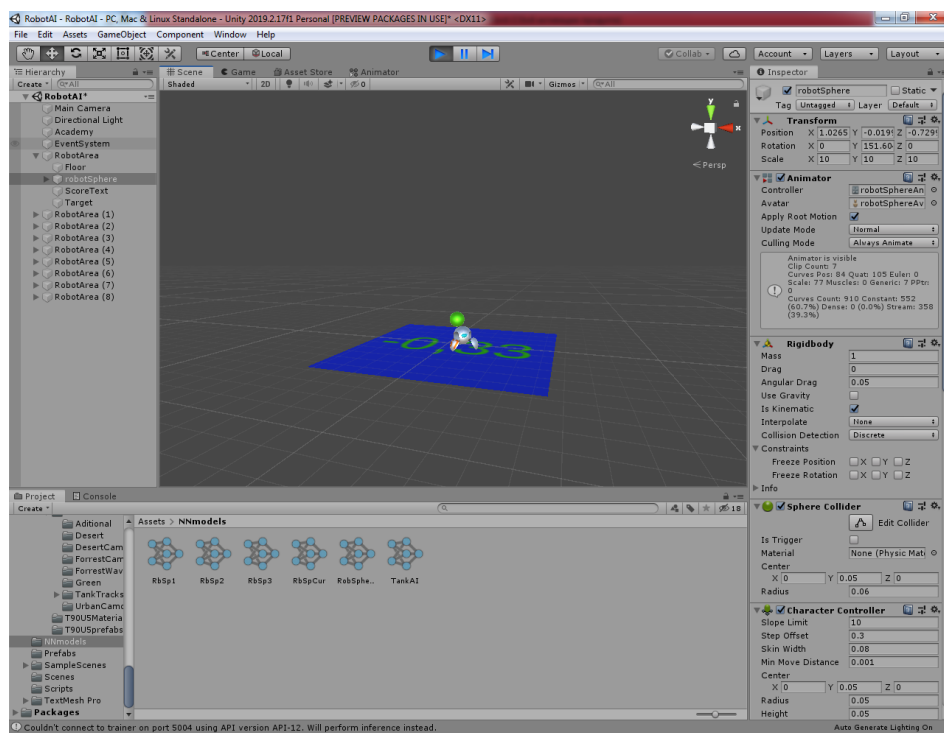


Рис. 7. Рабочее окно программы.

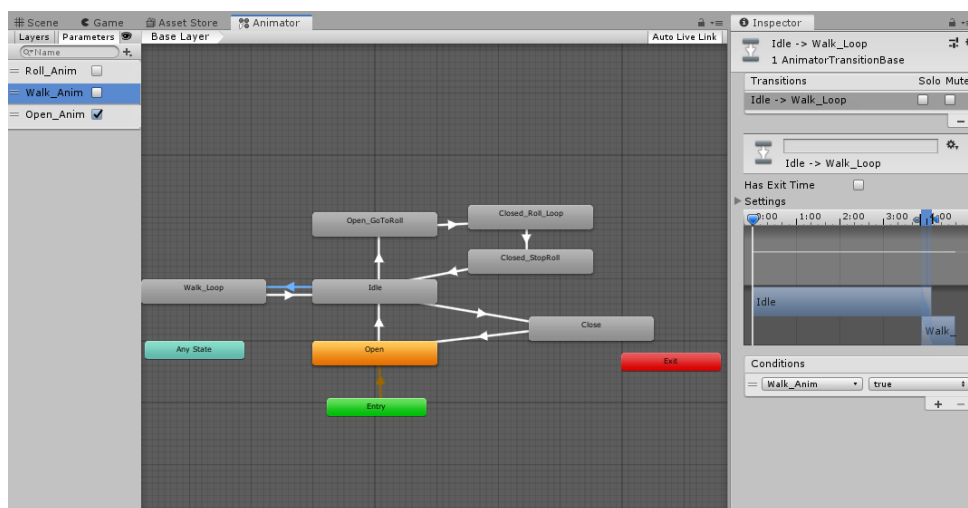


Рис. 8. Настройка анимации БР.

У любого расположенного на сцене объекта есть компонент Transform, который содержит информацию о его позиции, ориентации в пространстве и размерах. К БР добавлен компонент Rigidbody, отвечающий за физику. Компонент Sphere

Collider отвечает за физические границы БР. Цель(зелёная сфера) так же имеет эти три вышеуказанных компонента. С помощью компонента Character Controller осуществляются движения робота.

Отвечающий за действия агента класс RobotAgent наследуется от класса Agent и прикреплен к БР в виде компонента. Базовый класс Agent имеет следующие параметры:

Таб. 2. Параметры класса Agent.

Параметр	Тип/диапазон принимаемых значений	Значение
Max Step	int 0...	Максимальное кол-во действий, которое может выполнить агент в одном эпизоде. 0 – любое кол-во.
Reset On Done	bool true/false	Если true, то при встрече в коде оператора Done() автоматически вызывается функция AgentReset()
On Demand Decision	bool true/false	Если true, то агент выполняет действия в ответ на событие. Реакция на событие должна быть описана в методе RequestDecision().
Decision Interval	int 1...	Агент автоматически после каждого выполнения кол-ва таких шагов запрашивает действия и выполняет их на каждом шаге до следующего запроса.
Behavior Name	string	Имя Brain. Это значение будет присутствовать в имени файла обученной НС.
Vector Observation Space Size	1...	Кол-во наблюдений агента.
Vector Observation Stacked Vector	1 - 50	Кол-во наблюдений, которые будут помещены в стек перед отправкой в НС

Space Type	Continuous: float [-1,0] или [0,1]  Discrete: int 0 или 1	Тип входных сигналов.
Vector Action Space Size	1...	Кол-во входных сигналов
Model	Файл с расширением .nn /None	Файл nn. - обученная НС для демонстрации работы.  None - обучить и создать новую НС.
Inference Device	CPU/GPU	Проводить вычисления на CPU или GPU.
Behavior Type	Default/Heuristic Only/Inference Only	Переключение между режимами Brain.
Use Child Sensors	bool  true/false	Использовать ли сенсорные компоненты, прикреплённые к дочерним объектам данного агента.

Класс RobotAgent содержит переменные, отвечающие за скорость движения и поворота БР, действующего на него ускорения свободного падения и компонент Transform цели. Все эти переменные имеют тип public, что обеспечивает быстрый к ним доступ через интерфейс.

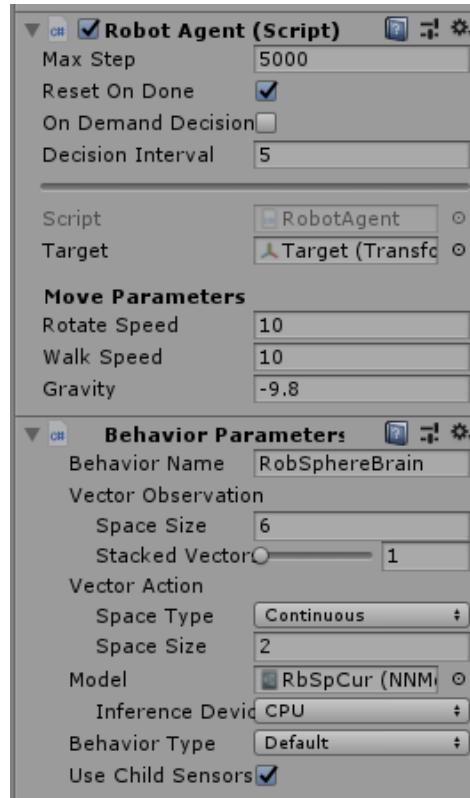


Рис. 9. Доступ к настройкам движения и обучения агента.

### 3.2.1 Модель №1.

В классе RobotAgent переопределены несколько методов родительского класса Agent:

- 1) Функция InitializeAgent() предназначена для начальной настройки агента, получения компонентов для его движения, анимации и обучения.

```
public override void InitializeAgent()
{
    characterController = GetComponent<CharacterController>();
    rb = GetComponent<Rigidbody>();
    agentArea = transform.parent.GetComponent<RobotArea>();
    RbAcademy = FindObjectOfType<RobotAcademy>();
    anim = GetComponent<Animator>();
}
```

- 2) В функции AgentReset() устанавливаются положения агента и цели при старте следующего эпизода. Если БР упал с платформы, то он возвращается в её центр с нулевой линейной и угловой скоростью.

```
public override void AgentReset()
{
    if (transform.localPosition.y < -2f)
    {
        rb.velocity = Vector3.zero;
        rb.angularVelocity = Vector3.zero;
        transform.localPosition = new Vector3(0, 0, 0);
    }
}
```

```

        target.localPosition = new Vector3( Random.Range(-5f, 5f), 0.5f, Random.Range(-5f,
5f));
    }

```

- 3) Функция `CollectObservations()` предназначена для сбора наблюдений агента. Агент получает свою позицию и координаты цели – всего 6 вещественных чисел. Любые наблюдения должны быть представлены в виде вещественных чисел и добавлены в вектор наблюдений. Кол-во наблюдений должно быть отражено в значении параметра `Vector Observation Space Size`.

```

public override void CollectObservations()
{
    AddVectorObs(transform.localPosition);
    AddVectorObs(target.localPosition);
}

```

- 4) Функция `AgentAction(float[] vectorAction)` получает в качестве параметра вектор с вещественными числами, служащими для выполнения действий агентом. Также в этой функции агент получает награду и изменяются ответственные за анимацию переменные. Если БР достиг цели, то ему присваивается награда 1, эпизод успешно завершается, вызывается функция `AgentReset()`. Если агент падает с платформы или выполняет максимальное кол-во действий, то эпизод считается неудачным и агенту присваиваются награды -0,05 и -1 соответственно. Для того чтобы БР не стоял на месте, а совершал действия, направленные на быстрое выполнение поставленной задачи, ему в каждый момент времени добавляется награда  $\frac{-1}{Max\ Step}$ . Для начала будем считать, что агент достиг цели если расстояние между ними меньше чем `target_distance = 1,42`.

```

public override void AgentAction(float[] vectorAction)
{
    rotAmount = vectorAction[0] * rotateSpeed;
    moveAmount = vectorAction[1] * walkSpeed;
    if (moveAmount != 0)
        anim.SetBool("Walk_Anim", true);
    else
        anim.SetBool("Walk_Anim", false);

    Vector3 rotateVector = transform.up * rotAmount;
    Quaternion direction = Quaternion.Euler(transform.localPosition.eulerAngles +
rotateVector * rotateSpeed);
    transform.localPosition = Quaternion.Lerp(transform.localPosition, direction,
rotateSpeed * Time.deltaTime);

    Vector3 movement = transform.forward * moveAmount;
    movement.y = gravity;
    movement = Vector3.ClampMagnitude(movement, walkSpeed);
    movement *= Time.deltaTime;
    characterController.Move(movement);
}

```

```

        float distanceToTarget = Vector3.Distance(transform.localPosition,
target.localPosition);
        if (distanceToTarget < target_distance)
        {
            SetReward(1f);
            agentArea.UpdateScore(GetCumulativeReward());
            Done();
            StartCoroutine(agentArea.SwapGroundMaterial(success: true));
        }
        else
        {
            AddReward(-1f / agentParameters.maxStep);
            agentArea.UpdateScore(GetCumulativeReward());
        }

        if (transform.localPosition.y < -2f)
        {
            SetReward(-1f);
            Done();
            StartCoroutine(agentArea.SwapGroundMaterial(success: false));
        }
    }
}

```

- 5) Функция `Heuristic()` служит для тестирования работы агента при помощи клавиатуры, значения от которой поступают в аргумент `vectorAction` функции `AgentAction()`. Для поворота БР вокруг своей оси используется ось «Horizontal», а для движения робота вперёд и назад используется ось «Vertical». Осям соответствуют клавиши ←(A), →(D) и ↑(W), ↓(S). Кол-во управляющих сигналов должно быть указано в параметре `Vector Action Space Size`.

```

public override float[] Heuristic()
{
    var action = new float[2];
    action[0] = Input.GetAxis("Horizontal");
    action[1] = Input.GetAxis("Vertical");
    return action;
}

```

Для обучения агента нужно открыть командную строку Windows и перейти в папку где утановлены конфигурационные настройки:

C:\Users\User>g:

G:\>cd G:\UnityAI\config

Далее нужно ввести команду для тренировки БР:

G:\UnityAI\config>mlagents-learn trainer\_config.yaml --run-id=RbSp0 --train

где `trainer_config.yaml` – файл содержащий гиперпараметры обучения нейронной сети, `RbSp0` – идентификатор обучения.

После чего будет выведена информация о параметрах и приглашение нажать на кнопку Play, чтобы запустить обучение.

```
C:\Windows\system32\cmd.exe
G:\Unity0\config\nlagents-learn trainer.config.yaml --run-id=RbSp0 --train
Error processing line 7 of g:\anaconda\lib\site-packages\pywin32.pth:

Traceback (most recent call last):
  File "g:\anaconda\lib\site.py", line 168, in addpackage
    exec(line)
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'pywin32_bootstrap'

Remainder of file ignored
WARNING:tensorflow:From g:\anaconda\lib\site-packages\tensorflow_core\python\co
mpat\v2_compat.py:65: disable_resource_variables (from tensorflow.python.ops.var
iable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term

Unity

Version information:
  nl-agents: 0.12.1,
  nl-agents-envs: 0.12.1,
  Communicator API: API-12,
  TensorFlow: 2.0.0
INFO:nlagents.trainers:CommandLineOptions(debug=False, num_runs=1, seed=-1, env_
path=None, run_id='RbSp0', load_model=False, train_model=True, save_freq=50000,
keep_checkpoints=5, base_port=5005, num_envs=1, curriculum_folder=None, lesson=0
, no_graphics=False, multi_gpu=False, trainer_config_path='trainer.config.yaml',
sampler_file_path=None, docker_target_name=None, env_args=None, cpu=False, widt
h=64, height=64, quality_level=5, time_scale=20, target_frame_rate=1)
Error processing line 7 of g:\anaconda\lib\site-packages\pywin32.pth:

Traceback (most recent call last):
  File "g:\anaconda\lib\site.py", line 168, in addpackage
    exec(line)
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'pywin32_bootstrap'

Remainder of file ignored
INFO:nlagents.envs:Listening on port 5004. Start training by pressing the Play b
utton in the Unity Editor.
INFO:nlagents.envs:Connected new brain:
RobSphereBrain
INFO:nlagents.trainers:Hyperparameters for the PPOTrainer of brain RobSphereBrai
n:
  trainer:
    batch_size: 1024
    beta: 0.005
    buffer_size: 10240
    epsilon: 0.2
    hidden_units: 128
    lambda: 0.95
    learning_rate: 0.0003
    learning_rate_schedule: linear
    max_steps: 5.0e4
    memory_size: 256
    normalize: False
    num_epoch: 3
    num_layers: 2
    time_horizon: 64
    sequence_length: 64
    summary_freq: 1000
    use_recurrent: False
```

Рис. 10. Запуск процесса обучения.

Mean Reward – средняя награда всех агентов, Std.of Reward – среднеквадратичное отклонение (standard deviation) награды, Step – кол-во прошедших шагов с начала обучения, Time Elapsed – время в секундах прошедшее с начала обучения.



```
Ca\Windows\system32\cmd.exe
summary_freq: 1000
use_recurrent: False
vis_encode_type: simple
reward_signals:
  extrinsic:
    strength: 1.0
    gamma: 0.99
  summary_path: RbSp0_RobSphereBrain
  model_path: ./models/RbSp0-0/RobSphereBrain
keep_checkpoints: 5
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 1000. Time Elapsed: 22.866
s Mean Reward: 0.489. Std of Reward: 0.652. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 2000. Time Elapsed: 31.960
s Mean Reward: 0.023. Std of Reward: 0.541. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 3000. Time Elapsed: 39.579
s Mean Reward: -0.575. Std of Reward: 0.425. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 4000. Time Elapsed: 47.120
s Mean Reward: -0.471. Std of Reward: 0.181. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 5000. Time Elapsed: 54.529
s Mean Reward: -0.116. Std of Reward: 0.722. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 6000. Time Elapsed: 61.961
s Mean Reward: -0.357. Std of Reward: 0.118. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 7000. Time Elapsed: 69.347
s Mean Reward: 0.186. Std of Reward: 0.590. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 8000. Time Elapsed: 77.217
s Mean Reward: 0.198. Std of Reward: 0.635. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 9000. Time Elapsed: 84.920
s Mean Reward: -0.453. Std of Reward: 0.057. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 10000. Time Elapsed: 93.039
s Mean Reward: 0.384. Std of Reward: 0.507. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 11000. Time Elapsed: 102.82
s Mean Reward: -0.045. Std of Reward: 0.500. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 12000. Time Elapsed: 111.92
s Mean Reward: 0.191. Std of Reward: 0.606. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 13000. Time Elapsed: 119.55
s Mean Reward: -0.180. Std of Reward: 0.409. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 14000. Time Elapsed: 127.32
s Mean Reward: 0.205. Std of Reward: 0.486. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 15000. Time Elapsed: 134.94
s Mean Reward: -0.002. Std of Reward: 0.518. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 16000. Time Elapsed: 142.77
s Mean Reward: 0.127. Std of Reward: 0.618. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 17000. Time Elapsed: 150.16
s Mean Reward: 0.024. Std of Reward: 0.673. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 18000. Time Elapsed: 157.96
s Mean Reward: -0.411. Std of Reward: 0.000. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 19000. Time Elapsed: 165.82
s Mean Reward: -0.967. Std of Reward: 0.000. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 20000. Time Elapsed: 173.47
s Mean Reward: 0.234. Std of Reward: 0.736. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 21000. Time Elapsed: 182.34
s Mean Reward: 0.704. Std of Reward: 0.182. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 22000. Time Elapsed: 189.98
s Mean Reward: 0.050. Std of Reward: 0.649. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 23000. Time Elapsed: 197.81
s Mean Reward: 0.369. Std of Reward: 0.589. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 24000. Time Elapsed: 207.72
s Mean Reward: 0.151. Std of Reward: 0.656. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 25000. Time Elapsed: 216.27
s Mean Reward: 0.163. Std of Reward: 0.679. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 26000. Time Elapsed: 225.42
s Mean Reward: -0.240. Std of Reward: 0.137. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 27000. Time Elapsed: 233.00
s Mean Reward: 0.338. Std of Reward: 0.420. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 28000. Time Elapsed: 246.82
s Mean Reward: 0.240. Std of Reward: 0.061. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 29000. Time Elapsed: 254.61
s Mean Reward: -0.504. Std of Reward: 0.226. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 30000. Time Elapsed: 262.62
s Mean Reward: -0.174. Std of Reward: 0.036. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 31000. Time Elapsed: 272.77
s Mean Reward: -0.293. Std of Reward: 0.360. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 32000. Time Elapsed: 282.36
s Mean Reward: 0.163. Std of Reward: 0.663. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 33000. Time Elapsed: 292.32
s Mean Reward: 0.259. Std of Reward: 0.603. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 34000. Time Elapsed: 300.58
s Mean Reward: 0.259. Std of Reward: 0.634. Training.
```

Рис. 11. Отображаемая информация о время обучения.

```
Ca\Windows\system32\cmd.exe --logdir=summaries --port=6006
3 s Mean Reward: -0.031. Std of Reward: 0.431. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 36000. Time Elapsed: 315.58
s Mean Reward: 0.079. Std of Reward: 0.323. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 37000. Time Elapsed: 322.81
s Mean Reward: -0.011. Std of Reward: 0.843. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 38000. Time Elapsed: 330.07
s Mean Reward: -0.059. Std of Reward: 0.154. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 39000. Time Elapsed: 337.35
s Mean Reward: 0.172. Std of Reward: 0.566. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 40000. Time Elapsed: 344.66
s Mean Reward: -0.399. Std of Reward: 0.189. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 41000. Time Elapsed: 352.56
s Mean Reward: 0.147. Std of Reward: 0.628. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 42000. Time Elapsed: 362.49
s Mean Reward: 0.474. Std of Reward: 0.612. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 43000. Time Elapsed: 373.11
s Mean Reward: 0.106. Std of Reward: 0.764. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 44000. Time Elapsed: 384.11
s Mean Reward: 0.180. Std of Reward: 0.537. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 45000. Time Elapsed: 394.31
s Mean Reward: 0.136. Std of Reward: 0.503. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 46000. Time Elapsed: 410.85
s Mean Reward: -0.289. Std of Reward: 0.068. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 47000. Time Elapsed: 423.13
s Mean Reward: 0.041. Std of Reward: 0.000. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 48000. Time Elapsed: 434.42
s Mean Reward: -0.145. Std of Reward: 0.645. Training.
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 49000. Time Elapsed: 443.93
s Mean Reward: -0.618. Std of Reward: 0.311. Training.
INFO:nlagents.trainers: Saved Model
INFO:nlagents.trainers: RbSp0: RobSphereBrain: Step: 50000. Time Elapsed: 454.07
s Mean Reward: 0.206. Std of Reward: 0.805. Training.
INFO:nlagents.trainers: List of nodes to export for brain :RobSphereBrain
INFO:nlagents.trainers: is_continuous_control
INFO:nlagents.trainers: version_number
INFO:nlagents.trainers: memory_size
INFO:nlagents.trainers: action_output_shape
INFO:nlagents.trainers: action_probs
INFO:nlagents.trainers: action_probs
Converting ./models/RbSp0-0/RobSphereBrain/frozen_graph_def.pb to ./models/RbSp0-0/RobSphereBrain.nn
IGNORED: StopGradient unknown layer
GLOBS: 'is_continuous_control', 'version_number', 'memory_size', 'action_output_shape'
IN: 'vector_observation': [-1, 1, 1, 61 => 'main_graph_0/hidden_0/BiasAdd']
IN: 'epsilon': [-1, 1, 1, 2] => 'mul'
OUT: 'action', 'action_probs'
DONE: wrote ./models/RbSp0-0/RobSphereBrain.nn file.
INFO:nlagents.trainers: Exported ./models/RbSp0-0/RobSphereBrain.nn file
G:\Unity\AI\config\tensorboard --logdir=summaries --port=6006
Error processing line 7 of g:\anaconda\lib\site-packages\pywin32.pth:
Traceback (most recent call last):
  File "g:\anaconda\lib\site.py", line 168, in addpackage
    exec(line)
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'pywin32_bootstrap'
Remainder of file ignored
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind all
TensorBoard 2.0.2 at http://localhost:6006/ (Press CTRL+C to quit)
```

Рис. 12. Отображаемая информация после завершения обучения.

После завершения обучения можно посмотреть статистику обучения в браузере при помощи программы визуализации данных Tensorboard.  
tensorboard --logdir=summaries --port=6006

Введём эту команду и перейдём на <http://localhost:6006/> для просмотра результатов. Графики приведены на рисунках 13-16 и показывают зависимость следующих величин от кол-ва шагов обучения:

1. Cumulative Reward – Средняя награда, полученная всеми агентами за эпизод. Должна возрастать при успешном обучении.
2. Episod Length – Средняя длина эпизода обучения всех агентов. Для данной задачи при успешном обучении должна убывать, так как БР нужно как можно быстрее достичь зелёной сферы.
3. Policy Loss – Среднее значение функции потерь. Должно убывать при успешном обучении.
4. Value Loss – Среднее значение ошибки функции оценки состояния. При успешном обучении должно сначала возрасть, а затем, когда награда начнёт стабилизироваться, убывать.
5. Entropy – Энтропия. Показывает насколько случайными являются действия агента. При успешном обучении должна медленно убывать.
6. Existing Reward – Среднее значение внешней награды всех агентов.
7. Extrinsic Value Estimate – Средняя оценка всех посещенных агентом состояний. При успешном обучении должна возрасть.
8. Learning Rate – Величина шага в алгоритме градиентного спуска. При успешном обучении должна убывать.

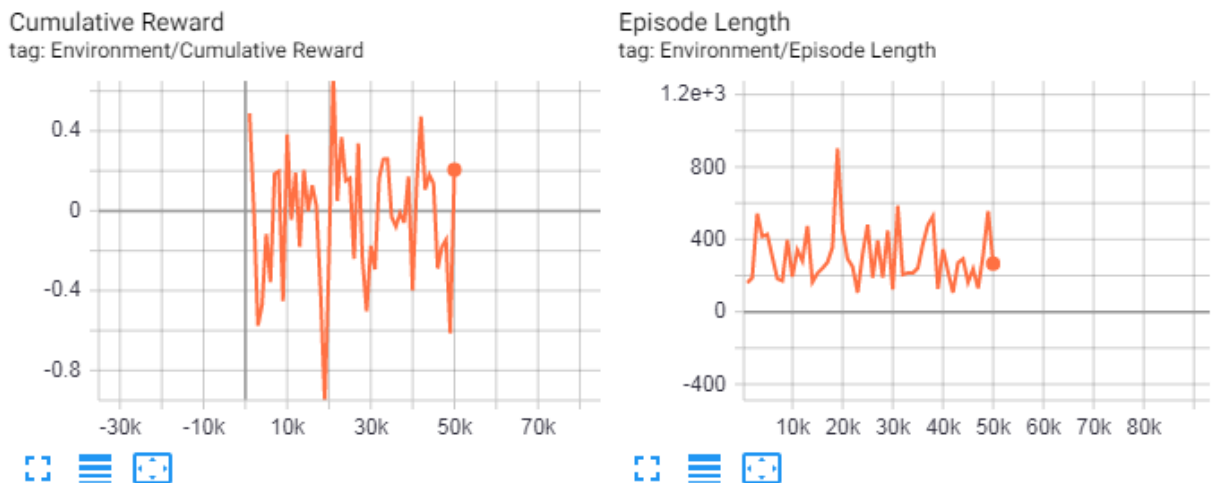


Рис. 13. Графики Cumulative Reward и Episode Length.



Рис. 14. Графики Policy Loss и Value Loss.

Policy

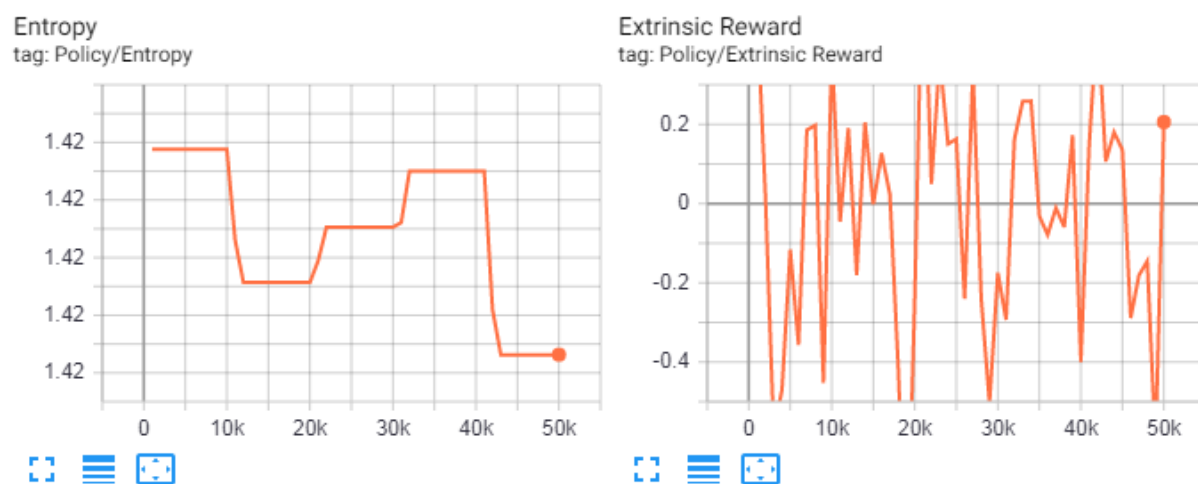


Рис. 15. Графики Entropy и Extrinsic Reward.

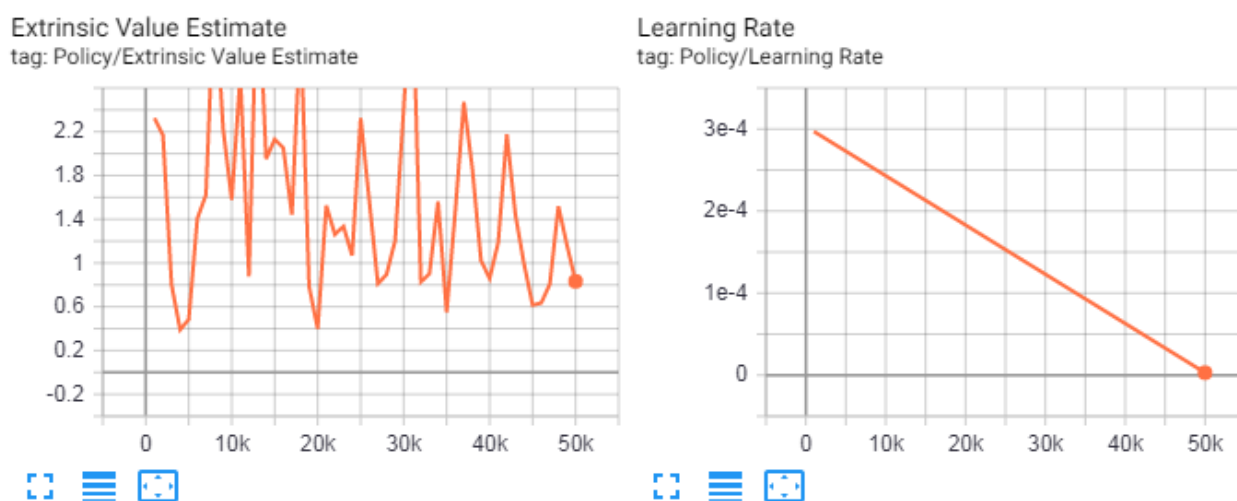


Рис. 16. Графики Extrinsic Value Estimate и Learning Rate.

Для изменения параметров обучения нужно открыть файл, в котором они содержатся. В данном случае `trainer_config.yaml`.

```

default:
  trainer: ppo
  batch_size: 1024
  beta: 5.0e-3
  buffer_size: 10240
  epsilon: 0.2
  hidden_units: 128
  lambda: 0.95
  learning_rate: 3.0e-4
  learning_rate_schedule: linear
  max_steps: 5.0e4
  memory_size: 256
  normalize: false
  num_epoch: 3
  num_layers: 2
  time_horizon: 64
  sequence_length: 64
  summary_freq: 1000
  use_recurrent: false
  vis_encode_type: simple
  reward_signals:
    extrinsic:
      strength: 1.0
      gamma: 0.99
FoodCollector:
  normalize: false
  beta: 5.0e-3
  batch_size: 1024
  buffer_size: 10240
  max_steps: 1.0e5
Bouncer:
  normalize: true
  max_steps: 1.0e6
  num_layers: 2
  hidden_units: 64
PushBlock:
  max_steps: 5.0e4
  batch_size: 128
  buffer_size: 2048
  beta: 1.0e-2
  hidden_units: 256
  summary_freq: 2000
  time_horizon: 64
  num_layers: 2

```

Рис. 17.Файл `trainer_config.yaml`

Таб. 3.Парметры обучения НС.

Параметр	Описание	Принимаемые значения
batch_size	Размер батча. Кол-во опыта агента(наблюдений ,действий и наград) в каждой итерации градиентного спуска.	Всегда должно быть делителем buffer_size.  Continuous:512-5120  Discrere:32-512
beta	Сила регуляризации энтропии. Влияет на кол-во случайных действий агента.	Если энтропия убывает слишком быстро, то следует увеличить beta.Если слишком медленно – уменьшить beta.  $10^{-4} \dots 10^{-2}$
buffer_size	Кол-во наблюдений, которые нужно собрать для обновления поведения модели.	Должно делиться нацело на batch_size. Обычно большие значения обеспечивают более стабильные обновления модели.

		2048-409600
epsilon	Допустимый порог расхождения между старыми и новыми поведением в обновлениях градиентного спуска.	Малые значения сделают обновления более стабильными, но замедлят процесс обучения.  0.1-0.3
hidden_units	Кол-во нейронов в каждом слое НС.	Для задач, в которых правильные действия зависят от простых комбинаций наблюдений, значение hidden_units должно быть небольшим.  32...512
lambd	Насколько агент полагается на полученные награды для получения значения ожидаемой награды.	0.9...0.95
learning_rate	Начальный размер шага в алгоритме градиентного спуска.	Должен быть уменьшен, если обучение нестабильно и награда не увеличивается.  $10^{-5} \dots 10^{-3}$
max_steps	Кол-во шагов в тренировочном процессе.	Должно быть большим для сложных задач.  $5 * 10^5 \dots 10^7$
normalize	Нормализовать ли значения в векторе наблюдений.	true/false
num_epoch	Кол-во проходов через буфер наблюдений в алгоритме градиентного спуска.	Небольшое значение приведёт к более стабильному, но долгому обучению.  3-10
num_layers	Кол-во скрытых слоёв в НС.	Небольшое значение подходит для простых задач.  1-3
behavioral_cloning	Использование демонстрации для обучения агента	demo_path – путь к файлу с демонстрацией  strength – сила влияний демонстрации на обучение  steps – кол-во шагов в обучении, отводимых для обучения с демонстрациями

reward_signals	Сигналы награждений	extrinsic , curiosity, gail
summary_freq	Через какое кол-во шагов каждый раз сохранять статистику обучения.	1000-5000
time_horizon	Какое кол-во опыта агента необходимо собрать для помещения в буфер	32-2048
trainer	Алгоритм обучения	ppo, sac, offline_bc, online_bc

После завершения обучения агента, создаётся файл модели НС, представляющий собой граф данных Tensorflow и содержащий математические операции и оптимизированные веса.

Для применения полученной модели следует перейти в config\models\RbSp0-0, выбрать файл с расширением nn, претащить его в проект и поместить во вкладку Model компонента Behavior Parameters. После чего нажать на кнопку Play.

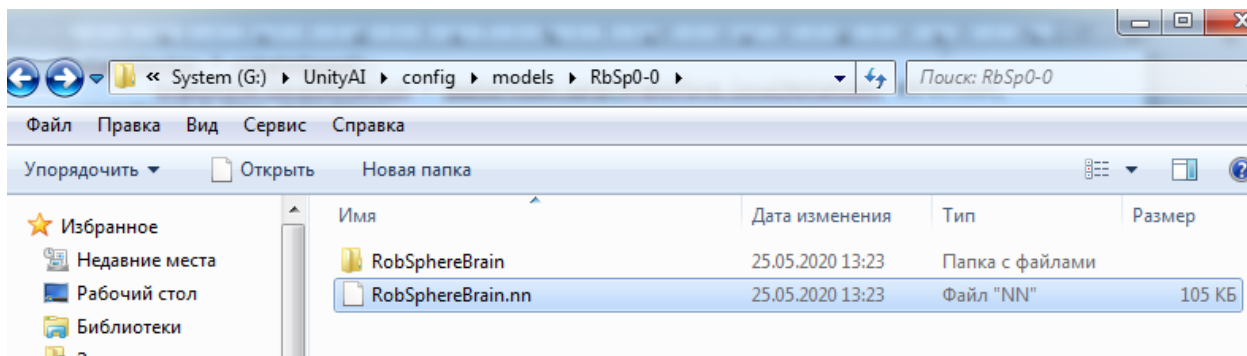


Рис. 18.Файл trainer\_config.yaml

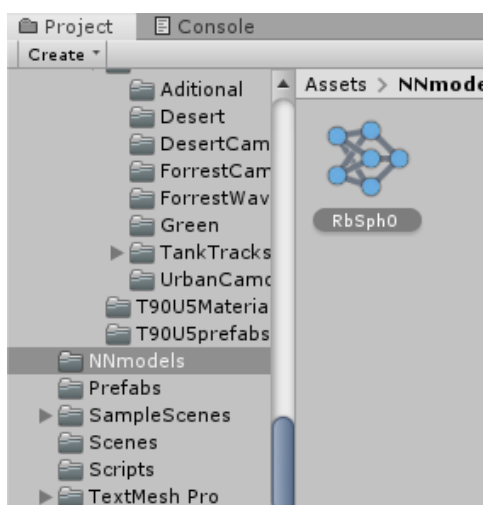


Рис. 19. Иконка модели НС



Рис. 20. Информация о модели НС

### 3.2.2 Модель №2

Чтобы ускорить обучение модели №1 можно создать ещё несколько сред с агентами и запустить их параллельно. Также улучшается поведение БР, так как каждый агент исследует пространство и действует не зависимо от других.

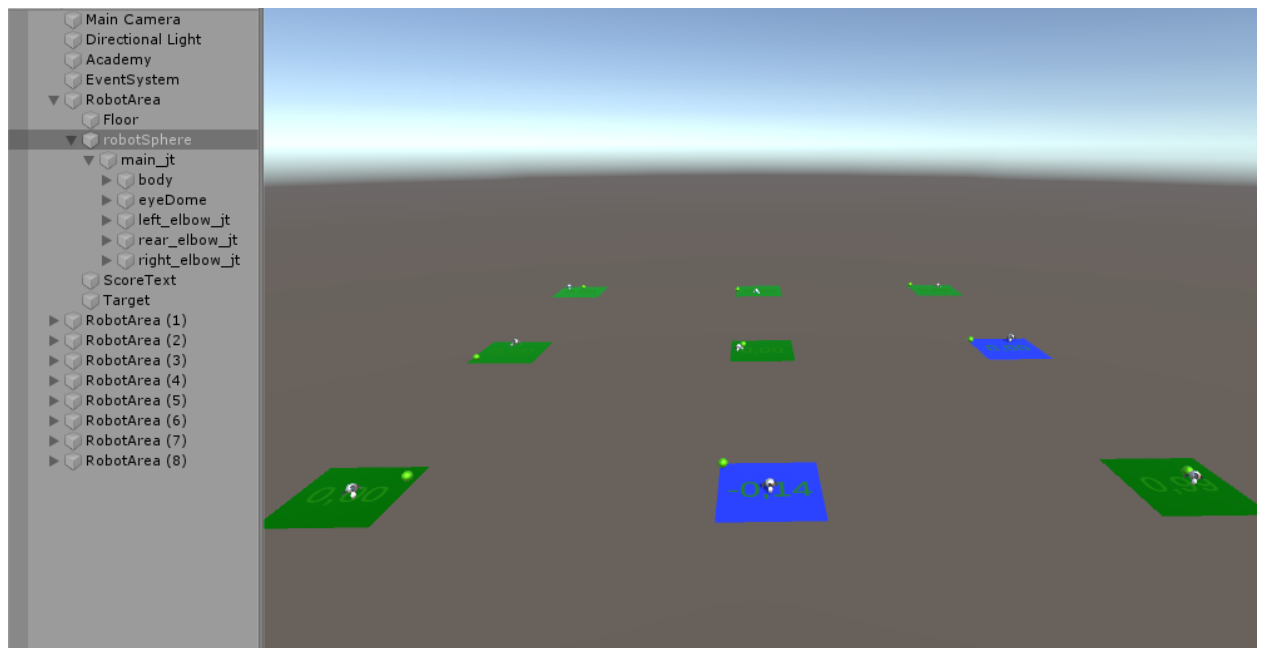


Рис. 21. Параллельное обучение агентов.

### 3.2.3 Модель №3

Добавим к модели №2 дополнительные наблюдения: направление движения агента +3, расстояние от агента до цели +1, нормализованное направление от агента до цели +3. Всего  $6+7 = 13$  наблюдений. Теперь функция наблюдений будет выглядеть так:

```
public override void CollectObservations()
{
    AddVectorObs(target.localPosition);
    AddVectorObs(transform.localPosition);

    AddVectorObs(transform.forward);
    AddVectorObs(Vector3.Distance(target.transform.position,
transform.position));
    AddVectorObs((target.transform.position - transform.position).normalized);
}
```

Нужные наблюдения помогут БР лучше понять исследуемое пространство.

### 3.2.4 Модель №4

Пусть теперь робот заранее не знает, где находится цель, он должен сам определить её местоположение с помощью датчиков. В роли датчиков будут выступать лучи, которые действуют на заданном расстоянии и определяют встретился ли им на пути объект с заданным тегом и на каком расстоянии. Чтобы получить эти лучи нужно прикрепит к агенту компонент



RayPerceptioSensorComponent3D. Назначим цели тег target, разместим 10 лучей с длиной 10. Информация от лучей автоматически добавляется к наблюдениям агента.

```
public override void CollectObservations()
{
    AddVectorObs(transform.localPosition);
    AddVectorObs(transform.forward);
}
```



Рис. 22.Компонент RayPerceptionSensor3D.

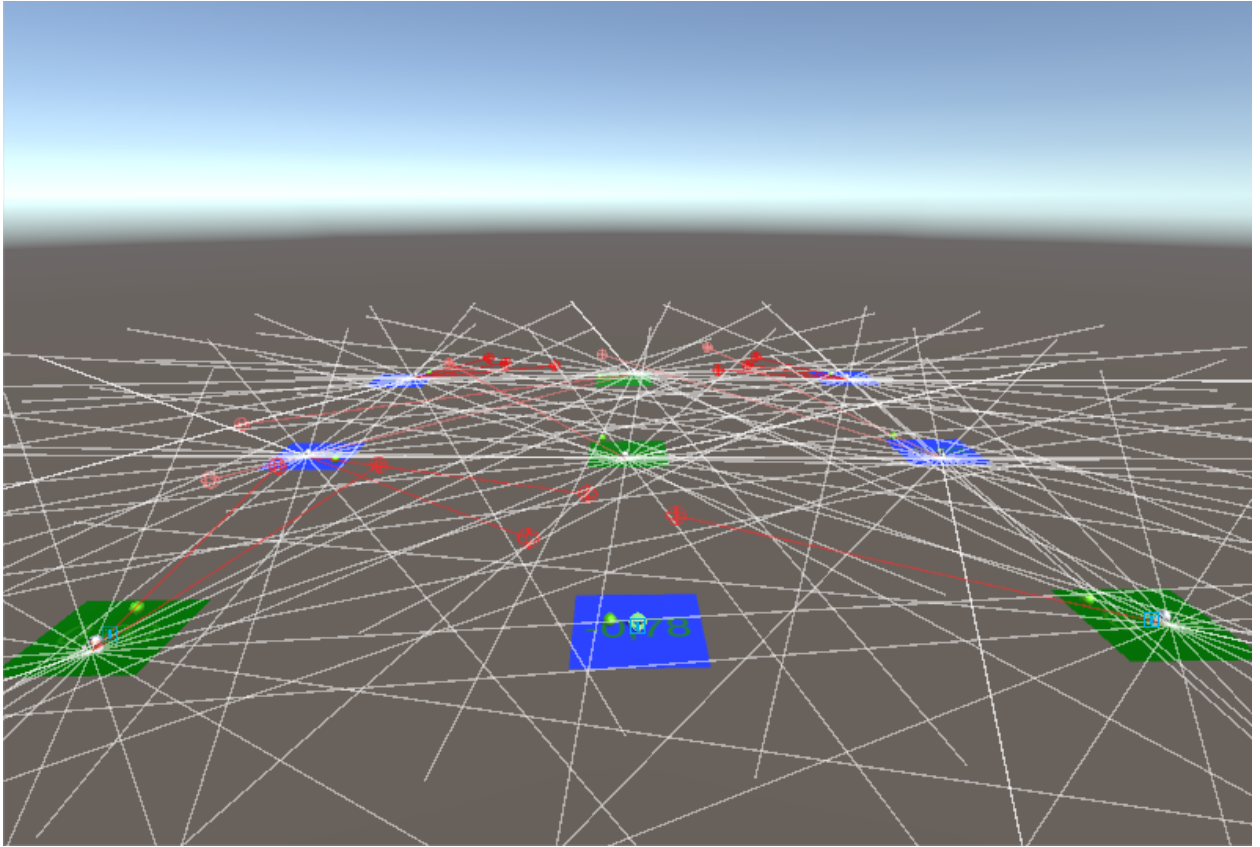


Рис. 23.БР с помощью лучей находят цель.

### 3.2.5 Модель №5. Curriculum learning

Основная идея Curriculum learning заключается в последовательном обучении от простого к более сложному. Вначале сделаем значение `target_distance` таким большим, чтобы агенту достаточно было сделать минимум движений по направлению к цели. Затем будем постепенно уменьшать это расстояние до нуля. В итоге агент должен быстрее понять взаимосвязь между наградой и выполняемыми им действиями.

Для обучения введём следующую строку:

```
G:\UnityAI\config>magents-learn trainer_config.yaml --curriculum curricula/RbSp --run-id=RbSP4 --train
```

где после `--curriculum` следует имя файла с изменяемыми параметрами.

```

1 {
2     "measure" : "reward",
3     "thresholds" : [0.1, 0.3, 0.5, 0.7, 0.8, 0.9],
4     "min_lesson_length" : 100,
5     "signal_smoothing" : true,
6     "parameters" :
7     {
8         "target_distance" : [3.0, 2.5, 2.0, 1.5, 1.0, 0.5, 0.0]
9     }
10 }
11
12

```

Рис. 24.Файл с параметрами для Curriculum learning.

Таб. 4. Параметры обучения для Curriculum learning.

Параметр	Значение
measure	Смотреть для перехода к следующему уроку на: reward – среднюю награду агентов progress – кол-во выполненных шагов
thresholds	Значения барьеров для переходов к следующим урокам
min_lesson_length	Если в measure выбрано: reward – среднее значение награды в этом кол-ве последних эпизодов будет определять следует ли перейти к следующему уроку. progress – минимальное кол-во завершённых эпизодов для перехода к следующему уроку.
Signal_smoothing	Если true , то будет обновлено 75% весов в следующем уроке.
parameters	Изменяемые параметры в обучении

Таким образом, если среднее значение полученной всеми агентами награды в последних 100 уроках больше ограничения(threshold), то агент переходит к новому уроку. Вначале обучения target\_distance=3, и для перехода к следующему уроку требуется, чтобы значение Cumulative reward было больше 0.1.В последнем уроке target\_distance = 0. БР придётся соприкоснуться с целью, чтобы получить свою награду.

Для получения значения target\_distance из файла нужно добавить следующую строку в функцию AgentReset():

```
target_distance= RbAcademy.FloatProperties.GetPropertyWithDefault("target_distance",
1.42f);
```

На рисунке 27 график Lesson показывает прогресс обучения. Приблизительно после 8 тыс. шагов БР перешёл к последнему 6-ому уроку.

### 3.2.6 Статистика моделей

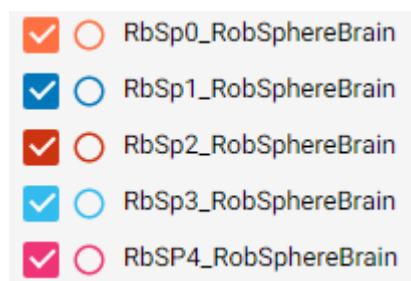
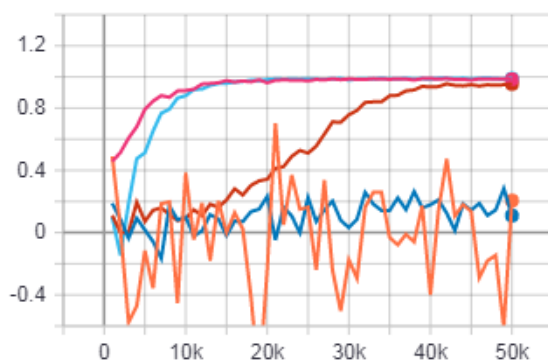


Рис. 25.Обозначение моделей.

Environment

Cumulative Reward  
tag: Environment/Cumulative Reward



Episode Length  
tag: Environment/Episode Length

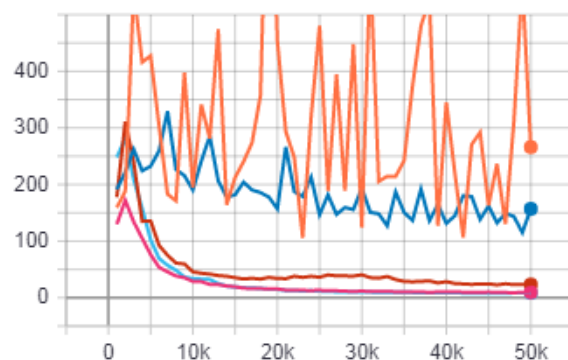


Рис. 26.Графики Cumulative Reward и Episode Length всех моделей.

Lesson  
tag: Environment/Lesson

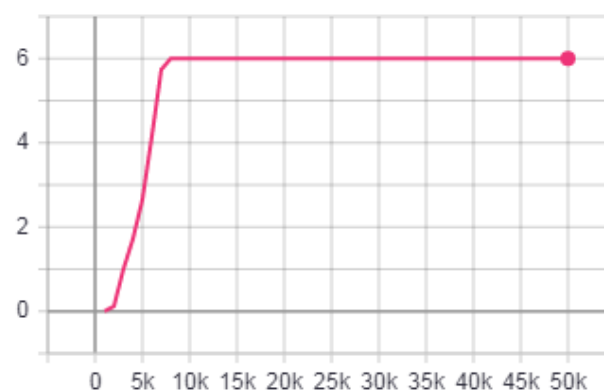
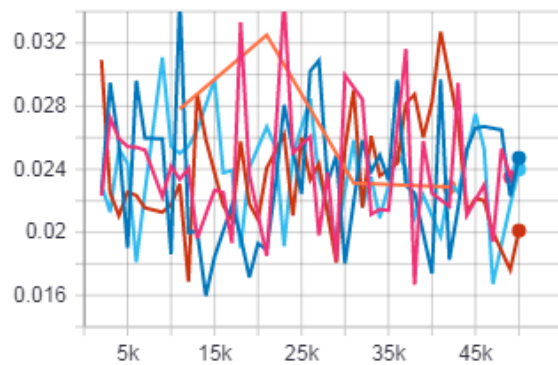


Рис. 27.График Lesson.

## Losses

Policy Loss  
tag: Losses/Policy Loss



Value Loss  
tag: Losses/Value Loss

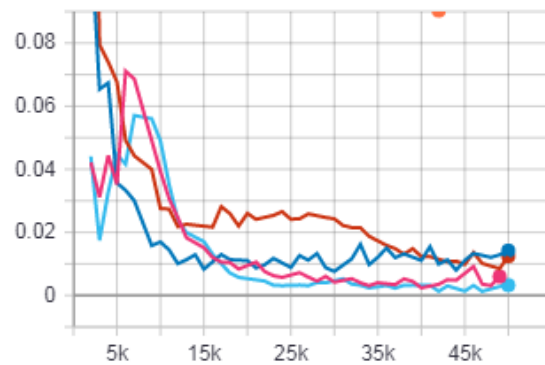
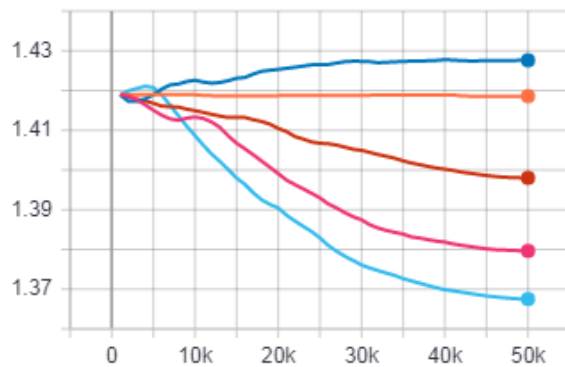


Рис. 28.Графики Policy Loss и Value Loss всех моделей.

## Policy

Entropy  
tag: Policy/Entropy



Extrinsic Reward  
tag: Policy/Extrinsic Reward

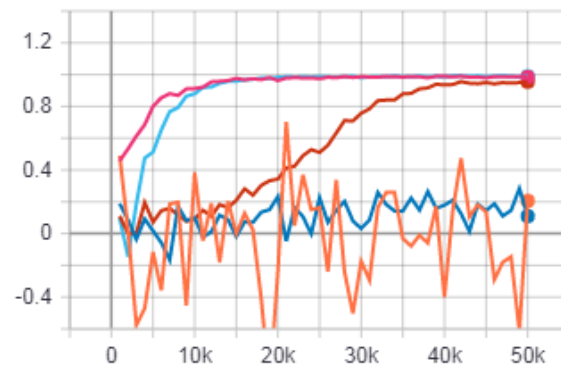


Рис. 29.Графики Entropy и Extrinsic Reward всех моделей.

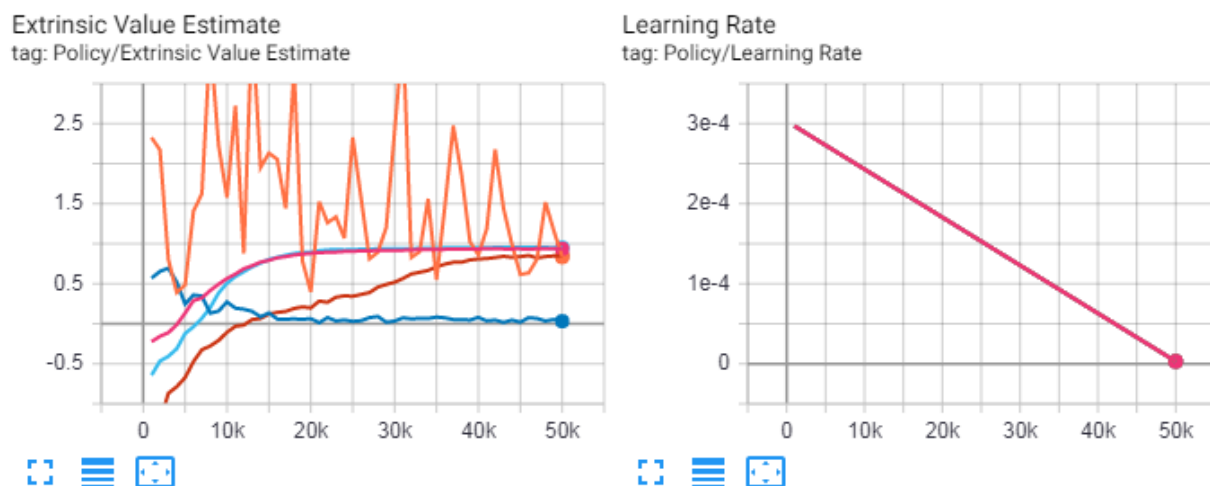


Рис. 30. Графики Extrinsic Value Estimate и Learning Rate всех моделей.

### 3.3 Боевой робот №2

БР №2 представляет собой российский танк Т-90. Данный БР может двигаться вперёд и назад, поворачивать корпус и башню влево и вправо, наклонять ствол вверх и вниз, стрелять из орудия. Задачей танка является нахождение цели и её уничтожение тараном или снарядом. Перезарядка орудия составляет 2с. и может быть легко изменена в окне Inspector. Целью является куб. Во время движения танка крутятся катки и гусеницы, а при выстреле из ствола вырывается пламя. При нажатии на клавишу «Q» меняется вид наблюдения и управления танком. Есть вид от места механика-водителя, командира, наводчика, вид сзади танка и вид сверху на всю карту.

БР находится на большой по площади местности, обнесённой по краям высокими скалами. В центральной части среды расположена равнина. Есть холмы, трава и деревья.



Рис. 31. Танк на местности

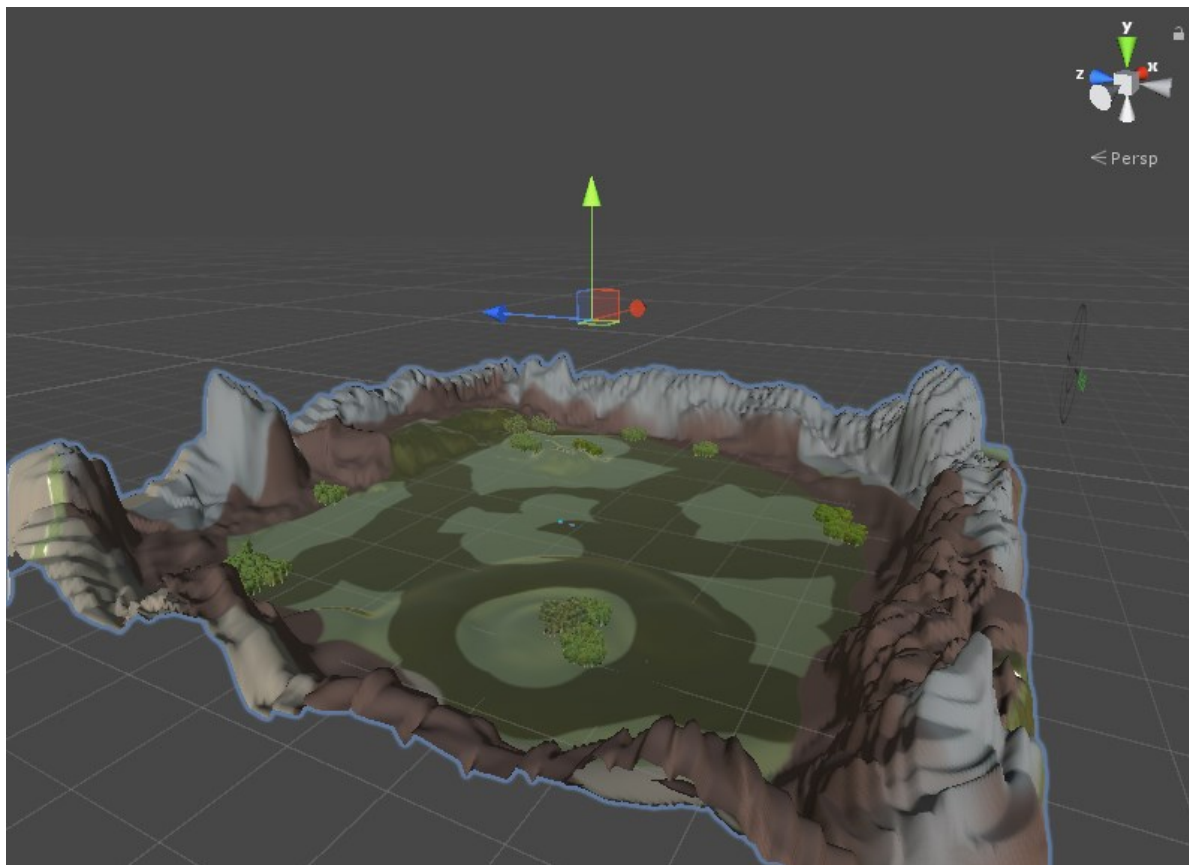


Рис. 32. Окружающая среда



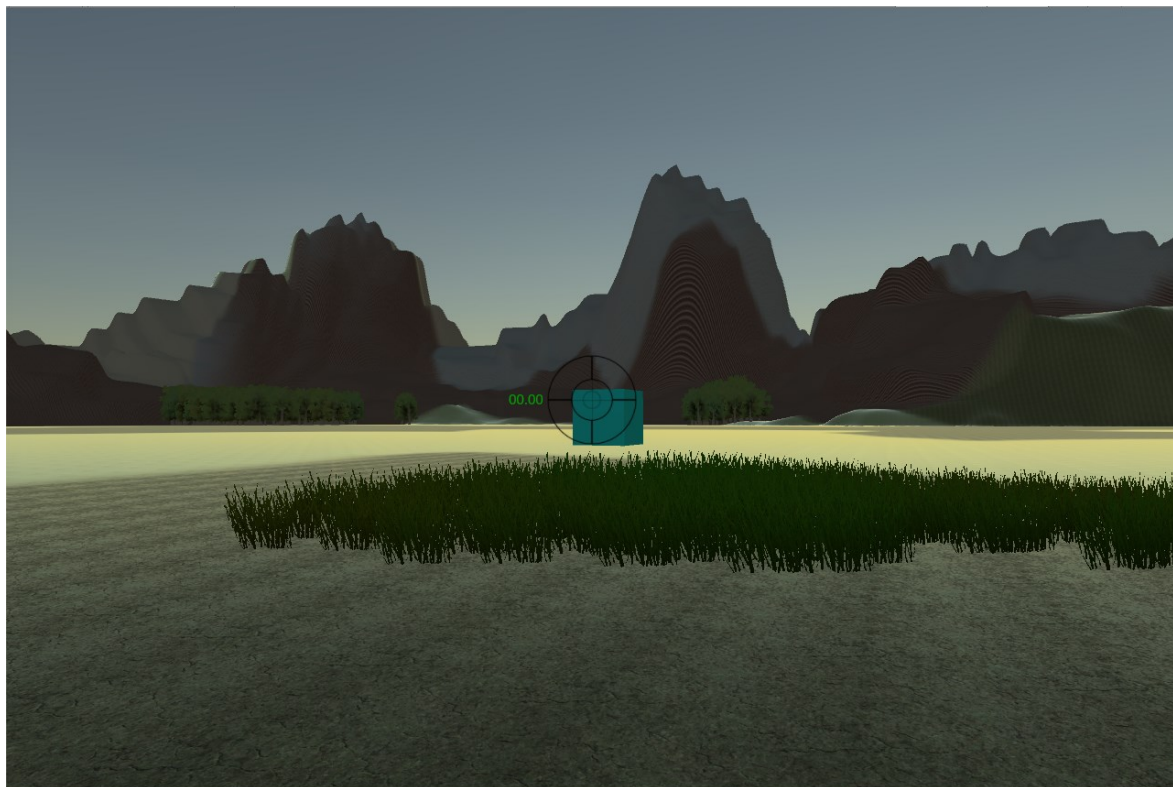


Рис. 33. Вид от лица наводчика танка

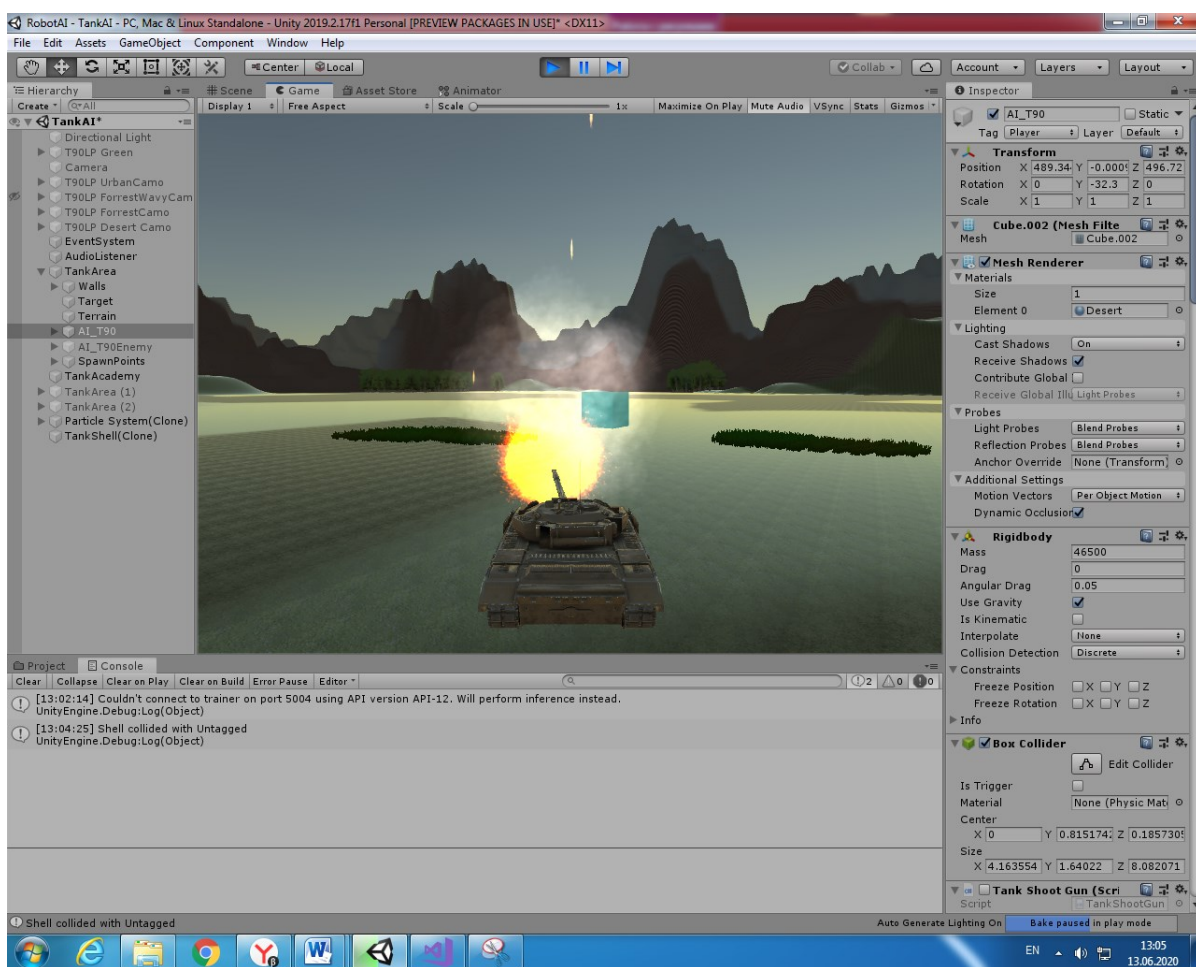


Рис. 34. Анимация выстрела



Танк получает информацию о своей позиции, направлении движения, местоположении конца ствола, направлении возможного выстрела и готовности произвести выстрел – всего 13 наблюдений. И при помощи нескольких компонентов RayPerceptionSensor3D, прикреплённых к корпусу и башне танка, обнаруживает цель посредством бросания лучей. Также БР с помощью компонента CameraSencorComponent получает информацию от находящейся на месте наводчика камеры.

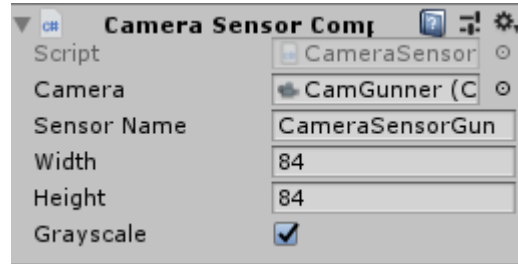


Рис. 35. Компонент CameraSencorComponent

```
public override void CollectObservations()
{
    AddVectorObs(transform.localPosition); //3
    AddVectorObs(transform.forward); //3
    AddVectorObs(gunEnd.position); //3
    AddVectorObs(gunEnd.forward); //3
    if (canShoot == true) //1
        AddVectorObs(1.0f);
    else
        AddVectorObs(0f);
}
```

Если снаряд поразил цель или танк столкнулся с ней (уничтожил тараном), то агент получает в качестве награды 1 и 0.5 за эпизод соответственно. Если же БР не смог поразить или достигнуть цель за эпизод, то он получает награду -1.

```
private void OnShellCollided(GameObject gameObject)
{
    if (gameObject.tag == "target")
    {
        SetReward(1f);
        Done();
    }
}
private void OnCollisionEnter(Collision col)
{
    if (col.gameObject.CompareTag("target"))
    {
        SetReward(0.5f);
        Done();
    }
}
```

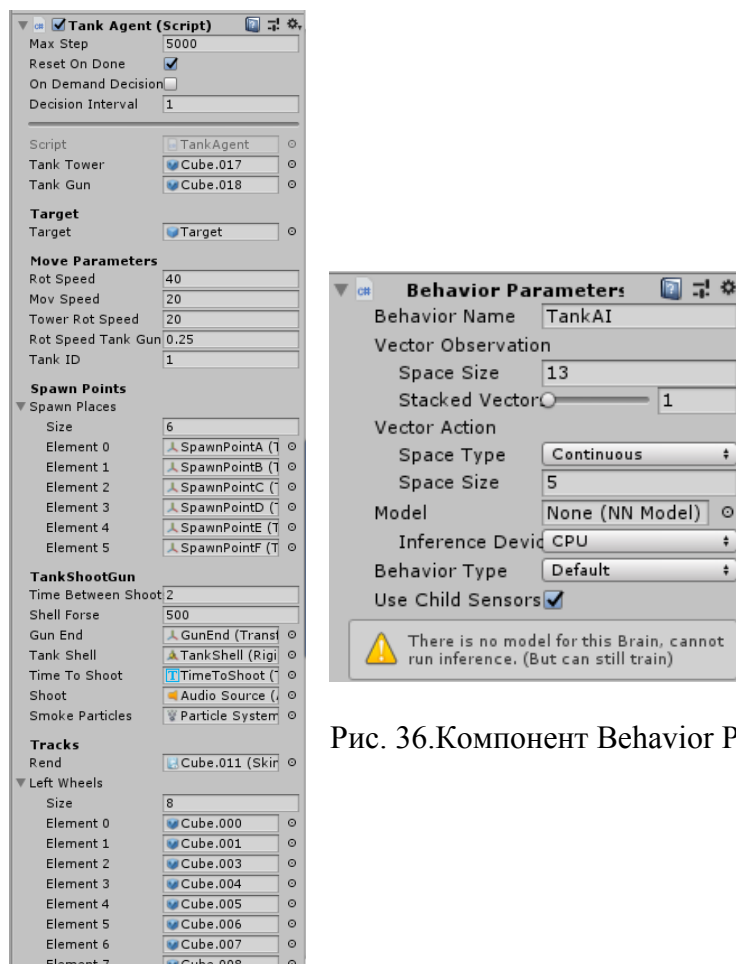


Рис. 36. Компонент Behavior Parameters.

Рис. 35. Компонент TankAgent.

### 3.3.1 Модель №1

Так как агент находится на большой по площади территории с редкими положительными наградами, то ему будет очень сложно понять, что ему нужно делать. Для выработки правильной стратегии БР может потребоваться миллион шагов, а может даже и после такого длительного процесса не обучиться. Выход – использовать не только внешние награды среды, но и внутренние, которые помогут агенту лучше исследовать среду.

На рисунке 36 показан файл `gail_config.yaml`, содержащий гиперпараметры обучения агента, в том числе и внутренние сигналы вознаграждений.

При добавлении параметра `curiosity` в `reward_signals` агент получает внутреннюю награду за любопытство – действий, приводящих к исследованию окружающей его среды и ведущих к открытию новых состояний.

```

TankAI:
    summary_freq: 1000
    time_horizon: 1024
    batch_size: 1024
    buffer_size: 10240
    hidden_units: 512
    lambda: 0.925
    num_layers: 3
    beta: 1.0e-2
    max_steps: 2.5e5
    num_epoch: 5
    normalize: false
    behavioral_cloning:
        demo_path: ../demos/TankAIgail.demo
        strength: 0.5
        steps: 25000
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
        curiosity:
            strength: 0.02
            gamma: 0.99
            encoding_size: 256
        gail:
            strength: 0.01
            gamma: 0.99
            encoding_size: 128
            use_actions: true
            demo_path: ../demos/TankAIgail.demo

```

Рис. 37. Файл с гиперпараметрами, отвечающими за внутренние награды.

Также может оказаться проще продемонстрировать агенту поведение, чем выработать его через метод проб и ошибок. При добавлении параметров `gail` или `behavioral_cloning` агент будет пытаться действовать так, как это делал эксперт в той же ситуации. То есть агент будет копировать поведение эксперта. Разница между этими двумя параметрами в том, что при помощи `gail` агент действует похоже на набор демонстраций и хорошо работает при ограниченном их количестве; а при использовании `behavioral_cloning` агент действует в точности как в демонстрации и хорошо работает при большом их количестве. Для создания демонстрации необходимо прикрепить к агенту компонент `DemonstrationRecorder`, поставить галочку напротив `Record`, ввести название файла с будущими демонстрациями, нажать кнопку `Play`, успешно завершить несколько эпизодов от лица эксперта, нажать на кнопку `Stop`.

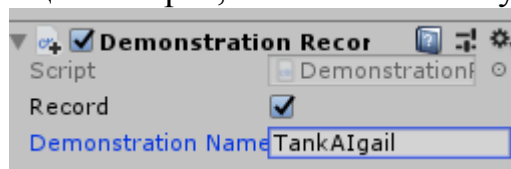


Рис. 38. Компонент `DemonstrationRecorder`.

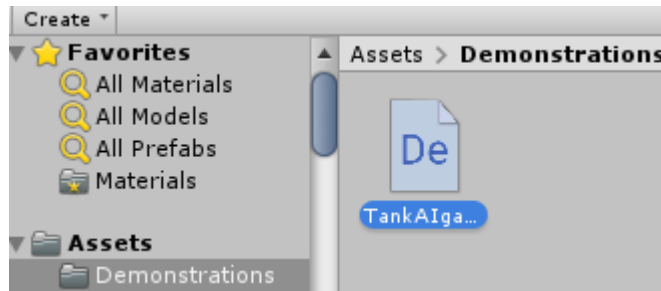


Рис. 39. Файл с демонстрациями

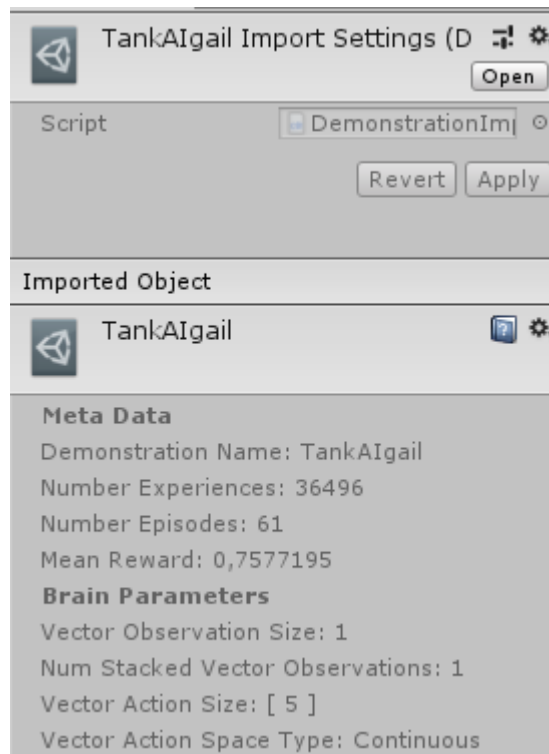


Рис. 40. Хранящаяся информация в файле с демонстрациями.

Для обучения дополнительно будем использовать curriculum learning со следующими изменяемыми параметрами:

ChPosXtarget и ChPosYtarget – величина изменения координат цели от её начального положения, SpawnPoints – кол-во начальных точек БР, TargetDistance – расстояние между танком и целью для успешного завершения эпизода.

```
{
  "measure" : "reward",
  "thresholds" : [-0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.97],
  "min_lesson_length" : 25,
  "signal_smoothing" : true,
  "parameters" :
  {
    "ChPosXtarget" : [0, 0.5, 1.0, 3.0, 5.0, 7.0, 10, 15, 17, 20, 25, 30, 35, 40, 45, 50, 55, 60, 70, 80, 90, 100],
    "ChPosYtarget" : [0, 0.5, 1.0, 3.0, 5.0, 7.0, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 130, 150, 180, 210, 250],
    "SpawnPoints" : [1, 1, 1, 1, 2, 2, 2, 2, 3, 2, 3, 2, 3, 2, 4, 2, 4, 2, 5, 2, 6, 6],
    "TargetDistance" : [45, 45, 45, 40, 40, 30, 30, 25, 25, 20, 15, 15, 15, 13.5, 10, 7.5, 5, 2.5, 0, 0, 0, 0]
  }
}
```

Рис. 41. Файл с изменяющимися параметрами Curriculum learning

```
G:\UnityAI\config>mlagents-learn gail_config.yaml --curriculum curricula/TankAI
--run-id=TrnkPPOCrriosCamCurGail --train
```

## Графики:

1. Curiosity Forward Loss – Среднее значение функции ошибки прямой модели curiosity. Показывает насколько модель хорошо предсказывает новое наблюдение.
2. Curiosity Inverse Loss – Среднее значение функции ошибки обратной модели curiosity. Показывает насколько модель хорошо предсказывает действие, предпринимаемое между двумя наблюдениями.
3. GAIL Loss – Среднее значение функции потерь дискриминатора GAIL. Показывает насколько хорошо модель имитирует демонстрационные данные.
4. Pretraining Loss – Среднее значение функции потерь поведенческого копирования. Показывает насколько хорошо модель имитирует демонстрационные данные.
5. Curiosity Reward – Показывает среднее кумулятивное внутреннее вознаграждение, генерируемое за эпизод.
6. Curiosity Value Estimate – Значение функции состояния для награды любопытства.
7. Extrinsic Value Estimate – Средняя ожидаемая внешняя награда.
8. GAIL Expert Estimate – Оценка дискриминатора для состояний и действий из демонстраций экспертов.
9. GAIL Policy Estimate – Оценка дискриминатора для состояний и действий агента.
10. GAIL Reward – Среднее кумулятивное вознаграждение за эпизод на основе дискриминатора.
11. GAIL Value Estimate – Средняя ожидаемая награда GAIL.

## Environment

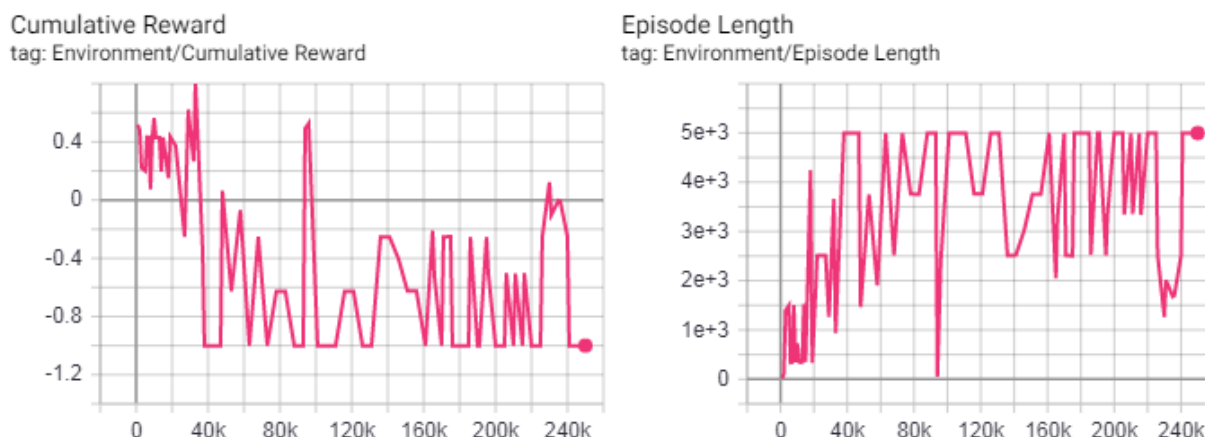


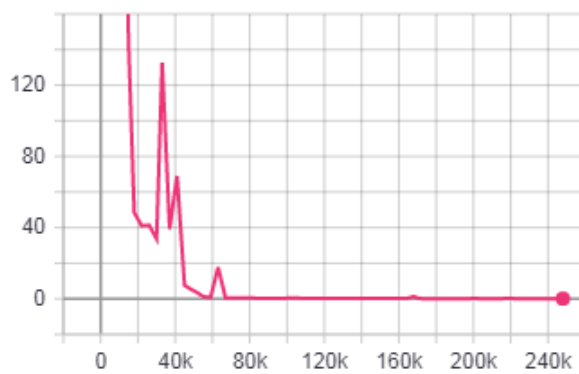
Рис. 42. Графики Cumulative Reward и Episode Length всех моделей.



Рис. 43. График Lesson.

## Losses

Curiosity Forward Loss  
tag: Losses/Curiosity Forward Loss



Curiosity Inverse Loss  
tag: Losses/Curiosity Inverse Loss

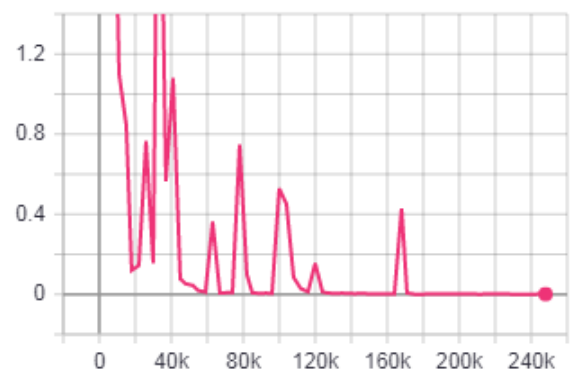
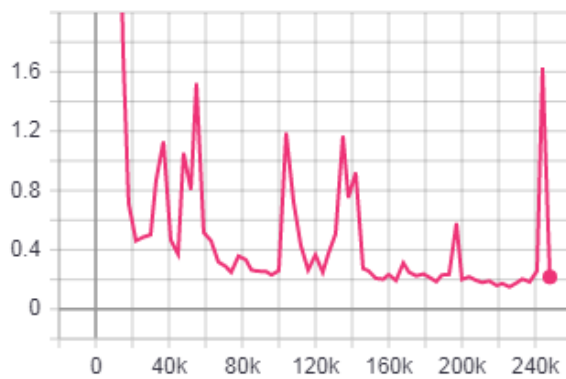


Рис. 44. Графики Curiosity Forward Loss и Curiosity Inverse Loss.

GAIL Loss  
tag: Losses/GAIL Loss



Policy Loss  
tag: Losses/Policy Loss

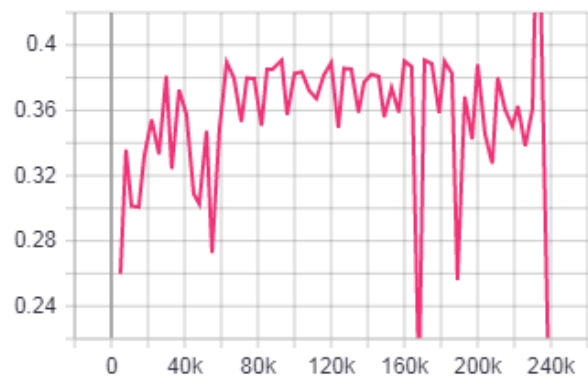
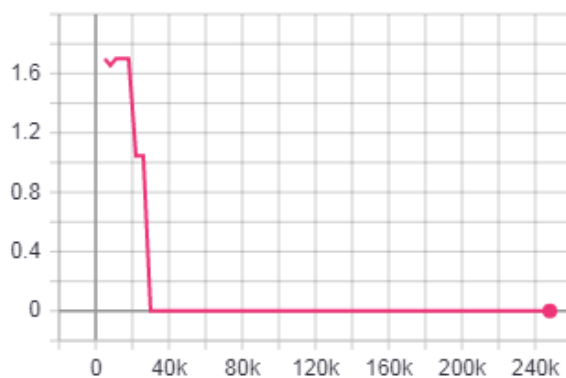


Рис. 45. Графики GAIL Loss и Policy Loss.

Pretraining Loss  
tag: Losses/Pretraining Loss



Value Loss  
tag: Losses/Value Loss

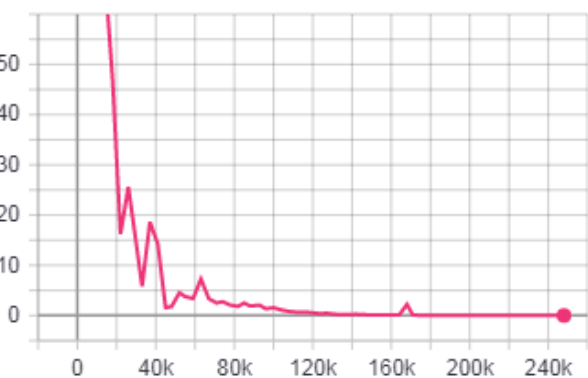
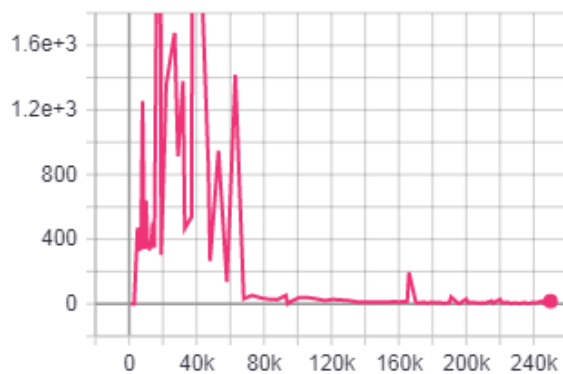


Рис. 46. Графики Pretraining Loss и Value Loss.

## Policy

Curiosity Reward  
tag: Policy/Curiosity Reward



Curiosity Value Estimate  
tag: Policy/Curiosity Value Estimate

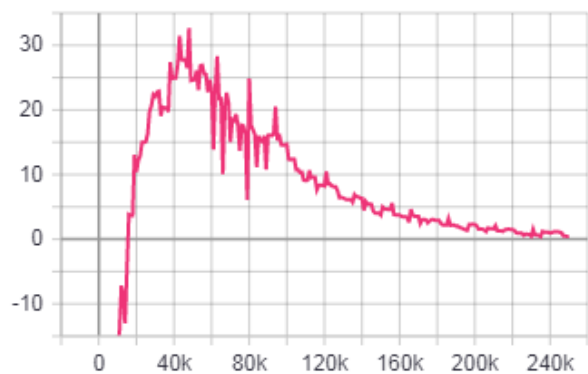
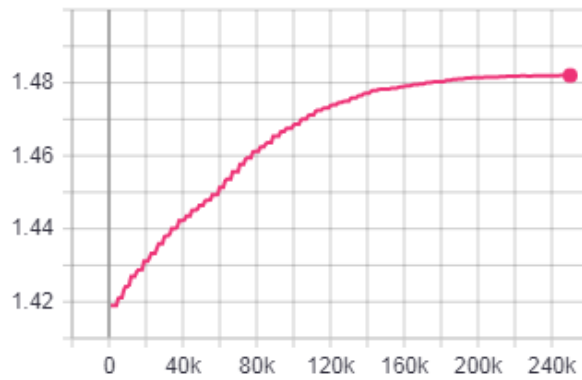


Рис. 47. Графики Curiosity Reward и Curiosity Value Estimate.

Entropy  
tag: Policy/Entropy



Extrinsic Reward  
tag: Policy/Extrinsic Reward

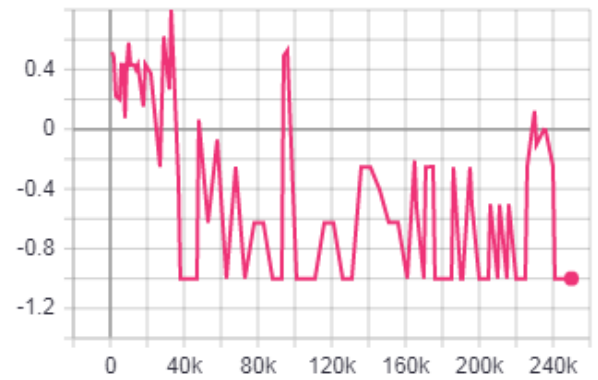
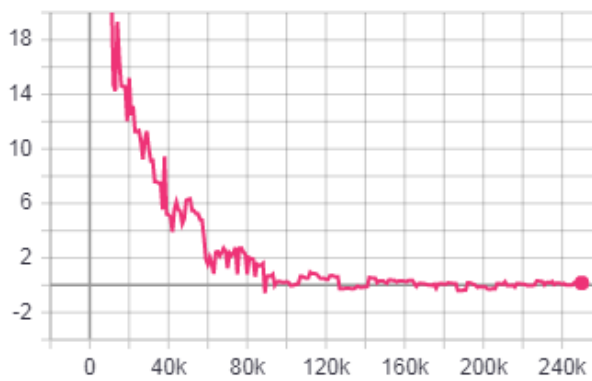


Рис. 48. Графики Entropy и Extrinsic Reward.

Extrinsic Value Estimate  
tag: Policy/Extrinsic Value Estimate



GAIL Expert Estimate  
tag: Policy/GAIL Expert Estimate

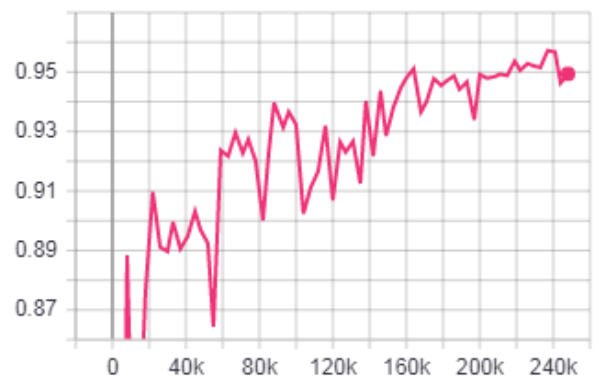
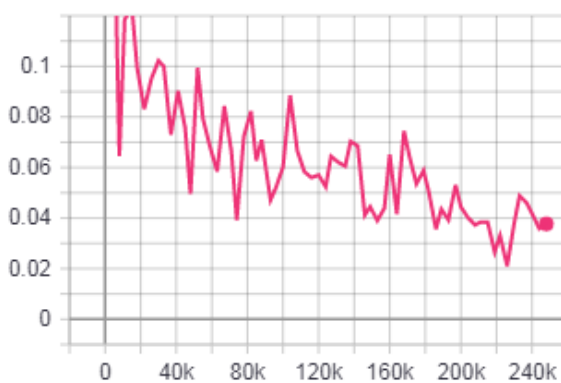


Рис. 49. Графики Extrinsic Value Estimate и GAIL Expert Estimate.

GAIL Policy Estimate  
tag: Policy/GAIL Policy Estimate



GAIL Reward  
tag: Policy/GAIL Reward

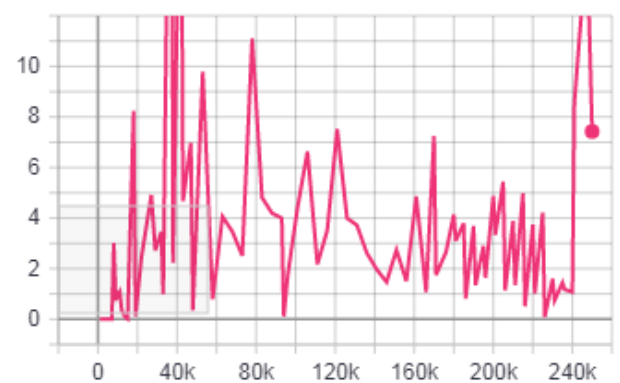


Рис. 50. Графики GAIL Policy Estimate и GAIL Reward.



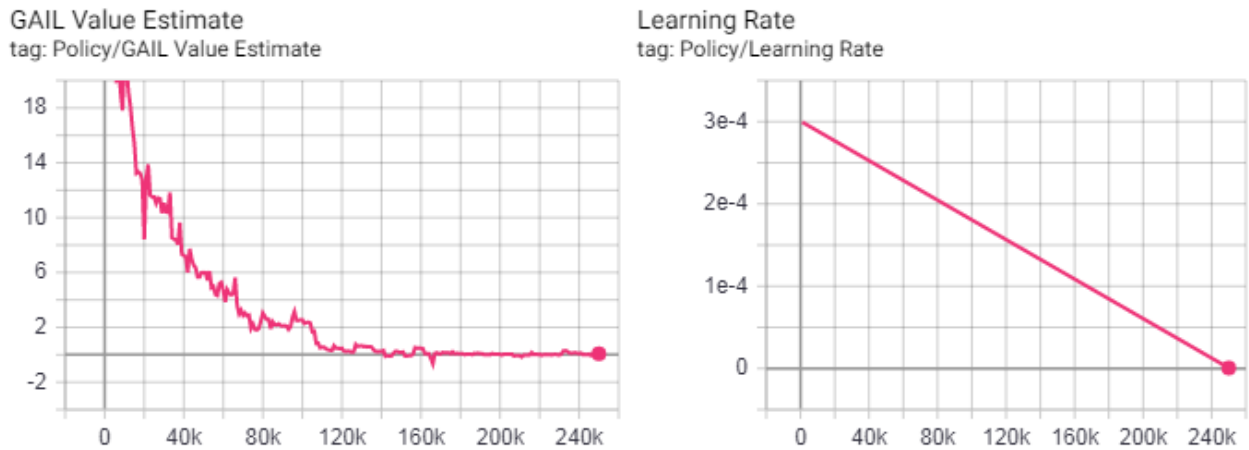


Рис. 51. Графики GAIL Value Estimate и Learning Rate.

### 3.3.2 Модель №2

Добавим второй танк в качестве противника. Первый танк будет иметь тег – TankTeamA, второй – TankTeamB. Так как эти БР идентичны, то они объединяются в одну группу для обучения одной НС. Добавим к наблюдениям сторону, за которую выступает БР:

```
if(this.gameObject.tag == "TankTeamA")
    AddVectorObs(0.0f);
else
    AddVectorObs(1.0f);
```

и их теги в RayPerceptioSensorComponent3D.

Если агент поразил противника снарядом, то ему присваивается награда 1; если его поразили -1; если танки столкнулись, то каждому -1. Также на каждом шаге агент получает награду  $\frac{-1}{Max Step}$ .

```
private void OnShellCollided(GameObject gameObject)
{
    if (gameObject.tag == "TankTeamA" && this.gameObject.tag != "TankTeamA"
        || gameObject.tag == "TankTeamB" && this.gameObject.tag != "TankTeamB")
    {
        SetReward(1f);
        Done();
    }
}

private void OnCollisionEnter(Collision col)
{
    if (col.gameObject.tag == "TankShell")
    {

```

```

        SetReward(-1f);
        Done();
    }

    if (col.gameObject.tag == "TankTeamA" || col.gameObject.tag == "TankTeamB")
    {
        SetReward(-1f);
        Done();
    }
}

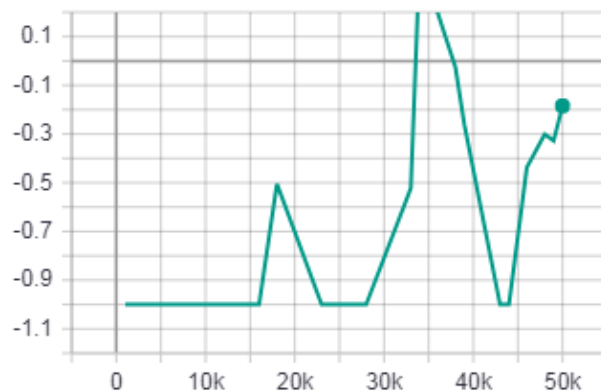
```

Проведём 50 000 шагов обучения с внешними(extrinsic) и внутренними для исследования(curiosity) наградами.



Рис. 52.Танковая дуэль.

Cumulative Reward  
tag: Environment/Cumulative Reward



Episode Length  
tag: Environment/Episode Length

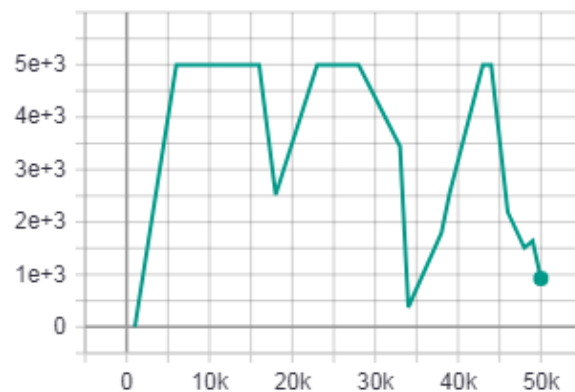
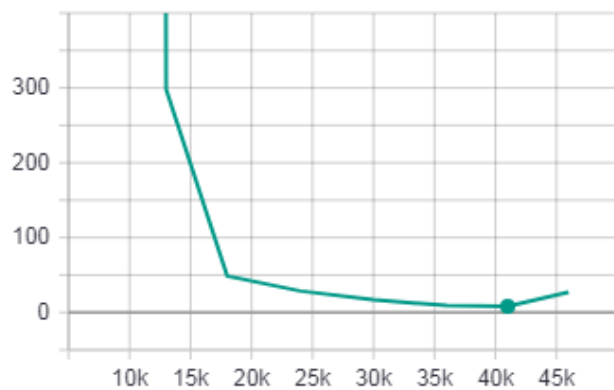


Рис. 53. Графики Cumulative Reward и Episode Length.

Curiosity Forward Loss  
tag: Losses/Curiosity Forward Loss



Curiosity Inverse Loss  
tag: Losses/Curiosity Inverse Loss

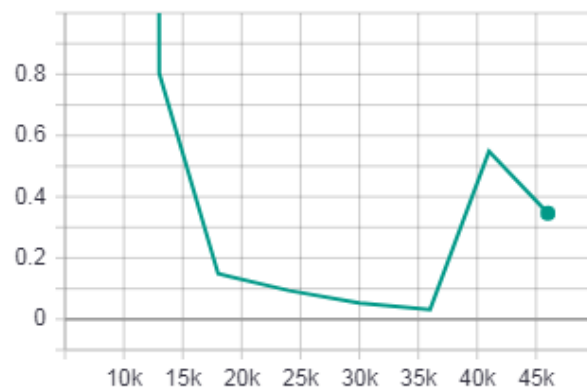
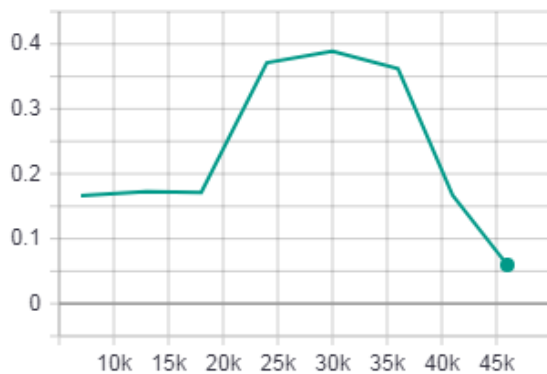


Рис. 54. Графики Curiosity Forward Loss и Curiosity Inverse Loss

Policy Loss  
tag: Losses/Policy Loss



Value Loss  
tag: Losses/Value Loss

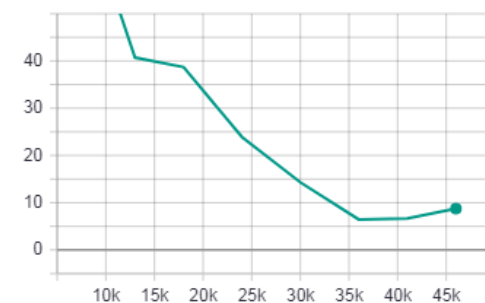
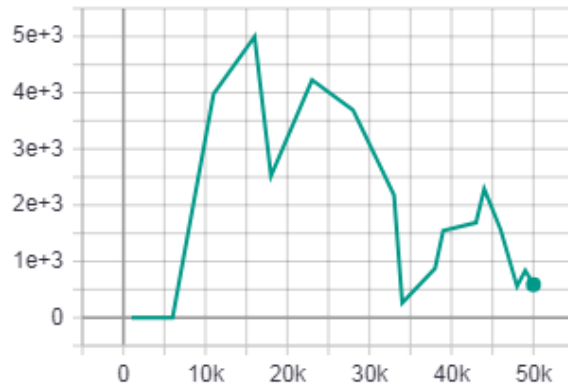


Рис. 55. Графики Policy Loss и Value Loss

Curiosity Reward  
tag: Policy/Curiosity Reward



Curiosity Value Estimate  
tag: Policy/Curiosity Value Estimate

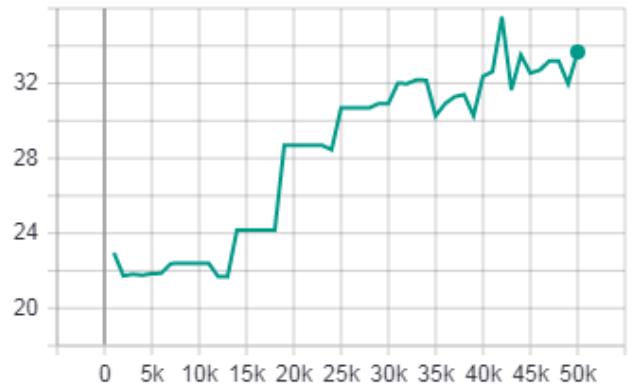
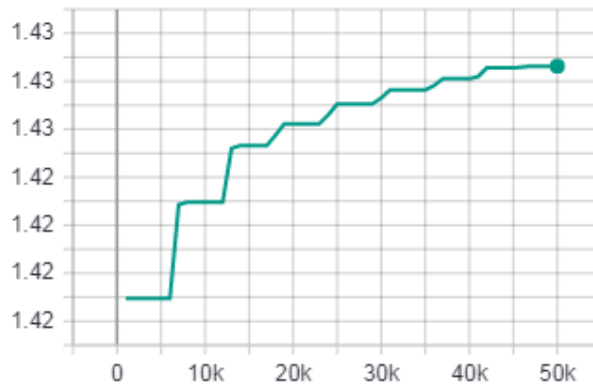


Рис. 56. Графики Curiosity Reward и Curiosity Value Estimate

Entropy  
tag: Policy/Entropy



Extrinsic Reward  
tag: Policy/Extrinsic Reward

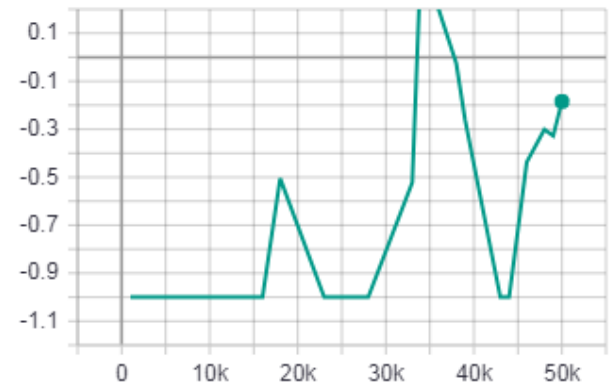
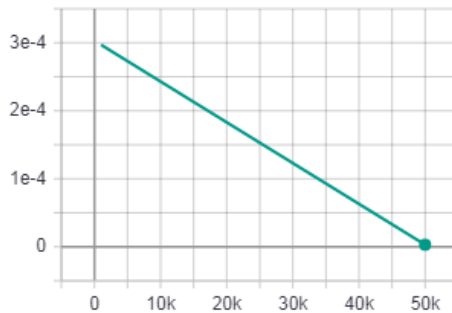


Рис. 57. Графики Entropy и Extrinsic Reward

Learning Rate  
tag: Policy/Learning Rate



Extrinsic Value Estimate  
tag: Policy/Extrinsic Value Estimate

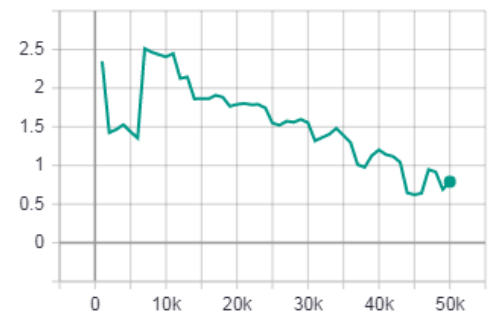


Рис. 58. Графики Learning Rate и Extrinsic Value Estimate

### 3.4 Выводы

Проанализировав полученные модели, можно сделать следующие выводы:

1. Увеличение количества агентов улучшает и ускоряет обучение. Так как каждый агент независимо от других агентов исследует окружающее его пространство, увеличивается общее количество собираемых данных и тем самым достигается стабильность обучения. Ускорение обучения возможно при наличии достаточных мощностей для параллельного обучения агентов.
2. Добавление нужных наблюдений улучшает поведение агента.
3. Для решения проблем целеполагания и переобучения можно использовать постепенное изменение нужных параметров в правильном направлении. В этом поможет curriculum learning.
4. Для мотивирования агента быстро выполнять задачи следует добавлять на каждом шаге небольшую отрицательную награду.
5. Для более стабильного обучения желательно, чтобы все наблюдения агенты были нормализованы, а награды лежали в  $[-1;1]$ .
6. Успешность обучения зависит от точной настройки большого количества гиперпараметров. Подобрать гиперпараметры можно опираясь на статистику обучения, посмотрев её с помощью Tensorboard.
7. Для успешного решения задач с большим количеством состояний и действий агента может потребоваться большое количество (сотни тысяч, миллионы) шагов обучения, что приводит к необходимости использования больших вычислительных мощностей и времени. Однако, можно легко сгенерировать огромное количество данных для обучения.
8. Для решения проблемы применения-исследования (exploitation vs. exploration problem) можно использовать внутренние награды для исследования среды (curiosity reward).
9. Если нужно, чтобы агент действовал точно так же как и эксперт в той же ситуации, то можно показать агенту как выполнять задачу, применив методы обучения на основе демонстраций: GAIL и Behavioral Cloning.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения выпускной работы с помощью обучения с подкреплением разработаны модели автономных боевых роботов.

Модель БР№1 может быть применена для быстрого выполнения задач снабжения, транспортировки грузов и перемещения на заданной территории.

Модель БР№2 может быть применена для нахождения и уничтожения угроз.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1) Робот-сапёр «Уран-6» [Электронный ресурс]. – Режим доступа: <https://topwar.ru/62494-robot-saper-uran-6.html>. – Заглавие с экрана. – (Дата обращения: 12.06.2020).
- 2) Ненадёжный и ненаблюдательный. О недостатках боевого робота «Уран-9» [Электронный ресурс]. – Режим доступа: <https://topwar.ru/143330-v-sirii-proyavilis-nedostatki-boevogo-robota-uran-9.html>. – Заглавие с экрана. – (Дата обращения: 12.06.2020).
- 3) Уран-14, робототехнический комплекс пожаротушения [Электронный ресурс]. – Режим доступа: <https://www.arms-expo.ru/armament/samples/880/70957/>. – Заглавие с экрана. – (Дата обращения: 12.06.2020).
- 4) Платформа-М [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Платформа-М>. – Заглавие с экрана. – (Дата обращения: 12.06.2020).
- 5) Капитан – разработка в области роботехники «ЦНИИ РТК» [Электронный ресурс]. – Режим доступа: <https://rtc.ru/solution/kapitan/>. – Заглавие с экрана. – (Дата обращения: 12.06.2020).
- 6) Ударный робот "Вихрь" повысит боевую мощь тактических подразделений ВС РФ [Электронный ресурс]. – Режим доступа: <https://tass.ru/armiya-i-opk/3610661>. – Заглавие с экрана. – (Дата обращения: 12.06.2020).
- 7) Израильский беспилотный автомобиль Guardium [Электронный ресурс]. – Режим доступа: <https://topwar.ru/57122-izrailskiy-bespilotnyy-avtomobil-guardium.html>. – Заглавие с экрана. – (Дата обращения: 12.06.2020).
- 8) Беспилотные летательные аппараты сил специального назначения ВВС США [Электронный ресурс]. – Режим доступа: <https://topwar.ru/162433-bespilotnye-letatelnye-apparaty-sil-specialnogo-naznachenija-vvs-ssha.html>. – Заглавие с экрана. – (Дата обращения: 12.06.2020).
- 9) Роботизированный комплекс «Суррогат»: малая подводная лодка для учений флота [Электронный ресурс]. – Режим доступа: <https://topwar.ru/105129-robotizirovannyy-kompleks-surrogat-malaya-podvodnaya-lodka-dlya-ucheniy-flota.html>. – Заглавие с экрана. – (Дата обращения: 12.06.2020).
- 10) Посейдон (подводный аппарат) [Электронный ресурс]. – Режим доступа: [https://ru.wikipedia.org/wiki/Посейдон\\_\(подводный\\_аппарат\)](https://ru.wikipedia.org/wiki/Посейдон_(подводный_аппарат)). – Заглавие с экрана. – (Дата обращения: 12.06.2020).

- 11) Панов А.И. Введение в методы машинного обучения с подкреплением. [текст] – Долгопрудный: Физтех-полиграф, 2019. – 52 с.
- 12) Николенко С., Кадури А., Архангельская Е. Глубокое обучение.[текст] – СПб.: Питер, 2018. – 480 с.
- 13) Д. Хокинг. Unity в действии. Мультиплатформенная разработка на C#. 2-е межд. изд. [текст] – СПб.: Питер, 2019. – 352 с.
- 14) M. Gaig. Unity Game Development in 24 hours. [текст] – 800 East 96th Street , Indianapolis , Indiana 46240 USA : Sams, 2018. – 618 с.
- 15) Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D. (2020). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627. [Электронный ресурс] – Режим доступа: <https://github.com/Unity-Technologies/ml-agents> . – Заглавие с экрана. – (Дата обращения: 12.06.2020).
- 16) Solving sparse-reward tasks with Curiosity[Электронный ресурс] – Режим доступа: <https://blogs.unity3d.com/2018/06/26/solving-sparse-reward-tasks-with-curiosity/> . – Заглавие с экрана. – (Дата обращения: 12.06.2020).



## ПРИЛОЖЕНИЕ А. Код боевого робота №1

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

public class RobotAgent : Agent
{
    public Transform target;

    [Header("Move Parameters")]
    public float rotateSpeed = 5f;
    public float walkSpeed = 5f;
    //public float runSpeed = 8f;
    public float gravity = -9.8f;

    private Academy RbAcademy;
    private float rotAmount;
    private float moveAmount;
    private RobotArea agentArea;
    private float target_distance = 1.42f;
    private Animator anim;
    private CharacterController characterController;
    private Rigidbody rb;

    public override void InitializeAgent()
    {
        characterController = GetComponent<CharacterController>();
        rb = GetComponent<Rigidbody>();
        agentArea = transform.parent.GetComponent<RobotArea>();
        RbAcademy = FindObjectOfType<RobotAcademy>();
        anim = GetComponent<Animator>();
    }

    public override void AgentReset()
    {
        if (transform.localPosition.y < -2f)
        {
            rb.velocity = Vector3.zero;
            rb.angularVelocity = Vector3.zero;
            transform.localPosition = new Vector3(0, 0, 0);
        }

        target.localPosition = new Vector3( Random.Range(-5f, 5f), 0.5f, Random.Range(-5f,
5f));
        target_distance =
RbAcademy.FloatProperties.GetPropertyWithDefault("target_distance", 1.42f);
    }

    public override void CollectObservations()
    {
        AddVectorObs(target.localPosition);
        AddVectorObs(transform.localPosition);
        AddVectorObs(transform.forward);
        AddVectorObs(Vector3.Distance(target.transform.position, transform.position));
        AddVectorObs((target.transform.position - transform.position).normalized);
    }
}
```

```

public override void AgentAction(float[] vectorAction)
{
    rotAmount = vectorAction[0] * rotateSpeed;
    moveAmount = vectorAction[1] * walkSpeed;
    if (moveAmount != 0)
        anim.SetBool("Walk_Anim", true);
    else
        anim.SetBool("Walk_Anim", false);

    Vector3 rotateVector = transform.up * rotAmount;
    Quaternion direction = Quaternion.Euler(transform.localRotation.eulerAngles +
    rotateVector * rotateSpeed);
    transform.localRotation = Quaternion.Lerp(transform.localRotation, direction,
    rotateSpeed * Time.deltaTime);

    Vector3 movement = transform.forward * moveAmount;
    movement.y = gravity;
    movement = Vector3.ClampMagnitude(movement, walkSpeed);
    movement *= Time.deltaTime;
    characterController.Move(movement);

    float distanceToTarget = Vector3.Distance(transform.localPosition,
    target.localPosition);
    if (distanceToTarget < target_distance)
    {
        SetReward(1f);
        agentArea.UpdateScore(GetCumulativeReward());
        Done();
        StartCoroutine(agentArea.SwapGroundMaterial(success: true));
    }
    else
    {
        AddReward(-1f / agentParameters.maxStep);
        agentArea.UpdateScore(GetCumulativeReward());
    }

    if (transform.localPosition.y < -2f)
    {
        SetReward(-0.05f);
        Done();
        StartCoroutine(agentArea.SwapGroundMaterial(success: false));
    }
}

public override float[] Heuristic()
{
    var action = new float[2];
    action[0] = Input.GetAxis("Horizontal");
    action[1] = Input.GetAxis("Vertical");
    return action;
}

private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "target")
    {
        SetReward(1f);
        agentArea.UpdateScore(GetCumulativeReward());
        Done();
        StartCoroutine(agentArea.SwapGroundMaterial(success: true));
    }
}

```

```

    }
}

using TMPro;
public class RobotArea : MonoBehaviour
{
    public Material successMaterial;
    public Material failureMaterial;
    public GameObject ground;
    public TextMeshPro scoreText;

    private Renderer groundRenderer;
    private Material groundMaterial;

    private void Start()
    {
        groundRenderer = ground.GetComponent<Renderer>();
        groundMaterial = groundRenderer.material;
    }

    public void UpdateScore(float score)
    {
        scoreText.text = score.ToString("0.00");
    }

    public IEnumerator SwapGroundMaterial(bool success)
    {
        if (success)
        {
            groundRenderer.material = successMaterial;
        }
        else
        {
            groundRenderer.material = failureMaterial;
        }

        yield return new WaitForSeconds(0.5f);
        groundRenderer.material = groundMaterial;
    }
}

```

## ПРИЛОЖЕНИЕ В. Код боевого робота №2

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

public class TankAgent : Agent
{
    public GameObject tankTower;
    public GameObject tankGun;
    [Header("Target")]
    public GameObject Target;

    [Header("Move Parameters")]
    public float rotSpeed = 10f;
    public float movSpeed = 10f;
    public float towerRotSpeed = 10f;
    public float rotSpeedTankGun = 0.25f;
    public int TankID = 1;
    [Header("Spawn Points")]
    public Transform[] spawnPlaces;

    private float maxTankGunUp = 15f;
    private float minTankGunDown = -5f;
    private float counter = 0f;
    private Rigidbody tankRb;

    [HideInInspector] private float Vertical;
    [HideInInspector] private float Horizontal;
    [HideInInspector] private float MouseX;
    [HideInInspector] private float MouseY;

    [Header("TankShootGun")]
    public float timeBetweenShoots = 2f;
    public float shellForse = 100f;
    public Transform gunEnd;
    public Rigidbody tankShell;
    public TextMeshProUGUI TimeToShoot;
    public AudioSource shoot;
    public ParticleSystem smokeParticles;

    private bool canShoot;
    private float shootTimer;
    Rigidbody tankShellRb;

    [Header("Tracks")]
    public Renderer rend;
    public GameObject[] LeftWheels;
    public GameObject[] RightWheels;

    float leftRotSpeed = 0;
    float rightRotSpeed = 0;

    Vector3 startTargetPos;
    Vector3 startTankPosition;
    Quaternion startTankRotation;
    Vector3 startTankTowerPosition;
    Quaternion startTankTowerRotation;
    Vector3 startTankGunPosition;
    Quaternion startTankGunRotation;
```

```

Rigidbody targetRb;

private int shellIDcollided;
private Academy TankAIAcademy;//for CurriculumLearning
private float ChPosXtarget = 0.5f;
private float ChPosZtarget = 0.5f;
private float targetDistance = 5f;
private int spawnPoints = 6;
private bool succes = false;
private void Start()
{
    tankRb = GetComponent<Rigidbody>();
    canShoot = true;
    TimeToShoot.color = Color.green;
    startTargetPos = Target.transform.localPosition;
    startTankPosition = transform.localPosition;
    startTankRotation = transform.localRotation;
    targetRb = GetComponent<Rigidbody>();
    startTankTowerPosition = tankTower.transform.position;
    startTankTowerRotation = tankTower.transform.rotation;
    startTankGunPosition = tankGun.transform.position;
    startTankGunRotation = tankGun.transform.rotation;

    Vertical = 0;
    Horizontal = 0;
    MouseX = 0;
    MouseY = 0;

    TankAIAcademy = FindObjectOfType<TankAcademy>();
    spawnPoints =
System.Convert.ToInt32(TankAIAcademy.FloatProperties.GetPropertyWithDefault("SpawnPoints",
1f));
    targetDistance =
TankAIAcademy.FloatProperties.GetPropertyWithDefault("TargetDistance", 5f);

}
private void Awake()
{
    Messenger<GameObject>.AddListener("ShellCollidedWith_", OnShellCollided);
}

private void OnDestroy()
{
    Messenger<GameObject>.RemoveListener("ShellCollidedWith_", OnShellCollided);
}
public override void AgentReset()
{
    ChPosXtarget = TankAIAcademy.FloatProperties.GetPropertyWithDefault("ChPosXtarget",
100f);
    ChPosZtarget = TankAIAcademy.FloatProperties.GetPropertyWithDefault("ChPosYtarget",
250f);
    spawnPoints =
System.Convert.ToInt32(TankAIAcademy.FloatProperties.GetPropertyWithDefault("SpawnPoints",
1f));
    targetDistance =
TankAIAcademy.FloatProperties.GetPropertyWithDefault("TargetDistance", 25f);

    float tposX = Random.Range(startTargetPos.x - ChPosXtarget, startTargetPos.x +
ChPosXtarget);
    float tposZ = Random.Range(startTargetPos.z - ChPosZtarget, startTargetPos.z +
ChPosZtarget);
    Target.transform.localPosition =new Vector3( tposX, startTargetPos.y , tposZ);

```

```

targetRb.velocity = Vector3.zero;
targetRb.angularVelocity = Vector3.zero;

if (!succes)
{
    tankRb.velocity = Vector3.zero;
    tankRb.angularVelocity = Vector3.zero;
    transform.localPosition = startTankPosition;
    transform.localRotation = startTankRotation;
    tankTower.transform.position = startTankTowerPosition;
    tankTower.transform.rotation = startTankRotation;
    tankGun.transform.position = startTankGunPosition;
    tankGun.transform.rotation = startTankGunRotation;
    spawnPoints =
System.Convert.ToInt32(TankAIAcademy.FloatProperties.GetPropertyWithDefault("SpawnPoints",
1f));
    int k = Random.Range(0, spawnPoints);
    transform.localPosition = spawnPlaces[k].localPosition;
    transform.localRotation = spawnPlaces[k].localRotation;
}
succes = false;
}

public override void CollectObservations()
{
    AddVectorObs(transform.localPosition);
    AddVectorObs(transform.forward);
    AddVectorObs(gunEnd.position);
    AddVectorObs(gunEnd.forward);
    if (canShoot == true)
        AddVectorObs(1.0f);
    else
        AddVectorObs(0f);
}

public override void AgentAction(float[] vectorAction)
{
    tankMovement(vectorAction);
    if (vectorAction[4]==1f && canShoot)
    {
        StartCoroutine("GunShoot");
    }

    Tracks(vectorAction);
    AddReward(-1f / agentParameters.maxStep);

    float distanceToTarget = Vector3.Distance(transform.localPosition,
Target.transform.localPosition);
    if ( distanceToTarget < targetDistance)
    {
        SetReward(0.5f);
        Done();
    }

    float rotX = Mathf.Abs(transform.localRotation.x) % 360;
    float rotY = Mathf.Abs(transform.localRotation.y) % 360;
    float rotZ = Mathf.Abs(transform.localRotation.z) % 360;
    if(transform.localPosition.y <-5f || rotX>55 || rotZ>55)
    {
        SetReward(-0.1f);
        succes = false;
        Done();
    }
}

```

```

private void tankMovement(float[] vectorAction)
{
    Horizontal = vectorAction[0];
    Vertical = vectorAction[1];
    MouseX = vectorAction[2];
    MouseY = vectorAction[3];

    Vector3 movement = transform.forward * movSpeed * vectorAction[1] *
Time.deltaTime;
    tankRb.MovePosition(tankRb.position + movement);

    float turn = rotSpeed * vectorAction[0] * Time.deltaTime;
    Quaternion turnRotation = Quaternion.Euler(0f, turn, 0f);
    tankRb.MoveRotation(tankRb.rotation * turnRotation);

    tankTower.transform.Rotate(0, vectorAction[2]* Time.deltaTime * towerRotSpeed, 0);

    float rotGun = rotSpeedTankGun * vectorAction[3];
    if (counter < maxTankGunUp && rotGun > 0)
    {
        tankGun.transform.Rotate(new Vector3(-rotGun, 0, 0));
        counter += rotGun;
    }
    if (counter > maxTankGunUp)
        counter = maxTankGunUp;

    if (counter > minTankGunDown && rotGun < 0)
    {
        tankGun.transform.Rotate(new Vector3(-rotGun, 0, 0));
        counter += rotGun;
    }
    if (counter < minTankGunDown)
        counter = minTankGunDown;
}

IEnumerator GunShoot()
{
    canShoot = false;
    TimeToShoot.color = Color.red;
    shoot.Play();
    ParticleSystem PS = Instantiate(smokeParticles, gunEnd.position,
Quaternion.identity) as ParticleSystem;
    PS.Play();

    tankShellRb = Instantiate(tankShell, gunEnd.position, gunEnd.rotation) as
Rigidbody;

    tankShellRb.velocity = shellForse * gunEnd.forward;
    shootTimer = timeBetweenShoots;
    while (shootTimer > 0)
    {
        TimeToShoot.text = shootTimer.ToString("00.00");
        yield return new WaitForSeconds(1f);
        shootTimer--;
    }
    TimeToShoot.text = shootTimer.ToString("00.00");
    canShoot = true;
    TimeToShoot.color = Color.green;
}

private void Tracks(float[] vectorAction)
{
    Horizontal = vectorAction[0];
    Vertical = vectorAction[1];
    if (Vertical > 0)

```

```

{
    leftRotSpeed = movSpeed * 10;
    rightRotSpeed = movSpeed * 10;
    rend.materials[0].mainTextureOffset = new Vector2(0, Time.time);
}
else if (Vertical < 0)
{
    leftRotSpeed = -movSpeed * 10;
    rightRotSpeed = -movSpeed * 10;
    rend.materials[0].mainTextureOffset = new Vector2(0, -Time.time);
}

if (Horizontal > 0)
{
    leftRotSpeed = movSpeed * 10;
    rightRotSpeed = -movSpeed * 10;
    rend.materials[0].mainTextureOffset = new Vector2(0, Time.time);
}
else if (Horizontal < 0)
{
    leftRotSpeed = -movSpeed * 10;
    rightRotSpeed = movSpeed * 10;
    rend.materials[0].mainTextureOffset = new Vector2(0, -Time.time);
}

if (Horizontal == 0 && Vertical == 0)
{
    leftRotSpeed = 0;
    rightRotSpeed = 0;
    rend.materials[0].mainTextureOffset = new Vector2(0, 0);
}

for (int i = 0; i < LeftWheels.Length; i++)
    LeftWheels[i].transform.Rotate(new Vector3(1, 0, 0) * leftRotSpeed *
Time.deltaTime);
for (int i = 0; i < RightWheels.Length; i++)
    RightWheels[i].transform.Rotate(new Vector3(1, 0, 0) * rightRotSpeed *
Time.deltaTime);
}

private void OnShellCollided(GameObject gameObject)
{
    if (gameObject.tag == "target")
    {
        SetReward(1f);
        Debug.Log("Shell collided with target!Success!");
        succes = true;
        Done();
    }
}

public override float[] Heuristic()
{
    var action = new float[5];
    action[0] = Input.GetAxis("Horizontal");
    action[1] = Input.GetAxis("Vertical");
    action[2] = Input.GetAxis("Mouse X");
    action[3] = Input.GetAxis("Mouse Y");
    if (Input.GetMouseButtonDown(0))
    {
        action[4] = 1f;
    }
    else
    {
        action[4] = 0f;
    }
}

```



```

    }

    return action;

}

private void OnCollisionEnter(Collision col)
{
    if (col.gameObject.CompareTag("target"))
    {
        SetReward(0.5f);
        succes = true;
        Done();
    }
}

}

public class Shell : MonoBehaviour
{
    [HideInInspector] public int ShellID;
    void Start()
    {
        Destroy(gameObject, 5f);
    }

    private void OnCollisionEnter(Collision coll)
    {
        Messenger<GameObject>.Broadcast("ShellCollidedWith_" , coll.gameObject );
        Debug.Log("Shell collided with " + coll.gameObject.tag);
        Destroy(gameObject);
    }

}

    if(this.gameObject.tag == "TankTeamA")
        AddVectorObs(0.0f);
    else
        AddVectorObs(1.0f);

private void OnCollisionEnter(Collision col)
{
    if (col.gameObject.tag == "TankShell")
    {
        SetReward(-1f);
        Done();
    }

    if (col.gameObject.tag == "TankTeamA" || col.gameObject.tag == "TankTeamB")
    {
        SetReward(-1f);
        Done();
    }
}

private void OnShellCollided(GameObject gameObject)
{
    if (gameObject.tag == "TankTeamA" && this.gameObject.tag != "TankTeamA"
        || gameObject.tag == "TankTeamB" && this.gameObject.tag != "TankTeamB")

```

```

        {
            SetReward(1f);
            Done();
        }
    }

public class CameraManeger : MonoBehaviour
{
    public GameObject[] cameras;
    public GameObject GunnerAim;
    public GameObject TimeToShoot;

    private int counter = 0;
    // Start is called before the first frame update
    void Start()
    {
        TurnOffAllCameras();
        cameras[0].SetActive(true);
        GunnerAim.SetActive(false);
        TimeToShoot.SetActive(false);

    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Q))
        {
            ChangeCamera();
        }

    }

    private void ChangeCamera()
    {
        TurnOffAllCameras();
        counter++;

        if (counter > cameras.Length-1)
            counter = 0;

        cameras[counter].SetActive(true);
        if(cameras[counter].name == "CamGunner")
        {
            GunnerAim.SetActive(true);
            TimeToShoot.SetActive(true);
        }
        else
        {
            GunnerAim.SetActive(false);
            TimeToShoot.SetActive(false);
        }

    }

    private void TurnOffAllCameras()
    {
        foreach (GameObject camera in cameras)
            camera.SetActive(false);

    }
}

```