МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Рецензент
_____
«___»_____ 2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2019 г.

Разработка компьютерной игры «A-live» на платформе Unity

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.п.н., доцент каф. ЭВМ
_____Ю.Г. Плаксина
«___»_____2019 г.

Автор работы,
студент группы КЭ-452
_____А.С. Барышников
«___»_____2019 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2019 г.

Челябинск-2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2019 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-452
Барышников Александр Сергеевич
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Разработка компьютерной игры **«**A-live» на платформе Unity» утверждена приказом по университету от 25 апреля 2019 г. №899

2. **Срок сдачи студентом законченной работы:** 1 июня 2019 г.

3. **Исходные данные к работе:**
   - Вагнер Б.М. C# Эффективное программирование – М.: Изд-во ЛОРИ, 2017. - 320 с.;
   - Мартин Р.С Чистый код: создание, анализ и рефакторинг. – М.: Изд-во Питер, 2018. – 464 с.;
   - Хокинг Д.Р. Unity в действии. – Спб: Изд-во Питер, 2016. – 704 с.;

4. **Перечень подлежащих разработке вопросов:**

- провести анализ компьютерных игр с аналогичным функционалом;
- провести анализ функциональных и нефункциональных требований;
- спроектировать и разработать компьютерную игру;
- протестировать работоспособность разработанной компьютерной игры в нагрузочном тестировании.

5. **Дата выдачи задания:** 1 декабря 2018 г.

Руководитель работы _____/Ю.Г. Плаксина/

Студент _____/А.С. Барышников/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2019	
Разработка модели, проектирование	01.04.2019	
Реализация системы	01.05.2019	
Тестирование, отладка, эксперименты	15.05.2019	
Компоновка текста работы и сдача на нормоконтроль	24.05.2019	
Подготовка презентации и доклада	30.05.2019	

Руководитель работы _____/Ю.Г. Плаксина/

Студент _____/А.С. Барышников/

Аннотация

А.С. Барышников. Разработка компьютерной игры «A-live» на платформе Unity. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2019, 66 с., 19 ил., библиогр. список – 50 наим.

Выпускная квалификационная работа была выполнена с целью разработки компьютерной игры.

В данной работе были изучены существующие методы разработки компьютерных игр, выполнен анализ требований, проектирование и тестирование разработанного приложения. Для разработки использовался движок Unity и технология процедурной генерации уровней. Основной целью разработки являлось создание продвинутого искусственного интеллекта неигровых персонажей.

Результатом выполненной работы является правильно функционирующая компьютерная игра.

ОГЛАВЛЕНИЕ

ГЛОССАРИЙ

Игровой движок (Game Engine) – программное обеспечение для разработки игр, которое предоставляет для этого все необходимые инструменты.

NPC (неигровой персонаж) – персонаж в игре, который находится под управлением самой игры.

Внутриигровая механика – совокупность определенных правил взаимодействия игры с игроком.

Коллизия – физический расчет столкновения нескольких игровых объектов.

Рендеринг – процесс получения изображения модели с помощью компьютерной программы.

Билд – подготовленная для использования программа.

Реиграбельность – качественная характеристика игры, которая показывает степень того, насколько игроки хотят сыграть в эту игру ещё раз.

Игровая локация – отдельная область виртуального мира игры.

Процедурная генерация – автоматическое создание игрового контента с помощью алгоритмов игры.

ВВЕДЕНИЕ

В настоящее время компьютерные игры пользуются большой популярностью[1]. По данным J'son Partners Consulting [2] рынок компьютерных игр является самым большим сегментом мирового рынка цифрового контента, ежегодно генерируя многомиллиардные доходы и привлекая огромную аудиторию[3].

В данный момент к разработке игр предъявляется ряд требований, реализуемых широким стеком различных технологий[4,5].

Среди наиболее важных требований выделяют:

– качество графики;

– звуковое сопровождение элементов игры;

– искусственный интеллект окружающего мира и неигровых персонажей(NPC);

– разветвленный, нелинейный сюжет;

– разнообразный геймплей.

В настоящее время представлено множество технологий [6,7] для реализации графического и звукового сопровождения.

Самыми популярными технологиями для отображения графики являются [8,9,10,11]:

– AMD TressFX;

– Voxel Cone Tracing;

– Tessellation;

– Realtime Ambient Occlusion.

В то время как для разработки искусственного интеллекта окружающего мира и NPC не существует готовых решений. Проработанный искусственный интеллект окружающего мира и NPC – интеллект, который позволяет NPC осуществлять множество взаимодействий с окружающим миром и персонажем

игрока, при этом действия NPC должны казаться игроку максимально естественными.

На создание подобных игр требуется достаточно много времени[12] и дополнительные финансовые вложения [13], но компьютерные игры с глубокой проработкой искусственного интеллекта пользуются большим спросом у игроков[14].

Создание игры с проработанным искусственным интеллектом является актуальной задачей и на сегодняшний день.

Цели и задачи

Целью работы является разработка компьютерной игры с проработанным искусственным интеллектом.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести обзор аналогичных решений и осуществить постановку задачи;
- провести обзор современных средств реализации;
- провести анализ требований и спроектировать компьютерную игру с проработанным искусственным интеллектом неигровых персонажей;
- реализовать компьютерную игру;
- провести тестирование реализованной компьютерной игры.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. ОБЗОР АНАЛОГОВ

На сегодняшний день существует множество компьютерных игр разных жанров, однако в играх жанра RPG[15] чаще всего встречается проработанный искусственный интеллект NPC и окружающий игрока мир[16]. Компьютерная ролевая игра (RPG) – жанр компьютерных игр, основанных на элементах традиционных настольных игр. В этих играх игрок управляет персонажем, с определенным набором численных характеристик и набором способностей[17].

Рассмотрим самые популярные компьютерные игры этого жанра за последние пять лет[18].

Witcher 3: Wild Hunt

Witcher 3: Wild Hunt [19, 20, 21] – компьютерная игра, разработанная компанией CD Project RED и выпущенная в 2015 году. Разработка игры велась на игровом движке REDengine 3 с использованием языков C++ и Java.

Достоинства:

- игровой мир, для полного исследования которого понадобится более 40 часов реального времени[22];
- игровые механики:
  - выслеживание;
  - выживание;
  - зельеварение и т.п. [23];
- вариативность прохождения, которая заключается в возможности выбора из нескольких вариантов ответов в диалогах, а также возможности прохождения заданий различными способами;
- множество концовок игры, получение каждой из которых зависит от различных действий игрока;
- проработанный искусственный интеллект;

– кроссплатформенность.

Недостатки:

– большое количество ошибок, связанных с неправильным расчетом коллизий (персонажи застревают в игровых объектах);
– некорректная работа с локализованной озвучкой персонажей, вызванная техническими особенностями игрового движка (фиксированная длина аудиодорожки);
– слабо проработанный баланс системы улучшений навыков персонажа игрока (часть навыков полезнее других, из-за чего вариативность улучшений навыков работает некорректно);

Divinity: Original Sin 2

Divinity: Original Sin 2 [24, 25, 26] – компьютерная игра, в жанре RPG с элементами пошаговой стратегии, разработанная компанией Larian Studios и выпущенная в сентябре 2017 года. Разработка игры велась на игровом движке Divinity Engine 2.

Достоинства:

– наличие однопользовательского и многопользовательского режимов;
– пошаговая система боев с возможностью взаимодействия с большим количеством объектов окружающего мира во время боя и влияние этих объектов на персонажей;
– сюжет с множеством диалогов и возможностью выбора одного из нескольких вариантов ответов в диалогах;
– наличие нескольких концовок игры;
– кроссплатформенность;
– игровые механики:
    – телепортация;
    – изучение неизвестных языков;

- режим скрытности, позволяющий становится невидимым в определенных местах и т.п. [27].

Недостатки:

- ошибки с невозможностью завершения заданий NPC, связанные с неправильной обработкой последовательности действий игрока;
- неправильный расчет количества урона персонажам, связанный с возможностью экипировать снаряжение невзирая на игровые ограничения (например возможность экипировать одежду, для ношения которой уровень игрока слишком мал);
- ошибки в локализации игры на разные языки.

The Banner Saga

The Banner Saga [28, 29, 30] – компьютерная игра с пошаговым боевым режимом, разработанная компанией Versus Evil. Разработка игры велась на игровом движке Banner Saga ENGINE.

Достоинства:

- игровые механики:
  - распределение пищи;
  - создание караванов;
  - заработок славы и т.п.[31];
- пошаговые бои, в которых необходимо учитывать процедурно генерируемую местность;
- сюжет с большим количеством диалогов с возможностью выбора одного из нескольких вариантов ответов в диалогах;

Недостатки:

- различные ошибки, связанные с некорректной обработкой нанесенного персонажам урона;
- ошибки в локализации игры.

Результаты обзора существующих аналогов представлены в табл. 1.

Таблица 1 - Результаты обзора существующих аналогичных продуктов

Название игры	**Кроссплат-форменность**	**Проработанный искусственный интеллект**	**Наличие нескольких игровых механик**	**Сравнительное количество ошибок в игровом процессе**
Witcher 3: Wild Hunt	+	+	+	Некритичные ошибки в игровом процессе
Divinity: Original Sin 2	+	-	+	Критичные ошибки в игровом процессе
The Banner Saga	-	-	+	Некритичные ошибки в игровом процессе

Критичные ошибки в игровом процессе – ошибки, которые делают дальнейший игровой процесс невозможным.

Из полученных результатов можно заметить, что большинство игр, помимо мелких недостатков, имеют существенный – непроработанный искусственный интеллект. Также просматривается тенденция на реализацию нескольких игровых механик и кроссплатформенности. Эти данные показывают актуальность разработки компьютерной игры с этими особенности.

1.2. АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

В настоящее время существует множество сред для разработки компьютерных игр различных жанров, имеющих разные характеристики. Рассмотрим достоинства и недостатки наиболее распространенных из них[32].

Unity

Unity [33,34,35] – кроссплатформенный игровой движок, разрабатываемый компанией Unity Technologies. Основной язык программирования для работы с Unity - C#. Unity позволяет выпускать игры для компьютеров, игровых приставок, мобильных телефонов.

Достоинства:

- кроссплатформенность;
- большое количество инструментов для создания искусственного интеллекта;
- легкость в освоении;
- универсальность.

Недостатки:

- закрытый исходный код;
- условно бесплатный[36].

Unreal Engine

Unreal Engine [37,38,39] – кроссплатформенный игровой движок, разрабатываемый компанией Epic Games. Для разработки игр в Unreal Engine используется язык C++, также имеется возможность использования визуального редактора скриптов BluePrint.

Достоинства:

- кроссплатформенность;
- открытый исходный код;
- высокое качество графики.

Недостатки:

- условно бесплатный[40];
- малое количество разработчиков и обучающего материала.

CryEngine

CryEngine [41,42,43] – игровой движок, разрабатываемый компанией Crytek. Разработка в CryEngine ориентирована на создание игр для персональных компьютеров.

Достоинства:

- высокое качество графики;
- удобство в программировании искусственного интеллекта.

Недостатки:

- коммерческая лицензия.

Ввиду большей ориентированности на создание проработанного искусственного интеллекта, рационально выбрать Unity для создания игры.

1.3. ВЫВОД

После проведения обзора существующих аналогов и анализа основных технологических решений были выявлены проблемы и определены возможности их устранения.

В сфере компьютерных игр высок спрос на игры с множеством механик [44]. Неотъемлемой частью игры должен стать проработанный искусственный интеллект окружающего мира и NPC. При этом игра должна быть спроектирована и реализована таким образом, чтобы допускалась возможность дальнейшего расширения игры путем добавления новых игровых механик, игровых локаций и неигровых персонажей.

При изучении статистических данных по играм [45,46] была замечена важность реиграбельности, которой можно достигнуть путем внедрения процедурной генерации уровней и звуковой составляющей.

Также, согласно данным обзора существующих аналогов [47], игра должна быть кроссплатформенной.

Для демонстрации и тестирования игры необходимо разработать демонстрационный билд, включающий в себя возможность прохождения игры и апробации всех созданных механик.

Таким образом, задача о реализации игры с поддержкой кроссплатформенности, с наличием проработанного искусственного интеллекта NPC и окружающего мира, процедурно генерируемым миром, множеством механик, а также минимальным количеством ошибок в игровом процессе является актуальной.

2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

К общим требованиям можно отнести минимальные системные требования:

- для мобильного телефона:
  - операционная система: Android 4.2 или выше;
  - процессор: 2 ядерный процессор с частотой 1500 МГц или лучше;
  - оперативная память: 2 GB или больше;
  - видеочип: Mali-400MP/PowerVR SGX544/Adreno 320/Tegra 3 или лучше;
- для персонального компьютера:
  - операционная система (ОС): Windows 7 или выше/Linux;
  - процессор: Intel Core i3 540/AMD A6-3620 или лучше;
  - оперативная память: 4 GB;
  - видеокарта: видеокарта с поддержкой DirectX 11.0 с 1 GB RAM (NVidia GeForce 460/ AMD Radeon 6850) или лучше;
  - DirectX: версии 11;
  - место на диске: 2GB;
  - звуковая карта: звуковая карта совместимая с DirectX.

Данные системные требования являются минимальными для продукции, которая создается на движке Unity.

2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

В рамках реализуемого приложения предусмотрен один актер – пользователь, который взаимодействует с проигрываемым сюжетом игры и компонентами меню.

На рисунке 1 представлена диаграмма вариантов использования.

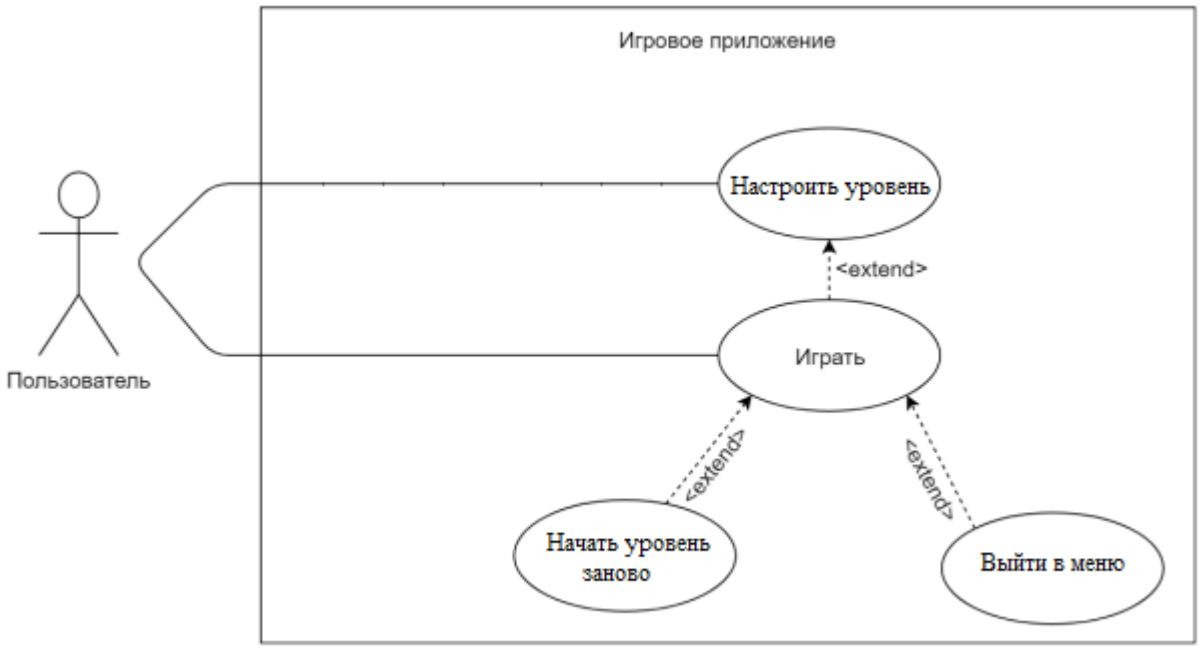

Рисунок 1 - Диаграмма вариантов использования

Пользователь может:

– *настроить уровень* – ему предоставляется возможность выбрать размер карты;

– *играть* – начать проходить уровень, который будет генерироваться с условиями, выбранными в настройках уровня;

– *начать уровень заново* – начать проходить уровень заново, сгенерировав новую карту с размерами указанными ранее;

– *выйти в меню* – пользователь может прервать игру и выйти в меню, чтобы выбрать новые параметры для игры или выйти из игры.

2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

К нефункциональным требованиям относятся:

– разработанное приложение должно соответствовать определенным минимальным системным требованиям;

– разработанное приложение должно быть написано на языке C# на платформе Unity;

– разработанное приложение должно иметь процедурную генерацию уровней;

– разработанное приложение должно иметь проработанный искусственный интеллект.

3. ПРОЕКТИРОВАНИЕ

3.1. АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ

Проект разработанного игрового приложения представляет собой список каталогов, содержащих в себе:

- сцены;
- игровые скрипты;
- готовые компоненты;
- текстуры;
- анимации.

На рисунке 2 представлена файловая структура приложения.

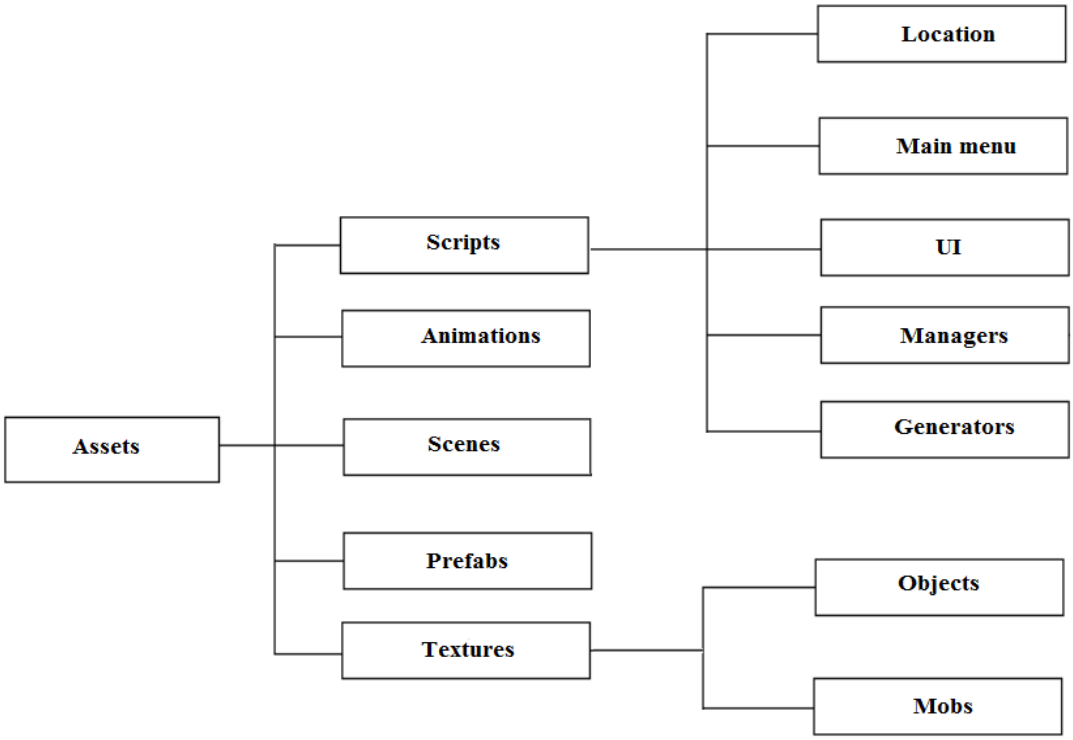

Рисунок 2 – Файловая структура приложения

3.2. АЛГОРИТМЫ ПРОЦЕДУРНОЙ ГЕНЕРАЦИИ

Для создания игровых уровней была использована технология процедурной генерации уровней.

Генерация уровня происходит в несколько этапов:

– определение размера локации и создание базового слоя игрового мира (игровой поверхности);

– заполнение игровой поверхности игровыми объектами;

– заполнение игровой поверхности NPC.

При запуске игрового уровня происходит инициализация данных, хранящихся в корневых папках игры, которые включают в себя все модели и текстуры, которые нужны для создания локации. Определяется размер локации из данных, которые игрок ввел заранее.

После проведенных действий создается базовый слой игрового мира, который состоит из девяти заранее заготовленных частей, восемь из которых являются краями игрового мира и образуют прямоугольник. Все пространство внутри этого прямоугольника заполняется копиями оставшейся заготовленной части.

Следующий шаг заключается в том, что с помощью генератора случайных чисел выбираются координаты на игровой локации, в которых будут размещены нужные нам игровые объекты. Количество игровых объектов вычисляется из размера локации. После этого проводится процедура проверки на оригинальность координат, чтобы исключить их совпадение. Впоследствии идет инициализация игровых объектов на этих координатах.

Часть объектов, такие как озера и леса с едой также создаются с помощью процедурной генерации. Это позволяет создавать их разной формы и размеров.

Генерация озер реализуется из заранее заготовленных восемнадцати частей. Для создания озера используется девять случайно выбранных из этих восемнадцати частей.

Генерация лесов протекает следующим образом. Координаты инициализации леса являются его центром. Вокруг точки инициализации

выделяется квадрат, который разбивается на девять более мелких квадратов. Именно с ними и происходит последующая работа. Центральный из этих девяти квадратов заполняется лесом. Из оставшихся восьми квадратов случайным образом выбирается два, которые также заполняются лесом.

На следующем шаге вычисляются координаты инициализации NPC. Происходит проверка на оригинальность координат. Проверка оригинальности начальных координат NPC также затрагивает координаты игровых объектов, чтобы исключить застревание NPC в игровых объектах.

Код процедурной генерации приведен в листинге 1 приложения А.

3.3. ОПИСАНИЕ РАБОТЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА НЕИГРОВЫХ ПЕРСОНАЖЕЙ

При создании игрового приложения было решено создать два вида неигровых персонажей:

- агрессивные NPC;
- неагрессивные NPC.

У всех видов NPC были реализованы потребности в пище и воде, которые они должны были удовлетворять. Реализовано это двумя характеристиками:

- насыщенность едой;
- насыщенность водой.

Показатель насыщенности уменьшается со временем, если показатель еды или воды упадет ниже определенного минимума, тогда NPC исчезнет из игрового мира.

Агрессивные и неагрессивные NPC будут пытаться найти водоемы или леса с едой при необходимости. Также при коллизии двух агрессивных или неагрессивных NPC друг с другом и при достаточной степени насыщенности водой и пищей, будут инициализированы двое новых NPC этого вида.

Агрессивные NPC при коллизии с неагрессивными NPC или персонажем игрока могут уничтожить того, с кем произошла коллизия при недостаточной насыщенности едой агрессивного NPC.

Код работы искусственного интеллекта NPC представлен в листинге 2 приложения Б.

В игровом приложении также реализована смена времен года. В разные времена года NPC имеют разный метаболизм, а также последовательность действий.

Неагрессивные NPC осенью ищут место для зимовки, а зимой не предпринимают никаких действий, тем самым эмулируют спячку.

Агрессивные NPC летом осуществляют поиск неагрессивных NPC, чтобы пополнить потребности в пище.

Баланс игры построен так, что в начале игры, количество агрессивных NPC гораздо меньше количества неагрессивных NPC, однако с каждым годом количество агрессивных NPC будет возрастать. Это сделано для того, чтобы игрок старался проходить игру как можно быстрее, так как со временем будет возрастать сложность игры из-за увеличивающегося количества агрессивных NPC.

4. РЕАЛИЗАЦИЯ

4.1. РЕАЛИЗАЦИЯ ИНТЕРФЕЙСОВ

В ходе создания приложения для взаимодействия программы с игроком были созданы следующие интерфейсы:

– главное меню;

– интерфейс персонажа.

Для большего удобства обращения игрока с приложением было разработано главное меню, включающее в себя следующие пункты:

– новая игра;

– настройки;

– авторы;

– выход.

В пункте новая игра игрок может выбрать параметры новой игры, которые включают в себя размер карты. В пункте настройки игрок может выбрать такие настройки как:

– настройка качества графики;

– настройка громкости звука;

– полноэкранный или оконный режим игры.

На рисунках 3 – 5 представлено главное меню.

Рисунок 3 – Главное меню игры

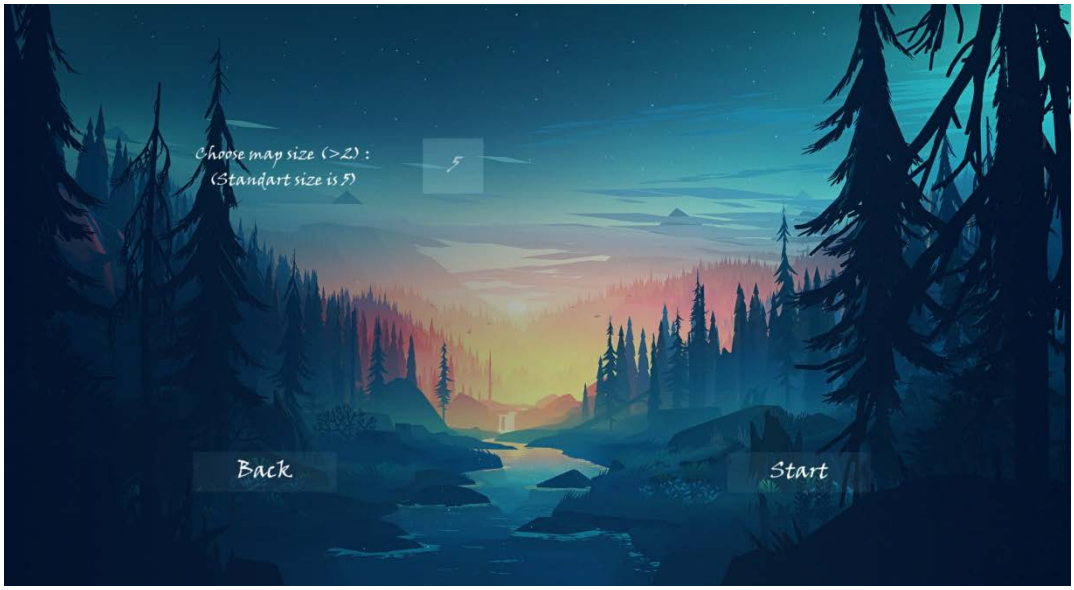

Рисунок 4 – Пункт меню новая игра

Рисунок 5 – Пункт меню настройки

Для лучшего понимания игроком действий, происходящих в игровом процессе, был реализован интерфейс персонажа, который включает в себя:

– количество собранных частей рации, необходимых для прохождения игры;

– насыщение персонажа едой;

– насыщение персонажа водой.

На рисунке 6 представлен интерфейс персонажа.

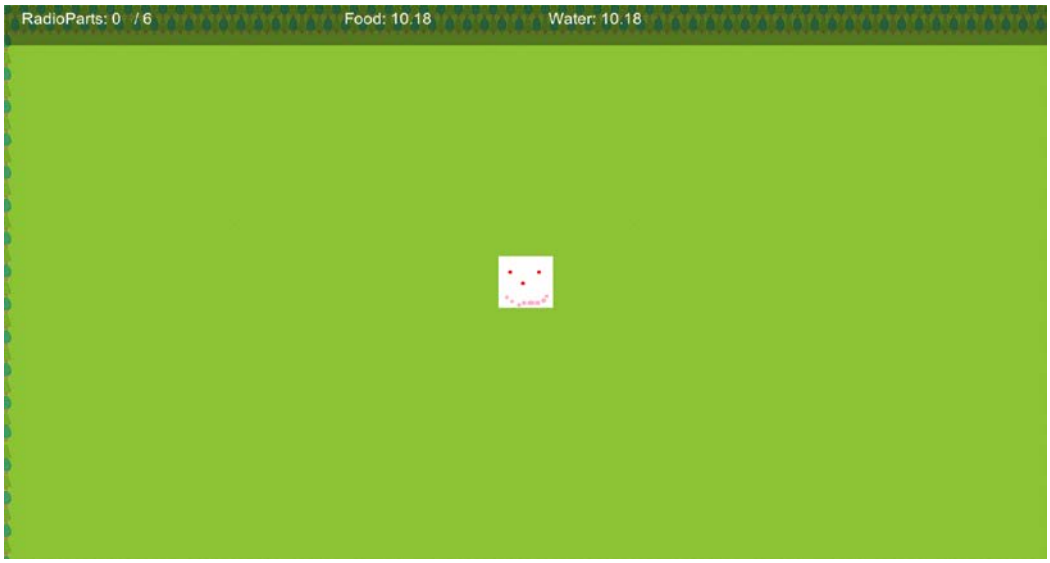

Рисунок 6 – Интерфейс персонажа

4.2. РЕАЛИЗАЦИЯ ПРОЦЕДУРНОЙ ГЕНЕРАЦИИ УРОВНЯ

В ходе создания приложения была реализована процедурная генерация уровней.

Примеры работы процедурной генерации уровней представлены на рисунках 7-9.

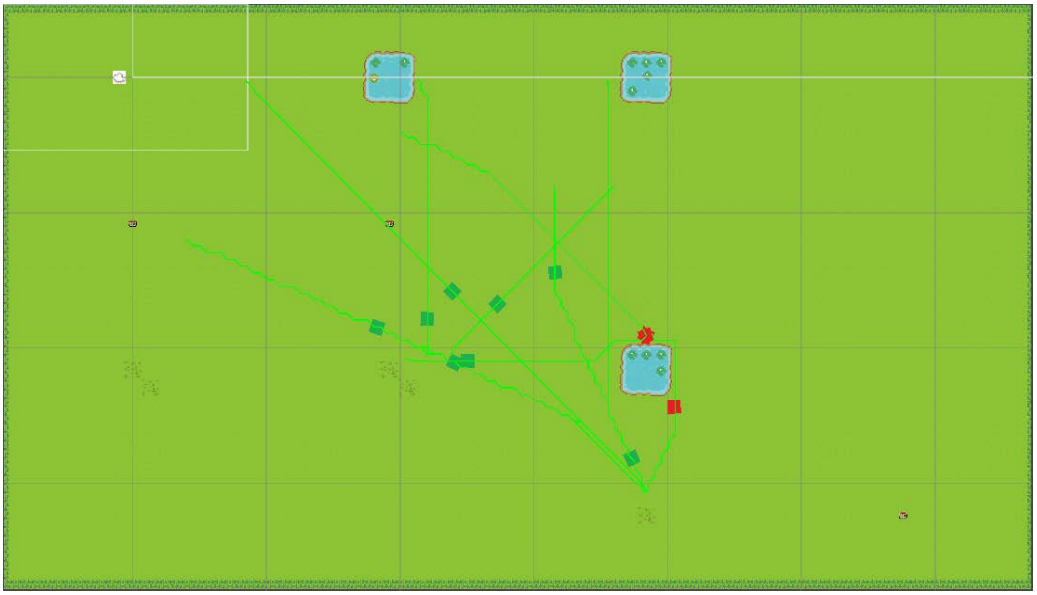

Рисунок 7 – Игровая локация размером 4

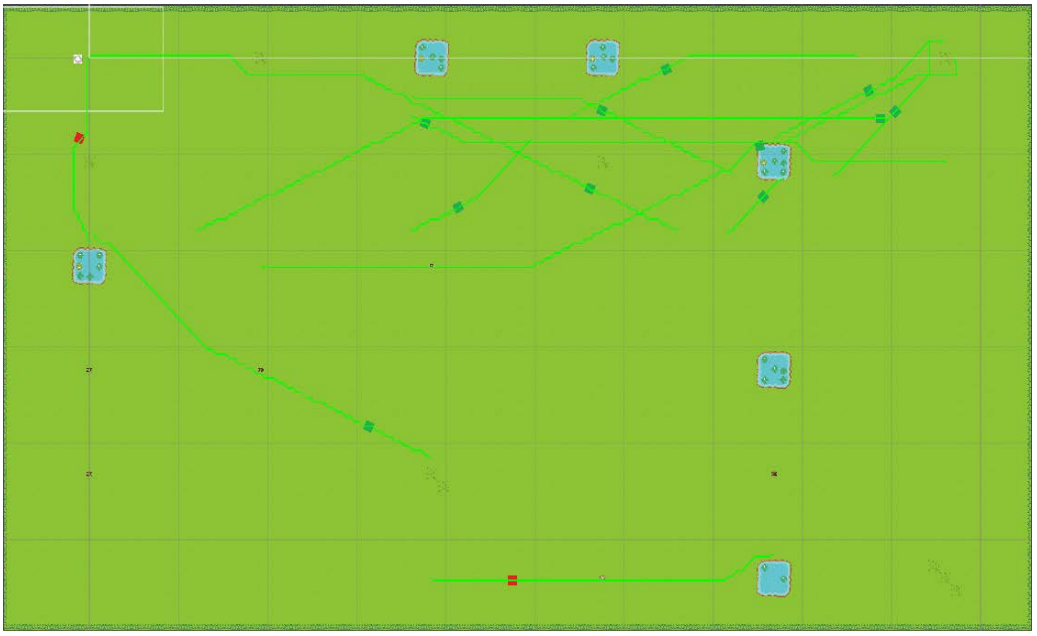

Рисунок 8 – Игровая локация размером 6

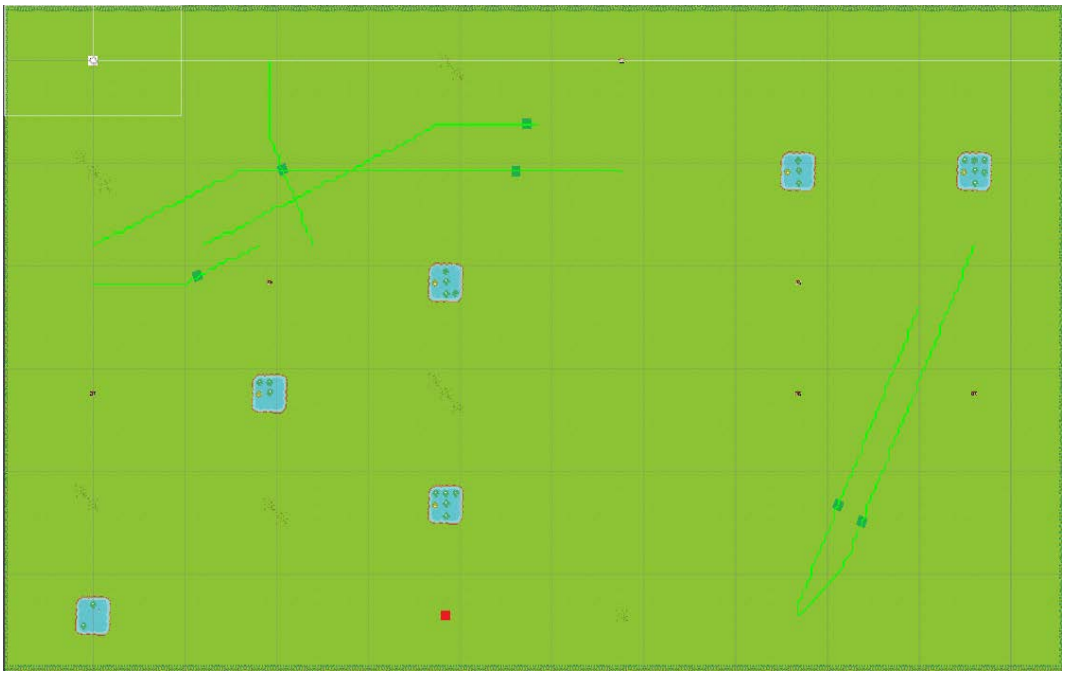

Рисунок 9 – Игровая локация размером 6

5. ТЕСТИРОВАНИЕ

5.1. МЕТОДОЛОГИЯ ТЕСТИРОВАНИЯ

В рамках тестирования компьютерной игры использовалось нагрузочное тестирование. Все тесты проводились при помощи разработанной демонстрационной версии игры.

5.2. ПРОВЕДЕНИЕ ПРОЦЕДУРЫ ТЕСТИРОВАНИЯ

В рамках нагрузочного тестирования были произведены тесты для определения производительности и времени отклика компьютерной игры. На нескольких системах с различными конфигурациями были произведены замеры времени на отрисовку одного кадра, а также среднее количество FPS при запуске уровня и через 2 минуты после запуска уровня. Все тесты были проведены пять раз[48], а результаты тестов усреднены.

Технические характеристики персонального компьютера №1 использованного в тестировании:

- операционная система (ОС): Windows 10;
- процессор: AMD Ryzen 1700 с частотой 3.9 МГц;
- оперативная память: 16 GB;
- видеокарта: AMD RX 580 с 4GB RAM;

Технические характеристики персонального компьютера №2 использованного в тестировании:

- операционная система (ОС): Windows 10;
- процессор: AMD FX-8320 с частотой 4.3 МГц;
- оперативная память: 12 GB;
- видеокарта: NVidia GTX 1070 с 8GB RAM;

Полученные данные для компьютера №1 представлены на рисунках 10-14.

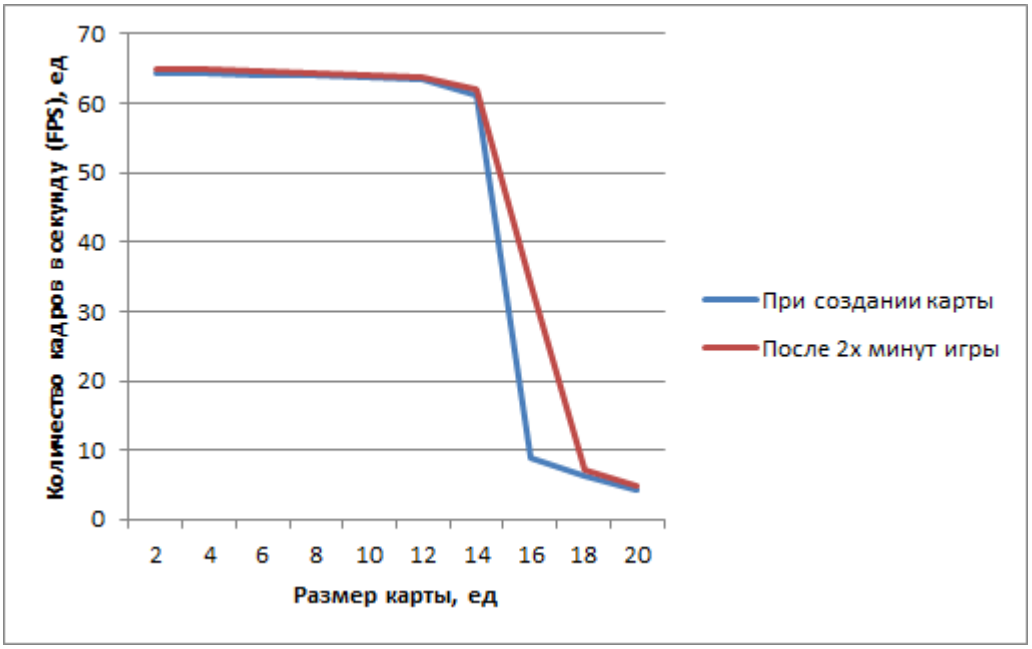

Рисунок 10 – Зависимость FPS от размера карты

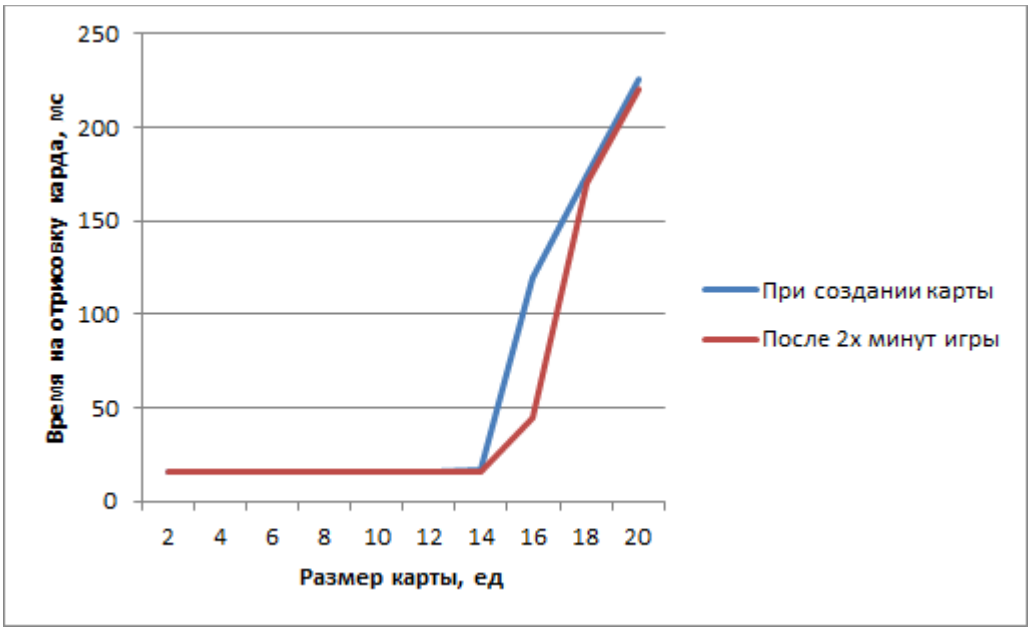

Рисунок 11 – Зависимость времени на отрисовку кадра от размера карты

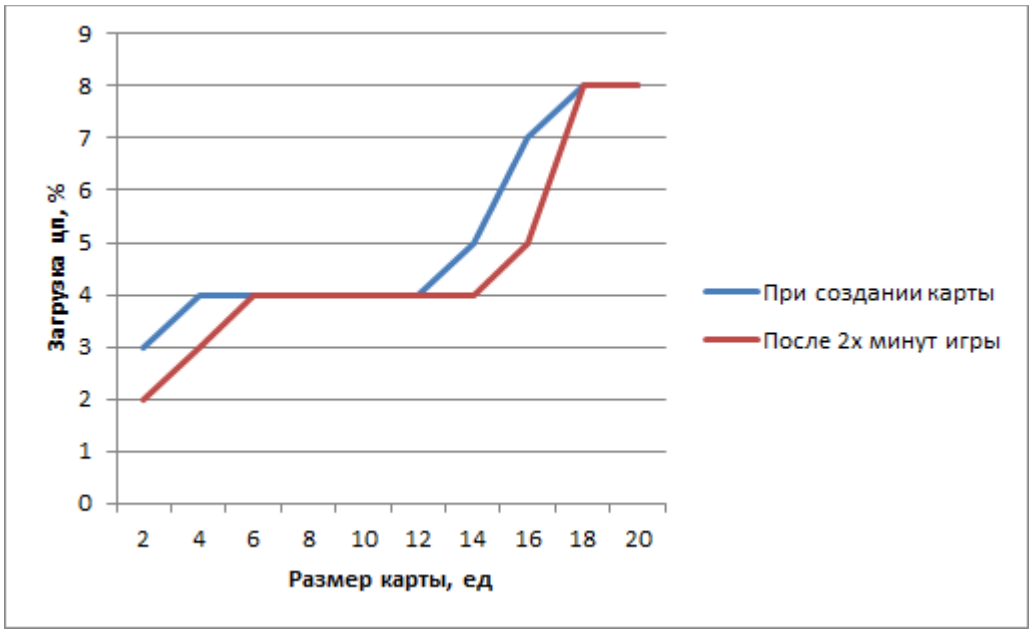

Рисунок 12 – Зависимость загрузки центрального процессора от размера карты

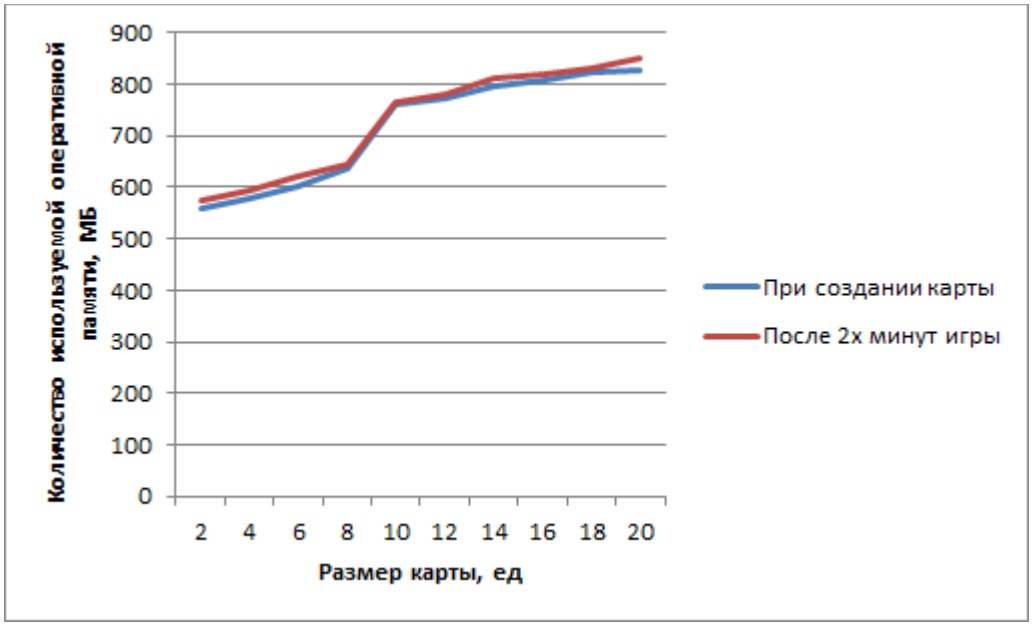

Рисунок 13 – Зависимость загрузки оперативной памяти от размера карты

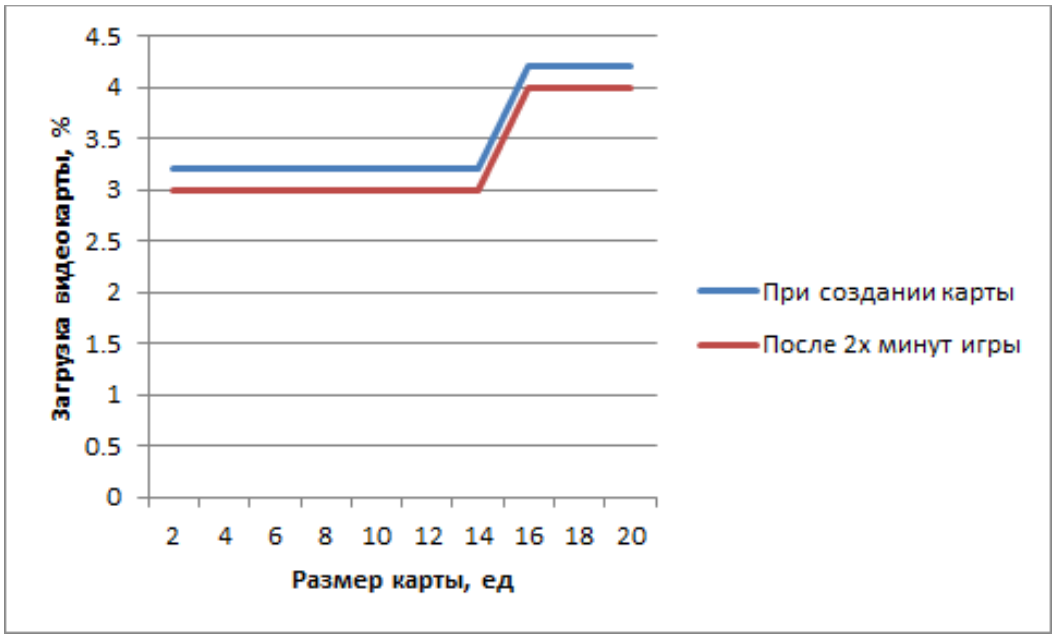

Рисунок 14 – Зависимость загрузки видеокарты от размера карты

Данные полученные для компьютера №2 представлены на рисунках 15-19.

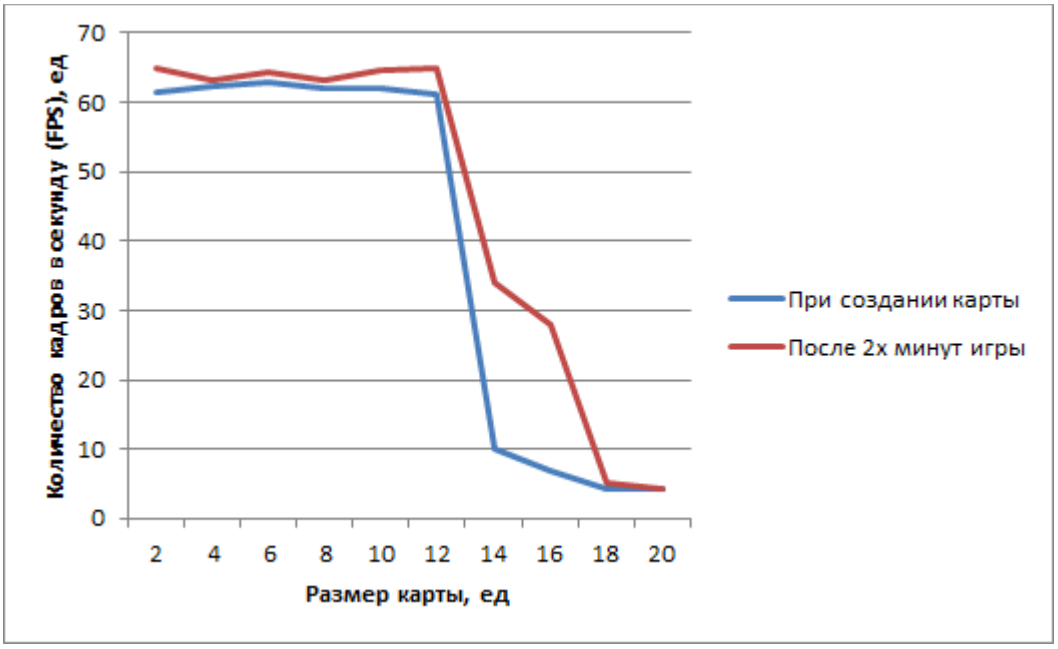

Рисунок 15 – Зависимость FPS от размера карты

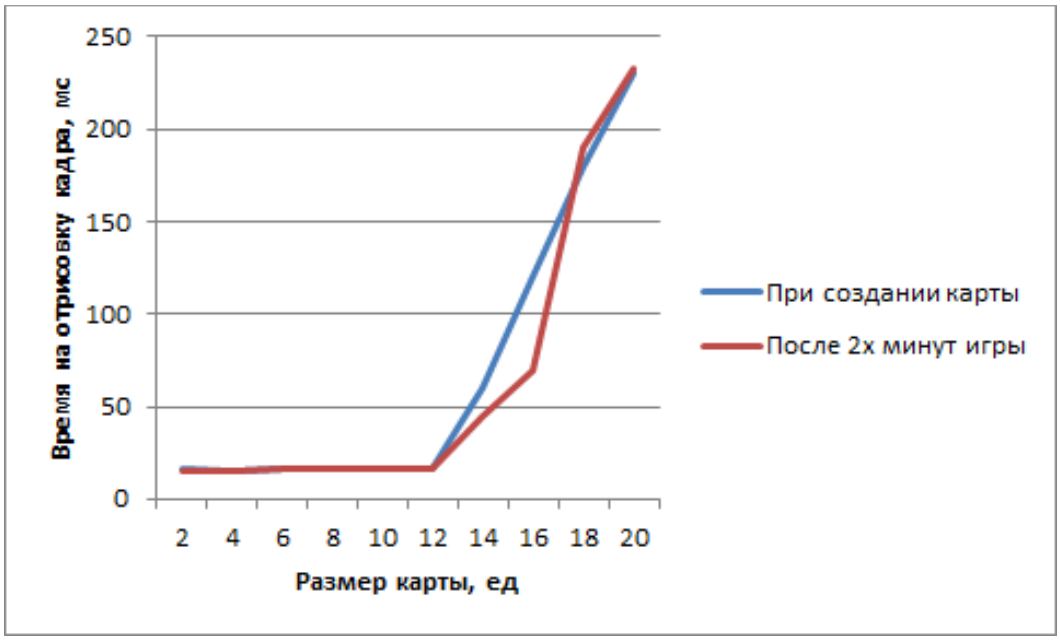

Рисунок 16 – Зависимость времени на отрисовку кадра от размера карты

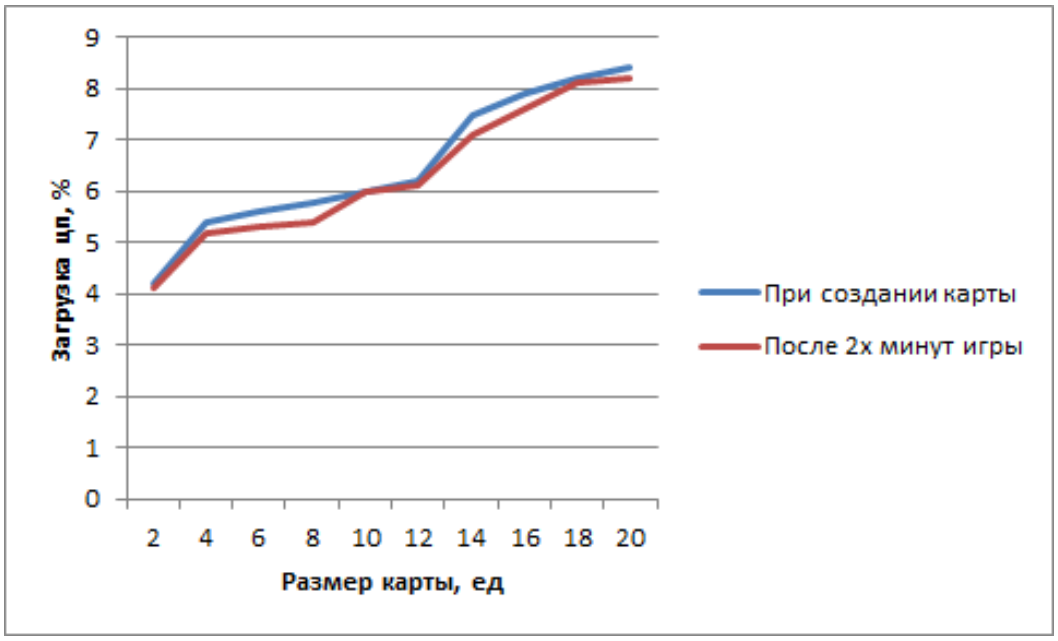

Рисунок 17 – Зависимость загрузки центрального процессора от размера карты

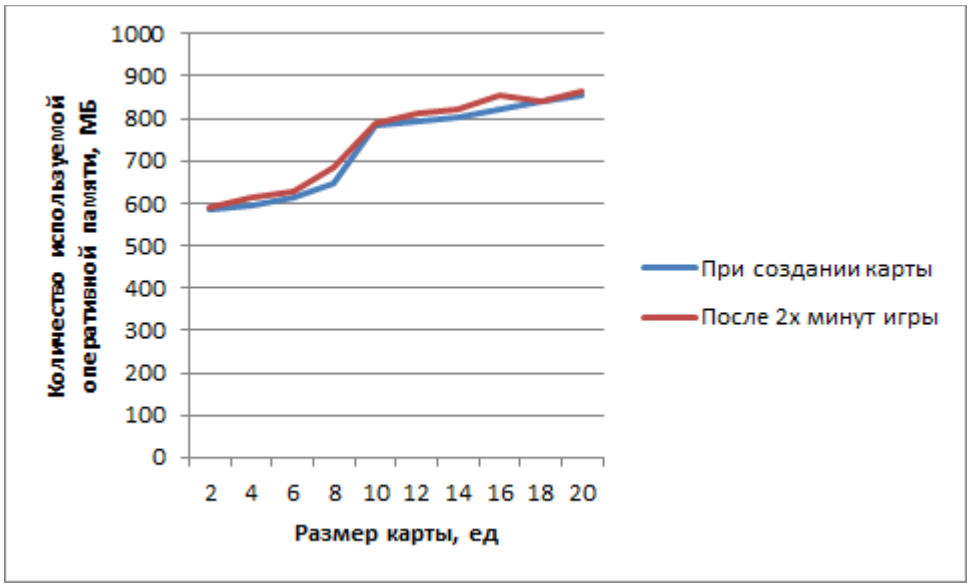

Рисунок 18 – Зависимость загрузки оперативной памяти от размера карты

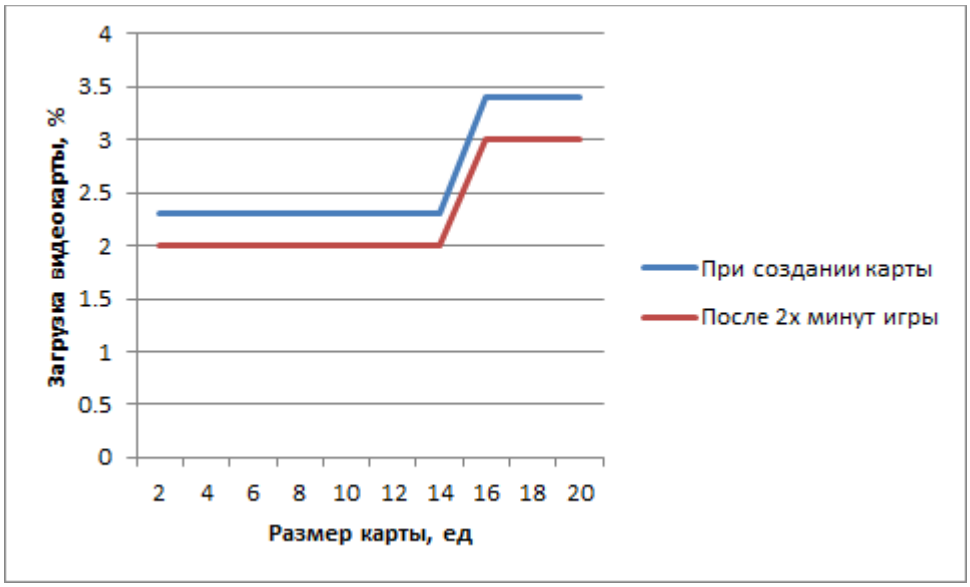

Рисунок 19 – Зависимость загрузки видеокарты от размера карты

На основании полученных данных можно сделать следующие выводы:

- разработанная компьютерная игра соответствует заявленным минимальным требованиям;
- на компьютерах с конфигурацией аналогичной той, что используются в тестах можно играть с комфортным FPS (30 или более кадров [49]) при размере карт не более 12.

Во время тестирования не было обнаружено никаких ошибок связанных с коллизией объектов и ошибочными расчетами траектории движения NPC.

6. ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы была разработана компьютерная игра «A-live» на платформе Unity.

При этом были решены следующие задачи:

- проведен обзор аналогичных решений и осуществлена постановка задачи;
- проведен обзор средств реализации;
- проведен анализ требований и спроектирована компьютерная игра;
- реализована демонстрационная версия игры;
- проведено тестирование реализованной версии игры.

Перспективы дальнейшего развития игры:

- адаптация на другие платформы;
- добавление новых NPC;
- увеличение количества игровых механик;
- увеличение количества игровых локаций;
- добавление игровых заданий.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Веб-сайт «Gamesindustry.biz». [Электронный ресурс]. URL: – https://www.gamesindustry.biz/articles/2018-12-18-global-games-market-value-rose-to-usd134-9bn-in-2018. (Дата обращения: 17.04.2019).

2. Веб-сайт «J'son Partners Consulting». [Электронный ресурс]. URL: – http://security.mosmetod.ru/internet-zavisimosti/83-analiz-rynka-igr-v-rossii-i-mire. (Дата обращения: 18.04.2019).

3. Веб-сайт «Gamesindustry.biz». [Электронный ресурс]. URL: – https://www.gamesindustry.biz/articles/2018-12-17-gamesindustry-biz-presents-the-year-in-numbers-2018. (Дата обращения: 16.04.2019).

4. Веб-сайт «Upgard.com». [Электронный ресурс]. URL: – https://www.upgrad.com/blog/6-components-of-video-game-design/. (Дата обращения: 16.04.2019).

5. Веб-сайт «Gamingilluminaughty.com». [Электронный ресурс]. URL: – https://www.gamingilluminaughty.com/what-are-the-most-important-aspects-of-video-games/. (Дата обращения: 16.04.2019).

6. Веб-сайт «Stopgame.ru». [Электронный ресурс]. URL: – https://stopgame.ru/blogs/topic/55667. (Дата обращения: 16.04.2019).

7. Веб-сайт «Stopgame.ru». [Электронный ресурс]. URL: – https://stopgame.ru/blogs/topic/55415. (Дата обращения: 16.04.2019).

8. Веб-сайт «Amd.com». [Электронный ресурс]. URL: – https://www.amd.com/ru/technologies/tressfx. (Дата обращения: 16.04.2019).

9. Веб-сайт «Research.nvidia.com». [Электронный ресурс]. URL: – https://research.nvidia.com/sites/default/files/pubs/2011-09_Interactive-Indirect-Illumination/GIVoxels-pg2011-authors.pdf. (Дата обращения: 16.04.2019).

10. Веб-сайт «Nvidia.com». [Электронный ресурс]. URL: – https://www.nvidia.com/object/tessellation.html. (Дата обращения: 16.04.2019).

11. Веб-сайт «Cse.chalmers.se». [Электронный ресурс]. URL: – http://www.cse.chalmers.se/~uffe/xjobb/Joakim%20Carlsson-Realistic%20Ambient%20Occlusion%20In%20Real-Time.pdf. (Дата обращения: 16.04.2019).
12. Веб-сайт «Mweb.co.za». [Электронный ресурс]. URL: – https://www.mweb.co.za/games/view/tabid/4210/Article/33048/Most-popular-core-PC-games-of-June-2018-shows-PUBG-beating-Fortnite.aspx. (Дата обращения: 16.04.2019).
13. Веб-сайт «Econdude.pw». [Электронный ресурс]. URL: – https://www.econdude.pw/2017/08/skolko-po-vremeni-delajut-igry.html. (Дата обращения: 16.04.2019).
14. Веб-сайт «Gamesradar.com». [Электронный ресурс]. URL: – https://www.gamesradar.com/best-pc-games/. (Дата обращения: 14.05.2019).
15. Веб-сайт «Metacritic.com». [Электронный ресурс]. URL: – https://www.metacritic.com/browse/games/genre/date/role-playing/pc. (Дата обращения: 15.05.2019).
16. Веб-сайт «Cadelta.ru». [Электронный ресурс]. URL: – https://cadelta.ru/games/id3318. (Дата обращения: 15.05.2019).
17. Веб-сайт «Makeusof.com». [Электронный ресурс]. URL: – https://www.makeuseof.com/tag/5-life-skills-video-games-can-help-develop/. (Дата обращения: 15.05.2019).
18. Веб-сайт «Pcgamer.com». [Электронный ресурс]. URL: – https://www.pcgamer.com/the-pc-gamer-top-100-new/. (Дата обращения: 15.05.2019).
19. Веб-сайт «Steam.com». [Электронный ресурс]. URL: – https://store.steampowered.com/app/292030/The_Witcher_3_Wild_Hunt/?l=russian. (Дата обращения: 15.05.2019).

20. Веб-сайт «Igromania.ru». [Электронный ресурс]. URL: – https://www.igromania.ru/game/14931/Vedmak_3_Dikaya_Ohota. (Дата обращения: 15.05.2019).
21. Веб-сайт «Thewitcher.com». [Электронный ресурс]. URL: – https://thewitcher.com/ru/witcher3. (Дата обращения: 16.05.2019).
22. Веб-сайт «Vedmak.fandom.com». [Электронный ресурс]. URL: – https://vedmak.fandom.com/wiki/. (Дата обращения: 16.05.2019).
23. Веб-сайт «Maximonline.ru». [Электронный ресурс]. URL: – https://www.maximonline.ru/guide/games/_article/the-witcher-wild-hunt/. (Дата обращения: 16.05.2019).
24. Веб-сайт «Steam.com». [Электронный ресурс]. URL: – https://store.steampowered.com/app/435150/Divinity_Original_Sin_2__Definitive_Edition/?l=russian/. (Дата обращения: 16.05.2019).
25. Веб-сайт «Divinity.game». [Электронный ресурс]. URL: – https://divinity.game/age-verification/. (Дата обращения: 16.05.2019).
26. Веб-сайт «Protondb.com». [Электронный ресурс]. URL: – https://www.protondb.com/app/435150/. (Дата обращения: 16.05.2019).
27. Веб-сайт «Cbreaker.net». [Электронный ресурс]. URL: – https://ru.cbreaker.net/divinity-original-sin-2-20297/. (Дата обращения: 16.05.2019).
28. Веб-сайт «Steam.com». [Электронный ресурс]. URL: – https://store.steampowered.com/app/237990/The_Banner_Saga/. (Дата обращения: 16.05.2019).
29. Веб-сайт «Bannersaga.com». [Электронный ресурс]. URL: – https://bannersaga.com/. (Дата обращения: 16.05.2019).
30. Веб-сайт «Thebannersaga.com». [Электронный ресурс]. URL: – https://thebannersaga.fandom.com/ru/wiki/. (Дата обращения: 16.05.2019).

31. Веб-сайт «Stoicstudio.com». [Электронный ресурс]. URL: – https://stoicstudio.com/. Дата обращения: 16.05.2019.
32. Веб-сайт «Websitetooltester.com». [Электронный ресурс]. URL: – https://www.websitetooltester.com/en/blog/best-game-engine/. (Дата обращения: 16.05.2019).
33. Веб-сайт «Unity.com». [Электронный ресурс]. URL: – https://unity.com/ru. (Дата обращения: 16.05.2019).
34. Веб-сайт «Opendu.ru». [Электронный ресурс]. URL: – https://openedu.ru/course/ITMOUniversity/UNITY/. (Дата обращения: 16.05.2019).
35. Веб-сайт «Unity-movement.ru». [Электронный ресурс]. URL: – http://www.unity-movement.ru/. (Дата обращения: 16.05.2019).
36. Веб-сайт «Unity.com». [Электронный ресурс]. URL: – https://store.unity.com/ru. (Дата обращения: 16.05.2019).
37. Веб-сайт «Unrealengine.com». [Электронный ресурс]. URL: – https://www.unrealengine.com/en-US/. (Дата обращения: 16.05.2019).
38. Веб-сайт «Uengine.ru». [Электронный ресурс]. URL: – https://uengine.ru/. (Дата обращения: 16.05.2019).
39. Веб-сайт «Vc.ru». [Электронный ресурс]. URL: – https://vc.ru/pixonic/51306-ue4-guide. (Дата обращения: 16.05.2019).
40. Веб-сайт «Uengine.ru». [Электронный ресурс]. URL: – https://uengine.ru/faq. (Дата обращения: 16.05.2019).
41. Веб-сайт «Cryengine.com». [Электронный ресурс]. URL: – https://www.cryengine.com/. (Дата обращения: 16.05.2019).
42. Веб-сайт «Crytek.com». [Электронный ресурс]. URL: – https://www.crytek.com/cryengine. (Дата обращения: 16.05.2019).
43. Веб-сайт «Cubiq.ru». [Электронный ресурс]. URL: – https://cubiq.ru/dvizhok-cryengine/. (Дата обращения: 16.05.2019).

44. Веб-сайт «Statista.com». [Электронный ресурс]. URL: – https://www.statista.com/topics/1680/gaming/. (Дата обращения: 16.05.2019).
45. Веб-сайт «Steam.com». [Электронный ресурс]. URL: – https://store.steampowered.com/stats/. (Дата обращения: 16.05.2019).
46. Веб-сайт «Wepc.com». [Электронный ресурс]. URL: – https://www.wepc.com/news/video-game-statistics/. (Дата обращения: 16.05.2019).
47. Веб-сайт «Bigfishgames.com». [Электронный ресурс]. URL: – https://www.bigfishgames.com/blog/2017-video-game-trends-and-statistics-whos-playing-what-and-why/. (Дата обращения: 16.05.2019).
48. Веб-сайт «Protesting.ru». [Электронный ресурс]. URL: – http://www.protesting.ru/testing/types/loadtesttypes.html. (Дата обращения: 16.05.2019).
49. Веб-сайт «Docrorrouter.ru». [Электронный ресурс]. URL: – http://www.doctorrouter.ru/skolko-fps-nuzhno-dlya-komfortnoj-igry/. (Дата обращения: 16.05.2019).
50. Барышников А.С. Исходный код компьютерной игры «A-live» Веб-сайт «Github.com». [Электронный ресурс]. URL: – https://github.com/LadyRavenFire/project_aperture. (Дата обращения 16.05.2019).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОЦЕДУРНОЙ ГЕНЕРАЦИИ

Листинг 1 - LocationGenerator

```
public class PlaceGeneration : MonoBehaviour
{
    private LocationGenerator _locationGenerator;

    public int[] XCoordsOfWater;
    public int[] YCoordsOfWater;

    public int[] XCoordsOfFood;
    public int[] YCoordsOfFood;

    public int[] XCoordsOfGreenMobs;
    public int[] YCoordsOfGreenMobs;

    public int[] XCoordsOfRedMobs;
    public int[] YCoordsOfRedMobs;

    public int[] XCoordsOfPartsOfRadio;
    public int[] YCoordsOfPartsOfRadio;

// Функция Start написана для инициализации данных

    void Start()    {
        _locationGenerator =
GameObject.Find("GenerationManager").GetComponent<LocationGenerator>();
        CreateNumberOfCoords();
        CreateRandomPlaces(XCoordsOfWater, YCoordsOfWater);
        CreateRandomPlaces(XCoordsOfFood, YCoordsOfFood);
        CreateRandomPlaces(XCoordsOfPartsOfRadio, YCoordsOfPartsOfRadio);
        CreateRandomPlaces(XCoordsOfGreenMobs, YCoordsOfGreenMobs);
        CreateRandomPlaces(XCoordsOfRedMobs, YCoordsOfRedMobs);
    }

// Функция CreateNumberOfCoords написана для создания массивов координат

    void CreateNumberOfCoords()
       {
        if (_locationGenerator.Size <= 4)
        {
            XCoordsOfWater = new int[_locationGenerator.Size - 1];
            YCoordsOfWater = new int[_locationGenerator.Size - 1];

            XCoordsOfFood = new int[_locationGenerator.Size - 1];
            YCoordsOfFood = new int[_locationGenerator.Size - 1];

            XCoordsOfPartsOfRadio = new int[_locationGenerator.Size -1];
            YCoordsOfPartsOfRadio = new int[_locationGenerator.Size -1];

            XCoordsOfGreenMobs = new int[4];
            YCoordsOfGreenMobs = new int[4];

            XCoordsOfRedMobs = new int[3];
```

```
    YCoordsOfRedMobs = new int[3];
}


if (_locationGenerator.Size >= 5 && _locationGenerator.Size < 10)
{



    XCoordsOfWater = new int[_locationGenerator.Size];
    YCoordsOfWater = new int[_locationGenerator.Size];

    XCoordsOfFood = new int[_locationGenerator.Size];
    YCoordsOfFood = new int[_locationGenerator.Size];

    XCoordsOfPartsOfRadio = new int[_locationGenerator.Size];
    YCoordsOfPartsOfRadio = new int[_locationGenerator.Size];

    XCoordsOfGreenMobs = new int[_locationGenerator.Size];
    YCoordsOfGreenMobs = new int[_locationGenerator.Size];

    XCoordsOfRedMobs = new int[_locationGenerator.Size-4];
    YCoordsOfRedMobs = new int[_locationGenerator.Size-4];
}

if (_locationGenerator.Size >= 10 && _locationGenerator.Size < 15)
{
    XCoordsOfWater = new int[_locationGenerator.Size *2];
    YCoordsOfWater = new int[_locationGenerator.Size *2];

    XCoordsOfFood = new int[_locationGenerator.Size *2];
    YCoordsOfFood = new int[_locationGenerator.Size *2];

    XCoordsOfPartsOfRadio = new int[_locationGenerator.Size * 2];
    YCoordsOfPartsOfRadio = new int[_locationGenerator.Size * 2];

    XCoordsOfGreenMobs = new int[_locationGenerator.Size * 2];
    YCoordsOfGreenMobs = new int[_locationGenerator.Size * 2];

    XCoordsOfRedMobs = new int[_locationGenerator.Size];
    YCoordsOfRedMobs = new int[_locationGenerator.Size];
}

if (_locationGenerator.Size >= 15)
{
    XCoordsOfWater = new int[_locationGenerator.Size * 3];
    YCoordsOfWater = new int[_locationGenerator.Size * 3];

    XCoordsOfFood = new int[_locationGenerator.Size * 3];
    YCoordsOfFood = new int[_locationGenerator.Size * 3];

    XCoordsOfPartsOfRadio = new int[_locationGenerator.Size * 3];
    YCoordsOfPartsOfRadio = new int[_locationGenerator.Size * 3];

    XCoordsOfGreenMobs = new int[_locationGenerator.Size * 3];
    YCoordsOfGreenMobs = new int[_locationGenerator.Size * 3];
```

```
            XCoordsOfRedMobs = new int[_locationGenerator.Size * 2];
            YCoordsOfRedMobs = new int[_locationGenerator.Size * 2];   }

    }

// Функция CreateRandomPlaces нужна для заполнения массивов координат. На вход функция
получает пустые массивы для заполнения.

    void CreateRandomPlaces(int[] XCoords, int[] YCoords)
    {
        for (int i = 0; i < XCoords.Length; i++)
        {
            var randomX = RandomCreateNumberForPlace();
            var randomY = RandomCreateNumberForPlace();

            bool check = true;

            while (check)
            {
                if (CheckForRepeat(randomX, randomY))
                {
                    randomX = RandomCreateNumberForPlace();
                    randomY = RandomCreateNumberForPlace();
                }
                else
                {
                    check = false;
                }
            }

            if (check == false)
            {
                XCoords[i] = randomX;
                YCoords[i] = randomY;
            }
        }
    }

// Фукнция RandomCreateNumberForPlace нужна для создания сгенерированного числа. На выход
она выдает сгенерированное число.

    int RandomCreateNumberForPlace()    {
        int answer = _locationGenerator.Rand.Next(0, _locationGenerator.Size);
        return answer;
    }

//Функция CheckForRepeat нужна для проверки на оригинальность пар координат. На вход функция
получает координаты. На выходе функция выдает значения True – если координаты оригинальны и
false – если координаты не оригинальны.

    bool CheckForRepeat(int Xcoord, int Ycoord)    {
        for (int i = 0; i < XCoordsOfWater.Length; i++)
        {
            if (XCoordsOfWater[i] == Xcoord)
            {
                if (YCoordsOfWater[i] == Ycoord)
                {
```

```
                    return true;
                }
            }

        }

        for (int i = 0; i < XCoordsOfFood.Length; i++)
        {
            if (XCoordsOfFood[i] == Xcoord)
            {
                if (YCoordsOfFood[i] == Ycoord)

                {
                    return true;
                }
            }
        }

        for (int i = 0; i < XCoordsOfPartsOfRadio.Length; i++)
        {
            if (XCoordsOfPartsOfRadio[i] == Xcoord)
            {
                if (YCoordsOfPartsOfRadio[i] == Ycoord)
                {
                    return true;
                }
            }
        }
        return false;
    }

       }


public class LocationGenerator : MonoBehaviour
{

    public GameObject LeftUpCorner;
    public GameObject RightUpCorner;
    public GameObject LeftDownCorner;
    public GameObject RightDownCorner;

    public GameObject BorderUp;
    public GameObject BorderDown;
    public GameObject BorderLeft;
    public GameObject BorderRight;

    public GameObject CenterBlock;


    public int Size;

    public float LongOfGrassBlock;
    public float HeightOfGrassBlock;

    public Random Rand;
```

```
// Функция Start нужна для инициализации данных.


    void Start ()    {
           LongOfGrassBlock = 19.2f;
           HeightOfGrassBlock = 10.8f;


           Rand = new System.Random();

           Size = PlayerPrefs.GetInt("SizeOfMap");

        LocationCreateEmpty();
       }

// Функция LocationCreateEmpty нужна для создания базового слоя мира.

    void LocationCreateEmpty()    {
        for (int i = 0; i < Size; i++)
        {
            for (int j = 0; j < Size; j++)
            {
                if (i == 0 && j == 0)
                {
                    Instantiate(LeftUpCorner, new Vector3(0, 0, 0), Quaternion.identity);
                }

                else
                {
                    if (i == 0 && j == Size - 1)
                    {
                        Instantiate(RightUpCorner, new Vector3(j * LongOfGrassBlock, 0, 0),
Quaternion.identity);
                    }
                    else
                    {
                        if (i == Size - 1 && j == 0)
                        {
                            Instantiate(LeftDownCorner, new Vector3(0, -(i *
HeightOfGrassBlock), 0), Quaternion.identity);
                        }
                        else
                        {
                            if (i == Size - 1 && j == Size - 1)
                            {
                                Instantiate(RightDownCorner, new Vector3(j *
LongOfGrassBlock, -(i * HeightOfGrassBlock), 0), Quaternion.identity);
                            }
                            else
                            {
                                if (j == 0 && i != 0 && i != Size - 1)
                                {
                                    Instantiate(BorderLeft, new Vector3(0, -(i *
HeightOfGrassBlock), 0), Quaternion.identity);
                                }
                                Else
```

```
                                {
                                    if (j != 0 && j != Size - 1 && i == 0)
                                    {
                                        Instantiate(BorderUp, new Vector3(j *
LongOfGrassBlock, 0, 0), Quaternion.identity);
                                    }
                                    else
                                    {
                                        if (j == Size - 1 && i != 0 && i != Size - 1)
                                        {
                                            Instantiate(BorderRight, new Vector3(j *
LongOfGrassBlock, -(i * HeightOfGrassBlock), 0), Quaternion.identity);
                                        }
                                        else
                                        {
                                            if (i == Size - 1 && j != 0 && j != Size - 1)
                                            {

                                                Instantiate(BorderDown, new Vector3(j *
LongOfGrassBlock, -(i * HeightOfGrassBlock), 0), Quaternion.identity);
                                            }
                                            else
                                            {
                                                Instantiate(CenterBlock, new Vector3(j *
LongOfGrassBlock, -(i * HeightOfGrassBlock), 0), Quaternion.identity);
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
public class FoodGeneration : MonoBehaviour
{
    private PlaceGeneration _placeGeneration;
    private LocationGenerator _locationGenerator;

    public int[] XCoordsOfFood;
    public int[] YCoordsOfFood;

    public GameObject FoodBlock;

    private float _sizeOfFoodBlock;

// Функция Start нужна для инициализации данных

    void Start()
    {
        _sizeOfFoodBlock = 1.3f;
        _placeGeneration =
GameObject.Find("GenerationManager").GetComponent<PlaceGeneration>();
```

```
        _locationGenerator =
GameObject.Find("GenerationManager").GetComponent<LocationGenerator>();
        XCoordsOfFood = new int[_placeGeneration.XCoordsOfFood.Length];
        YCoordsOfFood = new int[_placeGeneration.YCoordsOfFood.Length];

        for (int i = 0; i < _placeGeneration.XCoordsOfFood.Length; i++)
        {
            XCoordsOfFood[i] = _placeGeneration.XCoordsOfFood[i];
            YCoordsOfFood[i] = _placeGeneration.YCoordsOfFood[i];
        }

        Creation();
    }

// Функция Creation отвечает за создание кустов с едой.

    void Creation()
    {
        for (int i = 0; i < XCoordsOfFood.Length; i++)
        {
            Instantiate(FoodBlock,
                new Vector3(XCoordsOfFood[i] * _locationGenerator.LongOfGrassBlock,


                    -(YCoordsOfFood[i] * _locationGenerator.HeightOfGrassBlock), 0),
Quaternion.identity);

            if (_locationGenerator.Rand.Next(0,100) %2 == 0)
            {
                Instantiate(FoodBlock,
                    new Vector3(XCoordsOfFood[i] * _locationGenerator.LongOfGrassBlock -
_sizeOfFoodBlock,
                        -(YCoordsOfFood[i] * _locationGenerator.HeightOfGrassBlock) +
_sizeOfFoodBlock, 0), Quaternion.identity);
            }

            if (_locationGenerator.Rand.Next(0, 100) % 2 == 0)
            {
                Instantiate(FoodBlock,
                    new Vector3(XCoordsOfFood[i] * _locationGenerator.LongOfGrassBlock +
_sizeOfFoodBlock,
                        -(YCoordsOfFood[i] * _locationGenerator.HeightOfGrassBlock) -
_sizeOfFoodBlock, 0), Quaternion.identity);
            }


        }
    }


}
public class MobGeneration : MonoBehaviour
{
    private PlaceGeneration _placeGeneration;
    private LocationGenerator _locationGenerator;

    public int[] XCoordsOfRedMob;
```

```
    public int[] YCoordsOfRedMob;

    public int[] XCoordsOfGreenMob;
    public int[] YCoordsOfGreenMob;

    public GameObject GreenMob;
    public GameObject RedMob;


    void Start()// Функция для инициализации данных
    {
        _placeGeneration =
GameObject.Find("GenerationManager").GetComponent<PlaceGeneration>();
        _locationGenerator =
GameObject.Find("GenerationManager").GetComponent<LocationGenerator>();

        XCoordsOfGreenMob = new int[_placeGeneration.XCoordsOfGreenMobs.Length];
        YCoordsOfGreenMob = new int[_placeGeneration.YCoordsOfGreenMobs.Length];

        XCoordsOfRedMob = new int[_placeGeneration.XCoordsOfRedMobs.Length];
        YCoordsOfRedMob = new int[_placeGeneration.YCoordsOfRedMobs.Length];

        for (int i = 0; i < _placeGeneration.XCoordsOfGreenMobs.Length; i++)

        {
            XCoordsOfGreenMob[i] = _placeGeneration.XCoordsOfGreenMobs[i];
            YCoordsOfGreenMob[i] = _placeGeneration.YCoordsOfGreenMobs[i];
        }

        for (int i = 0; i < _placeGeneration.XCoordsOfRedMobs.Length; i++)
        {
            XCoordsOfRedMob[i] = _placeGeneration.XCoordsOfRedMobs[i];
            YCoordsOfRedMob[i] = _placeGeneration.YCoordsOfRedMobs[i];
        }

        Creation();
    }
// Функция Creation отвечает за создание NPC

    void Creation()
    {
        for (int i = 0; i < _placeGeneration.XCoordsOfGreenMobs.Length; i++)
        {
            Instantiate(GreenMob,
                new Vector3(XCoordsOfGreenMob[i] * _locationGenerator.LongOfGrassBlock,
                    -(YCoordsOfGreenMob[i] * _locationGenerator.HeightOfGrassBlock), 0),
Quaternion.identity);
        }

        for (int i = 0; i < _placeGeneration.XCoordsOfRedMobs.Length; i++)
        {
            Instantiate(RedMob,
                new Vector3(XCoordsOfRedMob[i] * _locationGenerator.LongOfGrassBlock,
                    -(YCoordsOfRedMob[i] * _locationGenerator.HeightOfGrassBlock), 0),
Quaternion.identity);
        }
}
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ИСКУССТВЕННОГО ИНТЕЛЛЕКТА NPC

Листинг 2 - GreenMobBehaviour

```
public class GreenMobMovement : MonoBehaviour
{
    private LocationGenerator _locationGenerator;
    private Vector3 _target;
    private Seeker _seeker;

    public GameObject GreenMobPrefab;

    private Vector3 _goingTarget;

    private GameObject[] _greenMobs;
    private GameObject[] _waters;
    private GameObject[] _foods;

    private WaterNeeds _waterNeeds;
    private FoodNeeds _foodNeeds;

    private TimeManager _timeManager;
    public string NameOfTimeManager = "LocationManager";

    private float _timeToBreeding;

    private Vector3 _changePlace;
    private float _timeToStuck;

    private enum WayPoint
    {
        Food,
        Water,
        Going,
        Choose,
        Walking,
        Breeding
    }

    private WayPoint _way;

// Функция Start нужна для инициализации данных
    void Start()
    {
        _locationGenerator                                                                    =
GameObject.Find("GenerationManager").GetComponent<LocationGenerator>();
        _seeker = GetComponent<Seeker>();

        _timeManager = GameObject.Find(NameOfTimeManager).GetComponent<TimeManager>();

        _timeToBreeding = 15f;

        _waters = GameObject.FindGameObjectsWithTag("Water");
        _foods = GameObject.FindGameObjectsWithTag("Food");
        _greenMobs = GameObject.FindGameObjectsWithTag("GreenMob");
```

```
        for (int i = 0; i < _greenMobs.Length; i++)
        {
            if (_greenMobs[i].name == gameObject.name)
            {
                _greenMobs[i] = null;
                _greenMobs = _greenMobs.Where(x => x != null).ToArray();
            }
        }

        _waterNeeds = GetComponent<WaterNeeds>();
        _foodNeeds = GetComponent<FoodNeeds>();

        _timeToStuck = 0f;
        _changePlace = new Vector3(0f,0f,0f);

        _way = WayPoint.Choose;
    }
//Функция FixedUpdate вызывается каждый кадр
    void FixedUpdate()    {
        ChangeFoodSpending();

        DoNotStuckMob();

        Way();
    }

    void FindMobs() // Поиск NPC на карте
    {
        _greenMobs = GameObject.FindGameObjectsWithTag("GreenMob");

        for (int i = 0; i < _greenMobs.Length; i++)
        {
            if (_greenMobs[i].name == gameObject.name)
            {
                _greenMobs[i] = null;
                _greenMobs = _greenMobs.Where(x => x != null).ToArray();
            }
        }
    }
// Функция ChangeFoodSpending отвечает за смену метаболизма от времени года
    void ChangeFoodSpending()
    {
        if (_timeManager.NowSeason == TimeManager.Season.Spring)
        {
            switch (_timeManager.NameOfTime)
            {
                case TimeManager.TimeName.Morning:
                    _foodNeeds.FoodSendInSecond = 0.035f;
                    break;
                case TimeManager.TimeName.Day:
                    _foodNeeds.FoodSendInSecond = 0.0265f;
                    break;
                case TimeManager.TimeName.Evening:
                    _foodNeeds.FoodSendInSecond = 0.0165f;
                    break;
                case TimeManager.TimeName.Night:
```

```
                _foodNeeds.FoodSendInSecond = 0.015f;
                _waterNeeds.WaterSendInSecond = 0.002f;
                break;
        }
    }

    if (_timeManager.NowSeason == TimeManager.Season.Summer)
    {
        switch (_timeManager.NameOfTime)
        {
            case TimeManager.TimeName.Morning:
                _foodNeeds.FoodSendInSecond = 0.03f;
                break;
            case TimeManager.TimeName.Day:
                _foodNeeds.FoodSendInSecond = 0.026f;
                break;
            case TimeManager.TimeName.Evening:
                _foodNeeds.FoodSendInSecond = 0.016f;
                break;
            case TimeManager.TimeName.Night:
                _foodNeeds.FoodSendInSecond = 0.01f;
                _waterNeeds.WaterSendInSecond = 0.002f;
                break;
        }
    }

    if (_timeManager.NowSeason == TimeManager.Season.Autumn)
    {
        switch (_timeManager.NameOfTime)
        {
            case TimeManager.TimeName.Morning:
                _foodNeeds.FoodSendInSecond = 0.02f;
                break;
            case TimeManager.TimeName.Day:
                _foodNeeds.FoodSendInSecond = 0.013f;
                break;
            case TimeManager.TimeName.Evening:
                _foodNeeds.FoodSendInSecond = 0.008f;
                break;
            case TimeManager.TimeName.Night:
                _foodNeeds.FoodSendInSecond = 0.005f;
                _waterNeeds.WaterSendInSecond = 0.002f;
                break;
        }
    }

    if (_timeManager.NowSeason == TimeManager.Season.Winter)
    {
        _foodNeeds.FoodSendInSecond = 0.003f;
        _waterNeeds.WaterSendInSecond = 0.003f;
    }

}
// Функция way отвечает за выбор действий взависимости от времени года
    void Way()    {
        if (_timeManager.NowSeason == TimeManager.Season.Spring)
        {
```

```
            FindMobs();
            SpringBehaviour();
        }

        if (_timeManager.NowSeason == TimeManager.Season.Summer)
        {
            FindMobs();
            SummerBehaviour();
        }

        if (_timeManager.NowSeason == TimeManager.Season.Autumn)
        {
            FindMobs();
            AutumnBehaviour();
        }

        if (_timeManager.NowSeason == TimeManager.Season.Winter)
        {
            WinterBehaviour();
        }

    }

    void DoNotStuckMob() // Защита от застревания NPC
    {
        if ((_timeManager.NowSeason != TimeManager.Season.Winter &&
            _timeManager.NameOfTime         !=         TimeManager.TimeName.Night)         ||
(_timeManager.NowSeason   ==   TimeManager.Season.Autumn   &&   _timeManager.NameOfTime   ==
TimeManager.TimeName.Night))
        {
            if (_changePlace == gameObject.transform.position)
            {
                _timeToStuck += Time.deltaTime;
            }
            else
            {
                _changePlace = gameObject.transform.position;
                _timeToStuck = 0f;
            }

            if (_timeToStuck >= 2f)
            {
                _way = WayPoint.Walking;
            }
        }
    }
//Функция SpringBehaviour отвечает за поведение NPC весной
    void SpringBehaviour()
    {
        if (_greenMobs.Length >= _locationGenerator.Size * 1.5)
        {
            _timeToBreeding = 20f;
        }

        if (_timeToBreeding > 0f)
        {
            _timeToBreeding = _timeToBreeding - Time.deltaTime;
```

```
        }
        else
        {
            _timeToBreeding = 0;
        }

        if (_way == WayPoint.Going)
        {
            if (_timeToBreeding <= 0f)
            {
                foreach (var t in _greenMobs)
                {
                    if (Math.Abs(t.transform.position.x - gameObject.transform.position.x) <
4f
                        &&                  Math.Abs(t.transform.position.y                 -
gameObject.transform.position.y) < 4f)
                    {
                        if (_foodNeeds.ReturnFood() > 50 && _waterNeeds.ReturnWater() > 50)
                        {
                            _way = WayPoint.Breeding;
                        }
                    }
                }
            }


            if (gameObject.transform.position.x > _target.x - 0.5f &&
                gameObject.transform.position.x <= _target.x + 0.5f &&
                gameObject.transform.position.y > _target.y - 0.5f &&
                gameObject.transform.position.y <= _target.y + 2.7f)
            {
                _way = WayPoint.Choose;
            }
        }

        if (_way == WayPoint.Choose)
        {
            if (_timeManager.NameOfTime == TimeManager.TimeName.Night)
            {
                _seeker.CancelCurrentPathRequest();
            }
            else
            {
                if (_waterNeeds.ReturnWater() < _foodNeeds.ReturnFood())
                {
                    _way    =    _waterNeeds.ReturnWater()    <    85    ?    WayPoint.Water    :
WayPoint.Walking;
                }
                else
                {
                    _way = _foodNeeds.ReturnFood() < 85 ? WayPoint.Food : WayPoint.Walking;
                }
            }
        }

        if (_way == WayPoint.Breeding)
        {
```

```
            _timeToBreeding = 15f;
            Instantiate(GreenMobPrefab,
                new          Vector3(gameObject.transform.position.x          +          0.3f,
gameObject.transform.position.y + 0.3f,
                    gameObject.transform.position.z), Quaternion.identity);
            _way = WayPoint.Going;
        }

        if (_way == WayPoint.Food)
        {
            int number = 0;
            var      distance      =      (Math.Abs(_foods[number].transform.position.x      -
gameObject.transform.position.x -
                                      _locationGenerator.LongOfGrassBlock / 2) +
                              Math.Abs(_foods[number].transform.position.y                -
gameObject.transform.position.y +
                                      _locationGenerator.LongOfGrassBlock / 2));

            for (int i = 0; i < _foods.Length; i++)
            {
                var need = _foods[i].GetComponent<FoodHave>();
                if (need.ReturnFood() > _foodNeeds.ReturnFoodNeeds())
                {
                    if              ((Math.Abs(_foods[i].transform.position.x              -
gameObject.transform.position.x -
                                _locationGenerator.LongOfGrassBlock / 2) +
                         Math.Abs(_foods[i].transform.position.y                          -
gameObject.transform.position.y +
                                _locationGenerator.HeightOfGrassBlock / 2)) < distance)
                    {
                        number = i;

                        distance     =     (Math.Abs(_foods[number].transform.position.x     -
gameObject.transform.position.x -
                                           _locationGenerator.LongOfGrassBlock / 2) +
                                    Math.Abs(_foods[number].transform.position.y           -
gameObject.transform.position.y +
                                           _locationGenerator.LongOfGrassBlock / 2));
                    }
                }
            }

            _target = _foods[number].transform.position;
            _seeker.StartPath(transform.position,  _target  +  new  Vector3(0f,  1.7f,  0),
OnPathComplete);
            _way = WayPoint.Going;
        }

        if (_way == WayPoint.Water)
        {
            int number = 0;

            var      distance      =      (Math.Abs(_waters[number].transform.position.x      -
gameObject.transform.position.x -
                                      _locationGenerator.LongOfGrassBlock / 2) +
                              Math.Abs(_waters[number].transform.position.y                -
gameObject.transform.position.y +
```

```
                                    _locationGenerator.LongOfGrassBlock / 2));

            for (int i = 0; i < _waters.Length; i++)
            {
                var need = _waters[i].GetComponent<WaterHave>();
                if (need.ReturnWater() > _waterNeeds.ReturnWaterNeed())
                {
                    if              ((Math.Abs(_foods[i].transform.position.x              -
gameObject.transform.position.x -
                                _locationGenerator.LongOfGrassBlock / 2) +
                         Math.Abs(_foods[i].transform.position.y                          -
gameObject.transform.position.y +
                                _locationGenerator.HeightOfGrassBlock / 2)) < distance)
                    {
                        number = i;
                        distance    =    (Math.Abs(_waters[number].transform.position.x    -
gameObject.transform.position.x -
                                            _locationGenerator.LongOfGrassBlock / 2) +
                                       Math.Abs(_waters[number].transform.position.y          -
gameObject.transform.position.y +
                                            _locationGenerator.LongOfGrassBlock / 2));
                    }
                }
            }

            _target = _waters[number].transform.position;
            _seeker.StartPath(transform.position,  _target  +  new  Vector3(0f,  2.5f,  0),
OnPathComplete);

            _way = WayPoint.Going;
        }

        if (_way == WayPoint.Walking)
        {
            _goingTarget = new Vector3(
                _locationGenerator.Size *
                _locationGenerator.Rand.Next(0, (int)_locationGenerator.LongOfGrassBlock * 9
/ 10),
                -_locationGenerator.Size *
                _locationGenerator.Rand.Next(0, (int)_locationGenerator.HeightOfGrassBlock /
2), 0f);
            _target = _goingTarget;

            _seeker.StartPath(transform.position, _target, OnPathComplete);
            _way = WayPoint.Going;
        }
    }
// Функция SummerBehaviour отвечает за поведение NPC летом
    void SummerBehaviour()
    {
        if (_greenMobs.Length >= _locationGenerator.Size * 5)
        {
            _timeToBreeding = 5f;
        }

        if (_timeToBreeding > 0f)
        {
```

```
            _timeToBreeding = _timeToBreeding - Time.deltaTime;
        }
        else
        {
            _timeToBreeding = 0;
        }

        if (_way == WayPoint.Going)
        {
            if (_timeToBreeding <= 0f)
            {
                foreach (var t in _greenMobs)
                {
                    if (Math.Abs(t.transform.position.x - gameObject.transform.position.x) <
4f
                        &&                  Math.Abs(t.transform.position.y                 -
gameObject.transform.position.y) < 4f)
                    {
                        if (_foodNeeds.ReturnFood() > 50 && _waterNeeds.ReturnWater() > 50)
                        {
                            _way = WayPoint.Breeding;
                        }
                    }
                }
            }


            if (gameObject.transform.position.x > _target.x - 0.5f &&
                gameObject.transform.position.x <= _target.x + 0.5f &&
                gameObject.transform.position.y > _target.y - 0.5f &&
                gameObject.transform.position.y <= _target.y + 2.7f)
            {
                _way = WayPoint.Choose;
            }
        }

        if (_way == WayPoint.Choose)
        {
            if (_timeManager.NameOfTime == TimeManager.TimeName.Night)
            {
                bool isRedMobNear = false;
                GameObject[] _redMobs = GameObject.FindGameObjectsWithTag("RedMob"); ;
                foreach (var t in _redMobs)
                {
                    if (gameObject.transform.position.x > t.transform.position.x - 0.5f &&
                        gameObject.transform.position.x <= t.transform.position.x + 0.5f &&
                        gameObject.transform.position.y > t.transform.position.y - 0.5f &&
                        gameObject.transform.position.y <= t.transform.position.y + 2.7f)
                    {
                        isRedMobNear = true;
                    }

                    if (isRedMobNear)
                    {
                        _way = WayPoint.Walking;
                    }
                }
```

```
            }
            else
            {
                if (_waterNeeds.ReturnWater() < _foodNeeds.ReturnFood())
                {
                    _way    =    _waterNeeds.ReturnWater()    <    85    ?    WayPoint.Water    :
WayPoint.Walking;
                }
                else
                {
                    _way = _foodNeeds.ReturnFood() < 85 ? WayPoint.Food : WayPoint.Walking;
                }
            }
        }

        if (_way == WayPoint.Breeding)
        {
            _timeToBreeding = 7f;
            Instantiate(GreenMobPrefab,
                new          Vector3(gameObject.transform.position.x          +          0.3f,
gameObject.transform.position.y + 0.3f,
                    gameObject.transform.position.z), Quaternion.identity);
            _way = WayPoint.Going;
        }

        if (_way == WayPoint.Food)
        {
            int number = 0;
            var      distance      =      (Math.Abs(_foods[number].transform.position.x      -
gameObject.transform.position.x -
                                      _locationGenerator.LongOfGrassBlock / 2) +
                                  Math.Abs(_foods[number].transform.position.y                -
gameObject.transform.position.y +
                                      _locationGenerator.LongOfGrassBlock / 2));

            for (int i = 0; i < _foods.Length; i++)
            {
                var need = _foods[i].GetComponent<FoodHave>();
                if (need.ReturnFood() > _foodNeeds.ReturnFoodNeeds())
                {
                    if              ((Math.Abs(_foods[i].transform.position.x              -
gameObject.transform.position.x -
                                _locationGenerator.LongOfGrassBlock / 2) +
                            Math.Abs(_foods[i].transform.position.y                            -
gameObject.transform.position.y +
                                _locationGenerator.HeightOfGrassBlock / 2)) < distance)
                    {
                        number = i;

                        distance     =     (Math.Abs(_foods[number].transform.position.x     -
gameObject.transform.position.x -
                                           _locationGenerator.LongOfGrassBlock / 2) +
                                       Math.Abs(_foods[number].transform.position.y           -
gameObject.transform.position.y +
                                           _locationGenerator.LongOfGrassBlock / 2));
                    }
                }
```

```
            }

            _target = _foods[number].transform.position;
            _seeker.StartPath(transform.position,  _target  +  new  Vector3(0f,  1.7f,  0),
OnPathComplete);
            _way = WayPoint.Going;
        }

        if (_way == WayPoint.Water)
        {
            int number = 0;

            var     distance     =     (Math.Abs(_waters[number].transform.position.x     -
gameObject.transform.position.x -
                                       _locationGenerator.LongOfGrassBlock / 2) +
                                Math.Abs(_waters[number].transform.position.y                -
gameObject.transform.position.y +
                                         _locationGenerator.LongOfGrassBlock / 2));

            for (int i = 0; i < _waters.Length; i++)
            {
                var need = _waters[i].GetComponent<WaterHave>();
                if (need.ReturnWater() > _waterNeeds.ReturnWaterNeed())
                {
                    if              ((Math.Abs(_foods[i].transform.position.x              -
gameObject.transform.position.x -
                                    _locationGenerator.LongOfGrassBlock / 2) +
                                Math.Abs(_foods[i].transform.position.y                           -
gameObject.transform.position.y +
                                     _locationGenerator.HeightOfGrassBlock / 2)) < distance)
                    {
                        number = i;
                        distance    =    (Math.Abs(_waters[number].transform.position.x    -
gameObject.transform.position.x -
                                              _locationGenerator.LongOfGrassBlock / 2) +
                                           Math.Abs(_waters[number].transform.position.y          -
gameObject.transform.position.y +
                                              _locationGenerator.LongOfGrassBlock / 2));
                    }
                }
            }

            _target = _waters[number].transform.position;
            _seeker.StartPath(transform.position,  _target  +  new  Vector3(0f,  2.5f,  0),
OnPathComplete);

            _way = WayPoint.Going;
        }

        if (_way == WayPoint.Walking)
        {
            _goingTarget = new Vector3(
                _locationGenerator.Size *
                _locationGenerator.Rand.Next(0, (int)_locationGenerator.LongOfGrassBlock * 9
/ 10),
                -_locationGenerator.Size *
```

```
                _locationGenerator.Rand.Next(0, (int)_locationGenerator.HeightOfGrassBlock /
2), 0f);
            _target = _goingTarget;

            _seeker.StartPath(transform.position, _target, OnPathComplete);
            _way = WayPoint.Going;
        }
    }
// Функция AutumnBehaviour отвечает за поведение NPC осенью
    void AutumnBehaviour()
    {
        if (_greenMobs.Length >= _locationGenerator.Size * 1.5)
        {
            _timeToBreeding = 20f;
        }

        if (_timeToBreeding > 0f)
        {
            _timeToBreeding = _timeToBreeding - Time.deltaTime;
        }
        else
        {
            _timeToBreeding = 0;
        }

        if (_way == WayPoint.Going)
        {
            if (_timeToBreeding <= 0f)
            {
                foreach (var t in _greenMobs)
                {
                    if (Math.Abs(t.transform.position.x - gameObject.transform.position.x) <
4f
                        &&                  Math.Abs(t.transform.position.y                  -
gameObject.transform.position.y) < 4f)
                    {
                        if (_foodNeeds.ReturnFood() > 50 && _waterNeeds.ReturnWater() > 50)
                        {
                            _way = WayPoint.Breeding;
                        }
                    }
                }
            }


            if (gameObject.transform.position.x > _target.x - 0.5f &&
                gameObject.transform.position.x <= _target.x + 0.5f &&
                gameObject.transform.position.y > _target.y - 0.5f &&
                gameObject.transform.position.y <= _target.y + 2.7f)
            {
                _way = WayPoint.Choose;
            }
        }

        if (_way == WayPoint.Choose)
        {
            if (_timeManager.NameOfTime == TimeManager.TimeName.Night)
```

```
            {
                _seeker.CancelCurrentPathRequest();
                _timeToBreeding = 20f;
            }
            else
            {
                if (_waterNeeds.ReturnWater() < _foodNeeds.ReturnFood())
                {
                    _way    =    _waterNeeds.ReturnWater()    <    85    ?    WayPoint.Water    :
WayPoint.Walking;
                }
                else
                {
                    _way = _foodNeeds.ReturnFood() < 85 ? WayPoint.Food : WayPoint.Walking;
                }
            }
        }

        if (_way == WayPoint.Breeding)
        {
            _timeToBreeding = 15f;
            Instantiate(GreenMobPrefab,
                new          Vector3(gameObject.transform.position.x          +          0.3f,
gameObject.transform.position.y + 0.3f,
                    gameObject.transform.position.z), Quaternion.identity);
            _way = WayPoint.Going;
        }

        if (_way == WayPoint.Food)
        {
            int number = 0;
            var      distance      =      (Math.Abs(_foods[number].transform.position.x      -
gameObject.transform.position.x -
                                      _locationGenerator.LongOfGrassBlock / 2) +
                                Math.Abs(_foods[number].transform.position.y                    -
gameObject.transform.position.y +
                                      _locationGenerator.LongOfGrassBlock / 2));

            for (int i = 0; i < _foods.Length; i++)
            {
                var need = _foods[i].GetComponent<FoodHave>();
                if (need.ReturnFood() > _foodNeeds.ReturnFoodNeeds())
                {
                    if              ((Math.Abs(_foods[i].transform.position.x              -
gameObject.transform.position.x -
                                  _locationGenerator.LongOfGrassBlock / 2) +
                             Math.Abs(_foods[i].transform.position.y                           -
gameObject.transform.position.y +
                                  _locationGenerator.HeightOfGrassBlock / 2)) < distance)
                    {
                        number = i;

                        distance     =     (Math.Abs(_foods[number].transform.position.x     -
gameObject.transform.position.x -
                                             _locationGenerator.LongOfGrassBlock / 2) +
                                       Math.Abs(_foods[number].transform.position.y           -
gameObject.transform.position.y +
```

```
                                        _locationGenerator.LongOfGrassBlock / 2));
                }
            }
        }

        _target = _foods[number].transform.position;
        _seeker.StartPath(transform.position, _target + new Vector3(0f, 1.7f, 0),
OnPathComplete);
        _way = WayPoint.Going;
    }

    if (_way == WayPoint.Water)
    {
        int number = 0;

        var distance = (Math.Abs(_waters[number].transform.position.x -
gameObject.transform.position.x -
                                _locationGenerator.LongOfGrassBlock / 2) +
                        Math.Abs(_waters[number].transform.position.y -
gameObject.transform.position.y +
                                _locationGenerator.LongOfGrassBlock / 2));

        for (int i = 0; i < _waters.Length; i++)
        {
            var need = _waters[i].GetComponent<WaterHave>();
            if (need.ReturnWater() > _waterNeeds.ReturnWaterNeed())
            {
                if ((Math.Abs(_foods[i].transform.position.x -
gameObject.transform.position.x -
                            _locationGenerator.LongOfGrassBlock / 2) +
                     Math.Abs(_foods[i].transform.position.y -
gameObject.transform.position.y +
                            _locationGenerator.HeightOfGrassBlock / 2)) < distance)
                {
                    number = i;
                    distance = (Math.Abs(_waters[number].transform.position.x -
gameObject.transform.position.x -
                                    _locationGenerator.LongOfGrassBlock / 2) +
                                Math.Abs(_waters[number].transform.position.y -
gameObject.transform.position.y +
                                    _locationGenerator.LongOfGrassBlock / 2));
                }
            }
        }

        _target = _waters[number].transform.position;
        _seeker.StartPath(transform.position, _target + new Vector3(0f, 2.5f, 0),
OnPathComplete);

        _way = WayPoint.Going;
    }

    if (_way == WayPoint.Walking)
    {
        _goingTarget = new Vector3(
            _locationGenerator.Size *
```

```
                _locationGenerator.Rand.Next(0, (int)_locationGenerator.LongOfGrassBlock * 9
/ 10),
                -_locationGenerator.Size *
                _locationGenerator.Rand.Next(0, (int)_locationGenerator.HeightOfGrassBlock /
2), 0f);
            _target = _goingTarget;

            _seeker.StartPath(transform.position, _target, OnPathComplete);
            _way = WayPoint.Going;
        }
    }
// Функция WinterBehaviour отвечает за поведение NPC зимой
    void WinterBehaviour()
    {
        if (_way == WayPoint.Choose)
        {
            if (gameObject.transform.position.x > _target.x - 0.5f &&
                gameObject.transform.position.x <= _target.x + 0.5f &&
                gameObject.transform.position.y > _target.y - 0.5f &&
                gameObject.transform.position.y <= _target.y + 2.7f)
            {
                _seeker.CancelCurrentPathRequest();
            }
        }

        if (_way == WayPoint.Going)
        {
            WinterPath();
        }

        if (_way == WayPoint.Breeding)
        {
            WinterPath();
        }

        if (_way == WayPoint.Food)
        {
            WinterPath();
        }

        if (_way == WayPoint.Water)
        {
            WinterPath();
        }

        if (_way == WayPoint.Walking)
        {
            WinterPath();
        }
    }

    void WinterPath() // Сбор NPC на зимнюю спячку
    {
        _greenMobs = GameObject.FindGameObjectsWithTag("GreenMob");
        int numberOfGreens = 0;
        for (int i = 0; i < _greenMobs.Length; i++)         {
            if (_greenMobs[i].name == gameObject.name)
```

```
            {
                numberOfGreens = i;
            }
        }

        _target = new Vector3(0 + numberOfGreens, 0, 0);

        _seeker.StartPath(transform.position, _target, OnPathComplete);

        _way = WayPoint.Choose;
    }

    public void OnPathComplete(Path p)
    {
        if (p.error)
        {
        }
        else
        {
        }
    }
}
```