Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
_____ О.В. Непомнящий
подпись инициалы, фамилия
«_____» _____2019 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника
код и наименование направления

Стратегическая игра с применением адаптивного искусственного интеллекта
тема

Руководитель	_____ подпись, дата	ст. преподаватель должность, ученая степень	И.В.Матковский инициалы, фамилия
Выпускник	_____ подпись, дата		С.А.Полонников инициалы, фамилия
Консультант	_____ подпись, дата	доцент, канд.тех.наук. должность, ученая степень	Л.И.Покидишева инициалы, фамилия
Нормоконтролер	_____ подпись, дата	доцент, канд.тех.наук. должность, ученая степень	В. И. Иванов инициалы, фамилия

Красноярск 2019

СОДЕРЖАНИЕ

Введение 3
1 Анализ предметной области 4
1.1 Техническое задание 4
1.2 Обзор аналогичных проектов 5
1.2.1 Искусственный интеллект 7
1.3 Выбор технологии разработки 7
1.3.1 Unreal Engine 4 10
1.3.2 CryEngine 5 12
1.3.3 Unity 13
1.3.4 Source 2 15
1.4 Итоги анализа 16
2 Этап проектирования 17
2.1 Описание основных механик 18
2.2 Описание общей структуры 19
2.3 Генерация поля боя 21
2.4 Алгоритмы для искусственного интеллекта 22
2.5 Итоги проектирования 23
3 Реализацция проекта 23
3.1 Основные данные игрока 23
3.2 Главное меню и вспомогательные менеджеры 26
3.3 Создание новой игры 27
3.4 Главная сцена 29
3.5 Боевая сцена 30
3.6 Искусственный интеллект 35
3.7 Система инвентаря 40
Заключение 43
Список использованных источников 44

ВВЕДЕНИЕ

За несколько последних десятилетий отрасль компьютерных игр проделала гигантский путь. Все началось с простейших игр, таких как *Pong* и *Pac–Man*, благодаря которым игроки могли на короткое время забыть о реальном мире.

В данное время, имеет место массовое распространение онлайн сервисов по продаже и продвижению проектов, таких как Steam, GoG, EGS, а также интернет–площадки сообществ, как например, DTF, поддерживающих независимых разработчиков и помогающих в продвижении. Поэтому на сегодняшний день каждый имеет возможность создать собственный проект и найти аудиторию.

По данным Ассоциации развлекательного программного обеспечения (ESA), современные геймеры обладают в среднем 13–летним опытом компьютерных игр, они привыкли к тому, что новые игры становятся все более сложными, увлекательными и умными [1].

Для разработчиков основная проблема состоит в том, что необходимо создавать все более захватывающие игры – игрокам интереснее играть в игры с все более интересными и глубокими механиками. Для решения этой задачи применяется и постоянно совершенствующийся управляемый компьютером искусственный интеллект (ИИ). В истории создания ИИ для игр уже выделено множество подходов в его реализации, практически для каждого из жанров существует собственное решение. Порой разработчики пытаются даже поэкспериментировать и создать ИИ, который действительно моделировал бы работу человеческого мозга, обычно с применением нейросетей, но таких примеров пока что единицы. В игровой индустрии быстро пришло понимание, что ни смотря на то, что за нейросетями и машинным обучением будущее, применять их при создании ИИ для игр крайне сложно. Вся причина в том, что обучить машину необходимо не тому, как выигрывать человека, а как проигрывать, чтобы игроку было интересно [2].

Создание хорошего искусственного игрового партнера, который способен приспосабливаться к действиям игрока, играть на высоком уровне и побуждать игрока совершенствоваться – весьма непростая задача. Ведь хороший ИИ позволяет игроку обманывать себя, он является предсказуемым. Хороший ИИ может взаимодействовать с игровыми системами, а также имеет другие цели, помимо банального уничтожения игрока.

ИИ в играх подразумевает не только управление определенных единичных игровых объектов, а также координации между ними, ориентация на карте и управлением развития игры в целом [2].

В своей работе я также собираюсь реализовать адаптивный ИИ, способный ориентироваться на процедурно сгенерированной карте чтобы предоставить игроку ситуации, которые будут создавать интерес в игре. Темой моей выпускной квалификационной работы стала разработка собственной игры с адаптивным ИИ.

1 Анализ предметной области

1.1 Техническое задание

В процессе выполнения работы должен быть проведен анализ аналогичных игровых проектов, таких как FTL: Faster Than Light, The Binding of Isaac и других игр в жанре rogue–like и реализован собственный проект с 2D графикой с Pixel Art стилизацией в окружении научной фантастики. Игра предназначена для одиночного прохождения, где юнитами противника управляет компьютер. Также необходимо реализовать систему для генерации различных локаций и ИИ, который сможет ориентироваться на них. Проект разрабатывается под операционную систему Windows.

Основные задачи в выполнении данной выпускной квалификационной работы:

- анализ аналогичных проектов;
- реализация различных игровых сцен с процедурной генерацией;

- реализация основных игровых механик;
- реализация ИИ противника и менеджеры для управления сценами;
- реализация сохранения;

1.2 Обзор аналогичных проектов

Поскольку разрабатанный проект представляет собой игру в жанре rogue-like, то необходимо рассмотреть представителей данного жанра и его ключевые особенности.

Свое название жанр получил от игры-родоначальницы – Rogue. Rogue–like, собственно, и переводится как подобные Rogue. Roguelike считается под-жанром компьютерных ролевых игр, Rogue в свое время не просто создал его, но и заложил основы механики всего жанра в целом [3].

Главные особенности жанра rogue–like:

- случайная генерация контента – мир и его составляющие в основном генерируются заново с каждой новой игрой;
- невосстанавливаемый проигрыш – ошибки игрока являются необратимыми, и если это приводит к проигрышу, то игру приходится начинать с самого начала;
- пошаговое управление – игрок делает один шаг, окружающий мир тоже проживает время одного шага, это позволяет делать в игре паузы любой продолжительности;
- общее управление – игроку должны быть доступны все команды в любой игровой ситуации, например, игрок может начать точить свой меч даже тогда, когда его начинает избивать какой-нибудь гоблин;
- открытый мир – продвижение в игре не должно быть строго линейным, у игрока должна быть возможность выбирать что и когда делать, это относится далеко не только к игровой карте;
- открытие игровых механик – игроку приходится самостоятельно изучать то, как работает этот мир и предметы в нем, а большое число контента делает это занятие более интересным;

Главным идейным вдохновителем для разрабатываемого проекта является компьютерная игра FTL: Faster Than Light, вышедшая в релиз 15 сентября 2012 и ставшая игрой, которая смогла привлечь широкое внимание индустрии к независимым от издательств играм, известных так же инди-игры [4].

В FTL игроку предстоит взять на себя командование космическим кораблем и членов экипажа, которые управляют его оборудованием и функциями, будь то стрельба, маневрирование, управление системами защиты и прочее (). Перед игроком предстоит пройти множество локаций, которые представляют собой сюжетное путешествие, каждый шаг которого станет уникальной задачей с несколькими вариантами решения.

Рисунок 1 – игра FTL: Faster Than Light.

В Faster Than Light очень многое полагается на случай. А в играх данного жанра этот фактор очень часто оказывается критическим. При старте у игрока находится определенное количество ресурсов, будь то деньги, топливо или экипаж, но главным является прочность корабля, при потере которого и происходит поражение. На пути к конечной цели игроку предстоит множество раз сражаться с вражескими кораблями, бой с которыми и представляет основной

геймплей. Цель в этом событии заключается в уничтожении вражеского корабля, либо же выхода из боя, чтобы не быть уничтоженным.

Игра состоит из 8 уровней, которые, в свою очередь, состоят из процедурно сгенерированной карты, на которой случайным образом расположены игровые события. При активации этих событий возможны совершенно разные исходы одинаковых, от прохождения к прохождению, диалогов, сражений и случайных явлений. Такой подход, вкупе с частым началом новой игры, обеспечивает высокую реиграбельность [6].

1.2.1 Искусственный интеллект

Поскольку основной геймплей представляет собой боевую сцену, то искусственный интеллект противника выражается в поведении корабля противника. Поскольку в жанре rogue–like все строится на случайности, то основным противником для игрока она и является – при входе в боевое событие игрок встречает один из нескольких типов противников, который снаряжен определенным оборудованием: ракеты, лазеры, бомбы и прочее.

Искусственный интеллект противника остается неизменным из раза в раз – он старается уничтожить корабль игрока, стреляя в один из модулей корабля, выбираемого случайным образом, но при любом попадании в корабль игроку предстоит решать последствия попадания, будь то пожар, потеря кислорода и прочее, переводя членов экипажа со своих постов в другие места и теряя тем самым какие-то возможности, как например, потеря бонуса для щитов, уклонения или шанс выйти из боя.

1.3 Выбор технологии разработки

Несмотря на то, что до сих пор некоторые крупные студии разрабатывают внутренние игровые движки, остается огромный выбор среди общедоступных, которые не уступают внутренним разработкам. Внутренние игровые движки –

игровые движки, разработчик которых использует их только в своих, внутренних проектах и не продает движок сторонним юридическим лицам. Внутренние игровые движки относятся к проприетарному программному обеспечению [5].

Внутренние движки имеют место быть из–за того, что движки по лицензии зачастую требуют отчисления от продаж, которые у крупных издательств составляют настолько будут составлять настолько большие суммы, что лучшим решением становится разработка собственного ПО. Также из причин можно выделить внедрение уникальных технологий, которые можно использовать для рекламы игр, как например, технологии, поддерживаемые движком Frostbite, которого помогали в продвижении игр от Electronic Arts.

Немаловажным фактом является то, что не внутренние движки не могут гарантировать сохранности коммерческой тайны, которой и является игра, ведь движки по лицензии зачастую собирают всю информацию об разработке для анализа и улучшения собственных решений.

Для независимых же разработчиков и малых команд разработка собственного ПО для работы является нецелесообразным решением, ведь это займет большую часть времени всей разработки. Предлагаемые разработчикам готовые игровые движки уже имеют все ключевые элементы для быстрой разработки.

Многие студии начали специализироваться не на выпуске собственных игр, а на разработке необходимого функционала для их создания, зарабатывая на лицензировании при их использовании.

В настоящее время популярны следующие игровые движки, не являющиеся внутренними:

- Unreal Engine 4.
- CryENGINE 5.
- Unity.
- Source 2.

В списке представлены движки, которые используются студиями с различными направлениями проектов.

Рассмотрим представленные движки по ключевым параметрам, таким как:

- лицензирования для некоммерческого использования – необходимые денежные отчисления за некоммерческое использование движка. В большинстве случаев это бесплатное распространение.
- лицензирования для коммерческого использования – необходимые денежные отчисления за коммерческое использование движка. Самым распространенным являются отчисления с продаж проекта.
- платформы – платформы на которые можно создавать решения и проекты на данном движке, а также наличие инструментов для анализа использованных ресурсов и отладки на данную платформу.
- язык Программирования – поддерживаемые языки программирования, на которых можно вести разработку программной части игры.
- графический API, Звуковой API – поддерживаемые API, на основе этих параметра можно предполагать, какие технологии могут поддерживаться данным движком. Особенно важно обращать на это внимание, если есть необходимость импортировать в движок то, что было создано вне его среды.
- физический движок – используемый физический движок, отвечающий за обработку всех физических явлений.
- наличие встроенных инструментов для работы с анимацией, материалами и прочим. Все это существенно ускоряет работу, если находится в одной среде разработки.
- поддержка разработки на основе 2D,3D, AR/VR. Большинство крупных движков хоть и поддерживают большинство типов разработки, специализируются на только на определенном. И лишь несколько предоставляют возможности для всех типов на равных, зачастую интегрируя сторонние решения в угоду разработчиков.

1.3.1 Unreal Engine 4

Unreal Engine – полный набор инструментов для создания любых видений игрового проекта на высоком технологическом уровне. Разрабатывается и поддерживается игровой движок студией EpicGames [5].

В таблице 1 представлен основной обзор UnrealEnfine4 по ключевым критериям:

Таблица 1 – Игровой движок Unreal Engine 4

Критерий	Unreal Engine 4
Лицензирования для некоммерческого использования	На бесплатной основе
Лицензирования для коммерческого использования	5% отчислений от продаж после достижения ими отметки в 3000$
Платформы	Windows, MacOS, Xbox One, PlayStation 4PlayStation Vita, iOS, Android, Nintendo Switch
Язык Программирования	C++,Визуальная система программирования Blueprint
ГрафическийAPI	DirectX 11, DirectX 12
Звуковой API	XAudio2
Физический движок	PhysX
Сетевая система	Unreal networking

Окончание таблицы 1

Критерий	Unreal Engine 4
Наличие редактора уровней	+
Наличие редактора кинематографичных сцен	+
Наличие редактора анимации	+
Поддержка разработки на основе 3D	+
Поддержка разработки на основе 2D	+
Встроенная поддержка AR/VR	–
Открытый исходный код	+

Движок поддерживает все огромное разнообразие современных технологий. В настоящее время UnrealEngine4 занимает одно из лидирующих позиций и рассматривается большим количеством разработчиков для своих проектов благодаря высокому техническому исполнению, большому количеству доступных платформ, полная поддержка 3D и возможность создавать 2D проекты, хоть движок имеет другую направленность, а также обширным обучающим материалам и магазину готовых решений от других разработчиков.

С самого первого дня UnrealEngine4 его популярность только растет, чаще на этот движок переходят те, кто ранее пользовался Unity, которые выросли

в профессиональном плане и готовы использовать инструменты, которые хоть и сложнее, но дают большую функциональность. Но в последнее время темпы его развития замедлились, ввиду смены приоритетов компании.

1.3.2 CryEngine 5

CryEngine – игровой движок, созданный немецкой частной компанией Crytek в 2002 году и первоначально используемый в шутере от первого лица Far Cry. «CryEngine» –коммерческий движок, который предлагается для лицензирования другим компаниям. С 30 марта 2006 года все права на движок принадлежат компании Ubisoft [6].

В игровой индустрии движок CryENGINE давно заработал репутацию одного из самых продвинутых в визуальном и физическом плане, поэтому долгое время его лицензировали крупные игровые компании для собственных разработок. В настоящее время новые версии этого движка все так же выступают неким эталоном в качестве картинки и является передовым в техническом плане.

В таблице 2 представлен основной обзор CryENGINE5 по ключевым критериям:

Таблица 2 – Игровой движок CryEngine 5

Критерий	CryEngine 5
Лицензирования для некоммерческого использования	На бесплатной основе, оплата на добровольной основе
Лицензирования для коммерческого использования	На бесплатной основе, оплата на добровольной основе
Платформы	Windows, MacOS,Linux,Xbox One, PlayStation 4
Язык Программирования	C++,C#, Lua
ГрафическийAPI	DirectX 11, DirectX 12
Звуковой API	FMOD

Физический движок	CryENGINE physics

Окончание таблицы 2

Критерий	CryEngine 5
Сетевая система	CryNetwork
Наличие редактора уровней	+
Наличие редактора кинематографичных сцен	–
Наличие редактора анимации	+
Поддержка разработки на основе 3D	+
Поддержка разработки на основе 2D	–
Встроенная поддержка AR/VR	+
Открытый исходный код	+

Среди разработчиков движок распространяется абсолютно бесплатно, но у пользователей есть возможность финансово поощрить работу компании. На странице, через которую распространяется CryEngine5, можно пожертвовать средства в пользу Crytek или Dev Fund – фонда поддержки независимых студий. Благодаря такому подходу движок, доступ к которому для коммерческих проектов имели только большие студии, стал общедоступным.

1.3.3 Unity

Поскольку ранее был сделан выбор в пользу последней версии фреймворка Laravel, который требует минимальную версию PHP 7.1.3, было принято решение виртуализировать среду разработки из-за сложностей в настройке одновременного функционирования нескольких версий PHP.

Unity – это кросс–платформенный игровой движок, разработанный UnityTechnologies. Его основное использование – разработка видеоигр и симуля-

торов. Видеоигры, которые были разработаны на этом движке, также могут быть выпущены для консолей, мобильных устройств и веб–сайтов. Этот движок затем был расширен до 27 других платформ [7].

В таблице 3 представлен основной обзор Unity по ключевым критериям:

Таблица 3 – Игровой движок Unity

Критерий	Unity
Лицензирования для некоммерческого использования	На бесплатной основе
Лицензирования для коммерческого использования	Без отчислений до отметки продаж в 100000$, после достижения этой отметки за каждое рабочее место выплачивается 1500$ либо 75$/мес.
Платформы	Android, Apple TV, BlackBerry, iOS, Linux, Nintendo 3DS Line, macOS, PlayStation4, PlayStation Vita, Unity Web Player, Wii, Wii U, Nintendo Switch, WindowsPhone8, Windows, Xbox360,Xbox One
Язык Программирования	C#, JavaScript, Action Script
ГрафическийAPI	DirectX9, DirectX10,DirectX11, DirectX12, OpenGL
Звуковой API	FMOD
Физический движок	PhysX
Сетевая система	Unity networking
Наличие редактора уровней	+
Наличие редактора кинематографичных сцен	+
Наличие редактора анимации	+
Поддержка разработки на основе 3D	+
Поддержка разработки на основе 2D	+
Встроенная поддержка AR/VR	+

Открытый исходный код	–

Движок используется на различных платформах и в разных проектах, а не только играх, поскольку в Unity совмещены самые различные модули для различных целей, со встроенными средствами анализа оптимизации. Как пример, именно этот движок использует «Сбербанк» для разработки своих решений для ритейла и финансового сектора.

Unity является самым распространенным игровым движком благодаря своей относительной легкости в инструментарии, хоть и менее функциональном, большому количеству обучающих материалов и огромному сообществу, которое сможет помочь. Также в Unity самый обширный список загружаемых материалов, созданных самим сообществом, и находящихся в свободном доступе.

1.3.4 Source 2

Source (официальное название: Valve Source Engine) –трехмерный движок, разработанный компанией Valve Corporation. Его уникальными особенностями считаются модульная основа и гибкость, художественность, рендеринг основанный на шейдерах, непревзойденная синхронизация губ и технология выражения эмоций и система физики, полноценно работающая по сети. Дебютом Source можно считать ноябрь 2004 года – выход первых игр, на его основе: Counter–Strike: Source и Half–Life 2 [8].

В таблице 4 представлен основной обзор Source2 по ключевым критериям:

Таблица 4–игровой движок Source 2

Критерий	Source 2
Лицензирования для некоммерческого использования	На бесплатной основе

Лицензирования для коммерческого использования	Только для внедрения в проекты Valve с отчислениями с продаж с различной комиссией
Платформы	Windows, Mac, Linux, Xbox 360, Xbox One
Язык Программирования	C++

Окончание таблицы 4

Критерий	Source 2
ГрафическийAPI	DirectX9, DirectX10,DirectX11
Звуковой API	Miles
Физический движок	Havok
Сетевая система	ValveNetwork
Наличие редактора уровней	+
Наличие редактора кинематографичных сцен	+
Наличие редактора анимации	+
Поддержка разработки на основе 3D	+
Поддержка разработки на основе 2D	–
Встроенная поддержка AR/VR	–
Открытый исходный код	–

Даже с учетом, что зарабатывать на играх, созданных на данном движке, нет возможности, только если не получать личное соглашение через компанию Valve, данный движок является очень популярным. Связано это с тем, что инструментарий Source позволяет очень легко создавать новый контент для уже существующих игр компании Valve и легко внедрять его, а получать доход именно с внутренней продажи внутри игр. Данный подход не вызывает внутри сообщества какой–либо негативной реакции, связано это с высокой лояльностью к компании, популярности её игр и хорошей репутации.

1.4 Итоги анализа

Разрабатываемый проект будет иметь 2D графику. Из рассмотренных движков только Unity и Unreal Engine предполагает разработку с ней и внедряют самые современные и удобные инструменты для работы. Но именно Unity имеет встроенные системы, которые ускоряют работу (как например, аниматор и пр.). Но, что самое важное, имеет функции в своей API, которые делают обработку 2D графики намного качественнее. Unreal Engine, в свою очередь, также имеет весь необходимый функционал, но все же пока является более ориентированным именно на 3D, который при работе с которым Unreal Engine предпочтительнее.

Не менее важным требованием является быстрота создания готовых шаблонов объектов для их последующей загрузки на сцены. И в Unity, и в Unreal Engine это позволяют сделать встроенные основные компоненты для всех типов объектов и UI, разработчику на начальных этапах предстоит только добавить эти компоненты на пустой объект. В данном случае движки не уступают друг другу.

Из потребности в удобном функционале для разработки проекта в 2D, распространенности движка, а также большего опыта работы именно с ним, мой выбор пал на Unity.

Также, после проведения анализа были выявлены основные особенности игрового жанра rogue-like и проведен анализ одного из главных представителей жанра – FTL: Faster than Light, установлены ключевые игровые механики и принципы работы искусственного интеллекта в игре. На основе этого анализа и будет создан собственный проект.

2 Этап проектирования

Создаваемый проект представляет собой пошаговую стратегию в научно-фантастическом окружении с элементами процедурной генерации локаций. Графическим стилем данного проекта была выбрана пиксельная стилизация.

Цель игры в жанре rogue–like обычно заключается в достижении определенной конечной точки, но, как и опять же общепринято, через процедурную генерацию локаций на которых и будут происходить все события.

Ориентирами и вдохновением для создания собственного проекта послужили такие игры в жанре rogue–like, как FTL: Faster Than Light, Moonlighter, а также космические RPG, например, EvE Online и Starbound, а также тактические игры, навроде XCOM.

2.1 Описание основных механик

Для постановки задачи изначально необходимо описать общие механики и принципы геймплея. Под управлением игрока находится один космический корабль, а также команда, которая и будет непосредственно участвовать в бою. Команда может состоять из нескольких юнитов, количество которых зависит от уровня модуля управления корабля. Сами юниты делятся на несколько типов: легкий класс, средний класс и тяжелый класс – каждый класс различается количеством очков прочности, типом носимого оружия и подвижностью. Каждый юнит обладает определенным количеством очков, который он может тратить на перемещение и атаку.

Перемещение по глобальной карте реализовано как в FTL: Faster Than Light – игроку необходимо переместиться из первой точки, в точку выхода в следующий сектор, сделать это предстоит через локацию, представляющую сгенерированный связный граф, вершины которого и являются зонами со случайными событиями.

На карте сектора есть множество локаций, такие как:

- начальная и конечная точки сектора – локации, которые служат как точка старта сектора и точка перемещения в следующий сектор.

- торговая зона – локация, где игрок может покупать всевозможные улучшения, а также продавать ненужные вещи и ресурсы, которые он найдет в других локациях.
- враждебная зона – локация, где игроку встречается боевое событие, и он может либо попытаться избежать его, либо атаковать.
- зона события – локация, где игроку предоставляется некоторое сюжетным событие, события в котором могут вызвать как положительные результаты, так и отрицательные.
- зона пустоты – локация, где игроку ничего не встречается, лишь пустота космоса.

Игра заканчивается при достижении последнего сектора и прохождении через финальную зону.

Боевая система представляет собой пошаговую стратегию, на сгенерированном поле, где могут быть расположены различные предметы, либо же препятствия. После завершения сражения, в зависимости от результата выполненных задач, происходит определенное событие и выдаются награды. В случае поражения в бою игра не заканчивается, но игрок теряет и часть ресурсов, имеет шанс потерять членов команды, а также прочность своего корабля.

Игра считается проигранной в случае потери всей прочности корабля. Но жанр rogue–like подразумевает перманентный проигрыш, чтобы игрок все заново и заново начинал игру с начала.

2.2 Описание общей структуры

В движке Unity все объекты располагаются на сценах. В игровых проектах их может быть большое множество. Ниже приведена схема проекта, на которой изображены все переходы между основными сценами и окнами, каждая имеет множество собственных объектов, выполняющих определенные функции (Рисунок 2 – Общая схема переходов).

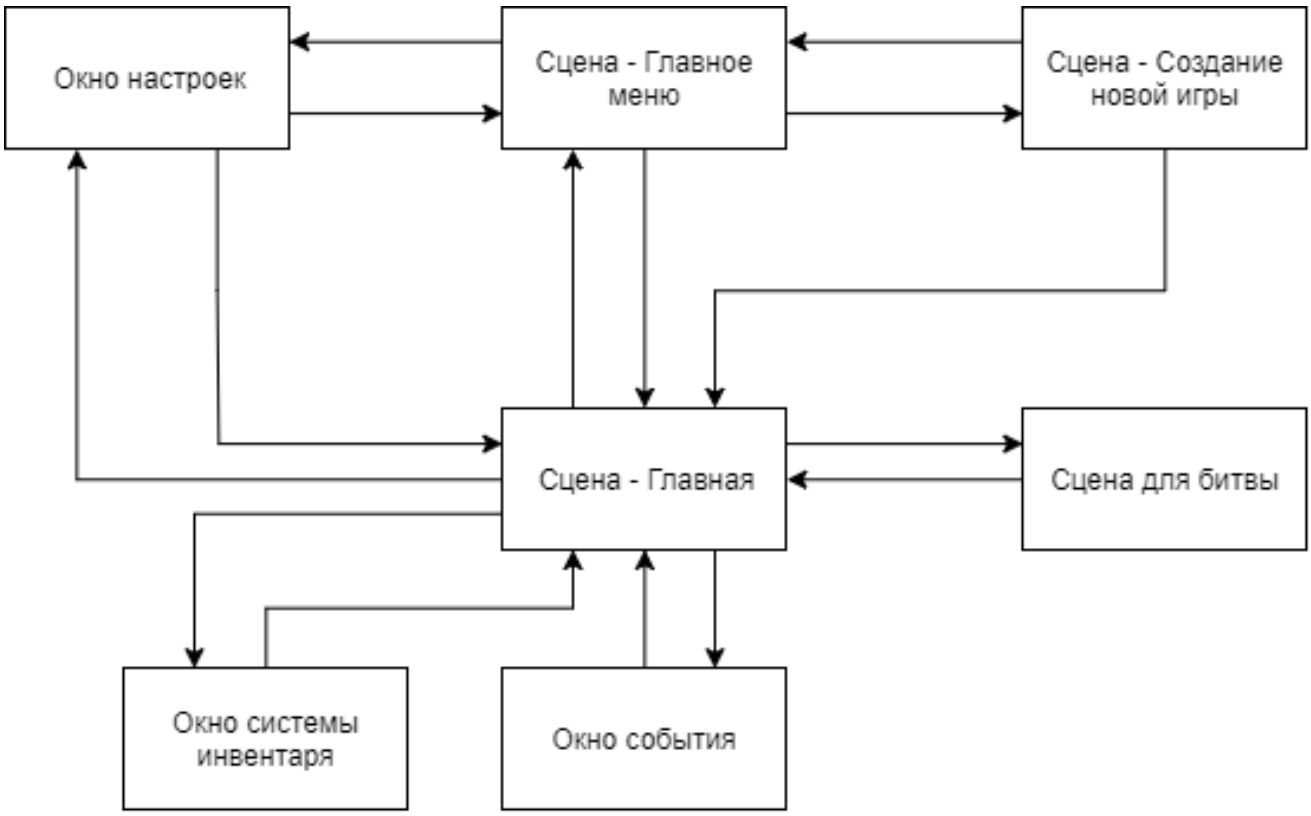

Рисунок 2 – Общая схема переходов

В схеме присутствуют следующие основные элементы:

- окно настроек, которое можно вызвать из главной сцены и сцены главного меню. В нем пользователь может настроить разрешение и звук.
- главное меню – сцена, из которой идет загрузка сохраненной игры и старт новой.
- создание новой игры – меню старта для новой игры, в котором игрок знакомится со стартовыми условиями игры.
- главная сцена – основная сцена игры, на которой игрок выбирает куда двигаться и где он имеет доступ к инвентарю и всей информации, из этой сцены осуществляется доступ ко всем остальным сценам.
- окно системы инвентаря – окно, где игрок распоряжается всем доступным ему снаряжением.
- окно события – окно, которое появляется при входе в зону события, служит для оповещения.

- сцена для битвы – сцена, которая содержит весь необходимый функционал для геймплея боевой системы, загружается при входе во враждебную зону.

2.3 Генерация поля боя

Для игр, принадлежащих к жанру rogue–like, обязательным условием является генерация случайной локации, чтобы обеспечивать игроку множество различных условий, в которых он может оказаться [9].

Необходимо реализовать класс, который сможет реализовать данный функционал, используя заранее созданные графические объекты, именуемые спрайтами, и сохранять в себе созданную карту, для последующей работы с ней.

Ниже приведены последовательные шаги, которые должен выполнять разрабатываемый класс (Рисунок 3 – Генерация поля для битвы).

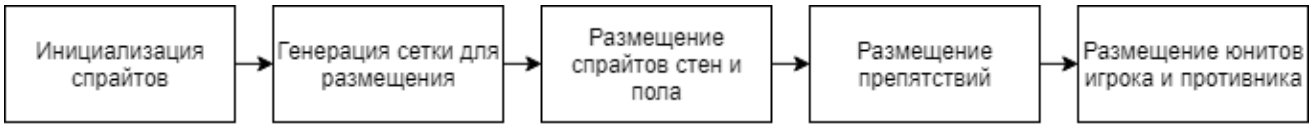

Рисунок 3 – Генерация поля для битвы.

Поскольку для игры важно разнообразие, то для начала необходимо создать набор спрайтов, которые будет выбираться случайно из набора для последующего размещения.

Также в подготовительный этап можно включить генерацию сетки для размещения. Данный этап необходим для дальнейшей работы алгоритма, поскольку размещение всех объектов будет происходить по заранее сгенерированным сеткам, включающими в себя координаты каждых из доступных точек размещения. При постановке объектов сетки корректируются – из них удаляются занятые координаты, что позволяет избежать повторения.

После идет непосредственное размещение спрайтов и всех игровых объектов для подготовки игровой сцены. Также все три последних шага генерации производят начальную инициализацию матрицы карты, которая в дальнейшем будет передана для использования искусственным интеллектом.

2.4 Алгоритмы для искусственного интеллекта

Поскольку карта поля боя генерируется случайным образом, то реализовать ИИ стандартными средствами Unity не представлялось возможным, а также поскольку управление игроком собственными юнитами не является прямым, ведь игра представляет собой пошаговую стратегию, то необходимо реализовать несколько алгоритмов для того, чтобы юниты могли ориентироваться в сгенерированном пространстве.

Главным способом ориентации в пространстве у юнитов служит использование тепловых карт и волновых алгоритмов для их заполнения, которые по–разному используются для юнитов игрока и юнитов под управлением компьютера. Через анализ карт юниты под управлением компьютера могли адаптироваться к действиям игрока, использую собственные очки действий в текущей ситуации эффективно.

Волновой алгоритм заполняет незанятые точки в матрице, начиная из точки, где находится выбранный юнит, и с каждой итерацией увеличивает индексы, заполняемых на текущем прохождении цикла, клеток. В результате получается карта, где клетки рядом с юнитом имеют наименьший индекс, а самые удаленные – наибольший. Тем самым юниты могут отслеживать необходимые пути до точек [10].

Тепловые карты, в свою очередь, используются исключительно юнитами компьютера, заполняются схожим образом, с той разницей, что начальных точек множество и у каждой может быть различный индекс. Карты выступают инструментом для анализа, которые позволяют определять приоритетные точки на поле [11].

2.5 Итоги проектирования

В ходе выполнения этапов проектирования была разработана общая структура программы, а также общие геймплейные возможности и особенности. Был разработан алгоритм для процедурной генерации локации, чтобы создать большее разнообразие в игре. А также разработан принцип работы искусственного интеллекта юнитов под управлением компьютера.

3 Реализация проекта

При выполнение выпускной квалификационной работы были реализованы все основные механики проекта.

3.1 Основные данные игрока

Ниже представлена диаграмма из классов, которые составляют данные об игроке (Рисунок 4– Диаграмма классов игрока). Основными классами являются Player и ShipPlayer. Они хранят в себе все элементы, через которые игрок взаимодействует с игрой.

Класс ShipPlayer служит для отображения корабля игрока на игровой сцене, с параметрами которого взаимодействует непосредственно игрок, перемещаясь в другой сектор, сменяя оснащение корабля или вносит прочие изменение характеристик из-за какого-либо события.

Класс Player является обозначением на сцене объекта игрока, он связан со всеми классами, которые находят под управлением игрока и с которыми он может взаимодействовать.

Класс UnitPlayer необходим для объектов юнитов игрока как при загрузке на сцене битвы, так и в главной сцене при распределении юнитов в отряде. Служит для взаимодействия игрока с юнитами, находящимися под его управлением.

Также диаграмма включает содержит вспомогательные сериализуемые классы, такие как StatesOfPlayer, StatesOfShip, StatesOfUnit и StatatesOfGun, они

необходимы для сохранения данных сессии, сохранение осуществляется через сторонний класс, в котором класс помещается через бинарный форматер в файл сохранения и загружает имеющиеся файлы при старте новой сцены.

Данные классы располагаются в объектах на всех геймплейных сценах проекта.

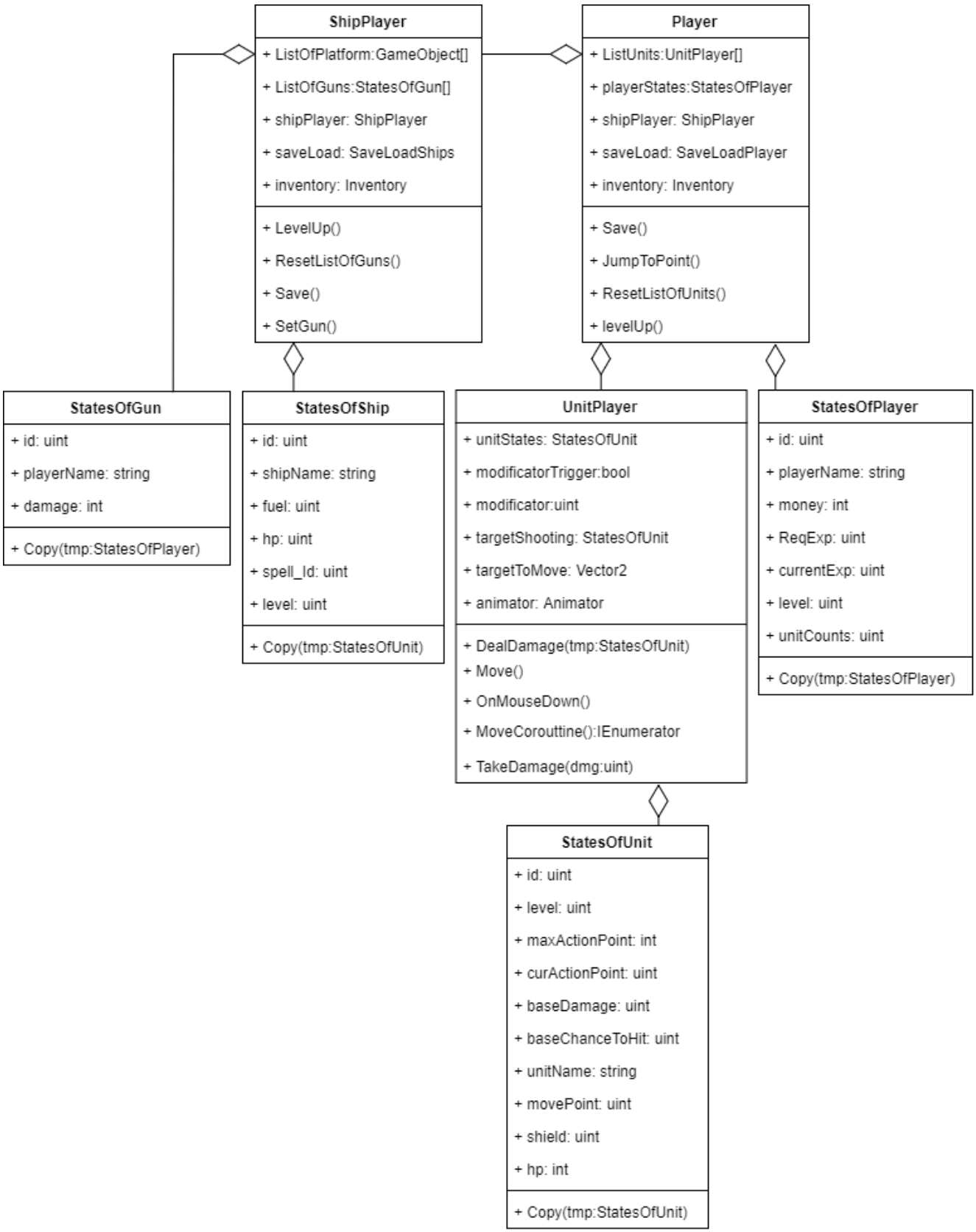

Рисунок 4 – Диаграмма классов игрока

3.2 Главное меню и вспомогательные менеджеры

Всю логику игровых сцен и их механик обеспечивают игровые менеджеры, каждый обеспечивает определенные функции, основными же для всех являются: инициация сохранения и загрузки данных пользователя, а также управление инициализаций объектов сцены. Все менеджеры являются объектами, которые представлены на сцене в единичном экземпляре.

Самым простым по функционалу является менеджер из главного меню – MainMenuManager (Рисунок 5 – Класс менеджера главного меню), он служит для обработки нажатия кнопок, проверяет возможность продолжить начатую ранее игру, а также запуск каких-либо случайных событий на экране меню (Рисунок 6 – Экран главного меню), чтобы меню не считалось пустым. С ним связаны только кнопки, а в себе хранит только объекты для отображения случайных событий.

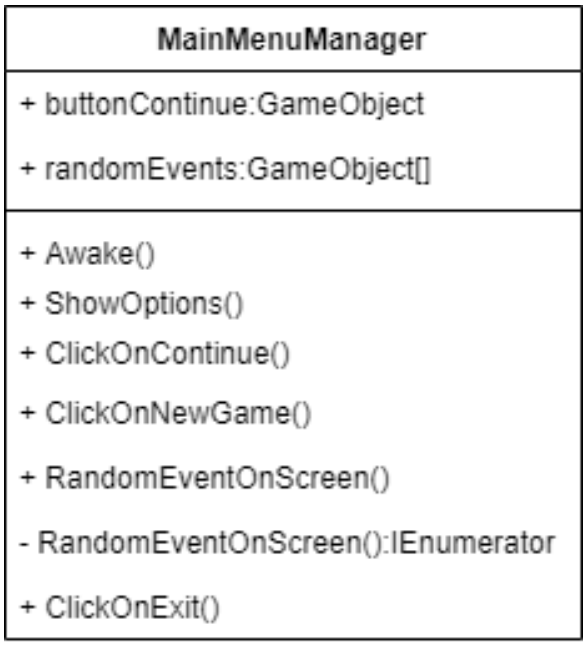

Рисунок 5 – Класс менеджера главного меню

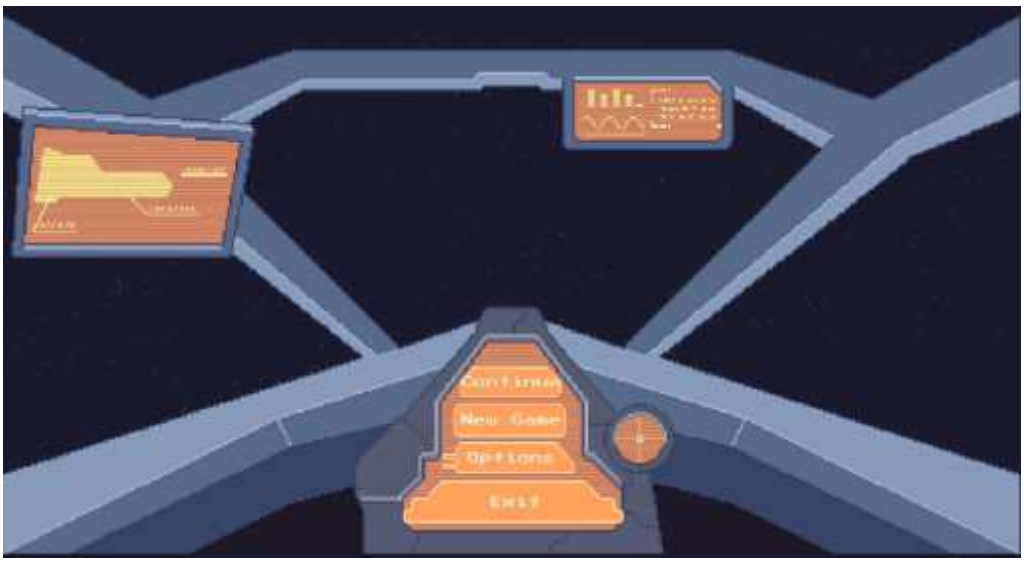

Рисунок 6 – Экран главного меню

Помимо классов менеджеров, которые отвечают непосредственно за логик игры, есть вспомогательные, как например, SoundManager, который отвечает за воспроизведение музыки и звуков, со случайным значением питч-шифтера, чтобы воспроизводимый звук не казался одинаковым из раза в раз. Данный класс является постоянным объектом, чтобы при переходе между сценами звук не прерывался, а мог и в дальнейшем воспроизводить музыку, необходимо лишь сменять воспроизводимые файлы.

3.3 Создание новой игры

В сцене создания новой игры менеджером является CreateMenuManager (Рисунок 7 – Диаграмма с классом менеджера создания новой игры), он занимается отображением выбранной заготовки персонажа, которая будет под управлением игрока на старте, демонстрацией этого на сцене (Рисунок 8 – Экран создания новой игры).

При старте игры менеджер сохраняет заготовку персонажа в файл и переходит к загрузке главной сцены игры, на которой и происходит загрузка ранее сохраненного файла.

Класс менеджера CreateMenuManager связан напрямую с классом Player, который, в свою очередь связан со всеми классами, предоставляющими данные об игроке и подконтрольных ему объектах, описанные выше.

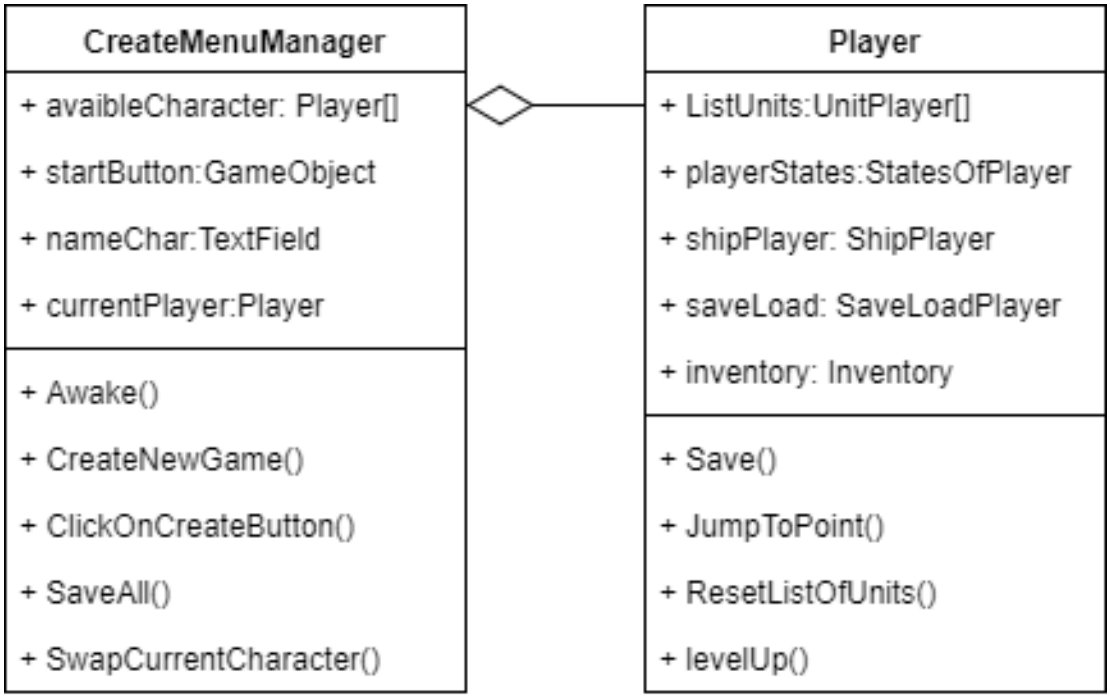

Рисунок 7 – Диаграмма с классом менеджера создания новой игры

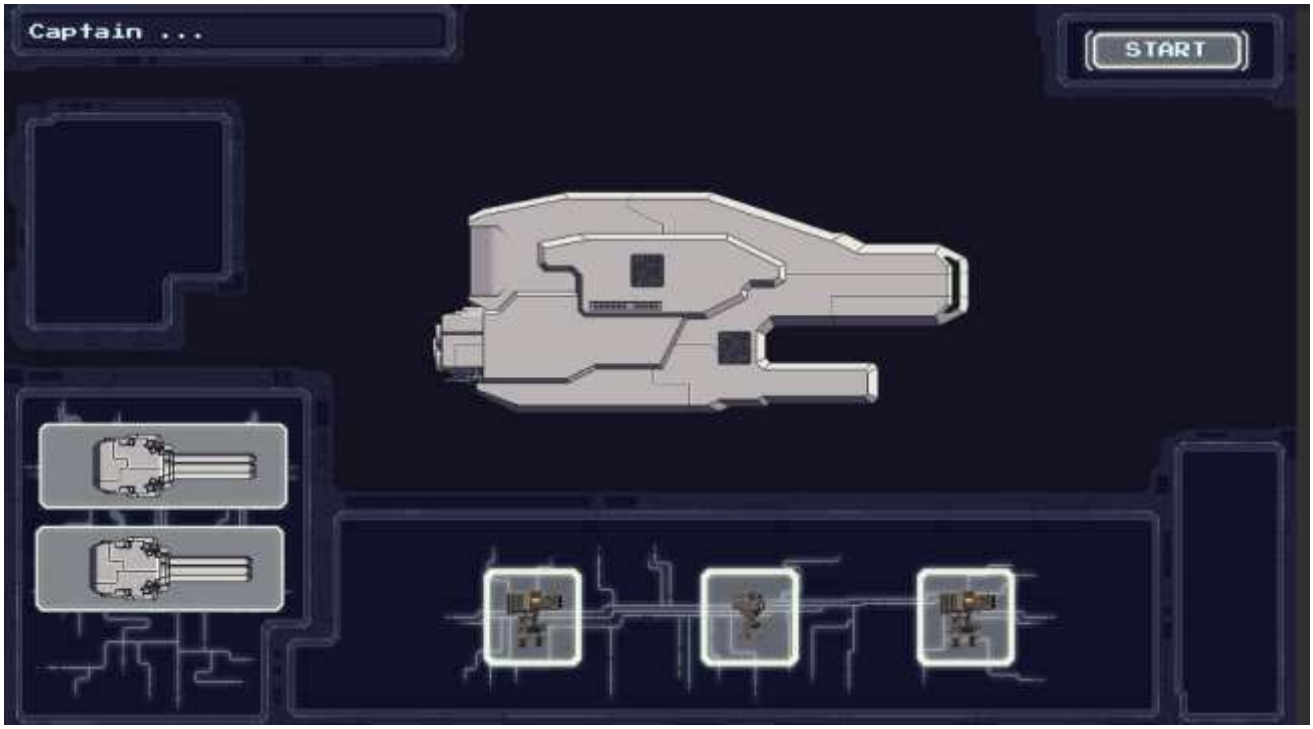

Рисунок 8 – Экран создания новой игры

3.4 Главная сцена

Менеджер главной сцены HubManager (Рисунок 9 – Диаграмма с классом менеджера главой сцены) обрабатывает все действия на этой сцене (Рисунок 8 – Экран главной сцены), как нажатие кнопок, переключение окон и обработку всех данных на сцене. Данный класс хранит ссылки на все окна, такие как окно опций, окно снаряжения корабля, окно игрока. Также менеджер хранит ссылку на текущего игрока – класс Player, и далее на все его данные, общая диаграмма которых была представлена выше.

Для управления сценой HubManager хранит данные локации. Менеджер обрабатывается продвижение игрока по локации и может подстраивать локации под игрока, на основе этих данных, в этом выражается адаптивность данного менеджера. Общая структура представлена ниже.

Класс локации Location, находящийся на данной сцене, представляет собой набор данных о зонах в массиве класса Point, а также хранит в себе сериализуемый класс StatesOfLocation, необходимый для хранения в файле общих характеристик всей локации. Класс Point, в свою очередь, необходим для представления отдельной зоны на карте и хранит в себе сериализуемый класс StatesOfPoint для последующего сохранения в файл и загрузки из него данных каждой отдельной зоны.

Рисунок 8 – Экран главной сцены

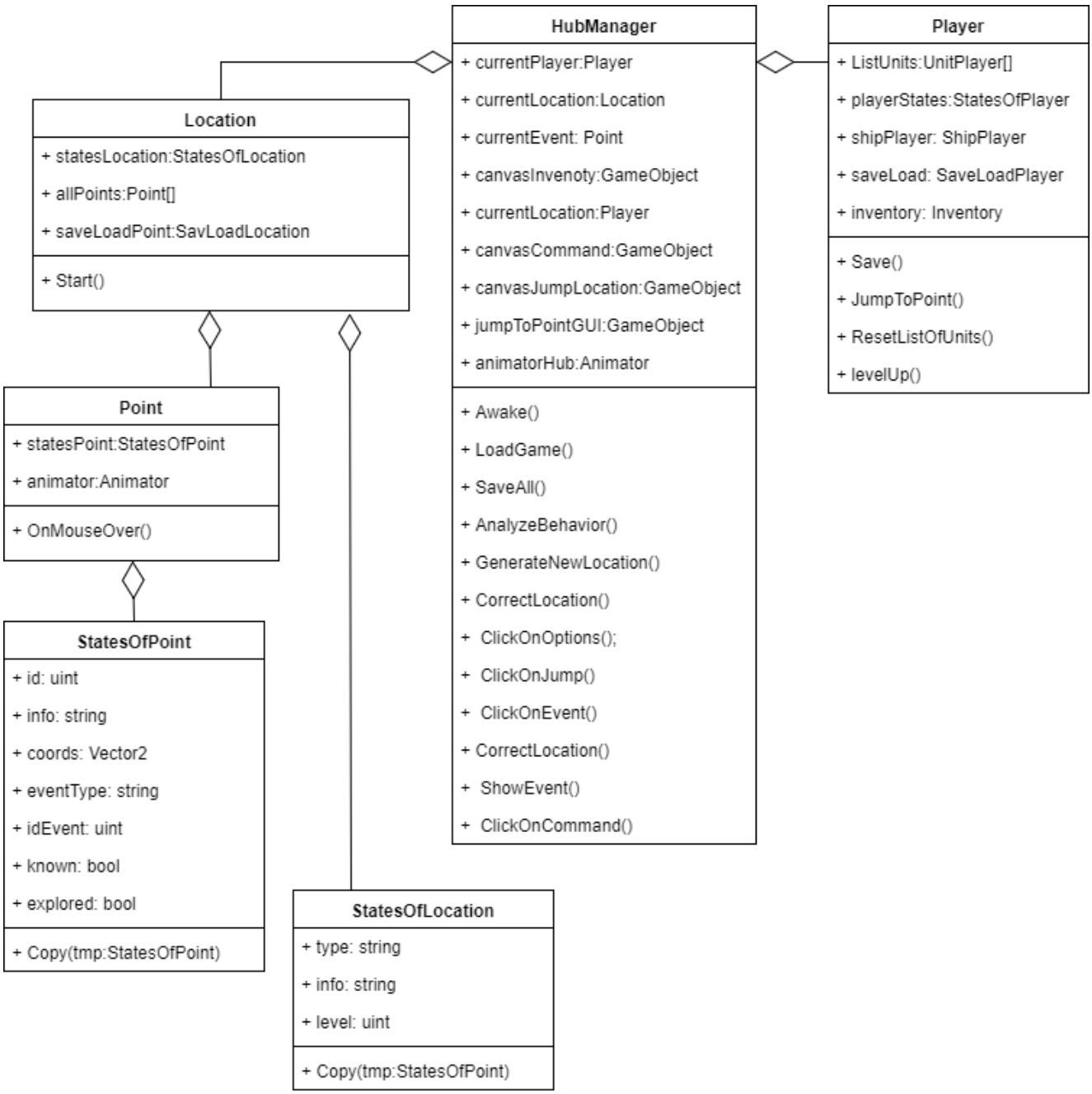

Рисунок 9 – Диаграмма с классом менеджера главой сцены

3.5 Боевая сцена

Поскольку основной геймплей представлен на сцене боевого события, данная сцена содержит самое большое количество реализованных классов и объектов. На сцене присутствуют два менеджера это менеджеры BoardManager и BattleManager (Рисунок 10 – Диаграмма с классом менеджера главой сцены).

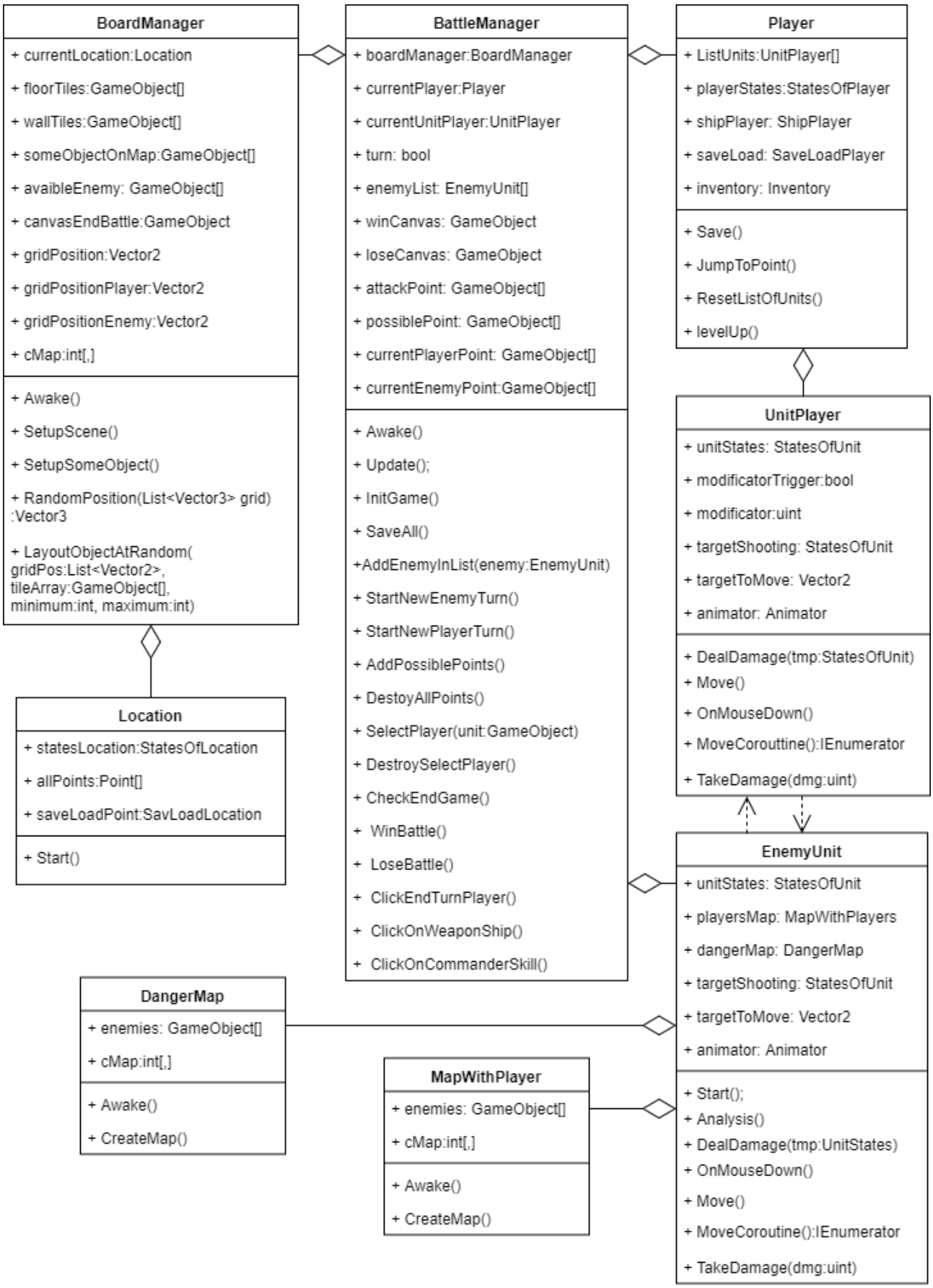

Рисунок 10 – Диаграмма с классом менеджера боевой

BoardManager выполняет на сцене генерацию карты из различных тайлов, представляющие собой игровые объекты с различными спрайтами (Рисунок 11 – Данные класса BoardManager в инспекторе движка). Данный менеджер обеспечивает элемент случайности при старте сцены битвы, создавая различающиеся условия

BoardManager начинает создание карты вызовом функции из BattleManager. Для начала класс создает сетку, на которой в дальнейшем размещаются объекты и подготавливает матрицу, которая впоследствии будет использоваться для тепловых карт.

Далее класс создает пустое пространство поля (Рисунок 12 – Результат работы после метода BoardSetup), размещая стены и пол в соответствии с заданными размерами комнаты. В данном методе объектами для размещения служат случайно выбранные объекты из массивов.

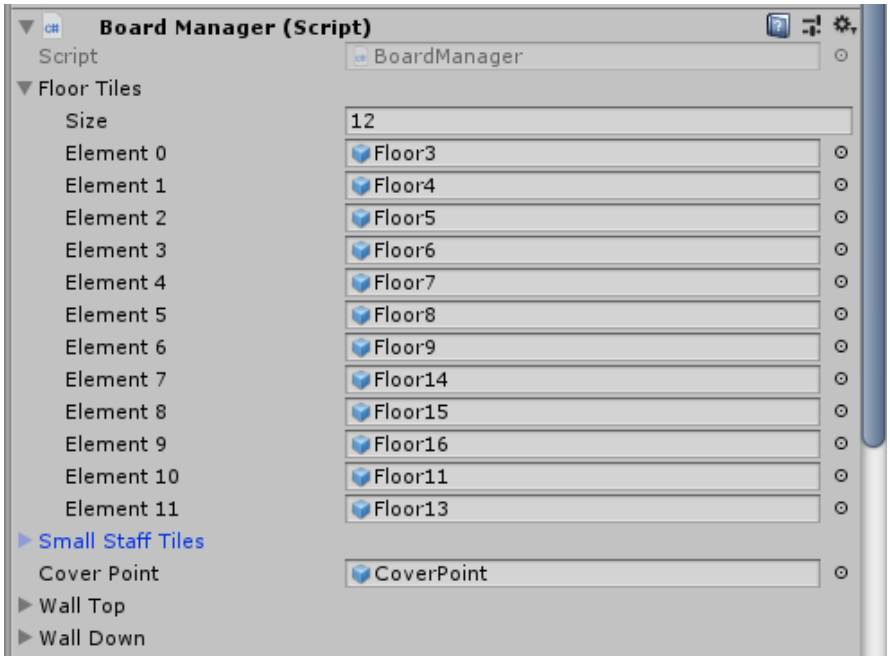

Рисунок 11 – Данные класса BoardManager в инспекторе движка

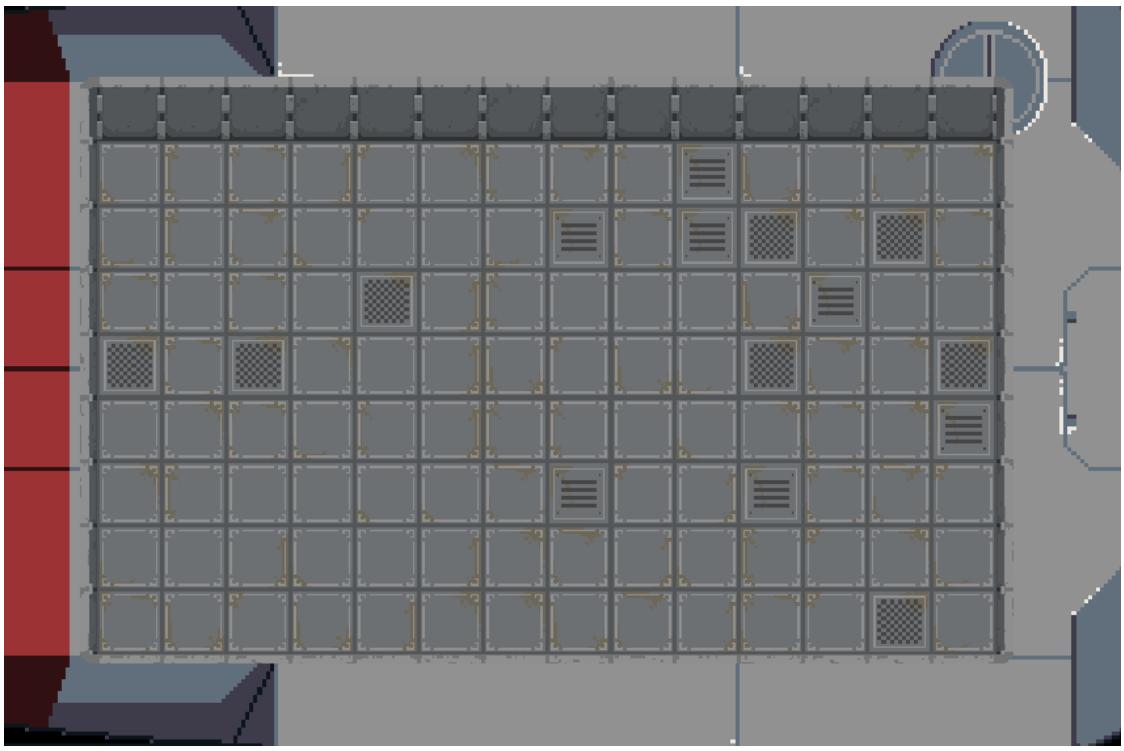

Рисунок 12 – Результат работы после метода BoardSetup

После завершения создания пустой локации необходимо создать объекты, которую будут служить укрытием для юнитов противника и игрока, а также непроходимыми препятствиями, чтобы создавать различные условия в боях (Рисунок 13 – Результат после размещения препятствий). В дальнейшем все установленные объекты будут анализироваться искусственным интеллектом противника.

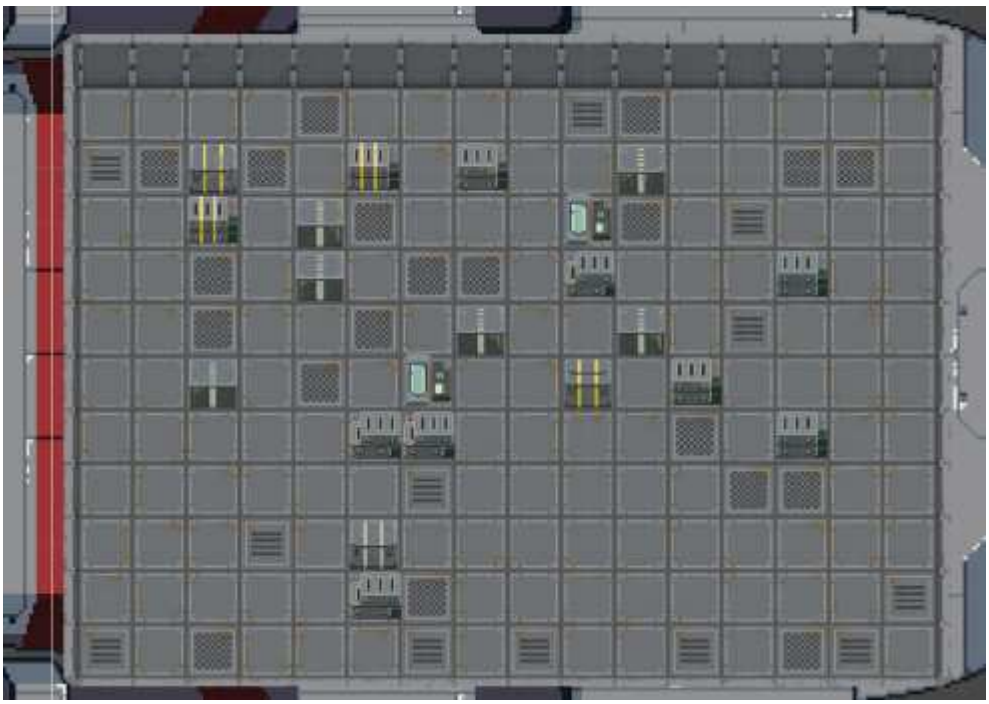

Рисунок 13 – Результат после размещения препятствий

Далее идет размещение персонажей противника и игрока, они размещаются с помощью тех же функций, что и размещаются объекты укрытий, за тем исключением, что там используются другие сетки размещения (Рисунок 14 – Результат после размещения юнитов).

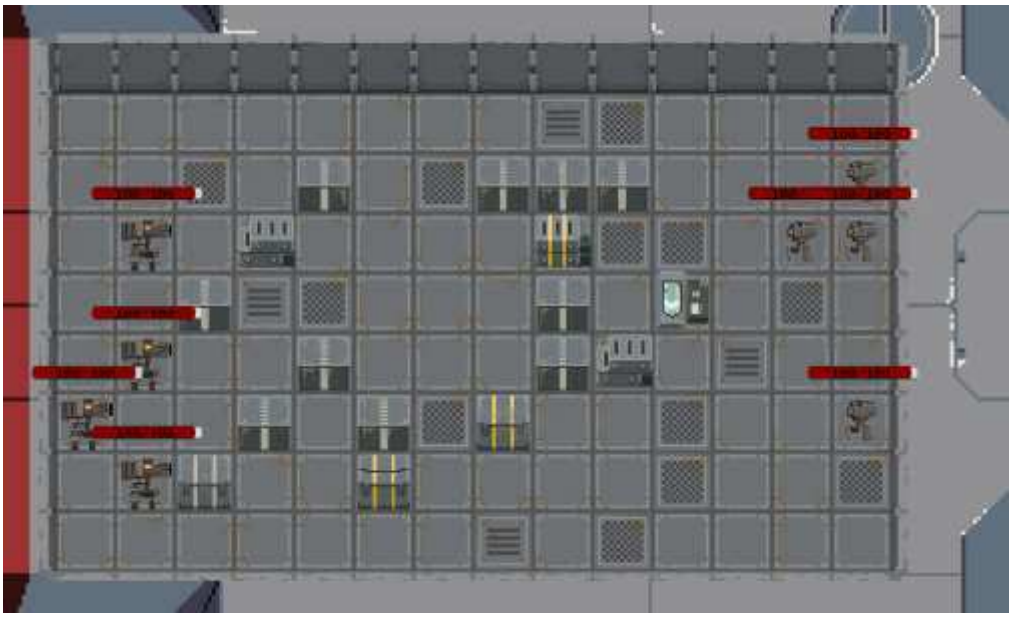

Рисунок 14 – Результат после размещения юнитов

BattleManager же, в свою очередь, обеспечивают всю игровую логику на сгенерированном поле: отслеживание очередности хода и его передачу противнику, обновление хода, отображение на карте путей маршрута передвижения и зон атаки, отслеживание конца боевого события. Класс хранит в себе ссылку на объект игрока и все данные, включая юнитов и корабль. Также менеджер хранит ссылки на всех персонажей противника, чтобы организовывать очередность хода и ссылку на менеджер игрового поля.

Для того, чтобы начать ход противника, менеджер проверяет количество доступных очков действия у каждого юнита игрока. Если же игрок не собирается тратить все доступные очки, он может закончить ход досрочно. После успешной проверки, что ход игрока закончен, менеджер восстанавливает очки действий для юнитов противника и инициирует ход противника у каждого из юнитов по очереди.

После завершения хода противника BattleManager проверяет наличи юнитов игрока, если на поле имеется хотя бы один юнит игрока, то инициируется начало нового хода. Происходит обновление очков действия у каждого из юнитов и показывается оповещение о начале нового хода.

EnemyUnit является базовым классом, по аналогии с UnitPlayer, в нем реализованы все базовые функции юнитов противника по атаке передвижению и анализу дальнейших действий.

3.6 Искусственный интеллект

Поскольку карта генерируется случайным образом, и она не может быть просчитана средствами движка Unity, поэтому для последнего EnemyUnit использует классы DangerMap и MapWithPlaayer для построения тепловых карт, с помощью которых юниты понимают куда необходимо направляться или же наоборот – отступить. EnemyUnit является базовым классом, при реализации новых типов юнитов будет необходимо просто переопределить методы класса, а обращаться к наследуемому классу через ссылку на EnemyUnit.

Главным способом ориентации в пространстве у юнитов служит использование волновых алгоритмов и тепловых карт. Которые по–разному используются для юнитов игрока и юнитов под управлением компьютера.

Алгоритм выполняется каждый раз при выборе юнита игроков, а также для расчетов при выполнении хода компьютера. Изначальная незаполненная карта хранится в классе BoardManager, а заполненные карты хранятся в скриптах MapWithPlayer, MapWithCover в объекте MapCollection на сцене для удобства последующего обращения у всех юнитов противника, также эти классы хранят в себе алгоритм для построения карты своего типа (Рисунок 15 – Объект MapCollection в инспекторе движка).

Рисунок 15 – Объект MapCollection в инспекторе движка

Использование волнового алгоритма необходимо как для юнитов противника, так и игрока. Его реализация заключается в том, что от точки, где находится объект пускается так называемая волна (Рисунок 16 – Первая итерация работы волнового алгоритма), и проходит определенное количество циклов, чтобы заполнить всю матрицу, помечая все незанятые точки индексом текущей итерации. Результатом выполнения будет матрица (Рисунок 17 – Результат работы волнового алгоритма), ячейки которых будут проиндексированы и объект сможет ориентироваться в пространстве, находя путь до конкретной точки (Рисунок 18 – Применение волнового алгоритма для перемещения юнита игрока).

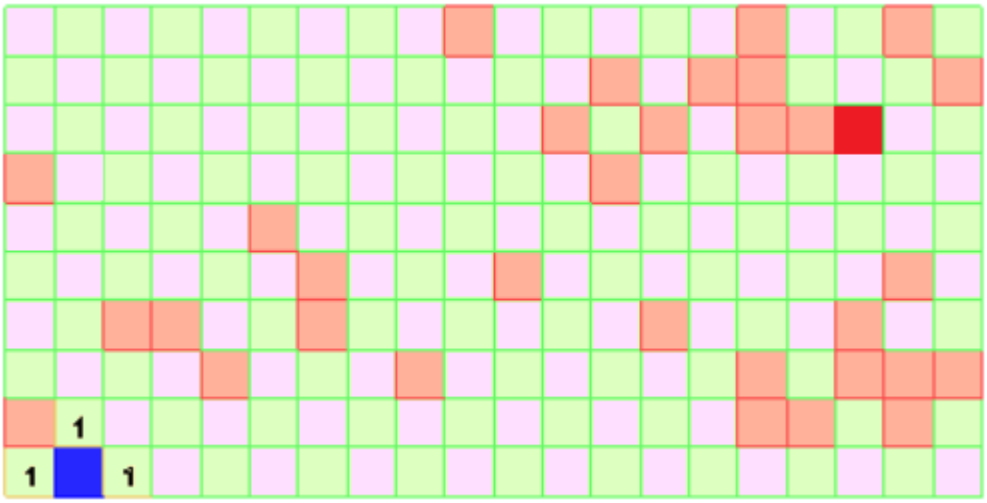

Рисунок 16 – Первая итерация работы волнового алгоритма

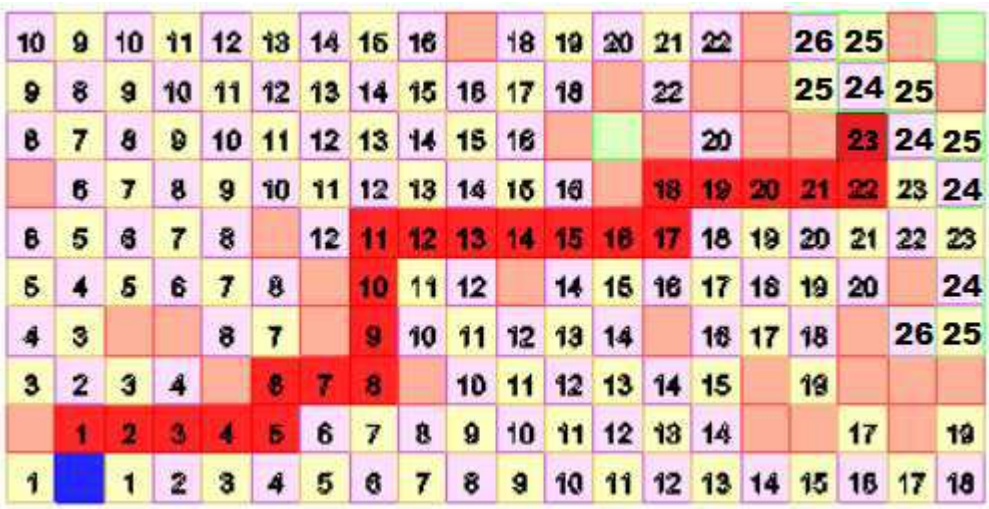

Рисунок 17 – Результат работы волнового алгоритма

Рисунок 18 – Применение волнового алгоритма для перемещения юнита игрока

Волновой алгоритм также применяется для создания тепловых карт, используемых юнитами противника (Рисунок 19 – Пример построенной тепловой карты на матрице карты). При генерации тепловой карты в матрице может указываться несколько тепловых точек с различным индексами, в соответствии с приоритетом цели, как например урон, который нанес конкретный юнит игрока и является представляет большую опасность или координаты укрытия, которое не находится в доступности для юнита игрока, далее волновой алгоритм индексирует все остальные ячейки.

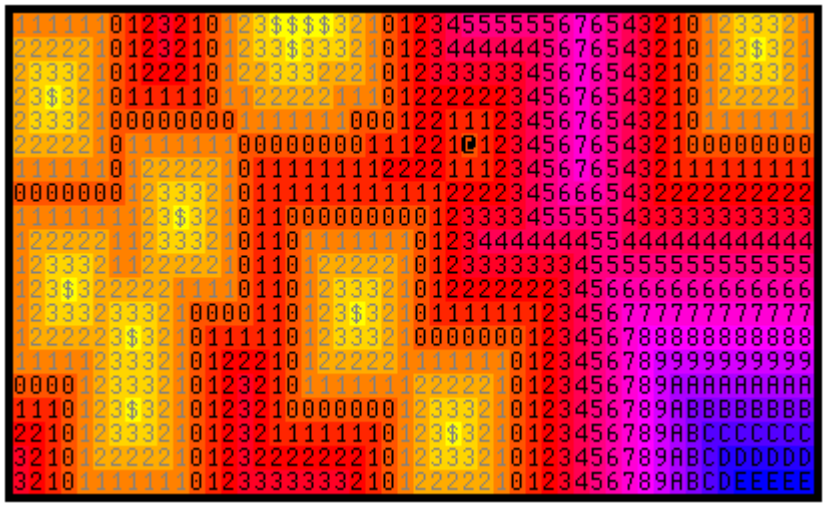

Рисунок 19 – Пример построенной тепловой карты на матрице карты

Ниже представлен фрагмент кода метод CreateMap в классе для генерации карты с укрытиями. Данный метод, после установки точек на всех укрытиях карты, последовательно проверяет доступность каждого из них посредством функции RayinPoint, которая проверяет видит ли какой-либо юнит игрока точку за данным укрытием с фланга, или же нет. После проверки и исключения всех неподходящих точек, с помощью волнового алгоритма строится карта, по которой юнит сможет понять, до какого укрытия ему лучше всего дойти.

Ниже представлена визуализация выполнения данного метода (Рисунок 20 – Выполнения метода CreateMap класса MapWithCover). На приведенном скриншоте красным подсвечен выбранный противником юнит, который ходит в

данный момент, а синим отображаются все возможные точки с укрытиями, до которых не достает игрок в данный момент.

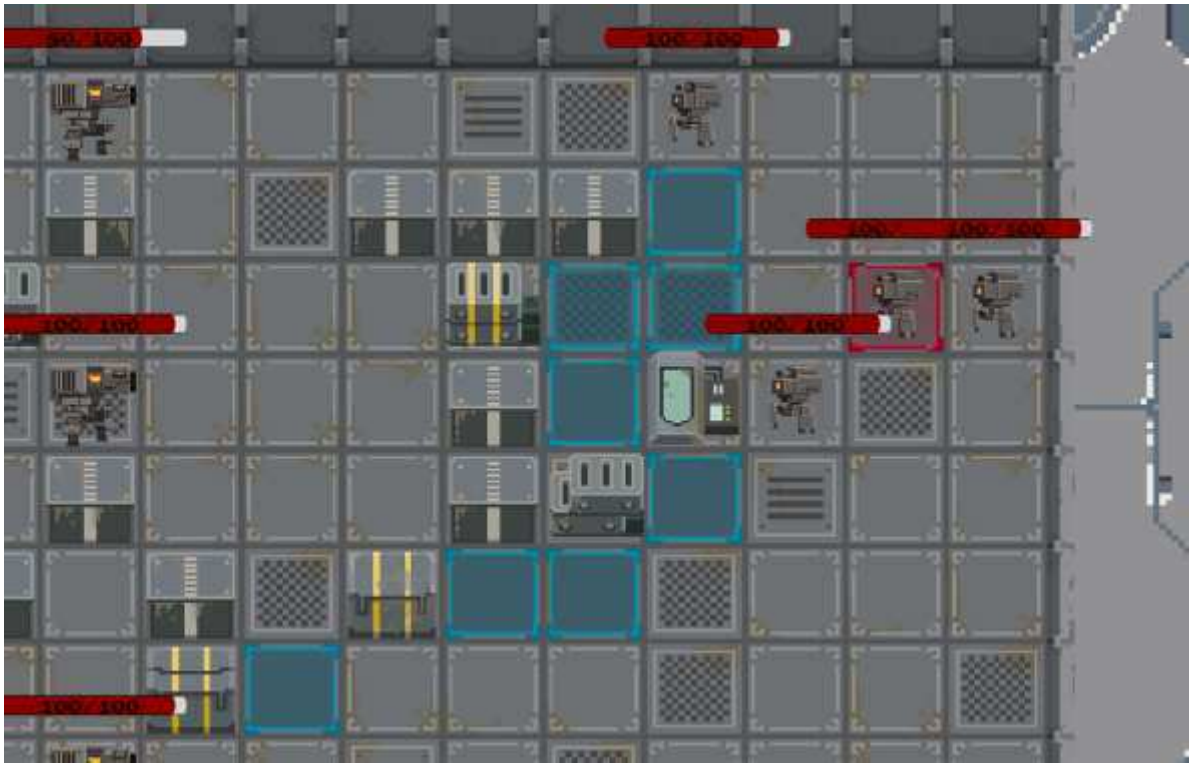

Рисунок 20 – Выполнения метода CreateMap класса MapWithCover

В случае с построением карты с юнитами противника, для определения того, какой из юнитов является более приоритетной целью, находится ближе и опаснее алгоритм работает со схожим принципом, за исключением того, что все юниты имеют свои индексы, которые могут быть различными, в зависимости от его типа и сколько урона он нанес.

После генерации тепловых карт у юнитов противника есть инструмент для анализа, и через обращение к родительскому классу Enemy происходит проверка того, что конкретно предпринять в данной ситуации. Происходит это с помощью проверок в соответствующих классах и взвешивании имеющихся карт, анализ происходит исходя из приоритетов. Приоритеты юнитов могут меняться в зависимости от их типа и состояния. Например, самый простой в реализации юнит будет иметь цель подобраться к игроку вплотную для нанесения урона в ближнем бою, следовательно, для анализа ему понадобится только одна карта – тепловая карта с юнитами игрока. Для более комплексного поведения,

юнит противника использует несколько карт, ставя на первое место собственную сохранность, поэтому в первую очередь проверяет находится ли он в укрытии от игрока и, если нет, – анализирует карту и отходит в подходящее укрытие, и только после этого через анализ тепловой карты с юнитами игрока будет производить дальнюю атаку.

3.7 Система инвентаря

Для изменения состава команды необходима реализации системы инвентаря, которая будет хранить в себе всех неактивных юнитов. Для данной системы необходимо реализовать несколько контейнеров на сцене, которые будут хранить в себе как активных юнитов, так и сам инвентарь, а также систему drag-and-drop.

Для начала необходимо реализовать класс Inventory, который хранит в себе массив со всеми неактивными юнитами и ссылку на объект инвентаря на самой сцене. Данный класс привязан к объекту игрока для удобства в получении данных. Он реализует функции для отображения showInventory и скрытия hideInventory инвентаря, а также удаления из инвентаря объектов.

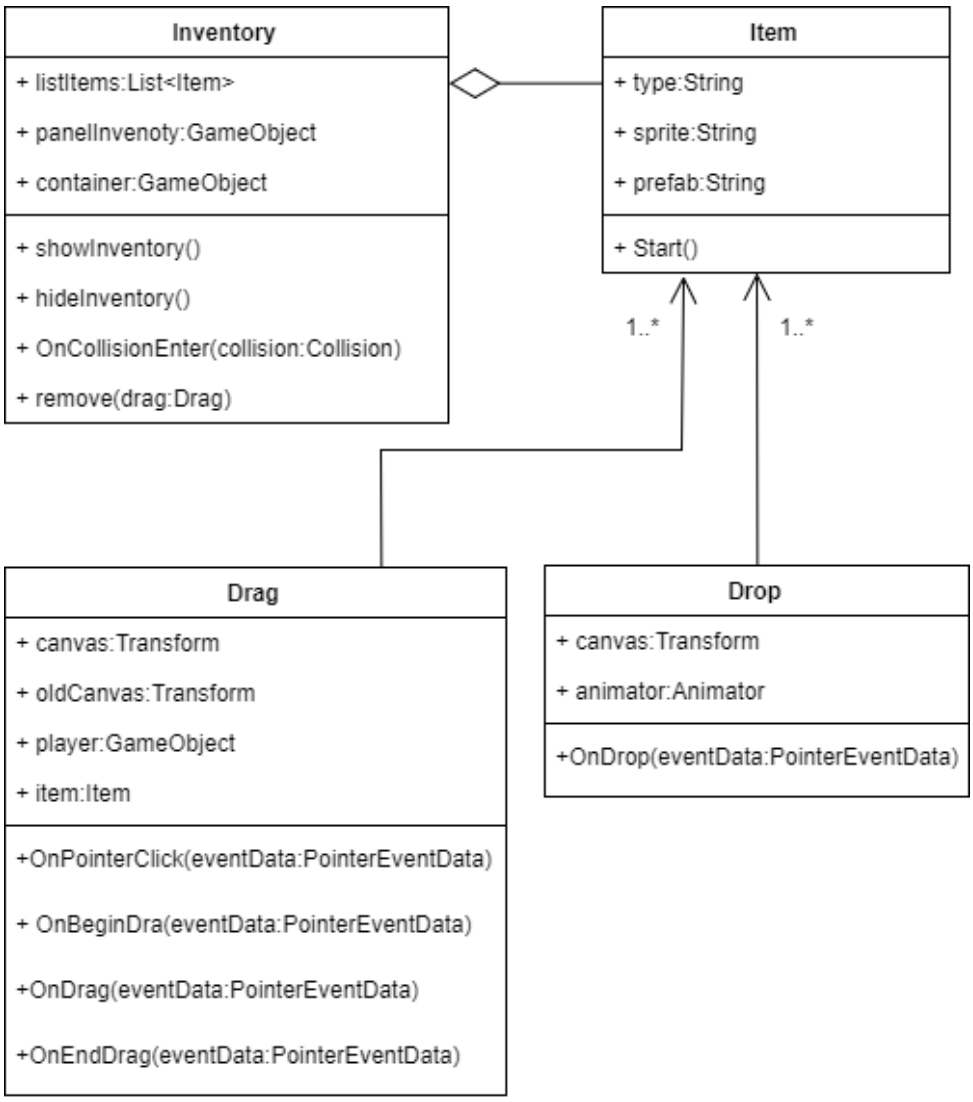

Рисунок 21 – Диаграмма классов системы инвентаря

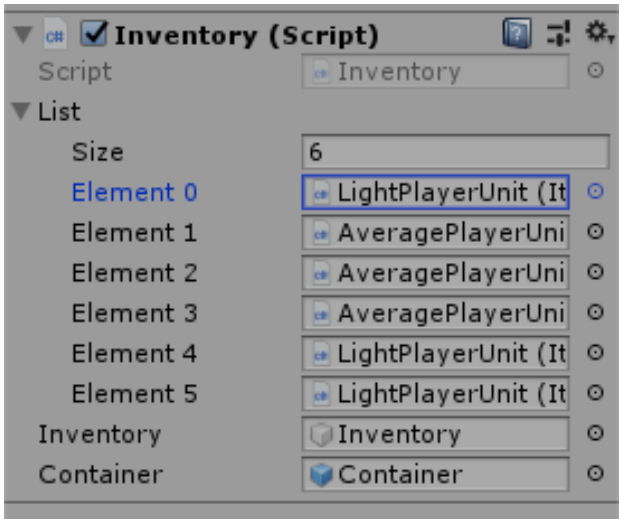

Рисунок 22 – Класс Inventory в инспекторе движка

Данные методы вызываются нажатием кнопки на главном экране либо же нажатием клавиши I на клавиатуре, обработка нажатия находится в стандартном методе update, данный метод определен во всех классах, наследуемых от встроенного в Unity класс MonoBehavior.

Поскольку необходима реализация перетаскивания юнитов из инвентаря на панель текущей команды, необходима реализация системы drag-and-drop. Для этой цели служат соответствующие классы Drag, обрабатывающий взятие предмета курсором и процесс перетаскивания и Drop, который обрабатывает завершение перетаскивания и передачу предмета в другую панель.

Все объекты, которые хранятся в инвентаре, находятся в массиве из класса Inventory. На сцене же хранится контейнер, отображающий только иконку хранимого объекта и ссылку на сам объект в папке Resources из которой может быть загружен в любой момент используя идентификатор в виде названия.

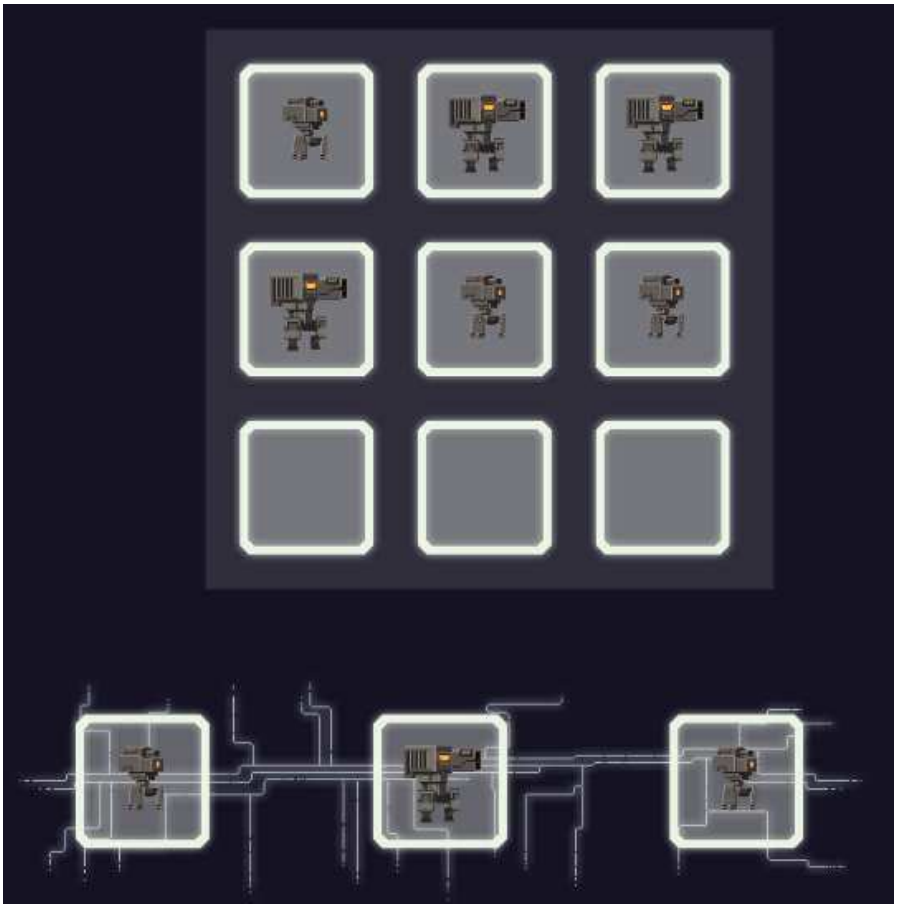

Рисунок 23 – Система инвентаря

ЗАКЛЮЧЕНИЕ

Помимо непосредственно программирования, при разработке компьютерной игры учитываются еще множество других составных частей, такие как звуковое оформление, графический дизайн, геймдизайн. Это комплексная разработка, требующая всестороннее развитие во множестве направлений. Но именно поэтому разработка игровых проектов и является настолько трудоемкой и интересной задачей.

В результате проделанной работы было спроектирован и реализован игровой проект в жанре rogue-like с процедурной генерацией локаций и искусственным интеллектом для ориентации на них.

Данный проект планируется дорабатываться посредством улучшения интерфейса и дополняясь новым контентом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Эндрюс, Дж. Проектирование архитектуры параллельного игрового движка. [Электронный ресурс]. – Режим доступа: http://www.uraldev.ru/articles/56/page/1 (дата обращения: 27.04.2019)
2. Анатомия Игровых Движков [Электронный ресурс]. – Режим доступа: https://www.extremetech.com/computing/50938–game–engine–anatomy–101–part–i/ (дата обращения: 27.04.2019)
3. Как создать Roguelike [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/428620 (дата обращения: 27.04.2019)
4. FTL: Faster Than Light [Электронный ресурс]. – Режим доступа: h https://store.steampowered.com/app/212680/FTL_Faster_Than_Light (дата обращения: 27.04.2019)
5. UnrealEngine. [Электронный ресурс]. – Режим доступа: https://www.unrealengine.com/en–US/what–is–unreal–engine–4 (дата обращения: 27.04.2019)
6. CryENGINE [Электронный ресурс]. – Режим доступа: habrahabr.ru/company/ua–hosting/blog/279529/ (дата обращения: 27.04.2019)
7. Unity [Электронный ресурс]. – Режим доступа: http://howtomakeavideogame.net/overview–gaming–engines–unity3d–unreal/ (дата обращения: 27.04.2019)
8. Source2 [Электронный ресурс]. – Режим доступа: http://www.hl–inside.ru/source/ (дата обращения: 27.04.2019)
9. Процедурная генерация в 2D играх [Электронный ресурс]. – Режим доступа: http://vc.ru/dev/10292-unity-world (дата обращения: 27.04.2019)
10. Алгоритм Ли [Электронный ресурс]. – Режим доступа: http:// su-vitruf.ru/2012/05/13/1176/volnovoj-algoritm-algoritm-li/ (дата обращения: 25.04.2019)

11. Создание искусственного интеллекта для игр. [Электронный ресурс]. – Режим доступа: https://software.intel.com/ru–ru/articles/designing–artificial–intelligence–for–games–part–1 (дата обращения: 27.04.2019)

12. Применение искусственного интеллекта к игре. [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru–ru/magazine/mt736456.aspx (дата обращения: 27.04.2019)

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
_____ О.В. Непомнящий
подпись инициалы, фамилия
« 27 » 06 2019 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника
код и наименование направления

Стратегическая игра с применением адаптивного искусственного интеллекта
тема

Руководитель	25.06.19 подпись, дата	ст. преподаватель должность, ученая степень	И.В.Матковский инициалы, фамилия
Выпускник	25.06.19 подпись, дата		С.А.Полонников инициалы, фамилия
Консультант	25.06.19 подпись, дата	доцент, канд.тех.наук. должность, ученая степень	Л.И.Покидышева инициалы, фамилия
Нормоконтролер	26.06.19 подпись, дата	доцент, канд.тех.наук. должность, ученая степень	В. И. Иванов инициалы, фамилия

Красноярск 2019