МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

Направление подготовки/специальность
09.03.01 Информатика и вычислительная техника ”

направленность (профиль)/специализация
“Технологии разработки программного обеспечения”

Выпускная квалификационная работа

Обучающегося 4 курса
очной формы обучения
Афанасьева Андрея Дмитриевича

Руководитель выпускной квалификационной работы:
кандидат педагогических наук, доцент,
Атаян Ануш Михайловна

Рецензент:
Ученая степень *(при наличии)*, ученое звание *(при наличии)*, должность
Ф. И. О. *(указывается в именительном падеже)*

СОДЕРЖАНИЕ

ВЕДЕНИЕ

Индустрия компьютерных игр появилась недавно, всего около 30 лет назад, но уже смогла развиться в огромную отрасль с поистине колоссальными доходами в несколько миллиардов долларов год. Понять подобный взрывной рост популярности виртуальных развлечений достаточно просто: все это благодаря широкому распространению компьютерных технологий, в том числе и всемирной сети «ИНТЕРНЕТ». Благодаря этому, в отличие от других видов развлечений, компьютерные игры намного доступнее для конечного пользователя. Для того чтобы просто поиграть пользователю нужно иметь только компьютер или игровую приставку, а так же копию самой игры. С развитием интренета, и интернет площадок, отпала необходимость в наличии физической копии, достаточно пары кликов на торговой площадке. Более того, потребителю не требуется иметь особых знаний для того, чтобы выбрать подходящую для него игру, в то время как для подавляющего большинства других видов развлечений, необходимо разбираться как минимум в экипировке, необходимой для времяпрепровождения. Так же стоит принять во внимание, что компьютерные игры в последнее время перестали позиционироваться как программы только для отдыха и развлечений. К примеру, сегодня, благодаря использованию игровых технологий, создаются специальные комплексы по симуляции, которые помогают в обучении различных специалистов в разных областях: от лесорубов, до пилотов реактивных самолетов.

В настоящее время на территории нашей страны, как и в большинстве других стран постсоветского пространства, развита крайне слабо. Связано это с тем, что культура компьютерных развлечений пришла к нам слишком поздно и практически не развивалась. Из-за этого, даже при достаточно высоком спросе, мы имеем слишком малое количество компаний-разработчиков, которые могли бы составить конкуренцию зарубежным компаниям. Поэтому развитие технологий в данном направлении можно считать одним из наиболее перспективных.

Объектом исследования является разработка компьютерной игры.

Предмет исследования: технологии разработки компьютерной игры в жанре приключение.

Жанр приключения является одним из наиболее увлекательных и популярных среди геймеров всех возрастов, так как он позволяет почувствовать себя героем настоящего приключения, где от каждого твоего решения зависит исход всей игры.

Цель выпускной квалификационной работы - разработать прототип компьютерной игры жанра 2D-платформер средствами Game Maker Studio 2.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) выбрать жанр, вид и платформу для компьютерной игры;

2) разработать сценарий, концепцию основных элементов;

3) выбрать и изучить средство реализации;

4) реализация прототипа игры.

ТЕОРИТИЧЕСКАЯ ЧАСТЬ

1.1 Исследование понятия и классификация компьютерных игр

Компьютерные игры - это программное обеспечение, созданное для развлечения и взаимодействия игроков с виртуальным миром через персональные компьютеры или другие устройства, такие как консоли для игр, смартфоны или планшеты. Эти игры могут иметь различные жанры, темы и уровни сложности.

Игроки могут играть в компьютерные игры в одиночку или в группах с другими игроками, как на локальном уровне, так и через интернет. Многие игры имеют определенную цель, такую как борьба за выживание, сражения, выполнение заданий и т.д. Часто в играх возможностей прокачать персонажа, купить новые предметы и совершенствовать свои навыки.

Компьютерные игры также могут быть использованы для обучения и развития различных навыков, включая улучшение реакции, координации и памяти. Некоторые игры также могут учить игроков новым технологиям, программированию, математике и другим наукам.

Компьютерные игры оказали столь существенное влияние на общество, что в последнее время в информационных технологиях появилась небольшая, но устойчивая тенденция к геймификации некоторых процессов. Так, например, в некоторых европейских школах начали использовать известную игру Minecraft для обучения, а для нужд армий начали создавать различные специальные симуляторы для тренировок и обучения солдат. Компьютерные игры так же с 2011 года официально признаны правительством США и американским Национальным фондом отдельным видом искусства, наряду с театром и кино, а в России киберспорт официально был причислен к видам спорта.

Все это показывает, что компьютерные игры плотно влились в нашу жизнь, и постепенно становятся неотъемлемой её частью. Более того, сфера их использования за последние 10 лет сильно выросла, теперь игры используют не только для развлечения и отдыха, но и для обучения людей и проведения научных исследований.

Компьютерные игры классифицируют по нескольким основным признакам:

1)Жанр

2)Количество игроков играющих в игру

3)Визуальный стиль

4)Тип игровой платформы

Жанр компьютерной игры - это классификация игры, основанная на общих характеристиках ее геймплея, сюжета, механики, настроения и графического оформления. Каждый жанр имеет свои особенности и типичные элементы, которые могут определять преимущественную целевую аудиторию.

Таблица 1 – жанры компьютерных игр

Жанр	Особенности	Поджанры
Аркада	Игры с примитивным, и простым геймплеем	Платформер, файтинг
Приключенческая игра	В основе игры лежит приключение героя	Квест, визуальная новелла, Roguelike
RPG (Ролевая игра)	Игрок практические не ограничен в своем выборе действий, и может отыгрывать ту роль, которую хочет	Тактическая RPG, Экшен RPG, JRPG
Экшен	Игрок большую часть находится в напряжении, и в центре событий.	Шутер, Слешер.
Стратегия	Компьютерные игры, в которых игроку необходимо планировать и управлять ресурсами, армией и территориями, разрабатывать тактику и стратегию	Пошаговая, в реальном времени, глобальная
Симулятор	Игры, которые пытаются имитировать какой-либо определенный процесс или действие, например, авиасимулятор, фермерский симулятор	
Головоломки	Игры, в которых игрокам нужно решать различные логические задачи и головоломки, используя	

	свой интеллект и креативность.	

Представленные здесь жанры нельзя назвать полными, так как в последнее время стали появляться игры собственных жанров, которые можно отнести как к одному из представленных здесь жанров, так и к самостоятельному отдельному от них.

По количеству игроков игры делятся на два типа:

Одиночные

Многопользовательские

Отличие одиночных игр от многопользовательских состоит в том, что в одиночных играх игрок играет сам, без участия других людей, а в многопользовательских играх игроки играют вместе или против друг друга, общаясь и взаимодействуя внутри игрового мира. В одиночных играх часто предлагается занимательный сюжет, требующий от игрока решения головоломок и выполнения заданий, тогда как в многопользовательских играх акцент делается на взаимодействии между игроками и развитии персонажа, а игроки могут соревноваться друг с другом или работать в команде для достижения общих целей.

По визуальному стилю игры так же отличаются. Существуют следующие стили:

- Текстовые

Текстовые игры - это игры, в которых взаимодействие между игроком и игровым миром осуществляется через текстовый интерфейс. Они являются одним из первых типов компьютерных игр и наиболее распространены в виде текстовых квестов.

- 2D игры

2D игры - это компьютерные или мобильные игры, которые используют двумерную графику, то есть игровое поле и персонажи отображаются на плоскости без трехмерных эффектов. Такие игры включают в себя платформеры, аркады, квесты, RPG и многие другие жанры. Они могут иметь разную

сложность, стиль и уровень графических возможностей, но в целом, 2D игры отличаются от 3D игр более простым оформлением и более доступным игровым процессом.

- 2,5D игры.

Как и 2D игры, 2,5D игры в основном разрабатываются в двухмерном пространстве. Однако 2,5D игры могут включать в себя добавочные элементы трехмерной графики, такие как различные углы камеры и эффекты глубины. Это может создать ощущение трехмерного мира внутри игры.

- 3D игры.

3D игры - это игры, в которых объекты, персонажи и окружение созданы в трехмерном пространстве и могут двигаться во всех направлениях, а игрок может управлять камерой для просмотра игрового мира с разных углов. Также в 3D играх обычно применяются более сложные и реалистичные графические эффекты, такие как освещение, тени, отражения и т. д.

- Псевдо 3D игры.

Псевдо 3D игры - это игры, которые имитируют трехмерную графику, но на самом деле они создаются в двумерном пространстве. Игрок видит сцену со стороны или сверху и может двигаться влево-вправо, вверх-вниз, но не может изменять точку зрения и приближаться/отдаляться от объектов на экране.

По типам игровой платформы игры различаются:

• персональные компьютеры;

• игровые приставки/консоли;

• мобильные телефоны;

• мультиплатформенные;

К сожалению, представленная здесь классификация не является полной и может быть дополнена. Так, на пример, к жанрам можно добавить такой специфичный вид игр как ритм игры, где основное внимание уделяется музыке и звукам, а многопользовательские игры в свою очередь могут делиться на несколько подкатегорий. Но так же стоит учитывать, что данной классификации достаточно для определения большинства из существующих игр, так как на

сегодняшний день не существует однозначной и полной классификации для компьютерных игр. Это происходит из-за того, что многие игры нельзя отнести к каким-либо критериям. Игра может соответствовать сразу нескольким жанрам, выйти сразу на нескольких платформах или иметь как однопользовательский режим игры, так и многопользовательский.

Все это обусловлено тем, что игровая индустрия, в отличие от других сфер развлечений, появилась сравнительно недавно. Но как уже было сказано, данной классификации достаточно, для того что бы определить большинство компьютерных игр существующих на данный момент.

Основываясь на данной классификации компьютерных игр было принято решение разработать двумерный однопользовательский платформер для персональных компьютеров, в жанре приключение, с поджанром Roguelike. Жанр игр Roguelike (также называемый Dungeon Crawl) представляет собой вид ролевых игр, основанный на первых версиях игры Rogue, созданной в 1980 году. В Roguelike игрок управляет персонажем, который следует на случайно генерируемую карту подземелья, сражаясь с монстрами, собирая сокровища и преодолевая различные препятствия.

В отличие от других RPG, важными элементами игры являются высокий уровень сложности, будучи одной из причин непредсказуемости прохождения игры, и постоянная смертность персонажа: если персонаж погибает, то игрок начинает игру сначала с новым персонажем. Кроме этого, у игрока ограниченный запас ресурсов, что делает каждое наступление на врага или использование предмета рискованным и необдуманным действием. В Roguelike игрок должен стараться сохранить максимально возможное количество жизней и ресурсов, чтобы продвигаться и побеждать все более сложных врагов и проходить больше подземелий.

Решение о создании игры такой направленности было принято по следующим причинам:

-Жанр приключение хорошо сочетается с 2D играми платформерами;

-Создание игр на 2D основе куда менее трудо- и ресурснозатратно, нежели 3D;

-Подобное сочетание легко реализуемо в рамках игры, даже для непотного разработчика;

-Использование жанра Roguelike позволяет разнообразить будущий игровой процесс, так как каждая новая игра, и вправду будет новой.

1.2 Алгоритм реализации проекта

Алгоритм разработки компьютерной игры имеет мало отличий от алгоритма разработки любого иного программного продукта и состоит из 2 больших этапов:

1) проектирование;

2) разработка;

На этапе проектирования определяются цель игры и средства её разработки.

Для придания направления игре, важны идея, жанр и сеттинг. Связь идеи и жанра очень тесна, так как идея инспирирует игрока на игру. Например, идея ролевой игры заключается в возможности игрока играть, как хочет, а идея шутера заключается в участии игрока в реальных или вымышленных боевых действиях. Жанр игры легко подбирается после определения основных идей. После выбора жанра и идеи, следующим шагом является выбор сеттинга игры. Сеттинг составляет место, время и условия действий игры. Выбор сеттинга должен опираться на вкус целевой аудитории, так как это может упростить процесс создания сценария. Важную роль в процессе разработки игры играют программный код и игровой движок. Выбор подходящего кода и движка влияет как на скорость работы игры, так и на ее долговечность в будущем. Выбор языка программирования зависит от платформы, на которой будет выпущена игра: если это браузер, то лучше использовать Java или Flash, а если это персональный компьютер, то лучшей опцией будет C#.

Функция игрового движка заключается в управлении основными аспектами игры, такими как физика объектов, графические эффекты и т.д. При выборе игрового движка разработчики должны учитывать состоятельность выбора языка программирования и доступность движка. Например, Unity позволяет создавать

игры на C# и предоставляется бесплатно, если средний доход компании не превышает $100,000 в год.

После выбора целей и инструментов разработки, фаза реализации проекта начинается с определения сюжета и игровой механики. Игровая механика основывается на целях игры, определяя необходимые объекты и правила взаимодействия между ними и игроком. Параллельно с разработкой игровой механики разрабатывается сюжет игры, который является важным фактором в заинтересованности игрока. Сюжет может быть в литературном или режиссерском форматах, первый описывает события и персонажей, а второй подробно описывает уровни и события на них. Так как один из жанров игры Roguelike, можно ограничиться малым сюжетом, так как игры данного жанра больше сконцентрированы на геймплее, нежели на сюжете.

Кроме того, на этой стадии начинается работа графических дизайнеров для создания концепт-артов, проектных изображений, на основе которых будет прорабатываться внешний вид игры и ее главных персонажей.

И, наконец, самая важная часть – разработка игры.

Основываясь на концепт артах, и набросках сюжета, начинается разработка уровней, и тестирование основных механик. В случае же, если уровень генерируется по ходу игры, а не создан заранее, то происходит отладка и настройка генератора уровней, что бы исключить нереальные к прохождению комнаты, но при этом и сохранить сложность.

Когда основные механики готовы и протестированы, собирается прототип игры, чтобы посмотреть как механики будет работать вместе, какие ошибки или баги могут возникнуть, и по необходимости внести исправления.

1.3 Подбор игрового движка

Игры разрабатываются с помощью игровых движков и графических редакторов. Игровые движки обеспечивают правильное функционирование игры и включают в себя все необходимые алгоритмы и инструменты для ее разработки. Существует множество игровых движков, отличающихся поддерживаемыми языками программирования, функциональностью и стоимостью лицензии.

Одними из самых ярких примеров являются:

-Game Maker Studio 2 один из самых популярных движков, используемых для разработки двухмерных игр для различных платформ. Этот движок использует язык программирования GML (Game Maker Language) и имеет несколько версий, каждая из которых предназначена для определенной платформы. GML (Game Maker Language) - это язык программирования, который был создан как часть программного обеспечения Game Maker, используемого для создания визуальных и простых игр. Язык GML изначально использовался для создания игр, однако сегодня он может использоваться для различных задач, связанных с разработкой приложений.

GML очень похож на язык программирования C++, он использует многие из тех же основных конструкций, таких как условные операторы, циклы и переменные. Другими словами, GML позволяет программистам создавать игры с помощью кода, что позволяет им управлять каждым аспектом игрового процесса.

Одним из преимуществ GML является простота и интуитивный интерфейс, что делает его доступным как начинающим разработчикам, так и опытным профессионалам. Он также обладает большой гибкостью и позволяет создавать игры с различной сложностью.

-Unreal Engine это специализированный игровой движок, на основе которого создаются многие современные видеоигры. Unreal Engine разрабатывается и поддерживается компанией Epic Games, и он представляет собой мощный инструмент для создания реалистичных игровых миров, персонажей, анимаций, света и многого другого.

Unreal Engine имеет набор готовых компонентов и инструментов для разработки игровых проектов, таких как визуальный редактор Blueprints, графический редактор материалов, систему физики, аудио-систему, инструменты для создания многоэкранных игр, совместную разработку и др.

Unreal Engine также предлагает большое количество документации, уроков, ресурсов и сообщества для разработчиков игр, что делает его очень доступным

для начинающих. Кроме того, Unreal Engine поддерживает множество платформ, включая Windows, MacOS, Linux, iOS, Android, Xbox, PlayStation и др.

Ключевыми преимуществами Unreal Engine являются графические возможности, которые позволяют создавать красивые и реалистичные игровые миры, а также высокая производительность, что позволяет запускать игры на широком диапазоне устройств и платформ.

Unity - это кроссплатформенный игровой движок, разработанный компанией Unity Technologies. Он позволяет создавать игры, как для настольных систем, так и для мобильных устройств и веб-браузеров.

Unity использует язык программирования C# для создания игровой логики и позволяет использовать множество графических API, включая DirectX, OpenGL и Vulkan. Он также имеет встроенный редактор, в котором можно создавать игровые объекты, настраивать их свойства и создавать сцены.

Средства создания интерактивных миросозданий помогают расширить возможности вашей игры add-on'ами. Различные пакеты игровых движков позволяют сделать игры для многих платформ, добавить платежи в игру и многое другое.

Unity также предоставляет широкий спектр инструментов для создания игровых анимаций, включая автоанимацию, настройку параметров анимации и поддержку рендеринга 2D и 3D.

Кроме того, Unity имеет много возможностей для создания многопользовательских игровых приложений с помощью его функции сетевого взаимодействия и встроенной поддержки многопользовательских игр.

Знание Unity в настоящее время является одним из ключевых навыков для разработки игр и приложений в индустрии разработки игр.

Среди представленных движков, выбор был сделан в пользу Game Maker Studio 2, так как он наиболее подходит под цели работы. Он изначально был направлен на создание 2D игр, имеет встроенный графический редактор, что позволит создавать спрайты сразу внутри движка, не прибегая к иным приложениям, его язык прост и понят для освоения, а схожесть с C++ ещё больше

облегчает работу с ним, в случае если пользователь был знаком с ним до начала разработки.

1.4 Сценарий

Для начала работы над проектом, также необходимо написать сценарий игры. На его основе будет разработан концепт проекта.

Главный герой приходит в себя в подземелье. Очнувшись, герой не помнит как сюда попал. В поисках выхода он продвигается по подземелью и встречает противника. В результате столкновения главный герой погибает. После смерти, герой вновь приходит в себя в подземелье, но уже в другой его части. Там его встречает старец, который рассказывает, что происходит в мире подземелья. В мире игры, есть две противоборствующих силы, Хаос и Порядок. После давней битвы, победителем которой оказался Порядок, в мире наступило спокойствие, но Хаос не был побежден до конца, и все это время он копил силы для повторного сражения. Из за прихоти судьбы ни Хаос, ни Порядок не могут начать сражение напрямую, а только косвенно, через своих последователей и сторонников. Последователи Хаоса, смогли провести один ритуал, который позволил их господину воздействовать на мир, и создать подземелье, наполненное его силой, из-за чего попытка пройти его обречена на провал, ведь каждый раз, подземелье будет новым, наполненное новыми противниками, а значит и подготовиться к прохождению не выйдет. Тем не менее, Порядок все же нашел чем ответить. Он наделил одного из сторонников своей силой, что позволяет ему каждый раз после смерти возвращаться к живым и пытаться вновь пройти подземелье, используя артефакты, что остаются от монстров Хаоса после их гибели.

2.1 Техническое задание

Назначение разработки

Дипломная работа «Разработка компьютерной игры в жанре приключение» является комплексным проектом, охватывающим различные аспекты разработки программного обеспечения, содержит все основные аспекты компьютерной приключенческой игры и является продуктом сферы компьютерных развлечений.

2.2 Требования к программе или программному изделию

2.2.1 Требования к функциональным характеристикам

Данный проект является компьютерной игрой, вследствие чего предусматривается одна категория пользователей - игроки. В процессе работы приложения пользователь является непосредственным участник игрового процесса и оказывает непосредственное влияние на него.

Программы должна обладать следующим функционалом:

а) графический функционал:

1) выбор полноэкранного или оконного режима;

б) звуковой функционал:

1) регулировка общей громкости;

2) регулировка громкости музыки;

3) регулировка громкости внутриигровых звуков;

в) внутриигровой функционал:

1) система развития игрока;

2) система взаимодействия игровых объектов;

3) боевая система;

г) интерфейс пользователя:

1) переходные сцены (вступительная, финальная, экран загрузки);

2) главное меню;

3) графический интерфейс пользователя.

2.2.2 Требования к входным и выходным данным

Входными данными в компьютерной игре являются игровые настройки пользователя, а также непосредственное управление во время игрового процесса с помощью компьютерной мыши и клавиатуры. Проект относится к играм в реальном времени, где в отличие от пошаговых - действия игрока незамедлительно оказывают влияние на игровой процесс.

Выходными данными являются графическая интерпретация игрового процесса на мониторе игрока и звук, сопровождающий его. Действия игрока

влияют на игровой процесс и текущее состояние игровой сцены. Игрок контролирует игрового персонажа с помощью интерфейса пользователя.

2.2.3. Требования к надежности

В программе должна присутствовать проверка входной информации на соответствие типов, принадлежность диапазону допустимых значений и соответствие структурной корректности. В случае возникновения ошибок предусмотреть возможность вывода информативных диагностических сообщений.

2.2.4. Требования к составу и параметрам технических средств

Минимальные системные требования:

– ОС (операционная система): Windows 7/8/10;

– Процессор: Intel Core 2 Duo @ 3.0 Ghz / AMD Athlon 64 X2 6000+;

– Оперативная память: 1 Gb;

– Жесткий диск: 10 Gb свободно;

– Видео память: 512 Mb;

– Видео карта: nVidia GeForce 9800 / AMD Radeon HD 4870;

– Звуковая карта: Совместимая с DirectX;

– DirectX 9.0c;

– Клавиатура, Мышь.

Рекомендуемые системные требования:

- ОС (операционная система): Windows 7/8/10;

- Процессор: Intel Core i5 @ 3.2 GHz / AMD Phenom II X4 @ 3.6 GHz;

- Оперативная память: 2 Gb;

- Жесткий диск: 10 Gb свободно;

- Видео память: 1 Gb;

- Видео карта: nVidia GeForce GTX 460 / AMD Radeon HD 5850;

- Звуковая карта: Совместимая с DirectX;

- DirectX 11;

- Клавиатура, Мышь.

2.2.5. Требования к информационной и программной совместимости

Программа должна функционировать под управлением ОС семейства Windows следующих версий: Windows 7, 8, 10. В приложении используются библиотеки платформы.NET Framework. Также требуется установленный DirectX 9.0c или более поздней версии.

2.2.6. Требования к программной документации

Программная документация должна быть представлена руководством пользователя.

ПРАКТИЧЕСКАЯ ЧАСТЬ

3 Разработка проекта

3.1 Подготовка

Перед началом разработки необходимо становить Game Maker Studio 2. Сделать это можно, скачав программу с официального сайт разработчика (рис. 1).

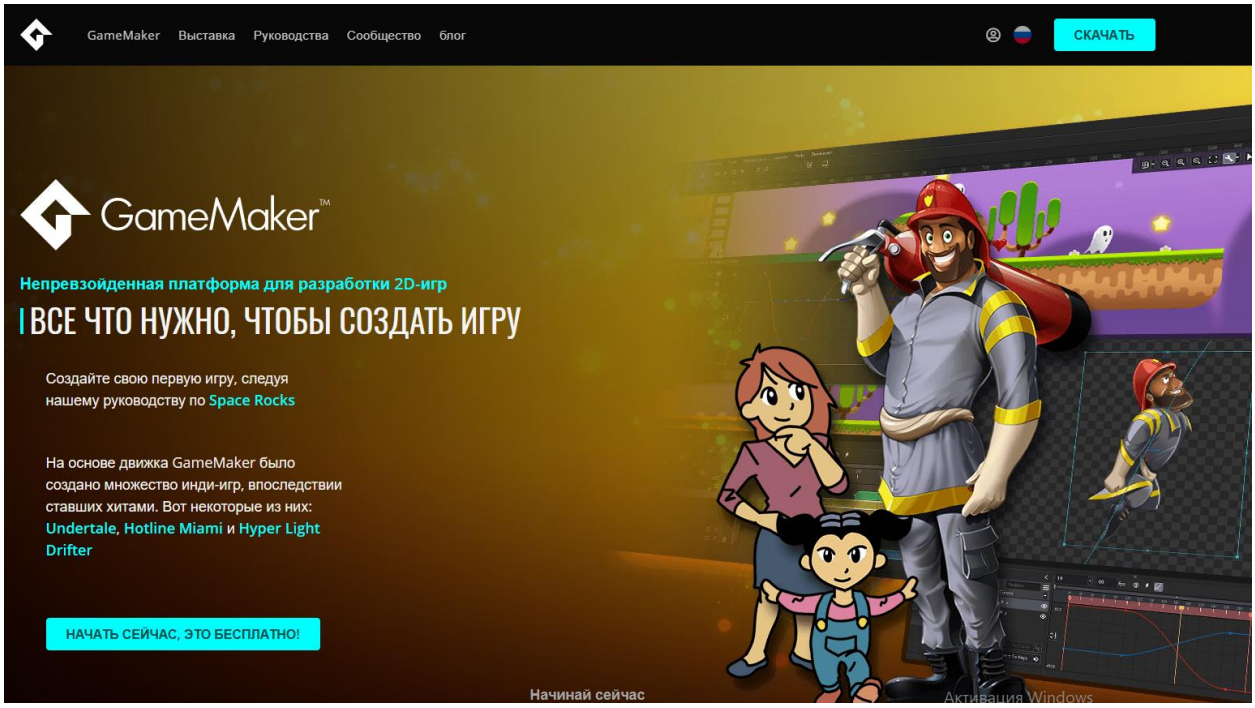

Рисунок 1 – главная страница официального сайта Game Maker

Процесс установки ни чем не отличается от установки других продуктов. Нам достаточно просто открыть файл установщика, указать путь, куда установить приложение, и ожидать конца установки.

При открытии программы нам будет необходимо войти в свой аккаунт, который можно либо тут же создать, либо использовать ранее созданный (рис. 2).

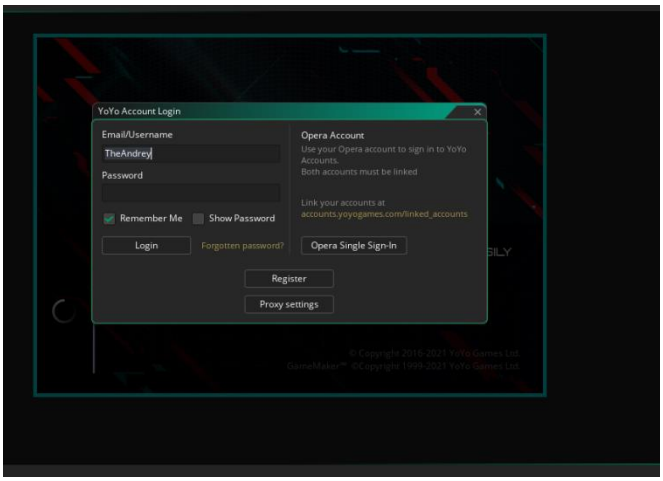

Рисунок 2 – вход в аккаунт

После входа в аккаунт, вам будет предложено создать новый проект, или загрузить уже созданный (рис. 3).

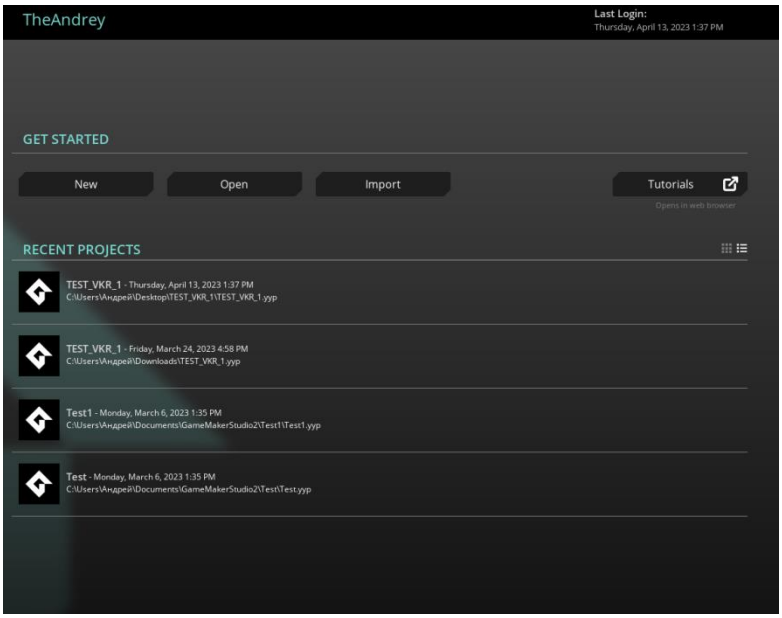

Рисунок 3 – окно проектов

После открытия проекта, Вас перенесет на рабочее пространство, большая часть которого поле отображения того, с чем вы работаете в данный момент. На правой боковой панели находится список папок с файлами внутри них. Стоит заметить, что все необходимые папки созданы заранее, но ничто не мешает Вам создать свои (рис. 4).

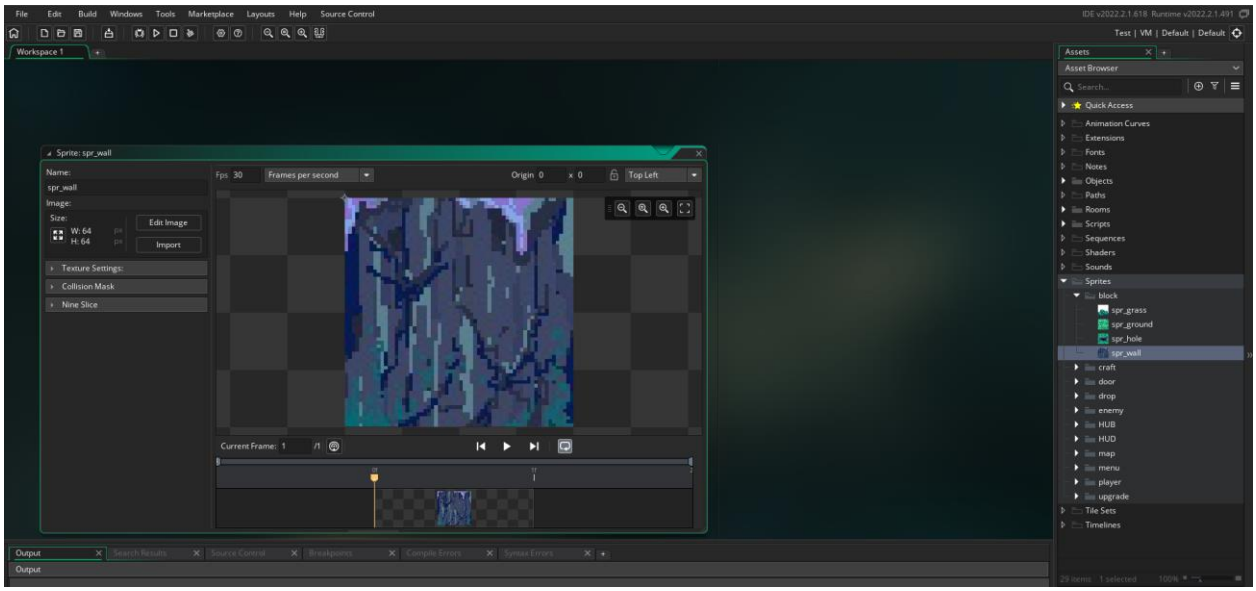

Рисунок 4 – интерфейс программы

Для создания чего либо, необходимо кликнуть правой кнопки мыши по папке, куда вы хотите расположить ваш файл. Можно использовать как заранее готовые папки, так и свои. По клику мышки, у вас откроется окно со списком, из которого и необходимо выбрать нужный вам файл (рис. 5).

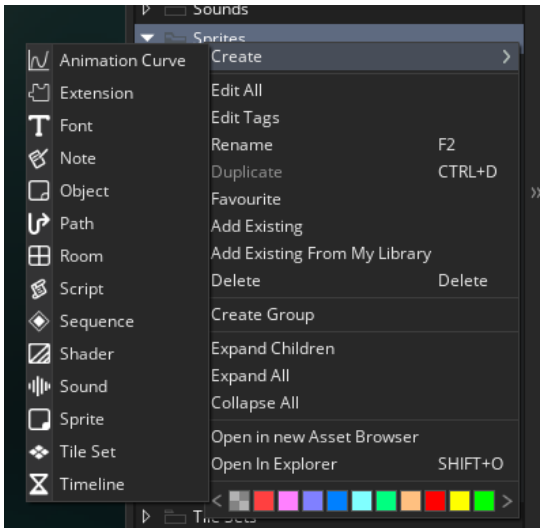

Рисунок 5 – окно создания файла

Как только вы создадите файл, на пример спрайт, у вас отроется окно работы с ним, в поле для отображения. В зависимости от типа файла, что Вы создали, окно может отличаться как по виду, так и по функционалу (рис. 6).

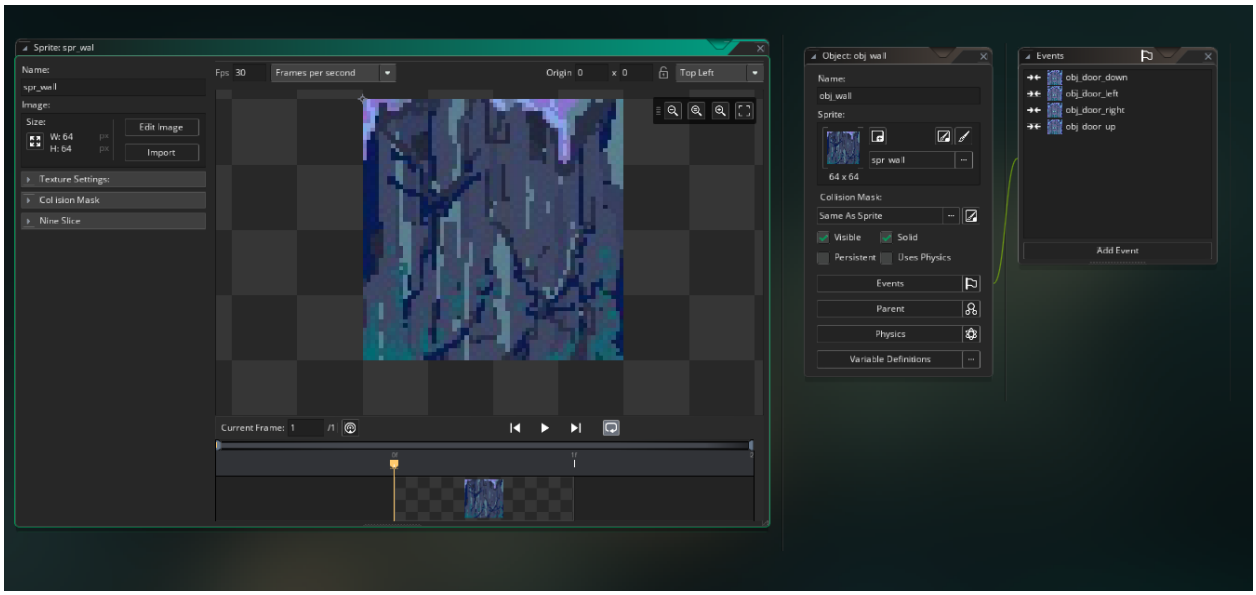

Рисунок 6 - открытые рядом окна Спрайта и Объекта

Для начала работы над игрой, нужно подготовить несколько изначальных спрайтов и объектов, что будут использоваться для первых тестов и проверок.

Создаем 5 спрайтов, и столько же объектов. Над внешним видом можно не работать, так как сейчас это тестовые заменители (рис. 7).

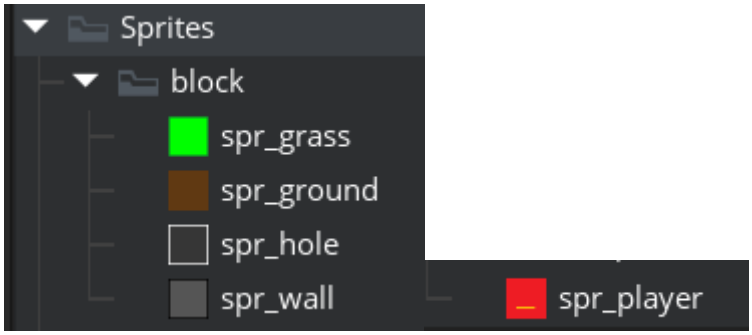

Рисунок 7 – тестовые спрайты

Спрайты из папки block, спрайты окружения, то есть статичные и неподвижные объекты, в то время как spr_player спрайт уже игрока, но на данный момент без анимации. После создания спрайтов, выдаем их как маску для объектов, и даем каждому соответствующее имя (рис.8).

Рисунок 8 – тестовые объекты

Отличие спрайта от объекта в том, что спрайт это по сути своей просто картинка, а вот объект, уже то, что может взаимодействовать с игровым пространством, и с игроком. Как только приготовления завершены, переходим к следующей части.

3.2 Создание уровня

Игра должна представлять набор комнат, которые собираются в один из этажей подземелья. И сам план этажа, или же уровня, и план комнаты генерируется случайно. Следовательно, необходимо создать скрипт, который при запуске игры будет либо создавать новый уровень, либо загружать существующий.

Первым действием определяем размер нашего этажа, выбирая случайное число от 5 до 10. Как только это случилось, создаем сетку будущего этажа, используя двумерный массив, где его координаты, это номер комнаты, а значение ячейки, её содержимое. Изначально все созданные комнаты получают значение none, что бы в будущем было проще взаимодействовать с ними. Как только сетка этажа сделана, создаем по его кроям границу, что бы при генерации не выйти за пределы, и не вызвать ошибки у игры. Для этого, все ячейки расположенные по краям этажа меняют свое значение с none на board. По завершению создания границ, скрипт переходит в центральную комнату, делая её стартовой, а так же высчитывая число комнат, которые будут полноценно построены. Делается это, умножением размера этажа на 2. Используя подобный способ, мы никогда не построим больше комнат, чем было заготовлено сеткой, так как этаж квадратный, и в случае выпадения 5, общее число комнаты 5x5, что

дает нам сетку на 25 комнат, из которых 10 будут построены и собраны во едино. Далее скрипт начинает постепенно создавать комнаты, опираясь на многие параметры. Нельзя создать комнату отдельную от остальных, а значит хотя бы один сосед не пустота быть должен. Так же нельзя построить комнату вместо границы, и кроме того, следует учитывать Roguelike составляющую, строить или не строить комнату надо решать, опираясь на условный бросок монетки. По итогу, мы получаем случайную, но вместе с тем и не создающую невозможных путей систему (рис. 9).

```
function level_creat()
{
    ini_open("GAME.ini")
    randomize()
    global.num_of_room = irandom_range(5,10)
    for (i = 0; i <= global.num_of_room; i += 1)
        {
            for (m = 0; m <= global.num_of_room; m +=1)
                {
                    global.floor[i,m] = "none"
                }
        }
    ini_write_real("DATA","num_of_room",global.num_of_room)

    i = 0
    for (m = 0; m <= global.num_of_room; m += 1)
    {
        global.floor[i,m] = "Board"
    }
    i = global.num_of_room
    for (m = 0; m <= global.num_of_room; m += 1)
    {
        global.floor[i,m] = "Board"
    }
    m = 0
    for (i = 0; i <= global.num_of_room; i += 1)
    {
        global.floor[i,m] = "Board"
    }
    m = global.num_of_room
    for (i = 0; i <= global.num_of_room; i += 1)
    {
        global.floor[i,m] = "Board"
    }



    num_pol = round(global.num_of_room/2)
    ini_write_real("DATA", "NOW_i", round(global.num_of_room/2))
    ini_write_real("DATA", "NOW_m", round(global.num_of_room/2))
    global.floor[num_pol,num_pol] = "start"
    num_of_build = global.num_of_room * 2
```

Рисунок 9 – код генератора уровня

3.3 Создание комнаты

За генератором уровня, необходимо создать генератор уже отдельной комнаты. Механизм генерации комнат схож с генератором уровня, но куда более сложный и большой. Стоит отметить, что смены комнат как таковой нет, когда игрок покидает комнату, он просто пересоздает уже существующую, сохраняя данные прошлой.Как и с генератором уровня, комната так же начинается с массива, но если в прошлом генераторе номер ячейки массива, был номером комнаты, то в этом случае все сложнее. Каждая комната в Game Maker Studio 2,

изначально разбита на координатную сетку, причем нулевая координата находится в левом вернем углу, и, опускаясь по оси y, мы его не уменьшаем, а увеличиваем. В данном генераторе номер ячейки массива, это координата комнаты, то есть, можно сказать что вся комната была перенесена в массив, которой уже и обрабатывается, но так как размер спрайтов, а значит и объектов 64x64, то и номер имеет не стандартный вид, а номер координат кратных 64. Это позволяет нам уже после обработки массива, и заполнения комнаты на его основе ровно расставить все объекты, не наслаивая, их друг на друга. Но перед полноценной обработкой массива, как и с этажом, необходимо создать границы, чтобы в комнате не было путей ведущих за её пределы в неположенном месте (рис. 10).

```
i = 0
for (m=0; m<=832; m+=64)
{
    coord = string(i) + string(m)
    instance_create_depth(i,m,0,obj_wall)
    ini_write_string(room_number,coord,"wall")
}
i = 1536
for (m=0; m<=832; m+=64)
{
    coord = string(i) + string(m)
    instance_create_depth(i,m,0,obj_wall)
    ini_write_string(room_number,coord,"wall")
}
m = 0
for (i=0; i<=1600; i+=64)
{
    coord = string(i) + string(m)
    instance_create_depth(i,m,0,obj_wall)
    ini_write_string(room_number,coord,"wall")
}
m = 832
for (i=0; i<=1600; i+=64)
{
    coord = string(i) + string(m)
    instance_create_depth(i,m,0,obj_wall)
    ini_write_string(room_number,coord,"wall")
}
```

Рисунок 10 – код создания границ

Как и в случае с генерацией этажа, ячейки, которые выступают границей, меняют свое значение на wall, остальные же клетки так и остаются none.

По завершению создания границ, наступает следующий этап, а именно расположение выходов из комнаты. Так как вести в никуда пути не могут, генератору необходимо обратиться к массиву этажа, и проверить с каких сторон существуют комнаты, и только в их направлении создать двери. А чтобы не допустить ситуацию, при которой путь к выходу из комнаты будет заблокирован,

генерируется отдельный прямой путь от двери до центра комнаты, тем самым, гарантируя возможность как войти в комнату, так и покинуть её. Все ячейки, которые попадают под этот путь, меняют свое значение на ground (рис. 11).

```
if global.floor[i+1,m] != "none" && global.floor[i+1,m] != "Board"
{
    instance_create_depth(1536,448,-1,obj_door_right)
    ini_write_string(room_number,"1536448","right")
    for (a=768;a<1536;a+=64)
    {
        instance_create_depth(a,448,0,obj_ground)
        coord = string(a) + "448"
        ini_write_string(room_number,coord,"ground")
    }
}
if global.floor[i-1,m] != "none" && global.floor[i-1,m] != "Board"
{
    instance_create_depth(0,448,-1,obj_door_left)
    ini_write_string(room_number,"0448","left")
    for (a=768;a>0;a-=64)
    {
        instance_create_depth(a,448,0,obj_ground)
        coord = string(a) + "448"
        ini_write_string(room_number,coord,"ground")
    }
}
if global.floor[i,m+1] != "none" && global.floor[i,m+1] != "Board"
{
    instance_create_depth(768,832,-1,obj_door_down)
    ini_write_string(room_number,"768832","down")
    for (a=448;a<832;a+=64)
    {
        instance_create_depth(768,a,0,obj_ground)
        coord = "768" + string(a)
        ini_write_string(room_number,coord,"ground")
    }
}
if global.floor[i,m-1] != "none" && global.floor[i,m-1] != "Board"
{
    instance_create_depth(768,0,-1,obj_door_up)
    ini_write_string(room_number,"7680","up")
    for (a=448;a>0;a-=64)
    {
        instance_create_depth(768,a,0,obj_ground)
        coord = "768" + string(a)
        ini_write_string(room_number,coord,"ground")
    }
```

Рисунок 11 – код генерации дверей

Последним этапом генерации комнаты выступает заполнения оставшегося пространства, что происходит быстро и легко, так как для каждой пустой ячейки идет бросок виртуального кубика, и в зависимости от его значения, ячейка принимает то или иное значение, которое в будущем будете переведено в игровой объект (рис. 12).

```
for (i=64; i<=1536; i+=64)
{
    for (m=64; m<=832; m+=64)
    {
        coord = string(i) + string(m)
        if ini_read_string(room_number, coord, "none") = "none"
        {
            ini_write_string(room_number, coord, tile[irandom_range(0,3)])
            if(ini_read_string(room_number, coord, "none")) = "ground"
            {
                instance_create_depth(i,m,0,obj_ground)
            }
            if(ini_read_string(room_number, coord, "none")) = "hole"
            {
                instance_create_depth(i,m,0,obj_hole)
            }
            if(ini_read_string(room_number, coord, "none")) = "wall"
            {
                instance_create_depth(i,m,0,obj_wall)
            }
            if(ini_read_string(room_number, coord, "none")) = "grass"
            {
                instance_create_depth(i,m,0,obj_grass)
            }
        }
    }
}
```

Рисунок 12 - код генерации комнаты

По завершению заполнения массива, скрипт начинает накладывать его на сетку комнаты, соотнося номер ячейки и координаты, а так же значение ячейки и объекта, создавая новую комнату, отличную от прошлых комнат (рис. 13-15).

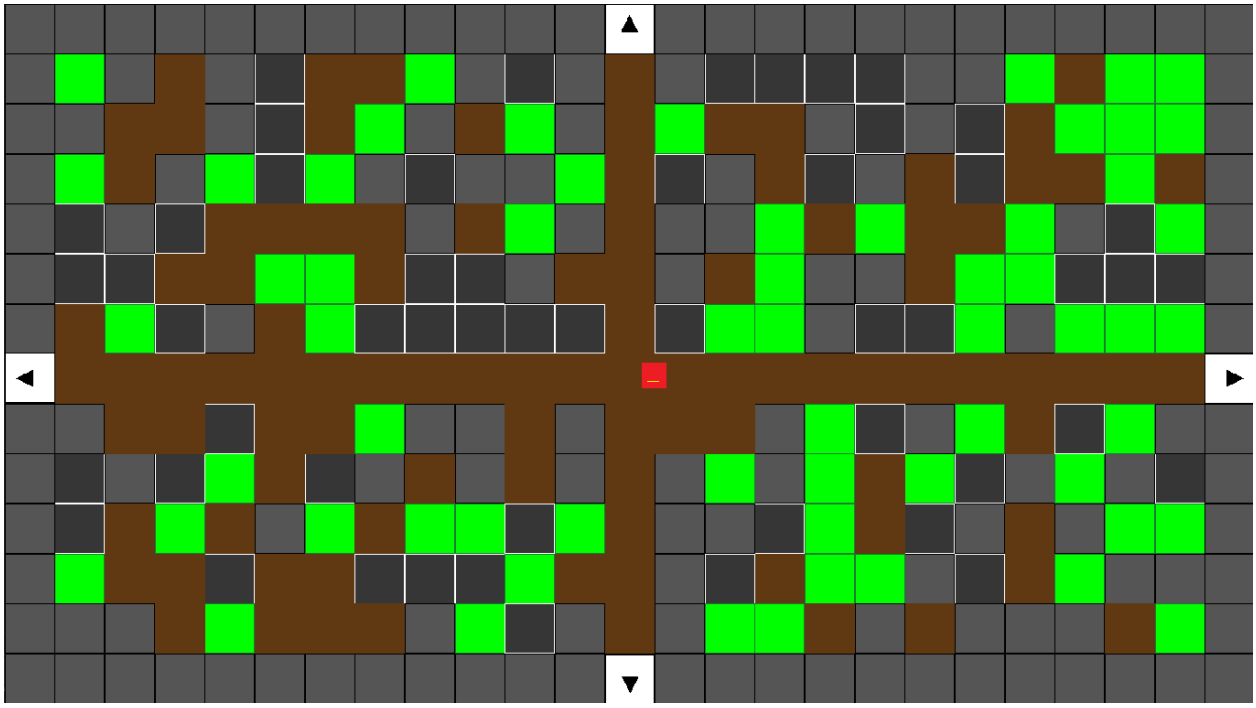

Рисунок 13 – пример комнаты

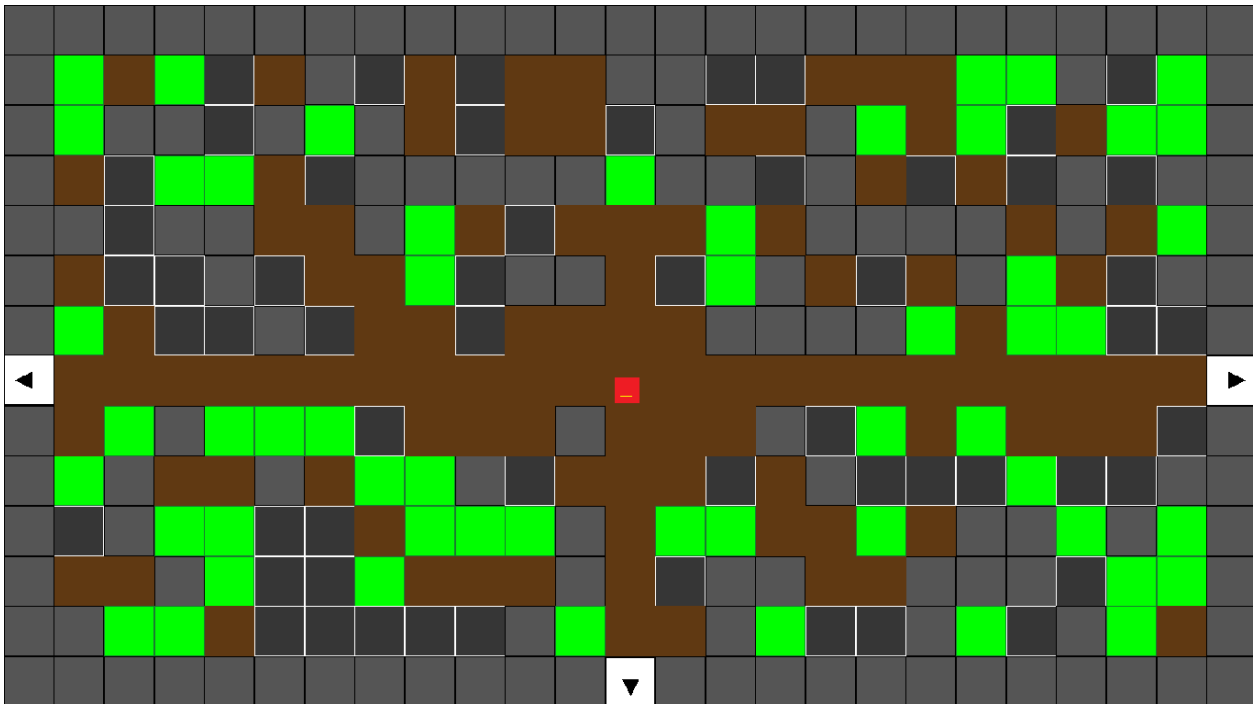

Рисунок 14- пример комнаты

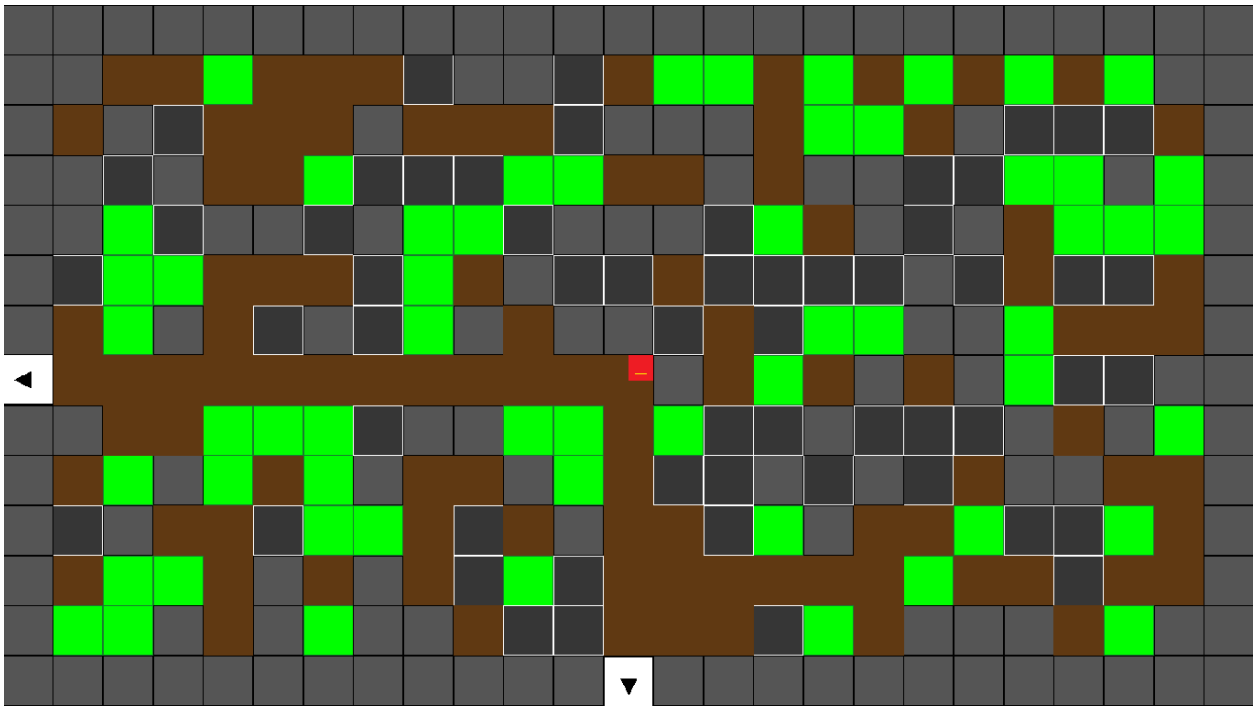

рисунок 15 – пример комнаты

3.4 Сохранение прогресса

По завершению создания генераторов, стоит озаботиться сохранением того, что было создано, что вызывает проблемы. Внутренние функции Game Maker Studio 2 не подходят для сохранения подобных вещей, что приводит к необходимости использовать внешние файлы, или же .ini файлы. Используя их, можно записать почти любые данные игры, причем сделать это достаточно

просто. Для любой работы с внешним файлом, нам необходимо открыть его внутри Game Maker Studio, что делается просто командой (рис. 16).

```
ini_open("GAME.ini")
```

Рисунок 16 – открытие внешнего файла

Стоит отметить, что даже если файла с данным именем не существует, то Game Maker Studio создаст его в папке с игрой. Так же надо внимательно подходить к написанию команды, так как если забыть написать расширение, или взять не те кавычки, Game Maker Studio просто не найдет нужный файл, но ошибки не выдаст. После открытия файла мы можем, как редактировать его наполнение, так и копировать его в игру. Чтобы корректно взаимодействовать с файлом, он должен иметь определенную структуру (рис. 17).

```
(section, key, value);
```

Рисунок 17 – структура файла сохранения

Где section это наименование секции внутри файла, key какой либо элемент этой секции, а value значение этого элемента.

Для примера, используем сохранение данных о положении персонажа в данный момент (рис. 18).

```
i = ini_read_real("DATA","NOW_i", 0)
m = ini_read_real("DATA","NOW_m", 0)
```

Рисунок 18 – сохранение данных в файл

Где i и m это координаты персонажа по x и y, ini_read_real команда для записи числа в файл, DATA секция внутри файла, а NOW_i и NOW_m элементы этой секции. Число 0 в данном случае стоит как некий предохранитель, в случае если значение которое мы хотим записать в файл некорректно, или иного типа, то в элемент будет поставлен 0. Используя внешний файл, мы можем сохранить данные о сгенерированных уровнях, и комнатах, но есть проблема. Так как по сути своей комната это двумерный массив, ячейки которого нам и нужно сохранить, возникает проблема. Каждая ячейка имеет три набора данных которые нам нужны. Это две координаты, и значение ячейки. Но структура ini файлов не

дает нам возможности сохранить столько данных, так что приходиться прибегнуть к обходному пути. В качестве секции мы будем использовать номер комнаты, а в качестве элемента секции координаты, которые мы получим превратив в строку и сложив вместе координату объекта, ну а значением элемента будет тип объекта. То есть, если нам нужно сохранить данные об условном объекте стены с координатами x=640 и y=640, то в файл это будет записано следующим образом (рис. 19).

```
coord = string(i) + string(m)
instance_create_depth(i,m,0,obj_wall)
ini_write_string(room_number,coord,"wall")
```

Рисунок 19 - сохранение данных в файл

Где coord это превращенные в строку и сложенные в одну координаты объекта, room_number номер комнаты, а wall объекта. В результате в файле мы получим запись (рис. 20).

```
640640="hole"
```

Рисунок 20 – запись в файле

При загрузке этих же данных обратно в игру, проделываем то же самое, то есть, заполняя комнату, мы каждую координату превращаем в строку, и уже в таком виде ищем нужные нам данные (рис. 21).

```
if room_exist = "Yes"
{
    for(i=0; i<=1536; i+=64)
    {
        for(m=0; m<=832; m+=64)
        {
            coord = string(i) + string(m)
            switch (ini_read_string(room_number,coord,"wall"))
            {
                case "ground":
                    instance_create_depth(i,m,0,obj_ground)
                break;
                case "wall":
                    instance_create_depth(i,m,0,obj_wall)
                break;
                case "hole":
                    instance_create_depth(i,m,0,obj_hole)
                break;
                case "grass":
                    instance_create_depth(i,m,0,obj_grass)
                break;
                case "right":
                    instance_create_depth(i,m,0,obj_door_right)
                break;
                case "left":
                    instance_create_depth(i,m,0,obj_door_left)
                break;
                case "up":
                    instance_create_depth(i,m,0,obj_door_up)
                break;
                case "down":
                    instance_create_depth(i,m,0,obj_door_down)
                break;
                default:
                    instance_create_depth(i,m,0,obj_ground)
                break;
            }
        }
    }
```

Рисунок 21 – поиск данных в файле

3.5 Персонаж и противники

Когда работа с окружением завершена, пора переходить к персонажу и его противникам. Настройка персонажа проста, так как ему не требуется задавать ничего особенного. Передвижение персонажа стандартно для многих игр, то есть опирается на W A S D (Рис. 22).

```
var a,d,s,w;

a = keyboard_check(ord("A"));
d = keyboard_check(ord("D"));
s = keyboard_check(ord("S"));
w = keyboard_check(ord("W"));

var dyr = point_direction(0,0,d-a,s-w);

if (d-a) != 0 || (s-w) != 0
    {
        hsp = lengthdir_x(pspeed,dyr);
        vsp = lengthdir_y(pspeed,dyr);
    }

hsp *= 0.7;
vsp *= 0.7;
```

Рисунок 22 – код передвижения персонажа

Этот код отвечает за движение объекта с помощью клавиш WASD на клавиатуре. Переменные a, d, s и w определяют, нажаты ли соответствующие клавиши на клавиатуре. Далее, код использует функцию point_direction для вычисления угла направления движения dyr, основываясь на координатах клавиш WASD. Если клавиши нажаты, то устанавливаются значения переменных hsp и vsp для горизонтального и вертикального движения объекта с помощью функций lengthdir_x и lengthdir_y, используя скорость pspeed и угол направления dyr. Затем значения hsp и vsp уменьшаются на 30% с помощью умножения на 0,7. Это позволяет замедлить движение объекта и сделать его более плавным.

Кроме того, стоит так же предусмотреть и ситуацию при столкновении игрока с объектами окружения (рис. 23).

```
function col_wall()
    {
        if place_meeting(x+hsp,y,obj_wall)
            {
                while !place_meeting(x+sign(hsp),y,obj_wall)
                {
                    x += sign(hsp);
                }
                hsp = 0;
            }

        if place_meeting(x,y+vsp,obj_wall)
            {
                while !place_meeting(x,y+sign(vsp),obj_wall)
                {
                    y += sign(vsp);
                }
                vsp = 0;
            }
        if place_meeting(x+hsp,y,obj_hole) && obj_player.fly = 0
            {
                while !place_meeting(x+sign(hsp),y,obj_hole)
                {
                    x += sign(hsp);
                }
                hsp = 0;
            }

        if place_meeting(x,y+vsp,obj_hole) && obj_player.fly = 0
            {
                while !place_meeting(x,y+sign(vsp),obj_hole)
                {
                    y += sign(vsp);
                }
                vsp = 0;
            }




    }
```

Рисунок 23 – код столкновения с объектами

Первый условный оператор if проверяет, происходит ли столкновение персонажа с объектом при перемещении на заданные значения переменных x и y с учетом скорости перемещения по оси x (hsp). Если столкновение происходит, то код выполняет цикл while, который изменяет значение переменной x, пока персонаж не перестанет сталкиваться со стеной. После этого значение hsp устанавливается равным нулю, что означает, что персонаж не движется по оси x. Аналогично, второй условный оператор проверяет столкновения персонажа с объектами при перемещении на заданные значения переменных x и y с учетом скорости перемещения по оси y (vsp).

Последнее что необходимо добавить персонажу игрока, так это возможность отбиваться от противников. В качестве вооружения, персонаж использует магию, и стреляет небольшими магическими шарами, тратя при этом определенное количество игровой энергии на каждый выстрел (рис. 24).

```
if global.player_mana >= global.magic_ball_cost && ball_reload = 0
    {
        instance_create_depth(self.x,self.y,-1,obj_energy_ball)
        global.player_mana -= global.magic_ball_cost
        ball_reload = 1
        alarm[2] = global.magic_ball_reload
    }
```

Рисунок 24 – код использования магии

Вначале проверяем хватает ли персонажу игровой энергии на выстрел, и если да, то на месте персонажа создается объект магического шара, который летит по направлению мышки, отминая при этом немного игровой энергии, а так же запуская таймер, который не даст игроку использовать магию бесперебойно.

Настройка противников будет чуть сложнее, ведь необходимо сделать их разнообразными, чтобы каждый новый противник имел отличие от прошлого противника. Сделать это можно случайным образом задавая модель поведения, (рис. 25), так и случайно распределяя их характеристики (рис. 26).

```
switch (irandom_range(1,2))
    {
        case 1:
            self.espeed = irandom_range(1,5)
            enemy_go_player(self.espeed)
            self.behavior = 1
        break;
        case 2:
            enemy_stay_shoot()
            self.allow_to_shot = 1
            self.behavior = 2
        break;
    }
```

Рисунок 25 – задача модели поведения противника

```
function enemy_go_player(espeed)
{
    mp_grid_add_instances(global.grid_mp,obj_wall,true)
    mp_grid_add_instances(global.grid_mp,obj_hole,true)
    path = path_add()
    mp_grid_path(global.grid_mp,path,self.x,self.y,obj_player.x,obj_player.y,1)
    path_start(path,espeed,0,0)
}


function enemy_stay_shoot()
{
    if self.allow_to_shot = 1 && !collision_line(self.x,self.y,obj_player.x,obj_player.y,obj_wall,0,0)
    {
        instance_create_depth(self.x,self.y,-1,obj_enemy_ball)
        self.allow_to_shot = 0
        alarm[0] = 50
    }
}
```

Рисунок 26 – создание характеристик противника

Сделаем две модели поведения. Первая модель, behavior преследует игрока, старясь нанести удар в ближнем бою, вторая, stay_shoot на месте, но имеет возможность стрелять по персонажу игрока.

Далее необходимо создать возможность противнику появляться в комнатах, а также возможность преследовать игрока. Что бы реализовать это, нам необходимо построить новую сетку, поверх комнаты (рис. 27)

```
global.grid_mp = mp_grid_create(0,0,1600 div 64,900 div 64,64,64)
```

Рисунок 27 – создание сетки

Добавляя эту строчку в скрипт создания комнаты, мы сразу же по завершению создания комнаты, тут же создаем сетку, на которую опираются противники. Так же добавим строчку, которая создает противника на клетке со случайным шансом, и только при условии, что от этой клетки можно добраться до игрока (рис. 28).

```
for (i=64; i<=1536; i+=64)
    {
        for (m=64; m<=832; m+=64)
        {
            coord = string(i) + string(m)
            if(ini_read_string(room_number, coord, "none")) = "grass" || (ini_read_string(room_number, coord, "none")) = "ground"
            {
                if irandom_range(0, 100) > 95
                    {
                        instance_create_depth(i,m,-1,obj_enemy)
                    }
            }
        }
    }
```

Рисунок 28 – появление противника

Сами противники, в случае если их модель поведения это преследование игрока, так же опираются на эту сетку при поиске пути (рис. 29).

```
mp_grid_add_instances(global.grid_mp,obj_wall,true)
mp_grid_add_instances(global.grid_mp,obj_hole,true)
path = path_add()
mp_grid_path(global.grid_mp,path,self.x,self.y,obj_player.x,obj_player.y,1)
path_start(path,espeed,0,0)
```

Рисунок 29 – нанесение пути на сетку

Первые две строчки добавляют объекты окружения в список недоступных для прохода, третья создает путь, который в следующей строке создается до персонажа игрока, после чего враг начинает ему следовать.

3.6 Инвентарь и предметы

Теперь, когда готовы основные механики, а так же настроены противники и персонаж, необходимо добавить ресурсы в нашу игру. А так как ресурсам необходимо место для хранения, начинать нужно с создания инвентаря. Инвентарь будет находиться в отдельной комнате, и попадать игрок в неё будет путем нажатия на клавишу I на клавиатуре (рис. 30).

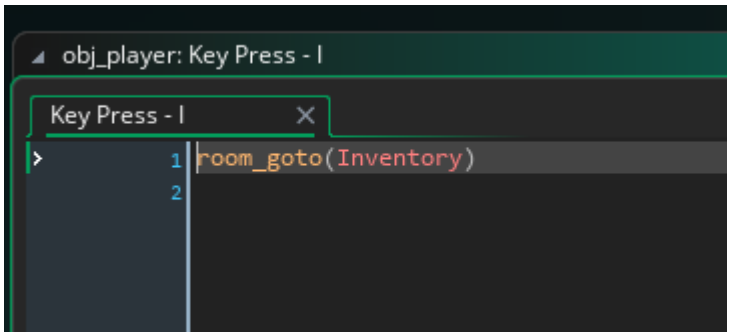

Рисунок 30 – переход в инвентарь

В самой комнате будет один объект контроля, который и будет производить все работы с инвентарем. Независимо от того подбирал ли игрок какие либо предметы, необходимо сразу создать все ячейки, пусть даже и пустые (рис. 31).

```
for (i = 64; i <= 1024; i += 64)
    {
        instance_create_depth(i,64,0,obj_inv_cell_empty)
    }
for (i = 64; i <= 1024; i += 64)
    {
        instance_create_depth(i,128,0,obj_inv_cell_empty)
    }
```

Рисунок 31 – создание ячеек

Когда ячейки подготовлены, создаем предметы, которые будут туда попадать, для прототипа игры хватит и двух объектов, “эссенций”. Появляться они будут со случайным шансом после гибели противника, и игрок будет иметь возможность поднять их, и переместить в инвентарь (рис. 32).

```
for (i = 1; i <= 25; i +=1)
    {
        if global.inv[i] = "none"
            {
                ini_open("GAME.ini")
                global.inv[i] = drop
                cell_number = string(i) + "cell"
                ini_write_string("INV",cell_number,drop)
                switch drop
                    {
                        case "es_phys":
                            {
                                ini_write_real("INV","count_es_phys",1)
                                break
                            }
                        case "es_spirit":
                            {
                                ini_write_real("INV","count_es_spirit",1)
                                break
                            }
                    }
                break
            }
        if global.inv[i] = drop
            {
                switch drop
                    {
                        case "es_phys":
                            {
                                count = ini_read_real("INV","count_es_phys",0) + 1
                                ini_write_real("INV","count_es_phys",count)
                                break
                            }
                        case "es_spirit":
                            {
                                count = ini_read_real("INV","count_es_spirit",0) + 1
                                ini_write_real("INV","count_es_spirit",count)
                                break
                            }
                    }
                break
```

Рисунок 32 – заполнение ячеек

Этот код проверяет все 25 ячеек инвентаря и ищет пустую ячейку или ячейку с предметом, который мы хотим добавить в инвентарь (предмет задан переменной drop). Если находится пустая ячейка, то в нее добавляется предмет, а информация о добавлении записывается в файл ini. Если находится ячейка с таким же предметом, то количество этого предмета в инвентаре увеличивается на 1 и также записывается в файл ini. В конце каждого if происходит выход из цикла, чтобы не проверять лишние ячейки.

Так же необходимо добавить функцию загрузки инвентаря, чтобы не терять все предметы после каждого выхода из игры (рис. 33).

```
function load_inventory(){
    ini_open("GAME.ini")
    for (i = 1; i <= 25; i += 1)
        {
            cell_number = string(i) + "cell"
            global.inv[i] = ini_read_string("INV",cell_number,"none")
            count_of_cell = string(i) + "count"
            global.cell_count[i] = ini_read_real("INV",count_of_cell,1)
            global.es_spirit = ini_read_real("INV","es_spirit",0)
            global.es_phys = ini_read_real("INV","es_phys",0)


        }
}
```

Рисунок 33 – загрузка инвентаря

Так же необходима функция сброса инвентаря, на случай если игрок захочет начать новую игру (рис. 34).

```
    ini_open("GAME.ini")
    for (i = 1; i <= 25; i +=1)
        {
            cell_number = string(i) + "cell"
            ini_write_string("INV",cell_number,"none")
            global.inv[i] = ini_read_string("INV",cell_number,"none")
            global.cell_count[i] = 0
            global.es_spirit = 0
            global.es_phys = 0
            ini_write_real("INV","es_spirit",0)
            ini_write_string("INV","es_phys",0)
        }
}
```

Рисунок 34 – сброс инвентаря

Можем проверить работоспособность, используя тестовые спрайты (рис. 35).

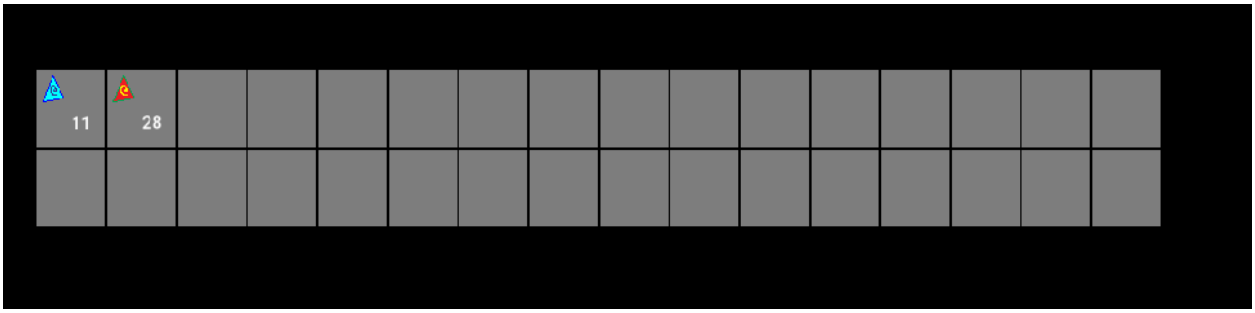

Рисунок 35 – тестирование инвентаря

Как видно, все работает, следовательно, можно переходить к следующему этапу.

3.7 Система улучшения персонажа.

Система инвентаря и предметов готова, необходимо создать систему, где это все применять. Для начала будет создана механика по улучшению героя, и его

магии. Создать подобное достаточно легко, а для облегчения доступа к этому, надо создать стартовую комнату, куда герой будет попадать либо вначале игры, либо после смерти, и где он сможет улучшить себя, или вновь спуститься в подземелье (рис. 36).

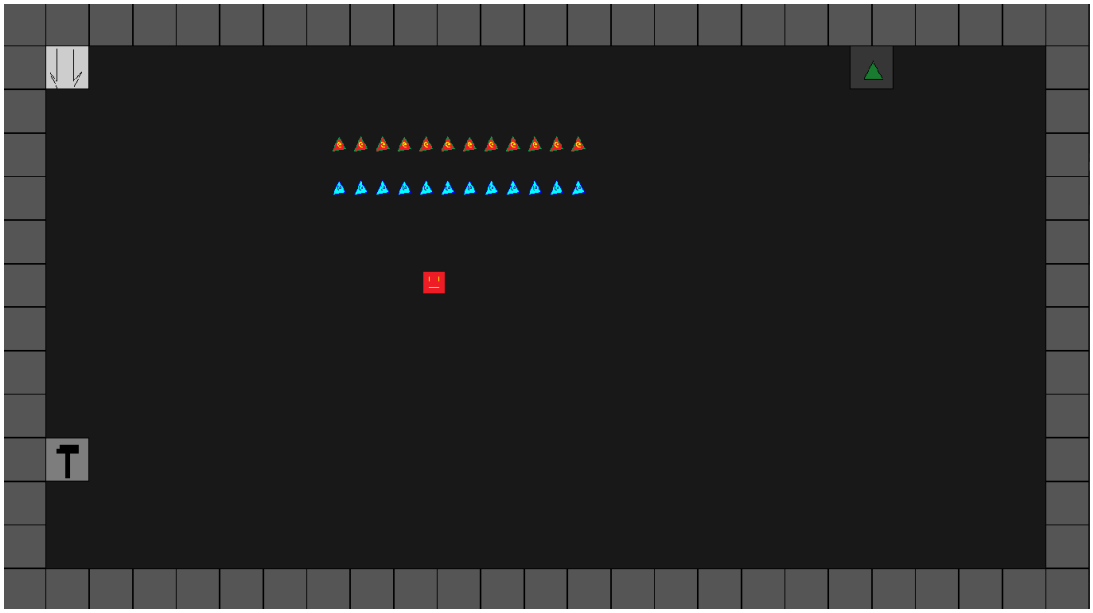

Рисунок 36 – стартовая комната

Чтобы разбавить процесс улучшения, и сделать игру более непредсказуемой, система улучшений так же будет включать элементы случайности. Каждый раз, тратя ресурсы на улучшение, размер этого самого улучшения будет случайным. Подобный подход заставит игрока постоянно искать новый способ прохождения, ведь нет гарантии, что с трудом добытые ресурсы точно окупят себя, а значит, стоит куда осторожнее подходить к выбору стратегии и тактики. Кроме того, чтобы игровой процесс постоянно бросал игроку вызов, каждый раз улучшая себя, игрок рискует улучшить и противника. Подобный подход позволит игре всегда бросать игроку вызов, заставляя того постоянно менять свой подход к прохождению.

Чтобы реализовать это, так же будут созданы отдельные комнаты, попасть в которые можно через стартовую комната. Каждая комната будет улучшать либо героя, либо его магию (рис. 37-38).

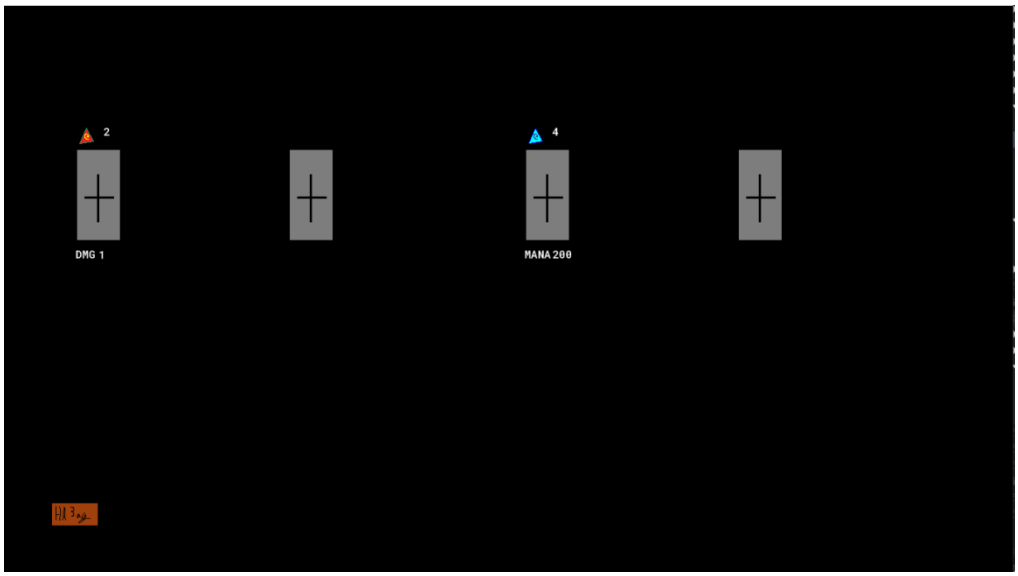

Рисунок 37 – тест-комната улучшений

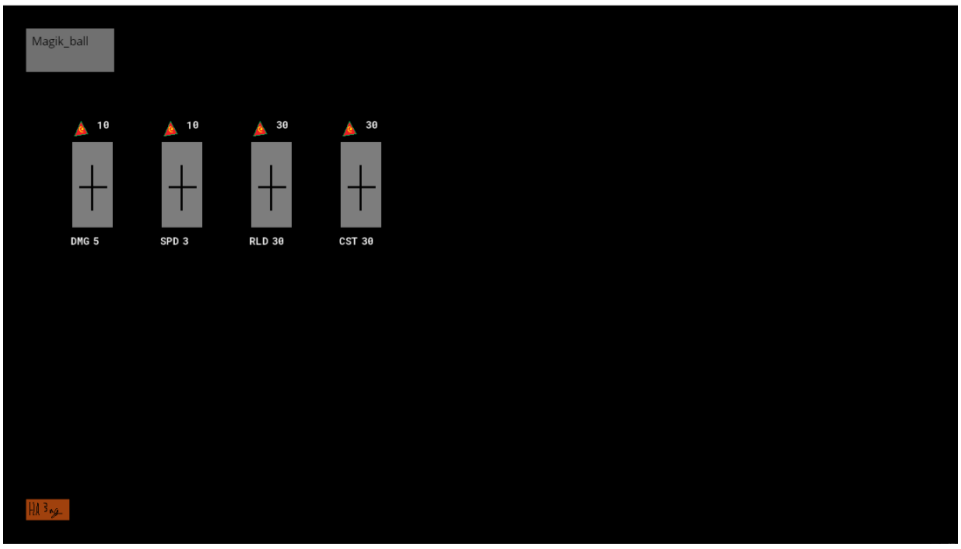

Рисунок 38 – тест-комната улучшений

Каждая кнопка улучшает свой параметр, в случае если на неё нажмут, и если у персонажа хватает ресурсов на улучшение (рис. 39).

```
if ini_read_real("INV","count_es_phys",1) >= price
    {
        global.player_dmg += irandom_range(1,5)
        count = ini_read_real("INV","count_es_phys",1)
        ini_write_real("INV","count_es_phys", count-price)
        ini_write_real("DATA","player_dmg", global.player_dmg)
    }
```

Рисунок 39 – код улучшения

Для удобства, все данные так же записываются и в ini файл, что позволяет сохранять их и загружать по необходимости.

Используя данную систему улучшения, как уже было сказано, игрок всегда будет находиться в процессе борьбы с игрой. Будет необходимо постоянно искать новый путь, и подбирать новую стратегию игры.

3.8 Меню и интерфейс

Последнее что стоит добавить в наш прототип, это меню, и минимальный худ.

Худ (HUD) в играх - это интерфейс, который отображает информацию о состоянии игрока, его здоровье, боеприпасах, инвентаре и других параметрах, а также показывает цели и задания. Он может включать в себя миникарту, индикаторы прогресса и другие элементы, которые помогают игроку ориентироваться в игровом мире и принимать решения.

Нам необходимо отображать только здоровье игрока, и уровень его игровой энергии. Сделать это можно, добавив два индикатора в углы экрана, красный для здоровья, и синий для игровой энергии.

При полном уровне параметров, оба объекта отображаются в полной мере, но если один из параметров начинает убывать, то спрайт соответствующего параметра начинает частично стираться, показывая тем самым, что запас игровой энергии или здоровья все меньше и меньше (рис. 40).

```
ini_open("GAME.ini")
draw_sprite(spr_back_stat,-1,0,0)
draw_sprite_part(spr_hp_bar,-2,0,0,sprite_get_width(spr_hp_bar)*(global.player_hp/ini_read_real("DATA","player_max_hp",100)),sprite_get_height(spr_hp_bar),0,0)
draw_sprite(spr_back_stat,-1,1400,0)
draw_sprite_part(spr_mana_bar,-2,0,0,sprite_get_width(spr_mana_bar)*(global.player_mana/ini_read_real("DATA","player_max_mana",200)),sprite_get_height(spr_mana_bar),1400,0)
```

Рисунок 40 – код интерфейса

Меню и содержимое позволит, либо начать новую игру, либо загрузить ранее сохраненную. Для каждой необходим свой блок кода, выполняющий ту или иную функцию. Так как уже в процессе игры, многие параметры записываются в файл, а по сути, сохраняются, необходимости в данной кнопке нет. Кнопка загрузки должна в случае необходимости достать данные из ini файла, и добавить их в игру (рис. 41).

```
ini_open("GAME.ini")
room_now = ini_read_real("DATA","room_last",0)
global.player_hp = ini_read_real("DATA","player_hp",50)
global.player_mana = ini_read_real("DATA","player_mana",50)
global.player_dmg = ini_read_real("DATA","player_dmg",1)
global.magic_ball_dmg = ini_read_real("DATA","magic_ball_dmg",5)
global.magic_ball_speed = ini_read_real("DATA","magic_ball_speed",3)
global.magic_ball_reload = ini_read_real("DATA","magic_ball_reload",30)
global.magic_ball_cost = ini_write_real("DATA","magic_ball_cost",30)
room_goto(room_now)
load_inventory()
```

Рисунок 41 – код загрузки игры

Кнопка же начать игру, должна сбрасывать все данные, и создавать все с нуля. Но так как многое в игре завязано на внешний файл, можно просто удалить его, а после создать по новой, заполнив начальными данными и его, и игру (рис. 42).

```
file_delete("GAME.ini")
inventory()
room_goto(HUB)
ini_open("GAME.ini")
ini_write_real("DATA","player_max_hp",100)
ini_write_real("DATA","player_max_mana",200)
ini_write_real("DATA","player_mana_regen",2)
ini_write_real("DATA","player_dmg",1)
ini_write_real("DATA","enemy_max_hp",30)
ini_write_real("DATA","magic_ball_dmg",5)
ini_write_real("DATA","magic_ball_speed",3)
ini_write_real("DATA","magic_ball_reload",30)
ini_write_real("DATA","up_ball_reload",1)
ini_write_real("DATA","magic_ball_cost", 30)
global.player_hp = ini_read_real("DATA","player_max_hp",100)
global.player_mana = ini_read_real("DATA","player_max_mana",200)
global.player_dmg = ini_read_real("DATA","player_dmg",1)
global.magic_ball_dmg = ini_read_real("DATA","magic_ball_dmg",5)
global.magic_ball_speed = ini_read_real("DATA","magic_ball_speed",3)
global.magic_ball_reload = ini_read_real("DATA","magic_ball_reload",30)
global.magic_ball_cost = ini_read_real("DATA","magic_ball_cost", 30)
```

Рисунок 42 – код начала игры

На этом работу с меню и интерфейсом можно считать законченной.

3.9 Обновление спрайтов

По завершению основных механик игры, осталось изменить временные спрайты на постоянные. Чтобы процесс был быстрее, а итог качественнее, необходимые спрайты будут заказаны у художника, который и предоставил их в срок. По получении готовых спрайтов, необходимо заменить их внутри игры, после чего можно считать, что прототип готов (рис. 43).

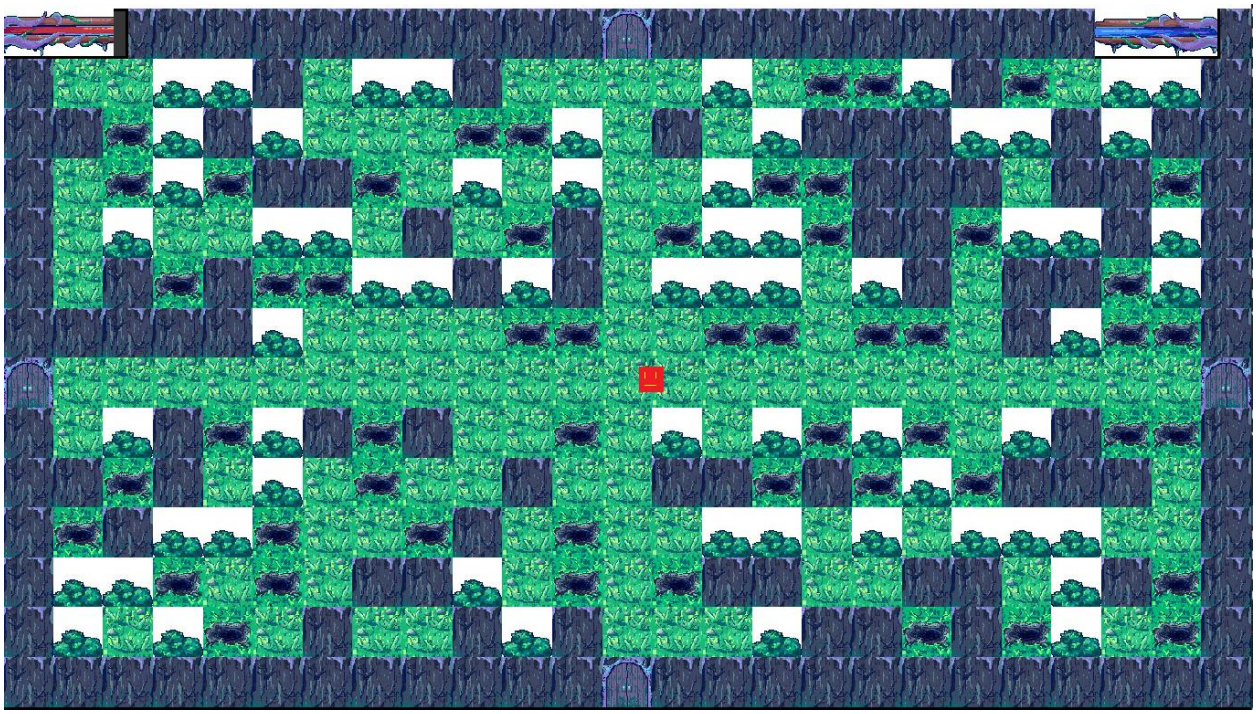

Рисунок 43 – обновленные спрайты

ЗАКЛЮЧЕНИЕ

При написании работы был проведен анализ доступных источников, что помогло определить понятие «компьютерная игра». Так же была проведена классификация компьютерных игр по нескольким критериям, хотя и из-за молодости индустрии, а так же того, что классификация компьютерных игр не систематизирована, составить подробную и точную классификацию не удалось. Был составлен алгоритм разработки игры. Проведён анализ популярных средств разработки. По итогам анализа, было выбрано наиболее актуальное средство разработки для начинающих разработчиков.

Основываясь на полученной в ходе исследования информации, было решено разработать прототип двумерного платформера для одного игрока на игровом движке Game Maker Studio 2. Такое решение было принято по нескольким причинам:

-Создание игр на 2D основе куда менее ресурснозатратно, нежели 3D;

-Использование жанра Roguelike позволяет разнообразить будущий игровой процесс, так как каждая новая игра будет новой;

-Game Maker Studio 2 распространяется в бесплатной версии, что позволяет легко достать её копию для начинающего разработчика

После выбора средств разработки было начато изучение Game Maker Studio, а так же разработка самого проекта. В ходе разработки был изучен игровой движок Game Maker Studio и были приобретены необходимые знания и умения,

Получение навыков разработки игр важно, в современном мире индустрия разработки игр все сильнее распространяется в нашем обществе. Игры перестали быть лишь предметом для развлечений, и теперь используются и в других областях, например, в науке или в обучении пользователей. Поэтому развитие в данном направлении можно считать одним из самых интересных в современном обществе.

В ходе реализации проекта были выполнены следующие задачи:

1) изучены особенности и состояние компьютерной индустрии России;

2) выбраны жанр, вид и платформа для компьютерной игры;

3) выбрано и изучено средство реализации;

4) реализован прототип игры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Компьютерные игры как искусство [Электронный ресурс]. — Режим доступа: http://gamesisart.ru

2. Game Maker: Studio [Электронный ресурс]. — Режим доступа: https://www.yoyogames.com/gamemaker

3.Game Maker Studio 2 [Электронный ресурс] —Режим доступа: https://manual.yoyogames.com/GameMaker_Language/GML_Reference/File_Handling/Ini_Files/Ini_Files.htm

4. Работа с файлами для сохранения и считывания данных [Электронный ресурс] —Режим доступа: https://forum.hellroom.ru/index.php?topic=12373

5. GameMaker Studio 2: Using .INI files [Электронный ресурс] —Режим доступа: https://www.youtube.com/watch?v=VioC1xk5770

6. Живи, умри, и снова: погружаемся в мир рогаликов [Электронный ресурс] —Режим доступа: https://habr.com/ru/companies/ruvds/articles/574252/

7. Бесконечные приключения: лучшие игры жанра roguelike для ПК [Электронный ресурс] —Режим доступа: https://dzen.ru/a/XvhWR2F0VwLL0GPa