

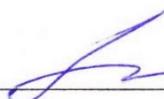


МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий
Кафедра корпоративных информационных систем

РАБОТА ДОПУЩЕНА К ЗАЩИТЕ

Заведующий
кафедрой



E.G. Андрианова

«01» июня 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по направлению подготовки бакалавров 09.03.04 Программная инженерия

На тему: Модуль принятия решений интеллектуальным агентом
в многопользовательском игровом веб-приложении

Обучающийся



подпись

Козеровский Дмитрий Никитич
Фамилия, имя, отчество

шифр 19И1682
группа ИКБО-08-19

Руководитель работы

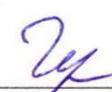


подпись

к.т.н., доцент

Советов П.Н.

Консультант



подпись

к.э.н., доцент

Черненькая И.Г.

Москва 2023 г.



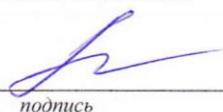
МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий

Кафедра корпоративных информационных систем

СОГЛАСОВАНО

Заведующий
кафедрой


подпись

Андианова Елена Гельевна

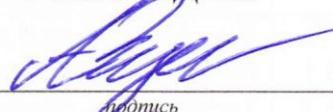
«7»

марта

2023 г.

УТВЕРЖДАЮ

Директор
института


подпись

Зуев Андрей Сергеевич

«7»

марта

2023 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы бакалавра

Обучающийся

Козеровский Дмитрий Никитич

Фамилия Имя Отчество

Шифр

19И1682

Направление
подготовки

09.03.04

Программная инженерия

индекс направления

наименование направления

Группа

ИКБО-08-19

1. Тема выпускной квалификационной работы: Модуль принятия решений интеллектуальным агентом в многопользовательском игровом веб-приложении

2. Цель и задачи выпускной квалификационной работы

Цель работы: разработка функционального модуля для предоставления пользователю игрового приложения возможности участия в матче с противниками под управлением искусственного интеллекта.

Задачи работы: изучение методов создания системы автоматизированного контроля игровыми персонажами в многопользовательских играх, исследование и обоснование выбора алгоритма работы интеллектуального агента, проектирование архитектуры модуля принятия решений, реализация и тестирование модуля принятия решений интеллектуальным агентом.

3. Этапы выпускной квалификационной работы

| № этапа | Содержание этапа выпускной квалификационной работы | Результат выполнения этапа ВКР | Срок выполнения |
|---------|--|--------------------------------|-----------------|
| 1 | Исследовательский раздел | | |
| 1.1 | Применение искусственного интеллекта (ИИ) в тактических играх | | |
| 1.2 | Методы ИИ, используемые в тактических играх | | |
| 1.3 | Изучение методов искусственного интеллекта, основанных на деревьях решений | | |
| 1.4 | Изучение методов искусственного интеллекта, основанных на обучении с подкреплением | | |
| 2 | Аналитический раздел | | |
| 2.1 | Проектирование архитектуры модуля принятия решений | | |
| 2.2 | Интеграция разрабатываемого модуля с ИС веб-приложения | | 22.03.2023 |
| 2.3 | Схема основных процессов анализа состояния окружения и принятия решений | | |
| 3 | Технологический раздел | | |
| 3.1 | Реализация модуля принятия решений | | |
| 3.2 | Тестирование модуля принятия решений | | 19.04.2023 |
| 3.3 | Анализ метрик работы модуля принятия решений | | |
| 4 | Экономический раздел | | |
| 4.1 | Организация и планирование работ по теме | | 17.05.2023 |
| 4.2 | Расчет стоимости проведения работ по теме | | |
| 5 | Введение, заключение, список источников, приложения | | 17.05.2023 |
| 6 | Презентационный материал, аннотация | | 25.05.2023 |
| 7 | Нормоконтроль | | 26.05.2023 |

4. Перечень разрабатываемых документов и графических материалов: печатная и электронная версии выпускной квалификационной работы бакалавра, презентационный материал с основными результатами выпускной квалификационной работы

5. Руководитель и консультанты выпускной квалификационной работы

| Функциональные обязанности | Должность в Университете | Фамилия Имя Отчество | Подпись |
|---------------------------------------|--------------------------|------------------------------|---------|
| Руководитель ВКР | доцент | Советов Петр Николаевич | |
| Консультант по экономическому разделу | доцент | Черненькая Ирина Геннадьевна | |

Задание выдал

Руководитель ВКР:

подпись

«7» марта 2023 г.

Задание принял к исполнению

Обучающийся:

подпись

«7» марта 2023 г.

Аннотация

Данная работа посвящена проблеме реализации системы автоматизированного контроля интеллектуальных агентов в игровой среде. В ходе работы произведено сравнение подходов и методов создания данной системы. На основании результатов сравнения, были реализованы наиболее эффективные из рассмотренных методов, а именно: алгоритм, использующий дерево принятия решений и алгоритм РРО на основе обучения с подкреплением.

Метрики, полученные при тестировании данных реализаций в игровом приложении, показывают, что дерево принятия решений лучше подходит для данной задачи.

Оглавление

| | |
|--|----|
| Введение..... | 6 |
| 1. Исследовательский раздел | 8 |
| 1.1. Применение ИИ в тактических играх | 8 |
| 1.2. Методы ИИ, используемые в тактических играх | 10 |
| 1.3. Изучение методов искусственного интеллекта, основанных на деревьях решений..... | 12 |
| 1.4. Изучение методов искусственного интеллекта, основанных на обучении с подкреплением..... | 13 |
| 2. Аналитический раздел..... | 19 |
| 2.1. Проектирование архитектуры модуля принятия решений..... | 20 |
| 2.2. Интеграция разрабатываемого модуля с ИС веб-приложения | 29 |
| 2.3. Схема основных процессов анализа состояния окружения и принятия решений..... | 31 |
| 3. Технологический раздел..... | 33 |
| 3.1. Реализация модуля принятия решений..... | 33 |
| 3.2. Тестирование модуля принятия решений | 40 |
| 3.3. Анализ метрик работы модуля принятия решений | 42 |
| 4. Экономический раздел | 47 |
| 4.1. Организация и планирование работ по теме | 47 |
| 4.2. Расчет стоимости проведения работ по теме | 48 |
| Заключение | 52 |
| Список использованных источников | 54 |
| Приложение А | 55 |
| Приложение Б | 56 |

Введение

В современном мире многопользовательские игры становятся все более популярными и востребованными формами развлечения. Они позволяют игрокам насладиться игрой с реальными соперниками и заработать виртуальные награды. По мере развития отрасли, игры становятся все более сложными и при их разработке возникает необходимость создания интеллектуальных агентов, которые могут принимать решения и выполнять определенные задачи, чтобы обеспечить более интересный и разнообразный игровой процесс. В частности, не всегда возможно быстро найти достаточное количество людей для начала партии, поэтому в таких случаях недостающих противником может заменить ИИ.

Цель данной работы заключается в исследовании методов создания интеллектуальных агентов, а также методов определения эффективных стратегий для принятия решений в многопользовательской пошаговой игры с последующей реализацией наиболее перспективных из них. Цель данного исследования состоит в разработке и оценке эффективности модуля принятия решений интеллектуальным агентом, который способен обучаться, адаптироваться и реагировать на действия противника.

В работе представлены результаты анализа алгоритмов, использованных при создании интеллектуального агента для многопользовательской пошаговой игры «Maze Game». Было проведено сравнение этих методов по критериям эффективности, скорости работы и простоты реализации. Кроме того, была реализована конкретная версия интеллектуального агента и его модуль принятия решений, на основе наиболее эффективных методов, выявленных в ходе исследования.

В процессе написания выпускной квалификационной работы автор (дипломник) руководствовался следующими нормативными актами:

1. «О защите населения и территории от чрезвычайных ситуаций природного и техногенного характера» от 21.12.1994 № 68-ФЗ.

2. «Об основах охраны здоровья граждан в Российской Федерации» от 21.11.2011 № 323-ФЗ
3. «О гражданской обороне» от 12.02.1998 № 28-ФЗ.
4. Приказ Минздравсоцразвития РФ от 04.05.2012 № 477н «Об утверждении перечня состояний, при которых оказывается первая помощь, и перечня мероприятий по оказанию первой помощи».
5. Трудовой кодекс Российской Федерации от 30.12.2001 № 197-ФЗ
6. СанПин – 2.2.2/542-96 «Гигиенические требования к видеодисплейным терминалам, персональным электронно-вычислительным машинам и организации работы»

1. Исследовательский раздел

В рамках данного раздела рассматриваются методы игрового искусственного интеллекта (ИИ) применимые к тактическим ролевым пошаговым многопользовательским играм, с процедурно генерируемой картой уровня.

Конечной целью данной работы является реализация модуля, расширяющего функциональность существующей информационной системы веб приложения. Приложение представляет из себя многопользовательскую пошаговую игру (цифровой аналог настольной игры) [1]. Данный модуль предоставляет пользователю приложения возможность участвовать в игровой партии с противниками под управлением ИИ.

1.1. Применение ИИ в тактических играх

Тактическая пошаговая игра – это жанр компьютерных игр, в котором игроки управляют отдельными юнитами (персонажами) и взаимодействуют друг с другом на игровом поле, состоящем из клеток определенной формы. Основным элементом игры является тактическое планирование и стратегические умения игрока, которые позволяют справляться с различными препятствиями на пути к победе.

Модели поведения агентов, использующие ИИ, могут хорошо соперничать с игроками. Использование различных методов ИИ позволяет разработчикам создавать более сложные, интересные и вызывающие увлечение игры.

Игровой ИИ сосредоточен на том, какие действия должен выполнять агент, исходя из условий среды, в которой находится. Обычно это называют управлением «интеллектуальными агентами», где агент является игровым персонажем, транспортным средством, ботом и т.п. В каждом из этих случаев важно, чтобы агент мог видеть своё окружение, принимать решения на основе информации об окружении и действовать соответственно. Это называется циклом Sense/Think/Act (восприятие/принятие решения/действие) [2]:

- Восприятие: агент получает или собирает информацию о среде в которой находится, которая может повлиять на его поведение (расположение противников, данные о предметах на поле, конфигурацию игрового уровня).
- Принятие решения: на основе наблюдения агент принимает решение о необходимом действии (анализирует, возможно ли безопасное движение в желаемую сторону или сначала он должен сражаться/скрываться).
- Действие: агент выполняет действия для выполнения предыдущего решения (например, производит движение к противнику или области на поле).
- Затем происходит изменение состояния среды в результате действий персонажей, и цикл повторяется с обновленными данными.

Обычно играм не требуется сложная система для извлечения информации, так как большая часть данных уже является ее неотъемлемой частью. Нет необходимости запускать алгоритмы распознавания изображений, чтобы определить, есть ли враг впереди — среда уже содержит данную информацию, которая может быть использована непосредственно в процессе принятия решений. Поэтому часть цикла «Восприятие» часто намного проще, чем «Принятие решения» и «Действие».

1.1.1. Ограничения игрового ИИ

При применении ИИ в играх есть ряд ограничений, которые необходимо соблюдать [2]:

- Действия агентов не должны быть оптимальными для игры против людей, поскольку игра должна развлекать и бросать вызов, но не быть непроходимой.
- Агенты должны подражать поведению реальных людей, чтобы игрокам казалось, что они играют против настоящих противников. Если игра имитирует человеческого противника, то действия, выбранные агентом, не должны слишком сильно отличаться от традиционных стратегий игры. Алгоритм должен быть оптимизирован таким образом, чтобы он принимал правдоподобные, а не идеальные решения.

- ИИ должен работать в допустимых временных рамках. Это значит, что алгоритм не может монополизировать использование ресурсов в течение длительного времени для принятия решений.
- Алгоритм должен быть устроен таким образом, чтобы оставалась возможность внести изменения в логику принятия решений агентом без изменения кода программы.

Последнее ограничение не является обязательным, но позволит сильно упростить процесс балансировки игры.

1.2. Методы ИИ, используемые в тактических играх

В пошаговых играх могут применяться различные методы ИИ для создания интеллектуальных агентов. Среди которых можно выделить алгоритмы, уже ставшие классическими для данной предметной области:

- Конечный автомат (Finite-state machine) – В FSM-алгоритме разработчик обобщает все возможные ситуации, с которыми может столкнуться ИИ, а затем программирует конкретную реакцию для каждой из них.
- Дерево принятия решений (Decision tree) – метод моделирования принятия решений, который используется для прогнозирования и ориентирования в принятии правильных решений. Один из способов отображения алгоритма, который содержит только операторы условного управления. Агент в игровой среде также использует дерево принятия решений для идентификации наиболее эффективного пути для выполнения задачи.
- Дерево поиска Монте-Карло (Monte Carlo Tree Search) – метод, основанный на генерации случайных выборок. MCTS-алгоритм обрабатывает все возможные ходы, доступные в конкретный момент времени. Затем для каждого из этих возможных ходов он анализирует все действия, которыми игрок мог бы ответить. После чего возвращается к оценке уже на основе информации о поступках игрока. В пошаговых играх, MCTS-алгоритм используются для симуляции множества возможных ходов и выбора наилучшего из них.

И подходы на основе машинного обучения [3]:

– Обучение с подкреплением (reinforcement learning) - алгоритмы, которые позволяют приучать агента действовать в среде определенным образом, используя отзывы в виде наград и штрафов. В пошаговых играх, наградой может быть победа, а штрафом поражение.

– Обучение с учителем (supervised learning) – методы машинного обучения, где учителями являются эксперты (люди или другие программы), которые дают алгоритму правильные ответы на заданных примерах. В пошаговых играх, такие примеры могут быть реализованы как наборы ходов, сделанные в ответ на определенные позиции на доске.

– Эволюционное обучение (evolutionary learning) – методы, основанные на идеях естественного отбора, где популяция агентов «размножается», создавая новых агентов, и только самые успешные агенты выживают. В пошаговых играх, эволюционное обучение может использоваться для оптимизации параметров алгоритмов поиска ходов.

Для рассматриваемой игровой среды, при условии, что карта заранее известна и при размере поля 5x5 для 4х игроков существует $(5 * 5)^4 = 390\,625$ начальных состояний, и при расчете в глубину данное число может увеличиться на несколько порядков. Таким образом для решаемой задачи методы, требующие большой размеченной обучающей выборки, не подходят ввиду невозможности сбора большого объема размеченных данных. Также невозможно использование методов, требующих расчета будущих состояний среды, из-за высокой скорости увеличения числа возможных состояний среды.

В средах, где число возможных состояний очень велико и не может быть рассчитано заранее наилучший результат показывают алгоритмы на основе обучения с подкреплением [4].

Другим методом ИИ, подходящим для данной задачи, является построение дерева принятия решений в котором описывается стратегия агента в зависимости от состояния среды в текущий момент времени. Данный метод позволяет гибко настраивать поведенческую модель агента и имитировать человеческое поведение.

1.3. Изучение методов искусственного интеллекта, основанных на деревьях решений

Задача агента в игровой среде может быть любой: от поиска определенных объектов до целенаправленного перемещения. Важно, чтобы агент мог принимать такие решения, которые приближают его к победе.

Созданное дерево принятия решений должно отражать возможные варианты действий агента и вычислять наиболее вероятные и эффективные для выполнения задачи. Важно учитывать, что в идеале агент должен иметь доступ к другой информации о состоянии игры и контексте, такой как положение других игровых персонажей или возможность получения помощи от союзников. Эта информация может изменить параметры использования дерева принятия решений и варианты действия агента.

В итоге, разработанное дерево принятия решений позволит агенту принимать лучшие решения, достигать своих целей и более эффективно использовать свои ресурсы в игровой среде.

В качестве примера рассмотрим дерево принятия решений для игры в Pong. Алгоритм проходит его, чтобы достичь «листа» — решения о том, какое действие предпринять. Блок-схема дерева решений для описания алгоритма движения платформы приведена на рисунке 1.1:

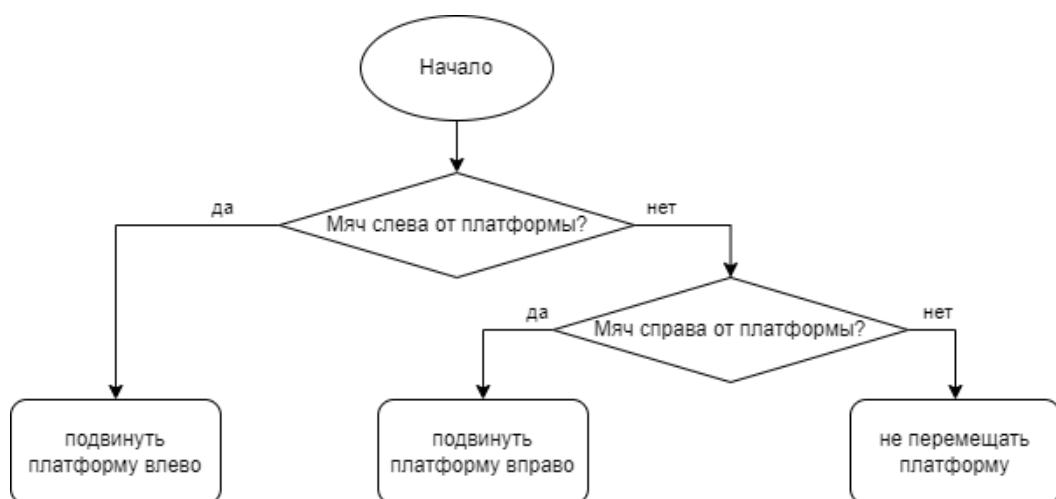


Рисунок 1.1 – Блок-схема дерева решений

Каждая часть дерева называется node (узел). Существует два типа узлов:

- Узлы принятия решений: выбор между двумя альтернативами на основе проверки некоторого условия, где каждая альтернатива представлена в виде отдельного узла.
- Конечные узлы: действие для выполнения, представляющее окончательное решение.

Алгоритм начинается с первого узла («корня» дерева). Он либо принимает решение о том, к какому дочернему узлу перейти, либо выполняет действие, хранящееся в узле, и завершается. Каждое решение может иметь только одно условие и два возможных результата [5].

Для построения дерева принятия решений для агента в игровой среде необходимо:

- Определить цели и правила игры;
- Собрать данные об игровом мире, такие как доступные действия, состояние окружающей среды, данные о других играх;
- Разработать модель агента, которая может принимать решения на основе данных об игровом мире и своих целей;
- Создать дерево принятия решений, которое будет отображать возможные действия агента и их последствия;
- Настроить дерево принятия решений на основе опыта агента и его реакций на различные игровые ситуации;
- Оптимизировать дерево принятия решений на основе метрик, таких как производительность агента, его успешность в игре и т.д.;
- Протестировать дерево принятия решений в различных игровых сценариях и внести необходимые изменения, если это необходимо.

1.4. Изучение методов искусственного интеллекта, основанных на обучении с подкреплением

Постановка задачи обучения с подкреплением [4]:

Взаимодействие агента со средой:

инициализация стратегии $\pi_1(a|s)$ и состояния среды s_1 ;

для всех $t = 1 \dots T$:

- агент выбирает действие $a_t \sim \pi_t(a | s_t)$;
- среда генерирует награду $r_{t+1} \sim p(r | a_t, s_t)$ и новое состояние среды $s_{t+1} \sim p(s | a_t, s_t)$;
- агент корректирует стратегию $\pi_{t+1}(a | s)$.

Основываясь на таком взаимодействии с окружающей средой, агент, обучающийся с подкреплением, должен выработать стратегию $\pi: S \rightarrow A$, которая максимизирует величину $R = \sum_{t=1}^n r_t$, где S – множество состояний среды, π – стратегия, A – пространство действий, r – награда.

Хоть простые системы реакций очень действенны, бывает много ситуаций, когда их недостаточно. Иногда нужно принимать различные решения, основанные на том, что агент делает в настоящий момент, но представлять это за условие тяжело. Иногда существует слишком много условий, чтобы эффективно представить их в дереве решений или скрипте, либо требуется заранее оценивать, как изменится ситуация, прежде чем принимать решение о следующем шаге. Для решения этих проблем нужны более сложные подходы.

Policy Gradient алгоритмы, в отличие от Deep Q Network (DQN), предполагают приближение нейронной сетью непосредственно самой политики. Это позволяет избежать необходимости решать задачу оптимизации – находить действие, имеющее максимальную ценность в данном состоянии. Для сред с небольшим числом допустимых действий решение задачи оптимизации не является проблемой, однако задача оптимизации может стать очень сложной в случае, когда число допустимых действий большое или действие является непрерывным, т.е. имеет бесконечное число допустимых значений [6].

На момент написания данной работы наиболее широкое распространение получили алгоритмы, основанные на Policy Gradient: A2C и PPO.

1.4.1. Алгоритм A2C (Advantage Actor Critic)

Алгоритм A2C (Advantage Actor Critic) является комбинацией двух других алгоритмов - Policy Gradient и Value Function. Градиент политики определяется следующей формулой (1.1):

$$\nabla J \approx E [Q(s, a) \nabla \log \pi(a|s)] \quad (1.1)$$

С практической точки зрения оптимизация градиента политики может быть реализована как оптимизация функции потерь (1.2) вида:

$$L = -Q(s, a) \log \pi(a|s) \quad (1.2)$$

Значение функции $Q(s, a)$ может быть разбито на две части: ценность состояния $V(s)$ и преимущество $A(s, a)$, которое даёт действие a . Использование $A(s, a)$ вместо $Q(s, a)$ в градиенте политики позволяет существенно повысить стабильность обучения. Для вычисления $V(s)$ используется отдельная нейронная сеть. Нейронная сеть, отвечающая за вероятности действий, называется actor, отвечающая за ценность состояния – critic [6].

Принцип работы A2C алгоритма состоит в следующем:

1. Агент взаимодействует с окружающей средой и получает награду за правильные действия.
2. Агент использует Policy Gradient для улучшения стратегии принятия решений.
3. Агент использует Value Function для оценки ожидаемой награды на основе текущего состояния и действия.
4. Агент использует Advantage Function для определения преимущества выбранного действия по сравнению с другими действиями.
5. Агент обновляет веса нейронной сети, используемой для вычисления Policy Gradient, Value Function и Advantage Function.

6. Агент повторяет шаги 1-5 до достижения желаемого уровня производительности. (рисунок 1.2)

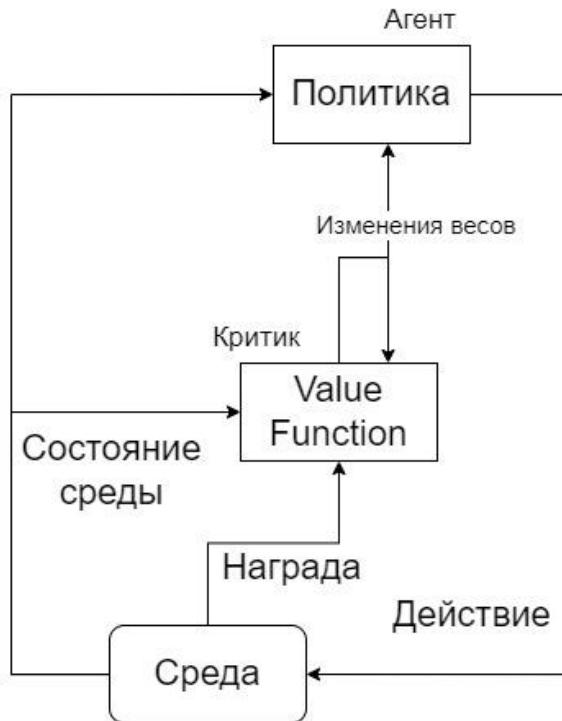


Рисунок 1.2 – Схема работы A2C алгоритма

В целом, A2C алгоритм является более эффективным, чем Policy Gradient и Value Function, потому что он сочетает в себе преимущества обоих методов и позволяет обучению быть более стабильным и быстрым.

1.4.2. Алгоритм PPO (Proximal Policy Optimization)

Дальнейшим улучшением A2C является алгоритм Proximal Policy Optimization (PPO), предложенный командой OpenAI в 2017 году, обеспечивающий правильный баланс между производительностью и пониманием агента среды. PPO в настоящее время считается передовым методом обучения с подкреплением. Эмпирически он конкурентоспособен с эталонными тестами качества, даже значительно превосходя их в некоторых задачах. В то же время он достаточно прост для практического внедрения широким кругом пользователей, чего нельзя сказать о каждом алгоритме RL. Основным предлагаемым алгоритмом улучшением является изменение формулы градиента политики. Градиент политики вычисляется как отношение

новой и старой политик, масштабируемое значением преимущества действия (1.3): [7]

$$J_\theta = E_t \left[\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta old}(a_t | s_t)} \right] At \quad (1.3)$$

Принцип работы алгоритма РРО заключается в анализе текущего состояния среды, принятии решения и выполнении действия, а затем определении, насколько хорошо это решение было для достижения поставленных целей.

Алгоритм РРО использует оптимизацию политики близости, которая позволяет избежать небольших изменений политики, которые приводят к большим изменениям в наградах.

РРО разделен на несколько этапов:

- Сбор данных: агент совершает действия в окружении и собирает данные в виде наблюдений, действий, наград и следующих состояний.
- Обработка данных: данные передаются в нейронную сеть, которая использует их для обучения.
- Обновление параметров: на этом этапе происходит обновление параметров нейронной сети таким образом, чтобы максимизировать суммарную награду.
- Повторение цикла: этот цикл продолжается до тех пор, пока нейронная сеть не достигнет необходимой степени точности в предсказании оптимального действия для текущей ситуации в окружении.

В целом, алгоритм РРО является частным случаем более общего алгоритма обучения с подкреплением, который используется для тренировки агентов-роботов и других систем искусственного интеллекта. В отличие от других подходов к обучению с подкреплением, РРО позволяет достичь более стабильных результатов в различных сценариях.

Таким образом обучение с подкреплением представляет собой мощный инструмент для разработки ИИ в тактических играх. Оно может быть использовано для обучения агентов, способных эффективно принимать решения и достигать целей в сложных и неструктурированных ситуациях.

В таблице 1.1 приведены результаты сравнения рассмотренных алгоритмов. На основе данного сравнения выбраны следующие алгоритмы для реализации: РРО и дерево принятия решений, – с целью сравнения их эффективности в рассматриваемой задаче.

Таблица 1.1 – Сравнение рассмотренных алгоритмов

| Тип алгоритма | Преимущества | Недостатки | Применимость в данной задаче |
|---------------------------|---|---|-------------------------------------|
| 1 | 2 | 3 | 4 |
| Эволюционное обучение | Хорошая сходимость и способность к глобальному поиску решения. | Требует большого числа поколений для обучения. Требует корректно реализованного агента. Сложность реализации. | нет |
| Конечный автомат | Хорошая производительность. | Предсказуемость действий агента. Необходимость в системе расчета состояния. Сложность реализации. | нет |
| Дерево поиска Монте-Карло | Позволяет решать сложные задачи без необходимости аналитического вычисления. | Требует большое количество итераций для обучения. Расчет состояний среды в глубину | нет |
| Обучение с учителем | Легко интерпретируемый и позволяет достаточно точно решить задачу. Простота реализации. | Требует большого количества размеченных данных. Сложность реализации. | нет |
| Дерево принятия решений | Легкость интерпретации решений, быстрота построения модели. Простота реализации. | менее точное поведение агента по сравнению с обучением с подкреплением | да |
| A2C | Учитывает динамику окружающей среды и даёт возможность создавать автономных агентов. Простота реализации. | Требует большого числа итераций для эффективного обучения. Не гарантирует оптимальных решений. | да |
| РРО | Более стабильный и эффективный, чем A2C, возможно параллельное обучение. Простота реализации. | В некоторых случаях может потребоваться большое число итераций для обучения. | да |

2. Аналитический раздел

Модуль принятия решений разрабатывается как часть ИС веб-приложения «Maze Game». Задачами данного модуля является построение среды для работы в ней интеллектуального агента и принятие решения о ходе исходя из состояния среды.

Игровые партии проходят по следующим правилам [1]:

- В игровой партии участвует несколько игроков, которые ходят поочередно. Каждый из игроков управляет своим персонажем.
- Игровая партия проходит на процедурно-генерируемом игровом уровне, состоящем из клеток и имеющем заданный размер. Игровой уровень включает в себя клетки различных типов, которые различным образом оказывают влияние на персонажей (например, реки имеют течение и меняют позицию персонажа, который в ней оказался, а при входе в медпункт здоровье персонажа будет восстановлено).
- В начале партии персонажи игроков появляются в указанных игроками клетках. При запуске партии игроки знают лишь позицию своего персонажа на карте и тип клетки в которой он находится, но не знают, как именно выглядит карта уровня.
- У персонажей есть набор характеристик (здоровье / запас бомб и стрел).
- В свой ход игрок может совершить одно из 13 действий: движение / выстрел из лука /бросок бомбы в одном из 4x направлений и взятие клада. При совершении действий игроки получают больше сведений о карте, что в конечном счете позволяет полностью ее восстановить. Игра заканчивается в случае если один из игроков вынесет с уровня «настоящий клад», либо игрок останется единственным выжившим.

Для того чтобы действия, совершаемые агентом, были похожи на действия живого игрока необходимо чтобы агент обладал такими же данными, как если бы он был обычным игроком.

Таким образом разрабатываемый модуль должен включать в себя:

- модуль обработки данных о ходах игроков для построения карты уровня;
- модуль для обработки и расчета информации о положении соперников;
- модуль отвечающий за принятие решений интеллектуальным агентом о действии в ходе на основе собранных данных.

На рисунке 2.1 приведена взаимодействие компонентов модуля принятия решений:

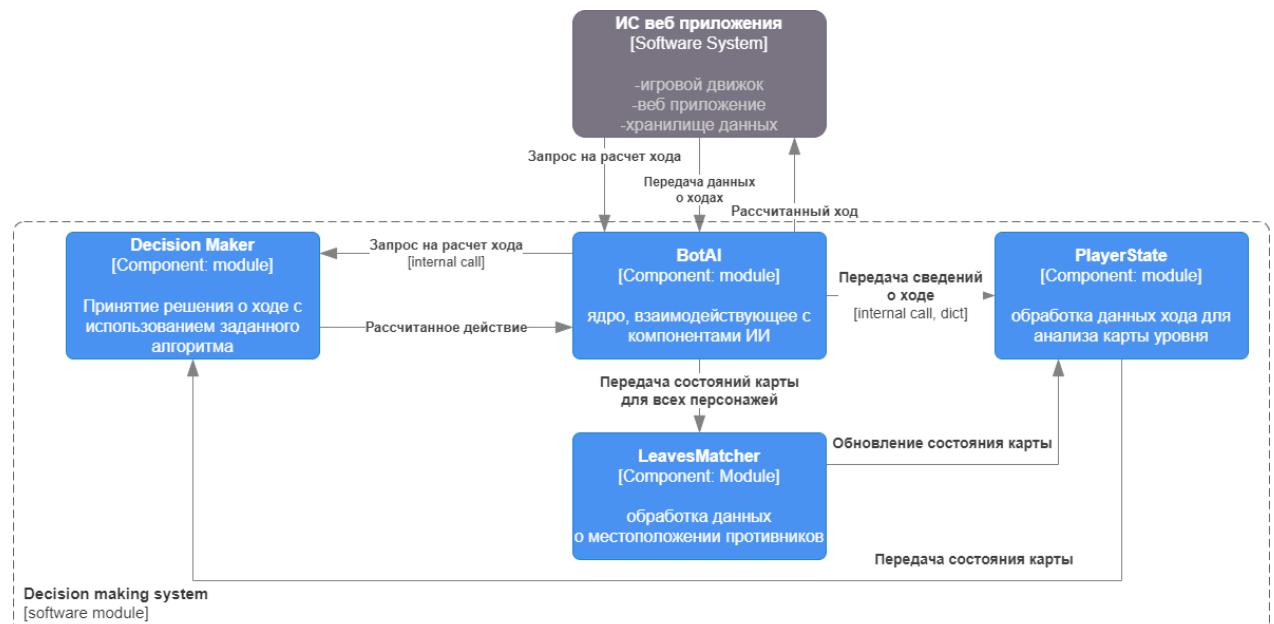


Рисунок 2.1 – Взаимодействие компонентов модуля принятия решений

2.1. Проектирование архитектуры модуля принятия решений

При совершении игроками (в том числе теми, за которых играет ИИ) ходов, модуль получает на вход данные об игроке, совершенном действии и ответ игры на данное действие. При запросе от внешней ИС, модуль должен вернуть рассчитанный ход для указанного игрока.

Для реализации данной функциональности необходимо хранить актуальное состояние среды, включающее в себя: положения игроков, их «знания» о карте, их текущие характеристики и расположение известных игрокам объектов на карте.

При запуске игровой партии происходит инициализация объекта (компонент BotAI), который содержит начальные состояния каждого

персонажа. При совершении игроком хода вызывается метод этого объекта, который обновляет хранимые состояния персонажей и подготавливает среду к обработке следующего хода. При запросе рассчитанного хода вызывается метод, производящий анализ текущего состояния среды с точки зрения указанного игрока, и выполняющий расчет оптимального действия. Верхнеуровневая архитектура модуля приведена на рисунке 2.2.

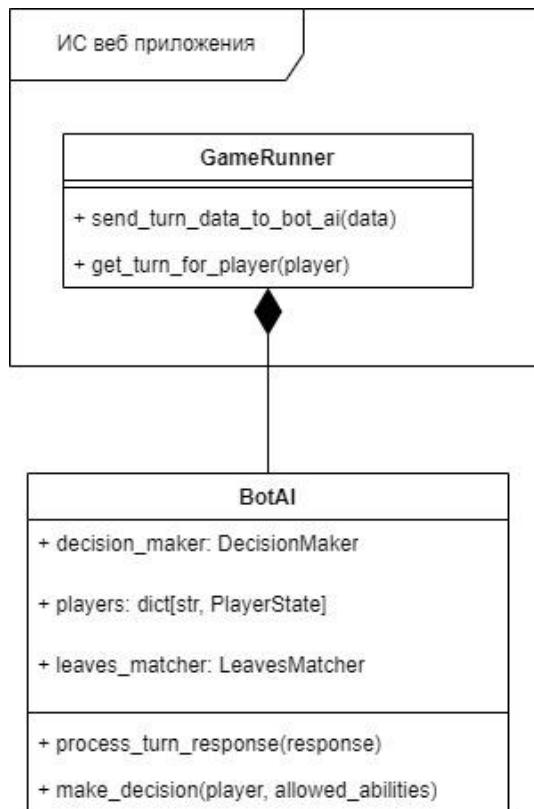


Рисунок 2.2 – Верхнеуровневая архитектура системы.

2.1.1. Архитектура подмодуля, отвечающего за обработку данных о ходах игроков для построения и хранения карты уровня

Как было сказано выше, необходимо хранить актуальное состояние среды. Данное состояние должно быть персонализировано для каждого персонажа, поскольку изначально не известно где находятся остальные персонажи. За обработку поступающих данных о ходе и хранения текущего состояния отвечает компонент PlayerState. Текущее состояние среды описывается деревом, узлы которого (компонент Node) содержат объекты FieldState. Листы данного дерева отображают все возможные конфигурации

среды на текущий момент с учетом прошлых ходов. Состояния являются равновероятными поскольку неизвестна точная конфигурация карты.

При запуске игры для каждого игрока в матче инициализируется дерево возможных начальных состояний карты для указанного игрока. То есть если поле было размером 5×5 , то при начальной генерации у каждого дерева будет 25 листов (каждый лист отображает появление игрока в одной из возможных клеток).

В начальный момент внутри дерева конкретного игрока содержится положение только его персонажа. В процессе игры информация о позициях соперников может добавляться.

При совершении хода сначала обрабатываются возможные изменения характеристик игрока (запаса здоровья / бомб / стрел / наличие клада в руках), за хранение которых отвечает компонент PlayerStats. Затем для каждого игрока формируется список его возможных состояний. Для каждого состояния из списка проверяется известна ли в нем позиция текущего игрока. В случае если позиция не известна состояние никак не изменяется. Иначе рассчитываются все возможные изменения состояния среды на основе данных о ходе, в результате чего появляется несколько ≥ 0 равновероятных новых состояний карты. В случае, если число новых состояний равняется нулю предыдущее состояние является ошибочным (т.е. исходя из новой информации о ходах можно сделать вывод о невозможности рассчитанной конфигурации карты), и затем рекурсивно производится удаление предшествующих состояний до тех пор пока у предшествующего состояния не будет ≥ 1 следующего состояния.

Диаграмма классов данного подмодуля приведена на рисунке 2.3:

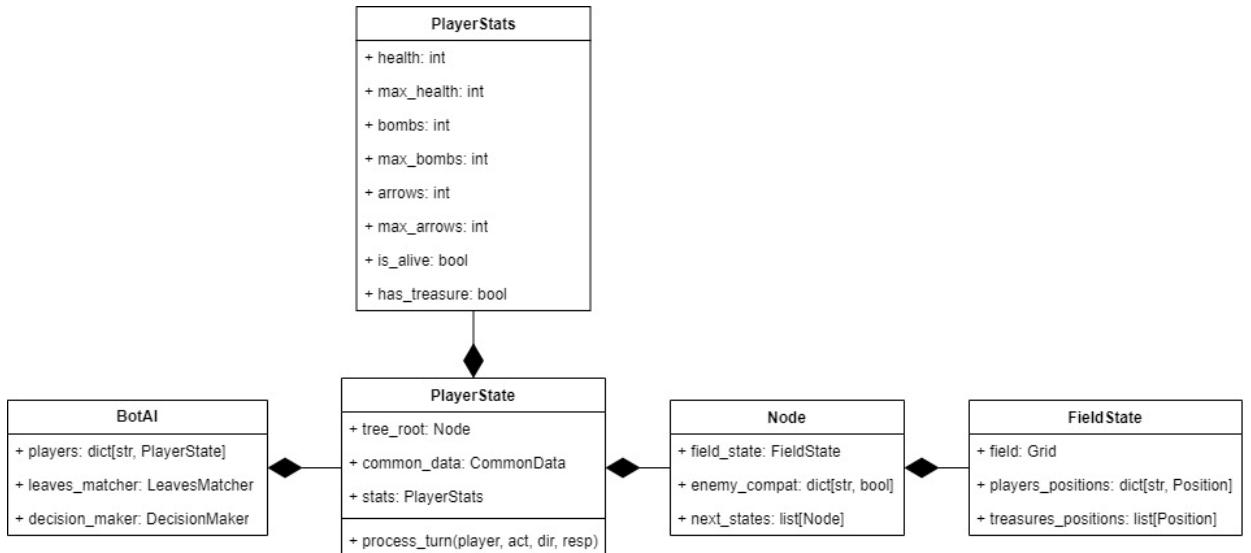


Рисунок 2.3 – Диаграмма классов подмодуля обработки данных о ходах

2.1.2. Архитектура подмодуля, отвечающего за обработку и расчет информации о положении соперников

Для расчета положения соперников необходимо выполнить наложение каждого из возможных состояний текущего игрока на каждое из всех состояний остальных игроков, положения которых еще не известны для данного состояния.

Эта операция производится после завершения обработки каждого хода. Поскольку данная операция является крайне ресурсоемкой (сложность $O(n^k)$, где k – число игроков, n – среднее число возможных состояний игроков), то, в целях оптимизации, ее выполнение производится в случае если число состояний текущего игрока меньше заданного (компонент LeavesMatcher).

В случае если $a_i \cap P_1 \cap \dots \cap P_{k-1} = \emptyset$, где a_i – одно из состояний текущего игрока, P_j – множество состояний противника, состояние является ошибочным и производится рекурсивное удаление аналогично случаю, описанному в разделе 2.1.1. Результатом успешного наложения служит объединение данных о карте одного игрока с данными другого – то есть множество новых объектов FieldState, которое сохраняется в объекте текущего игрока.

На рисунке 2.4 приведено графическое представление результата расчета ходов зеленого игрока и их частичное объединение с данными о ходах красного игрока.

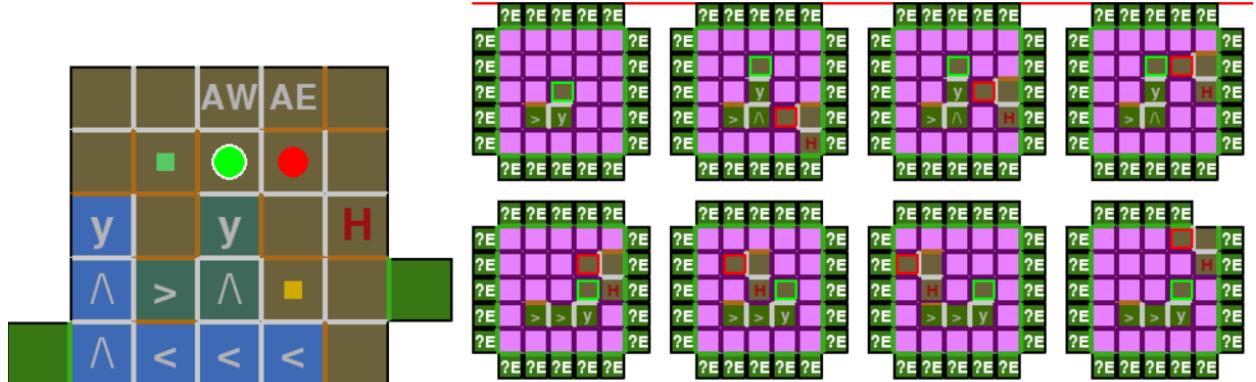


Рисунок 2.4 – Графическое представление карты игрового уровня и нескольких из возможных вариантов карты с точки зрения зеленого игрока

На рисунке 2.5 отображена диаграмма классов, задействованных в работе данного подмодуля.

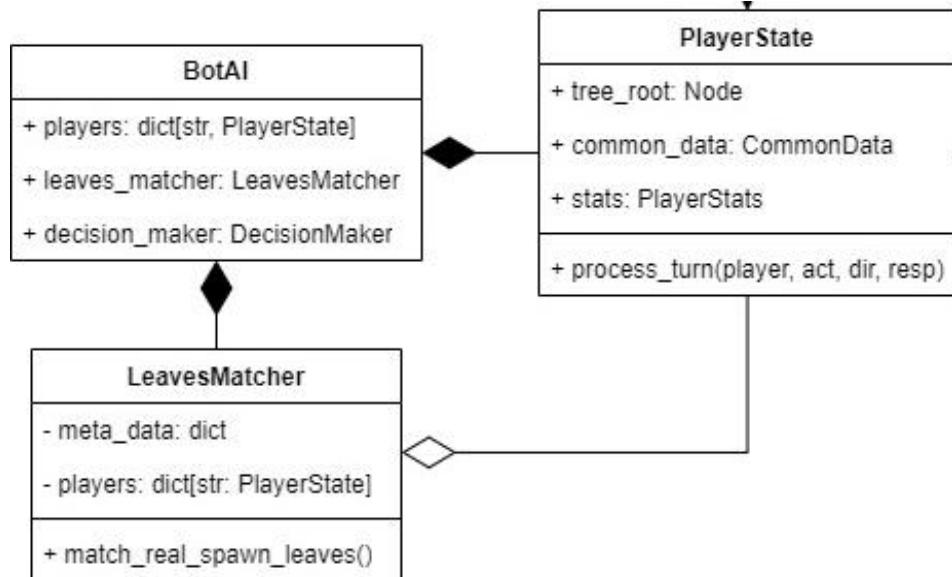


Рисунок 2.5 – Взаимосвязь компонентов подмодуля расчета положения противников

2.1.3. Архитектура подмодуля, отвечающего за принятие решения о ходе использующего дерево принятия решений

Для каждого возможного состояния карты с точки зрения текущего игрока, хранящегося в объекте `PlayerState`, строится взвешенный направленный граф переходов по карте (рисунок 2.6).

При построении графа учитывается наличие у игрока бомб (это позволит сломать стену). В случае если проход в нужную сторону не препятствует стена, вес ребра равняется 1, а действие – движение в указанную сторону, при наличии стены с заданной стороны, в случае если она разрушима, вес ребра равняется 2, а действие – бросок бомбы. Таким образом становится возможным применение алгоритмов поиска пути в графе на основе весов ребер.

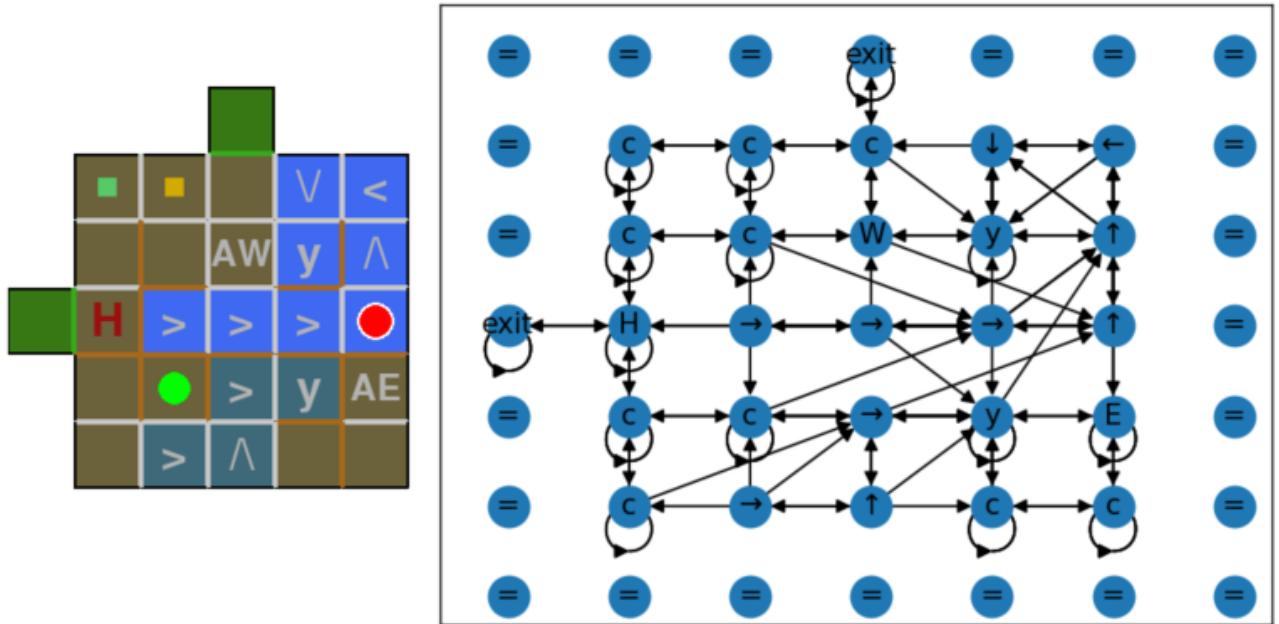


Рисунок 2.6 – Пример карты уровня и графа переходов по ней

С помощью алгоритма Дейкстры [8] из клетки, в которой находится текущий игрок, рассчитывается путь до всех остальных клеток и расстояние в ходах до каждой из них.

На основе данных о известных местоположениях оппонентов, их показателях характеристик, знаний о конфигурации карты и расстояний до других клеток производится выбор клетки в которой хочет оказаться агент. Действие, необходимое для достижения выбранной клетки кратчайшим маршрутом и будет итоговым выбранным действием в данном листе (компонент TargetCalculator). Отдельно проверяется возможность выстрела в противника и подбора сокровища. Данные действия имеют больший приоритет, чем движение (компонент DecisionMaker).

Упрощенное дерево принятия решения о ходе для одного листа приведено на рисунке 2.7.

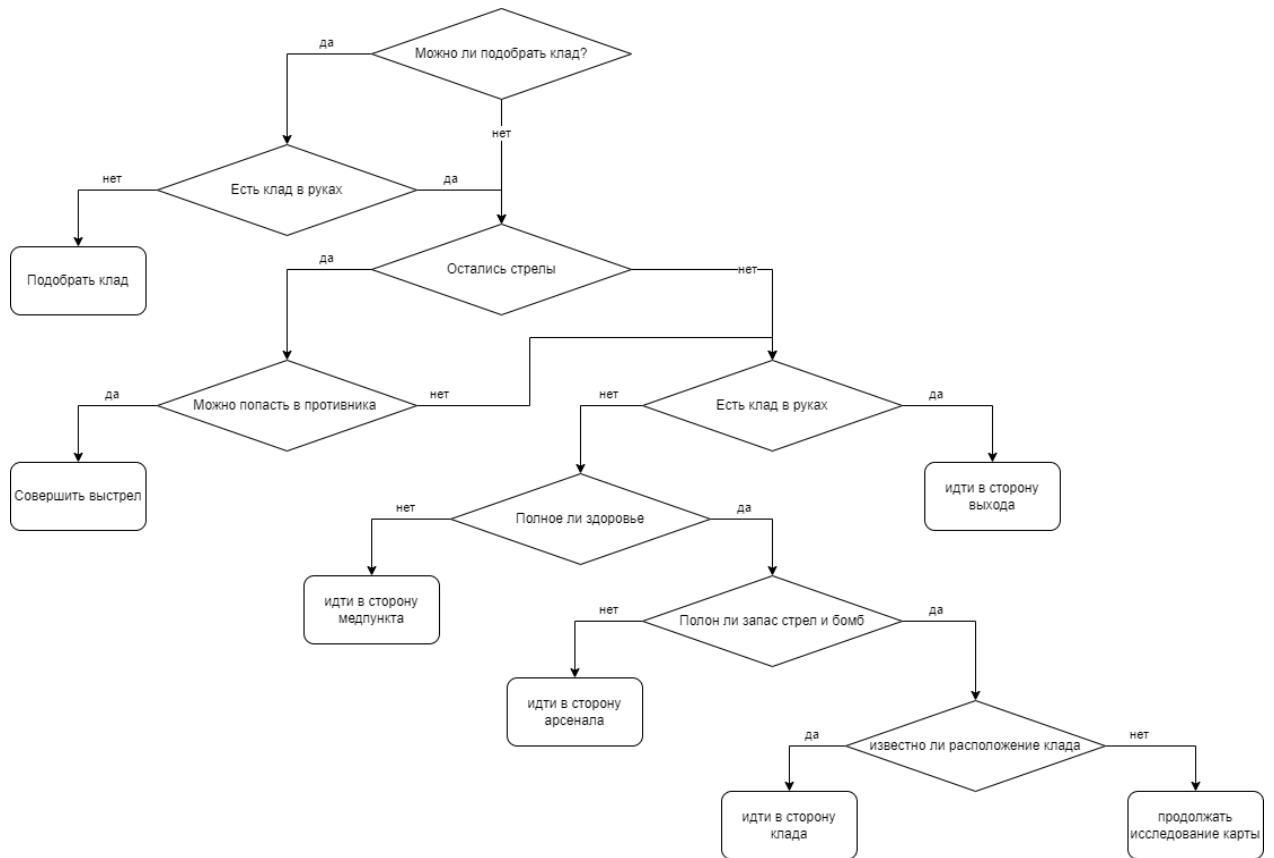


Рисунок 2.7 – Дерево принятия решений

В действительности условия в узлах дерева решений рассчитываются динамически исходя из таких факторов как: расстояния до объектов, положение и текущие показатели противников, расположение кладов и т.п.

Затем среди всех выбранных действий во всех листах выбирается самое часто встречающееся – это действие является итоговым результатом работы алгоритма агента.

На рисунке 2.8 отображена взаимосвязь компонентов данного подмодуля.

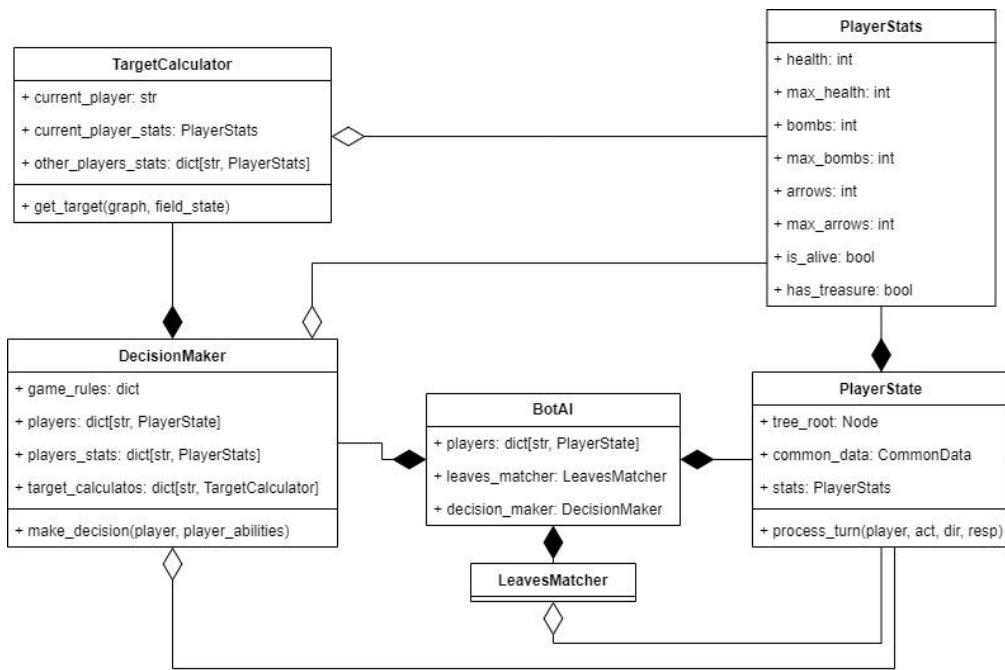


Рисунок 2.8 – Взаимосвязь компонентов подмодуля принятия решений

2.1.4. Архитектура подмодуля, отвечающего за принятие решения о ходе использующий обучение с подкреплением (PPO)

В отличие от алгоритма, использующего дерево принятия решений, данная реализация принимает решение о ходе на основе полной информации об игровой среде, полученной напрямую от игрового движка.

Компонент `MAMazeGameEnv` включает в себя методы и атрибуты, необходимые для симуляции ходов агента в игровом окружении, а именно:

- пространство допустимых действий агентов;
- пространство наблюдений за средой – набор признаков, описывающих игровое поле данные об агентах;
- метод, изменяющий состояние среды в зависимости от действия агента;
- метод, возвращающий награду агента за совершенное действие;
- метод, возвращающий текущее состояние среды для указанного агента.

По большей части этот компонент является оберткой объекта игры для совместимости с использующимся для обучения фреймворком (`Ray[RLlib]`).

Компонент RlGame содержит объект среды и объект алгоритма (PPO), рассчитывающего действия агента с помощью политики (PPOPolicy), которая содержит в себе следующие модели машинного обучения:

- MazeCnn: Набор свёрточных моделей – предобработка признаков игрового поля (отдельно признаков карты, данных об агентах и положении объектов на карте);
- ComplexInput: Объединение выходов свёрточных моделей
- A2CNetwork: actor и critic модели – получают результат объединения;
- ActionMaskModel: Выход модели-actor`а объединяется с маской действий
- PPOPolicy: К выходу модели-actor`а применяется TorchCategorical функция активации – результатом чего является вероятность выбора каждого из элемента пространства допустимых действий агента. Выход модели «критик» используется для оптимизации политики обучения. На этапе обучения используется алгоритм PPO.

Итоговым действием, рассчитанным политикой, является действие с наибольшей вероятностью выбора.

Обучение агента производится в партии с 4 игроками, так как данное число игроков является наиболее часто используемым. В каждой партии в процессе обучения одним из персонажей управляет обучаемый агент, политики остальных 3х противников являются константными. При запуске обучения обучаемый агент играет против «случайной» политики – т.е. действия его противников являются случайными. При достижении порогового процента побед, создаётся копия политики обучаемого агента и «случайные» политики заменяются данной копией, после чего процесс обучения продолжается.

Таким образом архитектура данного подмодуля должна иметь следующий вид (рисунке 2.9 отображена диаграмма классов данного подмодуля):

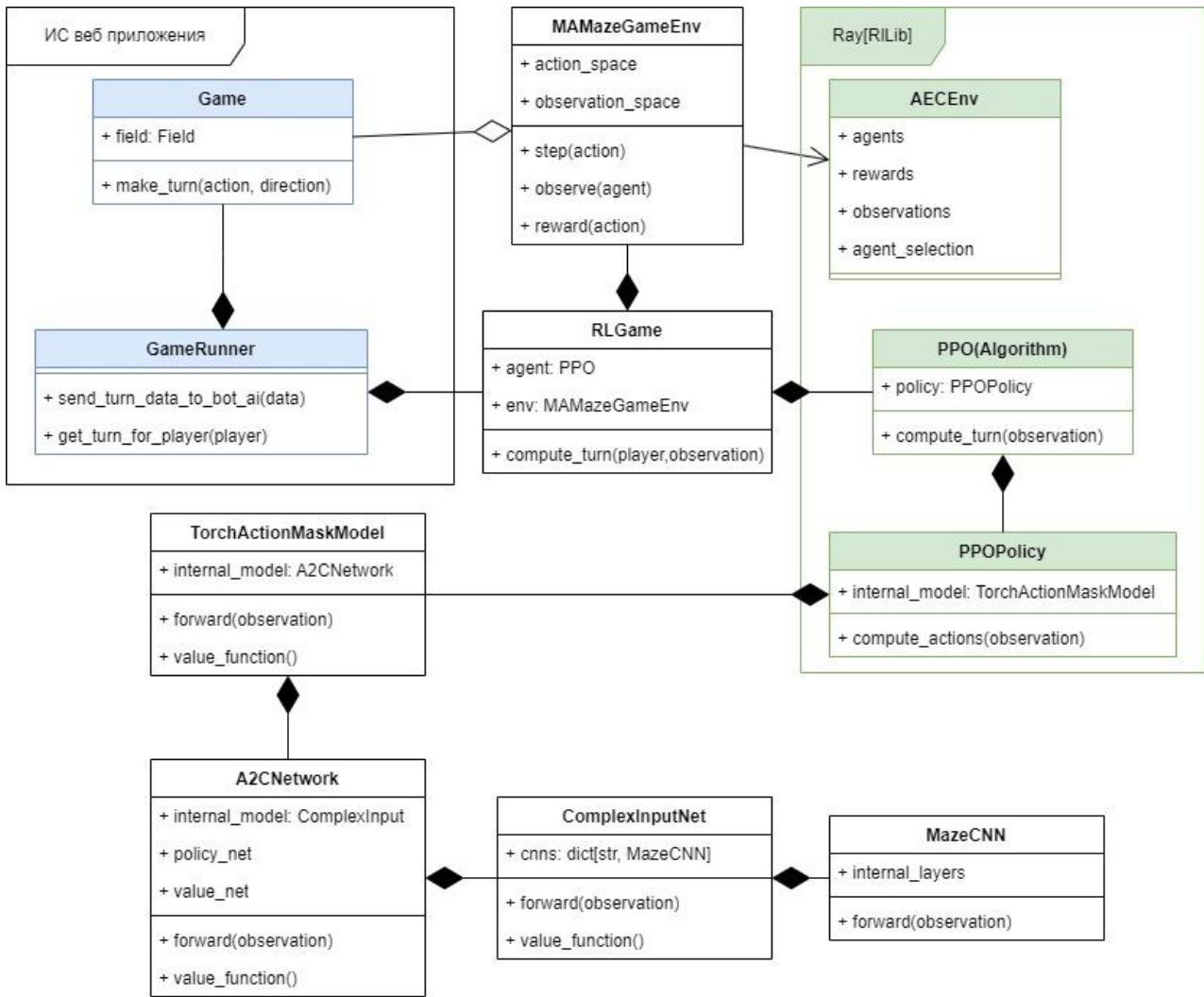


Рисунок 2.9 – Диаграмма классов подмодуля принятия решений

2.2. Интеграция разрабатываемого модуля с ИС веб-приложения

Поскольку данная ИС не является высоконагруженной, разрабатываемый модуль может взаимодействовать с ИС веб-приложения напрямую. Данные о ходах передаются в модуль принятия решений, а итоговый результат – действие интеллектуального агента, – передается обратно в ИС для дальнейшей обработки. Взаимосвязь компонентов ИС и разрабатываемого модуля представлена на рисунке 2.10:

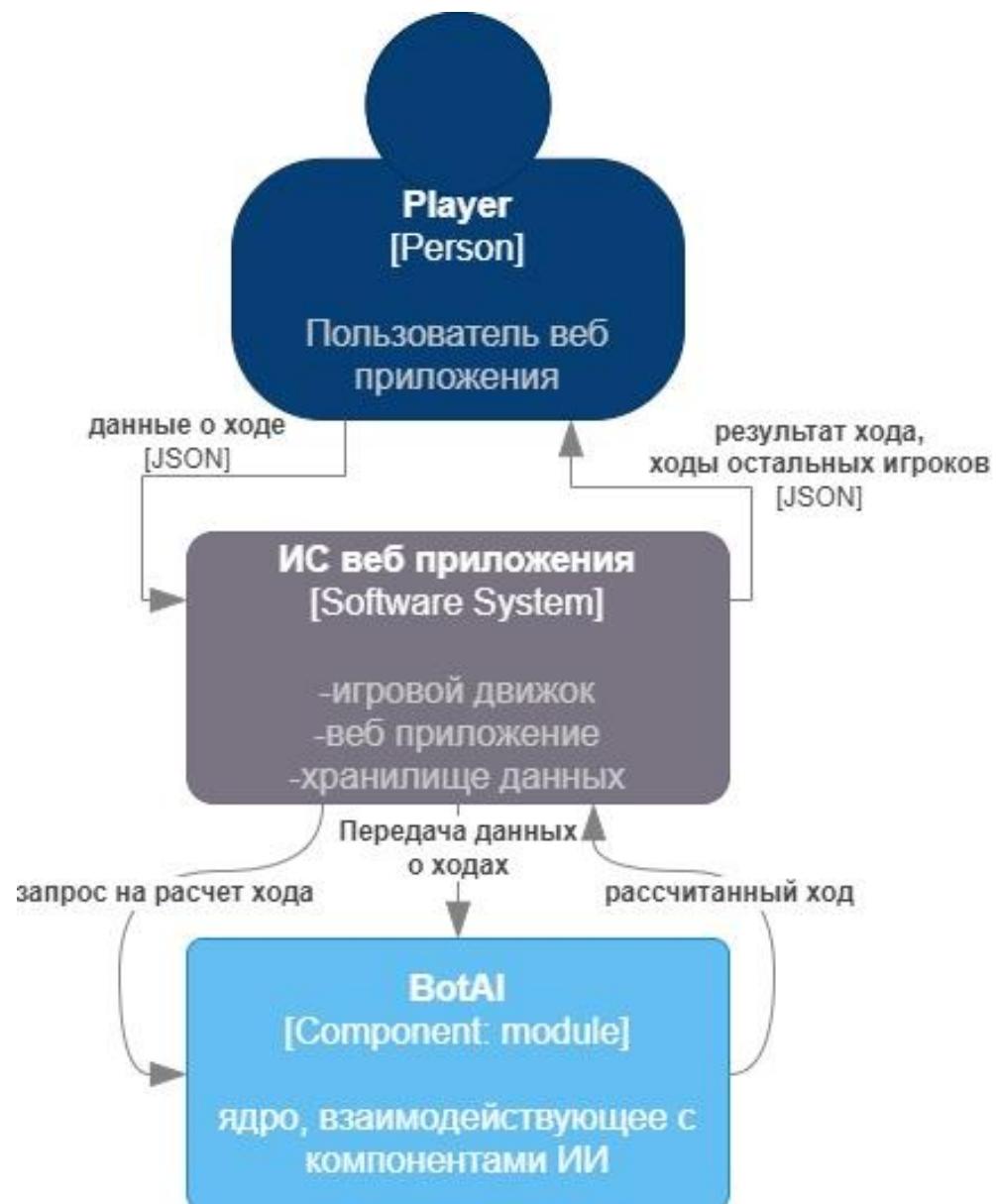


Рисунок 2.10 – Схема основных процессов при обработке ходов в системе

На рисунке 2.11 приведен пример партии игры с противниками под управлением ИИ.

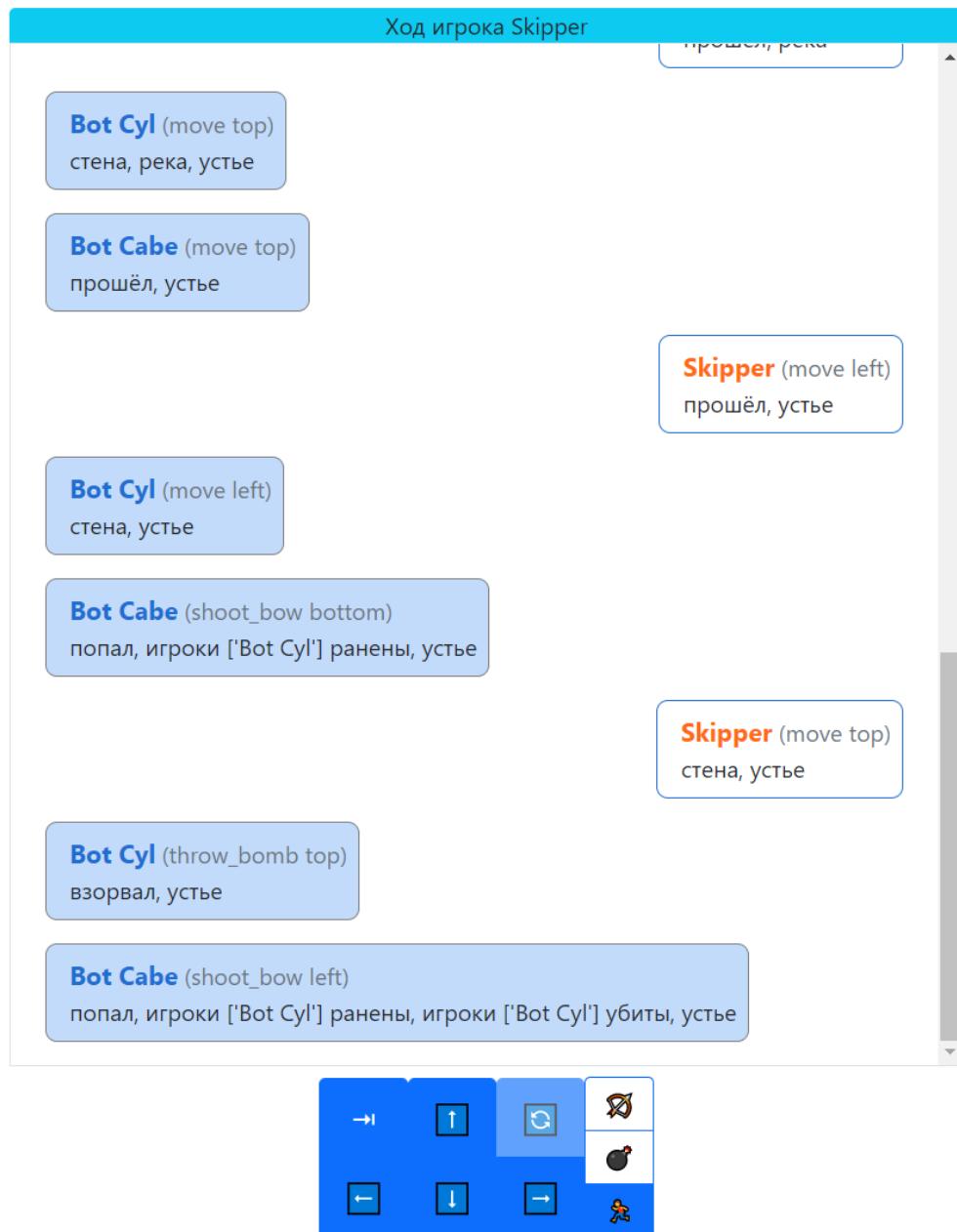


Рисунок 2.11 – Пример игровой партии против ИИ

2.3. Схема основных процессов анализа состояния окружения и принятия решений

Во время игровой партии люди совершают ходы, данные которых передаются на обработку в модуль принятия решений для сбора сведений об уровне. Когда ход переходит персонажу под управлением ИИ, ИС система приложения отправляет запрос на расчет хода модулю принятия решений. При обработке данного запроса рассчитывается ход для указанного персонажа и результат возвращается в ИС (рисунок 2.12).

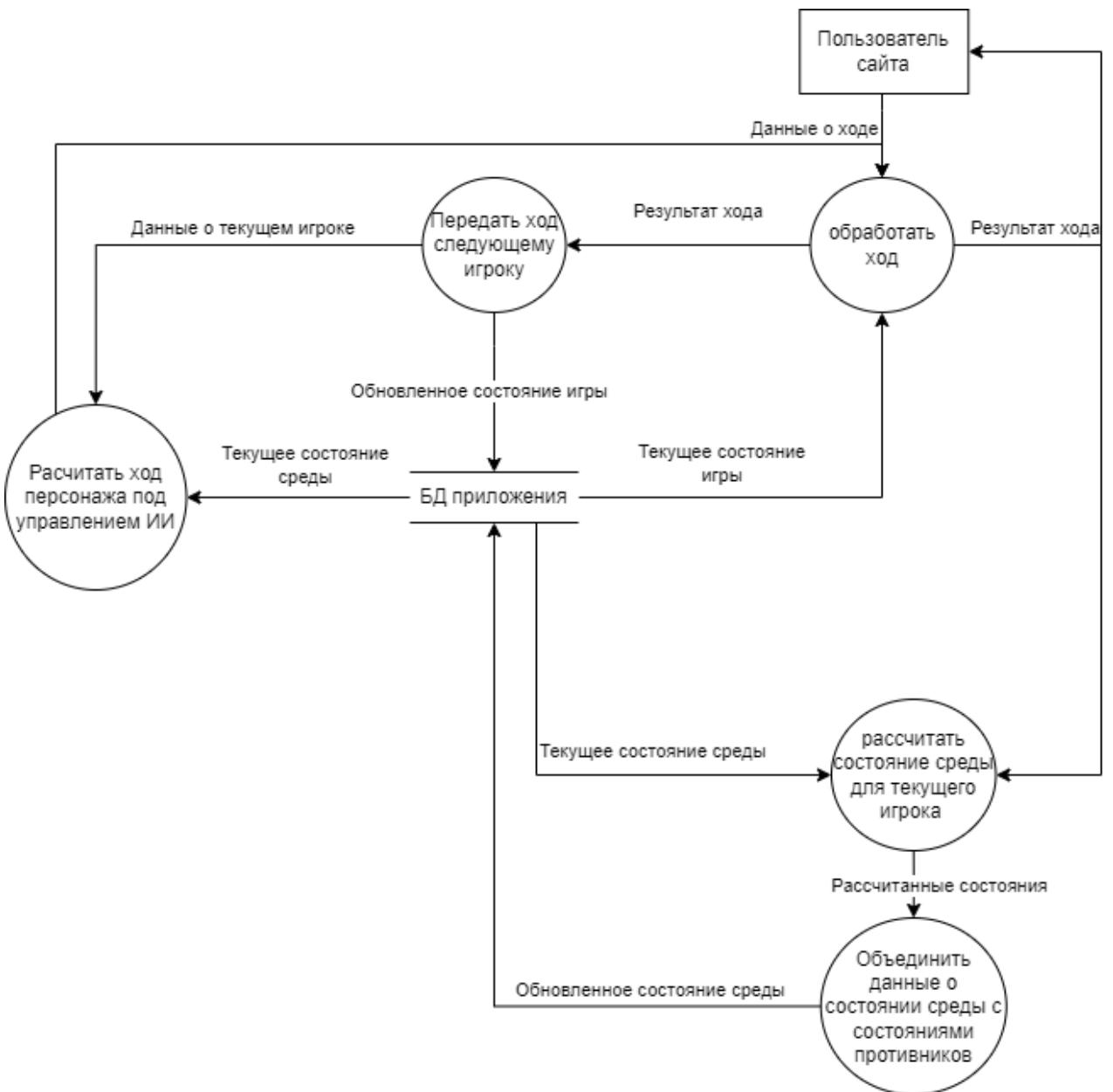


Рисунок 2.12 – Схема основных процессов при расчете ходов

3. Технологический раздел

3.1. Реализация модуля принятия решений

3.1.1. Реализация подмодуля, использующего дерево принятия решений

Модуль реализован на языке программирования Python поскольку текущая информационная система веб-приложения реализована на данном языке, а также вследствие поддержки множества библиотек, включающих алгоритмы для создания игрового интеллектуального агента.

На вход модулю передаются данные о ходах игроков и ответах «ведущего» в виде словаря, имеющего следующие поля:

- player_name: имя текущего игрока;
- action: действие игрока;
- direction: направление действия;
- type_out_treasure: тип вынесенного клада | None;
- cell_treasures_amount: число кладов в клетке игрока;
- had_treasure: был ли клад до подбора нового;
- hit: успешность выстрела;
- dmg_pls: список раненых игроков;
- dead_pls: список убитых игроков;
- drop_pls: список игроков, выронивших клад;
- destroyed: была ли взорвана стена;
- wall_passed: прошел ли в соседнюю клетку;
- wall_type: тип стены в которую ударился;
- diff_cells: прошел ли больше чем на 1 клетку;
- type_cell_after_wall_check: тип соседней клетки;
- type_cell_at_end_of_turn: тип клетки игрока в конце хода.

На основе этих данных производится симуляция среды, результатом чего является множество возможных вариантов карты и расположения агентов персонализированное для каждого агента.

Процесс принятия решения состоит из следующих этапов:

- Карта, построенная в процессе симуляции, интерпретируется в виде графа переходов между клетками (псевдокод данного алгоритма приведен в приложении А).
- Для каждого узла графа вычисляется расстояние в ходах и кратчайший путь из текущего узла;
- На основе полученного графа и текущего состояния игрока рассчитывается ценность для каждой клетки – то насколько агент стремиться в нее попасть. На ценность клетки влияют: тип клетки, наличие клада в руках, текущий запас здоровья, стрел и бомб, число ходов до данной клетки, наличие в клетке клада;
- Затем выбирается наиболее ценная клетка и агент выбирает кратчайший маршрут к ней;
- Данное действие повторяется для всех возможных состояний среды.
- Итоговым выбранным действием агента является наиболее часто выбранное первое действие среди всех выбранных маршрутов.

На рисунке 3.1 приведен построенный граф переходов по карте для одного из состояний среды и результат расчета ценности клеток.

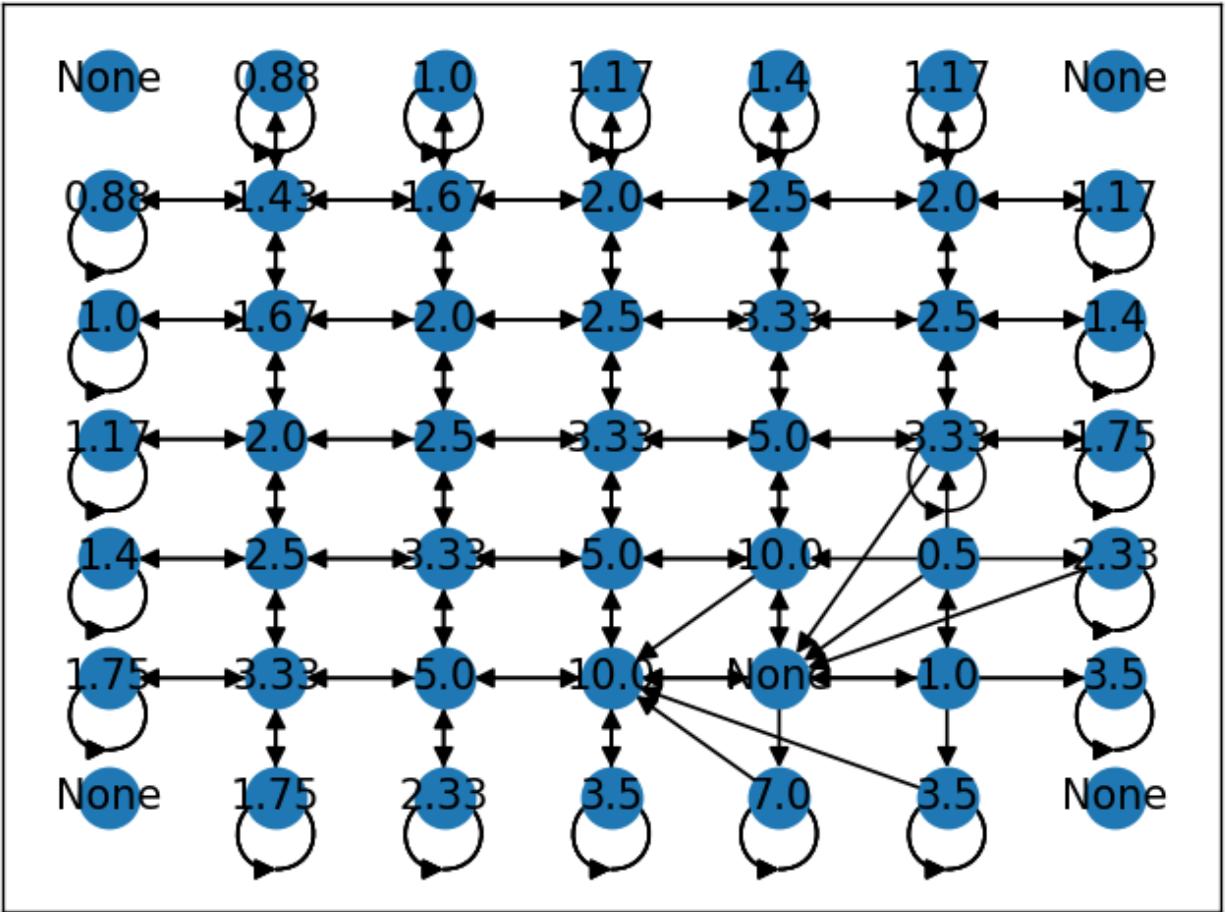


Рисунок 3.1 – Результат построения графа переходов по карте и расчета ценности клеток

3.1.2. Реализация подмодуля, использующего обучение с подкреплением

На вход модулю поступает текущее состояние среды, полученное от ИС. Для работы нейронной сети производится преобразование входных данных следующим образом:

- Клетки уровня кодируются с помощью унитарного кода: поскольку карта содержит клетки 11 типов, то вектор признаков, описывающих клетки имеет размер $11 \times N \times N$, где N – размер карты.
- Данные о стенах представляются вектором размерности $12 \times 2 \times 7 \times 7$. Каждую стену можно описать 3 бинарными признаками, для каждой клетки определены 4 стены – 12 признаков для стен одной клетки. Сохраняются данные за 2 последних наблюдения.

- Данные о сокровищах (объект карты) преобразуются в вектор размерности $1 \times N \times N$, где элемент $1*i*j$ – отражает число сокровищ в данной клетке.
- Характеристики текущего игрока представляются вектором размерности 6×2 (здоровье, относительный запас стрел / бомб, наличие клада в руках, координаты x / y). Сохраняются данные за 2 последних наблюдения.
- Аналогично данные о характеристиках оппонентов представляются вектором признаков размерности $K \times 6 \times 2$, где $K =$ (число игроков – 1).
- Данные о доступных агенту действиях представляются в виде вектора 13×1 .

Преобразованные данные о состоянии среды передаются модели, на выходе которой находится вероятностное распределение всех действий. Наиболее вероятное действие передается обратно в ИС для дальнейшей обработки и отображения пользователю.

Модель обрабатывает признаки следующим образом. Для каждого вида признака, полученный вектор проходит через отдельную глубокую сверточную нейронную сеть. Результаты свертки объединяются.

Полученный вектор передается на вход следующим внутренним моделям:

- Модель, содержащая полносвязную сеть из 5ти слоев, выходом являются категориальные признаки – вероятность выбора каждого действия, к данным вероятностям применяется маска, отражающая допустимость действия.
- Модель, содержащая полносвязную сеть из 5ти слоев, выходом является значения сети-критика, которое впоследствии используется РРО алгоритмом для обучения.

Модель реализована на фреймворке PyTorch. За этап обучения отвечает фреймворк Ray[RLLib], который включает в себя реализацию алгоритма РРО и методы для оптимизации гиперпараметров модели. На рисунке 3.2 приведена архитектура данной модели (полный перечень слоев нейронных сетей приведен в приложении Б).

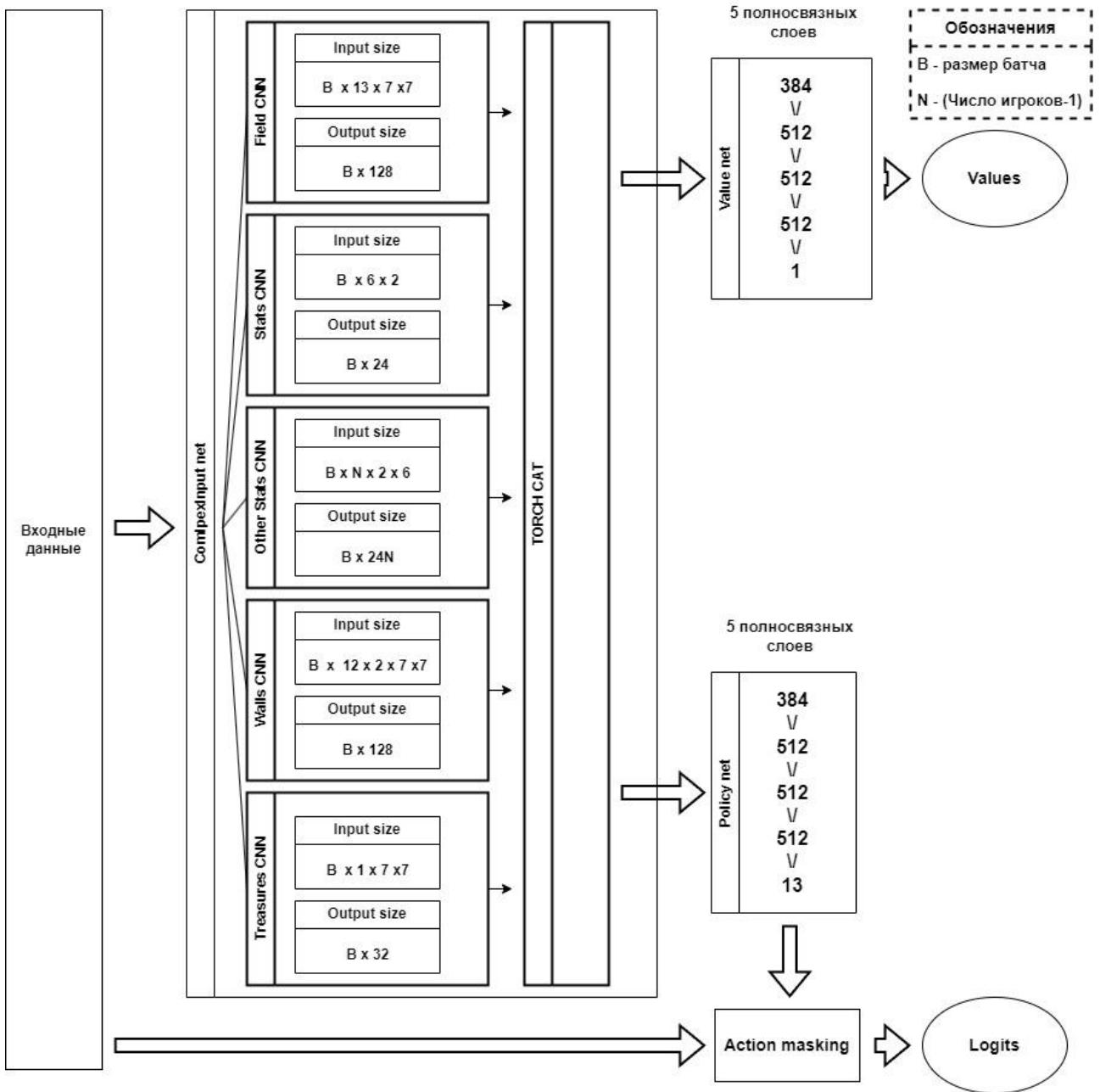


Рисунок 3.2 – Архитектура модели

Для обучения модели создан класс `MAMazeGameEnv`. Данный класс позволяет нескольким агентам выполнять шаги в игровой среде и получать награду за действия. Агент получает награду за ход в 2x случаях:

- при совершении победного действия $R = R_{\text{оставшаяся}}$;
- при убийстве противника (3.1).

$$R = \frac{\text{число убитых выстрелом}}{\text{число живых противников}} * \frac{R_{\text{оставшаяся}}}{3} \quad (3.1)$$

При инициализации партии устанавливаются доступные агентам награды: $R_{\text{оставшаяся}} = 1$. При получении агентом награды, доступная ему оставшаяся награда обновляется. Таким образом возможная награда агента за игру принадлежит диапазону $[0; 1]$. На рисунке 3.3 приведен график распределения полученных агентом наград во время обучения:

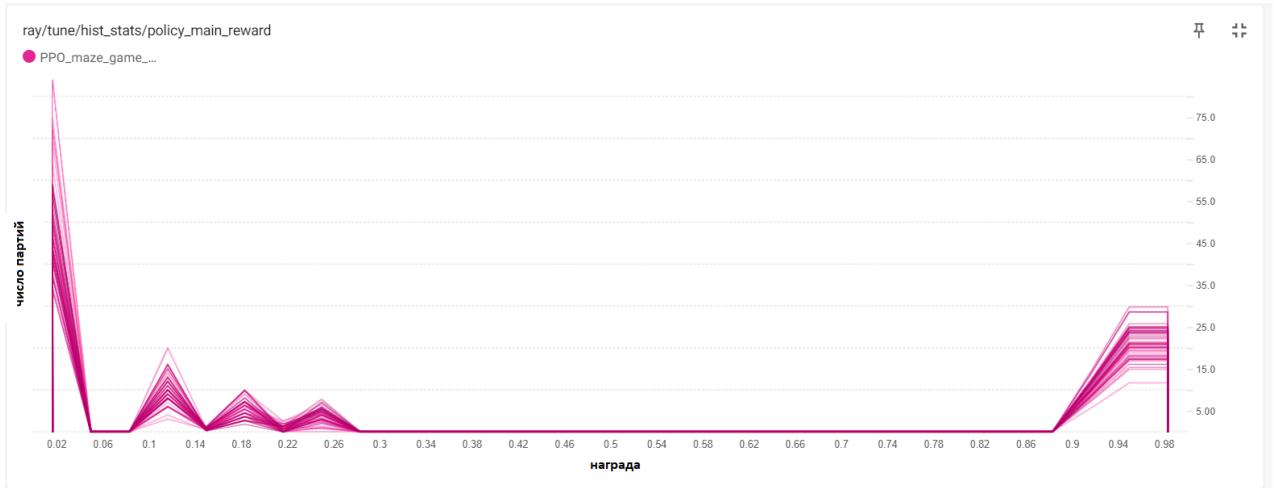


Рисунок 3.3 – Распределение полученных агентом наград за партии

Обучение модели производится следующим образом:

- Создается 4 политики – одна содержит в себе модель с весами проинициализированными случайным образом (основная), остальные выдают случайное допустимое действие;
- Политика, содержащая в себе модель, обучается до тех пор, пока вероятность ее победы не превысит 0.65;
- После достижения заданной вероятности, «случайные» политики заменяются копией «основной»;
- Затем цикл обучения повторяется.

Для предотвращения ситуации, когда ходящим первым игроком управляет одна и та же политика, политика ассоциируется с агентом случайным образом после итерации обучения.

На рисунках 3.4 – 3.6 приведена статистика обучения модели за 10^7 шагов (процент побед, средняя длительность игры, количество различных моделей поведения противников):

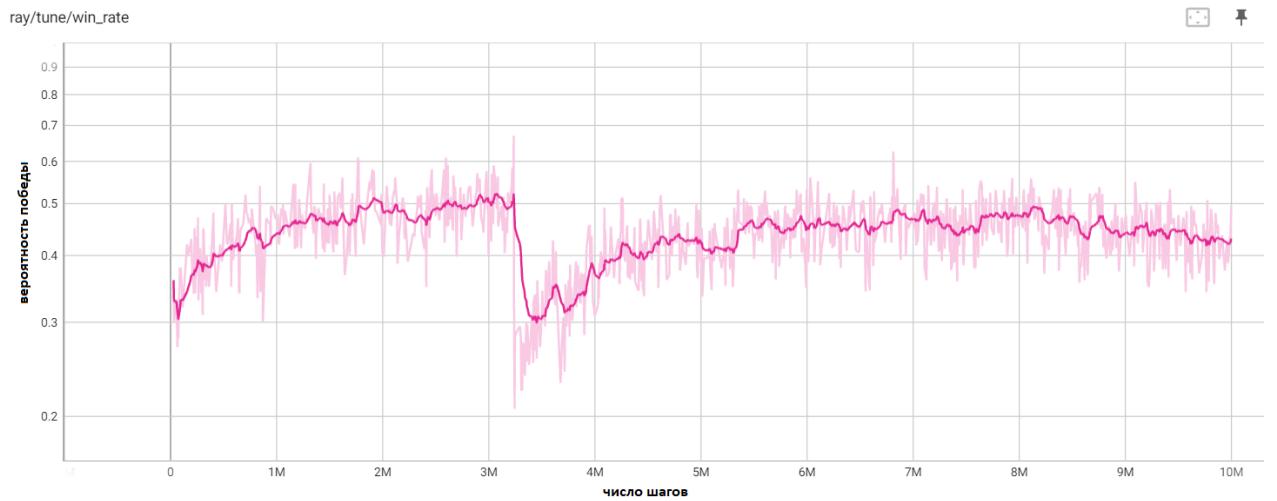


Рисунок 3.4 – Изменение процента побед, при обучении

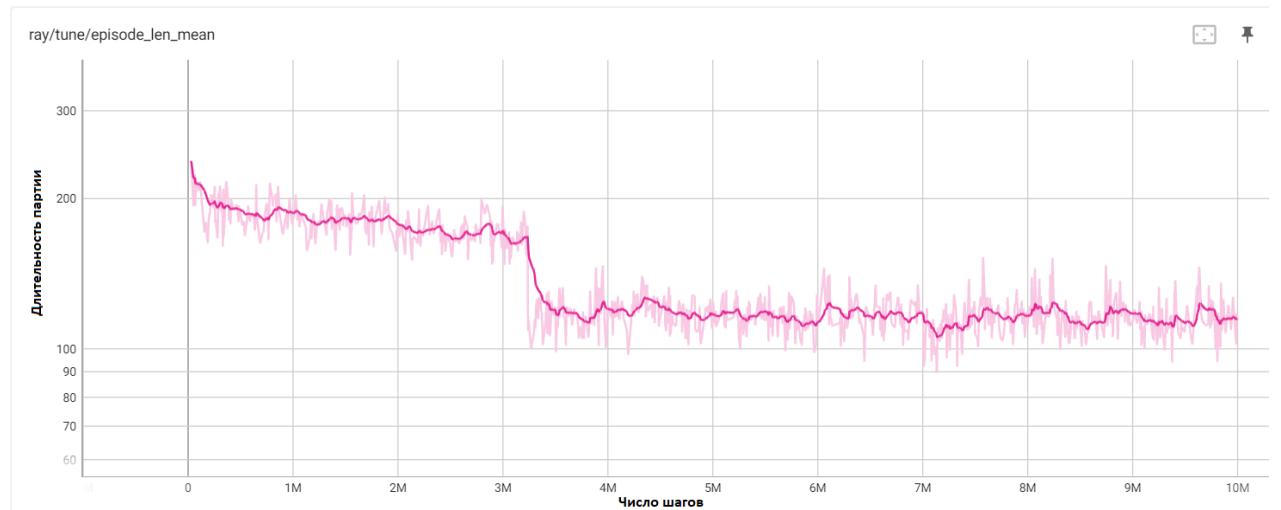


Рисунок 3.5 – Изменение продолжительности партии, при обучении

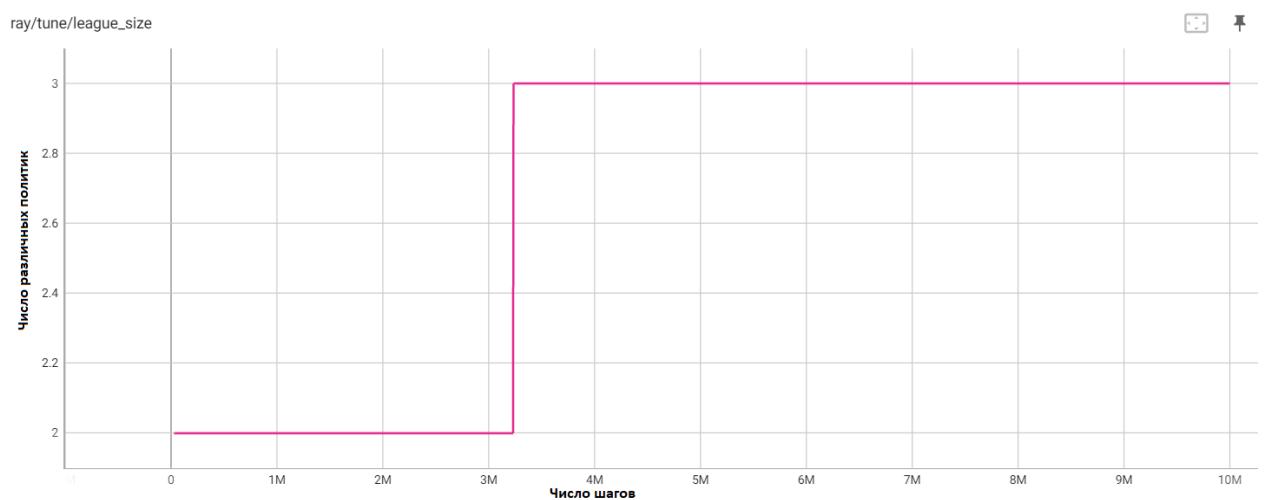


Рисунок 3.6 – Изменение числа различных моделей поведения агентов, при обучении

3.1.3. Интеграция с ИС веб приложения

В БД информационной системы добавлена таблица, содержащая поле для бинарного представления объекта, хранящего состояния среды, полученные в процессе симуляции. При совершении игроками ходов, данный объект десериализуется, обновляет состояние симуляции, в случае необходимости совершения ботом хода, данный ход также рассчитывается. Затем объект сохраняется в обновленном состоянии в БД.

3.2. Тестирование модуля принятия решений

Тестирование корректности работы алгоритма (дерево принятия решений):

Таблица 3.1 – Результат тестирования алгоритма

| № | Число сыгранных игр | Число сбоев | Число корректно завершенных игр |
|---|---------------------|-------------|---------------------------------|
| 1 | 2 | 3 | 4 |
| 1 | 100 | 0 | 100 |
| 2 | 100 | 1 | 100 |
| 3 | 100 | 0 | 100 |
| 4 | 100 | 0 | 100 |
| 5 | 100 | 1 | 100 |

Из 500 партий, в 2х произошел сбой симуляции карты. После сбоя выполнение алгоритма продолжилось корректно. На рисунке 3.7 представлена часть ходов одной из партий.

```
seed: 0.8581690722902962
{'player_name': 'Skipper', 'action': 'info', 'direction': ''} река
{'player_name': 'Tester1', 'action': 'info', 'direction': ''} суша
{'player_name': 'Tester2', 'action': 'info', 'direction': ''} река
{'player_name': 'Tester3', 'action': 'info', 'direction': ''} река
{'player_name': 'Skipper', 'action': 'move', 'direction': 'top'} стена, река, река
{'player_name': 'Tester1', 'action': 'move', 'direction': 'right'} прошёл, Склад оружия, запас стрел восстановлен
{'player_name': 'Tester2', 'action': 'move', 'direction': 'right'} прошёл, суша, клад (1шт.)
{'player_name': 'Tester3', 'action': 'move', 'direction': 'top'} стена, река, река
{'player_name': 'Skipper', 'action': 'move', 'direction': 'top'} прошёл, река, река
{'player_name': 'Tester1', 'action': 'move', 'direction': 'bottom'} прошёл, суша
{'player_name': 'Tester2', 'action': 'swap_treasure', 'direction': ''} подобрал клад
{'player_name': 'Tester2', 'action': 'move', 'direction': 'top'} стена, суша
{'player_name': 'Tester3', 'action': 'move', 'direction': 'top'} прошёл, река, река
{'player_name': 'Skipper', 'action': 'move', 'direction': 'top'} прошёл, река
```

Рисунок 3.7 – Текстовое представление данных о ходах игроков

На рисунке 3.8 приведен стек вызовов наиболее ресурсоемких функций, задействованных в симуляции среды и расчете действий агента.



Рисунок 3.8 – Стек вызовов функций при симуляции среды и расчете действия

Исходя из данных, отображенных на рисунке 3.8 можно сделать вывод что наибольшее время занимает расчет изменения состояния среды после хода игроков.

Тестирование работы бота (обучение с подкреплением):

Таблица 3.2 – Результат тестирования алгоритма

| № | Число сыгранных игр | Число сбоев | Число корректно завершенных игр |
|---|---------------------|-------------|---------------------------------|
| 1 | 2 | 3 | 4 |
| 1 | 100 | 0 | 99 |
| 2 | 100 | 0 | 100 |
| 3 | 100 | 0 | 99 |
| 4 | 100 | 0 | 100 |
| 5 | 100 | 0 | 100 |

Из 500 сыгранных партий 2 закончились ничьей по причине превышения максимального числа ходов. Это связано с тем, что в некоторых случаях агент не имеет возможности выиграть из-за специфической конфигурации игрового уровня.

На рисунке 3.9 приведен результат тестирования работы веб приложения с подключенным модулем принятия решений интеллектуальным агентом.

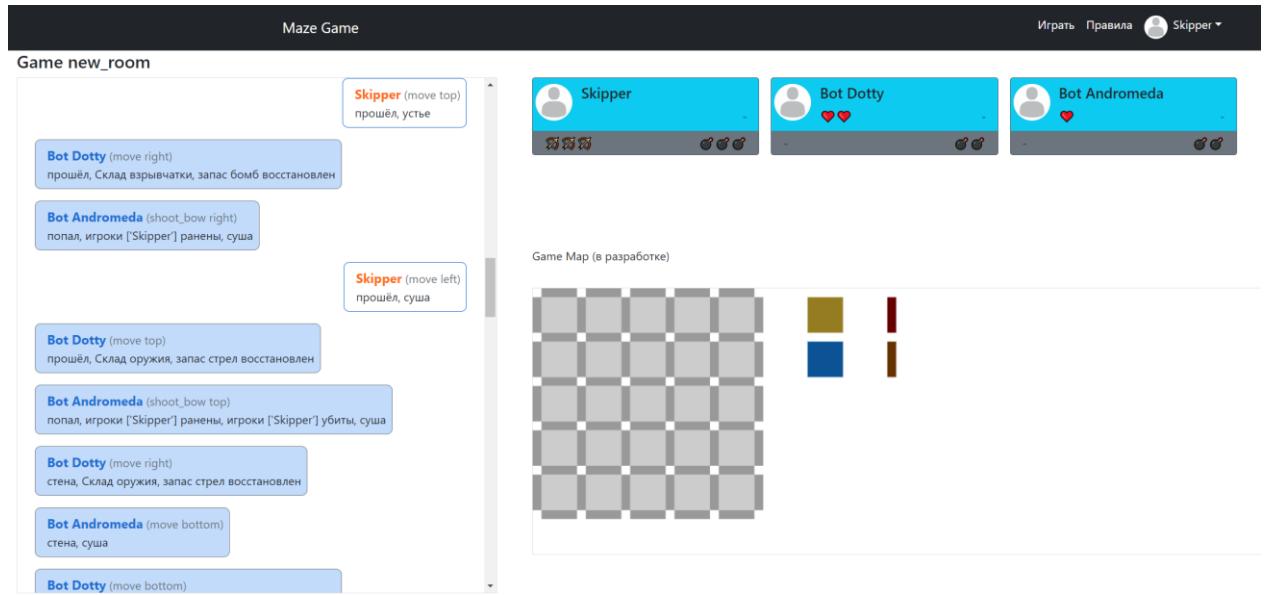


Рисунок 3.9 – Интерфейс веб приложения в партии против противников под управлением ИИ

3.3. Анализ метрик работы модуля принятия решений

Метрики работы алгоритмов были собраны для 100 запусков случайных карт со случайными местами появления агентов. Для каждой метрики оценивается ее среднее, максимальное и минимальное значение для данной выборки. Сравниваются следующие метрики:

- Число ходов до победы – отражает продолжительность партии;
- Число ходов до взятия клада – отражает стремление агента к исследованию карты и подбору клада;
- Среднее время, затраченное на ход – среднее время расчета хода для каждой партии;
- Максимальное время, затраченное на ход – максимальное время расчета хода для каждой партии;

- Минимальное время, затраченное на ход – минимальное время расчета хода для каждой партии;
- Вероятность попадания из лука – отражает точность стрельбы агентов.

В таблице 3.3 приведено сравнение данных метрик:

Таблица 3.3 – Сравнение метрик работы алгоритмов

| Метрика | Дерево принятия решений | | | Обучение с подкреплением | | |
|-----------------------------|-------------------------|----------|----------|--------------------------|----------|----------|
| | max | min | mean | max | min | mean |
| число ходов до победы | 123 | 10 | 58.57 | 356 | 8 | 110.58 |
| Число ходов до взятия клада | 76 | 0 | 15.67 | 62 | 0 | 12.57 |
| Mean времени на ход | 1.863758 | 0.003527 | 0.136759 | 0.007215 | 0.001028 | 0.003305 |
| Max времени на ход | 34.768184 | 0.052031 | 3.496378 | 0.062518 | 0.001999 | 0.005696 |
| Min времени на ход | 0.005968 | 0.0 | 0.00322 | 0.003001 | 0.000962 | 0.002643 |
| Процент попаданий из лука | $571 / 819 = 0.68$ | | | $475 / 1951 = 0.24$ | | |

Также было произведено сравнение данных алгоритмов при игре между собой. Для 500 сыгранных партий между двумя агентами (за поведение одного агента отвечает дерево принятия решений, другого – обученная модель) результат следующий:

- Дерево принятия решений – 63% побед
- Обучение с подкреплением – 37% побед

На рисунке 3.10 приведена диаграмма отображающая число ходов, сделанное игроком до взятия клада.

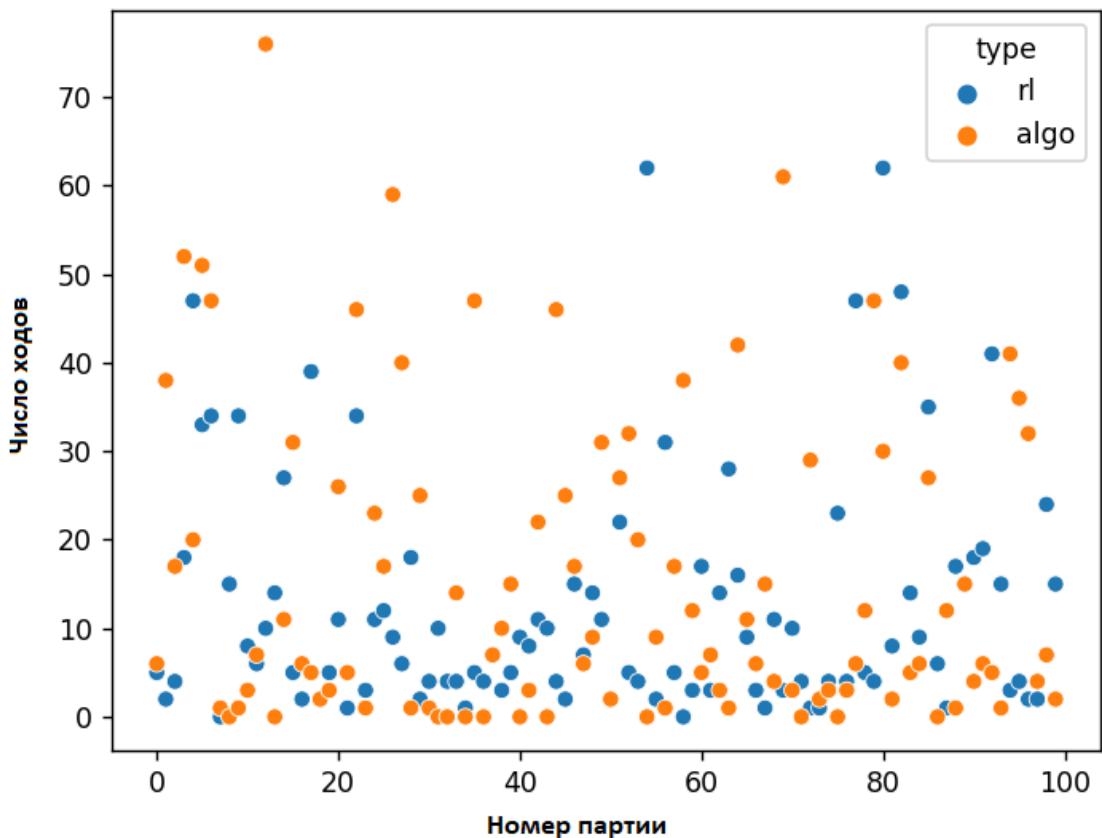


Рисунок 3.10 – Число ходов до взятия клада

На рисунке 3.11 приведена диаграмма отображающая число ходов, сделанное игроком до победы.

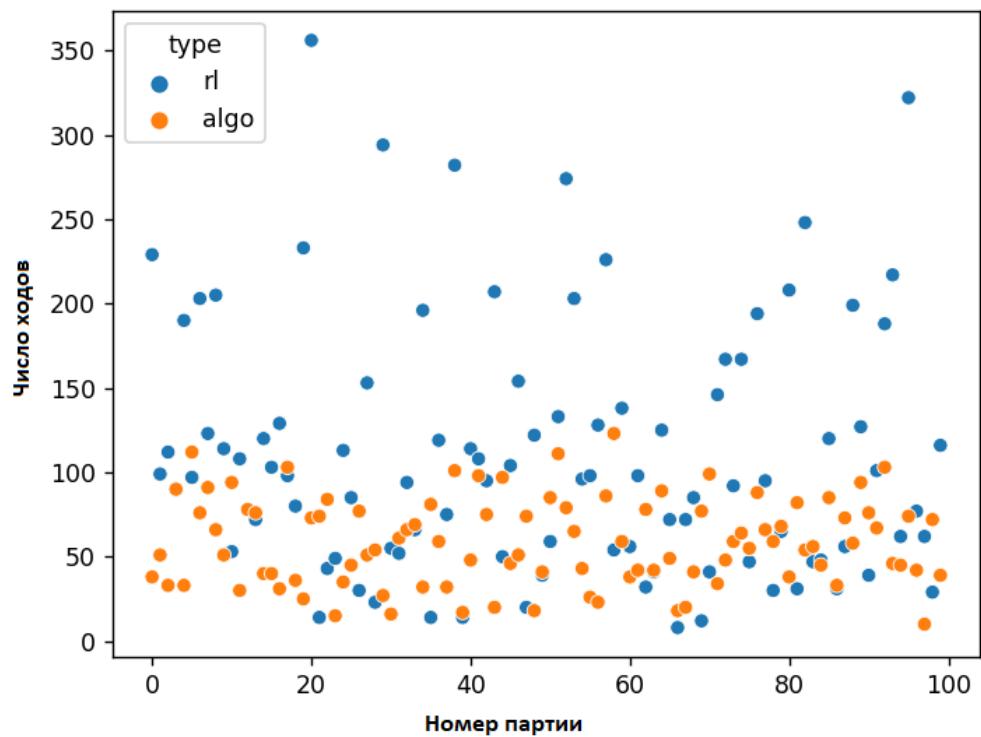


Рисунок 3.11 – Число ходов до победы

На рисунке 3.12 приведена диаграмма отображающая распределение затраченного времени на ход.

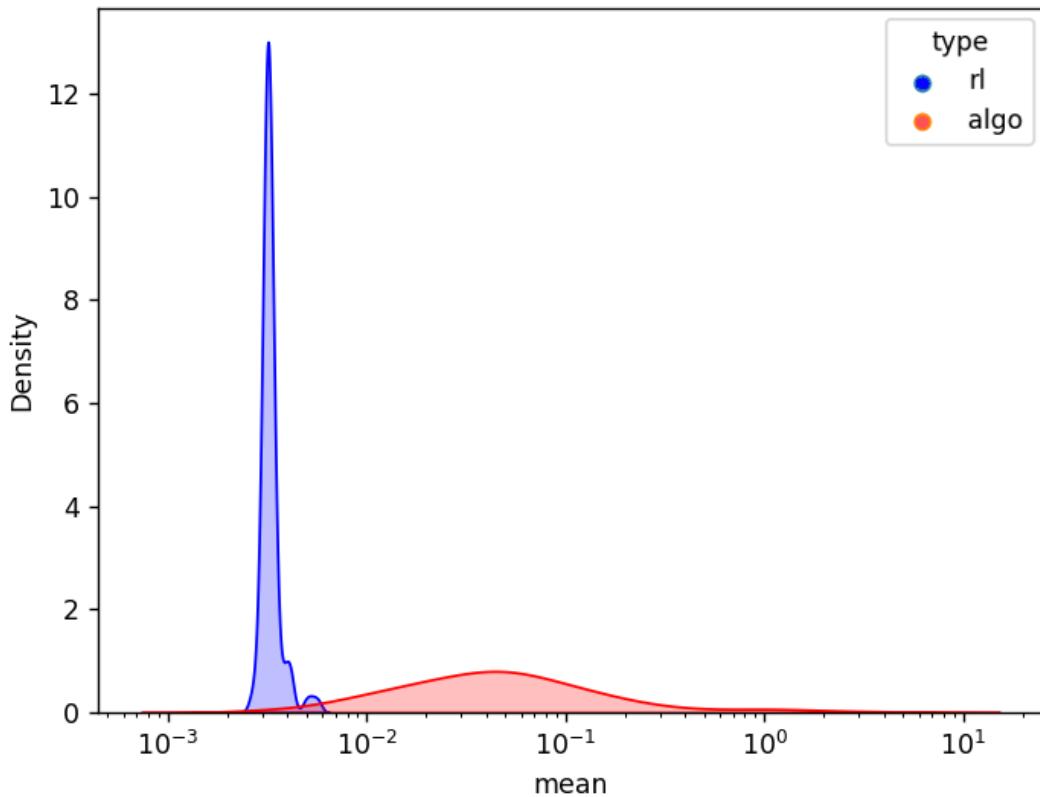


Рисунок 3.12 – Распределение среднего времени, затраченного на ход

Полученные метрики показывают, что алгоритм, основанный на дереве принятия решений (алгоритмическая реализация), позволяет точнее выбирать действие для наиболее быстрой победы в игре по сравнению с алгоритмом, основанном на обучении с подкреплением. При дальнейшем обучении модели, возможно, алгоритм с обучением сможет превзойти алгоритм с деревом решений.

Анализ метрик, связанных с затраченным временем на ход показывает, что быстродействие алгоритмической реализации зависит от конфигурации игрового уровня. В случае наличия на карте большого числа рек появляется множество возможных состояний среды, что приводит к существенной потере производительности. При этом даже в случае «простой» конфигурации игрового уровня, расчет хода все еще занимает больше времени по сравнению с алгоритмом, использующим обученную модель.

Визуально анализируя действия агентов непосредственно во время партии можно сказать, что:

- Агент, принимающий решения с помощью обученной модели, совершает много беспорядочных действий (например, движение туда-сюда), но возможно, это может быть связано с тем, что агент пытается избежать конфликта с оппонентом и не намерен двигаться в его сторону, либо с недостаточной обученностью модели и не оптимальностью ее архитектуры.
- Агент, принимающий решения используя дерево принятия решений, в некоторых случаях загоняет себя в «ловушку», что может быть связано с недостаточной сложностью дерева решений, либо не оптимальными критериями в его узлах.

4. Экономический раздел

4.1. Организация и планирование работ по теме

В составе работы задействовано 3 человека:

- Руководитель – отвечает за грамотную постановку задачи, контролирует отдельные этапы работы, вносит необходимые корректизы и оценивает выполненную работу в целом;
- Консультант – отвечает за консультирование экономической части выпускной квалификационной работы;
- Разработчик – реализация всех поставленных задач, в том числе проведение тестирования готового продукта и подготовка проектной документации.

Состав задействованных в работе участников представлен на рисунке 4.1.

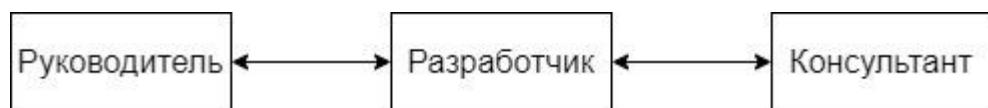


Рисунок 4.1 – Состав участников работы

4.1.1. Организация работ.

На разработку отводится 90 рабочих дней. Этапы разработки представлены в таблице 4.1:

Таблица 4.1 – Этапы разработки

| № | Название этапа | Исполнитель | Трудоемкость, чел/дни | Продолжительность работ, дни |
|-----|---|--------------|-----------------------|------------------------------|
| 1 | Разработка и утверждение технического задания | Руководитель | 2 | 3 |
| | | Разработчик | 3 | |
| 2 | Технические предложения | Руководитель | 5 | 5 |
| | | Консультант | 1 | |
| | | Разработчик | 5 | |
| 3 | Эскизный проект | | | |
| 3.1 | Анализ исходных данных и требований | Разработчик | 9 | |
| 3.2 | Постановка задачи | Руководитель | 1 | 16 |
| 3.3 | Разработка общего описания алгоритма функционирования | Руководитель | 1 | |
| | | Разработчик | 7 | |

Продолжение таблицы 4.1

| | | | | | |
|--------------|--|--------------|-----------|-----------|--|
| 4 | Технический проект | | | | |
| 4.1 | Определение формы представления входных и выходных данных | Разработчик | 3 | 13 | |
| 4.2 | Разработка структуры программы | Руководитель | 2 | | |
| | | Разработчик | 10 | | |
| 5 | Рабочий проект | | | 53 | |
| 5.1 | Программирование и отладка программы | Разработчик | 40 | | |
| 5.2 | Испытание программы | Разработчик | 4 | | |
| 5.3 | Корректировка программы по результатам испытаний | Разработчик | 5 | | |
| 5.4 | Подготовка технической документации на программный продукт | Консультант | 1 | | |
| | | Разработчик | 2 | | |
| 5.5 | Сдача готового продукта и внедрение | Руководитель | 2 | | |
| | | Консультант | 1 | | |
| | | Разработчик | 2 | | |
| Итого | | | 90 | | |

4.1.2. График проведения работ:

Календарный график исполнения работы представлен на рисунке 4.2. Из рисунка 4.2 так же видно, что общий срок разработки составит 90 дней.

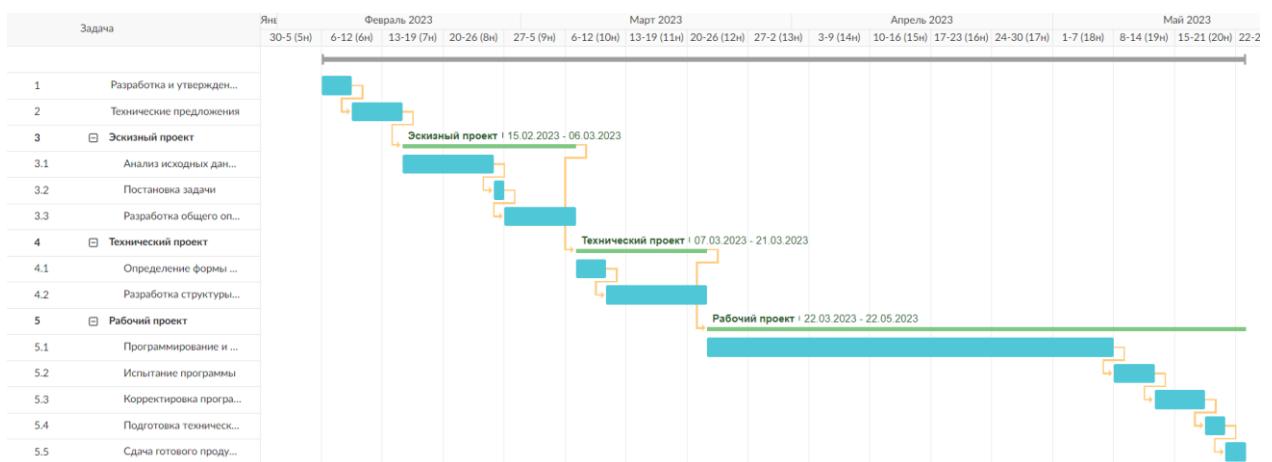


Рисунок 4.2 – Календарный график работ

4.2. Расчет стоимости проведения работ по теме

В выпускной квалификационной работе объем затрат на НИР и ОКР был проведен методом калькулирования.

1 статья «Материалы, покупные изделия и полуфабрикаты».

Расчет затрат по данной статье приведен в таблице 4.2.

Таблица 4.2 – Затраты по 1ой статье.

| № пп | Наименование материалов | Единицы измерения | Количество | Цена за единицу (руб) | Стоимость (руб) |
|--|-------------------------|-------------------|------------|-----------------------|-----------------|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | Бумага А 4 | пачка | 1 | 175 | 175 |
| 2 | Картридж для принтера | шт | 1 | 2350 | 2350 |
| 4 | Ручка | шт | 10 | 15 | 150 |
| 5 | Карандаш | шт | 10 | 7 | 70 |
| Итого материалов | | | | | 2745 |
| Транспортно-заготовительные расходы | | | | | 412 |
| Итого | | | | | 3157 |

2 статья «Специальное оборудование».

Расходы на специальное оборудование отсутствуют.

3 статья «Основная заработная плата».

Расчет основной заработанной платы приведен в таблице 4.3.

Таблица 4.3 – Затраты по 3ой статье.

| № пп | Наименование этапа | Исполнитель (должность) | Мес. оклад (руб) | Трудоемкость (чел/дни) | Оплата за день (руб) | Оплата за этап (руб) |
|--------------|--------------------|-------------------------|------------------|------------------------|----------------------|----------------------|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | ТЗ | Руководитель | 40 000 | 2 | 1818 | 3636 |
| | | Разработчик | 29 000 | 3 | 1318 | 3954 |
| 2 | ТП | Руководитель | 40 000 | 5 | 1818 | 9090 |
| | | Консультант | 35 000 | 1 | 1591 | 1591 |
| | | Разработчик | 29 000 | 5 | 1318 | 6590 |
| 3 | Эскизный проект | Руководитель | 40 000 | 2 | 1818 | 3636 |
| | | Разработчик | 29 000 | 16 | 1318 | 21088 |
| 4 | Технический проект | Руководитель | 40 000 | 2 | 1818 | 3636 |
| | | Разработчик | 29 000 | 13 | 1318 | 17134 |
| 5 | Рабочий проект | Руководитель | 40 000 | 2 | 1818 | 3636 |
| | | Консультант | 35 000 | 2 | 1591 | 3182 |
| | | Разработчик | 29 000 | 53 | 1318 | 69854 |
| Итого | | | | | 147 027 | |

4 статья «Дополнительная заработная плата».

$\text{ДЗП} = 147027 \times 0,2 = 29405,4$ руб. Дополнительная заработная плата научного и производственного персонала составляет по проекту 29405,4 руб.

5 статья «Страховые отчисления».

Отчисления на социальные нужды составляют 30% от фонда оплаты труда (ФОТ), который состоит из основной и дополнительной заработной платы.

$$\text{ФОТ} = \text{ОЗП} + \text{ДЗП} = 147027 + 29405,4 = 176\ 432,4 \text{ руб.}$$

$$\text{СВ} = \text{ФОТ} \times 30\% = 176\ 432,4 * 0,30 = 52\ 929,72 \text{ руб.}$$

6 статья «Командировочные расходы».

Расходы по данному разделу отсутствуют.

7 статья «Контрагентские услуги».

В процессе разработки данного проекта услуги сторонних организаций не использовались.

8 статья «Накладные расходы».

$$\text{НР} = \text{ОЗП} \times 200\% = 147027 * 2,0 = 332\ 234 \text{ руб.}$$

9 статья «Прочие расходы».

Для обучения нейросетевой модели использовался один компьютер, время работы которого под нагрузкой составило 120 часов. Исходя из расчета оплаты 10 рублей за 1 час машинного времени, сумма составит $\text{ПР} = 120 * 10 = 1200$ руб.

Полная себестоимость проекта приведена в таблице 4.4.

Таблица 4.4 – Полная себестоимость проекта.

| № пп | Номенклатура статей расходов | Затраты (руб) |
|--------------|---|----------------------|
| 1 | 2 | 3 |
| 1 | Материалы, покупные изделия и полуфабрикаты (за вычетом отходов) | 3157 |
| 2 | Специальное оборудование для научных (экспериментальных) работ | - |
| 3 | Основная заработка плата научного и производственного персонала | 147027 |
| 4 | Дополнительная заработка плата научного и производственного персонала | 29405,4 |
| 5 | Страховые взносы в социальные фонды | 52 929,72 |
| 6 | Расходы на научные и производственные командировки | - |
| 7 | Оплата работ, выполненных сторонними организациями и предприятиями | - |
| 8 | Накладные расходы | 294 054 |
| 9 | Прочие прямые расходы | 1200 |
| Итого | | 527 773,12 |

На основании полученных данных произведен расчёт договорной цены.

Цена договорная = себестоимость + прибыль

$$\Pi = 527\ 773,12 * 30\% = 158\ 331,94 \text{ руб.}$$

Поскольку разработка ведется частным лицом с упрощенной системой налогообложения (УСН), то работы не облагаются налогом на добавленную стоимость (НДС).

Таким образом, договорная цена будет представлять собой:

$$\text{ДЦ} = C + \Pi = 616\ 930,52 + 185\ 079,16 = 686\ 105,06 \text{ руб.}$$

Заключение

В данной работе были рассмотрены различные методы создания интеллектуальных агентов для многопользовательских игровых веб-приложений. Было проведено сравнение алгоритмов, основанных на обучении с подкреплением и на дереве принятия решений, а также их реализация в пошаговой игре «Maze Game».

В результате анализа полученных метрик работы реализованных алгоритмов можно сделать вывод, что алгоритм на основе дерева принятия решений показывает себя лучше в поставленной задаче, поскольку качество принимаемых им решений существенно выше. Однако данный алгоритм в большей степени зависит от правил игры, и требует больше вычислительных ресурсов для расчета действия по сравнению с использованием обученной модели. Способ моделирования среды, использованный для работы реализованного алгоритма, оказался крайне ресурсоемким, что означает, что при увеличении числа пользователей веб приложения будет необходимо вносить изменения, направленные на повышение производительности данного алгоритма.

Алгоритм на основе обучения с подкреплением, позволяющий агенту самостоятельно учиться и принимать решения на основе опыта, в данной задаче оказался менее эффективным с точки зрения качества принимаемых решений. Данная проблема может быть вызвана недостаточным временем обучения модели или не оптимальностью ее архитектуры.

Таким образом, в данной работе было показано, что выбор метода создания интеллектуального агента зависит от конкретной задачи и ее условий. Кроме того, оба алгоритма были реализованы в виде модуля ИС «Maze Game», что позволило сравнить их эффективность в практическом применении.

In this work, various methods for creating intelligent agents for multiplayer gaming web applications were considered. A comparison was made between

algorithms based on reinforcement learning and decision tree, as well as their implementation in the turn-based game "Maze Game".

Based on the analysis of the obtained performance metrics of implemented algorithms, it can be concluded that the decision tree algorithm performs better in the given task, as its decision-making quality is significantly higher. However, this algorithm depends more on the game rules and requires more computational resources to calculate action compared to using a trained model. The modelling approach used for the implemented algorithm proved to be extremely resource-intensive, which means that as the number of users of the web application increases, changes aimed at improving the performance of this approach will be necessary.

The reinforcement learning algorithm, which allows the agent to learn and make decisions based on experience, was found to be less effective in terms of the quality of decisions made in this task. This problem may be caused by insufficient training time for the model or suboptimal architecture.

Thus, this work has shown that the choice of intelligent agent creation method depends on the specific task and its conditions. In addition, both algorithms were implemented as a module in the "Maze Game" AI system, which allowed their effectiveness to be compared in practical applications.

Список использованных источников

1. Оригинальная настольная игра – [Электронный ресурс] – URL: <https://www.thegamecrafter.com/games/terra-incognita> (дата обращения 11.06.2021)
2. The Guide to Game AI – [Электронный ресурс] – URL: <https://www.gamedev.net/articles/programming/artificial-intelligence/the-total-beginners-guide-to-game-ai-r4942/> (дата обращения 16.02.2023)
3. Петер Флах. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. 2015. – С. 25-60.
4. Richard S. Sutton, Andrew G. Barto. Reinforcement Learning: An Introduction. 2015. Глава 1.
5. Левитин А. В. Алгоритмы: Введение в разработку и анализ – М.: Вильямс, 2006. – С. 409–417.
6. Зязюлькин С. П., Нестеренков С. Н. Использование actor-critic алгоритмов при обучении агентов для игр на Atrati 2600. 2020.
7. John Schulman, Filip Wolski, Prafulla Dhariwal. Proximal Policy Optimization Algorithms. 2017.
8. Томас Х. Кормен, Чарльз И. Лейзерсон. Алгоритмы: построение и анализ. – 2-е изд. – М.: «Вильямс», 2006. – С. 1296.
9. Экономика предприятия [Электронный ресурс]: учебно-методическое пособие / И.А. Назарова, А.С. Вихрова. – М.: РТУ МИРЭА, 2021. – С. 71.

Приложение А

Алгоритм построения графа переходов по карте

Листинг А.1 – Алгоритм построения графа переходов по карте, в виде псевдокода

```
build_from_map()
для каждого узла графа:
    если узел == None:
        продолжить;
    для каждого направления:
        если получить_стену(узел, направление) != None:
            соседний_узел = узел;
        иначе:
            соседний_узел = получить_соседний_узел(направление);

        если тип(соседний_узел) == река:
            если агента_смыло(соседний_узел, узел, направление):
                новый_узел = рассчитать_положение_агента();
                добавить_ребро(новый_узел, вес=1);
            иначе:
                добавить_ребро(соседний_узел, вес=1);
```

Ребра для хода с взрывом стены рассчитываются аналогично (вес данных ребер: 2).

Приложение Б

Модель ActionMask со вложенными слоями

Листинг Б.1 – вложенные слои модели ActionMask

```
TorchActionMaskModel(
    (internal_model): A2CNetwork(
        (internal_model): ComplexInputNetwork(
            (cnns): ModuleDict(
                (field): MazeCNN(
                    (_convs): Sequential(
                        (0): Conv2d(13, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
                        (1): Tanh()
                        (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
                        (3): Tanh()
                        (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0)
                        (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
                        (6): Tanh()
                        (7): Flatten(start_dim=1, end_dim=-1)
                    )
                )
                (other_stats): MazeCNN(
                    (_convs): Sequential(
                        (0): Conv2d(6, 24, kernel_size=(1, 2), stride=(1, 1), groups=6)
                        (1): Tanh()
                        (2): Flatten(start_dim=1, end_dim=-1)
                    )
                )
                (stats): MazeCNN(
                    (_convs): Sequential(
                        (0): Conv1d(6, 24, kernel_size=(2,), stride=(1,), groups=6)
                        (1): Tanh()
                        (2): Flatten(start_dim=1, end_dim=-1)
                    )
                )
                (treasures): MazeCNN(
                    (_convs): Sequential(
                        (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                        (1): Tanh()
                        (2): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
                        (3): Tanh()
                        (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0)
                        (5): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
                        (6): Tanh()
                        (7): Flatten(start_dim=1, end_dim=-1)
                    )
                )
                (walls): MazeCNN(
                    (_convs): Sequential(
                        (0): Conv3d(12, 32, kernel_size=(1, 3, 3), stride=(1, 1, 1),
padding=(0, 1, 1))
                        (1): Tanh()
                        (2): MaxPool3d(kernel_size=(1, 2, 2), stride=(1, 2, 2), padding=0)
                        (3): Conv3d(32, 64, kernel_size=(1, 3, 3), stride=(1, 1, 1),
padding=(0, 1, 1))
                        (4): Tanh()
                        (5): Conv3d(64, 128, kernel_size=(2, 3, 3), stride=(1, 1, 1))
                        (6): Tanh()
                        (7): Flatten(start_dim=1, end_dim=-1)
                    )
                )
            )
        )
    )
)
```


ОТЗЫВ
РУКОВОДИТЕЛЯ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Тема ВКР: Модуль принятия решений интеллектуальным агентом в многопользовательском игровом веб-приложении

Автор Козеровский Дмитрий Никитич

Институт информационных технологий

Кафедра корпоративных информационных систем

Направление 09.03.04 «Программная инженерия»

Профиль «Информационные системы управления ресурсами предприятия»

Руководитель Советов Пётр Николаевич,

доцент кафедры корпоративных информационных систем РГУ МИРЭА, к.т.н.

(Ф.И.О., место работы, должность, ученое звание, степень)

Оценка соответствия требованиям ФГОС подготовленности автора выпускной работы:

| Требования к профессиональной подготовке | Соответствует | В основном соответствует | Не соответствует |
|--|---------------|--------------------------|------------------|
| уметь корректно формулировать и ставить задачи (проблемы) своей деятельности при выполнении выпускной квалификационной работы, анализировать, диагностировать причины появления проблем, их актуальность | + | | |
| устанавливать приоритеты и методы решения поставленных задач (проблем) | + | | |
| владеть компьютерными методами сбора, хранения и обработки (редактирования) информации | + | | |
| уметь рационально планировать время выполнения работы, определять последовательность и объем операций и решений при выполнении поставленной задачи | + | | |
| уметь объективно оценивать полученные результаты расчетов, вычислений, используя для сравнения данные других направлений соответствующей отрасли науки | + | | |
| уметь анализировать полученные результаты и их интерпретации | + | | |
| уметь осуществлять деятельность в кооперации с коллегами, находить компромиссы | + | | |
| уметь делать самостоятельные обоснованные и достоверные выводы из проделанной работы | + | | |
| уметь пользоваться научной литературой профессиональной направленности | + | | |

Отмеченные достоинства работу отличает достаточная теоретическая и практическая проработка, в частности, заслуживают внимания описание и реализация современного алгоритма обучения с подкреплением РРО, а также результаты экспериментального анализа этого алгоритма, на хорошем уровне представлено детальное описание программной реализации многопользовательского игрового приложения

Отмеченные недостатки результаты работы не освещались на конференциях и в статьях, кроме того, желательно было бы иметь более детальное обоснование проигрыша алгоритма РРО простому дереву решений в рассматриваемой задаче.

Заключение считаю, что выпускная квалификационная работа Козеровского Дмитрия Никитича заслуживает высокой оценки, а ее автор достоин присвоения квалификации бакалавр по направлению 09.03.04 «Программная инженерия» профиль «Информационные системы управления ресурсами предприятия»

Руководитель проекта

подпись

фамилия и инициалы

«29» мая 2023 г.



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«МИРЭА – Российский технологический университет»
РТУ МИРЭА**

Институт информационных технологий
Кафедра корпоративных информационных систем

ЗАКЛЮЧЕНИЕ

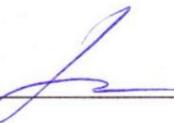
ВКР (бакалаврская работа) обучающегося Козеровского Дмитрия Никитича группы ИКБО-08-19 на тему «Модуль принятия решений интеллектуальным агентом в многопользовательском игровом веб-приложении» в соответствии с Порядком проведения проверки на объем заимствования и размещения в сети Интернет выпускных квалификационных работ и научных докладов об основных результатах подготовленных диссертаций СМКО МИРЭА 7.5.1/03.П.57–18 прошла автоматизированный анализ в системе РУКОНТЕКСТ Доля авторского текста (оригинальности) в результате автоматизированной проверки составила 82%.

Анализ результата автоматизированной проверки системой РУКОНТЕКСТ и мнение руководителя ВКР о достоверности, фактической доле оригинального текста и степени самостоятельности студента при написании работы позволяют допустить обучающегося Козеровского Дмитрия Никитича к защите выпускной квалификационной работы.

Руководитель ВКР


/Советов П.Н. /
«05» июня 2023 г.

Заведующий кафедрой


/Андранинова Е.Г. /
«05» июня 2023 г.

Справка о проверке на наличие заимствований

Имя файла: 090304_19И1682_Козеровский ДН.docx

Автор: Козеровский Дмитрий Никитич

Заглавие: Модуль принятия решений интеллектуальным агентом в многопользовательском игровом веб-приложении

Год публикации: 2023

Комментарий: Не указан

Коллекции: Интернет 2.0, Научные статьи 2.0, Русскоязычная Википедия, Англоязычная Википедия, Коллекция Энциклопедий, Библиотека Либрарусек, Университетская библиотека, Коллекция КФУ, ВКР Российского университета кооперации, Коллекция АПУ ФСИН, Коллекция ПГУТИ, Репозиторий открытого доступа СПб гос. ун-та, ВКР МИРЭА, Научная электронная библиотека "КиберЛенинка", ЦНМБ Сеченова, Авторефераты ВАК, Диссертации ВАК, Диссертации РГБ, Авторефераты РГБ, Готовые рефераты, ФИПС. Изобретения, ФИПС. Полезные модели, ФИПС. Промышленные образцы, Коллекция Руконт, Библиотека им. Ушинского, Готовые рефераты (часть 2), Открытые научные источники, НЭБ, БиблиоРоссика, Правовые документы I, Правовые документы II, Правовые документы III, Собрание законодательства Российской Федерации



Результат проверки

Оценка оригинальности документа: 82%

Оригинальные фрагменты: 81,90%

Обнаруженные заимствования: 15,34%

Цитирование: 2,76%



Работу проверил: Карусевич Т.Е.

Дата: 31.05.2023 2

Подпись: