МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра многопроцессорных систем и сетей

Стефанович
Артём Витальевич

ТЕХНОЛОГИЧЕСКИЕ АСПЕКТЫ РАЗРАБОТКИ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА В ИГРОВЫХ ПРИЛОЖЕНИЯХ НА ИГРОВОМ ДВИЖКЕ UNITY

Дипломная работа

Научный руководитель:
старший преподаватель
В. В. Рябый

Допущена к защите
«___»_____2025 г.
Зав. кафедрой многопроцессорных систем и сетей
кандидат физ.-мат. наук, доцент И.Е. Андрушкевич

Минск, 2025

РЕФЕРАТ

Дипломная работа 50 с., 17 рис., 4 табл., 22 источн.

Ключевые слова: ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, UNITY, ИГРОВЫЕ АЛГОРИТМЫ, ИГРОВАЯ ЛОГИКА, ПОВЕДЕНЧЕСКИЕ ПАТТЕРНЫ, СТРАТЕГИЧЕСКИЕ ИГРЫ, КОМПЬЮТЕРНЫЙ ПРОТИВНИК.

Объект исследования – процесс разработки компьютерного противника в стратегических играх на платформе Unity.

Цель работы – разработка прототипа интеллектуального агента для игры жанра стратегия на игровом движке Unity.

Методы исследования: анализ и исследование существующих методов, поиск и подготовка данных, оценка и тестирование конечного алгоритма игрового приложения.

В процессе работы проводилась экспериментальная разработка архитектуры компьютерного соперника в игре жанре стратегия.

В результате исследования был создан компьютерный соперник, обладающим, с использованием подхода Utility AI, настраиваемым поведением.

Область применения включает игровые проекты, направленные на создание разнообразных режимов в игровом приложении и уникального игрового опыта, что способствует повышению конкурентоспособности на рынке игровых приложений.

РЭФЕРАТ

Дыпломная праца 50 с., 17 мал., 4 табл., 22 крыніц.

Ключавыя словы: ШТУЧНЫ ІНТЭЛЕКТ, UNITY, ГУЛЬНЯВЫЯ АЛГАРЫТМЫ, ГУЛЬНЯВАЯ ЛОГІКА, ПАВОДЗІННЫЯ ПАТЭРНЫ, СТРАТЭГІЧНЫЯ ГУЛЬНІ, КАМП'ЮТАРНЫ ПРАЦІЎНІК.

Аб'ект даследавання – працэс распрацоўкі кампутарнага суперніка ў стратэгічных гульнях на платформе Unity.

Мэта працы – распрацоўка прататыпа інтэлектуальнага агента для гульні жанру стратэгія на гульнявым рухавічку Unity.

Метады даследавання: аналіз і даследаванне існуючых метадаў, пошук і падрыхтоўка дадзеных, ацэнка і тэставанне канчатковага алгарытму гульнявога дадатку.

У працэсе працы праводзілася эксперыментальная распрацоўка архітэктуры камп'ютарнага саперніка ў гульні жанру стратэгія.

У выніку даследавання быў створаны камп'ютарны сапернік, які валодае, з выкарыстаннем падыходу Utility AI, наладжвальнымі паводзінамі.

Вобласць прымянення – гульнявыя праекты з мэтай стварэння разнастайнасці рэжымаў у гульнявым дадатку і ўнікальнага гульнявога досведу для павышэння канкурэнтаздольнасці на рынку гульнявых дадаткаў.

ABSTRACT

Diploma thesis 50 p., 17 fig., 4 tables, 22 sources.

Keywords: ARTIFICIAL INTELLIGENCE, UNITY, GAME ALGORITHMS, GAME LOGIC, BEHAVIORAL PATTERNS, STRATEGY GAMES, COMPUTER OPPONENT.

Object of research – the process of developing a computer opponent in strategy games on the Unity platform.

Aim of work – development of a prototype of an intelligent agent for a strategy game on the Unity game engine.

Research methods: analysis and research of existing methods, data search and preparation, evaluation and testing of the final algorithm of the game application.

During the work, experimental development of computer opponent in a strategy genre game was carried out.

As a result of the research, a computer opponent was created, possessing configurable behavior using the Utility AI approach.

Field of application – game projects aimed at creating a variety of modes in the game application and unique gaming experience to increase competitiveness in the game applications market.

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

В настоящей работе применяют следующие термины с соответствующими определениями:

Ассеты (assets) – это ресурсы, используемые в разработке игр и приложений.

Геймплей – это совокупность механик и взаимодействий, которые игрок испытывает во время игры.

Инди-разработка – это процесс создания видеоигр независимыми разработчиками или небольшими командами, без поддержки крупных издателей.

Платформер – это жанр видеоигр, в котором игрок управляет персонажем, перемещающимся по уровням, состоящим из платформ.

Шутер – это жанр видеоигр, в котором основной акцент сделан на стрельбу и боевые действия.

Юнит – Название боевой/рабочей единицы, предполагающей возможность управления в компьютерных играх жанра «Стратегия» и «RPG».

MMO (Massively Multiplayer Online) – компьютерная онлайн-игра, в которой большое количество игроков взаимодействует друг с другом в «постоянном» игровом мире, расположенном на удалённом сервере.

NPC (Non-Player Character) – любой персонаж в игре, который не контролируется игроком.

RPG (Role-Playing Game) – жанр компьютерных игр, в котором игрок управляет одним или несколькими персонажами, каждый из которых описан набором численных характеристик, списком способностей и умений.

ВВЕДЕНИЕ

В современном мире игры стали неотъемлемой частью нашей культуры и развлечения, и их популярность продолжает расти. Разработка игровых приложений объединяет в себе творческий дизайн, программирование и инновационные идеи, чтобы создать увлекательные и захватывающие игровые опыты для пользователей.

Индустрия игр постоянно растёт. По данным ресурса Business of App оборот этой отрасли в прошлом году составил 77.2 миллиарда долларов. И в 2020 году количество игроков по сравнению с 2019 из-за пандемических ситуаций в мире выросло сразу на 12%. На сегодняшний день в мире играет практически треть населения, около 2.5 млрд человек. Особую роль в развитии игровой индустрии играет искусственный интеллект (ИИ), который становится все более сложным и адаптивным. Современные игровые движки, такие как Unity. предоставляют разработчикам мощные инструменты для создания продвинутых систем ИИ в играх. В стратегических играх искусственный интеллект отвечает за поведение компьютерных противников, принятие тактических решений и адаптацию к действиям игрока, что создает более реалистичный и захватывающий игровой процесс.

Объект исследования – процесс разработки компьютерного противника в стратегических играх на платформе Unity.

Предмет исследования – методы, инструменты и алгоритмы создания компьютерного противника в игровой среде Unity.

Цель работы – разработка прототипа интеллектуального агента для игры жанра стратегия на игровом движке Unity.

Для достижения поставленной цели необходимо решить следующие задачи:

1) Выполнить анализ разработки игр на фреймворке Unity;

2) Исследовать возможности применения искусственного интеллекта в игровых приложениях;

3) Изучить функционал и инструменты игрового движка Unity;

4) Проанализировать особенности реализации стратегических игр;

5) Разработать архитектуру для стратегической игры с компьютерным соперником.

6) Реализовать алгоритмы поведения компьютерного соперника в Unity.

Таким образом, работа вносит вклад в развитие методов создания стабильных и масштабируемых многопользовательских игр, сочетая проверенные решения с адаптацией под специфику стратегического жанра.

ГЛАВА 1 АНАЛИЗ ПЕРСПЕКТИВНОСТИ СФЕРЫ РАЗРАБОТКИ ИГР НА UNITY И ИСПОЛЬЗОВАНИЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА В ИГРОВЫХ ПРИЛОЖЕНИЯХ

1.1 Анализ перспективности сферы разработки игр на Unity

1.1.1 Актуальность разработки игр

Сегодня компьютерные игры являются одним из популярнейших развлечений и имеют как широкую аудиторию, так и большой спектр платформ: игровые автоматы, компьютеры, телефоны, планшеты и так далее. Если раньше компьютерные игры выпускались в основном различными профессиональными студиями разработчиков игр, то в наши же дни широкую популярность имеет инди-разработка игр. Инди-разработка отличается от разработки игры крупными студиями тем, что, как правило, в этом случае игру разрабатывает один человек или небольшая группа людей. Кроме того, инди-игры обладают небольшим бюджетом, а иногда и его отсутствием. Неплохим подспорьем в данной ситуации будет использование готовых игровых движков – это снизит как денежные, так и временные затраты на разработку [1].

Развитие игровой индустрии является одной из самых быстрорастущих и прибыльных отраслей в мире. Компьютерные игры стали популярным развлечением для миллионов людей по всему миру, предлагая огромный выбор игр по жанрам, игровому сеттингу и игровым режимам.

Кроме того, игры имеют культурное и художественное значение, они могут быть художественным произведением, которое затрагивает множественные проблемы современного общества и, возможно, дающие возможные ответы на них. Они могут также рассказывать увлекательные истории, вызывать эмоции у игроков и предлагать порой тяжелые с точки зрения морали выборы. Таким образом, игры также могут развивать эмоциональный интеллект у игроков и передавать им определенные ценности и мировоззрение.

Разработка игровых приложений предоставляет возможности для разработчиков, где они могут реализовать созданные воображением свои миры и делиться ими с другими людьми. Для этого от них требуются технические навыки, творчество и инновационные идеи для создания уникальных и захватывающих игр.

1.1.2 Рассмотрение аналогов

При разработке игр игровые движки играют ключевую роль, поскольку они предоставляют разработчикам мощные инструменты и функциональность для создания игровых приложений. Они предлагают готовые решения и инструменты, которые значительно упрощают процесс разработки игр. Поэтому выбор игрового движка является решающим для игрового проекта.

Игровые движки общего назначения предоставляют разработчикам широкий набор функциональности и инструментов для создания различных типов игр. Примеры таких движков включают Unity, Unreal Engine и Godet. Существуют также специализированные игровые движки, которые предназначены для разработки игр в определенных жанрах или для конкретных платформ. Например, RPG Maker, Adventure Game Studio, GameMaker и Cocos2 являются движками, специализированными для создания текстовых приключений, платформеров, RPG или MMO игр.

Для инди-разработчиков с ограниченными ресурсами существуют инди-движки, которые обеспечивают более легковесные и доступные инструменты для создания прототипов и разработки небольших проектов. Примеры таких движков включают GameMaker, Construct и Stency.

Архитектурные движки используются в архитектурной и строительной индустрии для создания виртуальных моделей и симуляций. Unity. Unreal Engine и Lumion являются примерами архитектурных движков, которые позволяют архитекторам и дизайнерам создавать интерактивные визуализации зданий, планировать пространство и тестировать различные концепции.

Наконец, облачные движки предоставляют возможность разработки и запуска игр непосредственно в облаке, без необходимости загрузки и установки на конкретное устройство. Примеры таких движков включают Amazon Lumberyard, Google Stadia и Microsoft Azure PlayFab. Эти движки позволяют переносить содержание игры на различные платформы и устройства.

Каждый из этих разновидностей игровых движков имеет свои преимущества и выбор зависит от потребностей и требований разработчиков и их проектов.

«На сегодняшний день самыми популярными игровыми движками являются:

- Unity.
- Unreal Engine.
- GameMaker Studio» [14].

«*GameMaker Studio* – обучающий движок, рассчитанный на начинающих разработчиков, который помогает создавать 2D игры. Часть процессов здесь адаптированы для людей, плохо знающих языки программирования, и реализованы путем обычного перетаскивания нужных элементов мышью.

Unreal Engine – это еще один из популярных игровых движков, разработанный компанией Epic Games. Он также предоставляет возможности для разработки игр в 2D и 3D. Unreal Engine известен своим мощным графическим рендерингом и физическим движком. Он широко используется для создания высококачественных игр и виртуальной реальности» [14].

Unity – один из самых популярных и широко используемых игровых движков. Он предоставляет мощные инструменты для разработки игр как в 2D, так и в 3D. Unity поддерживает множество платформ, включая ПК, консоли, мобильные устройства и виртуальную реальность. Он имеет обширную документацию и активное сообщество разработчиков.

1.1.3 Результат сравнения

Основываясь на особенностях каждого из игровых движков, выбор был сделан в сторону Unity, т.к. по скорости разработки и доступным возможностям другие движки уступают Unity.

Для реализации многопользовательского режима в Unity разработчики могут использовать различные технические решения. Каждое из этих решений предоставляет инструменты для синхронизации игровых объектов между игроками, обработки сетевого взаимодействия и управления подключениями клиентов. Выбор конкретного решения зависит от масштаба проекта, бюджета и требований к производительности, при этом все они позволяют реализовать как клиент-серверную архитектуру, так и прямое соединение между игроками. В таблице 1.1 показано сравнение игровых платформ.

Таблица 1.1 Сравнение игровых платформ

Критерии	**Unity**	**Unreal Engine**	**GameMaker Studio**
Порог входа	Средний – легко начать	Высокий	Низкий
Язык программирования	C# - популярный и простой	C++ - сложнее для новичков	GML – простой, но менее универсальный
Документация и обучение	Обширная, много уроков	Хорошая, но сложнее	Базовая
Стоимость	Бесплатно до $100K дохода	Бесплатно до $1M дохода	Платная подписка
Сообщество	Огромное, активное	Большое	Небольшое
Asset Store	Огромный выбор ассетов	Хороший выбор, дороже	Ограниченный выбор
Производительность	Хорошая	Отличная	Средняя
Кроссплатформенность	25+платформ	Основные платформы	Ограниченное количество
2D разработка	Отличная поддержка	Ограниченная поддержка	Специализируется на 2D
3D разработка	Хорошая поддержка	Превосходная поддержка	Ограниченная поддержка

1.2 Анализ перспективности использования искусственного интеллекта в игровых приложениях

1.2.1 Актуальность искусственного интеллекта

В современном мире ИИ становится ключевым драйвером технологического прогресса, трансформируя традиционные отрасли экономики и создавая новые возможности для развития бизнеса, науки и общества в целом. От медицины и промышленного производства до финансового сектора и образования искусственный интеллект демонстрирует способность существенно повышать эффективность процессов, автоматизировать рутинные задачи и открывать новые горизонты для инноваций.

В современном мире технологии искусственного интеллекта активно применяются для анализа больших данных, прогнозирования трендов, оптимизации логистических цепочек, персонализации сервисов и принятия управленческих решений. Особую значимость ИИ приобретает в контексте цифровой трансформации общества, где интеллектуальные системы становятся незаменимыми помощниками в решении сложных аналитических задач, обработке естественного языка, компьютерном зрении и роботизации производственных процессов. Развитие технологий машинного обучения и нейронных сетей открывает новые возможности для создания более совершенных алгоритмов, способных не только выполнять заданные функции, но и адаптироваться к изменяющимся условиям, учиться на основе

накопленного опыта и принимать все более сложные решения в условиях неопределенности.

1.2.2 Актуальность использования искусственного интеллекта в игровых приложениях

На современной игровой индустрии активно растут требования к качеству игрового опыта и необходимостью создания более глубоких и захватывающих игровых механик. Для удовлетворения данных потребностей активно используется ИИ, который становится ключевым инструментом в создании динамических игровых миров, где неигровые персонажи (NPC) демонстрируют реалистичное поведение, а игровые системы адаптируются к действиям и предпочтениям каждого игрока.

Поведенческие системы NPC обеспечивают реалистичные реакции персонажей на действия игрока и изменения игрового мира. Динамическое повествование позволяет создавать адаптивные сюжетные линии, меняющиеся в зависимости от выборов игрока, а оптимизация игрового баланса помогает автоматически настраивать игровые параметры для достижения оптимального геймплея.

Перспективы развития искусственного интеллекта в игровой индустрии открывают захватывающие возможности для создания еще более впечатляющих игровых проектов. Внедрение более совершенных систем машинного обучения позволит создавать по-настоящему «умных» противников, способных учиться и адаптироваться к стилю игры пользователя. Развитие технологий естественного языка обеспечит более реалистичное диалоговое взаимодействие с NPC, а использование нейронных сетей откроет новые горизонты для генерации уникального игрового контента Б реальном времени. Создание персонализированных игровых сценариев, адаптирующихся под стиль игры каждого пользователя, и интеграция эмоционального ИИ позволят достичь беспрецедентного уровня погружения в игровой процесс.

Таким образом, в стратегических играх искусственный интеллект приобретает особое значение для создания конкурентоспособных виртуальных противников, способных анализировать ситуацию, разрабатывать сложные стратегии и принимать тактические решения, приближенные к человеческим. Это позволяет обеспечить высокий уровень сложности и реиграбельности, даже когда игрок играет в одиночном режиме.

ГЛАВА 2 ОБЩИЕ СВЕДЕНИЯ О UNITY

2.1 Игровой Движок

Игровой движок – это программное обеспечение, которое предоставляет разработчикам игр инструменты и функции для создания и управления игровым контентом. Движок позволяет сосредоточиться на разработке игрового процесса, графики и звука, не тратя время на создание базовой инфраструктуры с нуля. Использование игрового движка значительно упрощает процесс разработки и позволяет быстрее воплощать идеи в жизнь. В современном мире, где игры становятся все более сложными и требовательными, наличие мощного игрового движка становится критически важным для успешной реализации проекта.

Игровые движки предоставляют разработчикам широкий спектр возможностей, начиная от базовых инструментов для создания простых игр и заканчивая сложными системами для разработки высококачественных AAA-проектов. Они включают в себя различные модули и компоненты, которые помогают автоматизировать и оптимизировать процесс разработки. Это позволяет разработчикам сосредоточиться на творческом аспекте создания игры, а не на технических деталях.

Игровой движок состоит из нескольких ключевых компонентов, каждый из которых выполняет свою роль в создании игры. Понимание этих компонентов поможет вам лучше разобраться в том, как работает игровой движок и какие возможности он предоставляет.

Графический движок отвечает за рендеринг (отрисовку) графики в игре. Он обрабатывает 2D и 3D модели, текстуры, освещение и эффекты, такие как тени и отражения. Графический движок обеспечивает высокую производительность и качество изображения, что важно для создания визуально привлекательных игр. Современные графические движки поддерживают передовые технологии, такие как трассировка лучей, что позволяет создавать реалистичные визуальные эффекты и улучшать общее качество изображения.

Графический движок также включает в себя инструменты для оптимизации производительности, такие как уровни детализации (LOD) и системы управления ресурсами. Эти инструменты помогают разработчикам создавать игры, которые могут работать на различных устройствах с разной производительностью, от мощных игровых ПК до мобильных телефонов.

Физический движок моделирует физические взаимодействия объектов в игре. Он отвечает за такие аспекты, как столкновения, гравитация, трение и другие физические свойства. Благодаря физическому движку объекты в игре

ведут себя реалистично, что делает игровой процесс более захватывающим и правдоподобным. Физический движок также может включать в себя системы разрушений и деформаций, которые позволяют создавать динамические и интерактивные игровые миры.

Особую значимость в современном контексте приобретает мультиплатформенная совместимость игровых движков. Передовые решения в этой области обеспечивают возможность создания универсальных проектов, адаптируемых под различные вычислительные архитектуры и операционные среды. Такая технологическая гибкость позволяет существенно расширить охват целевой аудитории, охватывая пользователей персональных компьютеров, игровых консолей, мобильных устройств и систем виртуальной реальности. Унифицированный подход к разработке значительно оптимизирует ресурсозатраты предоставляя возможность использования единой технологической базы для мультиплатформенной дистрибуции, одновременно гарантируя консистентность пользовательского опыта среди различных устройств.

Существенным преимуществом современных движков является расширяемость через создание кастомизированных инструментов разработки. Данная функциональность охватывает широкий спектр утилит: конструкторов игровых уровней до специализированных редакторов игровых ресурсов, систем конфигурации параметров и отладочного инструментария. Внедрение персонализированных инструментов существенно интенсифицирует процесс разработки, повышая эффективность и качество производственного процесса.

В итоге, игровые движки выступают как передовые технологические платформы, радикально упрощающие и ускоряющие процесс создания цифровых развлекательных продуктов. Их многогранная функциональность и гибкость позволяют разработчикам создавать высококачественные интерактивные произведения. Благодаря этим технологическим решениям, креативные концепции трансформируются в захватывающие цифровые миры, доставляющие удовольствие глобальному игровому сообществу.

2.2 Основы Unity

«*Unity* – кроссплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Выпуск Unity, состоялся в 2005 году и с того времени идёт постоянное развитие.

Редактор Unity имеет простой Drag&Drop интерфейс, состоящий из различных окон, благодаря чему можно производить отладку игры прямо в редакторе. Движок использует для написания скриптов C#. Ранее поддерживались также Boo (диалект Python, поддержку убрали в 5-й версии) и модификация JavaScript, известная как UnityScript (поддержка прекращена в Версии 2017.1) Расчёты физики производит физический движок PhysX от NVIDIA для 3D физики и Box2D для 2D физики. Графический API - DirectX (на данный момент DX 11, поддерживается DX 12)» [14].

Архитектура разработки в Unity, организована через систему интерактивных сцен, выступающих в качестве независимых виртуальных пространств с уникальной конфигурацией элементов, программных инструкций и конфигурационных параметров. В рамках каждой сцены разработчик оперирует двумя фундаментальными категориями объектов: визуализируемыми сущностями с геометрической репрезентацией и абстрактными элементами без визуального воплощения. Функциональность этих объектов обеспечивается интегрированными модулями, взаимодействующими с программными скриптами.

Каждый объект характеризуется идентификационным названием (с возможностью дублирования в пределах сцены), опциональной категоризацией через теги и принадлежностью к определенному слою визуализации. Ключевым атрибутом любой сущности выступает Transorm-компонент, определяющий пространственные характеристики по трем измерениям.

Платформа интегрирует продвинутые физические симуляции, включая динамику твердых тел, деформацию тканей и систему Ragdoll. Реализована иерархическая структура объектов, где подчиненные элементы наследуют трансформационные изменения родительской сущности. Программные скрипты интегрируются как модульные компоненты объектов.

Система текстурирования предоставляет широкий функционал генерации специализированных карт (alpha, mip, normal, light, reflection), при этом текстуры применяются через промежуточный слой материалов с настраиваемыми шейдерами. Встроенный редактор поддерживает как создание анимаций, так и импорт готовых анимационных последовательностей из специализированных 3D-пакетов.

Производительность оптимизируется через систему LOD, динамически адаптирующую детализацию моделей в зависимости от дистанции наблюдения, и механизм Occlusion culling, исключающий из обработки невидимые объекты. Компиляция проекта генерирует исполняемый файл с сопутствующими игровыми ресурсами и библиотеками.

Экосистема Unity, поддерживает широкий спектр форматов и включает инфраструктуру для обмена ресурсами через Unity Asset Store. Интегрированный Unity Asset Server обеспечивает инструментарий для коллаборативной разработки с системой контроля версий [14].

Платформа выделяется среди конкурентов визуальным подходом к разработке, мультиплатформенной поддержкой и компонентно-ориентированной архитектурой. При этом существуют определенные ограничения в работе с комплексными сценами, внешними библиотеками и prefab-шаблонами, а WebGL-реализация сталкивается с проблемами производительности и оптимизации.

2.3 Язык Разработки

C# – инновационный язык программирования, созданный корпорацией Microsoft в начале нового тысячелетия, который произвел революцию в экосистеме .NET. Этот передовой инструмент разработки воплотил в себе лучшие характеристики семейства C-подобных языков, одновременно предлагая более совершенный уровень абстракции и современный инструментарий для создания программного обеспечения.

Фундаментальная философия C# базируется на парадигме объектно-ориентированного программирования, реализуя такие ключевые концепции как инкапсуляция, наследование и полиморфизм. Эти принципы позволяют архитекторам программного обеспечения создавать масштабируемые и адаптивные решения.

Выбор C# в качестве приоритетного языка для разработки в среде Unity обусловлен рядом существенных преимуществ:

1. *Исключительная производительность:* Архитектура C# предусматривает компиляцию в промежуточный код (IL) с последующим исполнением на виртуальной машине .NET, что обеспечивает впечатляющую скорость работы приложений и оптимальную производительность игровых проектов.

2. *Глубокая синергия с Unity*. Язык демонстрирует безупречную интеграцию с Unity API, открывая разработчикам доступ к обширному функционалу движка. Программисты получают возможность эффективно манипулировать игровыми объектами, управлять физикой, звуковым сопровождением и другими аспектами геймдизайна.

3. *Низкий порог вхождения.* Синтаксис C# имеет схожие элементы с другими языками программирования, что делает его более доступным для новичков и облегчает переход от других языков.

4. Поддержка множества операционных систем. С использованием Xamarin, на C# можно создавать программы и приложения для различных операционных систем, включая iOS, Android, MacOS и Linux.

5. Фиксированный размер типов данных. Наличие фиксированного размера типов данных, таких как 32-битные int и 64-битные long, обеспечивает «подвижность» языка и упрощает программирование.

6. Эргономичность разработки: Элегантный синтаксис C# и наличие продвинутых языковых конструкций, включая LINQ, существенно упрощают процесс программирования и повышают читаемость кода.

Таким образом, симбиоз C# и Unity представляет собой мощную платформу для реализации игровых проектов, где технологическое совершенство языка гармонично сочетается с богатством возможностей игрового движка, создавая оптимальную среду для воплощения творческих замыслов разработчиков.

2.4 Реализации искусственного интеллекта в Unity

Реализация искусственного интеллекта Б Unity предоставляет разработчикам широкий спектр инструментов и возможностей для создания интеллектуального поведения игровых объектов. Unity предлагает встроенную систему навигации NaxMesh, которая позволяет реализовать базовое перемещение персонажей по игровому миру с учетом препятствий и особенностей ландшафта. Для более сложных поведенческих паттернов платформа поддерживает различные подходы к реализации ИИ, включая конечные автоматы (Finite State Machines), деревья поведения (Behavior Trees) и системы на основе машинного обучения через ML-Agents Toolkit.

ML-Agents – фреймворк, позволяющий создавать, тренировать и внедрять обучаемых агентов с использованием методов глубокого обучения и обучения с подкреплением. Этот инструмент предоставляет возможность разработчикам реализовывать сложные системы принятия решений, где агенты могут учиться на основе своего опыта взаимодействия с игровой средой.

Таким образом, Unity подтверждает свою позицию как одного из ведущих игровых движков благодаря сочетанию удобства, функциональности и широкой поддержки платформ. Интеграция с C# и инструментами ИИ делает его мощным решением для разработки современных игр с интеллектуальными системами. В дальнейшем Unity продолжит эволюционировать, предлагая новые технологии для создания инновационных игровых проектов.

ГЛАВА 3 ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

3.1 Общие сведения

Искусственный интеллект (ИИ) – это направление информатики, нацеленное на создание компьютерных систем, способных выполнять задачи, традиционно требующие человеческого интеллекта. Эти задачи включают распознавание речи и изображений, принятие решений, перевод между языками и понимание естественного языка.

История ИИ началась в 1950-х годах, когда Алан Тьюринг предложил свой знаменитый тест для определения интеллектуальности машины. В последующие десятилетия развитие шло через различные этапы: от простых экспертных систем до современных нейронных сетей и глубокого обучения. Особенно значительный прогресс произошел в 2010-х годах благодаря увеличению вычислительной мощности компьютеров и появлению больших наборов данных.

Современные научные изыскания в области искусственного интеллекта (ИИ) охватывают множество специализированных направлений, каждое из которых характеризуется собственным методологическим аппаратом и прикладными задачами. Ключевые векторы исследовательской деятельности включают разработку систем логического анализа, создание механизмов структурирования информации, совершенствование стратегического планирования, развитие технологий машинного обучения, разработку методов лингвистического анализа, совершенствование систем сенсорного восприятия и развитие робототехнических комплексов. Особое место в данной области занимает концепция универсального искусственного интеллекта, способного конкурировать с человеческим разумом в решении широкого спектра задач.

В современную эпоху наблюдается значительное усиление роли технологий искусственного интеллекта, что провоцирует масштабные трансформации в социально-экономической сфере. Происходит интенсивная цифровизация производственных процессов, внедрение систем автоматизированного принятия решений на основе аналитики больших данных, а также интеграция интеллектуальных систем во все ключевые сферы человеческой деятельности. Эти изменения существенно влияют на структуру занятости, модернизацию медицинского обслуживания, трансформацию государственного администрирования, реорганизацию промышленного сектора и модификацию образовательной системы.

3.2 Искусственный интеллект в игровых приложениях

Искусственный интеллект в видеоиграх представляет собой сложную систему алгоритмических решений, управляющих поведенческими паттернами компьютерных персонажей (NPC). Хотя игровой ИИ не обладает истинным сознанием или креативностью, его реакции и действия, тщательно запрограммированные создателями, способны адаптироваться к различным игровым ситуациям. Основная цель такого ИИ заключается не в превосходстве над игроком, а в создании увлекательного игрового опыта через имитацию осмысленного поведения виртуальных персонажей.

В контексте разработки игр, искусственный интеллект фокусируется на механизмах принятия решений виртуальными сущностями в соответствии с текущей игровой обстановкой. Специалисты в области ИИ определяют данный процесс как координацию "разумных агентов". Такими агентами могут выступать не только отдельные персонажи, но и технические объекты, коллективные единицы или даже абстрактные понятия вроде государств и цивилизаций. Каждый агент функционирует по принципу трехступенчатого цикла:

– *Сбор информации:* агент анализирует окружающую среду, получая данные о потенциальных угрозах, доступных ресурсах и стратегически важных локациях.

– *Аналитический процесс*: на основе полученных данных агент оценивает риски и возможности, формируя оптимальную стратегию действий.

– *Реализация решения:* агент воплощает выбранную стратегию в конкретных действиях, будь то перемещение, взаимодействие с объектами или вступление в конфликт.

3.2.1 Агент для поиска пути

Агент поиска пути (pathfinding agent) – это ключевой компонент искусственного интеллекта в играх, который позволяет неигровым персонажам (NPC) находить оптимальный маршрут от точки A до точки B в игровом мире. Без такого агента персонажи либо двигались бы по прямой, застревая в препятствиях, либо требовали бы ручного программирования каждого возможного маршрута, что крайне неэффективно. Самым популярным алгоритмом для реализации поиска пути является A*, хотя существуют и другие варианты вроде алгоритма Дейкстры или JPS.

При разработке игр одной из ключевых проблем является организация эффективной навигации неигровых персонажей в реальном времени. Традиционные алгоритмы поиска пути, анализирующие каждую точку

игрового пространства, могут создавать существенную нагрузку на процессор, особенно когда речь идет о множестве одновременно перемещающихся объектов. Одним из эффективных решений данной проблемы является использование системы waypoints (путевых точек), где игровое пространство заранее размечается узловыми точками, по которым персонажи могут строить свои маршруты. Вместо анализа каждого пикселя или юнита пространства, система использует заранее определённую сеть связанных между собой точек, что существенно снижает вычислительную нагрузку. Такой подход имеет ряд существенных преимуществ: значительно более быстрые расчёты маршрута из-за ограниченного количества возможных путей, возможность заранее прописать специальные действия в конкретных точках (прыжки, использование укрытий, смена анимации), простота редактирования и отладки путей, а также возможность добавлять дополнительную информацию к точкам, такую как уровень опасности или тип местности.

Однако у этого метода есть и свои ограничения Движение может выглядеть менее естественным, так как персонажи следуют по предопределённым путям, а также требуется тщательная ручная расстановка точек на уровне, что увеличивает время разработки. В современных играх часто используют гибридный подход, где основная навигация осуществляется по Waxpoints. а локальные перемещения И уклонение от препятствий обрабатываются динамически. Разработчики обычно располагают maxpoints в ключевых точках уровня: на перекрестках и развилках, у входов и выходов из помещений, в точках укрытий, местах, требующих специальных действий, и стратегически важных позициях.

Для создания агента поиска пути необходимо использовать алгоритмы реализации поиска пути. Существует множество алгоритмов, каждый из которых более применим под определенные условия. Также возможна их модификация под определенные задачи с целью увеличения их эффективности.

В области алгоритмов навигации особое место занимает метод A* (A-стар), зарекомендовавший себя как исключительно результативный инструмент маршрутизации. В основе его функционирования лежит эвристический подход, позволяющий рассчитывать оптимальную траекторию движения путем комбинирования двух ключевых метрик: фактически пройденного расстояния (g) и прогнозируемого пути до конечной точки (h). При условии корректности эвристической оценки, не превышающей реальную дистанцию, алгоритм неизменно находит оптимальный маршрут.

Метод A* выделяется своей способностью эффективно определять оптимальные траектории, что обусловило его широкое применение в игровой индустрии, особенно при проектировании обширных виртуальных

пространств. Однако при масштабных вычислениях или сложной топологии местности алгоритм может требовать существенных вычислительных ресурсов.

Механизм работы A базируется на анализе двух параметров:* g(n) – реальной стоимости пройденного пути и h(n) – прогнозируемой стоимости оставшегося маршрута. Суммарный показатель f(n) вычисляется как их сумма. Процесс поиска начинается с инициализации стартовой точки в открытом списке, после чего происходит последовательный анализ узлов с минимальным значением f(n). При достижении целевого узла маршрут считается найденным. Для каждой промежуточной точки производится оценка соседних узлов, обновление их параметров и, при необходимости, включение в список для Дальнейшего рассмотрения.

Алгоритм Дейкстры, в отличие от A*, специализируется на поиске кратчайших путей от исходной точки до всех остальных узлов графа без применения эвристических методов. Его преимущество заключается в гарантированном нахождении оптимальных маршрутов в графах с неотрицательными весами, что делает его особенно эффективным при отсутствии конкретной целевой точки. Однако при необходимости поиска пути до единственной цели он уступает A* в эффективности.

Волновой алгоритм (алгоритм Ли) – один из алгоритмов поиска пути, который использует принцип распространения волны для нахождения кратчайшего маршрута. Алгоритм последовательно маркирует доступные клетки числами, отражающими их удаленность от начальной точки, пока не достигнет цели или не исчерпает все доступное пространство. Восстановление пути происходит в обратном направлении путем выбора клеток с последовательно уменьшающимися значениями [2].

Этот метод особенно эффективен при решении задач маршрутизации в лабиринтах и на печатных платах, обеспечивая гарантированное нахождение кратчайшего пути при наличии препятствий любой конфигурации. Несмотря на высокие требования к памяти и вычислительным ресурсам при работе с крупными полями, волновой алгоритм сохраняет популярность Б специализированных приложениях с ограниченным игровым пространством.

3.2.2 Сравнение алгоритмов поиска пути

Выбор оптимального алгоритма поиска пути зависит от специфики конкретной задачи и требует баланса между производительностью, точностью и ресурсоемкостью. Метод A* оптимален для динамических открытых миров, алгоритм Дейкстры эффективен в стратегических играх, а волновой алгоритм

наиболее применим в головоломках и задачах с простой топологией. В таблице 3.1 представлено сравнение алгоритмов поиска пути.

Таблица 3.1 Сравнение алгоритмов поиска пути

Алгоритм	**Время выполнения**	**Память**	**Оптимальность пути**	**Подходит для**
A*	0(d) (в худшем случае) или быстрее, если хорошая Эвристика	Требует много памяти (хранение открытых и закрытых узлов	Оптимальный путь, если эвристика корректна	Открытые миры, динамичные карты, игры с большим числом объектов
Дейкстра	0((E+V) log V) при использовании очереди с приоритетом	Среднее использование памяти, зависит от структуры графа	Оптимальный путь	Стратегии, навигация по сетям, задачи с множеством целей
Волновой алгоритм	O(n*m) пропорционально количеству клеток	Экономит память, зависит от размера карты	Оптимальный путь	Лабиринты, головоломки, платформеры, статичные карты

Таким образом, для открытых миров и динамичных игр с высокими требованиями к производительности и оптимальности, лучше всего использовать A*, для анализов больших графов или сетей – Дейкстру, а для простых карт с одинаковыми затратами – Волновой алгоритм. Выбор конкретного алгоритма всегда зависит от специфики задачи и особенностей игрового процесса, и знание их сильных и слабых сторон поможет создать оптимальный опыт для игроков [2].

Следует подчеркнуть, что определение наиболее подходящего алгоритмического решения неразрывно связано со спецификой конкретного проекта, особенностями геймплея и техническими ограничениями платформы. Глубокое понимание преимуществ и ограничений каждого метода позволяет разработчикам создавать максимально эффективные и отзывчивые системы навигации, что непосредственно влияет на качество игрового опыта конечных пользователей.

3.2.3 Агент для принятия решений

Агент принятия решений (decision-making agent) – это компонент игрового ИИ, отвечающий за выбор действий и поведения неигровых персонажей в зависимости от текущей ситуации, состояния игрового мира и собственных целей. Этот агент определяет, когда NPC должен атаковать, отступать, искать укрытие, взаимодействовать с объектами или другими персонажами. В отличие от простых конечных автоматов, современные

системы принятия решений способны оценивать множество факторов одновременно и выбирать оптимальные действия на основе сложных критериев, что делает поведение NPC более реалистичным и адаптивным.

Основные задачи агента принятия решений включают в себя оценку текущей ситуации (например, уровень здоровья, наличие противников, доступные ресурсы), определение приоритетных целей, выбор подходящей тактики и координацию действий с другими NPC. Такие системы особенно важны в играх с открытым миром, где персонажи должны автономно существовать и реагировать на динамически меняющееся окружение, а также в стратегиях и симуляторах, где требуется имитация разумного поведения множества независимых агентов.

Utility AI, также известный как ИИ на основе потребностей, представляет собой поведенческую технологию искусственного интеллекта, которая определяет набор потребностей и которым присвоена кривая приоритетности и значение оценки, которая может увеличиваться или уменьшаться с течением времени. Этот паттерн особенно эффективен в ситуациях, где требуется выбирать между несколькими конкурирующими целями, учитывая множество факторов одновременно. Utility AI позволяет создавать уникальное и более естественное поведение для человека, избегая резких переключений между состояниями, характерных для конечных автоматов.

GOAP, или Goal-Oriented Action Planning, является методикой проектирования ИИ, при которой цепочка поведений агентов (NPC, абстрактных сущностей, "живых" препятствий) выбирается в реальном времени автоматически для удовлетворения цели. Таким образом, добавляется вариативность действий и некоторая непредсказуемость. Это делает поведение NPC живым и позволяет им находить нестандартные решения проблем [17].

3.2.4 Машинное обучение

Машинное обучение представляет собой совокупность технологий и методов искусственного интеллекта, особенностью которых является обучение машины в процессе поиска решений для более точного и оперативного выполнения задач в будущем. Машинное обучение в геймплее оптимизирует игру на основе её восприятия конкретным геймером и помогает решить уйму задач, включая настройку игрового баланса и адаптивное управление сложностью в зависимости от уровня игрока.

Одно из наиболее перспективных применений машинного обучения в играх – это обучение с подкреплением (reinforcement learning), где ИИ учится принимать решения через процесс проб и ошибок, получая награды за успешные действия. Например, в стратегиях ИИ может учиться оптимизировать свою тактику на основе предыдущих игр, а в файтингах – подстраиваться под стиль противника. Нейронные сети используются для создания более естественных анимаций персонажей, генерации диалогов и даже для процедурной генерации уровней [4].

Однако применение машинного обучения в играх сопряжено с определенными вызовами. Основные проблемы включают необходимость большого количества данных для обучения, сложность контроля над обученными системами (они могут находить неожиданные и нежелательные решения), высокие требования к вычислительным ресурсам во время обучения и потенциальные проблемы с воспроизводимостью результатов. Поэтому часто используется гибридный подход, где машинное обучение дополняет традиционные алгоритмы ИИ, а не полностью заменяет их.

3.3 Архитектурные решения

3.3.1 Конечная машина состояний

Машина состояний – это математическая модель, которая описывает поведение системы через конечное число состояний и переходов между ними. В контексте разработки игр, машины состояний используются для управления логикой персонажей, игровых объектов и даже уровней. Они помогают структурировать код и сделать его более понятным и поддерживаемым. Машины состояний позволяют разработчикам легко добавлять новые состояния и переходы, что делает их незаменимым инструментом в разработке сложных игровых систем.

Машина состояний состоит из:

– Состояний: различные состояния, в которых может находиться объект (например, персонаж может быть в состоянии "бег", "прыжок", "атака").

– Переходов: условия, при которых объект переходит из одного состояния в другое (например, если персонаж находится в состоянии "бег" и нажата кнопка прыжка, он переходит в состояние "прыжок").

– Действий: логика, выполняемая при входе в состояние, выходе из состояния и во время нахождения в состоянии.

Конечная машина состояний представлена на рисунке 3.1.

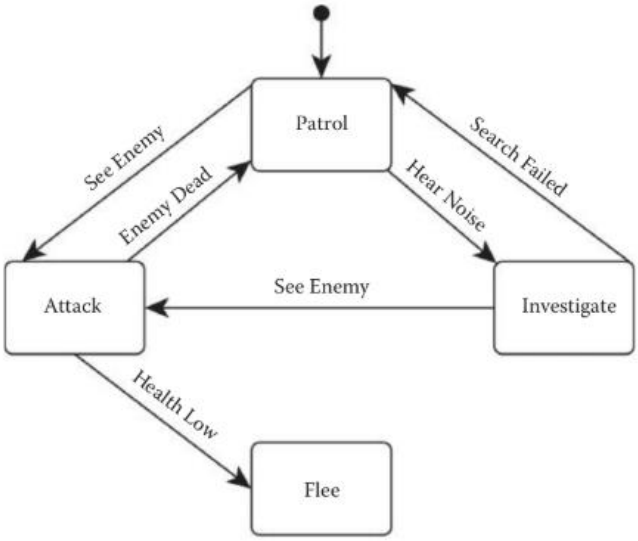

Рисунок 3.1 – Конечная машина состояний [22]

3.3.2 Иерархичная машина состояний

Иерархическая машина состояний (Hierarchical State Machine, HSM) представляет собой расширенную версию конечной машины состояний (FSM), где состояния могут быть вложенными и образовывать иерархическую структуру. В отличие от классической FSM, где все состояния находятся на одном уровне, HSM позволяет организовывать состояния в древовидную структуру, где родительские состояния могут содержать несколько дочерних состояний. Это позволяет более эффективно моделировать сложное поведение, избегая дублирования кода и уменьшая общую сложность системы. Например, состояние «Бой» может содержать подсостояния «Атака», «Защита» и «Отступление», каждое из которых может иметь свои собственные подсостояния.

Основные преимущества HSM включают улучшенную организацию кода, более простое управление сложными поведенческими паттернами и возможность повторного использования логики. При переходе между состояниями HSM автоматически обрабатывает вход и выход из вложенных состояний, что делает систему более надежной и предсказуемой. HSM широко используется в игровых проектах для реализации ИИ противников, где требуется моделировать сложное поведение с множеством взаимосвязанных действий. Например, в стратегических играх HSM может управлять поведением юнитов, где верхний уровень определяет общую стратегию (разведка, атака, оборона), а нижние уровни отвечают за конкретные тактические действия. В ролевых играх HSM эффективно применяется для создания сложных диалоговых систем и поведения NPC, где состояния могут

отражать как общее эмоциональное состояние персонажа, так и конкретные действия в рамках определенных сценариев. Иерархичная машина состояний представлена на рисунке 3.2.

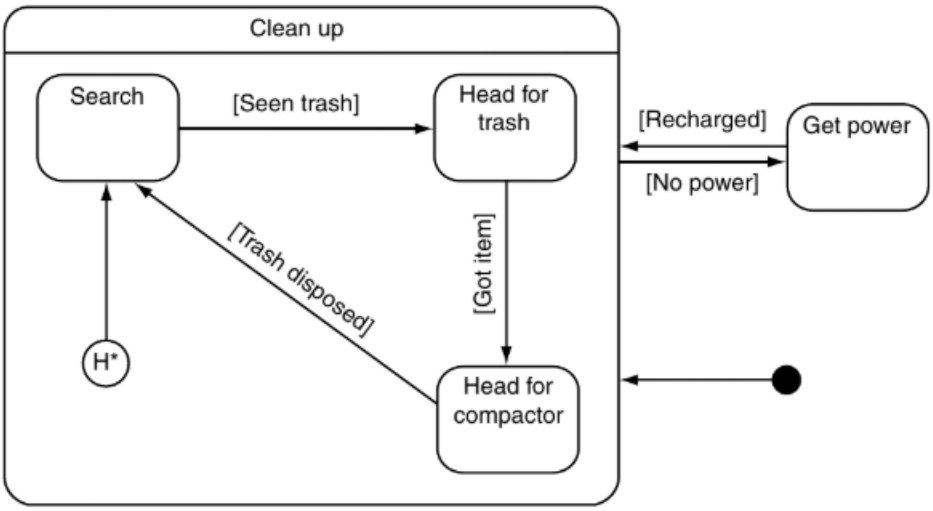

Рисунок 3.2 – Иерархичная машина состояний [21]

3.4.3 Дерево решений

Архитектура поведенческого дерева (Behavior Tree. BT) представляет собой направленную ациклическую графовую структуру, где каждый узел отображает определенный поведенческий паттерн искусственного агента. Горизонтальное измерение древовидной структуры отражает разнообразие доступных поведенческих актов, в то время как вертикальное развитие характеризует уровень комплексности каждой поведенческой цепочки.

В контексте BT каждый поведенческий элемент может находиться в одном из четырех функциональных состояний:

– Позитивное завершение характеризует успешную реализацию поведенческого паттерна.

– Негативное завершение сигнализирует о невозможности выполнения или несоответствии необходимым условиям.

– Активное состояние элемента. Указывает на текущее выполнение поведенческого.

– Критическое состояние возникает при непредвиденных программных сбоях.

Информация о статусе выполнения каскадно, передается от дочерних узлов к вышестоящему родительскому элементу. Навигация по структуре начинается с корневого узла, применяя метод глубинного поиска с приоритетом левосторонних ветвей. При наличии множественных подзадач в

рамках одного узла их исполнение следует принципу левостороннего приоритета.

Структура включает специализированные типы узлов: исполнительные узлы (action), последовательные компоненты (sequence), параллельные элементы (parallel), селективные узлы (selector), условные операторы (condition) и инверторы (inverter). Дерево решений представлено на рисунке 3.3.

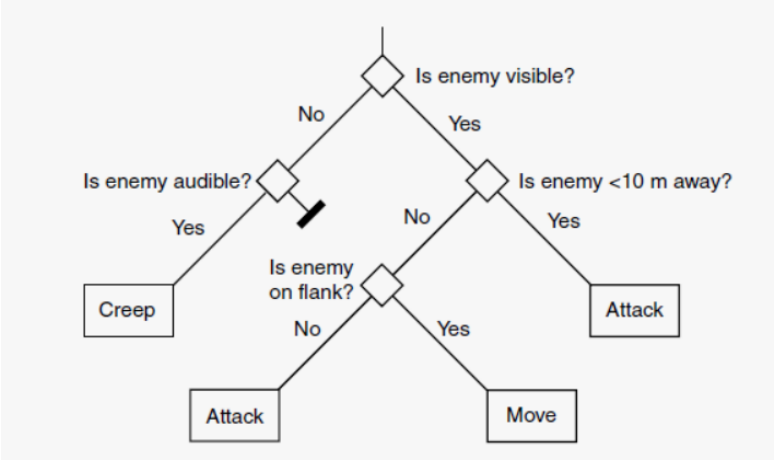

Рисунок 3.3 – Дерево решений [3]

Таким образом, современные технологии игрового ИИ предоставляют широкий спектр инструментов для создания интеллектуальных и реалистичных NPC. Оптимальный выбор методов зависит от типа игры, технических ограничений и желаемого уровня сложности поведения персонажей. В дальнейшем развитие ИИ в играх будет связано с усовершенствованием адаптивных систем, интеграцией машинного обучения и оптимизацией вычислительных алгоритмов.

ГЛАВА 4 ПРАКТИЧЕСКАЯ ЧАСТЬ

4.1 Постановка задачи

Для компьютерного соперника в стратегической пошаговой игре «Crystal Epochy» необходимо создать интеллектуального агента. Цель игры – набрать определенное количество победных очков, либо уничтожив все города соперника. Для создания построек и юнитов нужны ресурсы: энергия, кварц и жители. Энергия и кварц добываются специальными постройками, а жителей можно получить, строя города. Кварц нужен для строительства города и построек по сбору кварца и энергии, энергия нужна для строительства города и создания юнитов. Жители нужны для создания юнитов и строительства построек по сбору кварца и энергии. Ресурсы расходуются при создании постройки или юнита, при их уничтожении расходуемые ресурсы возвращаются. При уничтожении постройки, приносящий ресурс, неиспользуемый доступный ресурс сокращается.

Основными объектами, с которыми будет взаимодействовать игрок являются строения и юниты. Юниты – движимые игровые объекты, над которыми игрок может совершать действия. Они могут военными, с помощью которых можно уничтожить вражеские строения и юниты, а могут быть мирными, которые строят здания. На карте присутствуют уникальные клетки, владея которыми (то есть на клетке должен быть юнит), дает победные очки. Также можно строить уникальные здания, которые тоже дадут победные очки. Чтобы построить любое здание, мирный юнит должен встать на свободную клетку, и иметь достаточное количество ресурсов для строительства. Постройки можно улучшать, чтобы они давали большее количество ресурсов. Чтобы разрушить постройку, вражеский военный юнит должен атаковать постройку, снизив количество жизней у нее до нуля, после чего он становится на клетку постройки, разрушая ее, получая ресурсы от разрушенной постройки. Тоже самое с мирным юнитом, уничтожив его, соперник получит ресурс житель. Уничтожив военный юнит, соперник ничего не получает.

4.2 Разработка архитектуры искусственного интеллекта

4.2.1 Выбор архитектурного решения для поставленной задачи

Для решения поставленной задачи была выбрана иерархичная машина состояний, так как имеет возможность динамически добавлять новые состояния, упрощая дальнейшее расширение возможностей интеллектуального агента «Crystal Epochy». Также иерархичная машина

состояний позволяет избегать лишнего дубляжа кода и уменьшить сложность общей системы, разложив действия интеллектуального агента на слои.

Так же, исходя из жанра игры, интеллектуальный агент «Crystal Epochy» должен иметь возможность анализировать «полезность» действий, учитывая множество факторов, таких как возможная угроза со стороны игрока, наличие ресурсов для создания строений и юнитов, получение победных очков, отсутствие защиты строений у игрока и др. Таким образом, чтобы учитывать «полезность» действий интеллектуального агента, необходимо использовать подход Utility AI. В дальнейшем оценку приоритетности действий можно менять, создавая уникальное поведение для интеллектуального агента. К примеру, уменьшив приоритетность создания военных юнитов, компьютерного соперника можно сделать миролюбивым и наоборот, увеличив приоритетность создания армии, можно сделать более воинственным.

4.2.2 Построение модели

Для создания модели строения иерархической машины состояния, необходимо определить основные состояния интеллектуального агента, а также разбить их на слои. Начальным состоянием будет «Initialize», которое отвечает за анализ стартовой позиции и поиск ближайших ресурсов. От начального состояние будет осуществляться переход к состоянию «Production». Данное состояние отвечает за строительство и улучшение построек, расширение добычи ресурсов и создание юнитов. Данное состояние является составным, и состоит из состояний: «Build», «Hire». Состояние «Build» отвечает за строительство и улучшение построек, двигая и используя мирных юнитов для строительства, увеличиваем запас ресурсов. Если доступных мирных юнитов нет или есть угроза со стороны игрока, то происходит переход в состояние «Hire». Данное состояние нанимает военные и мирные юниты. Далее возможно два перехода, и, исходя из посчитанного приоритета, компьютерный игрок «Crystal Epochy» выберет переход с самым большим приоритетом. Если возникает угроза со стороны игрока, то он переходит в состояние «War». Если рядом с компьютерным игроком есть уникальные места, то он переходит в состояние «Victory Points».

Состояние «War» является составным и содержит два состояния: «Defend» и «Attack». Состояние «Defend» отвечает за защиту строений, мирных юнитов и ключевых мест, тогда как состояние «Attack» отвечает за нападения на вражеские строения, юнитов и ключевые места, контролируемые игроком. Если есть возможность получить победные очки, то происходит

переход к состоянию «Victory Points». Если же есть нехватка юнитов, то происходит переход к состоянию «Production».

Состояние «Victory Points» отвечает за получение победных очков путем строительства уникальных строений и контролирование уникальных мест. В случае возникновения угрозы со стороны игрока «Crystal Epochy», происходит переход в состояние «War», или если есть необходимость в ресурсах или юнитах, то в «Production». Архитектура интеллектуального агента представлена на рисунке 4.1.

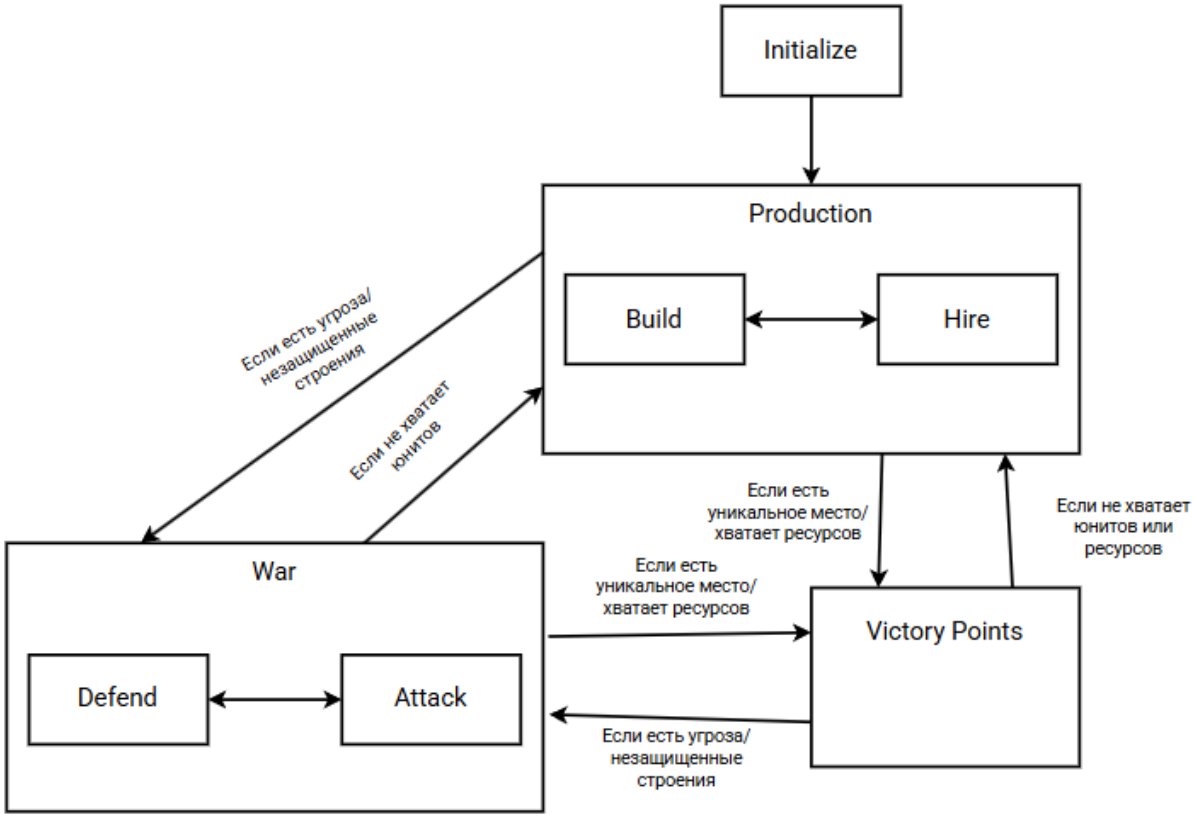

Рисунок 4.1 – Архитектура интеллектуального агента «Crystal Epochy»

4.3 Разработка основных классов

Для реализации машины состояний, необходимо разработать следующие классы: BaseStateMachine, BaseState, State, Transition и Action. Общая структура машины состояний представлена на рисунке 4.2.

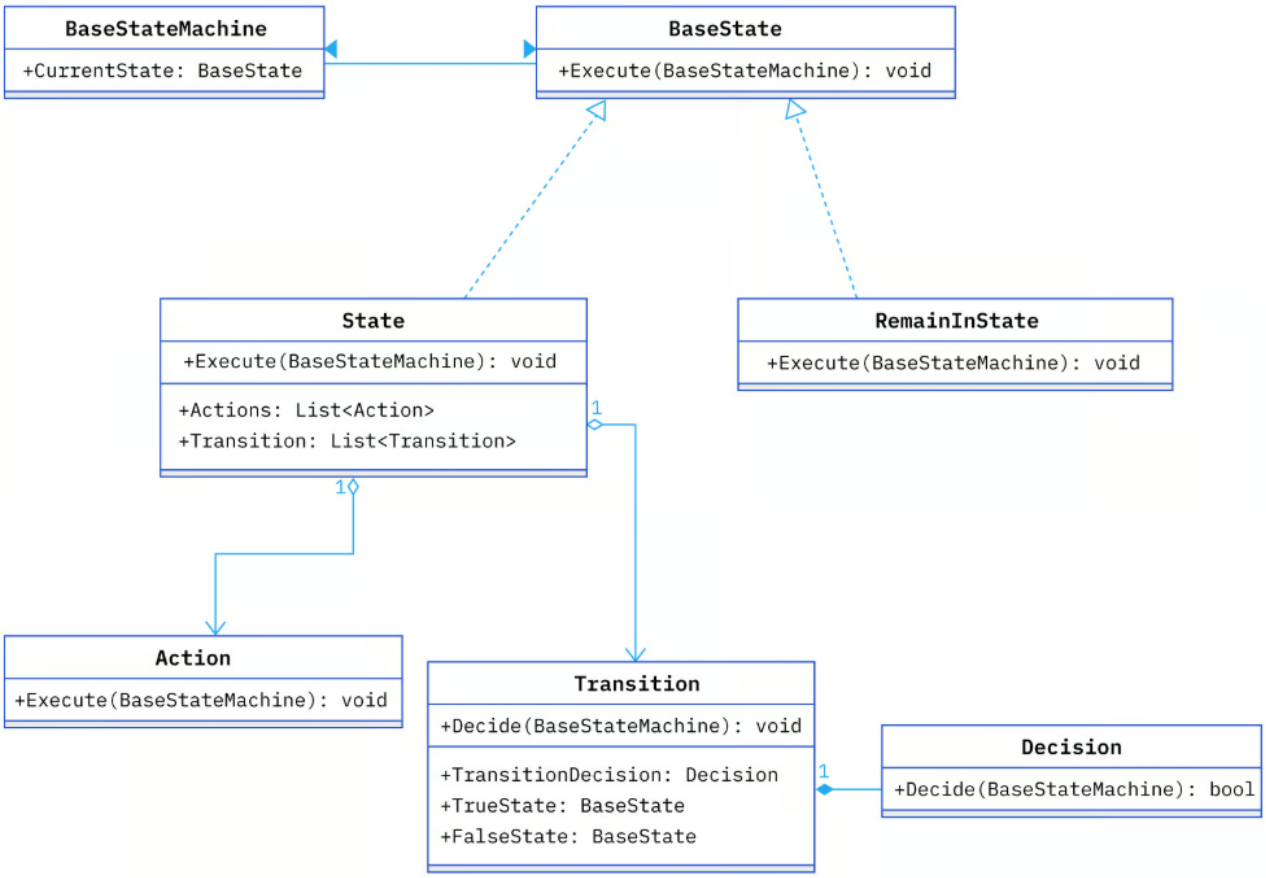

Рисунок 4.2 – Общая структура машины состояний

BaseStateMachine взаимодействует с текущим состоянием для выполнения действий и проверки переходов.

```
public class AIStateMachine : MonoBehaviour
{
    [SerializeField] private AIStateType currentState;
    private Dictionary<AIStateType, AIState> states;
    private List<Transition> transitions;
    private Player aiPlayer;

    private void Awake()
    {
        InitializeStates();
        InitializeTransitions();
        aiPlayer = GetComponent<Player>();
    }

    private void InitializeStates()
    {
        states = new Dictionary<AIStateType, AIState>
        {
            { AIStateType.Initialize, new InitializeState() },
            { AIStateType.EarlyExpansion, new EarlyExpansionState() },
            { AIStateType.ResourceFocus, new ResourceFocusState() },
            { AIStateType.MilitaryProduction, new MilitaryProductionState() },
            { AIStateType.Defend, new DefendState() },
            { AIStateType.Attack, new AttackState() },
            { AIStateType.VictoryPoints, new VictoryPointsState() },
        };
    }
```

```
private void InitializeTransitions()
{
    transitions = new List<Transition>
    {
        // Переходы из Initialize
        new Transition(
            AIStateType.Initialize,
            AIStateType.EarlyExpansion,
            (player) => player.HasInitialResourcesForExpansion(),
            1
        ),

        // Переходы из EarlyExpansion
        new Transition(
            AIStateType.EarlyExpansion,
            AIStateType.Defend,
            (player) => player.GetThreatLevel() > 0.7f,
            3
        ),
        new Transition(
            AIStateType.EarlyExpansion,
            AIStateType.ResourceFocus,
            (player) => player.HasBasicInfrastructure(),
            1
        ),

        // Переходы из ResourceFocus
        new Transition(
            AIStateType.ResourceFocus,
            AIStateType.MilitaryProduction,
            (player) => player.HasSufficientResourcesForMilitary(),
            2
        ),
        new Transition(
            AIStateType.ResourceFocus,
            AIStateType.Defend,
            (player) => player.GetThreatLevel() > 0.5f,
            3
        ),
    };
}
```

BaseState, производный от ScriptableObject, включает в себя метод Execute, который получает BaseStateMachine в качестве входных данных и передает ему действия и переходы. Класс State хранит ссылки на действия и переходы, поддерживает списки для обоих и переопределяет базовый метод Execute для вызова действий и переходов.

```
public abstract class AIState
{
    protected List<Transition> transitions = new List<Transition>();

    Ссылок: 0
    public virtual void Enter() { }
    Ссылок: 7
    public virtual void Execute(Player player) { }
    Ссылок: 0
    public virtual void Exit() { }
}
// Пример конкретного состояния с использованием переходов
ссылка: 1
public class ResourceFocusState : AIState
{
    [SerializeField] private int minValue;
    ссылка: 1
    public override void Execute(Player player)
    {

        var resourceAnalyzer = player.GetComponent<ResourceAnalyzer>();
        ResourceContainer resources = player.GetResources();

        // Логика улучшения добычи ресурсов
        for (int i = 0; i < 4; i++)
        {
            if (resources.GetResource(i) < minValue)
            {
                // Планирование строительства или улучшения добывающих построек
                PlanResourceBuilding(player, resources.GetResource(i));
            }
        }
    }
```

Класс Action не может независимо обрабатывать функции, поэтому мы определяем его как абстрактный класс. По мере развития разработки может потребоваться одно действие в нескольких состояниях. Также для реализации модели Utility AI, необходим класс Analyzer, который мог бы оценивать приоритетность каждого действия.

```
public abstract class AIAction
{
    Ссылок: 0
    public ActionType Type { get; protected set; }
    Ссылок: 2
    public int Priority { get; protected set; }
    ссылка: 1
    public float Cost { get; protected set; }
    ссылка: 1
    public float Value { get; protected set; }
    protected Player player;
    protected bool isExecuting;

    Ссылок: 0
    public AIAction(Player player, int priority)
    {
        this.player = player;
        this.Priority = priority;
        isExecuting = false;
    }

    Ссылок: 0
    public abstract bool CanExecute();
    Ссылок: 0
    public abstract void Execute();
    Ссылок: 0
    public abstract void Cancel();

    Ссылок: 0
    public virtual float CalculateUtility()
    {
        return Value / Cost * Priority;
    }
}
```

Класс Transition отвечает за определение условий перехода между состояниями в машине состояний. Его основная задача – инкапсулировать логику, определяющую, когда и при каких условиях ИИ должен перейти из одного состояния в другое.

```
public class Transition
{
    ссылка: 1
    public AIStateType FromState { get; private set; }
    ссылка: 1
    public AIStateType ToState { get; private set; }
    Ссылок: 2
    public TransitionCondition Condition { get; private set; }
    ссылка: 1
    public int Priority { get; private set; }

    Ссылок: 5
    public Transition(AIStateType fromState, AIStateType toState, TransitionCondition condition, int priority)
    {
        FromState = fromState;
        ToState = toState;
        Condition = condition;
        Priority = priority;
    }
    Ссылок: 0
    public bool CheckCondition(Player player)
    {
        return Condition(player);
    }
}
```

4.4 Реализация алгоритмов поиска пути

Алгоритмы поиска пути является неотъемлемой частью компьютерного соперника «Crystal Epochy». Он должен самостоятельно искать путь для своих юнитов, исходя из особенностей окружающей местности и других факторов, таких как наличие вражеских юнитов. Так же данный алгоритм может использовать сам игрок, облегчая игровой процесс, убирая необходимость вручную прокладывать маршрут для юнитов.

4.4.1 Задание координат

Особенностью игры является карта, разбитая на правильные шестиугольники. Такая карта называется гексагональным полем. Так как карта является двумерной, то имеет смысл использовать две координаты для задания местоположения. Однако возникают проблемы, связанные со смещением строк для покрытия прямоугольной области шестиугольниками. Гексагональное поле с двумя координатами представлен на рисунке 4.3.

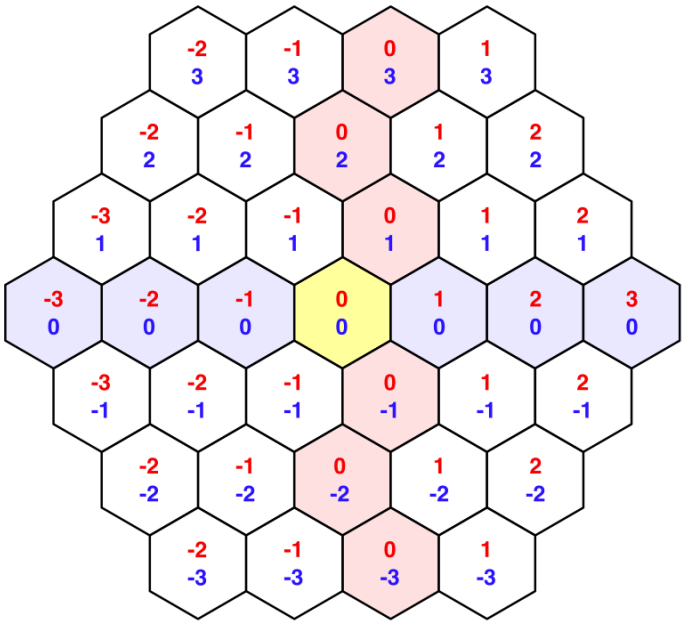

Рисунок 4.3 – Гексогональное поле с двумя координатами

Чтобы упростить нахождение местоположение на карте, выровняем вдоль прямой оси, тем самым, убирая горизонтальное смещение. Пусть данная ось будет осью X. Гексогональное поле с двумя координатами, где ось X выровнена по линии представлен на рисунке 4.4.

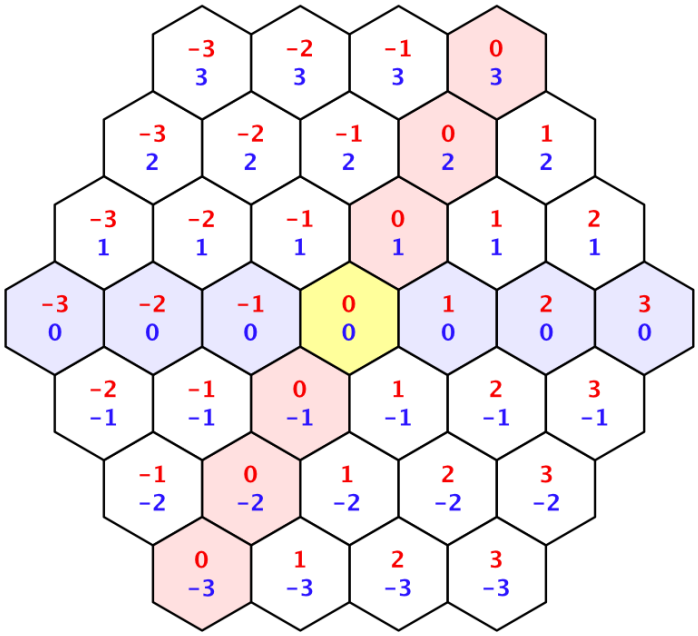

Рисунок 4.4 – Гексогональное поле с двумя координатами, где ось X выровнена по линии

Эта двухмерная система координат позволяет нам последовательно описывать движение и смещения в четырех направлениях. Однако два оставшихся направления все еще требуют специальной обработки. Это указывает на то, что есть третье измерение. Действительно, если измерение X отразить по вертикали, мы получили бы недостающее измерение. Назовем это осью Y. Гексагональное поле с тремя координатами представлен на рисунке 4.5.

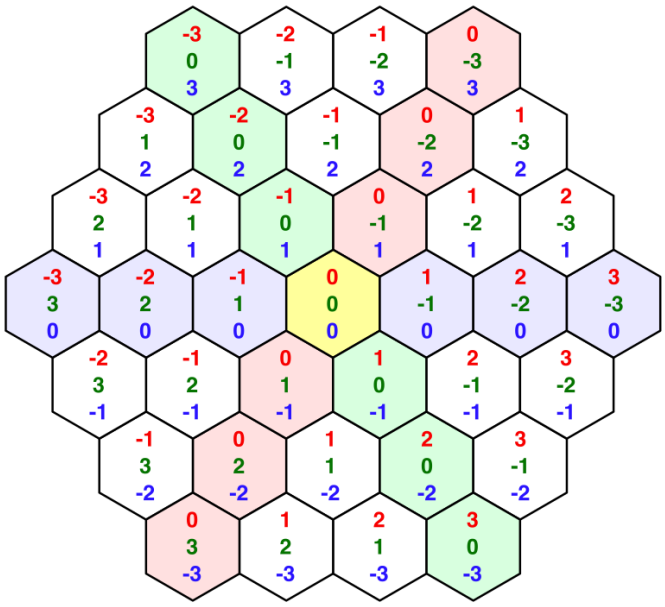

Рисунок 4.5 – Гексагональное поле с тремя координатами

Поскольку эти измерения X и Y отражают друг друга, сложение их координат всегда будет давать один и тот же результат, если сохранить Z постоянным. Фактически, если вы сложите все три координаты вместе, всегда получится ноль. Если увеличить одну координату, придется уменьшить другую. Действительно, это дает шесть возможных направлений движения. Эти координаты обычно известны как кубические координаты, поскольку они трехмерны и топология напоминает куб.

Для нахождения расстояния применим формулу (4.1):

$$d = \frac{|x_2 - x_1| + |y_2 - y_1| + |z_2 - z_1|}{2}. \quad (4.1)$$

Где x_1, y_1, z_1 – координаты первой клетки, x_2, y_2, z_2 – координаты другой клетки. Таким образом можно находить расстояние между любыми клетками. Знание значения расстояния между начальной и конечной точкой понадобится для реализации алгоритмов поиска путей. Также расчёт расстояния между клетками необходим при расчете компьютерным игроком дальнейших ходов, в частности нахождение расстояний до объектов.

4.4.2 Реализация алгоритма A*

Для поиска пути между двумя клетками будет реализован алгоритм A*, так как необходимо построение маршрута из одной точки в другую. Также, игровая карта имеет неровности в рельефе, такие как холмы, скалы и водные клетки, которые необходимо учитывать при построении маршрута.

Для начала необходимо определить стоимость передвижения между клетками. Пусть перемещение из одной клетки в другую, находящихся на одной высоте будет стоить 5 очков передвижения, а по склонам 10. Также, юниты не могут перейти из одной клетки в другую через скалу и не могут двигаться по водным клеткам.

Определив начальную и конечную клетку, алгоритм начнет вести поиск маршрута. Начиная со стартовой клетки, программа проверяет все достижимые соседние клетки, записывая расстояние до исходной клетки. Проверенную клетку делаем помеченной, и переходим к следующей. Далее нужно отслеживать, какие ячейки нужно посетить и в каком порядке. Эту коллекцию часто называют границей или открытым набором. Если бы стоимость перемещения между клетками было одинаковое, то сортировать границу не нужно. Однако для поставленной задачи и для нахождения

необходимо сортировать открытый набор после добавления нового элемента. В программировании используют контейнер приоритетная очередь. Сортировка происходит по расстоянию. Также необходимо проверять проверенные клетки, не является ли расстояние до нее меньше, чем при последнем проходе. Если это так, то менять значение на меньшее.

После нахождения пути до конечной клетки, необходимо восстановить пройденный маршрут. Для этого будет происходить отслеживание для каждой клетки, кроме начальной, из какой клетки прошёл маршрут в нее. Добавляя ячейку к границе, будет происходить процесс отслеживания, потому что она является соседом текущей ячейки. Если происходит отслеживание, из какой ячейки была достигнута каждая ячейка, мы получаем сеть ячеек. А именно, древовидную сеть с начальной ячейкой в качестве корня. Это можно использовать для построения пути, как только будет достигнут пункт назначения. Древовидная сеть, описывающая пути к центру представлена на рисунке 4.6.

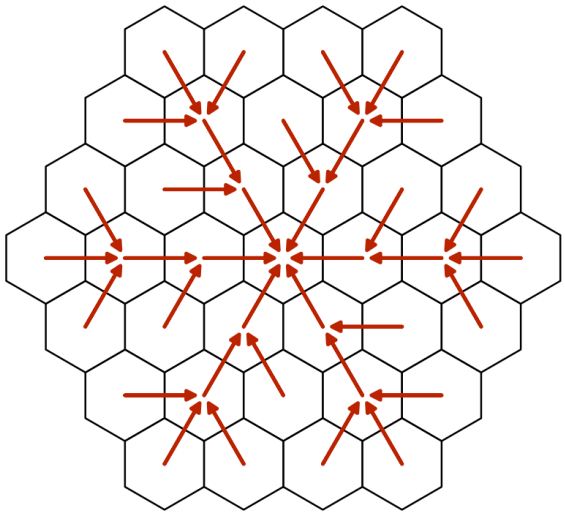

Рисунок 4.6 – Древовидная сеть, описывающая пути к центру

Полученная программа находит кратчайший путь, однако затрачивает много сил на проверку клеток, через которые очевидно, не будет следовать путь. Для оптимизации программы необходимо учитывать помимо расстояния ячейки от начала расстояние до пункта назначения. Подход, при котором используется оценка или предположение вместо точных данных называется эвристикой. Она представляет лучшее предположение об оставшемся расстоянии. Поэтому для использования эвристики необходимо при добавлении в открытый набор новой ячейки, необходимо рассчитывать расстояние до конечной клетки.

После добавления эвристики необходимо изменить расчет приоритета выбора клеток. Теперь он будет рассчитываться по следующей формуле (4.2):

$$f(n) = g(n) + h(n), \tag{4.2}$$

где:

$g(n)$ – фактическая стоимость пути от начальной точки до текущей клетки;

$h(n)$ – эвристическая оценка оставшегося пути от текущей клетки до целевой;

n – номер клетки [2].

Таким образом, реализовав алгоритм A*, удалось оптимизировать поиск кратчайшего пути, позволяя ускорить процесс хода компьютерного игрока, а также облегчить перемещение юнитов игроку, убирая необходимость вручную выстраивать маршрут. Построение оптимального маршрута при ходе игрока «Crystal Epochy» представлен на рисунке 4.10.

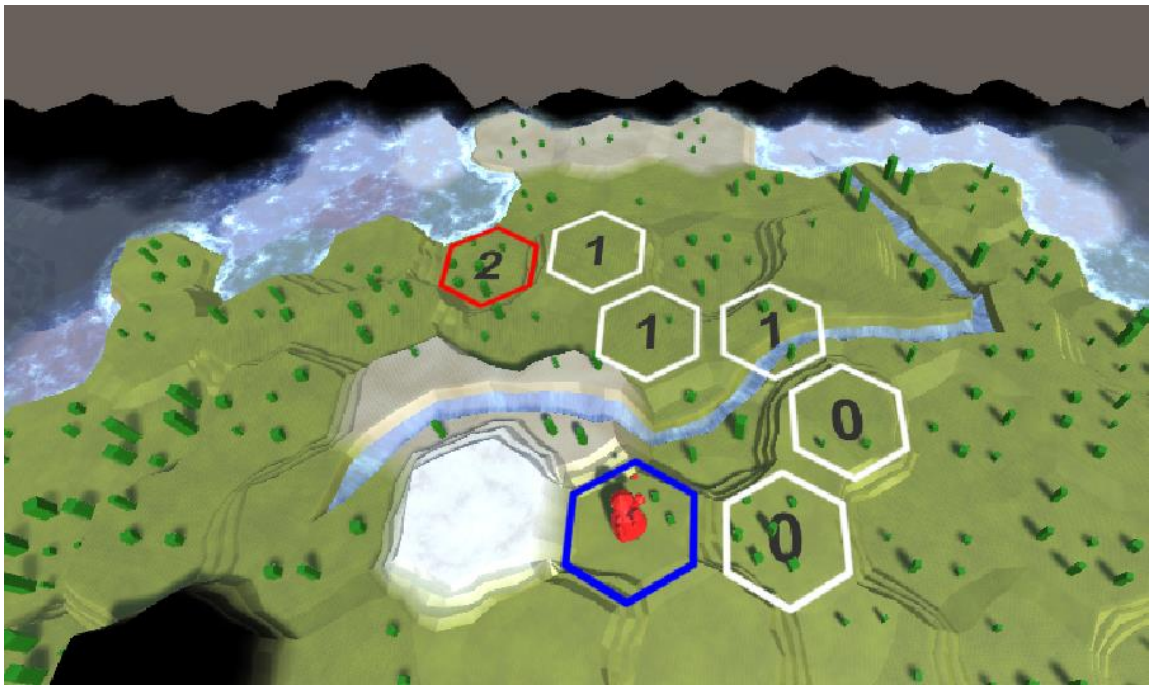

Рисунок 4.10 – Построение оптимального маршрута при ходе игрока в «Crystal Epochy»

Далее представлен метод Search, в котором реализуется алгоритм A* по нахождению оптимального маршрута.

```
bool Search (HexCell fromCell, HexCell toCell, HexUnit unit) {
    int speed = unit.Speed;
    searchFrontierPhase += 2;
    if (searchFrontier == null) {
        searchFrontier = new HexCellPriorityQueue();
    }
    else {
        searchFrontier.Clear();
    }

    fromCell.SearchPhase = searchFrontierPhase;
    fromCell.Distance = 0;
    searchFrontier.Enqueue(fromCell);
    while (searchFrontier.Count > 0) {
        HexCell current = searchFrontier.Dequeue();
        current.SearchPhase += 1;

        if (current == toCell) {
            return true;
        }
```

```
        int currentTurn = (current.Distance - 1) / speed;

        for (HexDirection d = HexDirection.NE; d <= HexDirection.NW; d++) {
            HexCell neighbor = current.GetNeighbor(d);
            if (
                neighbor == null ||
                neighbor.SearchPhase > searchFrontierPhase
            ) {
                continue;
            }
            if (!unit.IsValidDestination(neighbor)) {
                continue;
            }
            int moveCost = unit.GetMoveCost(current, neighbor, d);
            if (moveCost < 0) {
                continue;
            }

            int distance = current.Distance + moveCost;
            int turn = (distance - 1) / speed;
            if (turn > currentTurn) {
                distance = turn * speed + moveCost;
            }

            if (neighbor.SearchPhase < searchFrontierPhase) {
                neighbor.SearchPhase = searchFrontierPhase;
                neighbor.Distance = distance;
                neighbor.PathFrom = current;
                neighbor.SearchHeuristic =
                    neighbor.coordinates.DistanceTo(toCell.coordinates);
                searchFrontier.Enqueue(neighbor);
```

4.4.3 Исследование влияния эвристики на эффективность алгоритма поиска пути

Изменение эвристики при использовании алгоритма A* может сильно повлиять на время нахождения оптимального пути. Рассмотрим гексагональное поле, где между каждой клеткой стоимость пути составляет пять единиц. Необходимо найти кратчайший путь между двумя клетками, расстояние между которыми составляет 35 единиц. Пусть значение эвристики находится по следующей формуле: $h(n) = k * n$, где n – количество клеток между текущей и целевой клеткой, а k – множитель эвристики.

Рассмотрим три случая. В первом случае оценка эвристики не будет учитываться при подсчете, то есть $k = 0$. Алгоритм A*, в котором оценка эвристики равна нулю, является алгоритмом Дейкстры. Тогда реализованный алгоритм находит кратчайший путь, обойдя при этом 188 клеток.

Во втором случае множитель эвристики будет равен $k = 1$. В этом случае оценка эвристики участвует в расчете приоритета в выборе проверяемой клетки, но имеет не сильное влияние на расчет приоритета. Реализованный алгоритм находит кратчайший путь, обойдя 129 клеток.

В третьем случае множитель сделаем равным значению стоимости пути между клетками, а именно $k = 5$. В этом случае значение оценки эвристики имеет такой же вес, что и фактическая оценка стоимости пути. И при таком множителе алгоритм находит кратчайший путь, обойдя 37 клеток. Результат сравнения представлен в таблице 4.1.

Таблица 4.1 Оценка влияния эвристики скорость работы алгоритма по поиску кратчайшего пути

Значение множителя эвристики, k	Количество пройденных клеток
0	188
1	129
5	37

Таким образом, эвристика позволяет ускорить процесс поиска оптимального пути. На рисунке 4.7 и 4.8 видно, что эвристика помогает сократить количество проверяемых клеток, уменьшая время работы алгоритма.

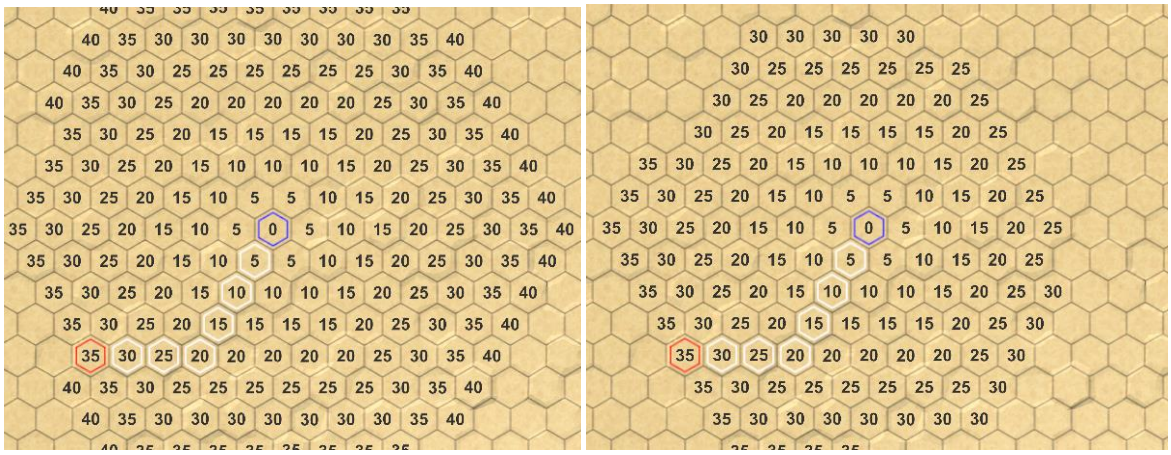

Рисунок 4.7 и 4.8 – Поиск оптимального пути без эвристики и с эвристикой,

Увеличив значение эвристики, можно ещё сократить поиск, который представлен на рисунке 4.9. Однако, если еще сильнее увеличить эвристику, то алгоритм начнет пропускать оптимальный путь, если он лежит в противоположенной стороне от целевой клетки. Поэтому важно находить оптимальное значение оценки эвристики.

Рисунок 4.9 Поиск оптимального пути с увеличенной эвристикой

4.5 Реализация паттерна Utility AI

Компьютерный соперник может совершать ходы, создавая юниты и здания, перемещая армию, оценивая возможные риски, чтобы получать победные очки. Однако игровой компьютер при схожих обстоятельствах будет действовать одинаково, что будет сильно заметно при множественных играх. Игрок «Crystal Epochy» за это время сможет найти оптимальное решение, чтобы обыгрывать его, и, тем самым, убирает его интерес к игре.

Также в играх присутствует сюжет, в котором присутствуют уникальные персонажи и цивилизации, которые будут отличаться не только своим внешним видом, но и характером и поведением. Они могут быть как миролюбивыми, которые будут мирным образом, строя здания и чудеса, получать победные очки, так и более воинственными, которые будут с помощью армии и войн достигать победы в игре «Crystal Epochy». Однако плохой практикой будет писать отдельно игровую логику для каждого типа поведения компьютерного соперника. Для этой цели существует паттерн проектирования Utility AI. Данный паттерн позволяет расставлять приоритеты между возможными вариантами хода компьютера. Ход реализуется участниками игры над юнитами, движимыми объектами игры.

В игре «Crystal Epochy», в которой реализуется компьютерный соперник, поведение может быть либо мирное, то есть компьютер будет стараться строить здания и чудеса для достижения победы, либо воинственное, из-за чего он будет стараться захватить игрока, создавая предпочтительно военные юниты. Для такой реализации в общие расчеты приоритета действий необходимо добавить коэффициент, который будет отвечать за воинственность поведения. Влиять этот коэффициент будет в следующих ситуациях:

– при выборе создания боевого юнита или рабочего, если достаточно ресурсов;

– при выборе нападать ли на игрока или нет, или до какого момента преследовать вражеского юнита;

– при расстановке приоритетности действий для каждого юнита, в частности обороны зданий, разведки местности и нападение на игрока.

Однако важно помнить, что коэффициент не должен повышать приоритетность действий, которые на определенном этапе будут ключевыми и их приоритетность должна быть максимальной. Например, при нехватке ресурсов компьютер в первую очередь должен построить здания, которые будут решать данную проблему, а не военные юниты, ведь если ресурсов больше не останется, он не сможет создать ни новых зданий, ни юнитов. Также, коэффициент не должен преуменьшать приоритет тех действий, которые на определенном этапе будут ключевыми. Например, случай, когда рядом со строениями компьютера будут находиться юниты игрока, которые могут запросто захватить их. Поэтому оптимальными значениями для работы игрового компьютера для коэффициента воинственности будут от 0,5 до 1,5.

4.6 Исследование влияния паттерна Utility AI на поведение компьютерного игрока

Чтобы показать влияние коэффициента воинственности на поведение компьютера, он будет помещен в ситуацию, где при достаточном количестве ресурсов и отсутствии угроз на сделает пять ходов. Начальные условия представлены на рисунке 4.11.

Рисунок 4.11 – Начальные условия для компьютера

Рассмотрим случай, когда коэффициент воинственности равен 0,5. Как видно на рисунке 4.12 компьютерный игрок отдал предпочтение строительство зданий, в частности и чуда, которое дает победные очки. Для этого он создал второго строительного юнита, построил вспомогательные здания для того, чтобы приобрести необходимое количество ресурсов. Компьютерный игрок «Crystal Epochy» не стал строить ни одного военного юнита, так как никакой угрозы со стороны игрока не было, а именно наличия юнитов игрока рядом со зданиями компьютера. Тем самым можно говорить о его миролюбивом поведении.

Рисунок 4.12 – Результат после пяти ходов с коэффициентом 0.5

Рассмотрим случай, когда коэффициент воинственности равен 1,5. На рисунке 4.13 можно увидеть, что компьютерный игрок «Crystal Epochy» создал три военных юнита. При этом, построенных зданий стало значительно меньше, так как он не приобрел второго строительного юнита. Из-за отсутствия военных юнитов на старте, он отдал приоритет их созданию. Первый юнит он поставил на здание, которое дальше всего стоит от остальных, другого отправил разведывать территорию. Однако, несмотря на высокий коэффициент воинственности, он также тратил ресурсы на создания построек. Если коэффициент воинственности сделать выше, то он будет отдавать предпочтение найму военных юнитов и перестанет строить новые здания, до момента, когда ресурсов станет минимальное количество.

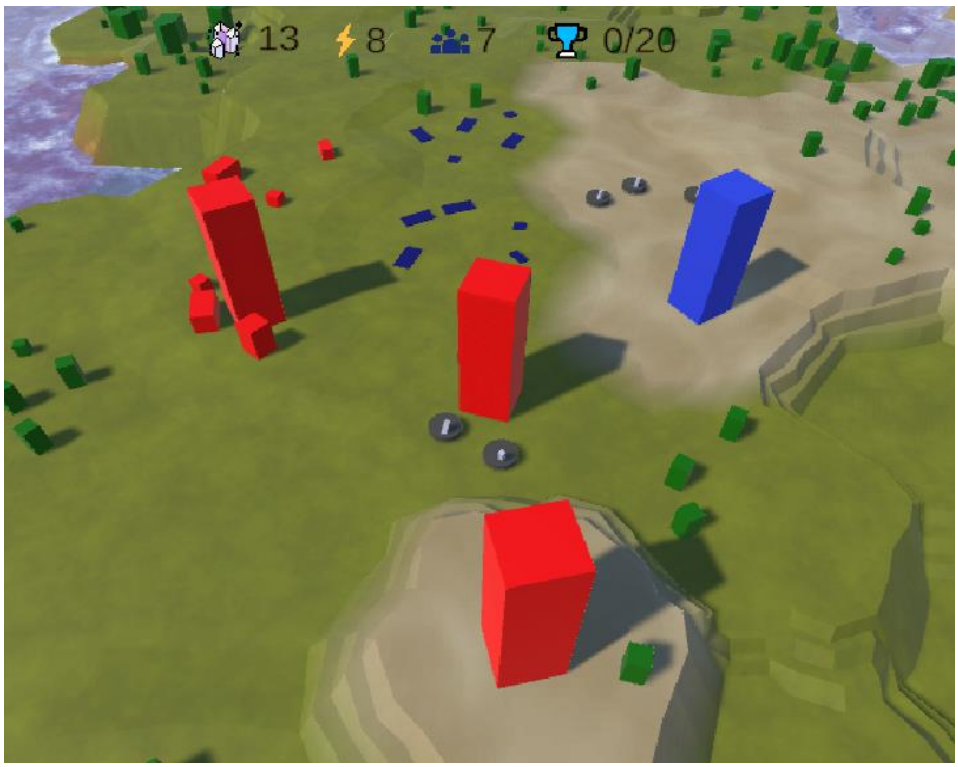

Рисунок 4.13 – Результат после пяти ходов с коэффициентом воинственности 1.5

Чтобы продемонстрировать полный отказ от строительства до момента, когда ресурсов станет минимальное количество, необходимо повысить коэффициент воинственности. Пусть его значение будет равно 2, а также не ограничиваться пятью ходами. Как видно на рисунке 4.14, компьютер полностью перестал строить здания, отдав предпочтение военным юнитам. Однако к шестому ходу, ресурсов стало минимальное количество, что изменило приоритет на строительство зданий и строительный юнит на шестой ход построил здание, дающее ресурс поселенцев. Всего он успел построить 4 военных юнита.

Рисунок 4.14 – Результат после шестого хода с коэффициентом воинственности 2

Проанализировав 3 полученных результата, можно сказать, что изменение коэффициента воинственности кардинально меняет поведение компьютерного игрока. Однако, в ключевые моменты он действует эффективно, предотвращая критические ситуации. Компьютерный соперник действительно приобретает некоторое поведение, которое проявляется в его ходах. Игрок тем самым приобретает разнообразие в игровом опыте, сталкиваясь с разным поведением его компьютерного соперника. Результаты эксперимента приведены в таблице 2.2.

Таблица 4.2 Сравнение результатов при различных значениях коэффициента воинственности

Значение коэффициента воинственности	Количество военных юнитов	Количество мирных юнитов	Количество зданий
0.5	0	2	7
1.5	3	1	6
2	4	1	4

Таким образом, был реализован паттерн Utility AI, проведена модификация путем добавления параметра «воинственность», с помощью которого можно настраивать поведение компьютерного игрока. Также было исследовано его влияние на поведение компьютерного игрока.

ЗАКЛЮЧЕНИЕ

В результате проведенного исследования была успешно разработана модель интеллектуального агента для стратегической игры «Crystal Epochy» на движке Unity, что подтверждает достижение основной цели работы. Проведенное исследование позволило сделать следующие конкретные и обоснованные выводы по каждой главе.

В первой главе была исследована актуальность разработки игровых приложений на сегодняшний день, а также проведен анализ и сравнение современных игровых движков и был выбран самый подходящий для решения поставленных задач при наличии ограниченных ресурсов.

Во второй главе был рассмотрен игровой движок Unity, используемый язык разработки в нем, а также существующие фреймворки и библиотеки для разработки интеллектуального агента в играх.

В третьей главе были рассмотрены особенности реализации агента поиска пути в играх, проведен сравнительный анализ самых популярных алгоритмов поиска кратчайшего маршрута, используемых в игровых приложениях. Были также изучены основные архитектурные решения для разработки интеллектуального агента в играх, а также используемые паттерны для создания реалистичной системы принятия решений компьютерным игроком.

В четвертой главе была разработана архитектура компьютерного соперника для стратегической игры и основные классы, реализующие его. Был внедрен агент поиска пути, а также было исследовано влияние эвристики на скорость нахождения кратчайшего маршрута. Был реализован паттерн Utility AI, который влияет на поведение компьютерного игрока. Данный паттерн был модифицирован путем добавления параметра «воинственность», влияющего на поведение компьютерного игрока, а также было исследовано его влияние на выбор действий компьютерного игрока.

Данная работа представляет перспективы для дальнейших исследований, которые охватывают внедрение машинного обучения для адаптации интеллектуального агента под стиль игры пользователя, оптимизацию алгоритмов, позволяющих эффективно обрабатывать масштабные игровые карты, а также создание системы, способной динамически адаптировать уровень сложности в процессе игры.

Таким образом, проведенное исследование демонстрирует эффективность предложенных решений и создает основу для разработки коммерческих стратегических игр с интеллектуальным компьютерным соперником.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Демин, В. В. Анализ игровых движков / В. В. Демин // Наука. Информатизация. Технологии. Образование : материалы XII международной научно-практической конференции, г. Екатеринбург, 25 февраля – 1 марта 2019 г. – Екатеринбург : Издательство РГППУ, 2019. – С. 292–296.
2. Ивашенцев А. С. Сравнительный анализ основных алгоритмов поиска пути в играх // Актуальные исследования. 2024. № 52–1 (234). С. 73–78
3. Как создать игровой ИИ: гайд для начинающих [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/pixonic/articles/428892/. – Дата доступа: 19.04.2025.
4. Компьютерная помощь: как технология машинного обучения может повлиять на игровую индустрию [Электронный ресурс]. – Режим доступа: https://dtf.ru/gamedev/25618-kompyuternaya-pomoshch-kak-tehnologiya-mashinnogo-obucheniya-mozhet-povliyat-na-igrovuyu-industriyu. – Дата доступа: 06.05.2025.
5. Кремер Н. Ш. Unity3D как средство реализации интерактивных приложений // Практическое руководство / Н.Ш. Кремер. – Москва, 2012. – 75 с.
6. Методические рекомендации по организации дипломного проектирования в Белорусском государственном университете [Электронный ресурс]. – Режим доступа: Metod-rekom-po-org-diplomnogo-proektir-BGU-2024.pdf. – Дата доступа: 19.04.2025.
7. Обзор техник реализации игрового ИИ [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/420219/. – Дата доступа: 19.04.2025.
8. Официальный сайт Unity3D [Электронный ресурс]. – Режим доступа: http://unity3d.com/. – Дата доступа: 12.03.2025.
9. Погорельцев Г.И. Мультиплатформенное игровое приложение «Untrevelled dream». В сборнике: электронные системы и технологии. 2022. С. 817–819.
10. Сагадиев Р. М., Абдыгаликова Г. А., Оспанова Т. Т. Применение компонентного подхода и инструментов разработки Unity. Сборники конференций НИЦ Социосфера. 2024. № 6. С. 7–9.
11. Самые интересные игры с utc и unreal engine за последние 10 лет // Редакция rating-gamedev [Электронный ресурс]. – 2025. – Режим доступа: https://rating-gamedev.ru/blog/samye-interesnye-igry-s-utc-i-unreal-engine-za-poslednie-10-let. – Дата доступа: 10.05.2025.

12. Система компьютерного моделирования Autodesk 3DS MAX 2013. [Электронный ресурс]. – Режим доступа: http://bourabai.kz/graphics/3dsmax.htm. – Дата доступа: 11.03.2025.

13. Создание анимации в Unity [Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/ru/235011/. – Дата доступа: 02.05.2025.

14. Unity // ВикипендиЯ [Электронный ресурс]. – 2004. – Режим доступа: https://ru.m.wikipedia.org/wiki/Unity_(%D0%B8%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA). – Дата доступа: 10.05.2025.

15. Префабы в Unity3D [Электронный ресурс]. – Режим доступа: http://unity.ogf.su/Documentation. – Дата доступа: 06.05.2025.

16. Программное обеспечение 3DS MAX для моделирования и визуализации. [Электронный ресурс] – Режим доступа: https://www.autodesk.ru/products/3ds. – Дата доступа: 11.03.2025.

17. Проектирование непредсказуемого интеллекта в играх. Часть 1 – архитектура. [Электронный ресурс] – Режим доступа: https://www.autodesk.ru/products/3ds. – Дата доступа: 11.03.2025.

18. Borisova S.V., Khobov A. A. Use of Games Forms of Learning in Distance Education on the Example of Physics Experiments [Electronic resource] – Mode of access: https://ieeexplore.ieee.org/document/9783010. – Date of access: 09.03.2025.

19. Fatima Rubio, The 5 Most Powerful Pathfinding Algorithms // Graphable 2023, [Electronic resource] – Mode of access: https://www.graphable.ai/blog/pathfinding-algorithms/. – Date of access: 09.03.2025.

20. Ian Millington, Artificial intelligence for games. / Millington, Ian. – Morgan Kaufmann Publishers, 2006. – 835 с. – ISBN 13: 978-0-12-497782-2.

21. Introduction to Hierarchical State Machines [Electronic resource] – Mode of access: https://barrgroup.com/blog/introduction-hierarchical-state-machines. – Date of access: 09.03.2025.

22. Tekoälyllä ohjataan pelihahmoja VR-harjoittelusimulaatiossa [Electronic resource] – Mode of access: https://blogit.lab.fi/labfocus/tekoalylla-ohjataan-pelihahmoja-vr-harjoittelusimulaatiossa/. – Date of access: 09.03.2025.