**Федеральное государственное автономное образовательное**

**учреждение высшего образования**

**«Национальный исследовательский университет**

**«Высшая школа экономики»**

**Факультет компьютерных наук**

**Основная образовательная программа**

**Прикладная математика и информатика**

**КУРСОВАЯ РАБОТА**

**на тему**

**Реализация обучаемого игрового агента на движке UNITY**

**Выполнил студент группы 163, 3 курса,**

**Логинов Михаил Алексеевич**

**Научный руководитель:**

**Шестаков Андрей Владимирович,**

**департамент больших данных и искусственного интеллекта**

**Куратор:**

**к.ф-м.н, доцент, Конушин Антон Сергеевич**

**Москва 2019**

**Оглавление**

[**Аннотация** 3](#_Toc9885657)

[**Введение** 4](#_Toc9885658)

[**Обзор литературы** 6](#_Toc9885659)

[**Создание игровой среды.** 8](#_Toc9885660)

[**Первая игровая среда** 8](#_Toc9885661)

[**Вторая игровая среда** 9](#_Toc9885662)

[**Третья игровая среда** 9](#_Toc9885663)

[**Реализация обучаемого игрового агента** 11](#_Toc9885664)

[**Обучение и подбор гиперпараметров.** 13](#_Toc9885665)

[**Заключение** 14](#_Toc9885666)

[**Источники и ссылки** 15](#_Toc9885667)

# **Аннотация**

В данной курсовой работе решается задача обучения интеллектуального агента с помощью методов reinforcement learning. Для решения этой задачи, в среде разработки Unity были созданы различные варианты уровней игры “The World’s Hardest Game”, а также изучен инструментарий библиотеки ML-Agents.

**Annotation**

In this course work I recreate the first level of the game “The World’s Hardest Game” using the Unity 3d computer-aided development environment, and then is used as a dynamic environment for the gaming agent. Then use the Proximal Policy Optimization (PPO) method to train the game agent in this game environment.

This course project addresses the task of AI agent implementation using reinforcement learning techniques. Unity 3d computer-aided development environment and ML-Agents toolkit were used in order to test agent’s capabilities in game scenarios of “The World’s Hardest Game”.

**Ключевые слова**

PPO, Unity3d, Ml-Agents toolkit, Learning brain, обучение с подкреплением, игровой агент

# **Введение**

В наше время машинное обучение стало очень популярным трендом в сфере it-технологий, которое применяется в широком спектре задач, начиная бытовыми и заканчивая прикладными. Создание ИИ – искусственного интеллекта, одна из задач, решаемых с помощью алгоритмов машинного обучения. ИИ уже сегодня широко распространен в разных сферах, таких как: Медицина, Промышленность и сельское хозяйство, бытовые задачи (например, умный дом), робототехника и компьютерные игры.

Отдельно хочется выделить компьютерные игры. Они являются отличным испытательным полигоном для решения задач алгоритмами машинного обучения, ведь, по сути, можно смоделировать множество проблем из реальной жизни, решить их внутри определенного игрового сеттинга, а затем перенести это решение в реальную жизнь. Собственно, такие компании-мастодонты, как Google, Microsoft, Unity и ряд других, тестируют и улучшают свои модели ИИ в компьютерных играх. Организовываются контесты с большими призовыми фондами на написание лучшего алгоритма. Так же проводятся соревнования с реальными людьми в режиме реального времени, которые должны противостоять уже обученным на миллионах партиях игровым агентам. И все чаще можно услышать новости о том, что алгоритм берет верх над человеком.

В этой курсовой работе речь пойдет о таком методе, как обучение с подкреплением (reinforcement learning), который будет применяться для прохождения одного из уровня достаточно популярной браузерной игры под названием “The World’s Hardest Game”. Это одно из разновидностей обучения, которое относится к обучению без учителя, т. к. в таких задачах изначально не дается обучающая выборка с ответами, на основе которой можно сделать некоторую гипотезу и построить готовую модель. Агенту приходится накапливать знания из того, что обычно его штрафуют за неправильные действия и поощряют за правильные, в виде уменьшения и увеличения выигрыша соответственно. Его задача, естественно, максимизировать выигрыш.

В данной работе перед нами стоит задача написание ИИ для игрового агента, который находится в определенной динамической среде, о которой ему ничего не известно. Ему нужно, минуя препятствия, попасть из точки A в точку B.

# **Обзор литературы**

Для примера возьмем следующую статью: “Teaching AI to play Flappy Bird with Unity Teaching AI to play Flappy Bird with Unity”. В данной статье описан пример решения другой задачи, но используется схожий инструментарий, который будет задействован в дальнейшем в этой работе.

Для реализации игрового уровня и интерфейса агента были использованы следующие инструменты: ML-Agents toolkit, Unity3d, UnitySDK.

Unity3d - среда, в которой можно создавать игры, и она отлично подходит под задачи обучения игровых агентов.

ML-Agents toolkit это плагин, с открытым исходным кодом, с помощью которого можно взаимодействовать с игровыми средами и обучать в них игровых агентов следующими методами: обучение с подкреплением (именно этот метод и будет в дальнейшем использован в этой курсовой работе), нейроэволюции, имитационное обучение и другие методы машинного обучения. Все эти методы реализованы с помощью Python API. Нас интересует имплементация в этот API библиотеки TensorFlow, с помощью которой идет обучение игрового агента, описанного в этой статье, и агента, который будет реализован в рамках данной курсовой работы.

Клонированный репозиторий ml-agents содержит директорию UnitySDK, где реализованы все необходимые классы, для написания агента. Так же, там содержатся порядка 16 игровых окружений, с уже готовыми TensorFlow-моделями и агентами, которые можно запустить сразу после установки всех необходимых пакетов.

Опираясь на документацию, которая прилагается к ML-agents toolkit, можно сделать вывод, что данный набор инструментов является наиболее актуальным и удобным среди прочих, чтобы заниматься обучением игровых агентов в игровой среде. Тогда можно сформулировать задачи на данную курсовую работу.

**Постановка задач**

1. Создание игровой среды с помощью Unity3d, в которой в дальнейшем будет обучаться игровой агент, и написание скрипта для управления контрагентами.
2. Написание скрипта для управления агентом, используя язык программирования C#.
3. Обучение агента, с корректно подобранными гиперпараметрами нейронной сети.

# В моей курсовой работе и в примере из статьи использовалось обучение с подкреплением. Хотелось бы формализовать постановку задачи обучения с подкреплением.

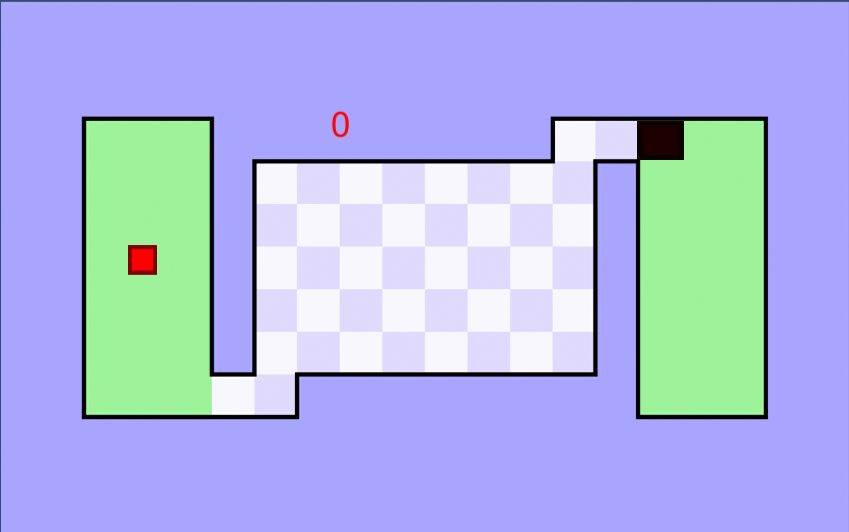
# 

# **Создание игровой среды.**

Для того, чтобы как можно лучше подобрать параметры для нейронной сети, чтобы обучение агента шло как можно лучше, создадим 3 игровые среды с помощью unity3d.

## **Первая игровая среда**

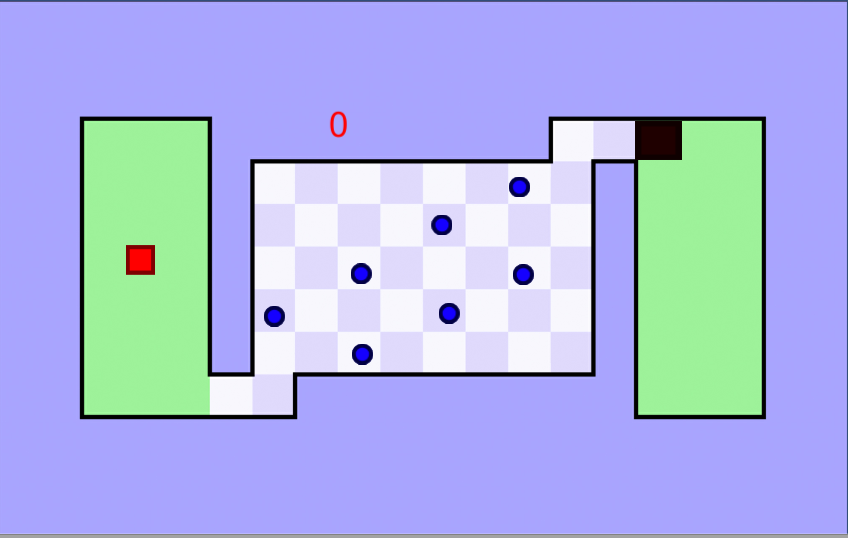
Первая игровая среда будет представлена статичной средой без контрагентов, чтобы как можно сильнее упростить задачу нашему агенту. Для этого нам нужно будет загрузить картинку из интернета с исходным уровнем, наложить стены поверх черной текстуры, чтобы в дальнейшем наш агент не выходил за пределы игровой зоны. Далее создаются сами игровые объекты с помощью функционала Unity3d, а именно: Box Collider 2d (задает прямоугольную форму объекту), RigidBody 2 (задает физику объекту). Так же, в Layer Collision Matrix указываем, что эти объекты могут наслаиваться друг-на-друга, чтобы в процессе обучения они не сталкивались и не мешались. Теперь остается указать зону “Goal Area”, которая будет выделена темным цветом, столкновение с которой считается за победу в игру. На рисунке 1 показано, как это выглядит.



**Рисунок 1. Статичная игровая среда без контрагентов.**

## **Вторая игровая среда**

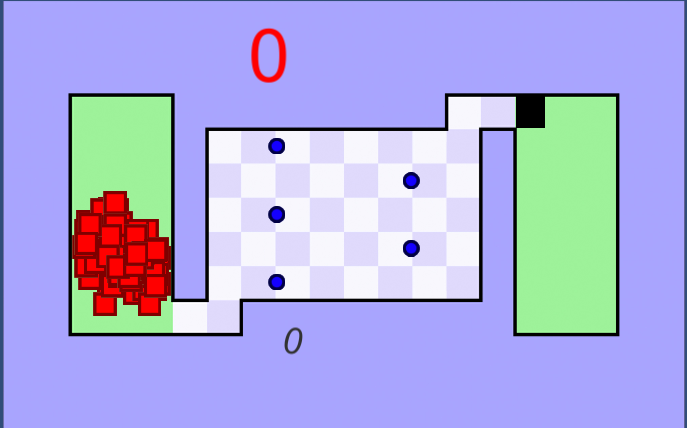
Вторая игровая среда отличается от первой лишь тем, что здесь мы добавляем статичных контрагентов, столкновение с которыми будет сильно штрафовать наших игровых агентов. Для этого опять воспользуемся функционалом Unity3d: Circle Collider 2d (задает круглую форму объекту), RigidBody 2d (задает физику объекту). На рисунке 2 показано, как это выглядит.



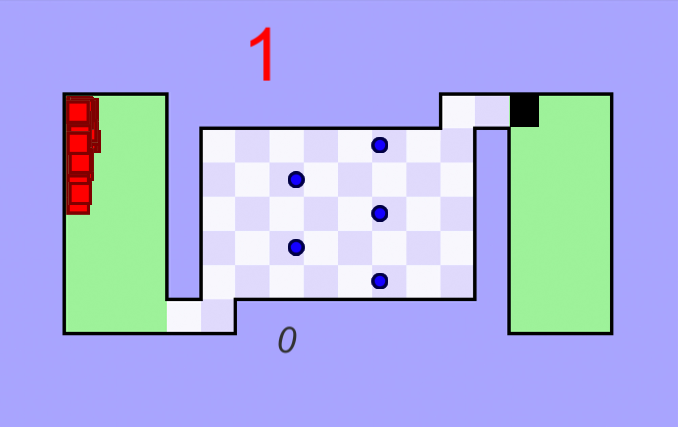
**Рисунок 2. Статичная игровая среда с контрагентами.**

## **Третья игровая среда**

Реализация третьей игровой среды будет чуть сложнее, ведь теперь её требуется из статичной сделать динамичной. Для этого, нужно будет написать скрипт на языке программирования C#, который будет контролировать перемещение контрагента по оси Ox в определенном интервале. Работает скрипт достаточно по простому сценарию: движение начинается либо из крайней левой заданной точки, либо из крайней правой. Как только достигается крайне правая или крайней левая точка соответственно, мы меняем направление на противоположное, как на рисунке 3 и 4.



**Рисунок 3. Динамичная игровая среда с контрагентами.**



**Рисунок 4. Динамичная игровая среда с контрагентами.**

# **Реализация обучаемого игрового агента**

Для реализации обучаемого агента нам предстоит

1. создать академию,
2. написать скрипт, который будет управлять агентом.

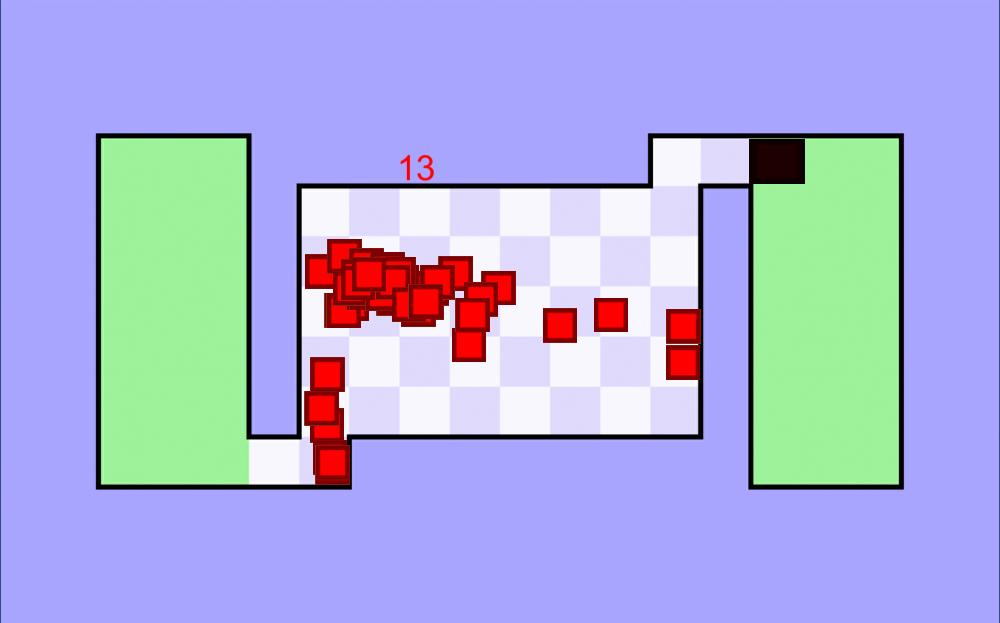
Академия нужна для управления всеми агентами и, в нашем случае, learning brain-ом. Learning brain имеет два состояния: обучающийся и обученный. Обучающийся используется в том случае, если у нас нет готовой TensorFlow-модели, и нам нужно обучить наш агент. Обученный используется в том случае, если готовая TensorFlow-модель уже есть. Тогда, нам остается ее только импортировать и запустить весь процесс. В таком случае, если модель хорошо обучена, наш агент должен выиграть с первой попытки, не сталкиваясь с врагами. В академии мы указываем конфигурацию обучения. То есть говорим, какой Learning brain нужно использовать для обучения и указываем некоторые другие параметры, например, такие Time Scale, с помощью которого, можно ускорить процесс обучения. Кстати, стоит отметить, что обучение происходит, используя ресурсы процессора.

Теперь, опишем работу скрипта по управление агентами. Важно отметить, что внутри нашего скрипта, у нас объявлен класс WHDAgent, который наследуется от уже реализованного класса Agent библиотеки ML-Agent, которую мы заблаговременно клонировали в наш проект. Теперь наша задача, переопределить такие методы, как: AgentAction(), AgentReset(), CollectObservations().

AgentAction определяет поведение нашего агента. На каждом шаге агента мы получаем вектор действий от нейронной сети, которые затем интерпретируются этим агентом. Мы смотрим на значения, которые лежат в vectorAction[0] (отвечает за перемещение по вертикали) и vectorAction[1] (отвечает за перемещение по горизонтали), может быть 3 исхода: 0, 1, 2. Тогда мы перемещаемся на 0 (остаемся на месте), на 1 и -1 соответственно. Именно в данном методе реализуется вознаграждение игрового агента. Стоит отметить, что в нашей модели используется только отрицательное вознаграждение. Пока агент не достиг своей цели, он стабильно получает штраф в виде константы (-0.01f / 30.0f), если агент сталкивается с контрагентом, он штрафуется уже сильнее на следующую константу: (-1.0f).

AgentReset нужен для того, чтобы вернуть агента в исходное состояние при столкновении с контрагентом и начать обучение заново с новым опытом.

CollectObservation нужен, чтобы собирать информацию о среде. Значение координаты x и y агента, дистанцию между агентом и Goal Area и информацию о столкновении агента с контрагентом.



**Рисунок 5. Пример поведение игрового агента в статичной игре.**

# 

# **Обучение и подбор гиперпараметров.**

Теперь, когда все готово, можем приступать к обучение наших агентов. ML-Agents плагин использует Proximal Policy Optimization (PPO). Данный метод обучения с подкреплением использует нейронную сеть для аппроксимации идеальной функции, которая отображает наблюдение агента на лучшее действие, которое агент может предпринять в данный момент времени. PPO реализован с помощью TensorFlow и выполняется в отдельном процессе с помощью python. Коммуникация с Unity3в происходит с помощью сокета.

Экспериментально удалось выяснить, что наиболее значимые параметры для обучения это:

* **beta** соответствует силе регуляризации энтропии. С помощью этого параметра можно добиться более равномерное изучение среды агентом. Чем больше этот параметр, тем более случайны будут его действия.

Типичный диапазон: 1e-4 – 1e-2

* **hidden\_units** соответствует количеству нейронов в каждом слое нейронной сети. Большее числа нейронов позволяет изучать более сложные функции.

Типичный диапазон: 32 - 512

* **num\_layers** соответствует количеству скрытых слоев.

Типичный диапазон: 1 – 3

# **Заключение**

С помощью таких инструментов как: ML-agents toolkit, Unity3d мне удалось воспроизвести первый уровень браузерной игры “The World’s Hardest Game” и успешно реализовать поведение контрагентов этой игры. Так же, был написан скрипт по управлению агентами внутри данного игрового сеттинга, который в последствии был использован для обучения модели-TensorFlow. Я считаю, что данный курсовой проект – отличное начало для знакомства с таким сложным и многогранным объектом изучения, как обучение с подкреплением. Благодаря ему, я смог понять изучить на базовом уровне Unity3d, что само по себе интересно, т. к. теперь я в состоянии писать простенькие игры. Так же, изучил очень популярную для разработки игровых агентов библиотеку ML-agents, у которой огромный потенциал и широкие возможности применения.

На данный момент я до сих пор занимаюсь подбором оптимальных параметров нейронной сети, т. к. они оказывают очень ключевое влияние на обучение, а мои модели пока-что далеки от идеала.

# 

# **Источники и ссылки**

1. https://github.com/Unity-Technologies/ml-agents
2. <https://becominghuman.ai/teaching-ai-to-play-flappy-bird-with-unity-70f7b661663d>
3. <https://unity3d.com/ru/machine-learning>
4. Reinforcement Learning An Introduction By Richard S.Sutton and Andrew G.Bartio
5. https://neerc.ifmo.ru/wiki/index.php?title=Обучение\_с\_подкреплением