

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS

Разработка эволюционной модели ИИ для адаптации к условиям окружающей среды
на локации

Обучающийся / Student Андреев Сергей Сергеевич

Факультет/институт/кластер/ Faculty/Institute/Cluster школа разработки видеоигр

Группа/Group J4221

Направление подготовки/ Subject area 09.04.03 Прикладная информатика

Образовательная программа / Educational program Технологии разработки
компьютерных игр 2022

Язык реализации ОП / Language of the educational program Русский

Квалификация/ Degree level Магистр

Руководитель ВКР/ Thesis supervisor Карсаков Андрей Сергеевич, кандидат технических
наук, Университет ИТМО, школа разработки видеоигр, доцент (квалификационная
категория "ординарный доцент")

Консультант/ Consultant Берзон Давид Ильич, ШРВ, инженер, неосн по совм.

Обучающийся/Student

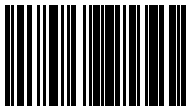
Документ подписан	
Андреев Сергей Сергеевич	
23.05.2024	

(эл. подпись/ signature)

Андреев Сергей
Сергеевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Карсаков Андрей Сергеевич	
22.05.2024	

(эл. подпись/ signature)

Карсаков
Андрей
Сергеевич

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Андреев Сергей Сергеевич

Факультет/институт/кластер/ Faculty/Institute/Cluster школа разработки видеоигр

Группа/Group J4221

Направление подготовки/ Subject area 09.04.03 Прикладная информатика

Образовательная программа / Educational program Технологии разработки компьютерных игр 2022

Язык реализации ОП / Language of the educational program Русский

Квалификация/ Degree level Магистр

Тема ВКР/ Thesis topic Разработка эволюционной модели ИИ для адаптации к условиям окружающей среды на локации

Руководитель ВКР/ Thesis supervisor Карсаков Андрей Сергеевич, кандидат технических наук, Университет ИТМО, школа разработки видеоигр, доцент (квалификационная категория "ординарный доцент")

Консультант/ Consultant Берзон Давид Ильич, ШРВ, инженер, неосн по совм.

Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

Тема в области прикладных исследований / Subject of applied research: да / yes

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Требуется разработать эволюционную модель ИИ для адаптации к условиям окружающей среды на локации, используя инновационный подход, основывающийся на актуальных исследованиях в соответствующей области.

Целью работы является разработка вышеназванной модели, её описание и получение экземпляра ИИ.

Задачами, решаемыми в рамках работы, являются: исследование научных работ по соответствующей тематике и анализ способов создания эволюционных моделей, анализ архитектур моделей, создание собственной архитектуры с учётом достоинств исследованных моделей, разработка собственной эволюционной модели ИИ для адаптации к условиям окружающей среды на локации, создание с помощью данной эволюционной модели экземпляра ИИ.

Форма представления материалов ВКР / Format(s) of thesis materials:

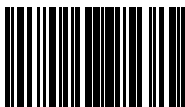
текст ВКР в формате pdf, презентация в формате pptx, программный код

Дата выдачи задания / Assignment issued on: 22.02.2024

Срок представления готовой ВКР / Deadline for final edition of the thesis 24.05.2024

СОГЛАСОВАНО / AGREED:

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Карсаков Андрей Сергеевич	
23.04.2024	

(эл. подпись)

Карсаков
Андрей
Сергеевич

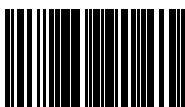
Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан	
Андреев Сергей Сергеевич	
30.04.2024	

(эл. подпись)

Андреев Сергей
Сергеевич

Руководитель ОП/ Head
of educational program

Документ подписан	
Карсаков Андрей Сергеевич	
22.05.2024	

(эл. подпись)

Карсаков
Андрей
Сергеевич

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Андреев Сергей Сергеевич
Факультет/институт/кластер/ Faculty/Institute/Cluster школа разработки видеоигр
Группа/Group J4221
Направление подготовки/ Subject area 09.04.03 Прикладная информатика
Образовательная программа / Educational program Технологии разработки компьютерных игр 2022
Язык реализации ОП / Language of the educational program Русский
Квалификация/ Degree level Магистр
Тема ВКР/ Thesis topic Разработка эволюционной модели ИИ для адаптации к условиям окружающей среды на локации
Руководитель ВКР/ Thesis supervisor Карсаков Андрей Сергеевич, кандидат технических наук, Университет ИТМО, школа разработки видеоигр, доцент (квалификационная категория "ординарный доцент")
Консультант/ Consultant Берзон Давид Ильич, ШРВ, инженер, неосн по совм.

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Оптимизация процесса прототипирования и разработки игрового ИИ при помощи средств эволюционного моделирования.

Задачи, решаемые в ВКР / Research tasks

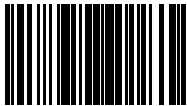
Анализ предметной области эволюционного моделирования в играх, формулировка гипотезы по оптимизации процесса разработки игрового ИИ при помощи эволюционного моделирования, составление плана работы и разработка собственной эволюционной модели.

Краткая характеристика полученных результатов / Short summary of results/findings

В результате работы был проведен анализ решений в области эволюционного моделирования, затем на основе данных решений создана собственная архитектура эволюционной модели и сама эволюционная модель, в том числе среда обучения ИИ. Проведено тестирование модели и её последующая оптимизация с учётом игровых задач ИИ, получены экземпляры ИИ. Оценка данных экземпляров ИИ показала возможность применения созданной эволюционной модели в качестве средства оптимизации процесса разработки игрового ИИ.

Обучающийся/Student

Документ подписан	
----------------------	--


	
Андреев Сергей Сергеевич	
23.05.2024	

(эл. подпись/ signature)

Андреев Сергей
Сергеевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Карсаков Андрей Сергеевич	
22.05.2024	

(эл. подпись/ signature)

Карсаков
Андрей
Сергеевич

(Фамилия И.О./ name
and surname)

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ.....	8
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ.....	9
ВВЕДЕНИЕ.....	10
1 Искусственный интеллект в компьютерных играх	12
1.1 Использование ИИ в компьютерных играх.....	12
1.2 Восприятие и реакция ИИ на игровое состояние	12
1.3 Использование эволюционного моделирования при создании ИИ.....	14
2 Создание ИИ при помощи эволюционного моделирования	15
2.1 Основные подходы эволюционного моделирования	15
2.2 Методы создания эволюционной модели.....	16
2.2.1 MCTS	16
2.2.2 RHEA	18
2.2.3 NTBEA.....	20
2.3 Сравнение методов создания эволюционных моделей ИИ	22
3 Анализ архитектуры эволюционных моделей	25
3.1 Эволюционная модель GeneBot.....	25
3.1.1 Постановка задачи.....	25
3.1.2 Решение задачи.....	26
3.2 Многоуровневая эволюционная модель Mario AI	29
3.2.1 Постановка задачи.....	29
3.2.2 Решение задачи.....	30
3.2.3 Выводы об архитектуре	33
4 Теоретическая модель.....	36
4.1 Постановка задачи.....	36
4.2 Созданная архитектура	36
5 Обзор средств построения эволюционной модели.....	39
5.1 Используемые программные средства.....	39
5.2 Unity ML-Agents	40

6 Создание эволюционной модели	43
6.1 Построение среды обучения	43
6.2 Настройка параметров агента	45
7 Тестирование эволюционной модели	47
7.1 Обучение агента	47
7.2 Первичные результаты обучения	48
8 Оптимизация эволюционной модели	51
8.1 Использование Imitation Learning	51
8.2 Модификация среды обучения	51
8.3 Повторное обучение	52
8.3.1 Параметры Imitation Learning	52
8.3.2 Обучение с сенсорами границ трека	55
8.3.3 Обучение с сенсорами поверхности	57
8.4 Результаты оптимизации модели	59
9 Выводы по результатам работы	61
9.1 Результаты тестирования и оптимизации	61
9.2. Результаты построения эволюционной модели	62
ЗАКЛЮЧЕНИЕ	64
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	65

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ИИ – искусственный интеллект

AI – artificial intelligence

API – application programming interface

BT – behavior tree

GAIL – generative adversarial imitation learning

GE – grammatical evolution

GVGAI – general video game artificial intelligence

GVGP – general video game playing

EA – evolutionary algorithm

MAB – multi-armed bandit

MCTS – Monte Carlo tree search

NN – neural network

NPC – non-playable character

NTBEA – N-tuple bandit evolutionary algorithm

PPO – proximal policy optimization

RHEA – rolling horizon evolutionary algorithm

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

AAA-проект – класс высокобюджетных компьютерных игр

Агент – автономная единица искусственного интеллекта, способная выполнять свои игровые задачи без контроля со стороны игрока

Движок – набор программного обеспечения, необходимый для создания и работы компьютерных игр

ML-Agents – инструментарий машинного обучения игровых агентов в среде Unity

Unity –движок, созданный компанией Unity Technologies

Unreal Engine –движок, созданный компанией Epic Games

ВВЕДЕНИЕ

Искусственный интеллект используется в компьютерных играх для создания отзывчивого, адаптивного или интеллектуального поведения, в первую очередь у неигровых персонажей, аналогичного человеческому интеллекту. Развитие игровой индустрии усложняет структуру игровых проектов, а следовательно, и механики взаимодействия ИИ с игровым пространством. Появляется множество внешних факторов, учитываемых ИИ, часто определяющий стиль игры ИИ. Поскольку данные наборы взаимодействий требуют программирования ИИ, процесс разработки весьма усложняется. Для оптимизации процесса создания ИИ существуют решения в виде эволюционных моделей, использующих различные алгоритмы и способные генерировать ИИ с использованием заданных параметров. Эффективность таких моделей определяется жанром игр, учитываемыми игровыми параметрами, а также используемыми при создании эволюционных моделей алгоритмами.

В данной работе рассматриваются основные аспекты создания эволюционных моделей для получения при их помощи игрового ИИ. Изучаются алгоритмы, которые могут быть применены для создания эволюционной модели ИИ, учитывающего внутриигровые факторы, в том числе, условия окружающей среды на локации, а также подробно изучаются особенности построения архитектуры эволюционных моделей. Определяются составляющие, их функции, причины, по которым они используются в моделях. Далее на основе достоинств ранее построенных архитектур строится собственная архитектура, которая используется для построения своей эволюционной модели. Был сделан обзор программных средств построения эволюционной модели, проведено сравнение с учетом возможностей оптимизации и производительности игровой среды. После выбора соответствующих задачам и возможностям средств построения в качестве среды был выбран движок Unity. В нем происходило обучение игровых агентов при помощи Unity ML-Agents.

После построения среды обучения и эволюционной модели было проведено первичное тестирование работы модели и оценка ее эффективности, сделаны выводы, определены дальнейшие направления разработки. Результаты первичного обучения позволили заключить, что, хотя эволюционная модель позволяет обучать ИИ выполнять свои игровые задачи, скорость и качество обучения являются недостаточными для получения

качественного ИИ. Для решения задач научной работы требовалось провести оптимизацию эволюционной модели. При оптимизации учитывались результаты первичного тестирования модели, а также особенности работы инструментария Unity ML-Agents. Для оптимизации модели была модифицирована среда обучения, а также введена новая методика обучения – Imitation Learning. Данная методика позволяет использовать для обучения агентов демонстрации поведения, за счёт этого скорость обучения может быть существенно увеличена.

Использование новой методики позволило ускорить процесс получения ИИ. За счёт ранее записанных демонстраций агенты начинали выполнять свои игровые задачи быстрее. Повторное обучения позволило заключить, что методика Imitation Learning является оптимальной для использования в данной эволюционной модели. Результатом стала разработка эволюционной модели ИИ, а также получение качественных экземпляров ИИ.

1 Искусственный интеллект в компьютерных играх

1.1 Использование ИИ в компьютерных играх

Искусственный интеллект в компьютерных играх представляет собой набор программных методов, используемых для создания иллюзии разума у NPC через поведение. ИИ в играх включает в себя алгоритмы теории управления, робототехники, компьютерной графики и информатики в целом [1].

Основной задачей игрового ИИ является определение действий сущности или агента в зависимости от текущих условий игры. Агентом, как правило, является персонаж игры, но это могут быть машина, робот или даже целая группа сущностей. В любом случае это объект, который отслеживает состояние окружения, принимающий на основании этого решения и действующий в соответствии с ними [2].

Данный цикл образуется с помощью последовательности «восприятие-мышление-действие» (Sense/Think/Act) [3]:

- 1) Восприятие: агент распознаёт – или ему сообщают – информацию об окружении, которая может влиять на его поведение (например, находящиеся поблизости опасности, собираемые предметы, важные точки и так далее);
- 2) Мышление: агент принимает решение о том, как поступить в ответ (например, решает, достаточно ли безопасно собрать предметы, стоит ли ему сражаться или лучше сначала спрятаться);
- 3) Действие: агент выполняет действия для реализации своих решений (например, начинает двигаться по маршруту к врагу или к предмету, и так далее);

Затем из-за действий персонажей ситуация изменяется, поэтому цикл должен повториться с новыми данными.

Подобный алгоритм позволяет добиться реалистичности множества игровых ситуаций, при этом определяющимися лишь диапазоном входных данных и набором возможных решений.

1.2 Восприятие и реакция ИИ на игровое состояние

Задачи ИИ в играх требуют от агентов восприятия состояния игровой среды и реакции на него. Факторами, на которые реагирует ИИ, могут быть игрок, объекты, другие агенты и игровые переменные. Особенности восприятия регулируются базовым программным обеспечением компьютерной игры (игровым движком). Поскольку такое программное

обеспечение может быть рассмотрено, как основание для разработки множества различных игр без существенных изменений, такая система позволяет пользоваться существующими алгоритмами восприятия либо создавать новые и использовать их в среде движка [4].

Задачи ИИ реального мира, особенно те, что актуальны сегодня, обычно сосредоточены на «восприятии». Например, беспилотные дроны должны получать изображения находящегося перед ними пространства, комбинируя их с другими данными (радара и лидара) и пытаться интерпретировать полученные данные. Обычно эта задача решается машинным обучением, которое особо хорошо работает с большими массивами данных реального мира (фотографиями или кадрами видео) и придаёт им значение, извлекая семантическую информацию [5].

Игры необычны тем, что для извлечения этой информации им не нужна сложная система, поскольку она является неотъемлемой частью симуляции. Нет необходимости выполнять алгоритмы распознавания изображений, чтобы обнаружить врага перед собой; игра знает, что там есть враг и может передать эту информацию непосредственно процессу принятия решений. Поэтому «восприятие» в этом цикле обычно сильно упрощено, а вся сложность возникает в реализации «мышления» и «действия» [3].

С текущими темпами развития игровой индустрии поле «действия» становится весьма разнообразным. Требования к поведению NPC возрастают, его стараются делать более реалистичным, заставляют ИИ учитывать множество игровых факторов, а вместе с этим усложняют и процесс «мышления». Так, например, алгоритм действий ИИ может изменяться в зависимости от поведения игрока или определённых игровых событий. Состояние окружающего мира в игре может заставлять ИИ существенно менять своё поведение. В качестве примера можно привести экшн-хоррор «Days Gone», где противники реагируют на внутриигровую погоду: в дождь их поведение становится более агрессивным, а в холодную погоду увеличивается урон. Если оба этих эффекта объединяются, противники получают наиболее эффективное состояние для атаки игрока.

Количество возможных комбинаций внешних факторов, влияющих непосредственно на ИИ, огромно: паттерны поведения игрока, внутриигровые события, условия игровой окружающей среды. В современных AAA-проектах такие факторы исчисляются сотнями, и все влияют непосредственно на поведение ИИ. В связи с этим, разработчикам необходимо создавать

алгоритмы поведения для каждого конкретного случая. Такой подход хоть и позволяет наиболее точно описать действия ИИ, всё же является крайне трудозатратным, в связи с чем процесс разработки игр замедляется [6].

Процесс создания такого ИИ может быть значительно ускорен с использованием эволюционного моделирования и машинного обучения.

1.3 Использование эволюционного моделирования при создании ИИ

Эволюционное моделирование представляет собой применение механизмов естественной эволюции при синтезе сложных систем обработки информации. Фактически, это замена процесс моделирования сложного объекта моделированием его эволюции. Подобный подход широко применяется при создании нейронных сетей и использовании машинного обучения [7].

Игровой ИИ, в отличие от алгоритмов машинного обучения, обычно не тренируется заранее. При разработке игры непрактично создавать нейронную сеть для наблюдения за десятками тысяч игроков, чтобы найти наилучший способ играть против них, поскольку игра ещё не выпущена и игроков у неё нет. Обычно предполагается, что игра должна развлекать и бросать игроку вызов, а не работать «оптимально». Поэтому, даже если и можно обучить агентов противостоять игрокам наилучшим образом, при создании ИИ необходимо действовать иначе. При этом к агентам предъявляют требование «реалистичного» поведения, чтобы игроки ощущали, что соревнуются с человекоподобными противниками [8].

Машинное обучение позволяет системе научиться анализировать определённую информацию в виртуальной среде и с её использованием создавать ИИ с поведением, более приближенным к реалистичному. Применение же универсальной эволюционной модели позволило бы создавать ИИ для игр различных жанров [9].

2 Создание ИИ при помощи эволюционного моделирования

2.1 Основные подходы эволюционного моделирования

Эволюционный алгоритм (англ. Evolutionary Algorithm, сокр. EA) представляет собой подмножество эволюционных вычислений, общего алгоритма метаэвристической оптимизации, основанного на популяции [10].

При метаэвристическом подходе могут использоваться различные принципы исследования пространства решений и стратегии адаптации для учета полученной информации. На каждой итерации метаэвристики могут выполняться операции с единственным текущим решением (или частичным решением), либо с набором (популяцией) решений. Подчиненные эвристики могут быть процедурами высокого или низкого уровня, простым локальным поиском, градиентным методом построения решения или классическими оптимизационными процедурами [11].

EA использует механизмы, вдохновленные биологической эволюцией, такие как размножение, мутация, рекомбинация и отбор. Возможные решения задачи оптимизации играют роль индивидов в популяции, а функция пригодности определяет качество решений. Затем, после повторного применения вышеуказанных операторов, происходит эволюция популяции. Данная система позволяет получить ИИ, способный эффективно решать соответствующий класс задач.

Эволюционное моделирование в области ИИ можно рассматривать, как управляемый случайный поиск популяции, удовлетворяющей определенным требованиям [12]. Существует 4 различных подхода, использующих данную концепцию, их перечень приведён в таблице 1.

Таблица 1 – Основные подходы эволюционного моделирования

Название подхода	Основные черты
1	2
Генетические алгоритмы	Использование основных операторов отбора, рекомбинации и мутации в генетически закодированных решениях данной задачи. Оператор создаёт рабочее решение, повторяя его на протяжении множества поколений.
Эволюционные стратегии	Отбор осуществляется путем детерминированного отбора наиболее результативных индивидуумов из популяции. Мутация работает со случайными дельтами в определенном распределении.

1	2
Генетическое программирование	Программы генерируются случайным образом из пула примитивов. Затем применяется генетический алгоритм. Работоспособность программы вычисляется путем ее выполнения и оценки ее производительности с помощью функции.
Эволюционное программирование	Отбор осуществляется путем детерминированного отбора наиболее результативных индивидуумов из популяции. Мутация работает с заданными параметрами.

Генетические алгоритмы являются наиболее часто используемой категорией эволюционных механизмов.

2.2 Методы создания эволюционной модели

При разработке игр и создании ИИ используется подход, называемый General Video Game Playing (GVGP). Целью GVGP является создание агентов, способных играть в многие потенциально неизвестные видеоигры без использования каких-либо предварительно закодированных данных об этих играх [13].

Наиболее успешные агенты GVGP в основном используют методы, основанные на Monte Carlo Tree Search (MCTS) и эволюционных подходах, часто в сочетании с многочисленными усовершенствованиями [14]. Другими наиболее популярными и исследованными эволюционными подходами к GVGP являются эволюционные алгоритмы Rolling Horizon Evolutionary Algorithm (RHEA) и N-Tuple Bandit Evolutionary Algorithm (NTBEA) [15].

2.2.1 MCTS

Monte Carlo Tree Search (MCTS) – алгоритм принятия решений, часто используемый в играх в качестве основы искусственного интеллекта [16].

Основное внимание MCTS уделяется анализу наиболее перспективных ходов, расширению дерева поиска на основе случайной выборки пространства поиска. Применение поиска по дереву Монте-Карло в играх основано на множестве розыгрышей, также называемых ролл-аутами. В каждом раунде игра проигрывается до самого конца путем случайного выбора ходов. Конечный результат каждой игры затем используется для взвешивания узлов в игровом дереве, чтобы в будущих играх с большей вероятностью были выбраны лучшие узлы.

Каждый раунд поиска по дереву Монте-Карло состоит из четырёх этапов [16]:

- 1) Выбор – путь начинается с корневого узла R, затем последовательно выбираются дочерние узлы, пока не будет достигнут конечный узел L. Корень R – текущее состояние игры, лист L – любой узел, у которого есть потенциальный дочерний элемент, с которого еще не было инициировано моделирование.
- 2) Расширение – если L не завершит игру решительно (например, победа / проигрыш / ничья) для любого игрока, создайте один (или несколько) дочерних узлов и выберите узел C из одного из них. Дочерние узлы – это любые допустимые ходы из игровой позиции, определенной L.
- 3) Моделирование – завершите одно случайное воспроизведение с узла C. Этот шаг иногда также называют воспроизведением или разворачиванием. Ход игры может быть таким же простым, как выбор равномерных случайных ходов до тех пор, пока игра не будет решена (например, в шахматах партия выиграна, проиграна или сыграна вничью).
- 4) Обратное распространение – результат воспроизведения используется для обновления информации в узлах на пути от C до R.

Пример работы алгоритма приведён на рисунке ниже:

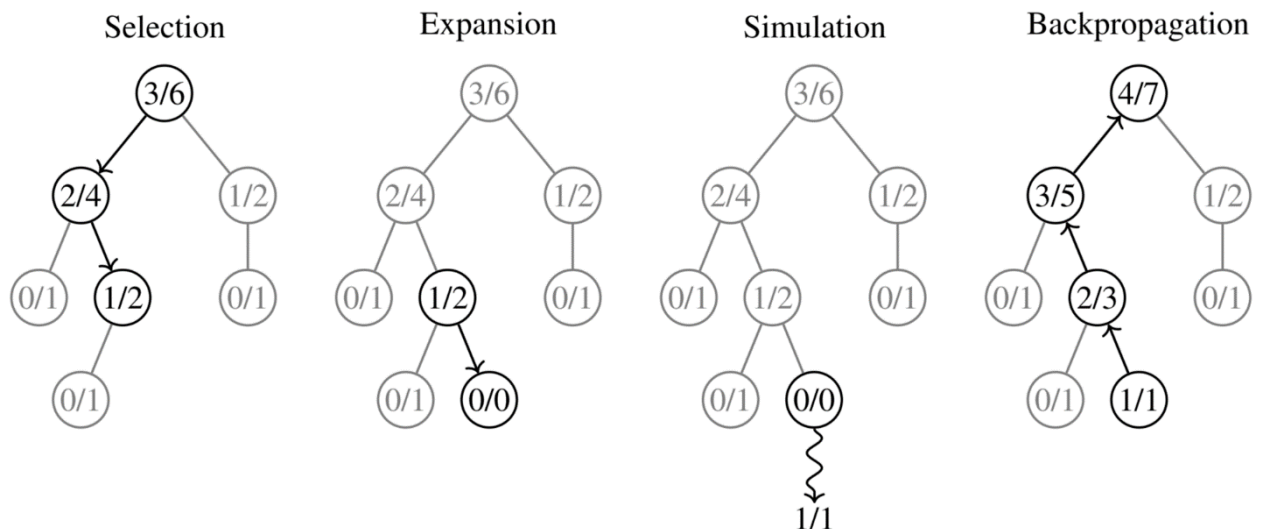


Рисунок 1 – Визуализация алгоритма MSTC

На данном графе показаны шаги, связанные с одним решением, причем каждый узел показывает отношение выигрышей к общему количеству игр с этой точки в игровом дереве для игрока, которого представляет этот узел. Раунды поиска повторяются до тех пор, пока остается время, отведенное на

ход. Затем в качестве окончательного ответа выбирается ход с наибольшим количеством выполненных симуляций (т.е. с наибольшим знаменателем).

2.2.2 RHEA

Rolling Horizon Evolutionary Algorithm (RHEA) – эволюционный алгоритм, основывающийся на выполнении внутриигровой последовательности действий на каждом игровом тике с ограниченным временем вычисления на выполнение [15].

В данном алгоритме генотип описывается как вектор целых чисел длины L (у каждого длина индивидуальна), где каждое целое число a находится в диапазоне $[0, N)$, N – максимальное количество действий в данном игровом состоянии S . Это позволяет применить генотип как основу решений фенотипа – совершаемую им последовательность действий, воспроизводимых при игре, начиная с состояния S . Для оценки состояния RHEA использует внутреннюю модель мира, на основе которого предпринимает последовательность действий. Достигнутое в конце состояние игры затем оценивается с помощью эвристической функции h , и это значение становится пригодностью индивидуума: следовательно, алгоритм разрабатывает последовательности действий, которые приводят к наилучшему результату игры, ограниченному диапазоном исследования L . Эвристическая функция h всегда поддерживается в общей форме на протяжении всех экспериментов, это направлено на максимизацию игрового счета, в то же время способствуя выигрышам и препятствуя проигрышам, в соответствии с формулой ниже:

$$h(s) = \begin{cases} 1 & \text{win} = \text{True} \\ 0 & \text{win} = \text{False}, \\ \text{score} & \text{otherwise} \end{cases} \quad (1)$$

где s – оцениваемое состояние игры, а score – оценка игры, нормализованная в промежутке $(0, 1)$.

Работа алгоритма RHEA начинается с инициализации популяции из P особей длины L , затем он случайным образом и оценивает их. После каждого раунда алгоритм генерирует P потомков путем многократного выбора родителей с помощью процесса генетического отбора, скрещивает их с помощью равномерного скрещивания для создания дочернего элемента и мутирует дочерний элемент с помощью равномерной мутации, прежде чем добавлять его в пул потомства. Лучшие особи $P - E$ как из родительского пула,

так и из пула потомства добавляются к следующему поколению, и процесс повторяется. Схема алгоритма изображена на рисунке 2:

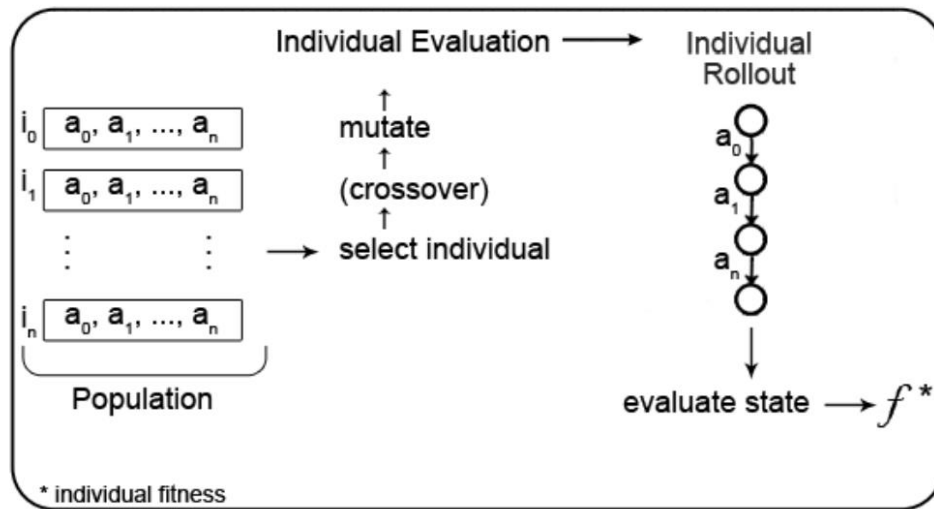


Рисунок 2 – Схема алгоритма RHEA

RHEA обладает также рядом преимуществ по сравнению с MCTS: RHEA легко распараллеливается, при этом каждый индивид независим от остальных и от процесса поиска. При этом существует возможность непрерывной работы, реализуемая в виде «скользящего горизонта»: время исследования может быть расширено применением нового временного промежутка [17].

Схема параллельной работы и расширение временного отрезка показаны на рисунке 3:

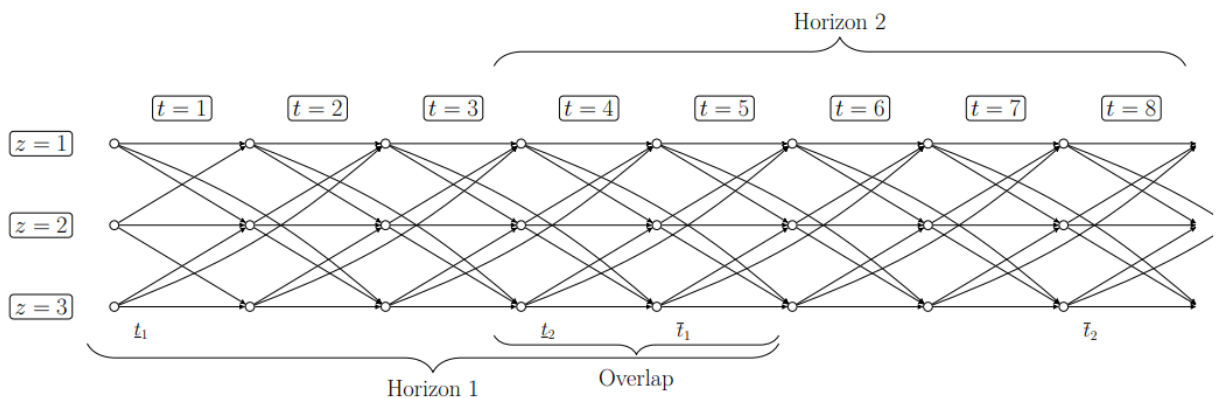


Рисунок 3 – Последовательность раундов RHEA

Алгоритм способен обрабатывать непрерывные действия без изменений, поэтому легко адаптироваться к играм для n игроков с минимальными изменениями благодаря совместной эволюции. Это делает его подходящим

для множества улучшений и модификаций, а следовательно – высокоадаптивным и управляемым для различных задач [18].

2.2.3 NTBEA

N-Tuple Bandit Evolutionary Algorithm (NTBEA) – алгоритм оптимизации, основанный на эволюционном алгоритме, разработанный для дорогостоящих задач дискретной (комбинаторной) оптимизации. NTBEA сочетает эволюционный поиск с алгоритмом Multi-Armed Bandit (MAB) [19].

Модель фитнес-ландшафта (степени приспособленности) строится и обновляется итеративно с использованием оценок решений. Поиск решений-кандидатов выполняется в пространстве моделей, т.е. в модели ландшафта пригодности бандитов с N-кортежами, где решения-кандидаты могут быть быстро оценены [19].

Затем происходит выборка в реальном (относительно дорогом) пространстве поиска проблем, где ожидается, что значение пригодности будет зашумленным. Затем модель обновляется значением пригодности этой точки решения. Процесс повторяется до тех пор, пока не будет выполнено какое-либо условие завершения.

Структура алгоритма показана на рисунке 4:

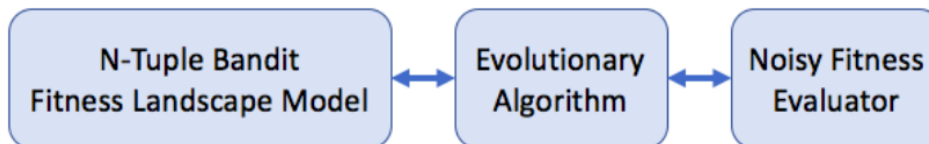


Рисунок 4 – Ключевые компоненты NTBEA

Ключевой частью алгоритма является функция значения, используемая для выборки в большом пространстве поиска. В этом контексте большой означает, что размер пространства поиска больше, чем количество разрешенных оценок пригодности, таким образом, невозможно правильно оценить (правильная оценка может включать повторную выборку из-за шума) каждую из возможных точек решения. Вместо этого необходимо смоделировать взаимосвязь между точками в пространстве поиска и соответствующим образом выполнить выборку [19].

Алгоритм UCSB представляет собой простой алгоритм многорукого бандита. Значение UCSB для каждой руки i (имеется ввиду, запуски игры для «бандита») определяется как:

$$UCB_i = \hat{X}_i + k \sqrt{\frac{\ln n}{n_i + \epsilon}}. \quad (2)$$

Эмпирическое среднее вознаграждение за игру i равно X_i – значение, называемое термином эксплуатации. Термин управляет исследованием, где n – общее количество раз, когда был сыгран этот бандит, а n_i – количество раз, когда была сыграна рука i . Термин k называется фактором разведки: более высокие значения k приводят к поисковому поиску, низкие значения приводят к более «эксплуатационному» поиску. Каждое измерение пространства поиска моделируется как независимый результат игры для каждого возможного значения. Это значение используется для управления тем, следует ли играть каждую руку хотя бы один раз.

Работа алгоритма начинается с случайной инициализации решения o или с заданного решения (называемого исходным кодом, а процесс – алгоритмом заполнения). Затем он оценивает по одному решению за раз, используя метод оценки, определенный конкретным приложением, и добавляет его к своей внутренней модели из n кортежей – то есть регистрируются все комбинации из n параметров, чтобы обеспечить соответствие оцениваемого решения. Используются кортежи 1, 2 и L , где L – длина решения.

Затем соседи решения генерируются путем равномерной случайной мутации с вероятностью $1 / L$, образуя окрестности N . Пригодность всех соседей оценивается на основе ранее наблюдаемых значений n -кортежей, после чего вычисляются две статистики: средние значения соседних кортежей, а также среднее количество раз, когда каждый кортеж был ранее исследован.

Эти две статистики используются в уравнении задачи одноруких бандитов, называемом «bandit equation» (формула 3). Это необходимо выбора следующего соседа, который станет текущим оцениваемым решением.

$$o' = \arg \max_{o \in N} \{Q(o) + k \times \exp(o) + noise\}. \quad (3)$$

Уравнение направлено на достижение баланса между многообещающими решениями за счет использования высоких значений пригодности $Q(o)$ и неопределенными решениями за счет изучения тех, которые менее заметны в процессе $\exp(o)$. Константа $k = 2$ задает фокус алгоритма, будь то более эксплуатационный или более исследовательский.

Небольшой случайный шум (максимум = 0,5) добавляется к конечному значению каждого соседа, чтобы случайным образом разорвать связи.

Система N-Tuple моделирует статистику, аппроксимируя пригодность и количество оценок каждой смоделированной комбинации параметров [19].

2.3 Сравнение методов создания эволюционных моделей ИИ

Описание методов, используемых при создании эволюционных моделей, приведённое выше, демонстрирует, что разные алгоритмы могут показывать различный уровень эффективности при решении различных типов задач. Дополнительно стоит учитывать возможности модификации и применении отдельных элементов алгоритмов вместе.

Так, сравнение алгоритмов RHEA и MCTS показывает, что созданный с их использованием ИИ демонстрирует различную степень эффективности в ряде тестов [20].

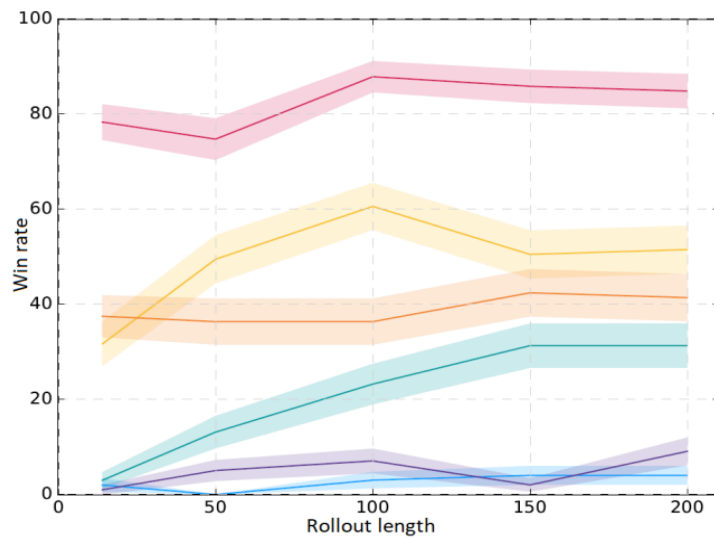
Результаты показывают, что общая тенденция заключается увеличении процента выигрышей ИИ при более длинном развёртывании L (число поколений при генерации ИИ). Однако есть момент, когда процесс улучшения результатов для RHEA останавливается (последняя колонка в таблице на рисунке 5).

Alg	L	Sparse	Dense	Overall
RHEA	14	25.59 (2.82)	58.04 (1.73)	48.31 (2.05)
	50	29.80 (3.30)	61.33 (1.66)	51.80 (2.14)
	100	36.36 (3.59)	61.04 (1.87)	53.55 (2.37)
	150	36.03 (3.59)	62.63 (2.10)	54.60 (2.53)
	200	37.04 (3.85)	60.89 (1.93)	53.70 (2.49)
MCTS	14	6.90 (1.79)	54.76 (1.65)	40.40 (1.69)
	50	14.31 (3.29)	53.54 (2.08)	41.70 (2.42)
	100	22.39 (3.79)	54.18 (1.58)	44.50 (2.23)
	150	26.77 (3.94)	53.75 (1.49)	45.60 (2.21)
	200	30.98 (4.14)	53.61 (1.52)	46.70 (2.29)

Рисунок 5 – Доля выигрышей ИИ для вариации с долгим развёртыванием

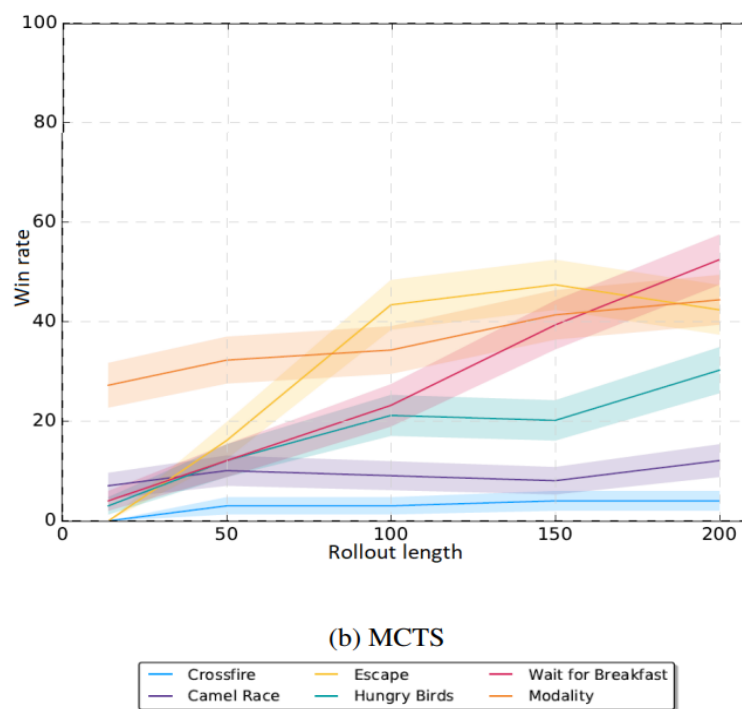
При рассмотрении различных систем вознаграждения улучшение заметно только в играх с редкими наградами, в то время как производительность в играх с постоянно увеличивающимся вознаграждением остается довольно постоянной.

Далее авторы [20] провели сравнение коэффициента выигрыша в играх с редким вознаграждением для двух алгоритмов (рисунки 6 и 7):



(a) RHEA

Рисунок 6 – Коэффициент выигрыша в играх с редкими наградами для RHEA



(b) MCTS

Рисунок 7 – Коэффициент выигрыша в играх с редкими наградами для MCTS

Данные, полученные в исследовании, позволили авторам заключить, что в случае для игр с редкими наградами алгоритм MCTS показывает большую эффективность, чем RHEA, что особенно заметно на примере «Wait for Breakfast».

Это показывает, что RHEA более чувствителен к играм, требующим быстрого принятия решений, тогда как MCTS извлекает выгоду из настроек, которые помогают ему традиционно разбираться в пошаговых играх [20].

Отмечается также, что, хотя в этих экспериментах используется фреймворк GVGAИ, традиционно соотносящийся исключительно с компьютерными играми, применимость результатов выходит за рамки этих игр. В частности, эта работа сосредоточена на модификациях, направленных на преодоление наличия редких вознаграждений, проблемы, присутствующей не только в других играх, но также и в других сценариях реальной жизни, таких как инженерия или робототехника.

3 Анализ архитектуры эволюционных моделей

3.1 Эволюционная модель GeneBot

3.1.1 Постановка задачи

В статье [21] исследуется производительность и результаты эволюционного алгоритма, специально разработанного для создания ИИ (в контексте статьи - бота), играющего в Planet Wars.

Planet Wars – это упрощенная версия игры Galcon [22], направленная на проведение боев ботов. Правила Galcon состоят в следующем:

- 1) Игроки начинают с 1-3 больших «домашних планет» и отправляют корабли, чтобы завоевать другие планеты вокруг них.
- 2) Цифры на каждой планете указывают, сколько кораблей потребуется, чтобы их завоевать.
- 3) Цифры на собственной планете игрока указывают на количество кораблей, находящихся на его планете.
- 4) Каждая планета, которой владеет игрок, производит корабли для этого игрока, при этом большее количество кораблей производится с большей скоростью, в зависимости от размера планеты.
- 5) Игроки могут выбирать, какой процент кораблей отправлять с планеты, и игроки могут перенаправлять корабли в полете.
- 6) Цель игры состоит в том, чтобы победить другого игрока (игроков), уничтожив все вражеские планеты.

Задача решалась авторами в рамках задач Google AI Challenge [23] – соревнования по созданию ИИ (бота), который способен играть с ботами других участников в Planet Wars.

В конкурсной версии игры от Google участвуют два игрока. Матч Planet Wars происходит на карте, содержащей несколько планет, и каждой из них присвоено число, которое представляет количество звездолетов, находящихся на планете в данный момент. Схема матча показана на рисунке 8.



В рамках решения задачи требовалось создать ИИ (бота), способного играть с ботами других участников.

Условия игры требовали, чтобы бот имел дело с несколькими целевыми планетами, достигая при этом определенной степени адаптивности, чтобы побеждать разных противников в разных сценариях.

Механизм принятия решений бота изначально основан на наборе правил, которые были определены после эмпирического исследования. Для определения его поведения нужно определить набор констант, весов и

вероятностей, которые влияют на эти правила и, следовательно, общее поведение бота. Первичный бот с заданным числом настраиваемых констант был назван авторами «AresBot».

На рисунке 9 представлено дерево поведения бота. Отдельные константы определяют выбор ветви поведения ботом на каждом узле графа.

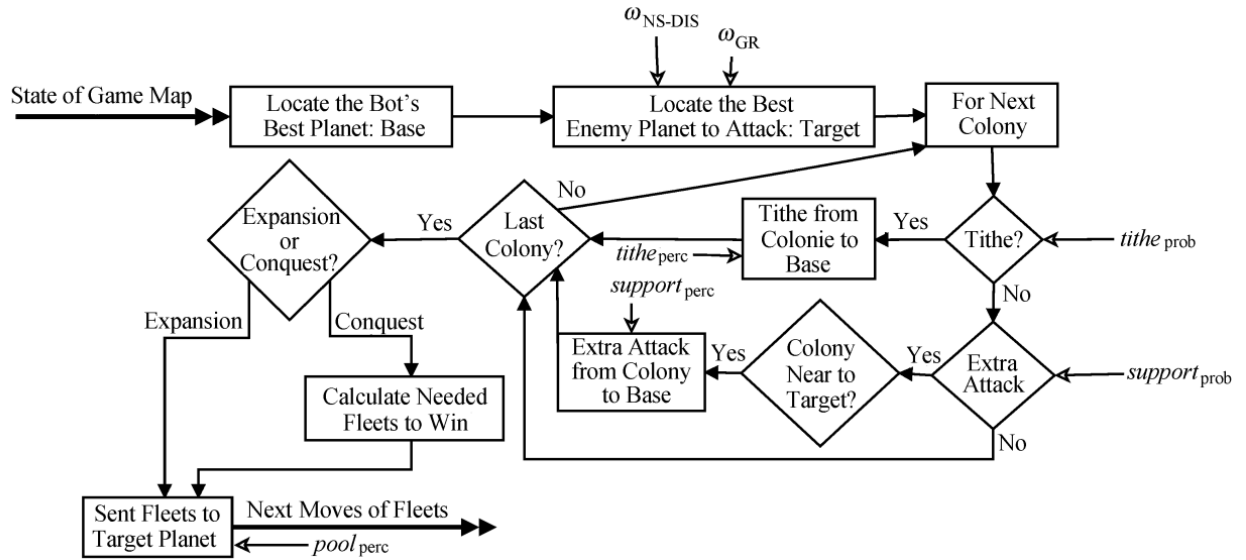


Рисунок 9 – Дерево поведения AresBot

Каждый параметр принимает значения в разном диапазоне, в зависимости от игрового состояния и значимости параметра. Эти значения учитываются в выражении, используемом ботом для принятия решений.

Например, функция, которая присваивает оценку/стоимость целевой планете p , определяется следующим образом:

$$Score(p) = \frac{p.NubStarships \times \omega_{NS-DIS} \times Dist(base, p)}{1 + p.GowthRate \times \omega_{GR}}. \quad (4)$$

Затем AresBot снабжается усовершенствованным механизмом принятия решений и тщательно анализируются результаты, полученные при соревновании с другими ботами (бот, предлагаемый Google в качестве спарринг-партнера, и закриптованный бот с заранее установленным поведением). Оценка возможных решений основана на результате недетерминированных сражений (и взаимодействии с окружающей средой) против других ботов, исход которых зависит как от случайных розыгрышей, так и от действий противников. Этому боту авторы дали название «GeneBot».

Пример оценки исхода партий на фитнес-ландшафте представлена на рисунке 10:

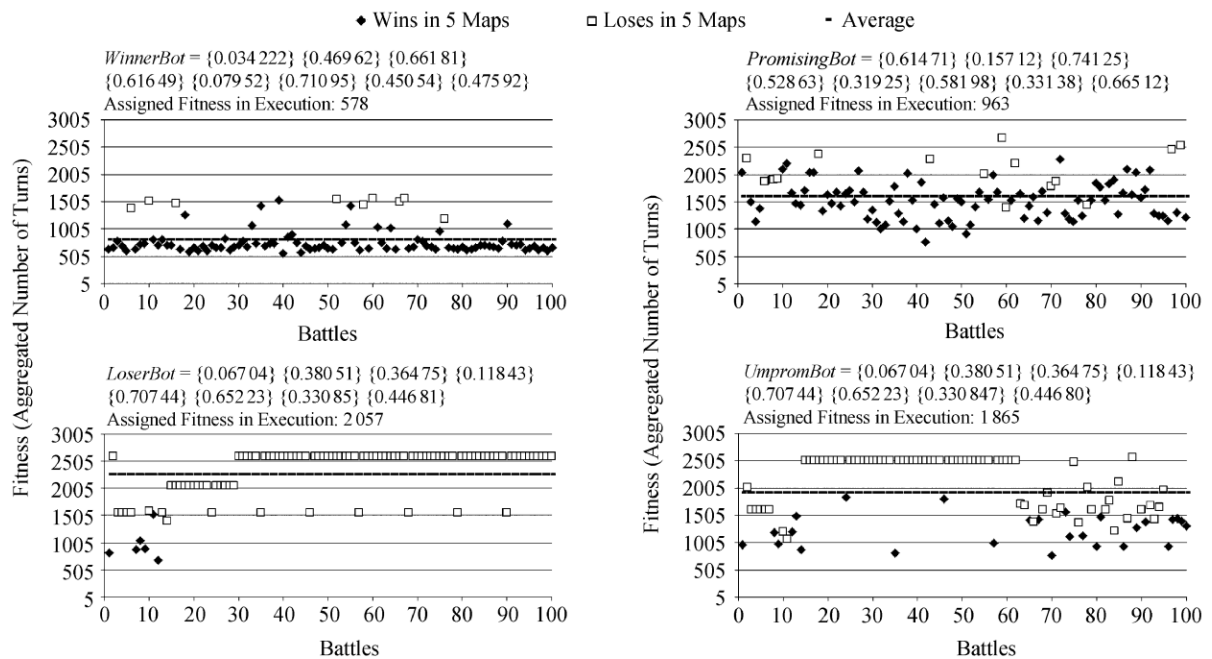


Рисунок 10 – Оценка пригодности 4 ботов, сыгравших в 100 играх

Такая оценка позволяет выбрать бота или нескольких, наиболее соответствующих требованиям отбора, либо же просто с лучшими показателями. Их начальные параметры будут использованы для создания следующей популяции.

3.1.3 Выводы об архитектуре

Главное преимущество модели GeneBot, по мнению авторов – снижение разброса успешности исхода партии для бота с помощью повторяющихся сражений и переоценок (в различных условиях, например, при выборе карты для каждой игры случайным образом).

Архитектура эволюционной модели для создания GeneBot представлена на рисунке 11:

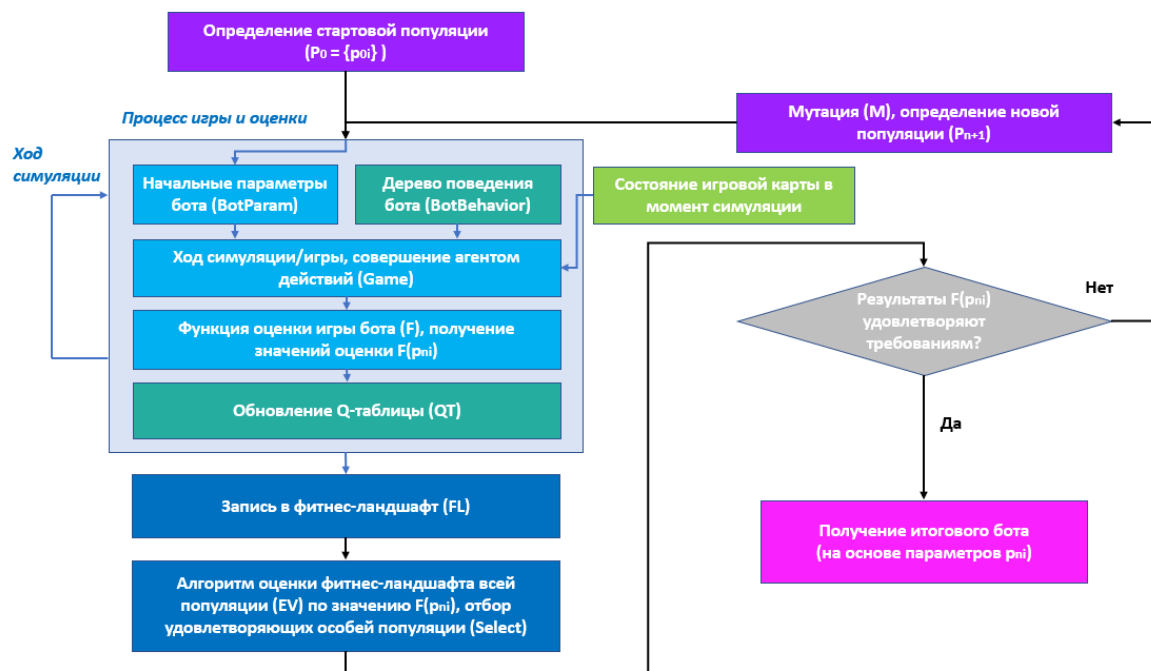


Рисунок 11 – Архитектура эволюционной модели GeneBot

В результате проведения заданного авторами числа игр была также получена численная статистика, показывающая эффективность ботов в заданном числе игр. Статистика представлена в таблице 1.

Таблица 2 – Результаты 100 матчей различных ботов с GoogleBot

	Number of Turns			Victories
	Average and Std. Dev.	Min	Max	
AresBot	217±157	49	1001	93
Best GeneBot	203±131	43	741	99
Average GeneBot	251±202	38	1001	91

Такой подход уменьшает эффект разброса и делает алгоритм более эффективным для создания ИИ, способного решать задачи данного типа.

3.2 Многоуровневая эволюционная модель Mario AI

3.2.1 Постановка задачи

В данном исследовании авторы рассматривали создание контроллеров ИИ для платформы Mario AI Benchmark с использованием системы Grammatical Evolution (GE) для развития структур дерева поведения (Behavior Tree или BT) [24].

Для анализа игровой среды Mario контроллер агента (персонажа Марио) может получить доступ к двум матрицам, одна из которых предоставляет

информацию о геометрии уровня, а другая указывает на наличие врагов и игровых объектов (рисунок 12).

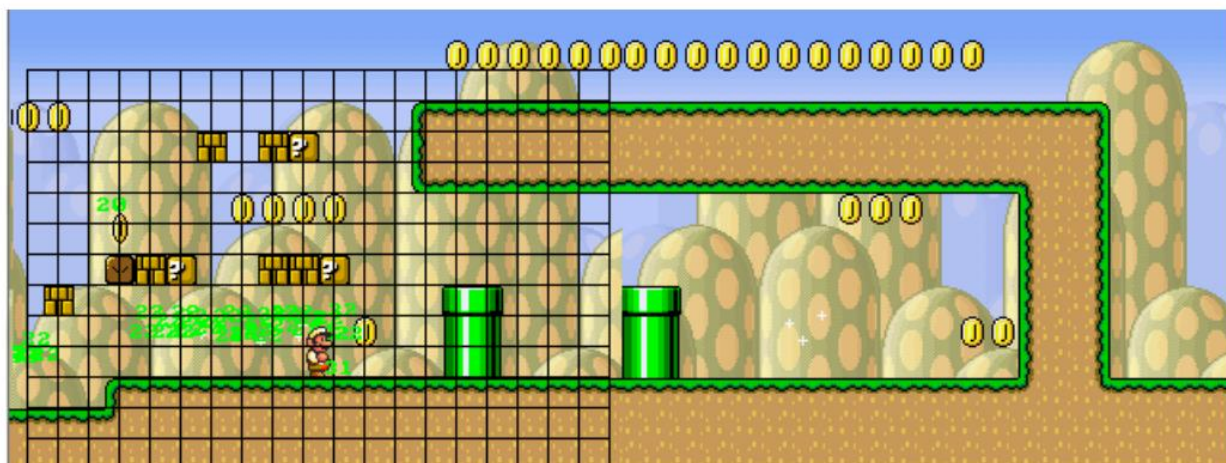


Рисунок 12 – Матрица информации об окружающей среде

Обе матрицы имеют размер 21×21 с центром в точке нахождения персонажа. Среда также предоставляет информацию о состоянии агента: его местоположение на уровне, его режим (Small, Big, Fire) и логические флаги, указывающие на дополнительную информацию, например, находится ли агент на земле, может ли он стрелять, прыгать, или наличие панциря черепахи. Помимо этого, доступна дополнительная информация, в том числе статус игры (running, won, lost), время, оставшееся до окончания игры, и статистика об убитых врагах. Эта информация используется контроллером для принятия решений во время игровой сессии.

3.2.2 Решение задачи

В центре внимания этой работы находится эволюция ВТ как контроллера для Mario AI. GE удалось объединить два необходимых аспекты поведения агента для этой игры: реактивность (работа с близкими врагами и опасностями) и навигация (определение путей перемещения по статическим элементам уровня). В обоих случаях использовались базовые игровые движения (влево, вправо, вниз, огонь) или их комбинации.

Для анализа важности навигационного компонента поведения алгоритма в этом исследовании были изучены два разных подхода, каждый из которых использовал свой набор процедур [24].

В первом, ReactiveMario (NoAstar), используется комбинация реактивности и (очень простых) навигационных процедур для развития ВТ.

Этот подход не использует явный поиск пути, агент сосредотачивается только на том, чтобы правильно реагировать на элементы игры.

Второй подход, PlanningMario (Astar), использует специальный алгоритм навигации (A*), что позволяет GE сосредоточиться в первую очередь на поведении реактивности.

Для использования алгоритма поиска пути, такого как A*, уровень должен быть представлен в виде графа с возможностью навигации, структура которого не предоставляется тестом. Таким образом, ответственность за динамическое создание этого графа на каждом шаге лежит на агенте.

Построение графа для этой игры сопряжено с рядом проблем. Во-первых, навигация по уровню может зависеть от состояния Марио (Маленький или Большой). Во-вторых, он должен иметь дело с асимметрией на краях. Поскольку игра ведется сбоку, горизонтальные и вертикальные края должны проходить по-разному (в отличие от вида сверху, зенитной перспективы). Для графа создаются следующие типы связей, которые также показаны в примере на рисунке 13.

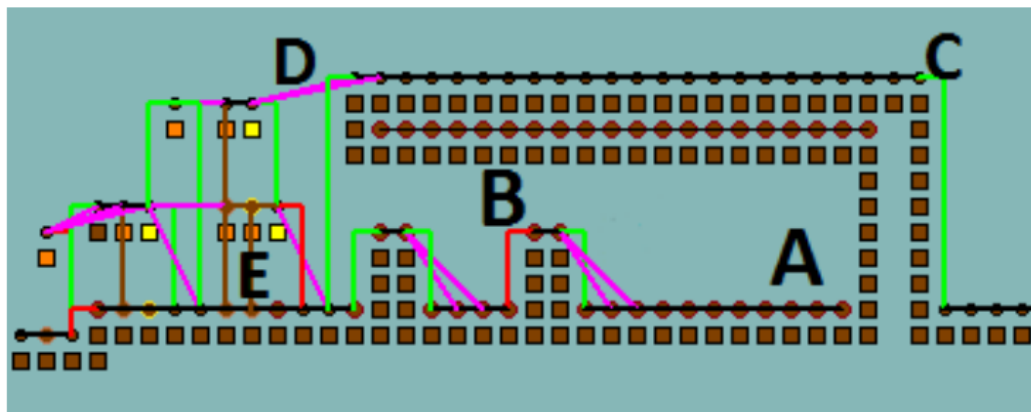


Рисунок 13 – Представление навигационного графа.

Представленные типы рёбер: A: Walk link. B: Jump link. C: Fall link. D: Faith jump link. E: Break jump link.

За выбор действий агента отвечает Behavior Tree (BT). Это структура, позволяющая организовать поведение в иерархическом порядке, разбивая изначально широкую задачу на несколько поддеревьев пониженной сложности [24]. Например, поведение игрового NPC можно разложить на различные подповедения, такие как патрулирование или нападение, вплоть до низкоуровневых действий для воспроизведения звуков или анимации.

Для BT авторами была использована следующая структура:

- 1) Корневой узел представляет собой селектор с переменным количеством поддеревьев блоков поведения (ВВ), кодирующих подповедения;
- 2) Каждый ВВ состоит из последовательности одного или нескольких условий, за которыми следует последовательность действий или поддеревьев;
- 3) Последний (необусловленный) ВВ, который представляет собой либо последовательность действий и поддеревьев, либо поведение навигации по умолчанию (при использовании A*).

Рисунок 14 иллюстрирует описанный синтаксис.

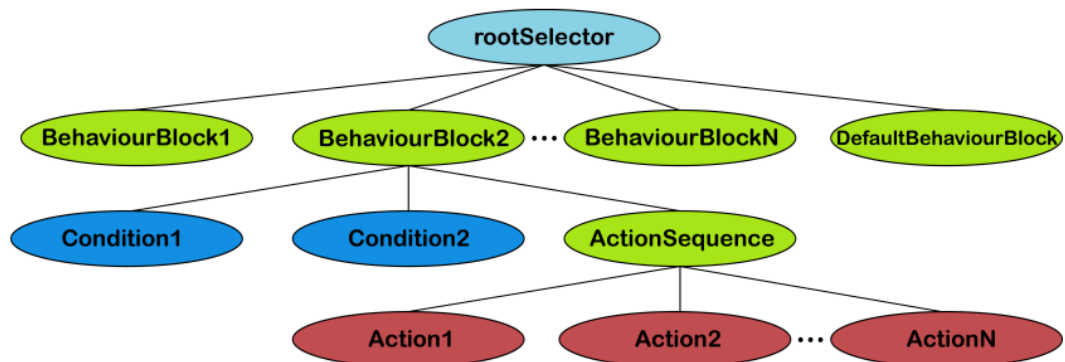


Рисунок 14 – Структура эволюционировавшего дерева поведения

В начале выполнения ВТ корневой селектор выбирает самый левый ВВ. Если связанные с ним условия не выполняются, выполнение следует в порядке приоритета слева направо. Поскольку предоставляемые условия являются сложными представлениями состояния игры, грамматика ограничивает количество связанных условий каждого ВВ до одного или двух, оставляя количество действий и поддеревьев неограниченным.

В случае Mario AI также кодировались поддерева для отдельных действий. На рисунке 15 представлено поддерево для планирования пути.

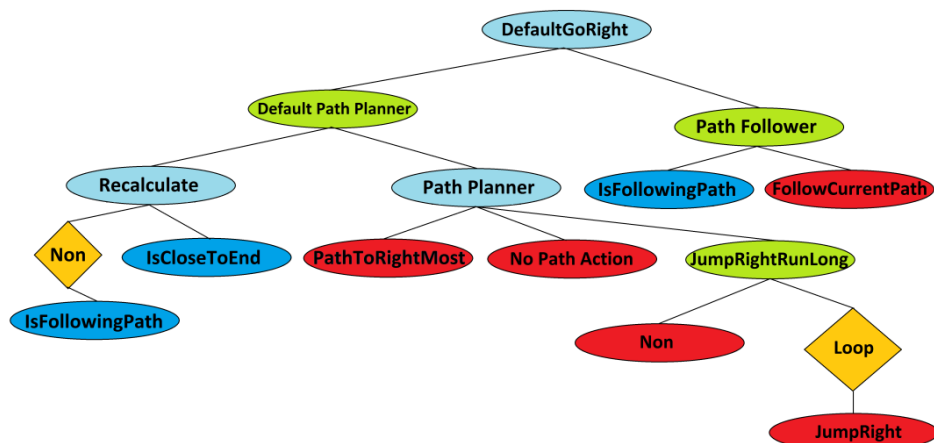


Рисунок 15 – Поддерево для планирования пути

Данная структура при необходимости вычисляет путь к самой правой доступной позиции (т.к. финиш находится в правой части уровня, и к нему должен стремиться агент).

3.2.3 Выводы об архитектуре

Архитектура эволюционной модели Mario AI представлена на рисунке 16:

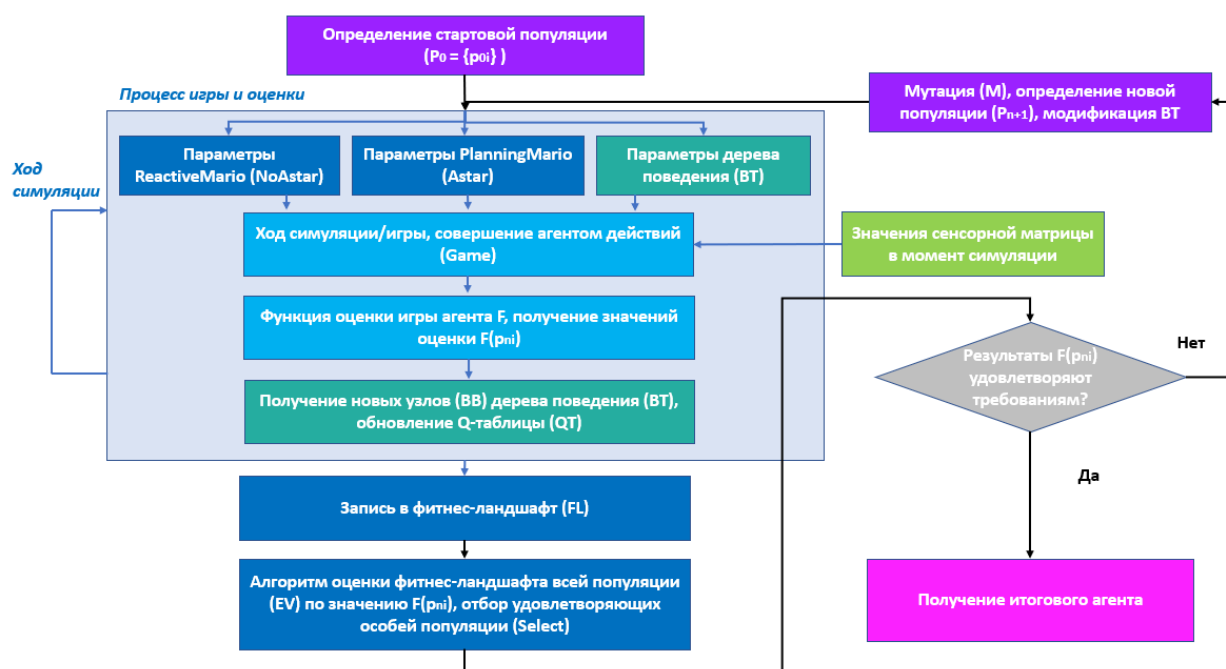


Рисунок 16 – Архитектура эволюционной модели Mario AI

Авторами [24] была проведена серия экспериментов, чтобы установить возможность развития BT в качестве контроллеров для Mario AI. BT были разработаны с использованием GE, а их обучение и результаты тестов контролировались с течением времени, наряду с другими статистическими

измерениями. Эти эксперименты также проверяли разделение процедур реактивности и навигации, а также то, приводит ли это к улучшению результатов. Наконец, был протестирован ряд различных подходов для работы с высокодинамичной средой, созданной тестом Mario AI Benchmark.

Для тестирования каждого усовершенствованного контроллера создается набор уровней Mario AI. Каждый набор карт состоит из 10 уровней (при этом присутствует 5 параметров, задающих сложность).

Для улучшения обобщения разработанных контроллеров, были протестированы пять различных подходов: Single, Five, Change1, Change 5 и Slide [24]. Они определяют порядок использования наборов карт для обучения контроллера. На рисунке 17 приведена оценка подходов.

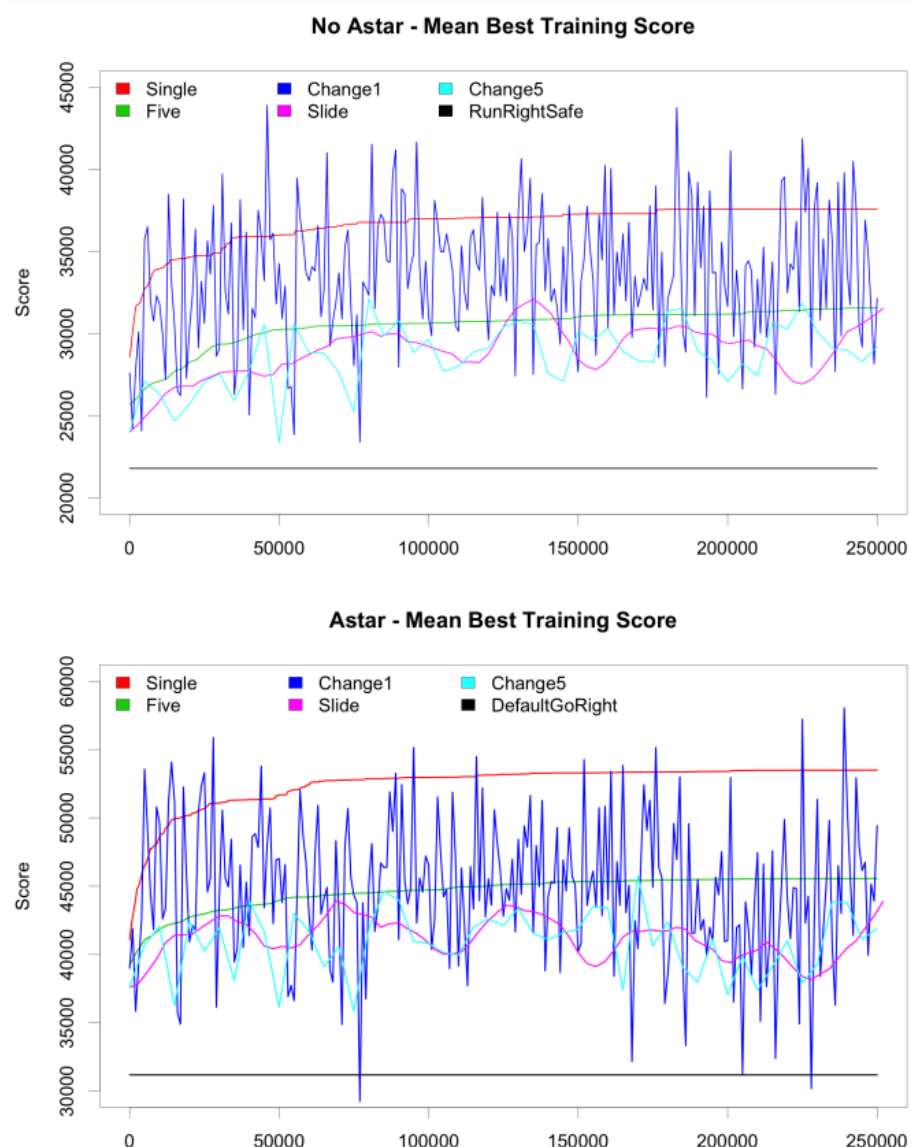


Рисунок 17 – Оценка подходов для NoAstar (вверху) и Astar (внизу)

Сочетание GE с VT обеспечивает гибкий подход к Mario AI. Полученные решения, по мнению авторов [24], удобочитаемы для человека, их легко анализировать и настраивать, что отвечает требованиям игровой индустрии относительно эволюционных подходов. Кроме того, это обеспечивает полный контроль над сложностью получаемых контроллеров VT, а также к играм, в которых желательна декомпозиция поведения [25]. Наконец, сочетание тщательно разработанного синтаксиса с определенными местами пересечения позволяет определять блоки поведения и обмениваться ими, ускоряя процесс эволюции.

4 Теоретическая модель

4.1 Постановка задачи

В рамках научной работы требуется создать эволюционную модель для обучения ИИ, который будет успешно выполнять свои игровые задачи, учитывая при этом текущее состояние локации. Данный критерий может быть важен в играх таких жанров, как стратегии, шутеры и спортивные симуляторы. Особенно важная роль ему отводится в играх с высокой степенью реалистичности, где к NPC предъявляются определенные стандарты внутриигрового поведения и отзывчивой реакции на действия игрока и окружающий мир.

Эволюционная модель будет считаться эффективной, если её создание и обучение требуемого ИИ будет более целесообразно, чем создание традиционными методами. В связи с этим, разрабатываемая модель будет учитывать в первую очередь те факторы, реакция на которые наиболее важна – характеристики состояния локации. Это должны быть критерии, которые агент ИИ сможет получать из игрового состояния посредством внутриигровых сенсоров либо прямой передачи из этого состояния.

Далее от агента, на основе получаемых данных, будет требоваться принять решение из числа доступных. Набор решений должен определяться доступными способами поведения и соотноситься с базовыми правилами игры выбранного жанра.

По результатам матча/отыгрыша и на основе принятых в процессе игры решений агент будет получать оценку. После этого, в зависимости от порядка эволюции, будет проводиться селекция наилучших особей, на основе которых с помощью мутации либо скрещивания будет получаться новое поколение для обучения.

4.2 Созданная архитектура

Исследованные примеры архитектуры эволюционных моделей позволили определить главные составляющие разрабатываемой модели.

Авторы [21] указали, что механизм принятия решений агента следует определить изначальным набором правил. Для его определения необходим набор констант, весов и вероятностей, которые влияют на эти правила и, следовательно, общее поведение агента. Данный блок будет реализован с учётом выбранного жанра и игровых задач агента.

Авторы [21] в том числе привели пример важности поэтапного тестирования агента в различных игровых условиях для проверки

однородности процесса обучения и выявления ошибок. Когда возможности обучения бота в различных условиях протестированы успешно, для обучения можно применить более изменчивую среду – например, случайно выбираемую для каждого матча локацию. Так обучение агента будет проходить более равномерно, и его реакция на игровые факторы будет отрегулирована однородно – не будет переобучения и неправильной, в рамках игровых требований, реакции.

Авторы [24] рассматривали возможность использования различных блоков архитектуры для разных задач: навигации и реакции на игровое состояние (противников, предметы и т.д.). Разделение данных функций эволюционной модели на отдельные блоки со своими параметрами позволило добиться от агента требуемой реакции на каждую из предоставляемых ему игровым процессом задач.

Авторы также указали важность равномерного обучения, которое может быть достигнуто с использованием методик рандомизации карт, а также уровня сложности (если таковой предполагается игровым процессом).

Объединяя подходы [21] и [24], для решаемой в рамках научной работы задачи может быть создана следующая архитектура:

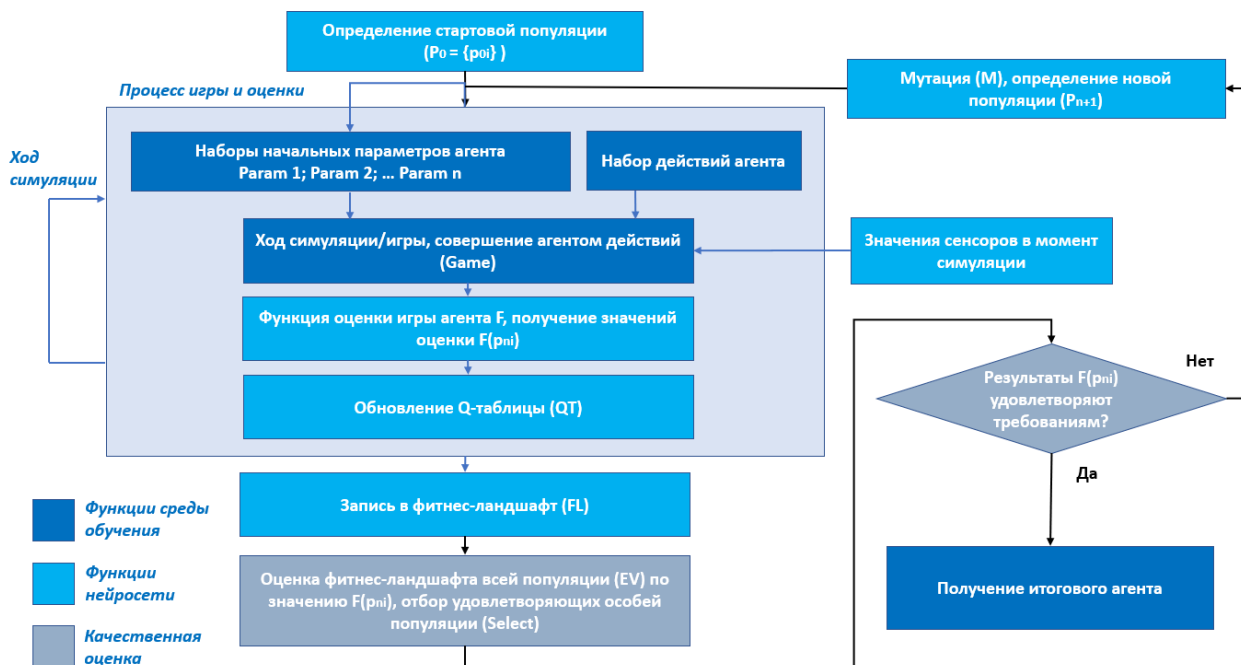


Рисунок 18 – Архитектура эволюционной модели ИИ, учитывающего текущее состояние локации

В структуре учтена возможность расширяемости параметров, задающих поведение агента – они могут добавляться в существующие блоки, либо добавляться отдельными наборами, блоки с которыми будут определять свои наборы действий и, следовательно, при создании новой популяции модифицироваться отдельно.

Для агента будет заранее определён набор поведения (вариантов действия). Это позволит однозначно определять возможность реакции NPC на то или иное игровое состояние и делать выбор действий для агента проще.

Во время проведения симуляции (игры) случайные действия агента будут приводить к получению им определенного вознаграждения. Выбор действий и значения сенсоров агента в данный момент симуляции будут записываться в Q-таблицу, в соответствии с принципами Q-learning. Поскольку большинство игр, соотносящиеся с темой научной работы, относятся к типу Real-Time, алгоритмы оценки, использующиеся при симуляции, будут придерживаться вида RHEA, с элементами NTBEA для оптимизации оценки. После определения необходимого числа вариативных симуляций (различные карты/уровни сложности) будет проведено первичное тестирование агентов на отдельных картах для калибровки алгоритмов обучения. После достижения соизмеримых результатов обучения для каждого вида симуляции будет проведено повторное обучение, но уже с комплексным подходом и вариативностью симуляций.

После завершения симуляции агент будет получать финальную оценку, а значение записываться в фитнес-ландшафт (функция пригодности). Поскольку игра проводится для N агентов (популяции), далее будет требоваться оценить полученный фитнес-ландшафт и провести отбор наилучших (удовлетворяющих) особей для создания на их основе следующей популяции.

Создание популяции будет производиться на основе параметров отобранных особей предыдущей популяции, с использованием алгоритмов скрещивания и, возможно, мутации – для расширения пространства поиска оптимальных параметров.

Такой вид архитектуры позволит тренировать различные типы агентов для игр различных жанров. Различные блоки параметров агента позволят отслеживать его обучение в рамках различных задач. Использование комплексного подхода при проведении симуляции позволит тренировать агента для игрового процесса на локациях в различных состояниях.

5 Обзор средств построения эволюционной модели

5.1 Используемые программные средства

Для создания эволюционной модели необходимы подходящие для реализации основных механизмов машинного обучения программные средства, библиотеки и фреймворки. Поскольку наибольшее количество решений в данной области предлагает Python [26], он был выбран в качестве языковой платформы для реализации модели.

Для данного языка существуют такие средства работы с механизмами машинного обучения, как PyTorch [27]. Фреймворк ориентирован на глубокое обучение, подвид машинного обучения, при котором используются многослойные обучаемые модели и нейронные сети. В глубоком обучении модель может обучаться за счет собственной обработки данных, и это позволяет решать больше разнообразных задач, чем в случае с классическим машинным обучением с узкоспециализированными моделями.

Для имплементации модели машинного обучения в игру необходима возможность её подключения и тренировки непосредственно на этапе создания игры. Для этого были проанализированы, в первую очередь, возможности подключения к доступным игровым движкам. В результате были обнаружены две реализации машинного обучения на популярных движках: Unity ML-Agents [28] для Unity, а также Epic Games Learning Agents [29] для Unreal Engine.

Unity ML-Agents – фреймворк, реализованный с помощью PyTorch и позволяющий обучать ИИ-агентов для различных видов игр [30]. Исследователи также могут использовать предоставляемый простой в использовании Python API для обучения агентов с использованием обучения с подкреплением, имитационного обучения, нейроэволюции или любых других методов [31].

Эти обученные агенты могут использоваться для множества целей, включая управление поведением NPC (в различных настройках, таких как мультиагентная и состязательная), автоматическое тестирование игровых сборок и оценку различных дизайн-решений игры перед выпуском. Инструментарий ML-Agents взаимовыгоден как для разработчиков игр, так и для исследователей искусственного интеллекта, поскольку он предоставляет центральную платформу, где достижения в области искусственного интеллекта могут быть оценены в сообществе пользователей Unity, а затем

доступны более широкому сообществу исследователей и разработчиков игр [28].

Epic Games Learning Agents представляет собой плагин машинного обучения для ИИ ботов, реализованный также с использованием PyTorch. Learning Agents позволяет обучать ИИ с помощью обучения с подкреплением и имитацией. Помимо создания агентов, плагин может использоваться для анимации на основе физики, автоматизированных QA-ботов и многого другого [29]. Learning Agents является сравнительно недавней разработкой Epic Games и продолжает оптимизироваться, в связи с чем исследований, связанных с ним меньше, чем посвященных Unity ML-Agents.

При обзоре доступных средств построения модели учитывались также возможности оптимизации и производительности игровой среды. Поскольку движок Unity имеет большие возможности для настройки и оптимизации, а его фреймворк – более стабилен, для построения эволюционной модели были выбраны Unity ML-Agents.

5.2 Unity ML-Agents

Данный инструментарий машинного обучения Unity может быть добавлен к существующему проекту Unity, позволяя создавать среду для обучения ИИ агентов. Это дополнение было создано как способ способствовать созданию новых сложных сред для исследования новых типов алгоритмов или самых современных алгоритмов, для тестирования этих алгоритмов в новых средах и для того, чтобы разработчики игр могли внедрять агенты, служащие ряду целей во время разработки [31].

Агент создается путем добавления к игровому объекту следующих скриптов: а) Agent, скрипт Unity для определения игрового объекта в качестве агента ML-Agents Toolkit; б) Behaviour Parameters, компонент агента, который функционирует как нейросеть для указанного агента; в) Decision Requester, сценарий связи для установления связи между Академией и агентом.

Академия (Academy) – это класс, структурированный в рамках инструментария ML-Agents [31]. Этот класс отвечает за несколько действий, например, за создание экземпляров агентов, отслеживание количества отмеченных шагов и количества эпизодов, запущенных с начала обучения.

На рисунке 19 представлена схема инструментария Unity ML-Agents.

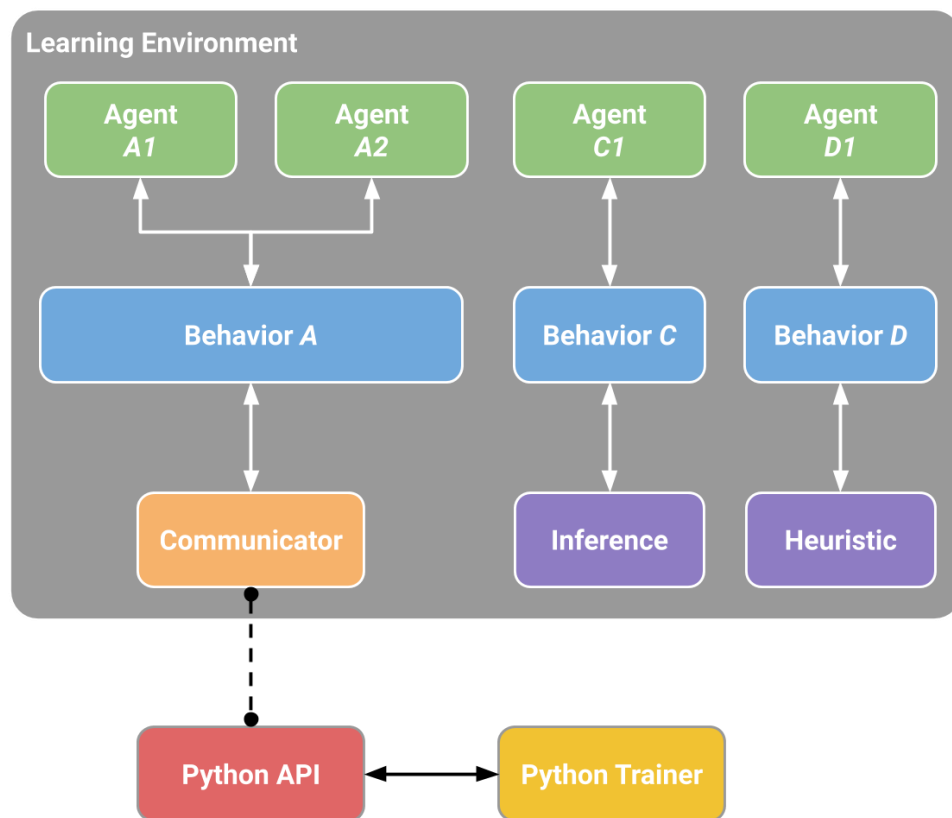


Рисунок 19 – Инструментарий Unity ML-Agents

Инструментарий ML-Agents обладает четырьмя основными элементами:

- 1) Среда обучения (Learning Environment) – сцена, созданная в Unity, где будет установлен агент. Здесь агент может проводить наблюдения за окружающей средой, выполнять действия, определяемые поведением (Behavior) и получать вознаграждения за выполнение конкретной задачи.
- 2) Низкоуровневый Python API. Внешний компонент, используемый для подключения обучающих тренажеров к агенту среды.
- 3) Коммуникатор (Communicator) – эта функция относится к среде обучения и обрабатывает связь между агентом и низкоуровневым Python API.
- 4) Python Trainer – внешняя функция, используемая для определения алгоритмов обучения.
- 5) Конечным продуктом, предоставляемым тренером, является модель нейронной сети (NN), также называемая поведенческой моделью или «мозгом», который управляет поведением агента [30].

Мозг агента находится в скрипте параметров поведения, прикрепленном к игровому объекту. Настраиваемые параметры приведены на рисунке 20. Функциональность мозга заключается в передаче наблюдений и

вознаграждений в API тренера и указании агенту, какое действие выполнить [32].

Behavior Parameters	
Behavior Name	PlayerBrain
Vector Observation	
Space Size	0
Stacked Vectors	1
Vector Action	
Space Type	Discrete
Branches Size	1
Branch 0 Size	4
Model	None (NN Model)
Inference Device	CPU
Behavior Type	Heuristic Only
Team Id	0
Use Child Sensors	<input type="checkbox"/>
Observable Attribute	Ignore

Рисунок 20 – Настраиваемые параметры «мозга» агента

- 1) Behavior Name – имя, данное набору гиперпараметров, используемых алгоритмом обучения;
- 2) Vector Observation – количество переменных, которые агент может наблюдать из состояния окружающей среды;
- 3) Vector Action – массив чисел с плавающей точкой или целых чисел, в зависимости от того, является ли это непрерывным или дискретным типом вектора. Используется агентом для выбора действий;
- 4) Model – соответствует тому, что NN использует мозг для определения применяемых действий. Может задаваться сохранённой моделью либо обучать агент с нуля при помощи пустой модели. Это также называется поведением агента.
- 5) Behaviour Type – мозг выберет, какое действие выполнить в соответствии с выбранным типом поведения. Существует три типа поведения.

Остальные параметры используются для более тонкой настройки обучения и сенсоров.

6 Создание эволюционной модели

6.1 Построение среды обучения

Для реализации эволюционной модели необходимо создать среду, в которой ИИ сможет выполнять игровые задачи, учитывая при этом текущее состояние локации. В качестве подходящего жанра были выбраны «гонки», поскольку в них зачастую присутствует движение по заданному пространству (треку), и во многих случаях требуется учитывать состояние поверхности дороги, которое может влиять, например, на физику движения (реализация движения автомобилей в сериях игр Grand Theft Auto, Need for Speed т.д.).

Среда обучения представляет собой сцену, созданную в Unity. Пример сцены представлен на рисунках 21 и 22.

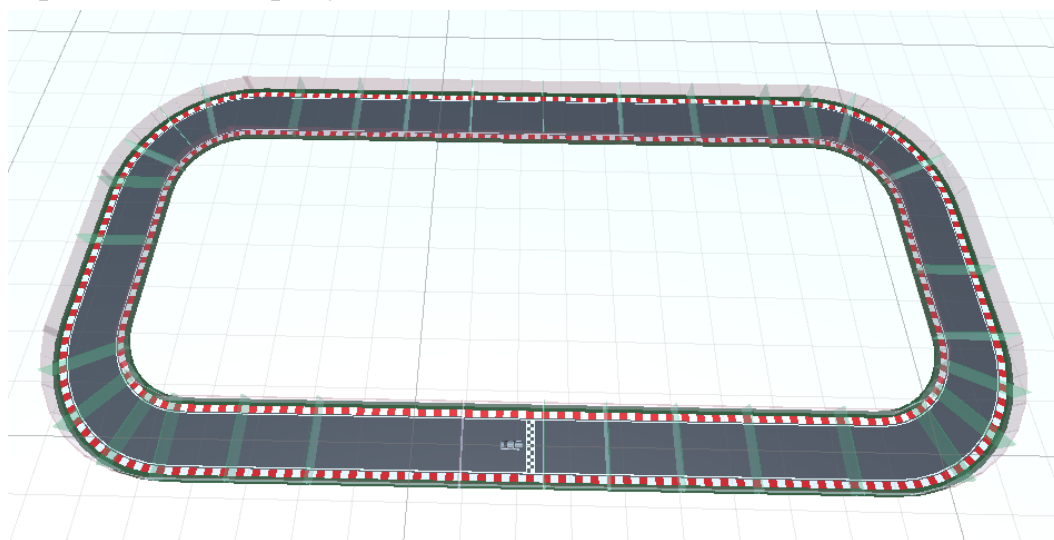


Рисунок 21 – Среда обучения ИИ (трек целиком)

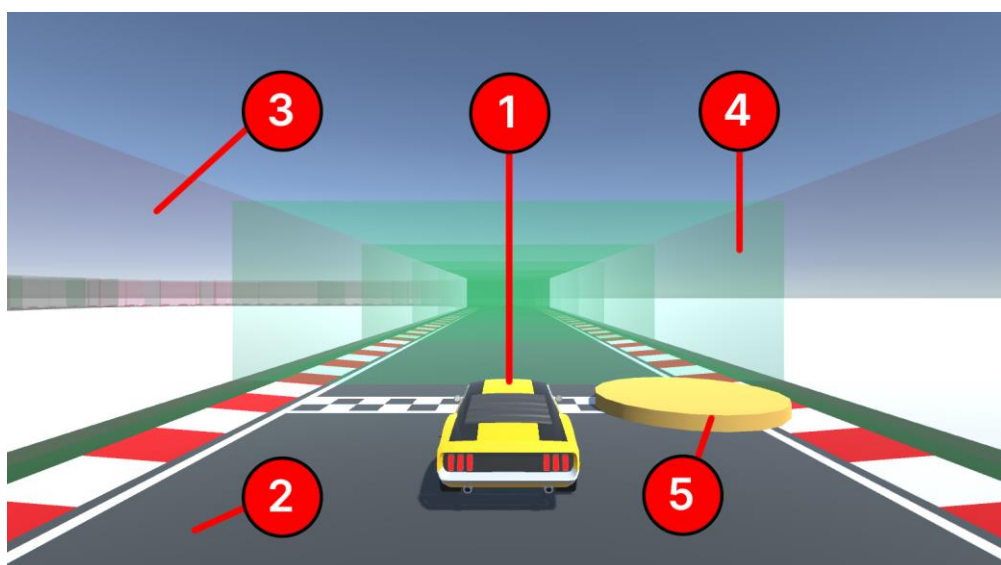


Рисунок 22 – Среда обучения ИИ (трек вблизи)

На игровой карте присутствуют следующие элементы:

- 1) Агент – модель машины с контроллером, который использует для движения физику встроенного компонента WheelCollider; обладает двумя уровнями сенсоров: для отслеживания границ трека и для отслеживания типа поверхности (рисунки 23 и 24);

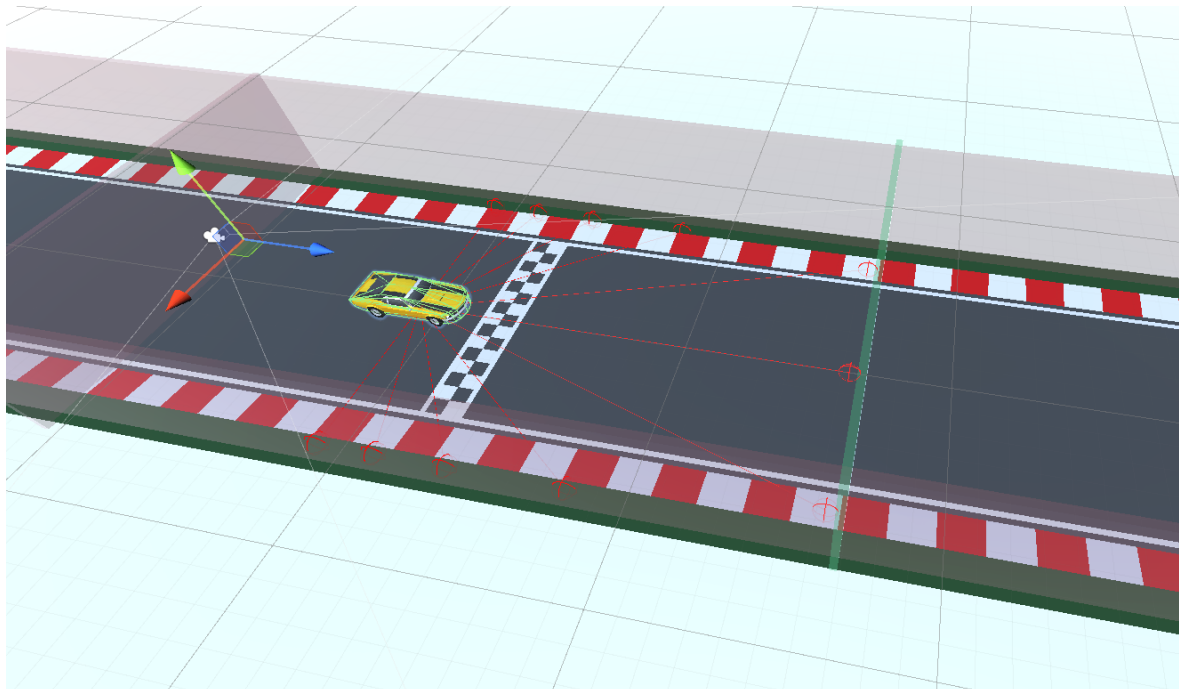


Рисунок 23 – Сенсоры для отслеживания границ трека

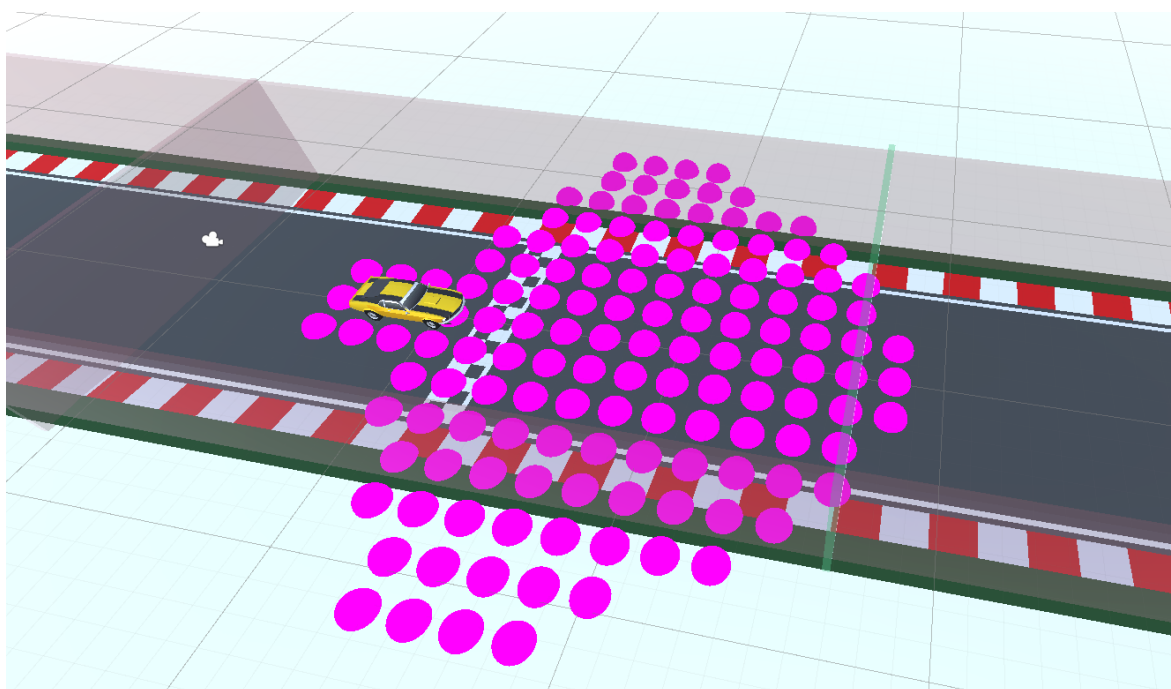


Рисунок 24 – Сенсоры для отслеживания типа поверхности

- 2) Трек – модульная структура дороги, используемая для визуализации траектории движения;
- 3) Стены – объекты, ограничивающие движение агента по треку; при касании агентом передают ему отрицательные значения вознаграждения; необходимы для обучения агента не выезжать за границы трека; при касании агентом стены текущий эпизод обучения завершается, и запускается следующий;
- 4) Чекпоинты – объекты, используемые для контроля движения агента по треку; при касании передают агенту положительные значения вознаграждения; необходимы для обучения агента движению вперед по треку;
- 5) Зоны отличающейся поверхности трека – зоны поверхности трека, отличающиеся от стандартной дороги и влияющие на движение агента; на данном этапе реализованы зоны песчаной поверхности, замедляющие агент; при попадании в подобные зоны агенту передаются отрицательные значения вознаграждения; данные зоны необходимы для соответствия задаче исследования и учета при создании ИИ текущего состояния локации.

6.2 Настройка параметров агента

Агент управляет контроллером при помощи компонента Decision Requester. Контроллер может выполнять три вида действий: движение вперед либо назад, поворот влево или вправо, торможение.

Для того, чтобы агент имел возможность выбора вариантов для каждого вида действий, необходимо указать количество переменных Discrete Branches, равное количеству видов действий. Каждый шаг Decision Requester присваивает переменной значение в диапазоне $[-1.0; 1.0]$. В зависимости от значения переменной выбирается конкретное действие, соответствие приведено в таблице 3.

Таблица 3 – Зависимость действий контроллера от Discrete Branches

Переменная	Discrete Branches [0]		Discrete Branches [1]		Discrete Branches [2]	
Действие	Вперёд	Назад	Вправо	Влево	Торможение	–
Значение	1.0	-1.0	1.0	-1.0	1.0	$[-1.0; 0.0]$

Для выбора действий агент использует значения переменных из Vector Observation. Количество дополнительных переменных определяется значением поля Space Size. На данном этапе агент учитывает скорость контроллера, положение forward-вектора ближайшего Чекпоинта, а также

значения переменных 140 окружающих поверхностных сенсоров. В дополнение к ним агент рассматривает значения пространственных сенсоров, подключаемых к Vector Observation автоматически.

Параметр Stacked Vectors определяет сколько наборов векторов объединяется для передачи нейросети. На начальном этапе исследования выбрано минимальное значение, чтобы исключить неправильное обучение.

На начальном этапе обучения используется пустая модель. Далее, при необходимости, может быть выбрана уже созданная модель, чтобы продолжить её обучение. Такой подход может быть использован при реализации селективной составляющей эволюционной модели. Параметры на момент начала обучения представлены на рисунке 25.

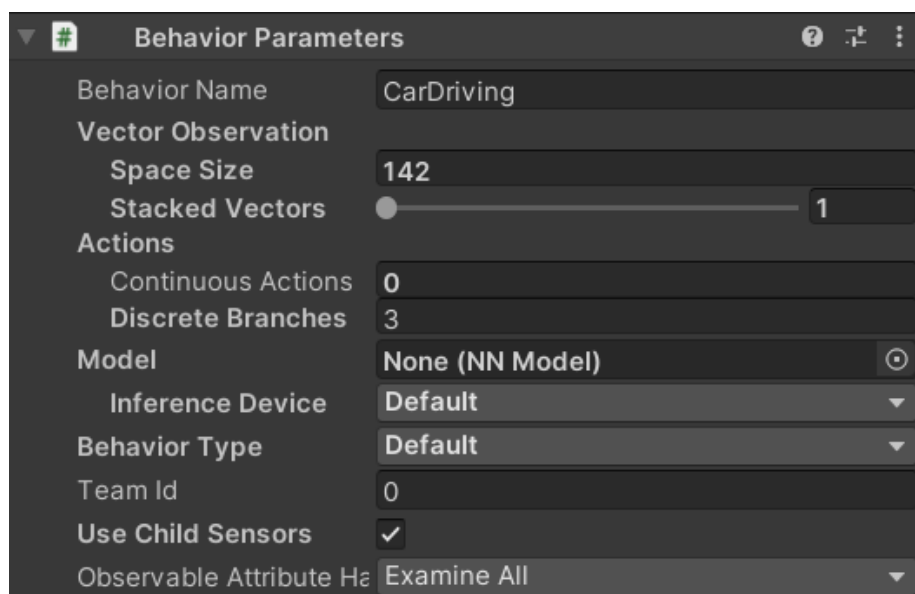


Рисунок 25 – Параметры агента на момент начала обучения

7 Тестирование эволюционной модели

7.1 Обучение агента

При помощи эволюционной модели должен быть получен ИИ агента, способный перемещаться по треку в необходимом направлении, не съезжая с нее, при этом поддерживать наибольшую возможную скорость и, соответственно, затрачивать наименьшее время на прохождение трека целиком.

Обучение агента на этапе тестирования возможностей эволюционной модели проходило поэтапно. Поскольку в среде обучения присутствуют объекты (стены), при взаимодействии с которыми агент получает отрицательной вознаграждение, на начальном этапе было принято решение отключить их функции, оставив только физические свойства – коллайдер, блокирующий выход агента за границы трека.

На треке в определенной последовательности расположены чекпоинты. За их нумерацию отвечает Game Object «Track Manager», использующий соответствующий скрипт. При старте симуляции чекпоинты заносятся в список. Первый чекпоинт по умолчанию записывается в переменную агента `nextCheckpoint`, в которой хранится ссылка на чекпоинт, как на объект. При контакте с чекпоинтом, отмеченным как `nextCheckpoint`, агент получает вознаграждение. После этого ссылка обновляется: теперь `nextCheckpoint` считается следующий чекпоинт в списке. Когда агент доходит до конца трека, значение обновляется на стартовое, и круг может быть пройден заново. В случае использования нескольких экземпляров агентов Track Manager учитывает свой набор чекпоинтов и `nextCheckpoint` для каждого агента. Также агент получал вознаграждение при достижении скорости движения, большей, чем определенное значение, каждую секунду.

Используемые в первом этапе обучения способы получения вознаграждений представлены в таблице 4.

Таблица 4 – Вознаграждения в первом этапе обучения

Вид действия	Касание агентом коллайдера чекпоинта	Достижение агентом скорости движения > 10
Значение вознаграждения	1.0	0.01*скорость

При настройке первого этапа обучения целью была возможность агента определить, что положительное вознаграждение он получает при движении по

треку в заданном направлении. При этом не должно было учитываться касание стен, чтобы исключить переобучение на данном этапе.

Во втором этапе на старте обучения использовалась нейросеть, полученная на первом этапе. Были возвращены свойства стен, и при касании с ними агент получал отрицательное значение вознаграждения. Также была изменена формула вознаграждения, получаемого при касании агентом коллайдера чекпоинта. В данном этапе требовалось обучить агента избегать касания со стенами, а также стараться достичь большей скорости при касании чекпоинта. Используемый во втором этапе обучения набор способов получения вознаграждений представлен в таблице 5.

Таблица 5 – Вознаграждения в первом этапе обучения

Вид действия	Касание агентом коллайдера чекпоинта	Достижение агентом скорости движения > 10	Касание агентом коллайдера стены
Значение вознаграждения	$1.0 + (\text{скорость}/100)$	$0.01 * \text{скорость}$	-0.1

В завершение первичного тестирования возможностей эволюционной модели был проведен третий этап обучения, на старте которого использовалась нейросеть, полученная на втором этапе. Система вознаграждения не изменялась. В данном этапе планировалось оценить первичную эволюционную модель и среду тестирования. По результатам оценки сделаны выводы о дальнейшей работе и требуемых улучшениях модели.

7.2 Первичные результаты обучения

В процессе тестирования происходила запись данных среды тестирования, предоставляемых ML-Agents. Были получены значения для трёх этапов обучения. Интерес представляют два параметра: Cumulative Reward, связанный непосредственно с успешностью выполнения агентом своих задач. Также для отслеживания поведения агента может быть приведен параметр Episode Length, дающий представление о частоте проигрышей агента.

Значения Cumulative Reward для трех этапов представлены на рисунке 26.

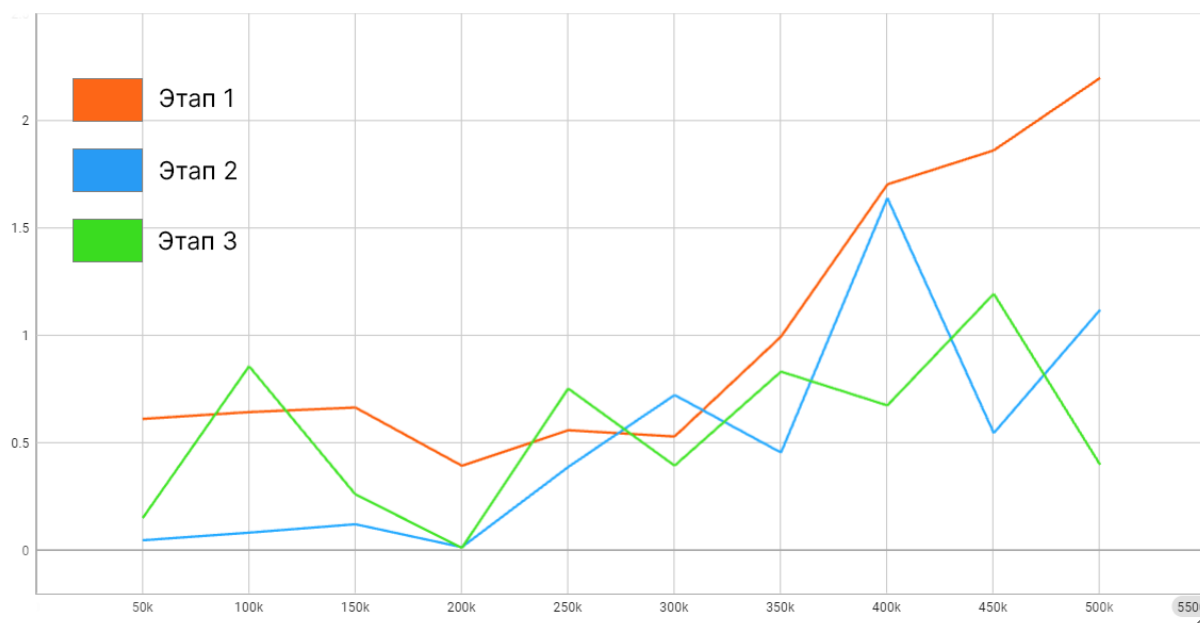


Рисунок 26 – Параметр Cumulative Reward

Каждый из этапов обучения длился 500.000 шагов. Данные собираются с начала симуляции и формируют значение на 50.000 шаге. В первом этапе проводилось обучение движению по треку без отрицательного вознаграждения. Во втором этапе была включена функция отрицательного вознаграждения при касании стен трека. В третьем этапе условия обучения сохранились. Агент показал тенденцию к прогрессии: полученные на третьем этапе значения Cumulative Reward – выше, чем на втором.

Параметр Episode Length позволяет оценить, насколько успешно агент избегает проигрыша. Агенты были ограничены эпизодами длиной в 3000 шагов, после чего текущий эпизод для них завершался, и запускался новый. В первом этапе не было способов завершить эпизод раньше, поэтому на протяжении всего этапа значение сохранялось неизменным.

На втором этапе можно отметить значительное снижение длины эпизода в начале обучения и повышение значений в конце обучения. Третий этап показывал схожие значения Episode Length, которые при сглаживании демонстрируют незначительные улучшения. Соответствующий график представлен на рисунке 27.

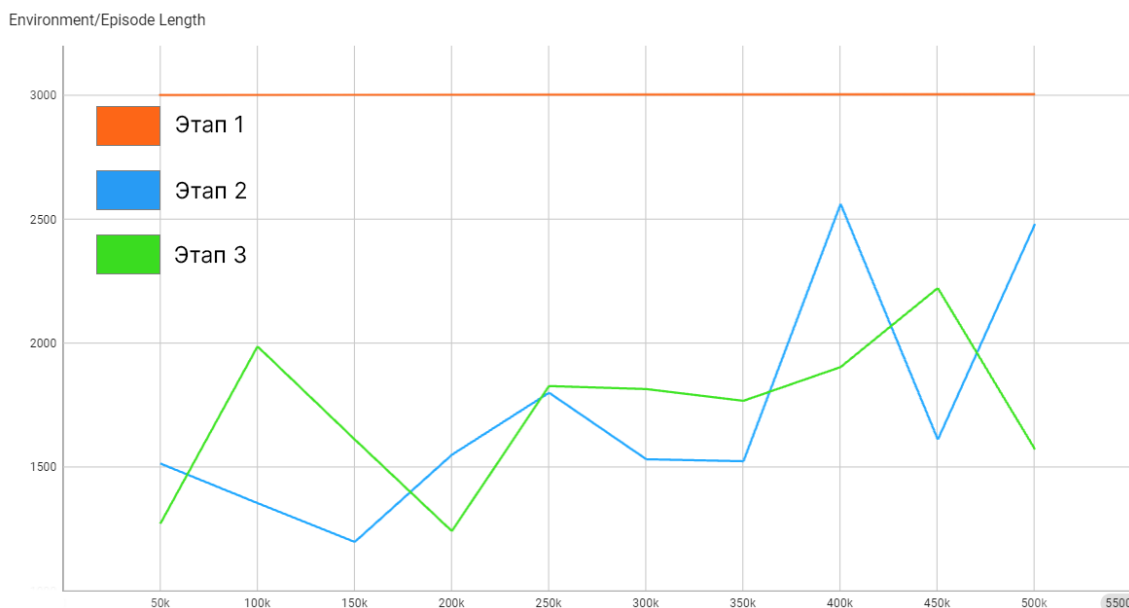


Рисунок 27 – Параметр Episode Length

По представленным выше данным можно было бы заключить, что агенты успешно выполняют задачи обучения. Однако данные первичные показатели – низкие для выбранной среды. Чтобы получить ИИ, поведение которого будет приближено к требуемому в рамках задач работы, необходимо оптимизировать модель. Для этого было принято решение использовать ещё один инструмент Unity ML-Agents – Imitation Learning [33].

8 Оптимизация эволюционной модели

8.1 Использование Imitation Learning

Imitation Learning – методика машинного обучения, называемая также имитационным обучением. Она представляет собой обучение агента при помощи ранее записанной демонстрации – готового соотношения действий и вознаграждений. С её помощью агент ML-Agents способен проанализировать записанные действия, соотнести с получаемым вознаграждением и получить готовую Q-таблицу, используемую в дальнейшем обучении [34].

Для имплементации инструментария Imitation Learning был изменён агент. К нему был добавлен компонент Demonstration Recorder, который позволил записывать демонстрации действий. Действия представляли собой ручной контроль передвижения агента и прохождение трека по заданной траектории. Запись состоит из серии эпизодов, что позволяет агенту делать оценку действий более равномерной.

8.2 Модификация среды обучения

Для оптимизации процесса обучения агентов была проведена модификация среды обучения. В ходе первичного тестирования возможностей Imitation learning было выявлено, что агент отдаёт предпочтение прохождению чекпоинтов, но при этом не учитывает траекторию движения. Это зачастую приводило к столкновениям с границей трека и, как следствие, ограничению длительности эпизодов обучения.

В связи с этим была изменена механика вознаграждения агента. Был добавлен компонент Vector Observation, отвечающий за пространственную ориентацию агента относительно следующего чекпоинта, представляющий собой скалярное произведение вектора направления агента на вектор направления чекпоинта. Чем ближе эта переменная была к значению 1, тем большее вознаграждение получал агент. Соответствующая формула представлена ниже:

$$R = 1 \times (a, b), \quad (5)$$

где a – forward-вектор агента (машины),

b – forward-вектор чекпоинта,

R – значение вознаграждения, получаемого агентом при прохождении чекпоинта.

Также был добавлен аналогичный компонент для второго следующего чекпоинта – это было сделано для выявления агентом изменения в направлении последующего движения и определения поворота трека.

После применения модификаций было проведено два этапа обучения. В первом этапе обучения использовались только сенсоры для отслеживания границ трека, что позволило определить эффективность новых методов оценки. Во второй серии обучения были добавлены сенсоры для отслеживания типа поверхности. Было выявлено, что первичный вид сенсоров поверхности в виде пространственной матрицы неэффективен, поскольку в своём изначальном виде занимает значительную часть Vector Observation и приводит к недостаточно эффективному обучению. В связи с этим, после серии тестов он был заменён на аналогичную пространственному сенсору группу лучей, исходящих из агента и реагирующих на пересечение с зонами отличающихся типов поверхности (рисунок 28).

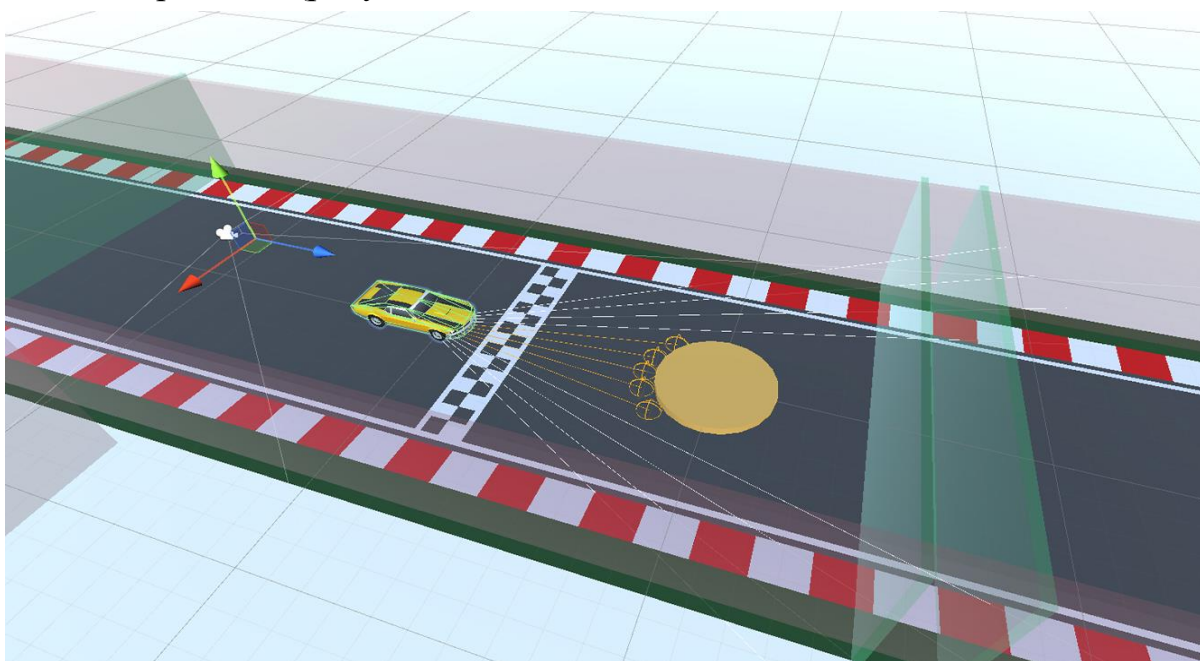


Рисунок 28 – Новые сенсоры для отслеживания типа поверхности

Такое изменение уменьшило количество единиц Vector Observation и позволило агенту обучаться эффективнее.

8.3 Повторное обучение

8.3.1 Параметры Imitation Learning

В процессе повторного обучения были протестирована работа инструментария Imitation Learning, а также проведена модификация среды обучения, в соответствии с обновлёнными требованиями агентов и решениями

по улучшению работы модели. Также происходила запись данных среды тестирования, предоставляемых ML-Agents.

При обучении агентов при помощи средств Imitation Learning требуется дополнительная конфигурация, определяющее влияние записанной ранее демонстрации на работу ML-Agents. Данная конфигурация представляет собой файл с набором параметров обучения. Подробное описание параметров приводится в [35].

В рамках же данной работы можно выделить основные параметры, наиболее влияющие на обучение агента при использовании Imitation Learning, они будут описаны далее.

Полная конфигурация параметров ML-Agents в первом этапе обучения приведена в таблице 6.

Таблица 6 – Конфигурация ML-Agents в первом этапе обучения

Название группы	Название параметра	Значение параметра
1	2	3
-	trainer_type	ppo
hyperparameters	batch_size	256
	buffer_size	10240
	learning_rate	3.0e-4
	beta	5.0e-4
	epsilon	0.2
	lambd	0.99
	num_epoch	3
	learning_rate_schedule	linear
network_settings	normalize	false
	hidden_units	128
	num_layers	2
reward_signal (extrinsic)	gamma	0.99
	strength	1.0
reward_signal (gail)	gamma	0.99
	strength	0.5
	use_actions	true
	use_vail	false

1	2	3
behaviorial_cloning	gamma	0.99
	strength	0.5
-	max_steps	100000
-	time_horizon	64
-	summary_freq	10000

В качестве алгоритма обучения в Unity ML-Agents по умолчанию применяется PPO (Proximal Policy Optimization). Данный алгоритм придерживается обучения стратегии с учетом границ для обновлений. PPO часто используется при обучении, поскольку обеспечивает стабильность и эффективно работает в различных средах. Он также широко используется при создании ИИ в играх, в связи с чем параметр `trainer_type` был сохранён.

Наиболее значимыми в рамках работы параметрами являются переменные, определяющие свободу обучения нейросети, а также влияние на него демонстрационной записи. К таким параметрам относятся две группы `reward_signal`, а также группа `behaviorial_cloning`.

Группа `reward_signal` (excintric) определяет степень приоретизации агентом получаемых сигналов о внешнем вознаграждении среды – тех значений, которые агент воспринимает и обрабатывает при симуляции. Параметр `strength` является коэффициентом, на который умножается вознаграждение, выдаваемое средой. Среднее значение параметра – 1.0, поскольку необходимо, чтобы агент получал актуальные данные о вознаграждении от среды.

Параметр `gamma` отвечает за коэффициент скидки для будущих вознаграждений, поступающих из среды. Он необходим для стимуляции агента к повышению получаемых значений вознаграждений с каждой последующей симуляцией. Поскольку основной задачей является обучение агента проходить максимальную дистанцию по треку за отведенное время, параметр имеет высокое значение (типичный диапазон `gamma` лежит в пределах 0.8 - 0.995).

Группа `reward_signal` (gail) представляет собой обучение при помощи GAIL (Generative Adversarial Imitation Learning)[34] отвечает за вознаграждение, получаемое агентом при выполнении действий, похожих на действия из набора, представленного в демонстрации.

GAIL может использоваться как с вознаграждениями за среду, так и без них, и хорошо работает при ограниченном количестве демонстраций. В этой структуре вторая нейронная сеть, дискриминатор, обучается различать действия в демонстрации и действия, произведённые агентом самостоятельно. Затем этот дискриминатор может изучить новое действие и назначить за него вознаграждение в зависимости от того, насколько, по его мнению, это новое действие соответствует предоставленным демонстрациям.

Здесь также присутствуют параметры `gamma` и `strength`. Параметр `strength` изменён, поскольку, согласно документации ML-Agents, для эффективного обучения GAIL должно влиять на агента в меньшей степени, чем внешнее вознаграждение. Параметр `use_actions` в значении `true` означает, что агент будет выполнять действия из демонстрации в точности. Параметр `use_vail` по умолчанию имеет значение `false`, и активируется только если результаты обучения имеют нестабильный характер.

Группа `behavioral_cloning` определяет точное копирование поведения агента при демонстрации. Параметр `strength` определяет степень восприятия значений вознаграждения при демонстрации, и, по аналогии с GAIL, выставлен на меньшее значение, чем для внешнего вознаграждения.

8.3.2 Обучение с сенсорами границ трека

Длительность первого этапа обучения составляла 100000 шагов, при условии, что каждый тренировочный раунд длился 5000 шагов. В данном этапе проходила проверка возможностей обучения при помощи Imitation Learning, поэтому велось обучение ИИ первостепенной задаче – передвижению в границах трека и преодоление наибольшего количества чекпоинтов за тренировочный раунд. Для ускорения обучения в Vector Observation учитывался только один тип сенсоров – сенсоры для отслеживания границ трека.

В первом этапе динамика обучения оказалась удовлетворительной. Агент продвигался по треку, избегал столкновений со стенами, а также отслеживал положение чекпоинтов. Агент воспринял значение получаемого, согласно формуле (5), вознаграждения, и по мере прохождения трека располагался практически перпендикулярно к каждому следующему чекпоинту, если тот оказывался на близком для реакции расстоянии. При этом агент сохранял среднюю скорость движения, что означает необходимость введения дополнительного стимула к ускорению в последующих этапах обучения при помощи данной модели.

Во втором этапе обучения использовалась нейросеть, полученная на первом этапе. Приоритет `behavioral_cloning` был уменьшен до 0,1. Таким образом мы пытаемся добиться от агента более самостоятельного обучения и реакции на конкретные игровые ситуации. Дополнительно была изменена формула получения вознаграждения:

$$R = \begin{cases} 1 \times (a, b) + 0.01 \times v & | v > 30, \\ 1 \times (a, b) & | v \leq 30 \end{cases}, \quad (6)$$

где a – forward-вектор агента (машины),

b – forward-вектор чекпоинта,

v – скорость движения агента (машины) в условных единицах,

R – значение вознаграждения, получаемого агентом при прохождении чекпоинта.

Теперь значение вознаграждения, получаемого при прохождении чекпоинта, зависит от скорости движения агента. Такое решение должно было побуждать агента двигаться по треку быстрее и приближаться к оптимальному для ИИ поведению. В начале оно действительно заставило его двигаться быстрее и, как следствие, получать большие значения Cumulative Reward. Однако затем произошло переобучение, и агент стал отдавать больший приоритет прохождению отдельных чекпоинтов на большой скорости, чем качественному прохождению трека.

Динамика обучения первого и второго этапов представлена в виде графиков параметров Cumulative Reward и Episode Length на рисунках 29 и 30:

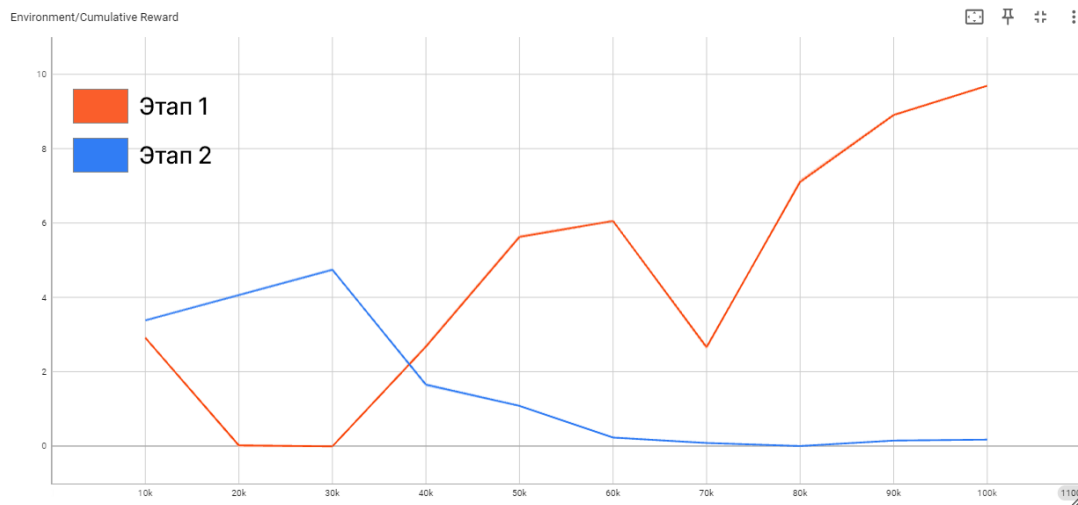


Рисунок 29 – Параметр Cumulative Reward в 1 и 2 этапах

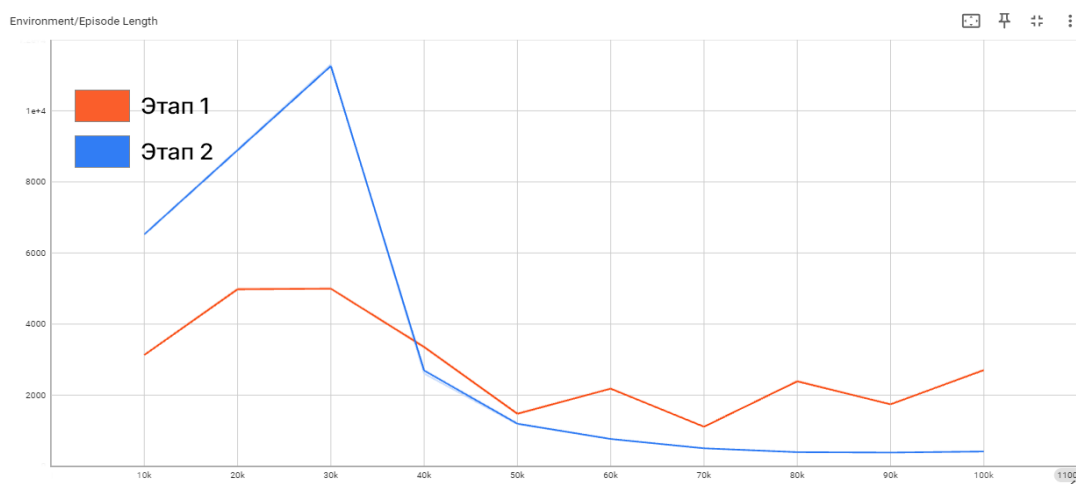


Рисунок 30 – Параметр Episode Length в 1 и 2 этапах

Первый и второй этапы обучения позволили заключить, что при использовании Imitation Learning возможно создать ИИ, способный к прохождению трека и выполнению своих игровых задач. Однако тестирование показало, что улучшение параметров ИИ при помощи дополнительной стимуляции требует тщательного выбора механизма такой стимуляции, и не должно приводить к переобучению.

8.3.3 Обучение с сенсорами поверхности

После тестирования нового метода обучения было необходимо ввести второй вид сенсоров – сенсоры для отслеживания типа поверхности, а после этого провести повторное обучение модели с нуля. Это обусловлено ограничениями, накладываемыми инструментарием ML-Agents: если агенту добавляются дополнительные элементы Vector Observation, необходимо, чтобы готовая нейросеть, используемая для последующего обучения, была получена при помощи анализа такого же количества компонентов. То же касается и записи демонстраций. Поэтому после добавления нового вида сенсоров была записана новая демонстрация, которая использовалась в последующем обучении с нуля. В данных условиях было проведено ещё два этапа обучения: третий и четвёртый.

В третьем этапе обучения были использованы оба вида сенсоров: агент учитывал и значения сенсоров границ трека, и значения сенсоров типа поверхности. Также на протяжении всего трека были расставлены зоны отличающегося типа поверхности («песок»). При попадании в них агент получает отрицательные значения вознаграждения. При помощи сенсоров

поверхности агент должен обучиться избегать подобных зон по мере прохождения трека. Пример нового трека представлен на рисунке 31:

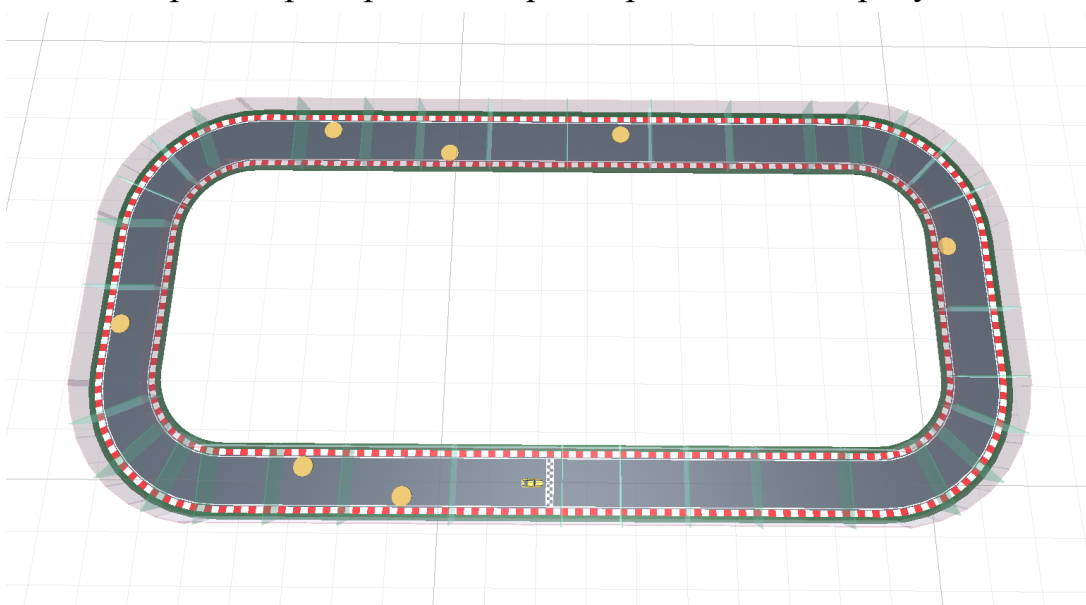


Рисунок 31 – Пример трека для обучения с сенсорами поверхности

В третьем этапе обучения значение `behavioral_cloning` было таким же, как и в первом этапе – 0,5. Остальные исходные параметры конфигурации обучения для первого и третьего этапа также были одинаковыми. Для расчёта вознаграждения использовалась формула (5).

Динамика обучения была сравнительно ниже, чем в первом этапе, что обусловлено новой средой обучения, а также необходимостью анализировать большее количество `Vector Observation`. Агент успешно преодолевал трек, но на меньшей скорости, чем делал это в первом этапе. Агент проявлял реакцию на зоны «песка», пытаясь изменить свою траекторию движения, что достаточно часто также выражалось в снижении скорости. Вероятно, это обусловлено конфликтом стремления агента найти оптимальную траекторию при движении к следующему чекпоинту и необходимости обойти зону «песка».

В четвёртом этапе обучения использовалась нейросеть, полученная на третьем этапе. Приоритет `behavioral_cloning` был также уменьшен до 0,1. Для расчёта вознаграждения использовалась формула (6). Агент показал незначительное снижение скорости передвижения. Динамика обучения третьего и четвёртого этапов представлена в виде графиков параметров `Cumulative Reward` и `Episode Length` на рисунках 32 и 33:

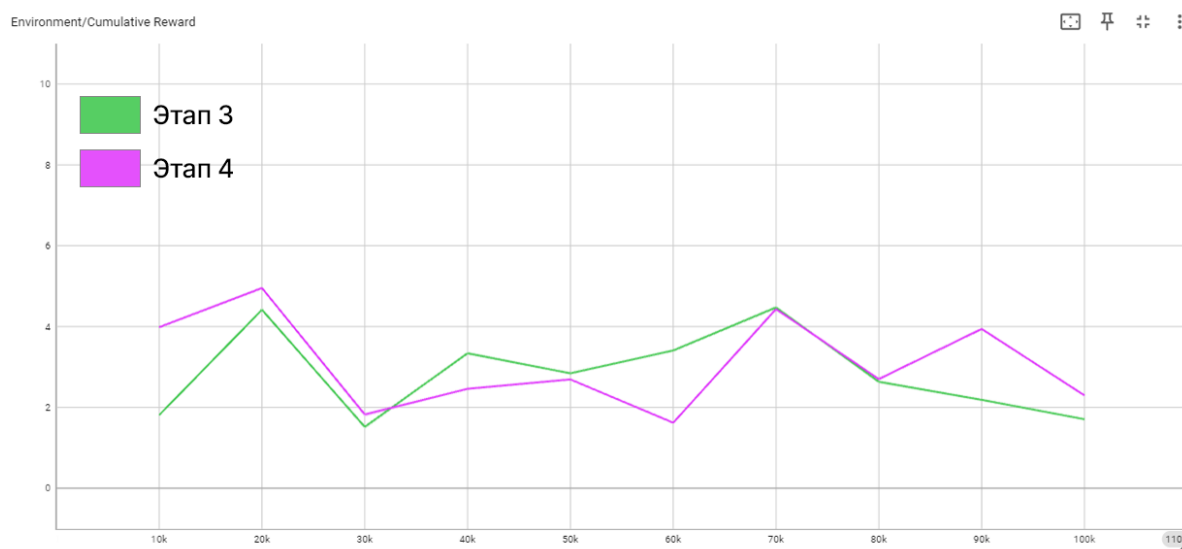


Рисунок 32 – Параметр Cumulative Reward в 3 и 4 этапах

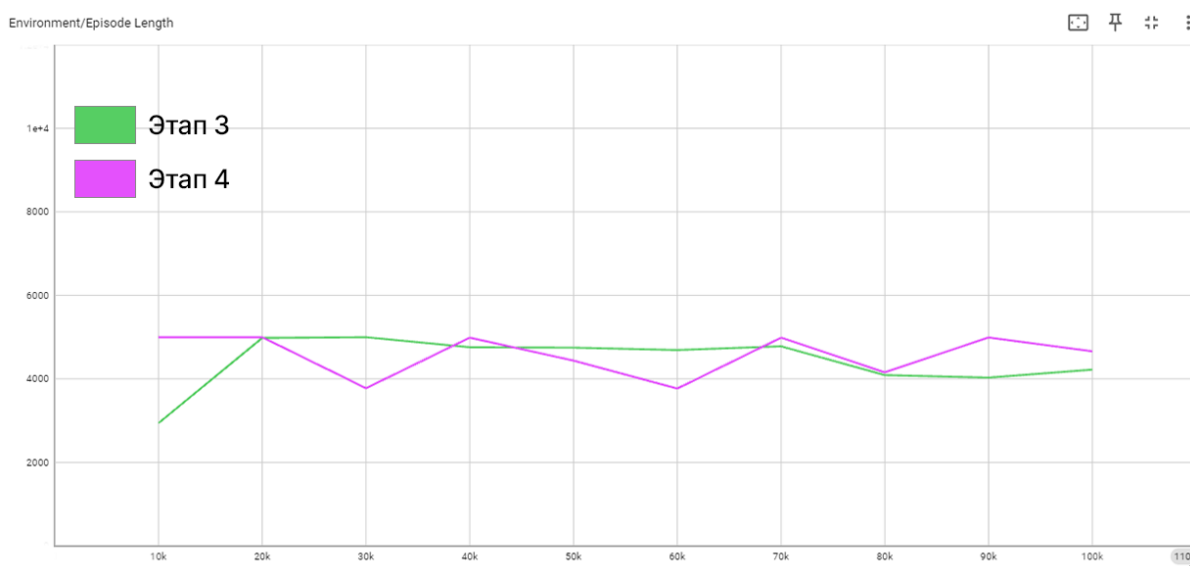


Рисунок 33 – Параметр Episode Length в 3 и 4 этапах

Третий и четвёртый этапы обучения позволили заключить, что при использовании Imitation Learning добавление дополнительных параметров Vector Observation усложняет процесс обучения нейросети. Однако такой подход по-прежнему позволяет обучать ИИ, выполняющий свои игровые задачи. При этом качество ИИ сильно зависит от стратегии обучения: выбранных методик вознаграждения, влияния демонстраций, а также отслеживания факторов, влияющих на переобучение.

8.4 Результаты оптимизации модели

Оптимизация модели заключалась в имплементации Imitation Learning и изменении среды обучения под нужды обновлённой методики обучения.

Поскольку Imitation Learning использует в своей основе демонстрационную запись, которая ложится в основу поведения ИИ, такой подход значительно сокращает время первичного обучения ИИ, и модель быстро начинает следовать требуемому паттерну действий. В результате ИИ быстрее и качественнее учится выполнять свои игровые задачи.

Успешность обучения при использовании такой методики определяется как демонстрацией и параметрами среды, так и переменными Vector Observation. В зависимости от их числа, а также влияния данных параметров на получаемое вознаграждение, агент будет отдавать предпочтение различным игровым задачам. В ходе повторного обучения было выявлено, что агент с меньшим числом переменных Vector Observation обучается быстрее и эффективнее. Соотношение получаемых в ходе двух серий обучения (этапы 1-2 и этапы 3-4) вознаграждений отображено на рисунке 34.

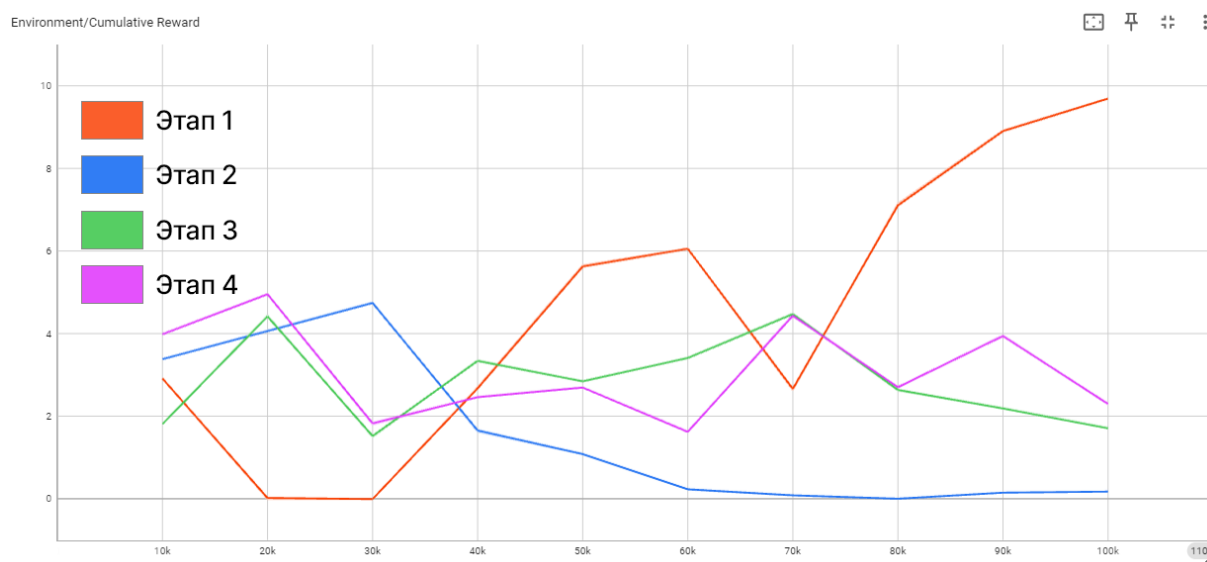


Рисунок 34 – Параметр Cumulative Reward в 1, 2, 3 и 4 этапах обучения

Значения Cumulative Reward в данных этапах обучения показывает, что при использовании только сенсоров границ трека агент достигает лучших результатов, чем при использовании сенсоров двух видов. Возможным решением могут стать более продолжительные этапы обучения, дальнейшая оптимизация среды, либо корректировка формул получения вознаграждения (включая введение новых переменных). Также необходимо учесть тенденцию к переобучению и отслеживать некачественные методики обучения, которые не должны быть включены в общую стратегию использования модели и создания ИИ.

9 Выводы по результатам работы

9.1 Результаты тестирования и оптимизации

В результате тестирования эволюционной модели было выявлен ряд недочётов первичной реализации. Была поставлена задача устранить её недостатки и применить более совершенный метод обучения, позволяющий получить качественный ИИ. В качестве нового метода обучения была выбрана присутствующая в Unity ML-Agents методика Imitation Learning. В процессе оптимизации была модифицирована среда обучения и усовершенствован механизм получения агентом вознаграждения. После применения Imitation Learning агент показал удовлетворительные способности к обучению, при помощи модели был получен ИИ, отвечающий своим игровым задачам.

Лучшими из полученных в результате работы являются ИИ 1 и 4 этапов обучения. ИИ 1 этапа способен достаточно успешно проходить трек целиком. ИИ 4 этапа в среднем проходит половину трека, однако также способен пройти весь трек. При этом ИИ 4 этапа передвигается сравнительно медленнее, однако учитывает присутствующие на локации зоны отличающегося типа поверхности («песок»). Была проведена серия раундов игры с вышеназванными видами ИИ. Раунды не были ограничены количеством шагов и учитывали число пройденных ИИ чекпоинтов. Общее число чекпоинтов на треке – 35. Результаты серии из 40 игр приведены на рисунке 35.

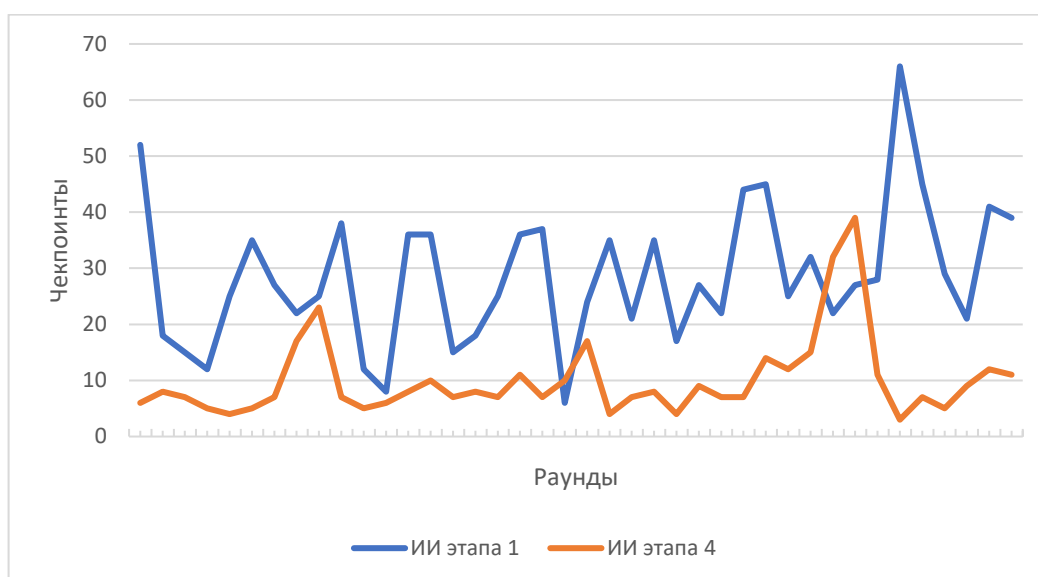


Рисунок 35 – Статистика прохождения трека ИИ 1 и 4 этапов

По представленным выше данным можно сделать вывод о том, что оба ИИ отвечают игровым задачам симуляции: они могут проходить гоночный

трек и адаптируются к условиям окружающей среды на локации. Оба вида ИИ могут быть использованы для дальнейшего обучения и в перспективе способны стать элементом игр жанра «гонки».

9.2. Результаты построения эволюционной модели

Результатом работы стала построенная эволюционная модель ИИ. Основываясь на ряде аналогичных исследований, она была создана с учётом достоинств других моделей. В главах 1 и 2 были проанализированы подходы к построению эволюционных моделей. Задачи научной работы определили выбор подхода RNEA, как наиболее предпочтительного: игры, требующие от ИИ учёта состояния локации, как правило, требуют поддержки непрерывного анализа данных и быстрого принятия решений. Этот же подход может использоваться и в других видах современных игр.

Для построения модели была разработана архитектура, определяющая общий вид эволюционной модели. В главе 4 описана сама архитектура, её состав, а также дополнительно приведены положения эволюционного моделирования, позволяющие добиться наиболее эффективных результатов при соответствующих условиях обучения и учёте ИИ состояния локации. Функции её элементов описаны в общем виде, что позволяет выбирать при разработке различные (при этом отвечающие данным функциям) программные средства построения.

Так, 5 и 6 главы описывают выбранные в качестве средств построения модели игровой движок Unity, фреймворк PyTorch и реализованный на его основе инструментарий Unity ML-Agents. Unity обеспечивает пространство симуляции, PyTorch – поддержку нейросети, а Unity ML-Agents – связь среды симуляции и действий агента, управляемого нейросетью. Оценка игры агента, а также дальнейший процесс отбора популяций требует возможности восприятия игрового процесса с позиции игрока, поэтому данные действия осуществлялись мануально.

При помощи данных средств была построена среда обучения для игр жанра «гонки». Возможность обучения игрового агента обеспечил инструментарий Unity ML-Agents. В ходе тестирования модели было установлено, что стандартный вариант обучения с подкреплением, используемый ML-Agents, является недостаточным для игр, требующих выполнения комплексных задач и отслеживания внутриигровых данных при помощи большого набора сенсоров. Обучение в таком случае происходит в крайне медленном темпе. Для повышения эффективности работы модели в

качестве оптимизации был использован подход Imitation Learning, доступный также в ML-Agents. Использование демонстраций позволило значительно ускорить темпы обучения агентов и получить ИИ, способный выполнять свои игровые задачи.

При построении модели следует учитывать, в первую очередь, возможность реализации с её помощью ИИ своих игровых задач. Эволюционное моделирование позволяет получить искусственный интеллект для конкретных задач, однако, если набор требуемых задач достаточно широк, а виды задач сильно отличаются, целесообразнее будет использовать эволюционную модель для одной либо нескольких задач одинакового типа. Эволюционная модель также может являться основой для создания ИИ различных игр, поэтому качественная модель будет более выгодным проектом с точки зрения вложения ресурсов и позволит сократить время работы над последующими проектами. Также необходимо уделить внимание доступным средствам построения эволюционной модели, особенно, если в качестве игрового движка используется ПО, разработанное сторонними компаниями. Данное программное обеспечение должно иметь возможность имплементации средств эволюционного моделирования.

Описанные выше принципы построения эволюционной модели могут быть использованы при разработке новых архитектур эволюционных моделей, а также реализации моделей при помощи существующих программных средств и решении задач по созданию игрового ИИ для игр различных жанров.

ЗАКЛЮЧЕНИЕ

В данной работе были проанализированы основные аспекты создания эволюционных моделей для разработки игрового ИИ. Были изучены алгоритмы, применяемые для создания эволюционных моделей ИИ, учитывающего внутриигровые факторы, такие как условия окружающей среды на локации.

Были рассмотрены особенности построения архитектуры эволюционных моделей. Определены основные компоненты архитектуры и их функции. По результату декомпозиции была создана собственная архитектура, которая затем использована для разработки собственной эволюционной модели. Был проведен обзор программных средств для построения модели, сравнение возможностей оптимизации.

После выбора Unity в качестве среды обучения игровых агентов с использованием Unity ML-Agents была построена эволюционная модель. Проведено первичное тестирование модели, оценена её эффективность, сделаны выводы о качестве работы модели. Результаты первичного обучения показали, что для получения ИИ требуемого качества необходима оптимизация модели. После модификации среды и внедрения методики Imitation Learning скорость обучения была значительно увеличена, что позволило получить качественные экземпляры ИИ.

Результаты ВКР, а также программный код, проект Unity и пример с инструкциями опубликованы в репозитории на GitHub. Получить доступ к вышеуказанным материалам с целью ознакомления можно по ссылке «https://github.com/JackArrow99/Evolutionary_AI_model_for_adapting_to_environmental_conditions_at_location».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Bourg D. M., Seemann G. AI for game developers. – " O'Reilly Media, Inc.", 2004.
2. Cohoon J., Kairo J., Lienig J. Evolutionary algorithms for the physical design of VLSI circuits //Advances in evolutionary computing. – Springer, Berlin, Heidelberg, 2003. – С. 683-711.
3. Faıçal B. S. et al. An adaptive approach for UAV-based pesticide spraying in dynamic environments //Computers and Electronics in Agriculture. – 2017. – Т. 138. – С. 210-223.
4. Guerrero-Romero C., Lucas S. M., Perez-Liebana D. Using a team of general ai algorithms to assist game design and testing //2018 IEEE Conference on Computational Intelligence and Games (CIG). – IEEE, 2018. – С. 1-8.
5. Емельянов В. В., Курейчик В. В., Курейчик В. М. Теория и практика эволюционного моделирования //М.: физматлит. – 2003. – Т. 432.
6. Gaina R. D. et al. Self-adaptive rolling horizon evolutionary algorithms for general video game playing //2020 IEEE Conference on Games (CoG). – IEEE, 2020. – С. 367-374.
7. Vikhar P. A. Evolutionary algorithms: A critical review and its future prospects //2016 International conference on global trends in signal processing, information computing and communication (ICGTSPICC). – IEEE, 2016. – С. 261-265.
8. Simpson R. Evolutionary Artificial Intelligence in Video Games //University of Minnesota. – 2012.
9. Justesen N. et al. Deep learning for video game playing //IEEE Transactions on Games. – 2019. – Т. 12. – №. 1. – С. 1-20.
10. Muñoz-Avila H. et al. Learning and game AI. – Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
11. Blum C., Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison //ACM computing surveys (CSUR). – 2003. – Т. 35. – №. 3. – С. 268-308.

12. Mautschke O. Evolutionary algorithms for Controllers in Games. – 2019.
13. Perez-Liebana D. et al. General video game ai: a multi-track framework for evaluating agents //Games and Content Generation Algorithms. – 2018.
14. Sironi C. F., Liu J., Winands M. H. M. Self-adaptive Monte Carlo Tree Search in general game playing //IEEE Transactions on Games. – 2018. – T. 12. – №. 2. – C. 132-144.
15. Gaina R. D. et al. Rolling horizon evolutionary algorithms for general video game playing //IEEE Transactions on Games. – 2021. – T. 14. – №. 2. – C. 232-242.
16. Chaslot G. M. J. B. et al. Progressive strategies for Monte-Carlo tree search //New Mathematics and Natural Computation. – 2008. – T. 4. – №. 03. – C. 343-357.
17. Gaina R. D. et al. Self-adaptive rolling horizon evolutionary algorithms for general video game playing //2020 IEEE Conference on Games (CoG). – IEEE, 2020. – C. 367-374.
18. Santos B., Bernardino H., Hauck E. An improved rolling horizon evolution algorithm with shift buffer for general game playing //2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames). – IEEE, 2018. – C. 31-316.
19. Lucas S. M., Liu J., Perez-Liebana D. The N-tuple bandit evolutionary algorithm for game agent optimisation //2018 IEEE Congress on Evolutionary Computation (CEC). – IEEE, 2018. – C. 1-9.
20. Gaina R. D., Lucas S. M., Pérez-Liébana D. Tackling sparse rewards in real-time games with statistical forward planning methods //Proceedings of the AAAI Conference on Artificial Intelligence. – 2019. – T. 33. – №. 01. – C. 1691-1698.
21. Mora A. M. et al. Effect of noisy fitness in real-time strategy games player behaviour optimisation using evolutionary algorithms //Journal of Computer Science and Technology. – 2012. – T. 27. – №. 5. – C. 1007-1023.

22. Goldberg D. E., Korb B., Deb K. Messy genetic algorithms: Motivation, analysis, and first results //Complex systems. – 1989. – Т. 3. – №. 5. – С. 493-530.
23. Jang S. H., Yoon J. W., Cho S. B. Optimal strategy selection of non-player character on real time strategy game using a speciated evolutionary algorithm //2009 IEEE Symposium on Computational Intelligence and Games. – IEEE, 2009. – С. 75-79.
24. Nicolau M. et al. Evolutionary behavior tree approaches for navigating platform games //IEEE Transactions on Computational Intelligence and AI in Games. – 2016. – Т. 9. – №. 3. – С. 227-238.
25. Murphy J. E., O'Neill M., Carr H. Exploring grammatical evolution for horse gait optimisation //Genetic Programming: 12th European Conference, EuroGP 2009 Tübingen, Germany, April 15-17, 2009 Proceedings 12. – Springer Berlin Heidelberg, 2009. – С. 183-194.
26. Lee W. M. Python machine learning. – John Wiley & Sons, 2019.
27. Paszke A. et al. Pytorch: An imperative style, high-performance deep learning library //Advances in neural information processing systems. – 2019. – Т. 32.
28. Unity ML-Agents Toolkit [Электронный ресурс]. URL: <https://unity-technologies.github.io/ml-agents/> (Дата обращения: 15.11.2023).
29. Learning Agents Introduction | Epic Developer Community [Электронный ресурс]. URL: <https://dev.epicgames.com/community/learning/tutorials/8OWY/unreal-engine-learning-agents-introduction> (Дата обращения: 03.12.2023).
30. Lanham M. Learn Unity ML-Agents–Fundamentals of Unity Machine Learning: Incorporate new powerful ML algorithms such as Deep Reinforcement Learning for games. – Packt Publishing Ltd, 2018.
31. Juliani A. et al. Unity: A general platform for intelligent agents //arXiv preprint arXiv:1809.02627. – 2018.
32. Metz L. A. E. P. An evaluation of unity ML-Agents toolkit for learning boss strategies : дис. – 2020.

33. Attia A., Dayan S. Global overview of imitation learning //arXiv preprint arXiv:1801.06503. – 2018.
34. Ho J., Ermon S. Generative adversarial imitation learning //Advances in neural information processing systems. – 2016. – Т. 29.
35. Training Configuration File [Электронный ресурс] // Unity ML-Agents Toolkit URL: <https://unity-technologies.github.io/ml-agents/Training-Configuration-File/> (Дата обращения: 03.03.2024).