**«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)»**
**(СПбГЭТУ «ЛЭТИ»)**

| | |
|---|---|
| **Направление** | 09.03.04 Программная инженерия |
| **Профиль** | Разработка программно-информационных систем |
| **Факультет** | КТИ |
| **Кафедра** | МО ЭВМ |

*К защите допустить*

И.о. зав. кафедрой А.А. Лисс

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

## ТЕМА: РАЗРАБОТКА ИГРЫ В ЖАНРЕ «ROUGELIKE» С ПРИМЕНЕНИЕМ МАШИННОГО ОБУЧЕНИЯ

| | | | |
|---|---|---|---|
| Студент | | _____ *подпись* | А.В. Шуняев |
| Руководитель | к.т.н., доцент | _____ *подпись* | В.В. Романцев |
| Консультанты | | _____ *подпись* | Т.В. Герасимова |
| | к.т.н., доцент | _____ *подпись* | А.Н. Иванов |
| | к.т.н., доцент | _____ *подпись* | М.М. Заславский |

Санкт-Петербург

2023

**ЗАДАНИЕ**
**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ**

Утверждаю
И.о. зав. кафедрой МО ЭВМ
_____ А.А. Лисс
«___»_____2023 г.

Студент Шуняев А.В. Группа 9304
Тема работы: Разработка игры в жанре «Rougelike» с применением машинного обучения
Место выполнения ВКР: Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)
Исходные данные (технические требования):
ОС Windows, технологии программирования выбираются исполнителем
Содержание ВКР:
Введение, Обзор предметной области, Формирование требований к разрабатываемой игре, Разработка игры, Безопасность жизнедеятельности, Заключение
Перечень отчетных материалов: пояснительная записка, иллюстративный материал.
Дополнительные разделы: Безопасность жизнедеятельности.

Дата выдачи задания «_04_»__апреля_____2023 г.
Дата представления ВКР к защите «_07_»__июня___2023 г.

Студент _____ А.В. Шуняев
Руководитель к.т.н., доцент _____ В.В. Романцев
Консультант _____ Т.В. Герасимова

**КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ**
**ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ**

Утверждаю
И.о. зав. кафедрой МО ЭВМ
_____ А.А. Лисс
«___»_____2023 г.

Студент Шуняев А.В. Группа 9304
Тема работы: Разработка игры в жанре «Rougelike» с применением машинного обучения

| № п/п | Наименование работ | Срок выполнения |
|---|---|---|
| 1 | Обзор литературы по теме работы | 04.04 – 19.04 |
| 2 | Постановка задачи | 20.04 – 21.04 |
| 3 | Разработка игры | 22.04 – 05.05 |
| 4 | Обучение модели противников | 06.05 – 10.05 |
| 5 | Безопасность жизнедеятельности | 11.05 – 12.05 |
| 6 | Оформление пояснительной записки | 13.05 – 15.05 |
| 7 | Предварительная защита работы | 16.05 |

Студент _____ А.В. Шуняев
Руководитель к.т.н., доцент _____ В.В. Романцев
Консультант _____ Т.В. Герасимова

# РЕФЕРАТ

Пояснительная записка 54 стр., 8 рис., 3 табл., 12 ист.

Roguelike, машинное обучение, компьютерные игры, Unity[1], Unity ML-Agents[2], обучение с подкреплением, модели противников.

**Объектом исследования** является разработка компьютерной игры в жанре «Roguelike» с использованием модели поведения противников на базе машинного обучения.

**Предметом исследования** является преимущества и недостатки использования моделей противников, обученных с применением машинного обучения, в «Roguelike» играх.

**Цель работы:** разработка и анализ модели поведения противника, в частности перемещение противника к игроку, на базе машинного обучения для компьютерной игры в жанре «Roguelike».

В данной работе проведена разработка компьютерной игры в жанре «Roguelike», в которой для реализации модели противника было использовано машинное обучение.

При выполнение работы была проанализирована предметная область, изучены методы машинного обучение, которые применяются в разработке компьютерных игр в жанре «Roguelike», рассмотрены технологии, с помощью которых можно реализовать игру в данном жанре с применением машинного обучения, а также рассмотрены существующие игры данного жанра и наработки для игр, которые реализуют противников с применением машинного обучения и без него. Также, на основе полученной информации, была разработана компьютерная игра, соответствующая требованиям жанра «Roguelike», в которой противники являются интеллектуальными агентами реализованными с помощью машинного обучения. По результатам разработки были выявлены преимущества и недостатки применения машинного обучения для разработки моделей противников для игр в жанре «Roguelike».

Для разработки игры использовался фреймворк для разработки игр Unity[1] в связке с пакетом библиотек для применения машинного обучения Unity ML-Agents[2].

Для обучения модели противников использовался метод обучения с подкреплением.

## ABSTRACT

In this work, a computer game in the Roguelike genre was developed, in which machine learning was used to implement the enemy model.

During the project, the subject area was analyzed, machine learning methods used in the development of Roguelike games were studied, technologies for implementing games in this genre with machine learning were examined, and existing games in this genre and approaches for implementing enemies with and without machine learning were explored. Based on the gathered information, a computer game was developed that meets the requirements of the Roguelike genre, where enemies are intelligent agents implemented using machine learning. The advantages and disadvantages of using machine learning for developing enemy models for Roguelike games were identified as a result of the development process.

The Unity[1] game development framework, along with the Unity ML-Agents[2] package, was used to develop the game.

Reinforcement learning was used as the training method for the enemy models

# СОДЕРЖАНИЕ

## ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ЯП – язык программирования;

IDE – Integrated Development Environment;

Фреймворк – это программная платформа, определяющая структуру программной системы;

Unity[1] – фреймворк для разработки компьютерных игр на ЯП C#;

PixelArt – это стиль искусства и графического дизайна, в котором изображения создаются путем использования маленьких квадратных пикселей. Каждый пиксель представляет собой отдельный элемент изображения и имеет определенный цвет или оттенок;

API (Application Programming Interface) – это набор определенных методов, функций, протоколов и инструментов, предоставляемых программным обеспечением, для обмена информацией и взаимодействия с другими программами или компонентами;

Тайл (tiles) – представляет собой небольшую часть графического элемента, такую как фон, текстура или объект, которая повторяется для создания большой области или сцены;

Тайлсет (Tileset) – это графический набор, состоящий из небольших изображений, называемых тайлами (tiles);

Контент (Content) – это все элементы и материалы, которые присутствуют в игре и создают игровой опыт для игрока. Он включает в себя графику, аудиоэффекты, анимации, музыку, текст, диалоги, игровые объекты, персонажей, уровни, сюжет и многое другое;

Датасет (Dataset) – это набор данных, который используется для обучения модели машинного обучения. Он состоит из входных признаков и соответствующих целевых переменных или выходных значений;

Ассет (Asset) – в контексте разработки игр обозначает любой ресурс, который используется при разработке игры, такой как графика,

звуковые эффекты, модели персонажей, текстуры, анимации, сцены, скрипты, настройки и многое другое;

Плагин (Plugin) - это программное расширение или модуль, который добавляет дополнительную функциональность или возможности в основное программное обеспечение. Он позволяет расширять и модифицировать функциональность программы без изменения ее исходного кода;

CPU (Central Processing Unit) - это основной процессор или центральный процессор компьютера. CPU является ключевым компонентом компьютерной системы и отвечает за выполнение основных операций и обработку данных в компьютере;

GPU (Graphics Processing Unit) - это графический процессор или ускоритель графики. Он является специализированным процессором, предназначенным для обработки и отображения графики, включая рендеринг 2D и 3D графики, выполнение сложных математических операций и управление выводом графических данных на монитор.

# ВВЕДЕНИЕ

Индустрия видеоигр, так называемый «GameDev», это одно из самых крупных, быстрорастущих и быстроизменяющихся сегментов индустрии развлечений. На момент начала 2021 года масштабы игровой индустрии были сопоставимы с киноиндустрией, а темпы роста за последние годы значительно ее опережают. Это делает данное направление одним из самых перспективных для разработчиков, а также привлекательным для инвесторов и издателей.

Жанр «Roguelike»[3] берет свои корни из классической компьютерной игры Rogue, разработанной Майклом Тоем, Гленном Вичманом и Кеном Арнольдом в 1980 году. Игра Rogue была пошаговой ролевой игрой с процедурно генерируемыми уровнями, где игрок управлял персонажем, исследовал подземелья, сражался с монстрами и собирал сокровища. Особенностью Rogue была его высокая сложность и перманентность, то есть при смерти персонажа нужно было начинать игру заново.

После выпуска Rogue было создано множество других игр, которые позаимствовали его особенности и развили жанр Roguelike. Эти игры также имели процедурно генерируемые уровни, пошаговый геймплей, случайно генерируемые предметы и высокую степень сложности.

В последующие годы жанр Roguelike продолжил развиваться и привлекать внимание игроков и разработчиков. Игры, такие как Nethack, Angband, Dungeon Crawl Stone Soup и Tales of Maj'Eyal, стали известными представителями жанра.

Со временем жанр Roguelike расширился и включил в себя различные вариации и поджанры. Некоторые игры добавили в жанр элементы экшена, платформера или головоломки. Возникли также игры с сетевым режимом, в которых несколько игроков могут соревноваться или сотрудничать в процессе исследования подземелий.

Жанр Roguelike стал популярным благодаря своей сложности, перманентности и процедурной генерации, что делает каждую игру

уникальной и вызывает у игроков желание повторить попытку и достичь больших успехов. В настоящее время Roguelike игры имеют широкую аудиторию и активно разрабатываются как независимыми разработчиками, так и крупными студиями.

Данный жанр можно описать как путешествия по случайно сгенерированным подземельям с небольшим сюжетом или без него совсем. Игроку бросает вызов как и само подземелье, так и монстры находящиеся в нем. Игрок раз за разом начинает сначала, каждый раз набираясь опыта, что позволяет ему пройти дальше. Еще одним удачным описанием жанра можно считать дословный перевод его названия - «игра наподобие Rogue». Так как оно очень емко описывает чего ждать от игры, при условие, что интересующийся знаком с игрой «Rogue».

Рандомизированная генерация подземелий сильно влияет на уникальность игрового процесса для каждого отдельного игрока. Однако, такая генерация подземелий может стать причиной плохой сбалансированности игрового процесса: игровая ситуация может быть либо слишком легкой, либо слишком сложной для игрока. Такие ситуации пагубно влияют на интерес игроков к игре, а как следствие, плохо сказываются на популярности игры и ее рентабельности. Машинное обучение имеет большой потенциал для решения данной проблемы. С его помощью можно обучить противников адаптироваться под изменения локаций, что поможет сохранить баланс игры. В таком случае случайная генерация подземелий сможет сильнее разнообразить содержание игры с минимальными затратами на разработку и увеличит количество уникальных ситуаций при каждой новой попытке прохождения игры.

**Цель:** Разработка игры в жанре «Roguelike», с применением машинного обучения противников, в частности перемещение противника к игроку, на базе машинного обучения для компьютерной игры в жанре «Roguelike».

**Задачи:** Был выделен ряд задач, решение которых должно привести к достижению цели:

1. Изучить методы машинного обучения, которые могут быть применены при разработке компьютерных игр, в частности для реализации перемещения моделей противников.
2. Анализ релевантных решений разработки противников с применением машинного обучения и без него;
3. Формирование требований к разрабатываемой компьютерной игре в жанре «Roguelike»;
4. Разработка игры и модели противника на базе машинного обучения;
5. Анализ преимуществ и недостатков моделей противников.

**Объект исследования:** В качестве объекта исследования работы была выбрана разработка компьютерной игры в жанре «Roguelike» с использованием модели поведения противников на базе машинного обучения.

**Предмет исследования:** Предметом исследования работы стал преимущества и недостатки использования моделей противников, обученных с применением машинного обучения, в «Roguelike» играх.

# 1 Анализ предметной области

## 1.1 Применение машинного обучения в разработке компьютерных игр в жанре «Roguelike»

### 1.1.1 Обзор методов машинного обучения применяемых в разработке компьютерных игр

Машинное обучение является мощным инструментом для моделирования поведения противников в играх, в том числе в жанре «Roguelike». Вот несколько методов машинного обучения, которые часто используются в разработке игр для создания интеллекта противников:

- Классический метод "Finite State Machine" (FSM)[4]: В этом методе противники представлены конечным автоматом с различными состояниями и переходами между ними. Каждое состояние определяет конкретное поведение противника, например, атаку, перемещение или ожидание. Разработчики могут вручную настраивать условия переходов между состояниями на основе определенных правил или событий в игре.
- Генетические алгоритмы (Genetic Algorithms)[4]: Этот метод использует эволюционные алгоритмы для создания и обучения поведения противников. В начале игры создается популяция случайных противников с разными стратегиями. Затем, с помощью механизма отбора и мутаций, выбираются и комбинируются наилучшие стратегии. Таким образом, противники постепенно обучаются и становятся более адаптированными к игровой среде.
- Обучение с подкреплением (Reinforcement Learning)[4]: Этот метод позволяет противникам учиться на основе награды и наказания, полученных в ходе игры. Противник, представленный в виде агента, принимает решения в определенных состояниях игры и получает положительную или отрицательную награду в зависимости от результата своих действий. Алгоритмы обучения с подкреплением, такие как Q-обучение или глубокие нейронные сети, могут

использоваться для обучения противников, улучшая их стратегии со временем.

- Обучение с учителем (Supervised Learning)[4]: В этом методе разработчики предоставляют модели данные, содержащие информацию о желаемом поведении противника. Например, разработчики могут записать игры опытных игроков и использовать эти данные для обучения модели предсказывать оптимальные действия противников в различных ситуациях.
- Метод имитационного обучения (Imitation Learning)[4]: Данный метод использует экспертные примеры поведения для обучения модели. Вместо итеративного исследования в реальной среде, модель изучает оптимальное поведение, наблюдая и имитируя действия экспертов. Этот подход позволяет быстро и эффективно обучать модель, избегая сложностей, связанных с исследованием в RL.

Это лишь несколько примеров методов машинного обучения, которые могут быть использованы при разработке компьютерной игры, в том числе и в жанре «Roguelike».

## 1.2 Фреймворк для разработки компьютерных игр Unity[1] и пакет для обучения моделей противников Unity[1] ML Agents

### 1.2.1 Фреймворк для разработки компьютерных игр Unity[1]

Unity[1] – кроссплатформенная среда разработки, которая предоставляет основные необходимые средства для разработки двух- и трехмерных компьютерных игр. Данный фреймворк был разработан на языке C++ и выпущен в 2005 году американской компанией Unity[1] Technologies. Фреймворки подобные Unity[1] называют игровыми движками.

Unity[1] – один из самых популярных фреймворков для разработки компьютерных игр. Он имеет большую, хорошо описанную документацию и, что не мало важно, большое сообщество разработчиков, которые делятся

своими наработками, создают уроки по разработке компьютерных игр, пишут плагены и многое другое. Также есть официальная площадка, на которой пользователи могут выставлять ассеты на продажу или для свободного скачивания.

Данный игровой движок использует объектно-ориентированную парадигму программирования (ООП) и предоставляет возможность описывать механики с помощью скриптов на языке C#. Это очень упрощает разработку, так как в ЯП C# реализован сборщик мусора (Garbage Collector), который берет на себя ответственность за управление памятью, что уменьшает количество возможных ошибок и ускоряет разработку.

Основном элементом при разработке игры на Unity[1] является GameObject. Это базовый элемент, который имеет свой жизненный цикл и который можно модернизировать с помощью различных компонентов, в том числе и скрипты описанные на ЯП C#.

Для написания скриптов можно использовать различные IDE, которые подключатся к Unity[1]. Одной из таких IDE является Rider[5].

Rider[5] - это интегрированная среда разработки (IDE) от компании JetBrains, предназначенная для разработки приложений на платформе .NET, включая Unity[1]. Она предоставляет широкий набор инструментов и функций, упрощающих процесс разработки, отладки и управления проектами.

Вот некоторые основные особенности и преимущества Rider[5]:

- Интеграция с Unity[1]: Rider[5] обеспечивает полную интеграцию с Unity[1], позволяя разработчикам создавать, отлаживать и управлять проектами Unity[1] в удобной IDE. Он поддерживает функции, специфичные для Unity[1], такие как автозавершение кода, рефакторинги, быстрое переключение сцен, просмотр компонентов и другие.
- Богатый набор инструментов: Rider[5] предлагает широкий спектр инструментов для повышения производительности

разработчиков. Это включает в себя интеллектуальное автодополнение кода, рефакторинги, систему контроля версий, отладчик, профилировщик производительности, визуализацию структуры проекта и другие функции, упрощающие разработку и отладку.

- Поддержка различных языков и фреймворков: Rider[5] поддерживает не только C# и Unity[1], но также другие популярные языки и фреймворки, такие как Java, Kotlin, JavaScript, TypeScript, Python и другие. Это позволяет разработчикам работать над различными проектами с использованием разных технологий в одной среде.
- Кросс-платформенность: Rider[5] доступен для различных операционных систем, включая Windows, macOS и Linux. Это позволяет разработчикам работать на предпочитаемой платформе, не теряя возможностей и удобства IDE.
- Расширяемость: Rider[5] предлагает возможность расширения функциональности через плагины. С помощью плагинов можно интегрировать сторонние инструменты, добавлять новые функции и настраивать среду разработки под свои потребности.

Rider[5] предоставляет разработчикам Unity[1] мощную и удобную среду для работы над проектами, обеспечивая широкий набор функций и инструментов для повышения производительности и удобства разработки.

### 1.2.2 Пакет для обучения моделей противников Unity[1] ML Agents

Unity ML-Agents[2] – это пакет библиотек обучения и использования интеллектуальных агентов с помощью методов машинного обучения на базе движка Unity[1]. Данный пакет взаимодействует с фреймворком PyTorch. Даннай фреймворк позволяет создавать и обучать различные модели при помощи машинного обучения. Также он имеет открытый исходный код

В данный пакет входят три основные компонента:

- ML-Agents – это библиотека, которая предоставляет возможности для описания сценариев обучения агентов непосредственно в Unity[1] при помощи скриптов. Данная библиотека написана на ЯП C#. Она взаимодействует с API фреймворка PyTorch, написанном на Python, для обучения моделей интеллектуальных агентов.
- mlagents-learn – та самая библиотека, которая предоставляет API для фреймворка PyTorch.
- Unity Barracuda – это библиотека, позволяющая использовать уже обученные модели. Данная библиотека позволяет работать с моделями, обученными с помощью популярный фреймворков для машинного обучения: TensorFlow и PyTorch. Barracuda позволяет проводить вычисления как на CPU, так и на GPU, при этом она использует compute shaders. Данные шейдеры кроссплатформенные и поддерживают различные интерфейсы для описания шейдеров (Metal graphics, Vulkan, OpenGL). Для использования уже обученных моделей применяется Unity[1] Inference Engine, который основан на Barracuda и является часть стандартных инструментов Unity[1]. Схема компонентов пакета Unity ML-Agents[2] представлена на рис. 1.


Рисунок 1 - Схема компонентов пакета Unity ML-Agents[2]

### 1.2.3 Методов машинного обучения применяющиеся в Unity ML-Agents[2]

При обучение интеллектуальных агентов с использованием Unity ML-Agents[2] в основном применяются методы обучения с подкреплением и метод имитационного обучения. Рассмотрим их по отдельности.

**Метод обучения с подкреплением (Reinforcement Learning, RL)**

Метод обучения с подкреплением является подразделом машинного обучения, который фокусируется на обучении агента принимать последовательность действий в определенной среде с целью максимизации некоторой числовой награды.

Вот некоторые основные компоненты RL и их использование в Unity[1]:

- Агенты (Agents): Агенты в RL представляют управляемые единицы, которые взаимодействуют с игровой средой. В Unity[1] агенты могут быть реализованы в виде объектов, которые имеют способность принимать решения и выполнять действия в зависимости от состояния окружающей среды.
- Среда (Environment): Среда представляет игровое пространство, с которым агент взаимодействует. В Unity[1] среда может быть создана с использованием сцены или уровня игры. Среда определяет правила, состояния и возможные действия, которые агент может выполнять.
- Награды (Rewards): В RL награды представляют числовую оценку, которую агент получает от среды в результате выполнения действий. В Unity[1] награды могут быть определены на основе заданных правил и условий игры, и они могут быть представлены в виде переменных или компонентов объектов.
- Обучение и исследование (Learning and Exploration): В RL агенты обучаются путем взаимодействия с средой, экспериментируя и исследуя различные действия и стратегии. Unity[1] предоставляет возможности для реализации алгоритмов RL, таких как Q-обучение (Q-Learning), глубокие нейронные сети (Deep Neural Networks) или

генетические алгоритмы (Genetic Algorithms), с использованием библиотеки ML-Agents. Схема обучения с подкреплением представлена на рис. 2.

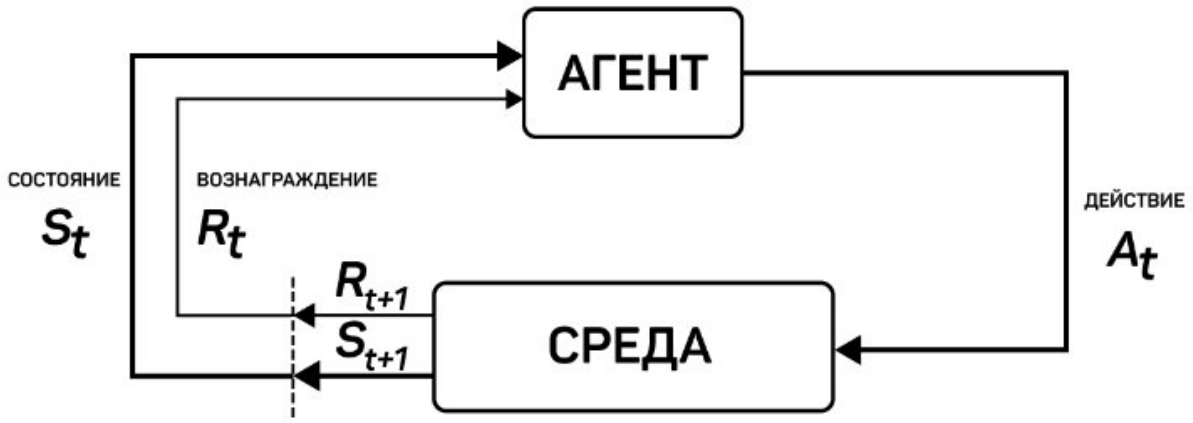


Рисунок 2 - Схема обучения с подкреплением

**Метод имитационного обучения (Imitation Learning)**

Метод имитационного обучения — это подход в машинном обучении, при котором модель обучается на основе примеров экспертного поведения. Вместо того чтобы использовать традиционные RL-алгоритмы, которые требуют долгого итеративного обучения в реальной среде, имитационное обучение позволяет модели изучать желаемое поведение, наблюдая действия эксперта.

Вот основные компоненты и шаги, связанные с применением этого метода в Unity[1]:

- Сбор экспертных данных: Первый шаг в имитационном обучении - это сбор данных, содержащих примеры экспертного поведения. Это могут быть записи игры опытных игроков или экспертных демонстраций. Unity[1] позволяет записывать игровые сеансы и получать доступ к информации о действиях и состояниях в игре.
- Создание датасета: Собранные данные экспертного поведения используются для создания датасета, который состоит из пар

состояние-действие. Каждое состояние представляет собой информацию о текущем состоянии игры, а действие - действие, выполненное экспертом в данном состоянии.

- Обучение модели на датасете: После создания датасета можно обучить модель на этих данных, используя алгоритмы машинного обучения, такие как нейронные сети. В Unity ML-Agents[2] можно использовать алгоритмы, такие как Behavioral Cloning или Generative Adversarial Imitation Learning (GAIL), для обучения модели на основе датасета.
- Оценка и тестирование модели: После обучения модель можно оценить, выполнив ее в среде и наблюдая ее поведение. Затем модель можно протестировать в реальных игровых условиях и настраивать ее параметры для достижения оптимального поведения.

Изучив данные методы было принято решение использовать метод обучения с подкреплением, так как для него не требуется заготовленный датасет, так как с учетом особенностей жанра «Roguelike», а именно случайной генерации подземелий, собранный датасет может быть недостаточным для обучения, что приведет к некорректной работе модели.

## 1.3 Анализ существующих решений для создания моделей поведения противников

### 1.3.1 Отбор аналогов

В ходе изучения предметной области рассматривались компьютерные игры в жанре «Roguelike» или наработки для них, которые реализовывали модели противников с применением машинного обучения и без него. В результате были выбраны следующие аналоги:

**Cogmind[6]**- Roguelike игра, в которой машинное обучение применялось для создания противников с адаптивным поведением. Противники в игре обучались анализировать игровое поле и принимать

решения на основе текущей ситуации, что делает их более умными и реалистичными.

**Roguelike от damiandiaz212[7]** – ассет, с открым исходным кодом, реализующий базовые механики компьютерных игр в жанре «Roguelike». В данной реализации для моделей противников применяется алгоритм A* (AStar). Хотя противники и выполнены на совсем примитивном уровне, тем не менее они способны выполнять поставленные задачи.

**The Binding of Isaac[8] –** Roguelike игра, в которой противники в The Binding of Isaac[8] обладают простым искусственным интеллектом, который управляет их действиями и принятием решений. Это может включать перемещение в сторону игрока, использование атакующих навыков или попытку избежать опасностей.

**Caves of Qud** - Глубокая Roguelike игра с открытым миром, где машинное обучение использовалось для создания противников с разнообразным поведением. Противники в игре обучались различным стратегиям и умениям, что делает каждое столкновение уникальным и интересным.

В итоге были отобраны 4 компьютерных игры в жанре «Roguelike» или наработки для разработки игр. Каждая из них реализует модели поведения противников.

### 1.3.2 Критерии сравнения аналогов

1. Применение машинное обучение – было использовано машинное обучение при создание моделей поведения противников;
2. Факторы влияющие на разнообразие игрового процесса – каким способом достигается разнообразие игровых ситуаций
3. Открытость реализации моделей противников – являются ли общедоступными методы и технологии, с помощью которых были разработаны модели поведения противников.

Сравнение аналогов по данным критериям представлено в таблице 1

Таблица 1 – Сравнение аналогов по критериям

| | Cogmind | Roguelike от damiandiaz212 | The Binding of Isaac | Caves of Qud[9] |
|---|---|---|---|---|
| Применение машинное обучение | Да | Нет | Нет | Да |
| Факторы влияющие на разнообразие игрового процесса | Случайная генерация локаций, увеличение разнообразия поведения противников за счет машинного обучения, возможность изменять ландшафт локаций своими действиями | Случайная генерация локаций, возможность изменять ландшафт локаций своими действиями | Случайная генерация локаций, Большое количество разнообразных способностей, как игрока так и противников | Случайная генерация локаций, увеличение разнообразия типов противников с различным поведением за счет машинного обучения |
| Открытость реализации моделей противников | Нет | Да | Нет | Нет |

### 1.3.3 Итоги сравнения аналогов

По итогам сравнения можно сделать вывод, что реализации игровых моделей без применения машинного обучения не являются плохим вариантом решения, однако для того, чтобы игровые ситуация действительно были разнообразными, к ним необходимо добавлять большое количество дополнительного контента. Такой способ обширно применятся в игровой индустрии, однако он требует больших вложений, так как для увеличения количества контента, надо не только реализовывать новые механики, а также балансировать их, делать новые анимации и эффекты. Модели построенные с применением машинного обучения могут достичь близкого уровня уникальности игровых ситуации, при этом для их реализации нужно сильно меньше ресурсов. Также, если все-таки добавить к моделям построенным при помощи машинного обучения больше количество другого контента, это выведет компьютерные игры на новый уровень технологического развития.

# 2 Формирование требований к разрабатываемой компьютерной игре

## 2.1 Постановка задачи

Для достижения цели необходимо разработать компьютерную игру, которая соответствует жанру «Roguelike», а именно должна включать в себя основные механики такие как: случайная генерация уровней, пермаментная смерть игрового персонажа с необходимостью начинать сначала, а также управление ограниченными ресурсами.

Также в данной игре противники должны быть реализованы с применением машинного обучения.

## 2.2 Требование к решению

Компьютерная игра должна соответствовать следующим требованиям:

1. Компьютерная игра должна содержать случайную генерацию уровней;
2. При гибели игрового персонажа, прогресс должен сбрасываться;
3. Должна быть реализована система ресурсов, таких как например очки здоровья и патроны;
4. Противники должен быть разработаны с применением машинного обучения.
5. Для обучения моделей противников должен применять метод обучения с подкреплением.

## 2.3 Обоснование выбранных требований

Так как в работе проводится исследование применения машинного обучения в компьютерных играх в жанре «Roguelike», разрабатываемая игра должна соответствовать жанру, что может быть достигнуто только через реализацию необходимых механик присущих жанру.

Противники должны быть реализованы с применением машинного обучения, так как на основе полученных моделей можно будет сделать вывод

о целесообразности использования машинного обучения при разработки игры данного жанра.

Метод обучения с подкреплением был выбран на основании того, что данный вид обучения не требует изначальных исходных данных, кроме как окружения, в котором будет обучаться модель. В добавок к этому, случайная генерация присущая жанру, может предоставить дополнительные данные для лучшего обучения модели.

### 2.4 Выбор средств разработки

Был выбран игровой движок Unity[1], как основной фреймворк для разработки игр, так как он имеет большое сообщество разработчиков, который к текущему моменту уже произвели огромное количество материалов, которые будут полезны при разработке. Так же в ряду преимуществ данного фреймворка: хорошая документация, мощный редактор для разработки игрового окружения как двухмерных, так и трехмерных игр, а также удобные средства для обучения интеллектуальных агентов.

В качестве IDE для написания скриптов механик был выбран Rider[5], так как он предоставляет удобный для разработки программ на C#, имеет интеграцию с Unity[1], а также предоставляет бесплатную подписку для студентов.

В качестве основного средства для реализации противников был выбран пакет Unity ML-Agents[2], так как он имеет интеграцию с игровым движком Unity[1]. Данный пакет будет с может внести много пользы как в разработку сценариев для обучения противников, так и для использование уже обученных агентов в игровом процессе.

Использования уже обученных агентов будет происходить с использованием Unity[1] Barracuda – одной из библиотек пакета Unity ML-Agents[2], которая использует для расчетов вычислительные шейдеры. Для

расчетов действий агентов будет использоваться как вычислительных ресурсов GPU, так и CPU.

# 3 Разработка компьютерной игры в жанре «Roguelike»

## 3.1 Выбор стилистики игры

### 3.1.1 Подбор графических ассетов окружения

Для реализации компьютерной игры в жанре «Roguelike» было принято решение использовать двухмерную графику, в стиле PixelArt так как это более аутентично для жанра. Также очень легко найти достаточное количество бесплатных графических ассетов для создания окружение.

Был выбран следующий набор тайлов для создания окружения. Выбранный тайлсет представлен на рис. 3.

Рисунок 3 - Набор тайлов для игры

В данном тайлсете есть достаточное количество тайлов для создание окружения, однако еще не хватает спрайтов игрока и врагов.

### 3.1.2 Создания графических ассетов игрока и противников

Необходимые спрайты игрока и противников были нарисованы с помощью программы Aseprite[10]. Aseprite[10] является одним из самых популярных инструментов для создания PixelArt. Он предлагает множество функций, таких как удобные редакторы для рисования и редактирования спрайтов и палитр, анимирования изображений, а также экспорт в различные форматы. На рис. 4 представлены нарисованные спрайты игрока и врагов.

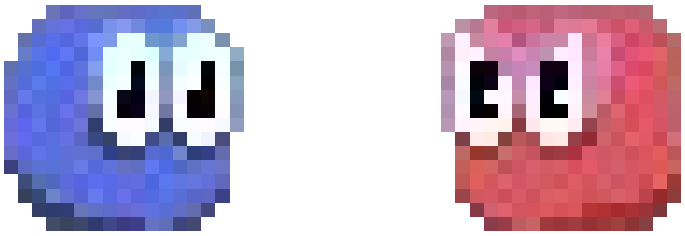

Рисунок 4 - Спрайты игрока и врагов

Данные спрайты выполнены в разрешение 16x16 пикселей.

## 3.2 Архитектура приложения

При построение архитектуры приложения было выделено три основных модуля:

- Игровой менеджер, который объединяет остальные модули
- Существа, которые представляют собой игрока и противников;
- Игровое окружение, то есть генерация карты.

Игровой менеджер, с помощью данных модулей, заполняет игровую сцену следующим способом. Сначала генерируется карта, состоящая из нескольких комнат. Далее выбирается комната игрока. В ее центр размещается игрок. После этого во все оставшиеся комнаты в случайном порядке размещаются противники. После размещения противников, размещаются ресурсы. Ресурсы не имеют разработаны на основе стандартных компонентов Unity, поэтому не вынесены в отдельный модуль. На рис. 5 представлена UML-диаграмма модулей.

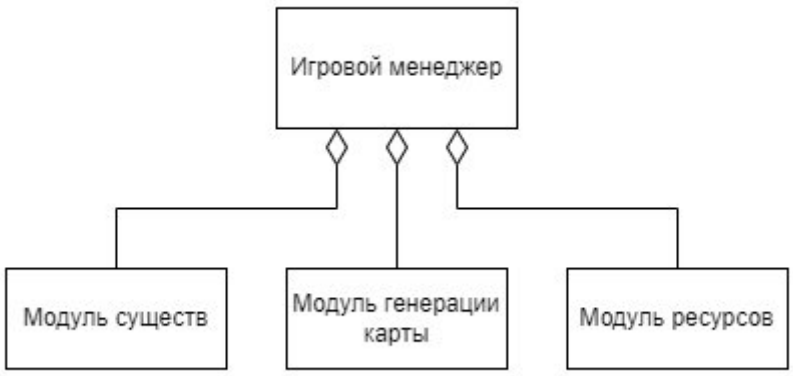


Рисунок 5 - UML-диаграмма модулей

### 3.2.1 Описания модуля существ

Для объектов, который представляют собой существ был определен набор характеристик и разработан

Игрок имеет следующие базовые характеристики:

- Здоровье – количество единиц здоровья, когда данный показатель падает до 0 - игрок умирает;
- Скорость передвижения – скорость, с которой двигается игрок;
- Оружие – объект, который задает урон и скорость полета снаряда;
- Патроны – количество зарядов, которое может выпустить игрок.

Противник имеет следующие базовые характеристики:

- Здоровье – количество единиц здоровья, когда данный показатель падает до 0 - противник умирает;
- Скорость передвижения – скорость, с которой двигается противник;
- Урон – количество единиц здоровья, отнимаемых у игрока, когда противник сталкивается с ним.

Был выделен ряд факторов, которые объединяют объекты игрока и противника: очки здоровья, скорость, урон, способность двигаться и атаковать. Исходя из этого было принято решение реализовать архитектуру так, чтобы общие функции переиспользовались или переопределялись, но сохраняли общий интерфейс.

Чтобы достичь такой гибкости системы, был создан базовый абстрактный класс персонажа, который определяет основные общие характеристики, а также необходимые методы такие как: движение, атака, поворот. Также данный класс содержит функционал для получения данных от внешнего управления.

От данного класса наследуются два класса: игрока и абстрактный класс противника. Данные классы переиспользуют, реализуют или переопределяют необходимые механики под себя. При этом от класс противника также наследуются подклассы противников, которые могут изменить необходимые методы. Такая система позволяет достичь возможности расширять количество типов противников, при этом дает возможность не вносить изменения в уже существующие типы и сократить дублирование кода до минимума.

Так как объекты игрока и противников имеют общий интерфейс для управления. Был реализованы два контроллера: контроллер игрока и контроллер противников. Каждый из них передает необходимую

информацию объектам такую как направления движения и нацеливания, а также вызывает метод атаки. Данные контроллеры различаются тем откуда они берут необходимую информацию. Контроллер игрока получает информацию конкретно от человека играющего в игру через систему ввода. Контроллер противников же получает необходимую информацию от агента обученного с помощью машинного обучения. На рис. 6 представлена UML-диаграмма классов описывающая объекты игрока и противников.

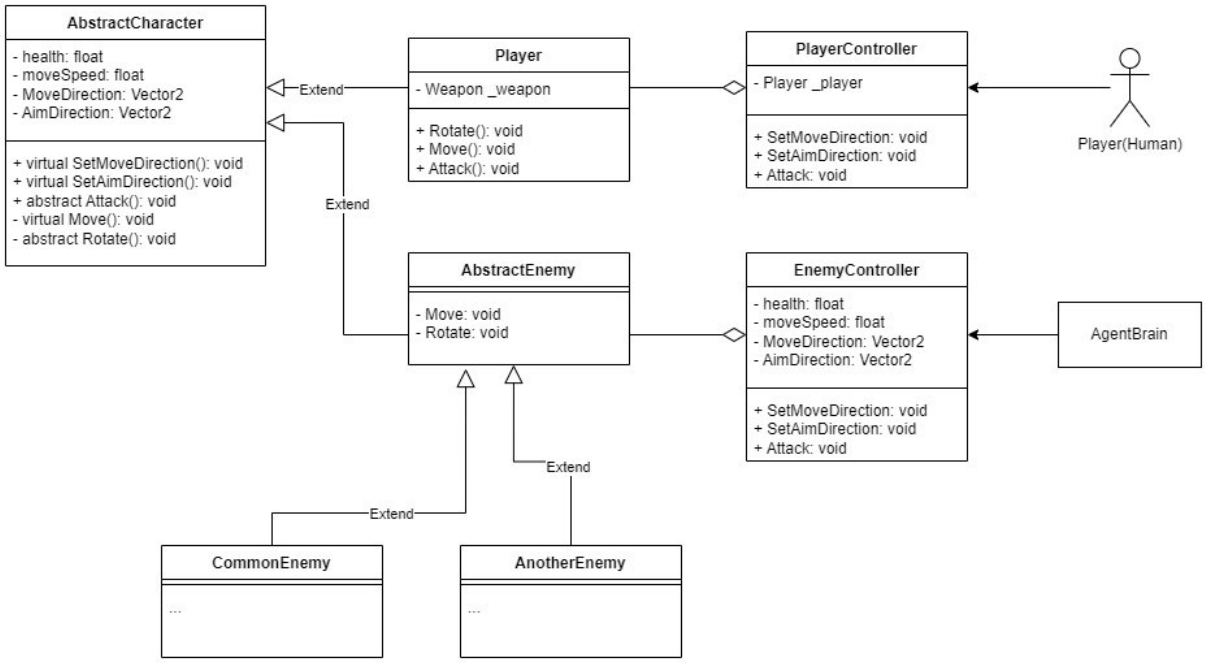


Рисунок 6 - UML-диаграмма классов объектов игрока и противников

Игроку доступен следующий набор действий:

- Перемещение – объект игрок двигается в различные направления в соответствии с нажатыми клавишами W,A,S,D;
- Слежение за курсором мыши – объект игрока поворачивается в направлении курсора.
- Выстрел – при нажатии левой кнопки мыши игрок выпускает снаряд, который летит по направлению курсора.

Для реализации этих действий была использована новая система управления (New Input System) Unity[1].

Данная система ввода, предназначенная для обработки пользовательского ввода в играх. Она предоставляет удобные и гибкие средства для работы с различными устройствами ввода, такими как клавиатура, мышь, геймпады, сенсорные экраны и другие.

Она обладает следующим радом преимуществ:

- Унифицированный интерфейс: New Input System предоставляет унифицированный интерфейс для работы с разными устройствами ввода. Это означает, что разработчику не нужно заботиться о различиях в обработке ввода для каждого устройства отдельно;
- Гибкость и настраиваемость: Система обладает высокой гибкостью и позволяет настраивать обработку ввода в соответствии с потребностями игры. Вы можете создавать свои собственные пользовательские события ввода и настраивать параметры обработки, такие как чувствительность или пороговые значения;
- Многоуровневая обработка ввода: New Input System позволяет легко определять различные уровни обработки ввода, такие как глобальный, игровой объект или отдельные компоненты. Это позволяет упростить организацию обработки ввода в игре;
- Низкая задержка и высокая производительность: New Input System разработана с учетом минимизации задержки ввода и обеспечения высокой производительности даже при обработке большого количества ввода;
- Поддержка разных платформ: Система поддерживает разные платформы, включая ПК, консоли и мобильные устройства. Это позволяет разработчикам легко адаптировать свою игру под разные платформы и устройства.

На основе данной системы ввода был реализован контроллер, который сообщает объекту игрока необходимую информацию для совершения действий.

### 3.2.2 Генерация карты

Был написан статический класс, в котором представлены реализации алгоритмов, также был разработан класс, визуализирующий сгенерированную карту. Также был разработан генератор карты, который использует эти классы. Данный генератор наследуется от абстрактного класса. При такой структуре, при необходимости, можно реализовывать новые генераторы, при этом сохранив интерфейс взаимодействия. Размеры карты, а также размеры комнат передаются в скрипт как настройки компонентов. На рис. 7 представлена UML-диаграмма классов описывающая модуль генерации карт.

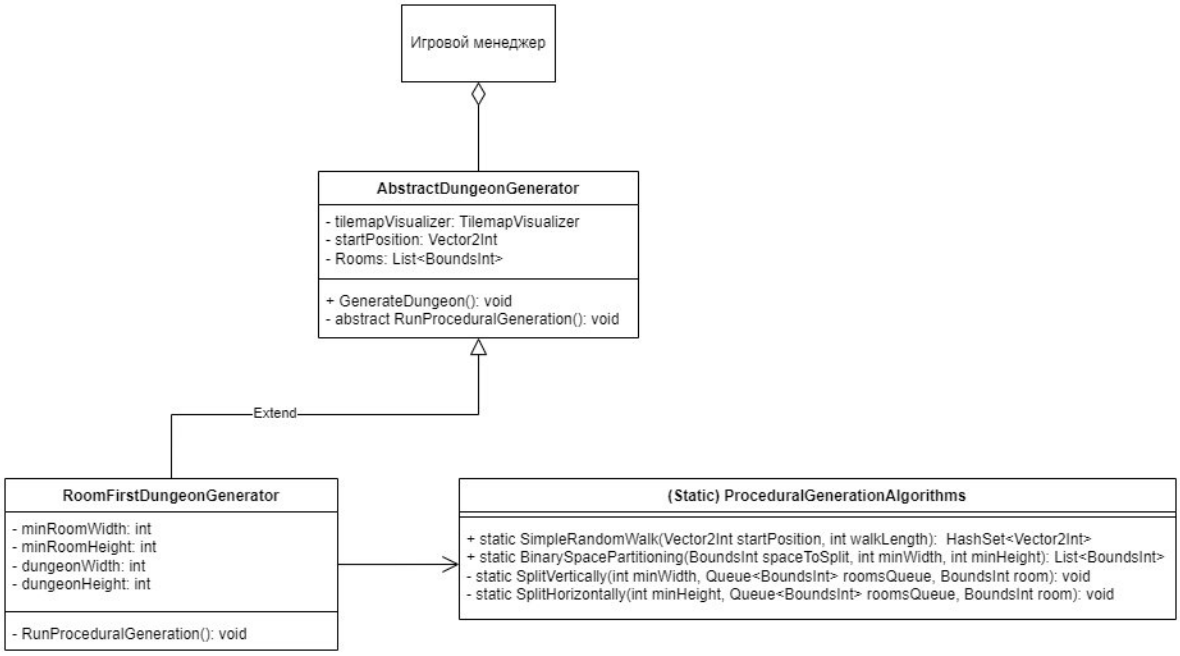


Рисунок 7 - UML-диаграмма модуля генерации карт

Генерация карты была реализована с помощью алгоритмов BSP (Binary Space Partitioning) Tree и RandomWalk[12]. Данный алгоритмы является популярным методами генерации карт для Roguelike игр.

BSP Tree[11] использует двоичное дерево для разделения пространства карты на прямоугольные области, что позволяет создавать комнаты и коридоры с определенной структурой и иерархией.

В общем виде данный алгоритм можно разбить на несколько шагов:

- Создание корневого узла – на данном этапе создается корневой узел дерева, который представляет собой всю карту;
- Рекурсивное разделение узлов – текущий узел делится на две подобласти рекурсивно до определенной грубины;
- Создание комнат - На каждом уровне глубины дерева создаются комнаты в каждом листовом узле. Комнаты могут быть разных размеров и форм, их расположение внутри подузла определяется случайным образом;
- Создание коридоров – создание коридоров между комнатами;
- Оптимизация и обработка – проверка сгенерированной карты на отсутствие изолированных и других проблем с проходимостью;
- Пост-обработка (опционально) – добавление различных эффектов для улучшение визуального качества карты.

Алгоритм RandomWalk[12] (случайное блуждание) имитирует случайное перемещение виртуального "агента" по сетке, в результате чего формируются пещеры, лабиринты или другие интересные структуры.

В общем виде данный алгоритм можно разбить на несколько шагов:

- Инициализация сетки - Создается сетка, представляющую карту. Каждая ячейка может иметь состояние, такое как "стена" или "пустота". Изначально все ячейки могут быть заполнены состоянием "стена".

- Размещение стартовой точки: Выберите случайную ячейку на вашей сетке и установите ее состояние как "пустота". Это будет стартовая точка для агента.
- Движение агента: Агент начинает свое случайное блуждание по сетке. На каждом шаге агент перемещается в случайном направлении (вверх, вниз, влево или вправо) на одну ячейку.
- Изменение состояния ячеек: При каждом шаге агента изменяйте состояние ячеек вокруг его текущего положения. Например, вы можете установить состояние ячейки в "пустота", если она находится в определенном радиусе от агента.
- Повторение шагов: Повторяйте шаги 3 и 4, пока агент не достигнет определенного количества шагов или другого условия остановки.
- Пост-обработка (опционально) – добавление различных эффектов для улучшение визуального качества карты.

Данных алгоритмы позволяют создавать разнообразные карты, с различным количеством комнат и проходов, а их хаотичность и уникальность положительно влияет на игровой опыт. На рис. 8 представлен пример сгенерированной карты.

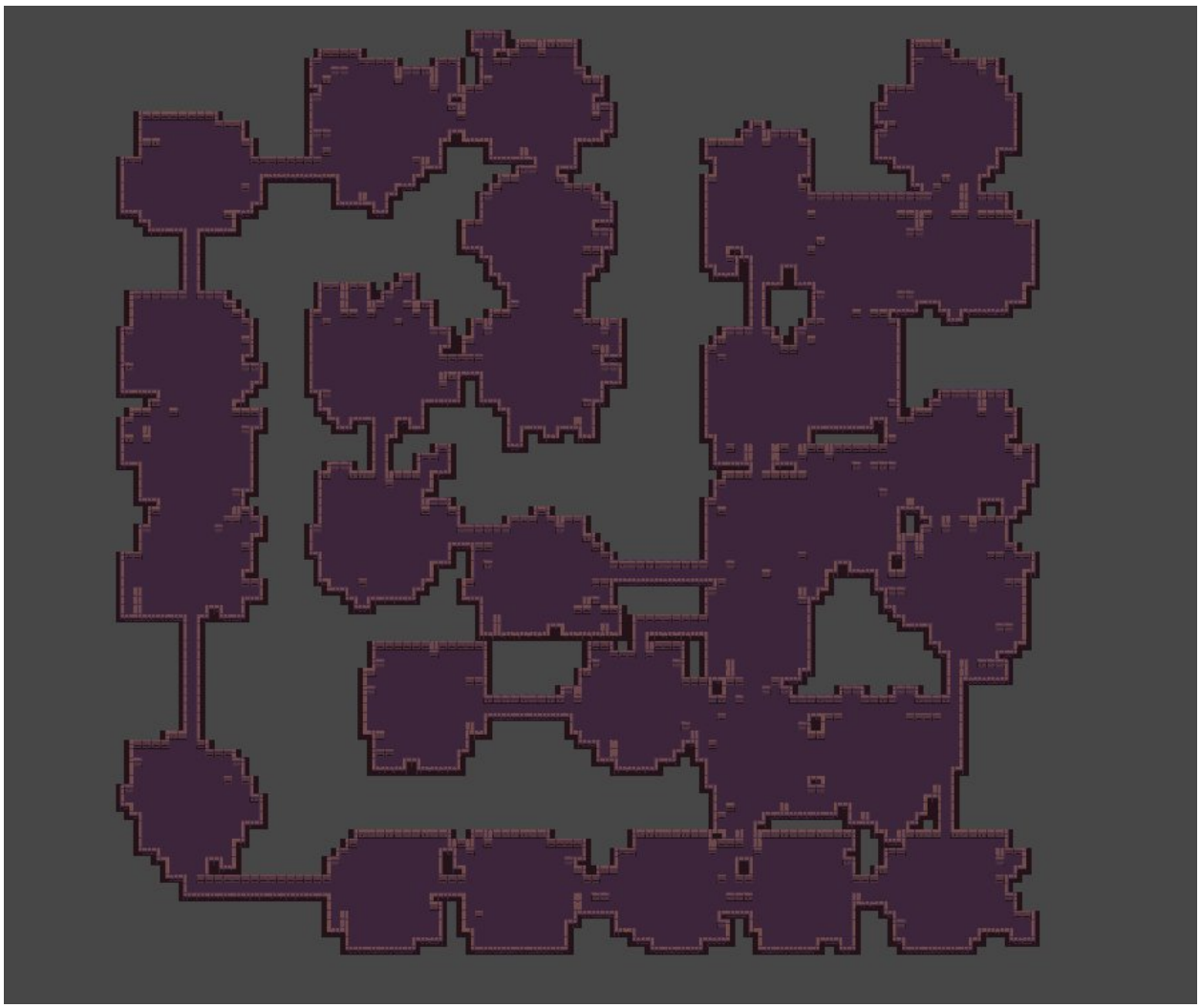

Рисунок 8 - Пример сгенерированной карты

### 3.3 Обучение противников

#### 3.3.1 Требуемые характеристики модели

Для того чтобы поведение противников казалось более чистым и уникальным, а также чтобы они выполняли поставленную цель, при обучение были выдвинуты следующие:

1. Обнаружение игрока: Модель противника должна быть способна обнаруживать наличие игрока в окружающей среде. Это может включать использование датчиков, алгоритмов поиска или анализа игрового поля для определения положения игрока;
2. Преследование игрока: Модель должна быть обучена принимать решения и преследовать игрока с целью минимизации расстояния

или достижения определенной позиции. Это может включать использование алгоритмов движения, таких как поиск пути, чтобы найти оптимальный маршрут к игроку;

3. Реакция на изменения среды: Модель должна быть гибкой и способной адаптироваться к изменениям в игровой среде. Это может включать обновление плана движения или изменение стратегии преследования, когда игрок меняет свое положение или взаимодействует с окружением;
4. Избегание препятствий: Модель должна обучаться избегать препятствия и препятствовать столкновениям с другими объектами в игровом мире. Это может включать анализ окружающей среды и выбор альтернативного маршрута или применение алгоритмов избегания препятствий.

Все эти требования позволяют создать противников, которые будут проявлять более интеллектуальное и реалистичное поведение в игре как наедине с игроком так и в положении большинства, что улучшит игровой опыт и сделает игру более интересной.

При обучение используются локации, на которых расположен игрок и 8 противников. Также в них присутствую разнообразные элементы окружения.

### 3.3.2 Описание данных, которые получает противник

В Unity[1], при обучении с подкреплением, observation (наблюдение) представляет собой информацию, которую агент получает о текущем состоянии игровой среды. Observation используется для принятия решений и обучения агента на основе полученных данных.

В Unity ML-Agents[2], observation может быть представлено в виде числовых массивов, изображений, векторов или других форматов данных. Observation включает в себя информацию о положении агента, его

окружении, объектах в сцене, а также другие сведения, которые агенту необходимы для принятия решений.

Наблюдение может быть как частичным (partial), когда агент видит только часть информации о среде, так и полным (full), когда агент имеет доступ ко всей информации. Частичное наблюдение может имитировать ограниченное видение агента или ограниченные данные о состоянии среды, что добавляет сложность в обучении и требует от агента принятия более осмысленных решений на основе ограниченной информации.

Важно правильно определить observation в соответствии с целями обучения и требованиями игры. Определение observation требует баланса между достаточным количеством информации для обучения и обработки, а также эффективным использованием вычислительных ресурсов.

Использование observation в Unity ML-Agents[2] позволяет агентам взаимодействовать с окружением, анализировать данные и принимать решения на основе полученной информации, что делает обучение с подкреплением более эффективным и гибким процессом. В таблице 2 представлена информация о наблюдениях используемых при обучении.

Таблица 2 – Описание используемых наблюдений

| Название | Описание | Тип | Количество переменных |
|---|---|---|---|
| Дистанция до стены между агентом и игроком | Дистанция до стены между агентом и игроком по прямой линии. Показывает видит ли агент игрока или нет. Если стены нет значение -1 | float | 1 |
| Направление к игроку | Направление в котором расположен игрок. Если есть стена – значение | Vector2 | 2 |

| | | | |
|---|---|---|---|
| | нулевой вектор | | |
| Дистанция до игрока | Расстояние от позиции агента до позиции игрока. Если | float | 1 |
| Наличие стен вокруг | От агента исходит 8 лучей которые сталкиваются со стенами. Наблюдение хранит дистанцию до стены. Если стены нет – значение -1 | float | 8 |
| Наличие других агентов вокруг | От агента исходит 8 лучей которые сталкиваются со другими агентами. Наблюдение хранит дистанцию до агента. Если агента нет – значение -1 | float | 8 |

### 3.3.3 Решения, которые принимает агент

В Unity ML-Agents[2], дискретные действия (discrete actions) являются одним из типов действий, которые агент может предпринимать в игровой среде. Дискретные действия ограничены конечным набором дискретных значений или категорий, которые агент может выбирать.

Когда агент принимает дискретное действие, он выбирает одно из возможных значений из определенного набора. Например, если агент управляет персонажем в игре, дискретные действия могут включать движение вперед, назад, влево, вправо или выпадение атаки. Агент может выбрать только одно из этих действий в каждый момент времени.

В Unity ML-Agents[2] дискретные действия могут быть представлены в виде целочисленных значений. Каждое дискретное действие имеет свое уникальное целочисленное значение, которое соответствует конкретной категории действия.

Для определения дискретных действий в Unity ML-Agents[2] необходимо настроить параметры действий в агентском скрипте. Можно определить количество дискретных действий и присвоить каждому действию соответствующее целочисленное значение.

Так как агенту нужно добраться до игрока, было выделена пара возможных действий: движение по оси X или Y. Каждое из действий задается набором от 0 до 10. При этом данные значения конвертируются в набор от -1 до 1 с шагом 0.2. Таким образом была достигнута гибкая система управления агентом, в которой агент может выбирать множество разнообразных направлений и регулировать скорость движения по осям в зависимости от принятых решений.

### 3.3.4 Система наград и наказаний

В обучении с подкреплением при использовании Unity ML-Agents[2] награды и наказания играют важную роль в формировании поведения агента. Они служат в качестве обратной связи для агента, помогая ему определить, какие действия приводят к положительным результатам и какие — к негативным.

Награды и наказания обычно определяются разработчиком игры на основе целей и правил игры. В процессе обучения агента, награды используются для поощрения желательных действий и стремления агента к достижению определенных целей. С другой стороны, наказания применяются для отрицательного воздействия на агента в случае нежелательных действий или нарушения правил.

Награды и наказания задаются в коде с помощью функций наград и штрафов. В Unity ML-Agents[2] это может быть реализовано с помощью

метода SetReward() для наград и EndEpisode() для наказаний (преждевременное завершение эпизода).

Например, предположим, что у нас есть агент-робот, который должен собирать монеты в игровой среде. Мы можем назначить положительную награду, когда агент собирает монету, с помощью метода SetReward(1f). Наоборот, если агент сталкивается со стеной, мы можем применить наказание, вызвав метод SetReward(-1f) и, возможно, завершив эпизод с помощью метода EndEpisode().

Определение подходящих наград и наказаний является важной частью процесса обучения с подкреплением. Необходимо балансировать награды и наказания таким образом, чтобы агент мог эффективно исследовать и учиться в среде, достигая поставленных целей, избегая нежелательного поведения и максимизируя получаемую награду на протяжении обучения.

Использование правильных наград и наказаний является ключевым элементом для достижения успешного обучения и формирования желаемого поведения агента в игре. Набор наград и наказаний использованный при обучение представлен в таблице 3.

При обучение был выставлен таймер на эпизод (30 секунд). По истечению таймера эпизод заканчивается.

Таблица 3 – Система наград и наказаний

| Название | Описание | Значение |
|---|---|---|
| Достижение игрока | Данная награда выдается за то что агент смог столкнуться с игроком. После чего эпизод заканчивается | 5 * на остаток времени в эпизоде |
| Приближение к игроку | Агент награждается за то, что приближается к игроку на 10 или меньше | 3 |

| | | |
|---|---|---|
| | | |
| Удаление от игрока | Агент наказывает за то, что удаляется от игрока больше чем на 10 | -3 |
| Сталкивание с окружением | За соприкосновение с окружением агент наказывается и эпизод перезапускается | -5 * на остаток времени таймера |
| Столкновение с другими агентами | За столкновение с другими агентами выдается наказание | -1 |

Данная система помогает достичь необходимых характеристик агента таких как:

- Преследование игрока;
- Избегание столкновения с окружением;
- Избегание столкновения с другими агентами;
- Сохранение баланса между агрессивностью и безопасностью

### 3.4 Подведение итогов разработки

В результате проделанной работы была разработана компьютерная игра в жанре «Roguelike». В ней были реализованы основные механики жанра такие как: случайная генерация уровней, перманентная гибель игрового персонажа при потери всех жизненных ресурсов, управление ограниченными ресурсами, а именно патронами и запасом здоровья.

Модели противников были разработаны с применением машинного обучения с помощью метода обучения с подкреплением. Для реализации

использовался игровой движок Unity[1] и пакет библиотек для машинного обучения Unity ML-Agents[2].

### 3.4.1 Преимущества использования противников обученных с помощью методов машинного обучения

Были выделены следующие преимущества применения противников реализованных с применением машинного обучения:

1. Поведение противников отличается даже при, казалось бы, незначительных изменениях игровой ситуации;
2. Реализация требует существенно меньшего количества написанного кода;
3. Агенты хорошо взаимодействуют между собой, что увеличивает реалистичность поведения.

При сравнение данной реализации с игрой The Binding of Isaac[8] и набором решений для игр в жанре «Roguelike» Roguelike от damiandiaz212[7] можно заметить, что реализации представленной в работе противник, находясь примерно в одинаковом положение чаще выбирают различные пути к игроку, в то время как в представленных аналогах одинаковые соперники чаще кучкуются.

### 3.4.2 Недостатки использования противников обученных с помощью методов машинного обучения

Были выделены следующие недостатки применения противников реализованных с применением машинного обучения:

1. Обучение противников требует много вычислительных ресурсов;
2. Одной реализации противников недостаточно для создания действительно интересной игры. Необходимо также разнообразить ее дополнительным контентом;

3. Создание противников обученных с помощью машинного обучения так или иначе занимает много времени и требует много внимания со стороны разработчика.

Исходя из данных недостатков можно сделать вывод, что машинное обучение не является волшебной возможностью для удешевления и улучшения производства игр, однако это мощный инструмент, который в умелых руках может показать достойные результаты.

В сравнении с играми Cogmind[6]и Caves of Qud[9] разработанная игра получилась более “сырой”. Однако данные игры разрабатывались командами разработчиков для коммерческого рынка, что объясняет большую проработанность игр. The Binding of Isaac[8] также имеет больший бюджет и время на разработку поэтому игра насыщена большим количеством контента.

# 4 Безопасность жизнедеятельности

При разработке программного обеспечения, включая пользовательский интерфейс, необходимо учесть множество аспектов, чтобы создать удобное и понятное решение, соответствующее потребностям пользователей.

Чтобы программное обеспечение соответствовало эргономическим стандартам был изучен ГОСТ Р ИСО 9241-100, который определяет структуру стандартов по эргономике программного обеспечения. В рамках этого стандарта были выделены принципы разработки диалога между пользователем и системой:

- Пригодность для выполнения задач
- Информативность
- Соответствие ожиданиям пользователя
- Пригодность для обучения
- Управляемость
- Устойчивость к ошибкам
- Пригодность для индивидуализации

Все эти принципы были учтены при разработке программного обеспечения, чтобы обеспечить его соответствие эргономическим стандартам.

## 4.1 Приемлемость организации диалога для выполнения поставленных задач

### 4.1.1 Предоставлении информации об успешном завершении производственного задания

Отображается счетчик поверженных врагов. При прохождение всего уровня появляется выпадающее окно с поздравлением.

#### 4.1.2 Избежание предоставления избыточной информации

Во время игры игроку доступна только для обзора только тот кусок уровня (комната) в которой он находится.

#### 4.1.3 Соответствие формата ввода и вывода производственному заданию

Реализованна система ввода соответствующая стандартным подходам игровой индустрии. Клавиши W,A,S,D отвечают за перемещение игрока, курс определяет направление взгляда игрока, левая кнопка мыши отвечает за выстрел.

### 4.2 Информативность

#### 4.2.1 Предоставленная пользователю информация на любом шаге диалога должна способствовать завершению диалога

Для начала игры, игроку необходимо выбрать определенный пункт в меню. После этого начинается игровой процесс, в котором игроку предстоит достигнуть своих целей и выполнить задачи, установленные в игре. По завершении уровня игрок возвращается в меню, где может выбрать опцию для продолжения игры или завершения текущей сессии.

#### 4.2.2 Необходимость в обращении к руководству пользователя и другой внешней информации должна быть сведена к минимуму

Основные механики объясняются игроку при старте игрового процесса. Более глубокая детализация заложена как часть исследования в рамках повторяющихся игровых сессий.

#### 4.2.3 Пользователя необходимо держать в курсе возможных изменений в состоянии интерактивной системы

Когда у игрока возникают проблемы с запуском игры или в процессе игровой сессии, ему будет показано специальное окно с информацией о

причинах возникших проблем и предложенными вариантами дальнейших действий. Это окно поможет игроку понять, почему игра не работает должным образом, и предоставит рекомендации о том, что можно сделать для решения проблемы.

### 4.3 Соответствие ожиданиям пользователя

#### 4.3.1 В интерактивной системе должна быть использована терминология, которую применяет пользователь при выполнении производственного задания

У игроков, которые играют в компьютерные игры в жанре Roguelike, нет необычной терминологии или специальных терминов, которые они используют.

#### 4.3.2 Пользователь должен быть обеспечен оперативной и удобной обратной связью, соответствующей его ожиданиям

Реализованная систему ввода является новейшей технологий представленной используемым программных обеспечение. Она обеспечивает четкость движений и скорость отклика.

#### 4.3.3 Если реальное время реакции системы на действие пользователя значительно отклоняется от времени, ожидаемого пользователем, то пользователь должен быть проинформирован об этом

В рамках игрового процесса компьютерной игры в жанре «Roguelike» все действия происходят в реальном времени.

#### 4.3.4 Форматы должны соответствовать культурным и лингвистическим соглашениям

Вся представленная информация в разработанной игре отображается на русском языки или с использованием цифр.

#### 4.3.5 Обратная связь или сообщения, предоставляемые пользователю, должны быть сформулированы и представлены в конструктивном стиле

Вся представленная информация содержит только необходимые зная о действиях отображаемых кнопок, правилах игры а также начале и конце игровой сессии.

### 4.4 Пригодность для обучения

#### 4.4.1 Правила и базовые концепции полезные для обучения, должны быть доступны пользователю

В меню игры представлена вкладка описывающая основные механики и возможности управления.

#### 4.4.2 Интерактивная система должна давать возможность пользователю выполнять производственное задание с минимальным изучением диалога

В разработанном программном обеспечение нет возможности переназначить клавиши управления. Используется схема заданная при разработке.

### 4.5 Контролируемость

#### 4.5.1 Темп взаимодействия между пользователем и системой не должен зависеть от функциональных возможностей и ограничений интерактивной системы

Игрок имеет возможность самостоятельно выбирать с какой скоростью и в каком направление двигаться, а также он в любой момент может поставить игровую сессию на паузу или завершить ее.

### 4.5.2 Пользователь должен иметь возможность выбора вариантов продолжения диалога

В любой момент времени и на любом этапе диалога пользователь имеет выбор. Как на начале игры пользователь может выбирать из пунктов меню, так и вовремя игры имеет различные возможности управление а так же возможность поставить на паузу или завершить сессию.

## 4.6 Устойчивость к ошибкам

Игрок в любой момент может завершить игровую сессию по своему усмотрению. Однако жанр игры подразумевает полное изменение игрового окружения при каждой новой сессии, с сохранением основных механик. Исследование новых подземелий подразумевает накопление опыта за попытки прохождения предыдущих.

## 4.7 Пригодность для индивидуализации

Пользователь не ограничен в своих решения по исследованию уровней. Он самостоятельно может выбирать каким именно способом решать поставленные задачи. А также имеет возможно комфортно настроить громкость игры.

## 4.8 Выводы

По изучение стандартов ГОСТ Р ИСО 9241-110-2016 были выделены требования, которые необходимо учесть при разработке компьютерной игры в жанре «Roguelike».

Разработанная компьютерная игра учитывает и соответствует всем выдвинутым требованиям.

# ЗАКЛЮЧЕНИЕ

По итогам работы было проведено исследование предметной области, которое показало, что жанр компьютерных игр «Roguelike» развивается уже больше 40 лет, имеет много последователей и является неотъемлемой частью игровой индустрии.

Также были рассмотрены технологии для разработки игры и обучения интеллектуальных агентов. Для разработки игры были выбран игровой фреймворк Unity[1], имеющий большое количество полезных материал помогающих в разработке игр. В связке с ним был использованы пакет библиотек Unity ML-Agents[2], предназначенный для использования машинного обучения в разработке игр. Он реализует как библиотеки для создания сценариев обучения, так и для использования обученных агентов в игровом процессе. Также он реализует API для взаимодействия с популярными фреймворками для машинного обучения такими как PyTorch и Tensorflow. В добавок к вышеперечисленным технологиям использовалась IDE Rider[5] от компании JetBrains, которая позволяет удобно описывать скрипты игровых механик.

Было рассмотрены методы машинного обучения применяемые при разработке компьютерных игр в жанре «Roguelike». Наиболее детально рассматривались методы обучения с подкреплением и имитационного обучения, так как их реализации представлены в выбранном стэке технологий.

Для разработки игры был выбран метод обучения с подкреплением так как он не требует заготовленных датасетов и может быть применен непосредственно в игровом окружение. Данный метод использует систему наград и наказаний, а обучаемые агенты стремятся набрать максимальный счет до завершения эпизода.

Также был проведен обзор аналогов, который показал, что поведение противников, обученных с помощью машинного, превосходит их более

простые аналоги в вариативности принимаемых решений и создании уникальных игровых ситуаций.

На основании полученной информации была разработана двухмерная компьютерная игра в стиле PixelArt соответствующая жанру «Roguelike», противники в которой были обучены с помощью машинного обучения. В ходе реализации механик присущих жанру была разработана случайная генерация уровней на основе алгоритма BSP Tree[11].

На основании опыта полученного в разработке были выявлены преимущества и недостатки применения противников, обученных с применением машинного обучения в игра данного жанра.

В качестве преимуществ был выделен следующий ряд заключений:

1. Поведение противников отличается даже при, казалось бы, незначительных изменениях игровой ситуации;
2. Реализация требует существенно меньшего количества написанного кода;
3. Агенты хорошо взаимодействуют между собой, что увеличивает реалистичность поведения.

В качестве недостатков были отмечены следующие пункты:

1. Обучение противников требует много вычислительных ресурсов;
2. Одной реализации противников недостаточно для создания действительно интересной игры. Необходимо также разнообразить ее дополнительным контентом;
3. Создание противников обученных с помощью машинного обучения так или иначе занимает много времени и требует много внимания со стороны разработчика.

На основании проведенных исследований был сделан вывод, что машинное обучения, как инструмент создания противников не является ультимативным, представляет собой достаточно мощное средство разработки, которое, в совокупности с другими элементами разработки игр,

может показать хорошие результаты и вывести игры данного жанра на новый технологический уровень.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Unity [Электронный ресурс]. URL: https://unity.com/ru (дата обращения: 24.03.2021).
2. Unity ML-Age [Электронный ресурс]. URL: https://github.com/Unity-Technologies/ml-agents (дата обращения: 24.03.2021).
3. История жанра Roguelike [Электронный ресурс]. URL: https://habr.com/ru/articles/493890/ (дата обращения: 24.03.2021).
4. Машинное обучение для людей [Электронный ресурс]. URL: https://vas3k.blog/blog/machine_learning/ (дата обращения: 24.03.2021).
5. IDE Rider [Электронный ресурс]. URL: https://www.jetbrains.com/rider/ (дата обращения: 24.03.2021).
6. Игра Cogmind [Электронный ресурс]. URL: https://www.gridsagegames.com/cogmind/ (дата обращения: 24.03.2021).
7. Unity RogueLike Project от damiandiaz212 [Электронный ресурс]. URL: https://github.com/damiandiaz212/roguelike (дата обращения: 24.03.2021).
8. Игра The Binding of Isaac [Электронный ресурс]. URL: https://bindingofisaac.fandom.com/wiki/The_Binding_of_Isaac_Wiki (дата обращения: 24.03.2021).
9. Игра Caves of Qud [Электронный ресурс]. URL: https://www.cavesofqud.com/ (дата обращения: 24.03.2021).
10. Aseprite [Электронный ресурс]. URL: https://www.aseprite.org/ (дата обращения: 24.03.2021).
11. BSP Tree algorithm [Электронный ресурс]. URL: https://www.geeksforgeeks.org/binary-space-partitioning/# (дата обращения: 24.03.2021).
12. RandomWalk algorithm [Электронный ресурс]. URL: https://www.baeldung.com/cs/random-walk (дата обращения: 24.03.2021).