Пермский филиал федерального государственного автономного образовательного учреждения высшего образования   
«Национальный исследовательский университет   
«Высшая школа экономики»

*Факультет экономики, менеджмента и бизнес-информатики*

Чепоков Елизар Сергеевич

**РЕАЛИЗАЦИЯ КОМПЬЮТЕРНОЙ ИГРЫ "ЛАБИРИНТ"**

*Курсовая работа*

студента образовательной программы «Программная инженерия»

по направлению подготовки *09.03.04 Программная инженерия*

Руководитель:

старший преподаватель

кафедрыинформационных

технологий в бизнесе

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

О. И. Гордеева

Пермь, 2019 год

**Аннотация**

Название: Реализация компьютерной игры "Лабиринт"

Автор: Чепоков Елизар Сергеевич, студент первого курса образовательной программы «Программная инженерия».

Руководитель: Гордеева Ольга Игоревна, старший преподаватель кафедры информационных технологий в бизнесе.

Данная курсовая работа посвящена разработке компьютерной игры «Лабиринт»

Работа включает 67 страниц формата А4, из них в основной части 29 страниц.

Основная часть работы включает в себя 23 иллюстраций и 2 таблиц.

Библиографический список состоит из 13 публикаций.

Работа включает в себя 6 приложений.

# Оглавление

[Оглавление 3](#_Toc4425845)

[Введение 4](#_Toc4425846)

[Глава 1. Анализ 6](#_Toc4425847)

[1.1. Изучение особенностей лабиринтов 6](#_Toc4425848)

[1.2. Обзор и сравнение программного обеспечения для создания игр 7](#_Toc4425849)

[1.3. Обзор игр-аналогов 11](#_Toc4425850)

[1.4. Требования к разрабатываемой игре 15](#_Toc4425851)

[Глава 2. Проектирование 16](#_Toc4425853)

[2.1. Проектирование алгоритмов 16](#_Toc4425854)

[2.2. Проектирование интерфейса 19](#_Toc4425855)

[Глава 3. Разработка и тестирование. 21](#_Toc4425858)

[3.1. Обоснование выбора средств разработки 21](#_Toc4425865)

[3.2. Описание используемых функций. 21](#_Toc4425866)

[3.3. Тестирование программного продукта. 25](#_Toc4425867)

[Заключение 27](#_Toc4425868)

[Список сокращений и условных обозначений 28](#_Toc4425869)

[Библиографический список 29](#_Toc4425870)

[Приложение А. Виды лабиринтов в играх 30](#_Toc4425871)

[Приложение Б. Таблицы для сравнения ПО 31](#_Toc4425872)

[Приложение В. Блок-схема движения персонажа 32](#_Toc4425873)

[Приложение Г. Полный листинг скриптов игры 33](#_Toc4425874)

[Приложение Д. Тестирование по критериям черного ящика 65](#_Toc4425875)

[Приложение Е. Документация пользователя 66](#_Toc4425876)

[Пользовательская документация 66](#_Toc4425877)

[Введение 66](#_Toc4425878)

[Назначение и условия применения 66](#_Toc4425879)

[Подготовка к работе 66](#_Toc4425880)

[Начало игры 66](#_Toc4425881)

[Для экстренных ситуаций 67](#_Toc4425882)

# Введение

В настоящее время игры пользуются популярностью и дают человеку расслабиться и получить новые эмоции[6]. Существует много, пользующихся спросом, хороших игр, но в основном, подобные игры – это игры с открытым миром в жанре   
«Action/RPG» в которых очень много элементов окружения, такие игры, как правило яркие и требуют постоянного зрительного контакта. Так же с каждым годом игры занимают все больше и больше пространства на компьютере.

Maze – жанр компьютерных игр. Игры данного жанра характеризуются тем, что основной игровой процесс происходит в лабиринте и успех игрока зависит от навигации и ориентации в нём. Сами же лабиринты могут иметь различный вид: сверху, сбоку, с видом от первого лица, либо быть «скрытыми»

Актуальность данной темы состоит в том, что игры подобного жанра очень редко встречаются на игровом рынке и, соответственно, не имеют большой популярности. Большинство подобных проектов – это игры от инди разработчиков, которые пытаются заполнить пустующую нишу и не заморачиваются над самим созданием. Поэтому игры в этом жанре получаются сырыми и недоделанными из-за чего не пользуются спросом. Для игр подобной тематики нужен свежий взгляд и новая интерпретация жанра «Maze». Именно поэтому разработка игры в жанре «Maze» актуальная тема.

*Объектом* данного исследования являются компьютерные игры в жанре «Maze». *Предметом* исследования являются методы и средства разработки компьютерных игр.

*Целью* данной работы является создание компьютерной игры «Лабиринт» с присутствием сюжета.

Задачи, для достижения поставленной цели:

1. Проанализировать все темы, которые понадобятся при разработке игры.
2. Проанализировать и выбрать визуализатор для игры.
3. Изучить строение лабиринтов и строение уровней в игре.
4. Спроектировать интерфейс для игры.
5. Провести опрос и проанализировать ответы по поводу интерфейса, внести изменения.
6. Написать код программы и создать её интерфейс.
7. Провести полное тестирование и отладку по критериям черного ящика.
8. Провести Альфа тестирование, для выявления недочетов и сбора критики пользователей.
9. Написать пользовательскую документацию для программы.

На данный момент не существует точного аналога игры, что делает проблему отсутствия игр жанра «Maze» актуальной. Всего существует несколько игр, приближенных к освещаемой проблеме, это игра «Project Druid», игра «BELOW» и игра «Dark Echo».

Так же для того, чтобы лучше изучить данную тему я буду использовать такие методы исследования как:

* Измерение – данный метод будет использоваться для измерения доступности и интуитивности игры и измерения количества времени, потраченного на прохождение.
* Ранжирование – первые 20 процентов усилий следует потратить на исследование книг, работ и разработку интерфейса (80 процентов результата), а оставшиеся 80 процентов усилий будут потрачены на создание самой игры.
* Моделирование – метод будет применятся для создания структурированного лабиринта и проработки сюжета.
* Визуализация – метод будет использован, при создании интерфейса игры и при внедрении сюжета в игру.
* Дискретизация – метод будет использоваться для оптимизации игры, чтобы иметь максимальный отклик действий и сократить затрачиваемые ресурсы компьютера.

По итогу после создания данного проекта появиться качественная игра в жанре «Maze», что поднимет интерес к данному жанру. Так же покажет обучающимся НИУ ВШЭ, что хорошие игры могут создаваться, не только большими компаниями, но и в одиночку, а также быть крайне минималистичными по своему концепту.

# Анализ

В данной главе будут рассмотрены особенности строения лабиринтов и их особенности. Будут изучены часто используемые визуализаторы для игр и выявлены их особенности. Так же будут рассмотрены игры с использованием лабиринтов, выявлены их недостатки и преимущества. Результатом первой главы будут требования к разрабатываемой игре.

## Изучение особенностей лабиринтов

Построение лабиринтов в играх не является строгим. В некоторых играх количество уровней и обширность игрового мира может составлять тысячи или десятки тысяч локаций. Игровое пространство может организовываться по-разному, из основных видов построения выделяют:

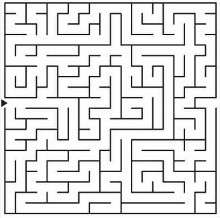
1. Текстовая графика (см. Приложение А, рисунок А.1).
2. Двумерная графика (см. Приложение А, рисунок А.2).
3. Изометрическая графика (см. Приложение А, рисунок А.3).

Для создания игрового поля различают процедурную и фиксированную генерацию.

Для процедурной генерации характерно автоматическое создание и построение игрового контента с использованием алгоритмов, данный метод повышает реиграбельность каждой новой игры, но приводит к усложнению игрового процесса, шанс выигрыша при такой генерации значительно уменьшается, а сложность игры растет. Фиксированная генерация применяется чаще, так как при таком варианте игру можно спроектировать так, чтобы обеспечить её прохождение[4].

В лабиринте может быть одна тропа, без ответвлений и с множеством поворотов, так и несколько троп с поворотами в тупики. В первом случае игровой процесс облегчается, так как не дает права выбора игроку. Во втором же случае игрок проводит в лабиринте больше времени, что дает ему повод и интерес играть дальше, чтобы найти выход[3].

В играх чаще всего используются многосвязные лабиринты (рисунок 1.1), так как они легче в построении и соответствуют условиям.

Условия, которые соблюдаются при построении лабиринта:

**Рисунок 1.1. Пример многосвязного лабиринта**

1. В лабиринте должен быть единственный вход и единственный выход.
2. Дорожка к выходу «сворачивает сама за себя», постоянно изменяя направление.
3. Вернуться ко входу можно только по тому же пути, по которому и пришли.
4. Дорожка должна заполнять все внутреннее пространство и следовать максимально непрямым путем.

## Обзор и сравнение программного обеспечения для создания игр

Для создания и отображения игр требуется визуализатор, чтобы отобразить код программы в изображение. В качестве визуализатора для простых и не затратных игр можно использовать и обычные визуализаторы языка, например, Visual Studio. Но для реализации более сложных алгоритмов приходится более глубоко углубляться в изучение языка и принципы работы разных классов. Таким образом даже имея хорошую квалификацию программист может легко допустить трудноуловимые ошибки, некоторые из которых могут оказаться фатальными[1].

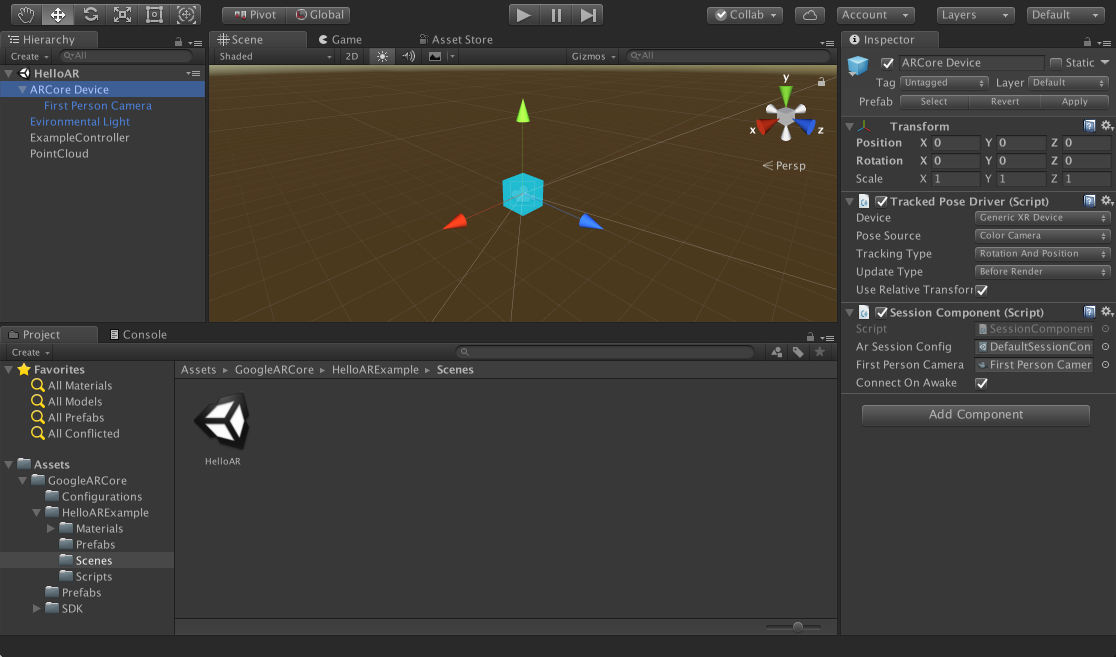
Ввиду этого были разработаны программы для создания игр, которые не только облегчают написание игры и избавляют от большого количества ошибок, но и привносят что-то новое, чего нельзя достичь обычными визуализаторами. Это программное обеспечение, или по-другому «game engine», сейчас используется почти во всех играх.

Ниже приведен перечень самых используемых программных обеспечений для создания игр:

1. Unity
2. Unreal Engine
3. CryEngine
4. Construct

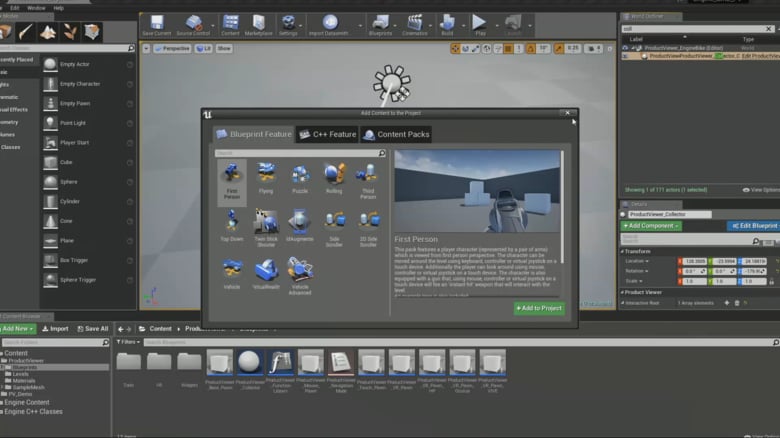
**Unity**. Среда для разработки компьютерных игр, позволяющая создавать приложения на более двадцати различных операционных системах. Первый выпуск состоялся в 2005 году и с того времени постоянно обновлялся, вводя что-то новое и исправляя старое. Основные преимущества Unity это наличие визуальной среды разработки, обширная база данных, межплатформенная поддержка и абсолютная бесплатность для начинающих разработчиков. Поддерживается создание как 2D, так и 3D игр. Основной язык программирования – C#, так же поддерживается и JavaScript. На рисунке 1.2 показан интерфейс программы[10].

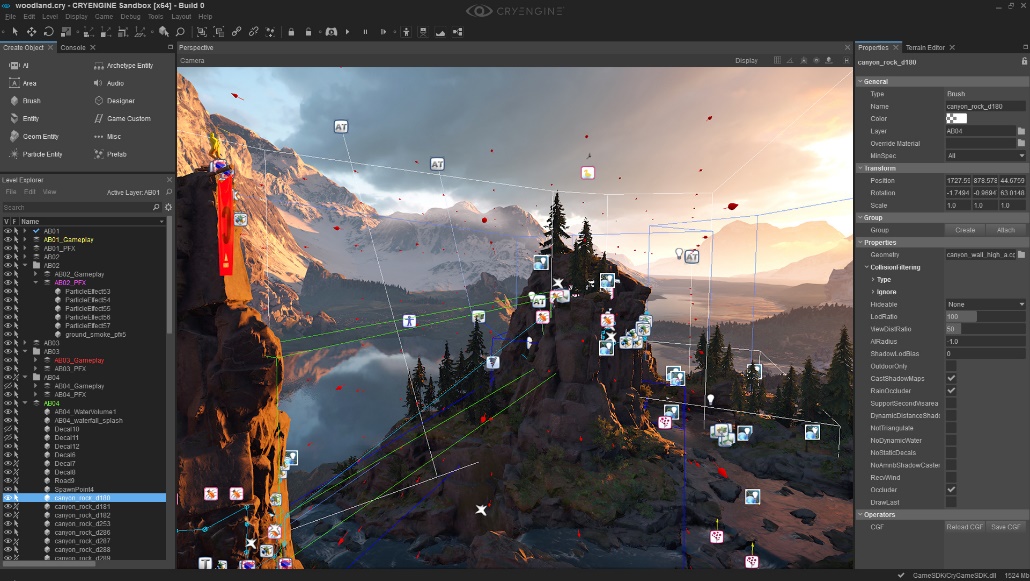
**Рисунок 1.2. Интерфейс Unity**



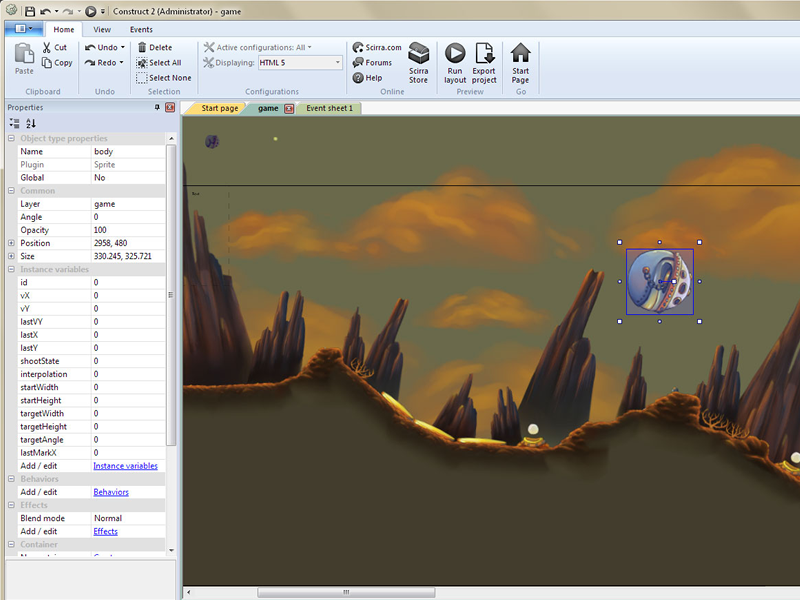
**Unreal Engine**. Среда для разработки компьютерных игр, созданная в 1998 году и принадлежащая компании «Epic Games». Последняя версия – Unreal Engine 4, вышедшая в 2014 году, позволяет создавать приложения на десяти операционных системах. Из преимуществ выделяют инструмент для создания уровней, встроенный в программу и не имеющий аналогов и поддержка различных систем рендеринга. С 2015 года стала бесплатной, для всех разработчиков. Основной язык программирования – С++. В основном направлена на создание 3D игр и фильмов. Интерфейс программы представлен на рисунке 1.3.

**Рисунок 1.3. Интерфейс Unreal Engine**



**CryEngine**. Программное обеспечение, выпущенное в 2002 году и полностью принадлежащее компании «Ubisoft». Всего было выпущено 5 поколений, последним является CryEngine V, выпущенный в 2016 году. В этом же году лицензия перешла на модель «плати сколько хочешь», теперь приобрести лицензию можно за любую сумму, но запрещается использование для неигровых приложений. Данная среда, поддерживает разработку для пяти операционных систем. Направлена на создание 3D игр. Выделяют несколько преимуществ, реалистичная графика персонажей и предметов и детализированная графика частиц и ландшафта. Основной язык программирования – С++, так же поддерживается С#, в качестве расширения. На рисунке 1.4 представлен интерфейс программы.

**Рисунок 1.4. Интерфейс CryEngine**

**Construct**. Среда для создания игр, выпущенная в 2007 году и принадлежащая компании «Scirra». Последняя версия была выпущена в 2011 году. Программное обеспечение специализируется на создание 2D игр для Windows, но с возможностью портирования и на другие операционные системы и браузеры. Из преимуществ выделяют легкость в написании 2D игр любой сложности и увеличение производительности для игр в браузерах. Конструктор имеет ограниченную, бесплатную версию и платную лицензию. Основной язык программирования – JavaScript. Рисунок 1.5 демонстрирует интерфейс приложения.

**Рисунок 1.4. Интерфейс Construct**

Для того чтобы выбрать программное обеспечение для написания игры «Лабиринт», нужно сравнить все программы по критериям. В таблицах А.1-А.3 Приложения Б представлен и сгруппирован весь технический обзор программ. На основе имеющихся данных можно определить какая среда для написания игр лучше, оптимизированнее и лучше остальных. В таблице 1.1 выявляем основные требования к среде в которой будем создавать игру. Как видно из таблицы, удобными в использовании являются Unreal Engine и Construct. Главными же факторами, для начинаюзего разработчика являются наличие документации и бесплатная лицензия. Таким образом лучшим вариантом для первого проекта служит программное обеспечение Unity из-за большого количества документации, бесплатной лицензии и языка С#. Так же игру стоит написать в 2D формате, иначе на разработку уйдет много времени. Хоть Unity и не совсем удобная в использовании из-за отсутствия некоторых инструментов, которые есть в других программах, но она все же остается самой простой в освоении.

**Таблица 1.1. Критерии сравнения среды для написания игры**

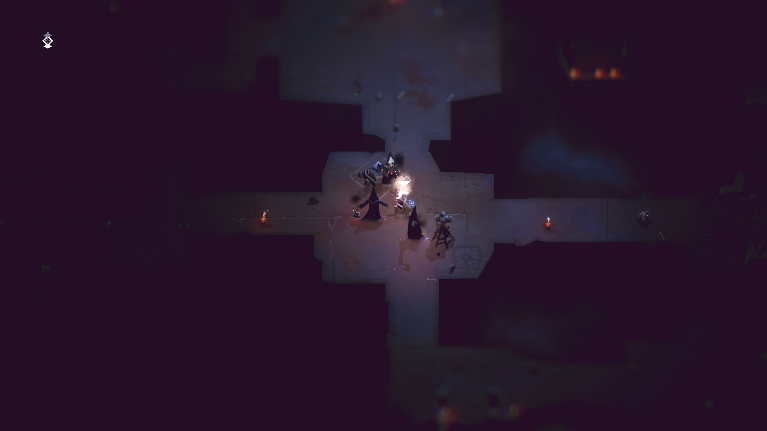
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Среда** | **Поддержка 2D** | **Удобность в использовании** | **Использование языка C#** | **Наличие документации** | **Бесплатность лицензии** |
| 1. Unity | да | частично | да | да | да |
| 2. Unreal Engine | частично | да | нет | да | да |
| 3. CryEngine | нет | нет | нет | частично | частично |
| 4. Construct | да | да | нет | частично | нет |

## Обзор игр-аналогов

До создания игры, нужно изучить игры-аналоги и выявить их преимущества и недостатки для игры «Лабиринт» существуют следующие аналоги:

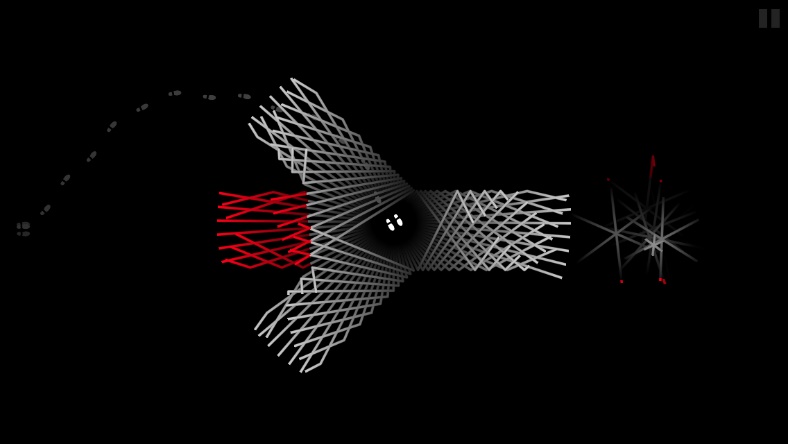
1. Project Druid
2. BELOW
3. Dark Echo
4. Pac-Man
5. Doom

Игра «Project Druid», от разработчика Cemil Tasdemir, выпущенная в 2015 году. Игра создана в 2D формате и имеет 6 уровней, для прохождения уровня, нужно найти выход и ключ. По заверению разработчика игра находилась в разработке полтора года. В игре отсутствуют какие-либо подсказки, интерфейс меню и сохранения. Игра написана в двумерной графике, из-за чего воспринимать карту проще и понятнее. Из плюсов можно выделить двумерную графику и проработанность уровней. Минусов в игре больше чем плюсов, например: интерфейс (рисунок 1.5) не несущий почти никакой информации и неприятный для восприятия, отсутствие меню, а соответственно сохранения, загрузки, настройки звука и управления, отсутствие подсказок, из-за чего не понятно что нужно делать и где найти выход, непроработанные текстуры на уровнях, которые не дают пройти, отсутствие какого либо сюжета. Игра не в полной мере раскрывает данную проблему, так как разработана в жанре «Casual», без сюжета и привязки к лабиринту, так же игра получилась не затягивающая и бессмысленная, что доказывают многочисленные отрицательные отзывы (44 отзыва, из которых 8% положительные).

Игра «BELOW», от разработчика Capybara Games, выпущенная 14 декабря 2018 года. Игра предполагалась к выпуску в 2019, из-за чего на этапе выбора темы курсовой, нельзя было утверждать, что игра аналогична теме курсовой. Данная игра разработана в жанре «Action-adventure». Из преимуществ игры, минималистичный интерфейс, проработанность сюжета и приятная двумерная графика (рисунок 1.6). Из минусов выделяют: процедурную генерацию и, соответственно, сложность прохождения. Игра оценена пользователями как средняя (445 отзывов из которых 70% положительные) Она раскрывает большинство аспектов, которые будут задействованы, такие как: двумерная графика, минималистичность и сюжет.

**Рисунок 1.6. Интерфейс и игровой процесс игры BELOW**

**Рисунок 1.5. Интерфейс и игровой процесс игры Project Druid**

Игра «Dark Echo», от разработчика RAC7, выпущенная в 2015 году. Игра была разработана в жанре «Adventure, Maze». В игре делается акцент на взаимодействие с окружением через звук, из-за чего нет визуализации лабиринта (рисунок 1.7). Игра написана в двумерной графике и имеет минималистичный интерфейс, так же к плюсам можно отнести проработанный сюжет, который прослеживается даже без подсказок и аудио дорожек. Из минусов можно выделить отсутствие визуализации лабиринта, из-за чего трудно проходить некоторые уровни. Эта игра обрела популярность после публикации, что доказывают лестные отзывы (534 отзыва, 93% положительные). На данный момент это самый близкий аналог к данной теме курсовой.

**Рисунок 1.7. Интерфейс и игровой процесс игры Dark Echo**

Игра «Pac-Man», от разработчика Namco, выпущенная в 1980 году. Из плюсов можно выделить двумерную графику, приятный интерфейс. Минусы в том, что игра была выпущена давно и в основном предназначалась, для аркадных автоматов. На рисунке 1.8 представлен интерфейс игры.

**Рисунок 1.8. Интерфейс игры Pac-Man**

Игра «Doom», от разработчика id Software. Игра была выпущена в 1993 году и произвела фурор в игровой индустрии. Плюсы: трёхмерная графика (проекция двумерной графики на трёхмерную) и затягивающий сюжет. Из минусов можно выделить сложность прохождения. Интерфейс игры представлен на рисунке 1.9.

**Рисунок 1.9. Интерфейс игры Doom**

Для систематизации полученных данных можно создать таблицу и выделить требования к создаваемой игре. Для этого выявляем основные требования к новой игре и переносим их уже на рассмотренные программы. В таблице 1.2 представлены аспекты, которые должны сочитаться в новой игре. Таким образом игра должна сочитать двумерную графику, минималистичный интерфейс, в том числе и интерфейс меню, в игре должен быть сюжет и видимый лабиринт. Чтобы не усложнять игровой процесс, а так же исключить появление ошибок генерация лабиринта должна быть фиксированной.

**Таблица 1.2. Критерии сравнения игр**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Игра** | **Двумерная графика** | **Минималистичный интерфейс** | **Присутствие сюжета** | **Визуализация лабиринта** | **Фиксированная генерация лабиринта** |
| 1. Project Druid | да | нет | нет | да | да |
| 2. BELOW | да | да | да | да | нет |
| 3. Dark Echo | да | да | да | нет | да |
| 4. Pac-Man | да | да | нет | да | да |
| 5. Doom | частично | нет | да | да | да |

## Требования к разрабатываемой игре

Основываясь на всём вышесказанном разрабатываемая игра должна выполнять следующие требования:

1. Игра должна быть написана на языке С# в среде разработки Unity.
2. Программа должна взаимодействовать с игроком и откликаться на действия игрока.
3. Интерфейс игры должен быть минималистичен и не содержать ничего лишнего.
4. Должно присутствовать меню.
5. Интерфейс меню должен быть минималистичен и не содержать ничего лишнего.
6. Лабиринты должны быть сгенерированы фиксировано и отображаться на экране.
7. Все лабиринты должны быть построены на основе многосвязного лабиринта.

Так же можно выделить входные и выходные данные:

**Входные данные:**

1. Нажатие кнопок “W”, “A”, “S”, “D” или “стрелка вверх”, “стрелка вниз”, “стрелка влево”, “стрелка вправо”
2. Нажатие кнопки “Esc”

**Выходные данные:**

1. Передвижение по плоскости уровня вверх, вниз, влево или вправо.
2. Вывод интерфейса с настройками, сохранением и выходом из игры.

# Проектирование

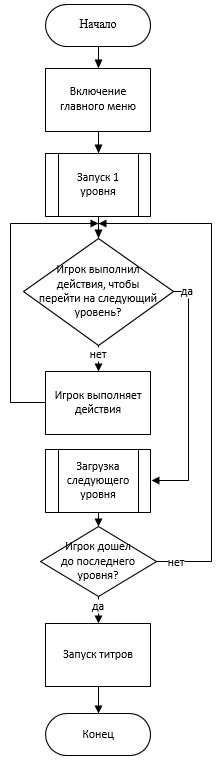
В данной главе будут рассмотрены алгоритмы, которые потребуются для реализации игры и для удовлетворения выдвинутых требований. Результатом данной главы является спроектированная игра и её интерфейс.

## Проектирование алгоритмов

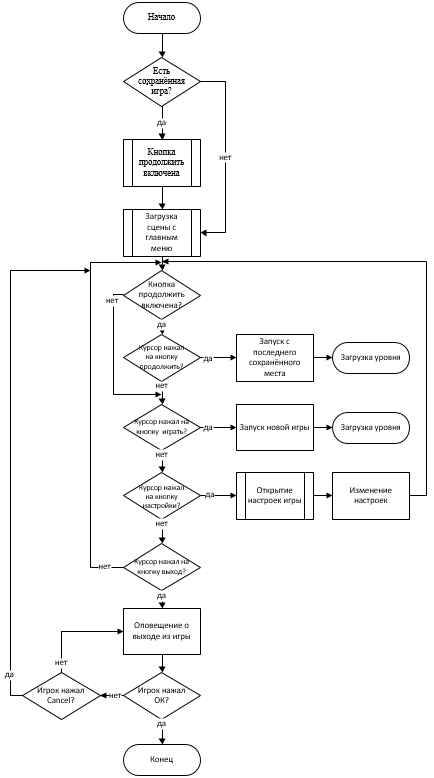
Проектирование алгоритма является одним из важнейших этапов создания любого приложения или игры. Алгоритмы существуют для того чтобы иметь представление, как должна работать программа. Для создания алгоритмов могут использоваться как письменные алгоритмы, так и алгоритмы из рисунков, но чаще всего используют блок-схемы.

Для того чтобы представлять и адаптировать стабильную работу программы, нужно разобрать алгоритм всей программы целиком. На рисунке 2.1 представлен данный алгоритм.

**Рисунок 2.1. Блок-схема главного алгоритма**



Так как игра предполагается многоуровневой, для начала нужно рассмотреть алгоритм главного меню, в которое попадает игрок, после запуска игры (рисунок 2.2).

На данном алгоритме виден весь процесс выбора в меню. Таким образом в первый щапуск игры у игрока будет всего три кнопки: «Новая игра», «Настройки» и «Выход», так как сохранений в игре еще нет и кнопка «Продолжить» будет недоступна. В последующие включения кнопка будет активирована и, нажав на кнопку «Продолжить», игрок будет появляться на том уровне на котором закончил в прошлый раз.

**Рисунок 2.2. Блок-схема главного меню**

Рассмотрим алгоритмы меню и взаимодействия с ним:

* 1. Кнопка «Продолжить»

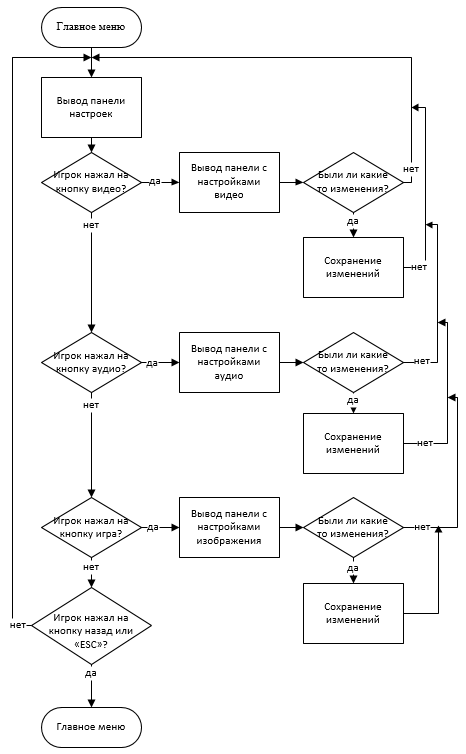
Загружает последний уровень, на котором был игрок. До первого сохранения находится в отключенном режиме.

* 1. Кнопка «Новая игра»

Начинает новую игру и загружает первый уровень в игре. Постоянно включена.

* 1. Кнопка «Настройки»

Открывает панель настроек. В панели настроек можно изменять кнопки для взаимодействия, звук в игре, разрешение окна игры. При нажатии кнопки «Принять изменения» на панели настроек, принимает все изменения, выставленные игроком. На рисунке 2.3 представлена полная блок-схема панели настроек.



**Рисунок 2.3. Блок-схема панели настроек**

* 1. Кнопка «Выход»

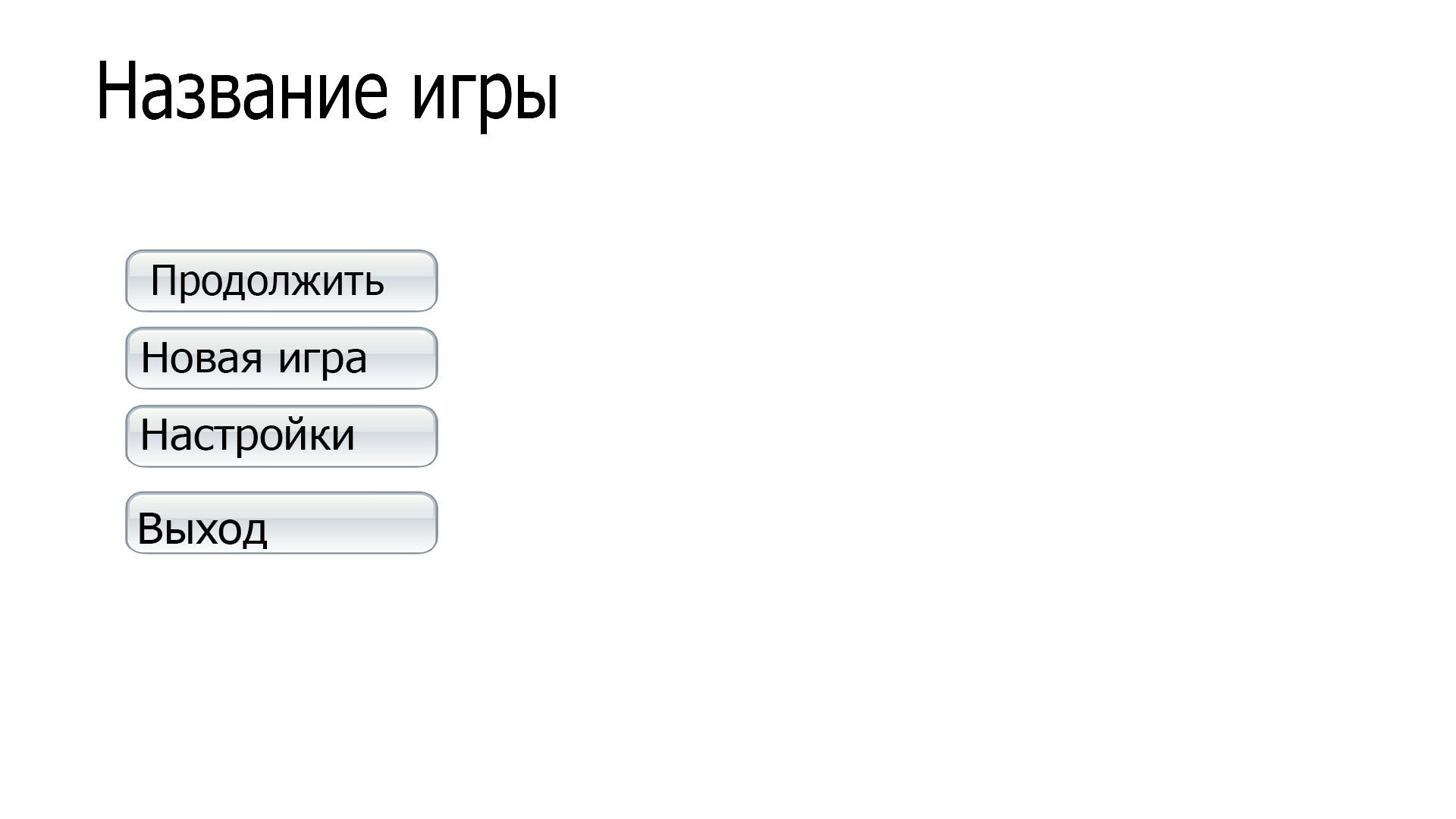
Показывает табличку с вариантами выбора и текстом «Вы точно хотите выйти?». При нажатии на кнопку «OK» или «Enter» закрывает игру. При нажатии кнопки «Cancel» или «Esc» возвращает в меню.

Так же нужно рассмотреть алгоритмы для движения персонажа и его взаимодействия с предметами[9]. Так как в разрабатываемой игре подразумевается минималистичный интерфейс, движение персонажа будет простым[8]. Таким образом будет реализовано движение влево – кнопка «A» или «Стрелка влево», движение вправо – кнопка «D» или «Стрелка вправо», движение вверх – кнопка «W» или «Стрелка вверх», движение вниз – «S» или «Стрелка вниз», а также использование предметов – кнопка «E» и установка факелов, для отметки пути – кнопка «F». На изображении В.1, приложения В представлен полный алгоритм движений.

## Проектирование интерфейса

При запуске игры, должно сразу показываться главное меню игры, чтобы игрок мог перед началом игры установить нужные ему параметры разрешения.

Так как запланирован минималистичный интерфейс, не должно быть ничего лишнего. Так же не должно быть и большого количества кнопок, так как иначе будет большое количество информации и пользователь может запутаться. На рисунке 2.4 представлен примерный вариант главного меню



**Рисунок 2.4. Эскиз интерфейса**



# Разработка и тестирование.

Данная глава обозревает реализацию и тестирование конечного продукта. Здесь рассматриваются все принципы создания программы и тестирование с проверкой полноты по критериям чёрного ящика. Результатом третьей главы является конечный продукт.



## Обоснование выбора средств разработки

Для достижения поставленной цели, в качестве среды проектирования игры было выбрано программное обеспечение Unity с поддержкой Visual Studio, на основе языка программирования С#.

* 1. Данная система была выбрана по критериям, представленным ниже:
  2. Большое количество документации и справочников, помогающие разобраться в написании игры.
  3. Наличие большого набора структурных типов данных.
  4. Лёгкость и удобность программного обеспечения.
  5. Лицензия программного обеспечения является бесплатной.
  6. Поддержка разработки двумерных игр.

Инструменты интегрированного Visual Studio помогают создавать профессиональный код и исправлять большее количество ошибок, упущенных на этапе написания кода. Инструменты проектирования Unity позволяют легко позиционировать разные объекты на сцене проекта[13]. Интегрированные в Unity системы отладки позволяют быстра компилировать и устранять любые дефекты[7].

## Описание используемых функций.

Любая функция, используемая в скриптах игры, в программном обеспечении Unity наследуется из одного определённого класса: MonoBehavior[11]. Таким образом, чтобы код исправно работал в начале каждого скрипта нужно явно наследовать данный класс (рисунок 3.1). Листинг программного продукта, включая все описанные далее функции, находится в Приложении Г. Листинг программы.

**Рисунок 3.1. Наследование класса MomoBehavior**

Для того чтобы определенный скрипт работал и вызывался в самой игре существуют обязательные функции:

1. Awake
2. Update

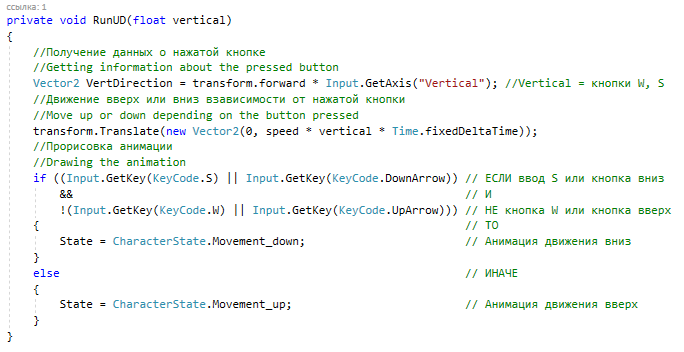
Функция «Awake» вызывается при загрузке экземпляра скрипта. Данная функция используется для инициализации переменных и состояния игры до того, как игра будет загружена. Так как эта функция вызывается всего один раз после запуска, а также в случайном порядке среди других скриптов, её лучше не использовать для передачи информации или установки связи между скриптами. Вместо неё можно использовать функцию «Start», которая позволяет упорядочить инициализацию скриптов[12].

Функция «Update» вызывается перед отрисовкой кадра игры и перед прорисовкой внимации объектов, используется для анимирования и просчета кадров статичных объектов. Для не статичных объектов существует отдельная функция «FixedUpdate», которая просчитывает физику и анимацию объекта и обновляется фиксированными по времени шагами[12].

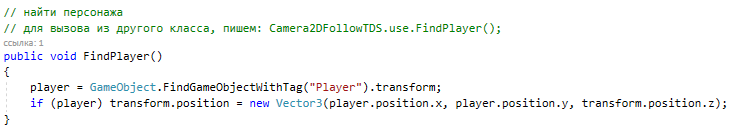
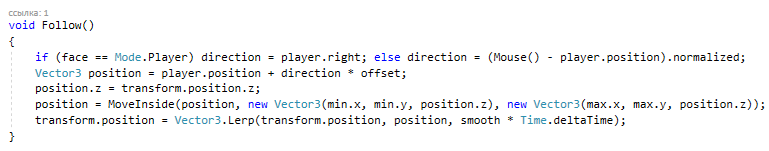
Одним из главных скриптов в игре, является скрипт для персонажа. В данном скрипте использовались следующие функции:

* 1. Awake – инициализирует персонажа, присваивает ему физическое тело и анимацию.
  2. FixedUpdate – отрисовывет кадр и ждет команды игрока, пока нет команды, запускает анимацию бездействия. Входными данными явдяются кнопки «W», «S», «A», «D», «E», «F» и «стрелка вверх», «стрелка вниз», «стрелка влево», «стрелка вправо». Выходными данными является сторона движения или взаимодействие с объектом.
  3. OnCollisionEnter2D – встроенная функция Unity, которая вызывается, когда один объект начал соприкосновение с другим. Содержит информацию о точках соприкосновения, скорости воздействия и т.д. Входными данными является коллизия объектов, а выходными разрешение на переход с уровня на уровень.
  4. RunLR – функция, предназначенная для передвижения влево и вправо, а так же прорисовки анимации в зависимости от направления движения. Входными данными являются кнопки «A», «D», «стрелка влево», «стрелка вправо».
  5. RunUD – функция, предназначенная для передвижения вверх и вниз, а так же отрисови анимации в зависимости от направления движения. Входными данными являются кнопки «W», «S», «стрелка вверх», «стрелка вниз».

**Рисунок 3.2. Функция FixedUpdate**

На рисунке 3.2 представлен фрагмент кода, отвечающий за основные действия персонажа. Таким образом видно, пока игрок ничего не нажал персонаж пребывает в положении бездействия. После того как игрок нажмет кнопки для движения, встроенный параметр, Input.GetButton(), позволяет узнать какие кнопки для движения были нажаты. Таким образом при нажатии кнопок, именуемых Vertical – кнопки «W», «S», «Стрелка вверх» и «Стрелка вниз» запускается функция RunUD представленная на рисунке 3.3.

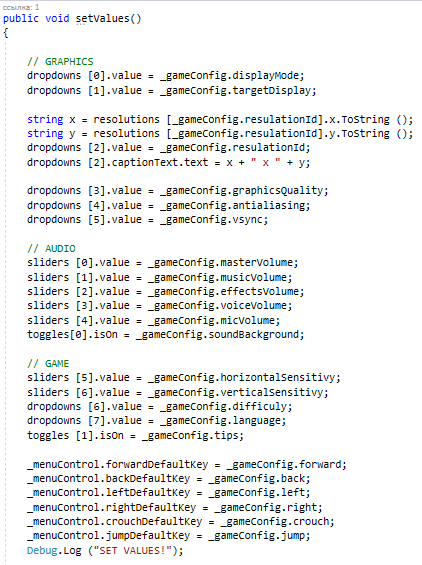
**Рисунок 3.3. Функция RunUD**

Для камеры, следящей за персонажем тоже потребовался скрипт. Данный скрипт (рисунок 3.4 и 3.5) находит персонажа и не дает камере уходить за черные рамки

**Рисунок 3.5. Функция для преследования игрока**

**Рисунок 3.4. Функция для поиска игрока**

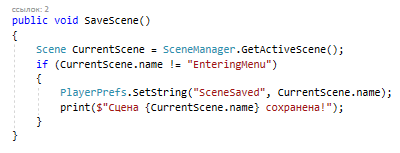
Меню стало самым трудозатратным, так как на него ушло больше всего сил и больше всего времени. Главные из скриптов это контроллер меню, меню настроек и сохранение игры.

На рисунке 3.6 показана функция, отвечающая за установление и редактирование настроек в игре. При изменении настроек в игре, строки принимают значение int, в зависимости от выбранного вами варианта.

**Рисунок 3.6. Функция для изменения настроек в игре**

Сохранение в игре происходит таким образом:

1. До первой загрузки новой игры кнопка «Продолжить» не работает.
2. После нажатия кнопки «Новая игра» игрок попадает на первый уровень.
3. На первом уровне игра сразу сохраняет номер уровня.
4. Далее на каждом уровне игра автоматически будет сохраняться.
5. При выключении игры и повторном её включении кнопка «Продолжить» уже будет работать.
6. Загружаться будет тот уровень на котором был игрок в последний раз.

На рисунке 3.7 показана функция для сохранения игры.

**Рисунок 3.6. Функция сохранения игры**

## Тестирование программного продукта.

Тестирование программы проводилось на основе критериев чёрного ящика на персональном компьютере с установленной операционной системой Windows 10. Само тестирование расположено в Приложении Д.

После написания кода в программе были реализованы все функции, определенные в предыдущих главах.

Так же помимо тестирования по критериям черного ящика было проведено альфа тестирование, в котором принимали участие учащиеся «НИУ ВШЭ-Пермь» и пользователи сети «Интернет», добровольно согласившиеся на участие[5]. Для этого тестирования специально была написана «Документация пользователя», представленная в приложении Е.

В ходе всего тестирования были выявлены ошибки в работе меню. При переключении языка переводились не все таблички и диалоговые окна (рисунок 3.7). После соответствующих правок изменений не последовало, из-за чего было предпринято решение убрать поддержку английского языка.

Так же была выявлена ошибка с проходом сразу в конец игры на первом уровне. Данная ошибка была сразу же устранена. Ошибка вызывалась из-за не правильного расположения границ объектов (рисунок 3.8)[2]/

**Рисунок 3.8. Помеченные границы объектов.**

**Рисунок 3.7. Ошибка с переводом с английского на русский**

# Заключение

По итогу данной работы, была разработана игра в жанре «Maze». Для этого были изучены основы игростроения и рассмотрена литература на эту тему. Произведен разбор современного программного обеспечения для создания игр, на основе которого делался вывод, в каком ПО лучше создавать игру. А также произведет обзор игр в жанре «Maze» для выявления плюсов и минусов, и заимствования некой игровой составляющей.

При проектировании были использована библиотека C#, необходимая для, написания кода приложения. Графический интерфейс был спроектирован и разработан благодаря инструментам Unity, с поддержкой Visual Studio.

Программу возможно усовершенствовать, добавив некоторое количество уровней, дорисовав текстуры и усложнив игровой процесс. Так же можно добавить поддержку разных языков. Но все же на данном этапе изучения программирования на C# разработанных функций приложения вполне достаточно, для стабильного функционирования игры.

В течении работы над программой были усовершенствованы навыки ООП и знания в сфере создания игр

Положительным результатом проведенной работы является усовершенствование навыков написания программ на объектно-ориентированном языке программирования C# и создания игр под Windows и MacOS. Изученный материал по разработке игры, однозначно пригодится в будущем, так как именно в этой сфере я планирую развиваться в будущем и преуспеть как программист..

# Список сокращений и условных обозначений

Action – жанр компьютерных игр с упором на эксплуатацию физических возможностей игрока, таких как координация глаз и рук или скорость реакции.

RPG – жанр компьютерных игр с упором ролевую игру с большим количеством диалогов, свободой в выборе путей решения различных задач, проработанным миром и сюжетом.

Альфа тестирование – работа с системой потенциальными пользователями. Чаще всего альфа-тестирование проводится на ранней стадии разработки продукта.

Game engine – базовое программное обеспечение компьютерной игры.

Casual – компьютерная игра, предназначенная для широкого круга пользователей. Казуальные игры не требуют от пользователя особой усидчивости и каких-либо особых навыков.

Action-adventure - смешанный жанр компьютерных игр, сочетающий элементы квеста и экшена.

ПО – программное обеспечение.

# Библиографический список

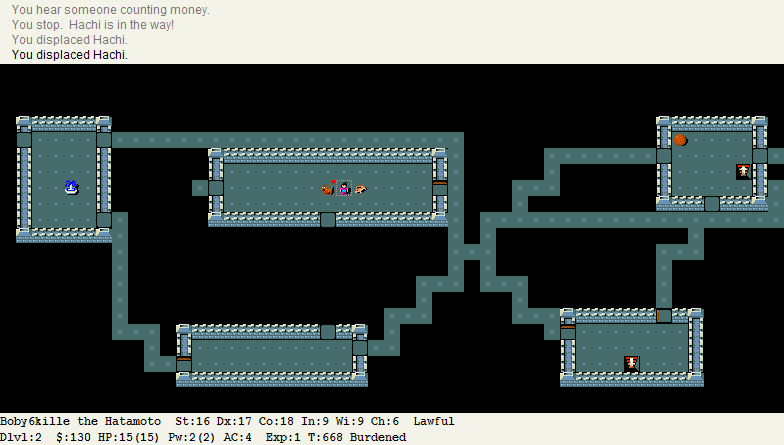
1. Арсаж, Ж. Программирование игр и головоломок/Ж. Арсаж, -М.: Книга по Требованию, 2012.
2. Бочкарев Н. А., Молотов Р. С. Подходы к трансформации объектов виртуальных пространств в среде Unity //Вестник Ульяновского государственного технического университета. – 2016. – №. 3 (75).
3. Георгий Евсеев. Лабиринты в компьютерных играх (рус.) // PC Review.: журнал. – 1994. – 1 февраля (№ 2)
4. Грунская В.И. Об отличимости плоских шахматных лабиринтов//Интеллектуальные системы. -2004. -Т. 8.
5. Грэтхем Д. Семь мифов о независимости спецификаций и тестирования//Открытые системы. СУБД. 2002. № 11.
6. Думиньш А. А., Зайцева Л. В. Компьютерные игры в обучении и технологии их разработки //Образовательные технологии и общество. – 2012. – Т. 15. – №. 3.
7. Канер С., Фолк Дж., Нгуен Е. К. Тестирование программного обеспечения/Пер. с англ. М.: ДиаСофт, 2001.
8. Лизяев С. Д., Молотов Р. С. Особенности создания анимации при разработке обучающих симуляторов в среде Unity //Вестник Ульяновского государственного технического университета. – 2016.
9. Сальникова Е.И. Особенности разработки персонажей для двумерных компьютерных игр.//Творчество молодых: дизайн, реклама, информационные технологии сборник трудов XIII Международной научно-практической конференции студентов и аспирантов. Научный редактор Л. М. Дмитриева. Омск, 2014.
10. Хокинг, Дж. Unity в действии. Мультиплатформенная разработка на C#/Дж. Хокинг, пер. с англ. И. Рузайкиной -СПб.: Питер,
11. Blow J. Game development: Harder than you think //Queue. – 2004. – Т. 1. – №. 10.
12. Claypool K., Claypool M. Teaching software engineering through game design //ACM SIGCSE Bulletin. – 2005. – Т. 37. – №. 3.
13. Hejlsberg A., Wiltamuth S., Golde P. The C# programming language. – 2006.

# Приложение А. Виды лабиринтов в играх

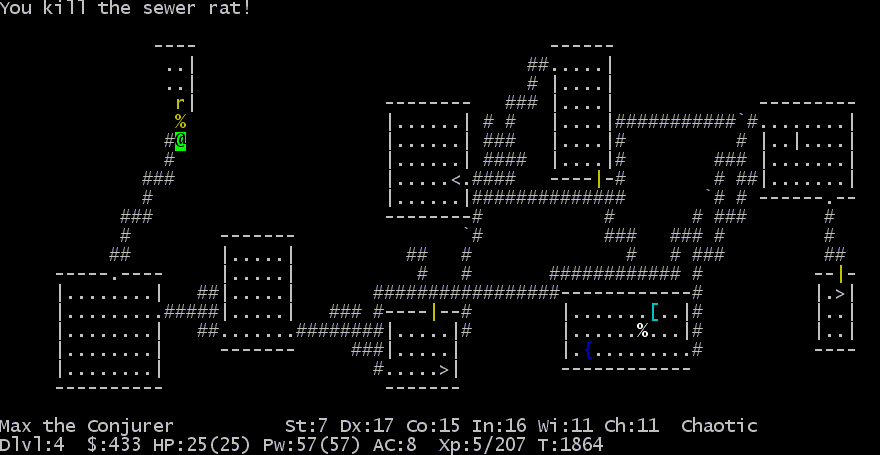
**Рисунок А.3. Пример лабиринта в изометрической графике**



**Рисунок А.2. Пример лабиринта в двумерной графике**



**Рисунок А.1. Пример лабиринта из текстовой графики**



# Приложение Б. Таблицы для сравнения ПО

**Таблица А.1. Доступность ПО**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Среда** | **Windows** | **Mac OS** | **Linux** | **Android** | **iOS** | **Xbox** | **PlayStation** | **Лицензия** |
| 1. Unity | да | да | да | да | да | да | да | бесплатная |
| 2. Unreal Engine | да | да | да | да | да | да | да | бесплатная |
| 3. CryEngine | да | нет | да | нет | нет | да | да | частично бесплатная |
| 4. Construct | да | нет | нет | да | да | нет | нет | платная |

**Таблица А.2. Основной функционал ПО**

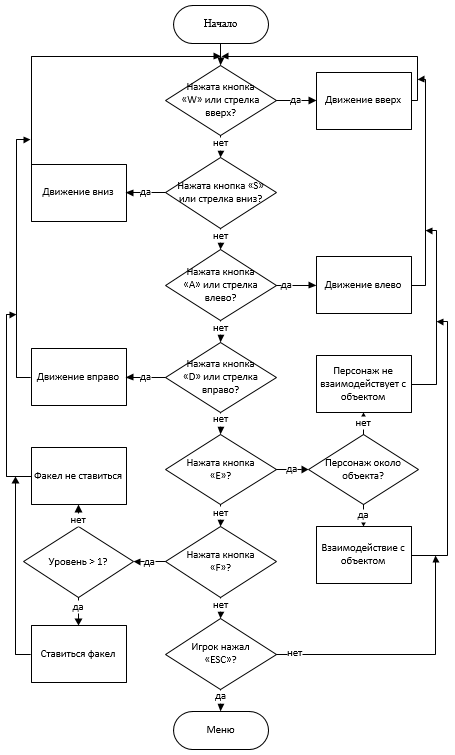
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Среда** | **Поддержка 2D** | **Поддержка 3D** | **Язык программирования** | **Наличие документации** | **Исправление ошибок** |
| 1. Unity | да | да | С# | да | постоянно |
| 2. Unreal Engine | частично | да | C++ | да | редко |
| 3. CryEngine | нет | нет | C++ | частично | редко |
| 4. Construct | да | нет | JavaScript | частично | очень редко |

**Таблица А.3. Удобность интерфейса ПО**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Среда** | **Интуитивность интерфейса** | **Удобность в использовании** | **Затраты времени** | **Наличие блочного программирования** | **Сложность создания** |
| 1. Unity | да | частично | средне | да | средне |
| 2. Unreal Engine | да | да | быстро | да | сложно |
| 3. CryEngine | нет | нет | долго | частично | сложно |
| 4. Construct | нет | да | средне | да | легко |

# Приложение В. Блок-схема движения персонажа

**Рисунок В.1. Блок схема движения**



# Приложение Г. Полный листинг скриптов игры

**Скрипт для камеры:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[RequireComponent(typeof(Camera))]

public class Camera2DFollowTDS : MonoBehaviour

{

private enum Mode { Player, Cursor };

[SerializeField] private Mode face; // вектор смещения, относительно "лица" персонажа или положения курсора

[SerializeField] private float smooth = 2.5f; // сглаживание при следовании за персонажем

[SerializeField] private float offset; // значение смещения (отключить = 0)

[SerializeField] private SpriteRenderer boundsMap; // спрайт, в рамках которого будет перемещаться камера

[SerializeField] private bool useBounds = true; // использовать или нет, границы для камеры

private Transform player;

private Vector3 min, max, direction;

private static Camera2DFollowTDS \_use;

private Camera cam;

public static Camera2DFollowTDS use

{

get { return \_use; }

}

void Awake()

{

\_use = this;

cam = GetComponent<Camera>();

cam.orthographic = true;

FindPlayer();

CalculateBounds();

}

// переключатель, для использования из другого класса

public void UseCameraBounds(bool value)

{

useBounds = value;

}

// найти персонажа, если он был уничтожен, а потом возрожден

// для вызова из другого класса, пишем: Camera2DFollowTDS.use.FindPlayer();

public void FindPlayer()

{

player = GameObject.FindGameObjectWithTag("Player").transform;

if (player) transform.position = new Vector3(player.position.x, player.position.y, transform.position.z);

}

// если в процессе игры, было изменено разрешение экрана

// или параметр "Orthographic Size", то следует сделать вызов данной функции повторно

public void CalculateBounds()

{

if (boundsMap == null) return;

Bounds bounds = Camera2DBounds();

min = bounds.max + boundsMap.bounds.min;

max = bounds.min + boundsMap.bounds.max;

}

Bounds Camera2DBounds()

{

float height = cam.orthographicSize \* 2;

return new Bounds(Vector3.zero, new Vector3(height \* cam.aspect, height, 0));

}

Vector3 MoveInside(Vector3 current, Vector3 pMin, Vector3 pMax)

{

if (!useBounds || boundsMap == null) return current;

current = Vector3.Max(current, pMin);

current = Vector3.Min(current, pMax);

return current;

}

Vector3 Mouse()

{

Vector3 mouse = Input.mousePosition;

mouse.z = -transform.position.z;

return cam.ScreenToWorldPoint(mouse);

}

void Follow()

{

if (face == Mode.Player) direction = player.right; else direction = (Mouse() - player.position).normalized;

Vector3 position = player.position + direction \* offset;

position.z = transform.position.z;

position = MoveInside(position, new Vector3(min.x, min.y, position.z), new Vector3(max.x, max.y, position.z));

transform.position = Vector3.Lerp(transform.position, position, smooth \* Time.deltaTime);

}

void FixedUpdate()

{

if (player)

{

Follow();

}

}

}

**Скрипт для персонажа в доме:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Character : MonoBehaviour

{

[SerializeField]

private int life = 1;

[SerializeField]

private float speed = 3.0F;

new private Rigidbody rigidbody;

private Animator animation;

private SpriteRenderer sprite;

private bool ready = true;

private CharacterState State

{

get

{

return (CharacterState)animation.GetInteger("State");

}

set

{

animation.SetInteger("State", (int)value);

}

}

private void OnCollisionEnter(Collision collision)

{

if (collision.gameObject.tag == "EndLevel")

{

ready = true;

}

}

private void Awake()

{

rigidbody = GetComponent<Rigidbody>();

animation = GetComponent<Animator>();

sprite = GetComponentInChildren<SpriteRenderer>();

Cursor.lockState = CursorLockMode.Locked;

}

private void FixedUpdate()

{

State = CharacterState.Idle;

float v = Input.GetAxis("Vertical");

if (Input.GetButton("Horizontal"))

{

RunLR();

}

if (Input.GetButton("Vertical"))

{

RunUD(v);

}

if (Input.GetKeyDown(KeyCode.E) && ready)

{

Application.LoadLevel(Application.loadedLevel + 1);

}

}

private void RunLR()

{

Vector3 HorizontDirection = transform.right \* Input.GetAxis("Horizontal");

transform.position = Vector3.MoveTowards(transform.position, transform.position + HorizontDirection, speed \* Time.deltaTime);

sprite.flipX = HorizontDirection.x < 0.0F;

State = CharacterState.Movement\_right;

}

private void RunUD(float vertical)

{

Vector2 VertDirection = transform.forward \* Input.GetAxis("Vertical");

transform.Translate(new Vector2(0, speed \* vertical \* Time.fixedDeltaTime)); // движение по лестнице

if (Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow) && !(Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow)))

{

State = CharacterState.Movement\_down;

}

else

{

State = CharacterState.Movement\_up;

}

if ((Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow)) && !(Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow)))

{

State = CharacterState.Movement\_up;

}

else

{

State = CharacterState.Movement\_down;

}

}

}

public enum CharacterState

{

Idle,

Movement\_right,

Movement\_left,

Movement\_up,

Movement\_down

}

**Скрипт для персонажа в лабиринте:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Character\_ad : MonoBehaviour

{

[SerializeField]

private int life\_ad = 1;

[SerializeField]

private float speed\_ad = 3.0F;

private Rigidbody rigidbody\_ad;

private Animator animation\_ad;

private SpriteRenderer sprite\_ad;

private bool ready\_ad = false;

private CharState State

{

get

{

return (CharState)animation\_ad.GetInteger("State");

}

set

{

animation\_ad.SetInteger("State", (int)value);

}

}

private void OnCollisionEnter(Collision collision)

{

if (collision.gameObject.tag == "endLevel")

{

ready\_ad = true;

}

}

private void Awake()

{

rigidbody\_ad = GetComponent<Rigidbody>();

animation\_ad = GetComponent<Animator>();

sprite\_ad = GetComponentInChildren<SpriteRenderer>();

//Cursor.lockState = CursorLockMode.Locked;

}

private void FixedUpdate()

{

State = CharState.Idle\_ad;

float v = Input.GetAxis("Vertical");

if (Input.GetButton("Horizontal"))

{

RunLR\_ad();

}

if (Input.GetButton("Vertical"))

{

RunUD\_ad(v);

}

if (Input.GetKeyDown(KeyCode.E) && ready\_ad)

{

Application.LoadLevel(Application.loadedLevel + 1);

}

}

private void RunLR\_ad()

{

Vector3 HorizontDirection = transform.right \* Input.GetAxis("Horizontal");

transform.position = Vector3.MoveTowards(transform.position, transform.position + HorizontDirection, speed\_ad \* Time.deltaTime);

State = CharState.Movement\_left\_ad;

sprite\_ad.flipX = HorizontDirection.x > 0.0F;

}

private void RunUD\_ad(float vertical)

{

Vector2 VertDirection = transform.forward \* Input.GetAxis("Vertical");

transform.Translate(new Vector2(0, speed\_ad \* vertical \* Time.fixedDeltaTime)); // движение по лестнице

if (Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow) && !(Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow)))

{

State = CharState.Movement\_down\_ad;

}

else

{

State = CharState.Movement\_up\_ad;

}

if ((Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow)) && !(Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow)))

{

State = CharState.Movement\_up\_ad;

}

else

{

State = CharState.Movement\_down\_ad;

}

}

}

public enum CharState

{

Idle\_ad,

Movement\_right\_ad,

Movement\_left\_ad,

Movement\_up\_ad,

Movement\_down\_ad

}

**Скрипт для изменения клавиш:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class ChangeKeyboard : MonoBehaviour {

public CustomInput customInput;

public Button[] button;

public bool canChangeKey;

public KeyCode newKey;

private Event eventKey;

public int indexChange=0;

// Use this for initialization

void Start () {

canChangeKey = false;

SetKeys ();

UpdateTextUI ();

}

// Update is called once per frame

void Update () {

}

void OnGUI(){

button [0].onClick.AddListener (delegate {

ChangeKeyButton (0);

});

button [1].onClick.AddListener (delegate {

ChangeKeyButton (1);

});

button [2].onClick.AddListener (delegate {

ChangeKeyButton (2);

});

button [3].onClick.AddListener (delegate {

ChangeKeyButton (3);

});

button [4].onClick.AddListener (delegate {

ChangeKeyButton (4);

});

button [5].onClick.AddListener (delegate {

ChangeKeyButton (5);

});

ChangeKey ();

if (Input.anyKeyDown) {

if (canChangeKey == true) {

if (indexChange == 0) {

customInput.forward = newKey;

} else if (indexChange == 1) {

customInput.back = newKey;

} else if (indexChange == 2) {

customInput.right = newKey;

} else if (indexChange == 3) {

customInput.left = newKey;

} else if (indexChange == 4) {

customInput.crouch = newKey;

} else if (indexChange == 5) {

customInput.jump = newKey;

}

SetKeyDefault ();

UpdateTextUI ();

button [indexChange].enabled = true;

canChangeKey = false;

}

}

}

public void ChangeKeyButton(int indexKey){

indexChange = indexKey;

canChangeKey = true;

button [indexKey].GetComponentInChildren<Text> ().text = "WAIT FOR KEY";

button [indexKey].enabled = false;

}

void ChangeKey(){

eventKey = Event.current;

newKey = eventKey.keyCode;

}

void UpdateTextUI(){

button [0].GetComponentInChildren<Text> ().text = customInput.forward.ToString ();

button [1].GetComponentInChildren<Text> ().text = customInput.back.ToString ();

button [2].GetComponentInChildren<Text> ().text = customInput.right.ToString ();

button [3].GetComponentInChildren<Text> ().text = customInput.left.ToString ();

button [4].GetComponentInChildren<Text> ().text = customInput.crouch.ToString ();

button [5].GetComponentInChildren<Text> ().text = customInput.jump.ToString ();

}

public void SetKeyDefault(){

customInput.forwardDefaultKey = customInput.forward.ToString ();

customInput.backDefaultKey = customInput.back.ToString ();

customInput.rightDefaultKey = customInput.right.ToString ();

customInput.leftDefaultKey = customInput.left.ToString ();

customInput.crouchDefaultKey = customInput.crouch.ToString ();

customInput.jumpDefaultKey = customInput.jump.ToString ();

}

public void SetKeys(){

customInput.forward = (KeyCode) System.Enum.Parse(typeof(KeyCode), customInput.forwardDefaultKey);

customInput.back = (KeyCode) System.Enum.Parse(typeof(KeyCode), customInput.backDefaultKey);

customInput.right = (KeyCode) System.Enum.Parse(typeof(KeyCode), customInput.rightDefaultKey);

customInput.left = (KeyCode) System.Enum.Parse(typeof(KeyCode), customInput.leftDefaultKey);

customInput.crouch = (KeyCode) System.Enum.Parse(typeof(KeyCode), customInput.crouchDefaultKey);

customInput.jump = (KeyCode) System.Enum.Parse(typeof(KeyCode), customInput.jumpDefaultKey);

}

}

**Скрипт для изменения языка:**

using System.Collections;

using System.Collections.Generic;

using System.IO;

using UnityEngine;

using UnityEngine.UI;

public class LanguageControl : MonoBehaviour {

//[Header("JSON File Names")]

//public string mainScreenLanguage = "MainScreenLanguage";

//public string optionsScreenLanguage = "OptionsScreenLanguages";

public Text[] main\_titles;

public Text[] buttons\_options;

public Text[] options\_video;

public Text[] options\_audio;

public Text[] options\_game;

public WS\_MAIN\_LANGUAGE ws\_main\_language;

public WS\_OPTIONS\_LANGUAGE ws\_options\_language;

private MenuControl \_menuControl;

private OptionsControl \_optionsControl;

//Editor

[HideInInspector]

public int currentTab;

[HideInInspector]

public int currentTabTwo;

[HideInInspector]

public int previousTab = -1;

[HideInInspector]

public int previousTabTwo = -1;

// Use this for initialization

void Start () {

ws\_main\_language = new WS\_MAIN\_LANGUAGE ();

ws\_options\_language = new WS\_OPTIONS\_LANGUAGE ();

\_menuControl = this.GetComponent<MenuControl> ();

\_optionsControl = this.GetComponent<OptionsControl> ();

}

// Update is called once per frame

void Update () {

}

// SET ALL CHANGES

public void SetLanguageInGame(){

SetMainLanguage ();

SetOptionsLanguage ();

}// END

// MAIN MENU

public void SetMainLanguage(){

if (\_optionsControl != null && \_optionsControl.\_gameConfig != null && ws\_main\_language != null) {

if (\_optionsControl.\_gameConfig.language == 0) { // ENGLISH

//Debug.Log ("Change To English!");

if (\_menuControl.inGame == false)

{

main\_titles[0].text = ws\_main\_language.play\_en;

main\_titles[1].text = ws\_main\_language.options\_en;

\_menuControl.menu[0].SetText(ws\_main\_language.continue\_en);

\_menuControl.menu[1].SetText(ws\_main\_language.play\_en);

\_menuControl.menu[2].SetText(ws\_main\_language.options\_en);

\_menuControl.menu[3].SetText(ws\_main\_language.credits\_en);

\_menuControl.menu[4].SetText(ws\_main\_language.exit\_en);

}

else

{

\_menuControl.menu[0].SetText(ws\_main\_language.resume\_en);

\_menuControl.menu[1].SetText(ws\_main\_language.options\_en);

\_menuControl.menu[2].SetText(ws\_main\_language.exit\_en);

}

} else if (\_optionsControl.\_gameConfig.language == 1) { // PTBR

//Debug.Log ("Change To PTBR");

if (\_menuControl.inGame == false) // not Pause Menu

{

main\_titles[0].text = ws\_main\_language.play\_ptbr;

main\_titles[1].text = ws\_main\_language.options\_ptbr;

\_menuControl.menu[0].SetText(ws\_main\_language.continue\_ptbr);

\_menuControl.menu[1].SetText(ws\_main\_language.play\_ptbr);

\_menuControl.menu[2].SetText(ws\_main\_language.options\_ptbr);

\_menuControl.menu[3].SetText(ws\_main\_language.credits\_ptbr);

\_menuControl.menu[4].SetText(ws\_main\_language.exit\_ptbr);

}

else // In Pause Menu

{

\_menuControl.menu[0].SetText(ws\_main\_language.resume\_ptbr);

\_menuControl.menu[1].SetText(ws\_main\_language.options\_ptbr);

\_menuControl.menu[2].SetText(ws\_main\_language.exit\_ptbr);

}

}

}

}// END

// OPTIONS LANGUAGE

public void SetOptionsLanguage(){

if (\_optionsControl != null && \_optionsControl.\_gameConfig != null && ws\_main\_language != null) {

if (\_optionsControl.\_gameConfig.language == 0) { // ENGLISH

// BUTTONS

buttons\_options[0].text = ws\_options\_language.video\_en;

buttons\_options[1].text = ws\_options\_language.audio\_en;

buttons\_options[2].text = ws\_options\_language.game\_en;

buttons\_options[3].text = ws\_options\_language.apply\_en;

buttons\_options[4].text = ws\_options\_language.return\_en;

// VIDEO

options\_video[0].text = ws\_options\_language.displayMode\_en;

options\_video[1].text = ws\_options\_language.targetDisplay\_en;

options\_video[2].text = ws\_options\_language.resolution\_en;

options\_video[3].text = ws\_options\_language.graphicsQuality\_en;

options\_video[4].text = ws\_options\_language.antialiasing\_en;

options\_video[5].text = ws\_options\_language.vsync\_en;

// AUDIO

options\_audio [0].text = ws\_options\_language.masterVolume\_en;

options\_audio [1].text = ws\_options\_language.musicVolume\_en;

options\_audio [2].text = ws\_options\_language.effectsVolume\_en;

options\_audio [3].text = ws\_options\_language.voiceVolume\_en;

options\_audio [4].text = ws\_options\_language.micVolume\_en;

options\_audio [5].text = ws\_options\_language.soundBackground\_en;

// GAME

options\_game [0].text = ws\_options\_language.horizontalSensitivy\_en;

options\_game [1].text = ws\_options\_language.vericalSensitivy\_en;

options\_game [2].text = ws\_options\_language.difficulty\_en;

options\_game [3].text = ws\_options\_language.language\_en;

options\_game [4].text = ws\_options\_language.forward\_en;

options\_game [5].text = ws\_options\_language.back\_en;

options\_game [6].text = ws\_options\_language.left\_en;

options\_game [7].text = ws\_options\_language.right\_en;

options\_game [8].text = ws\_options\_language.crouch\_en;

options\_game [9].text = ws\_options\_language.jump\_en;

options\_game [10].text = ws\_options\_language.tips\_en;

} else if (\_optionsControl.\_gameConfig.language == 1) { // RUS

// BUTTONS

buttons\_options[0].text = ws\_options\_language.video\_ptbr;

buttons\_options[1].text = ws\_options\_language.audio\_ptbr;

buttons\_options[2].text = ws\_options\_language.game\_ptbr;

buttons\_options[3].text = ws\_options\_language.apply\_ptbr;

buttons\_options[4].text = ws\_options\_language.return\_ptbr;

// VIDEO

options\_video[0].text = ws\_options\_language.displayMode\_ptbr;

options\_video[1].text = ws\_options\_language.targetDisplay\_ptbr;

options\_video[2].text = ws\_options\_language.resolution\_ptbr;

options\_video[3].text = ws\_options\_language.graphicsQuality\_ptbr;

options\_video[4].text = ws\_options\_language.antialiasing\_ptbr;

options\_video[5].text = ws\_options\_language.vsync\_ptbr;

// AUDIO

options\_audio [0].text = ws\_options\_language.masterVolume\_ptbr;

options\_audio [1].text = ws\_options\_language.musicVolume\_ptbr;

options\_audio [2].text = ws\_options\_language.effectsVolume\_ptbr;

options\_audio [3].text = ws\_options\_language.voiceVolume\_ptbr;

options\_audio [4].text = ws\_options\_language.micVolume\_ptbr;

options\_audio [5].text = ws\_options\_language.soundBackground\_ptbr;

// GAME

options\_game [0].text = ws\_options\_language.horizontalSensitivy\_ptbr;

options\_game [1].text = ws\_options\_language.vericalSensitivy\_ptbr;

options\_game [2].text = ws\_options\_language.difficulty\_ptbr;

options\_game [3].text = ws\_options\_language.language\_ptbr;

options\_game [4].text = ws\_options\_language.forward\_ptbr;

options\_game [5].text = ws\_options\_language.back\_ptbr;

options\_game [6].text = ws\_options\_language.left\_ptbr;

options\_game [7].text = ws\_options\_language.right\_ptbr;

options\_game [8].text = ws\_options\_language.crouch\_ptbr;

options\_game [9].text = ws\_options\_language.jump\_ptbr;

options\_game [10].text = ws\_options\_language.tips\_ptbr;

}

}// END

}

}

using UnityEngine;

public class WS\_MAIN\_LANGUAGE {

// ENGLISH

public string continue\_en = " CONTINUE";

public string play\_en = " PLAY";

public string options\_en = " OPTIONS";

public string credits\_en = " CREDITS";

public string exit\_en = " QUIT";

public string resume\_en = "RESUME";

// RUS

public string continue\_ptbr = " Продолжить";

public string play\_ptbr = " Новая игра";

public string options\_ptbr = " Настройки";

public string credits\_ptbr = " Титры";

public string exit\_ptbr = " Выйти";

public string resume\_ptbr = "Продолжить";

}

using UnityEngine;

public class WS\_OPTIONS\_LANGUAGE {

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ENGLISH \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\\

public string video\_en = "VIDEO";

public string audio\_en = "AUDIO";

public string game\_en = "GAME";

public string apply\_en = "APPLY";

public string return\_en = "RETURN";

// VIDEO

public string displayMode\_en = " DISPLAY MODE";

public string targetDisplay\_en = " TARGET DISPLAY";

public string resolution\_en = " RESOLUTION";

public string graphicsQuality\_en = " GRAPHICS QUALITY";

public string antialiasing\_en = " ANTIALIASING";

public string vsync\_en = " V-Sync";

// AUDIO

public string masterVolume\_en = " MASTER VOLUME";

public string musicVolume\_en = " MUSIC VOLUME";

public string effectsVolume\_en = " EFFECTS VOLUME";

public string voiceVolume\_en = " VOICE VOLUME";

public string micVolume\_en = " MIC VOLUME";

public string soundBackground\_en = " SOUND BACKGROUND";

// GAME

public string horizontalSensitivy\_en = " HORIZONTAL SENSITIVY";

public string vericalSensitivy\_en = " VERTICAL SENSITIVY";

public string difficulty\_en = " DIFFICULTY";

public string language\_en = " LANGUAGE";

public string forward\_en = " UP";

public string back\_en = " DOWN";

public string left\_en = " LEFT";

public string right\_en = " RIGHT";

public string crouch\_en = " CROUCH";

public string jump\_en = " JUMP";

public string tips\_en = " TIPS";

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RUS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\\

public string video\_ptbr = "Видео";

public string audio\_ptbr = "Аудио";

public string game\_ptbr = "Игра";

public string apply\_ptbr = "Принять изменения";

public string return\_ptbr = "Назад";

// VIDEO

public string displayMode\_ptbr = " Дисплей";

public string targetDisplay\_ptbr = " Монитор";

public string resolution\_ptbr = " Разрешение";

public string graphicsQuality\_ptbr = " Качество графики";

public string antialiasing\_ptbr = " Сглаживание";

public string vsync\_ptbr = " V-Sync";

// AUDIO

public string masterVolume\_ptbr = " Общая громкость";

public string musicVolume\_ptbr = " Громкость музыки";

public string effectsVolume\_ptbr = " Громкость эффектов";

public string voiceVolume\_ptbr = " Громкость голоса";

public string micVolume\_ptbr = " Громкость микрофона";

public string soundBackground\_ptbr = " Громкость окружения";

// GAME

public string horizontalSensitivy\_ptbr = " Горизонтальная чувствительность";

public string vericalSensitivy\_ptbr = " Вертикальная чувствительность";

public string difficulty\_ptbr = " Сложность";

public string language\_ptbr = " Язык";

public string forward\_ptbr = " Вверх";

public string back\_ptbr = " Вниз";

public string left\_ptbr = " Налево";

public string right\_ptbr = " Направо";

public string crouch\_ptbr = " Присесть";

public string jump\_ptbr = " Прыжок";

public string tips\_ptbr = " Атака";

}

**Скрипт для меню:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.Events;

[System.Serializable]

public class Menu : MonoBehaviour {

[Header("Menu Settings")]

public bool active; // Responsible variable if the button is active or not

public int maxFontSize = 22; // Maximum font size

public int minFontSize = 18; // Minimum font size

public Color mouseEnter; // MenuItem color when the mouse enters

public Color mouseExit; // MenuItem color when the mouse exits

public Color mousePressed; // MenuItem color when the mouse pressed / click

public Color deactivatedColor; // MenuItem color when disabled

[Header("Line Settings")]

public bool enableLine; // Enable underline

public bool enableLineEffect; // Enable effect when mouse enters

public float widthLine=200f; // Width underline

public float heightLineMin=1f; // Minimum height underline

public float heightLineMax=2f; // Maximum height underline

[Header("Animation Settings")]

public bool initAnim; // Initial animation

public float timerInitAnim; // Time for animation starts

public float menuXStart = -233f; // Initial X-axis

public float menuXEnd = 115f; // End X-axis

public float speedAnim = 400f; // Speed animation

[Header("Events")]

[SerializeField]

private UnityEvent Enter = new UnityEvent ();

[SerializeField]

private UnityEvent Exit = new UnityEvent ();

[SerializeField]

private UnityEvent Click = new UnityEvent ();

// Variables that the user does not need to change

private MenuControl \_menuc; // Menu Control Component

private Image \_effectSelected; // Underline Component

private Text \_text; // Text Component

private Vector3 \_initPos; // Initial position

private RectTransform \_rect; // RectTransform component

// Use this for initialization

void Start () {

getComponents ();

basicSettings ();

}// END

// Update is called once per frame

void Update () {

if (initAnim == true) {

updateAnimation ();

}

}// END

//-----------------------------------------------------------------------------START METHODS MENUITEM--------------------------------------------------------------------\\

// Get the components

void getComponents(){

\_rect = this.GetComponent<RectTransform> (); // Get the RectTransform component of this object

\_menuc = FindObjectOfType<MenuControl> (); // Get the Control Menu (there should only be one)

\_text = this.GetComponent<Text> (); // Get the Text component of children

\_effectSelected = this.GetComponentInChildren<Image> (); // Get the Image component of children

}// END

// Basic and necessary settings

void basicSettings(){

\_initPos = \_rect.localPosition; // Get initial position

if (initAnim == true) { // If the initial animation is true

\_rect.localPosition = new Vector3(menuXStart, \_initPos.y, \_initPos.z); // Arrow the position of the object to the X axis of the variable "menuXStart"

}

//Set Default Color

\_text.color = mouseExit;

\_effectSelected.color = mouseExit;

// If the button is not active

if (active == false) {

\_text.color = deactivatedColor; // Set color to "deactivatedColor"

}

// If the underline is false it defines the effect of the underline as false too

if (enableLine == false) {

\_effectSelected.enabled = false;

}

}// END

// Update initial animation

void updateAnimation(){

// If the time to start the animation is over

if (timerInitAnim <= 0) {

\_rect.transform.localPosition = Vector2.MoveTowards (\_rect.transform.localPosition, new Vector2 (menuXEnd, \_initPos.y), speedAnim \* Time.deltaTime); // Starts Animation

}

if (timerInitAnim >= 0) {timerInitAnim -= Time.deltaTime;} // If the animation time is greater than zero (ie not started) it starts to go down 1 second

}// END

//-----------------------------------------------------------------------------END METHODS MENUITEM----------------------------------------------------------------------\\

//-----------------------------------------------------------------------------START METHODS ON/OFF----------------------------------------------------------------------\\

// Method by setting the default menuItem again (mouseExit)

public void menuDisable(){

if (active == true) { // If the button is active

\_text.color = mouseExit; // Sets the default color

\_effectSelected.color = mouseExit; // Sets the default color (underline)

if (enableLineEffect==true) { // If the underline effect is active

\_effectSelected.rectTransform.sizeDelta = new Vector2 (widthLine, heightLineMin); // Set a new size

}

\_text.fontSize = minFontSize; // Sets the default font size

}

}// END

// Method by setting the active menuItem (mouseEnter)

public void menuEnable(){

if (active == true) { // If the button is active

\_text.color = mouseEnter; // Sets new color (mouseEnter)

\_effectSelected.color = mouseEnter; // Sets new color for underline (mouseEnter)

if (enableLineEffect==true) { // If the underline effect is active

\_effectSelected.rectTransform.sizeDelta = new Vector2 (widthLine, heightLineMax); // Set a new size

}

\_text.fontSize = maxFontSize; // Sets font size to max font size

}

}// END

// Set underline is active or no

public void setMenuLine(bool value){

enableLine = value; // Set value

\_effectSelected.gameObject.SetActive (value); // Set value

}// END

// Activate an object and mask it

public void enableObject(GameObject obj){

obj.SetActive (true); // Active the object

if (\_menuc.inGame == false)

{

\_menuc.mask.SetActive(true); // Active the mask

\_menuc.setAlphaMask(0.5f); // Set alpha of mask to 0.5f

}

}// END

//-----------------------------------------------------------------------------END METHODS ON/OFF------------------------------------------------------------------------\\

public void CallTheEvent(int index){

if (index == 0) {

Enter.Invoke ();

} else if (index == 1) {

Exit.Invoke ();

} else if (index == 2) {

Click.Invoke ();

}

}// END

public void SetText(string newText){

this.GetComponent<Text> ().text = newText;

}

public void showMessageInConsole(string s){

Debug.Log (s);

}

}

**Скрипт для контроллера меню:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using System.IO;

using UnityEngine.SceneManagement;

public enum PlaysType{

Direct, Modes, LevelIsIncomplete

}

public enum CreditsAnimationType

{

Direction, Scale

}

public enum DirectionCredits{

up, down

}

[RequireComponent(typeof(OptionsControl))]

[RequireComponent(typeof(LanguageControl))]

[RequireComponent(typeof(MenuSelect))]

[RequireComponent(typeof(LevelSelect))]

[RequireComponent(typeof(UITransition))]

public class MenuControl : CustomInput {

[Header("Objects Settings")]

[Space]

public GameObject mask; // Mask to make non clicable and dark effect

public GameObject mainScreen; // Object of the main screen

public GameObject creditsScreen; // Object of the credits screen

public GameObject modesScreen;

public GameObject levelScreen;

public GameObject optionsScreen;

public GameObject creditsTextBack; // Object of text credits "Back to go main screen"

public GameObject Canvas;

[Header("Global Menu Settings")]

[Space]

//[Tooltip("This option disables underscores in the menu items")]

//public bool disableLine; // Underline in the menu items

public bool useCustomCursor=true; // Enable custom cursor?

public bool useKeyboard = false; // Enable navigation using keyboard

[HideInInspector]

public bool isActiveCanvas = false; // Enable canvas/pause?

public bool inGame = false;

public Dropdown language;

public Texture2D cursor; // Custom Cursor Image

public Menu[] menu = new Menu[0]; // List of all menu item

[Header("Play Settings")]

[Space]

[Tooltip("If there is a game mode (example: Singleplayer, multiplayer, training), it is disabled")]

//public bool noModes = true; // There are game modes

public PlaysType playsType;

[Tooltip("If you turn on every time you click play, the initial animation in the mode screen is active")]

public bool resetAnim = true; // Do you want to enable / disable animation of the modes screen whenever you click play

public string nameLoadLevel; // Name of the game level you want to load

public Mode[] modes; // List of all modes of game

[Header("Credits Settings")]

[Space]

public CreditsAnimationType animationType; // Type of animations for Credits Screen

//public bool animCreditsScale = true; // Enable scaling animation

//public bool animDirectionCredits = true; // Enable direction animation

public float YCreditsEnd; // Y axis where the direction animation ends

public DirectionCredits dirCredits; // Direction where credits go

public float initScale = 0.2f; // Initial Scale

public float maxScale = 1f; // Maximum scale

public float speedAnimScaleCredits = 0.75f; // Scale animation speed

public float speedAnimDirCredits = 50f; // Steering animation speed

[Header("Audio Manager")]

[Space]

public bool canPlayBackgroundSong=true;

public AudioSource backgroundAudio;

public AudioSource[] audio = new AudioSource[0];

[Header("Custom Input")]

[Space]

// Variables that the user does not need to change

private UITransition \_uitransition;

// Cursor

private Vector2 \_hotSpot = Vector2.zero;

// Credits

private Vector3 \_initCreditsPos; // Variable responsible for the initial position of creditScreen

private RectTransform \_rectCredits; // Variable responsible for the RectTransform component

private float \_currentScale; // Variable responsible for the current scale of Animation Credits Scale

private bool \_startScaleAnimCredits = false; // Responsible variable if scaling animation can start

private bool \_startDirAnimCredits = false; // Responsible variable if the steering animation can start

private bool \_inCreditsScreen = false; // Responsible variable if the credits screen is active or not

private bool \_inModesScreen = false; // Responsible variable if the modes screen is active or not

private bool \_inLevelScreen = false; // Responsible variable if the level screen is active or not

private bool \_inOptionsScreen = false; // Responsible variable if the options screen is active or not

//Editor

[HideInInspector]

public int currentTab;

[HideInInspector]

public int currentTabTwo;

[HideInInspector]

public int previousTab = -1;

[HideInInspector]

public int previousTabTwo = -1;

[Header("Organize Menu")]

public int spaceMenu=45;

public int xSpace=100;

public int yAdjust=0;

[Header("Organize Modes")]

public int spaceModes=250;

public int xStart=250;

// Use this for initialization

void Start () {

basicSettings ();

firstCreditsSettings ();

reportErrors ();

if(inGame == true)

Canvas.SetActive (isActiveCanvas);

}

// Update is called once per frame

void Update () {

//-------------------------------------------------------------------------START MODES/LEVEL: METHODS CALL---------------------------------------------------------------\\

quitModesScreen ();

quitLevelScreen ();

//-------------------------------------------------------------------------END MODES: METHODS CALL-----------------------------------------------------------------------\\

//-------------------------------------------------------------------------START OPTIONS: METHODS CALL-------------------------------------------------------------------\\

quitOptionsCredits ();

//-------------------------------------------------------------------------END OPTIONS: METHODS CALL---------------------------------------------------------------------\\

//-------------------------------------------------------------------------START CREDITS: METHODS CALL-------------------------------------------------------------------\\

updateAnimScaleCredits (); // Animation of scale for the Credits

updateAnimDirectionCredits (); // Animation of direction for the Credits

quitScreenCredits (); // Exit credit screen

//-------------------------------------------------------------------------END CREDITS: METHODS CALL---------------------------------------------------------------------\\

UpdateCanvasEnable();

}

//-----------------------------------------------------------------------------START METHODS MENU CONTROL--------------------------------------------------------------------\\

// Set the basics settings

void basicSettings(){

// Active or deactivated first objets in scene

mainScreen.SetActive (true);

creditsScreen.SetActive (false);

creditsTextBack.SetActive (false);

modesScreen.SetActive (false);

levelScreen.SetActive (false);

optionsScreen.SetActive (false);

QualitySettings.masterTextureLimit = 1;

if (!File.Exists (Application.persistentDataPath + "/gamesettings.json")) {

GameConfig \_gameConfig;

\_gameConfig = new GameConfig ();

string jsonData = JsonUtility.ToJson(\_gameConfig, true); // Converts the class to the json file

File.WriteAllText (Application.persistentDataPath + "/gamesettings.json", jsonData); // Save the jsonData (class \_gameConfig) in persistenceData

Debug.LogWarning ("MENU\_KIT: gamesettings.json was created!");

}

if (inGame == false)

setAlphaMask (0.5f);

else

setAlphaMask (0.7f);

// Get all components

\_rectCredits = creditsScreen.GetComponent<RectTransform>();

\_uitransition = this.GetComponent<UITransition> ();

// Set Custom Cursor

if(useCustomCursor == true && inGame == false)

Cursor.SetCursor(cursor, \_hotSpot, CursorMode.ForceSoftware);

// Play background song

if (canPlayBackgroundSong) {

playBackgroundAudio ();

Debug.Log ("Background Audio is Play");

}

}// END

void UpdateCanvasEnable()

{

if (inGame == true)

{

if (Input.GetKeyDown(KeyCode.Escape))

{

CanvasCloseOpen ();

setAlphaMask(0.7f);

}

}

}

public void CanvasCloseOpen()

{

isActiveCanvas = !isActiveCanvas;

if (isActiveCanvas == false) {

Cursor.SetCursor(null, Vector2.zero, CursorMode.Auto);

Time.timeScale = 1;

Cursor.lockState = CursorLockMode.Locked;

}

else {

mask.SetActive(true);

Cursor.lockState = CursorLockMode.None;

Cursor.SetCursor(null, Vector2.zero, CursorMode.ForceSoftware);

Time.timeScale = 0;

}

Canvas.SetActive(isActiveCanvas);

//resetMenu();

}

// Report all erros in console

void reportErrors(){

if (mask == null) {Debug.LogError ("MENUKIT: Variable 'mask' is not declared");}

if (mainScreen == null) {Debug.LogError ("MENUKIT: Variable 'mainScreen' is not declared");}

if (creditsScreen == null) {Debug.LogError ("MENUKIT: Variable 'creditsScreen' is not declared");}

if (modesScreen == null) {Debug.LogError ("MENUKIT: Variable 'modesScreen' is not declared");}

if (creditsTextBack == null) {Debug.LogError ("MENUKIT: Variable 'creditsTextBack' is not declared");}

}// END

// RESET ALL MENUS TO STANDARD STATE

void resetMenu(){

foreach (Menu m in menu) { // Scrolls through the entire menu list

m.menuDisable (); // Become standard

}

}// END

// RESET ALL MODE TO STANDARD STATE

void resetMode(){

foreach (Mode m in modes) { // Scrolls through the entire modes list

m.mouseExit (); // Become standard

}

}// END

// Set Alpha Mask with value

public void setAlphaMask(float value){

mask.GetComponent<CanvasRenderer>().SetAlpha(value); // Set Alpha of CanvasRenderer

}

//-------------------------------------------------------------------------END METHODS MENU CONTROL----------------------------------------------------------------------------\\

//-------------------------------------------------------------------------START METHODS PLAY, MODES, LEVEL SCREEN--------------------------------------------------------------\\

// Play Button

public void StartGame(){

if (playsType == PlaysType.Direct) {

fadeInEffect (); // Load Level Game

} else if (playsType == PlaysType.LevelIsIncomplete) {

setLevelScreen (true); // The level screen is active

} else if (playsType == PlaysType.Modes) {

setModesScreen (true); // The mode screen is active

}

}// END

// Fade Effect

void fadeInEffect(){

// Start Fade Effect

mask.SetActive (true);

\_uitransition.Play();

}// END

// Loads game level

public void loadLevelGame(){

if (nameLoadLevel != null) { // If the name of the game level to be loaded is not empty

//Application.LoadLevel (nameLoadLevel); // Load game level

SceneManager.LoadScene (Application.loadedLevel + 1, LoadSceneMode.Single);

} else { // ELSE

Debug.LogWarning ("Variable nameLoadLevel is "+nameLoadLevel); // Warns that the variable is empty

}

}// END

// Updating the Modes Exit Button

void quitModesScreen(){

// If you press the "esc" key and the modes screen is active

if (\_inModesScreen == true) {

if (Input.GetKeyDown (KeyCode.Escape)) {

setModesScreen (false); // Disable modes screen

}

}

} // END

// Updating the Level Exit Button

void quitLevelScreen(){

// If you press the "esc" key and the modes screen is active

if (\_inLevelScreen == true) {

if (Input.GetKeyDown (KeyCode.Escape)) {

setLevelScreen (false); // Disable level screen

}

}

} // END

// Changes between the main screen and the modes screen

void setModesScreen(bool value){

if (value) {

mainScreen.SetActive (false); // Disables the Main Screen

modesScreen.SetActive (true); // Active the Mode Screen

\_inModesScreen = true; // In modeScreen

resetMode(); // Reset all modes for the standart state

if (resetAnim) { // If the option to enable / reset the initial animation of the modes screen

foreach (Mode mode in modes) { // It passes through all modes modes within the variable "modes"

mode.isAnim = true; // Can move

mode.resetAnim (); // And call the method by resetting position and time

}

}

} else {

modesScreen.SetActive (false); // Disables the Modes Screen

mainScreen.SetActive (true); // Active the Main Screen

\_inModesScreen = false; // Quit mode screen

resetMenu(); // Reset all menus to standart state

}

}// END

// Changes between the main screen and the level select screen

void setLevelScreen(bool value){

if (value) {

mainScreen.SetActive (false); // Disables the Main Screen

levelScreen.SetActive (true); // Active the Level Screen

\_inLevelScreen = true; // In levelScreen

} else {

mainScreen.SetActive (true); // Disables the Modes Screen

levelScreen.SetActive (false); // Active the Main Screen

//\_inLevelScreen = false; // Quit level screen

resetMenu(); // Reset all menus to standart state

}

}// END

//-------------------------------------------------------------------------END METHODS PLAY/MODES SCREEN---------------------------------------------------------------------\\

//-------------------------------------------------------------------------START METHODS CREDITS SCREEN----------------------------------------------------------------------\\

// Starts initial credit settings

void firstCreditsSettings(){

\_initCreditsPos = \_rectCredits.localPosition; // Get First Position of init Credits

if (animationType == CreditsAnimationType.Scale) {

\_rectCredits.localScale = new Vector2 (initScale, initScale); // Set scale for initScale

\_currentScale = initScale;

}

} // END

// Update the scale animation of credits

void updateAnimScaleCredits(){

if (animationType == CreditsAnimationType.Scale) { // If a scale animation to active

if (\_currentScale <= maxScale && \_startScaleAnimCredits == true) { // If the scale is smaller than the maximum scale and you can start the scale animation

\_currentScale += speedAnimScaleCredits \* Time.deltaTime; // The "currentScale" variable begins to increase

\_rectCredits.localScale = new Vector2 (\_currentScale, \_currentScale); // The scale of the credits object is updated with the variable "currentScale"

}

if (\_currentScale >= maxScale) { // If the scale of the object is already in the defined size

\_startDirAnimCredits = true; // Move animation can start

}

}

} // END

// Updating the Credits Exit Button

void quitScreenCredits(){

// If you press the "esc" key or the credits finish the direction animation and the credit screen is active

if (\_inCreditsScreen == true) {

if (Input.GetKeyDown (KeyCode.Escape) || \_rectCredits.localPosition.y >= YCreditsEnd) {

setCreditsScreen (false); // Disable credit screen

}

}

} // END

// Update the direction animation of credits

void updateAnimDirectionCredits(){

if (animationType == CreditsAnimationType.Direction && \_startDirAnimCredits == true) { // If the steering animation is active and it can start

if (dirCredits == DirectionCredits.up) { // If the animation is up

// Rises with speed defined in variable speedAnimDirCredits

\_rectCredits.localPosition = new Vector2 (\_rectCredits.localPosition.x, \_rectCredits.localPosition.y + (speedAnimDirCredits \* Time.deltaTime));

} else { // If the animation is down

// Descends with the velocity defined in the variable speedAnimDirCredits

\_rectCredits.localPosition = new Vector2 (\_rectCredits.localPosition.x, \_rectCredits.localPosition.y - (speedAnimDirCredits \* Time.deltaTime));

}

}

} // END

// Changes between the main screen and the credits screen

public void setCreditsScreen(bool value){

// Go to Credits Screen

if (value == true)

{

// All back to the initial configuration

\_rectCredits.localPosition = \_initCreditsPos; // Return to starting position

if (animationType == CreditsAnimationType.Scale)

{ // If scale animation is active

\_rectCredits.localScale = new Vector2(initScale, initScale); // Back to credits scale for the initial

\_currentScale = initScale; // The current scale has the same value as the initial

\_startScaleAnimCredits = true; // Scaling animation starts

}

else

{ // ELSE

\_startDirAnimCredits = true; // Direction animation starts

}

// Active or deactivated the objects

creditsTextBack.SetActive(true); // Active text "back to go main screen"

mainScreen.SetActive(false); // Disables the Main Screen

creditsScreen.SetActive(true); // Active the Credits Screen

if (inGame == false)

{

setAlphaMask(0.5f);

mask.SetActive(true); // Active the Mask

}

\_inCreditsScreen = true; // Credits screen is active

}

else

{ // Go to Main Screen

// Animations are stopped

\_startScaleAnimCredits = false; // Scaling animation is disabled

\_startDirAnimCredits = false; // Direction animation is disabled

// Active or deactivated the objects

creditsTextBack.SetActive(false); // Disables text "back to go main screen"

creditsScreen.SetActive(false); // Disables the Credits Screen

mainScreen.SetActive(true); // Active the Main Screen

if (inGame == false)

mask.SetActive(false); // Disables the Mask

\_inCreditsScreen = false; // Credits screen is disables

resetMenu(); // Reset all menus to standart state

}

}// END

//-------------------------------------------------------------------------END METHODS CREDITS SCREEN------------------------------------------------------------------------\\

//-------------------------------------------------------------------------START METHODS OPTIONS SCREENS---------------------------------------------------------------------\\

public void setOptionsScreen(bool value){

if (value) { // If true

mainScreen.SetActive (false); // Disable main screen

optionsScreen.SetActive (true); // Active Options Screen

\_inOptionsScreen = true; // On in Options Screen

} else { // ELSE

optionsScreen.SetActive (false); // Disable Options Screen

mainScreen.SetActive (true); // Active Main Screen

\_inOptionsScreen = false; // Exits of Options Screen

resetMenu (); //

}

}// END

// Updating the Credits Exit Button

void quitOptionsCredits(){

// If you press the "esc" key or the credits finish the direction animation and the credit screen is active

if (\_inOptionsScreen == true) {

if (Input.GetKeyDown (KeyCode.Escape)) {

setOptionsScreen (false); // Disable credit screen

}

}

}// END

//-------------------------------------------------------------------------END METHODS OPTIONS SCREENS-----------------------------------------------------------------------\\

// Play audio by index in list AUDIO

public void playAudio(int index){

audio [index].Play ();

}// END

// Play background song

public void playBackgroundAudio(){

if (backgroundAudio != null) {

backgroundAudio.Play ();

}

}// END

public void SetDefaultValues(){

useCustomCursor = true;

resetAnim = false;

nameLoadLevel = "Scenes/Home/Home";

animationType = CreditsAnimationType.Direction;

YCreditsEnd = 400;

dirCredits = DirectionCredits.up;

initScale = 0f;

maxScale = 1f;

speedAnimDirCredits = 0.75f;

speedAnimDirCredits = 50f;

spaceMenu=45;

xSpace=100;

yAdjust=0;

spaceModes=250;

xStart=250;

}

public void OrganizeMenus(){

for (int i = 0; i < menu.Length; i++){

float a = -spaceMenu \* i;

menu [i].transform.localPosition = new Vector3 (xSpace, yAdjust+a, menu[i].transform.localPosition.z);

}

Debug.Log ("Menus Organized!");

}

public void OrganizeModes(){

for (int i = 0; i < modes.Length; i++){

float a = spaceModes \* i;

modes [i].transform.localPosition = new Vector3 (-xStart+a, modes[i].transform.localPosition.y, modes[i].transform.localPosition.z);

}

Debug.Log ("Modes Organized!");

}

void OnDrawGizmosSelected(){

}

}

using System.Collections;

using System.Collections.Generic;

using System.IO;

using UnityEngine;

using UnityEngine.Events;

using UnityEngine.UI;

public class OptionsControl : MonoBehaviour {

[Header("Basic Settings")]

public CanvasScaler canvasScaler; // Get CanvasScaler component

public GameObject[] panelsOptions; // All options panels(example: Video, Audio and Game)

[Space]

public GameObject[] buttons; // All buttons of menu

public Color colorNormal, colorSelected; // Color(Default ou Selected) of buttons

//public Button applyButton; // Apply button for confirm changes in settings

[Header("Events")]

[SerializeField]

private UnityEvent InitSettings = new UnityEvent();

[Header("Options Settings")]

public Text[] amounts; // All amounts of sliders

public Slider[] sliders; // All sliders

public Dropdown[] dropdowns; // All dropdowns

public Toggle[] toggles; // All checkbox

public Vector2[] resolutions; // All resolutions available

private Display[] \_displays; // All displays available

[HideInInspector]public GameConfig \_gameConfig; // GameConfig responsible for storing the settings

private int \_selected = 0; // Variables responsible for knowing which menu (Video, Audio and Game) is selected

private MenuControl \_menuControl;

private LanguageControl \_languageControl;

// EDITOR

[HideInInspector]

public int currentTab, currentTabTwo, previousTab=-1, previousTabTwo=-1;

// Use this for initialization

void Start () {

basicSettings (); // Call the method

loadConfig (); // Call the method

}// END

// Update is called once per frame

void Update () {

updateAmountsText (); // Call the method

changeSettingsGame ();

}// END

//------------------------------------------------------------------------START METHODS OPTIONSCONTROL--------------------------------------------------------------\\

void basicSettings(){

changeSelected (\_selected); // Set Menu Selected

addListeners (); // Add Listerners

\_gameConfig = new GameConfig (); // Create class GameConfig

\_displays = Display.displays; // Get all displays available

\_menuControl = FindObjectOfType<MenuControl> ();

\_languageControl = this.GetComponent<LanguageControl> ();

addResolutions ();

addDisplays (); // Call the method

}// END

//------------------------------------------------------------------------END METHODS OPTIONSCONTROL----------------------------------------------------------------\\

//------------------------------------------------------------------------START METHODS ADD DROPDOWN----------------------------------------------------------------\\

// Adds resolutions in dropdown

public void addResolutions(){

dropdowns [2].ClearOptions ();

foreach (Vector2 resolution in resolutions) { // Walks by all resolutions in vector2

dropdowns [2].options.Add (new Dropdown.OptionData (resolution.x+" x "+resolution.y)); // Add resolution in dropdown and transform to string

}

}// END

// Adds all displays in dropdown

void addDisplays(){

for (int i = 0; i < \_displays.Length; i++) { // Walks by all diaplys

dropdowns[1].options.Add(new Dropdown.OptionData("MONITOR "+(i+1))); // Adds the display as "MONITOR" and the identification number

if (i == 0) { // If it is the first monitor added

dropdowns[1].captionText.text = "MONITOR " + (i + 1); // Makes it selected

}

}

}// END

//------------------------------------------------------------------------END METHODS ADD DROPDOWN------------------------------------------------------------------\\

//------------------------------------------------------------------------START METHODS MENU------------------------------------------------------------------------\\

// Change menu

public void changeSelected(int sel){

\_selected = sel; // Put temporary variable in \_selected

foreach (GameObject go in panelsOptions) { // Walk by all game objects in panelsOptions

go.SetActive (false); // Disables

}

foreach (GameObject go2 in buttons) { // Walk by all game objects in buttons

go2.GetComponent<Image> ().color = colorNormal; // Define to default color(

}

panelsOptions [\_selected].SetActive (true); // Activate the panel according to a variable

buttons [\_selected].GetComponent<Image> ().color = colorSelected; // Put colorSelected on the active button

}// END

// Updates all amount text

public void updateAmountsText(){

for (int i = 0; i <= amounts.Length-1; i++) { // Walks up to the length of the vector amounts

amounts [i].text = "" + sliders [i].value.ToString("F2"); // Sets the text of the amounts as the value of the slider and I define F2 formatting (example: 0.50)

}

}// END

// Adds listeners (You can do it manually)

public void addListeners(){

//applyButton.onClick.AddListener(delegate {changeButtonApply();}); // Add listener by clicking the button

}// END

// By clicking the apply button

public void changeButtonApply(){

saveGameConfig (); // Call the method

saveConfig (); // Call the method

changeSettingsGame (); // Call the method

}// END

//------------------------------------------------------------------------END METHODS MENU--------------------------------------------------------------------------\\

//------------------------------------------------------------------------START METHODS JSON------------------------------------------------------------------------\\

// Saves settings to JSON file

public void saveConfig(){

string jsonData = JsonUtility.ToJson(\_gameConfig, true); // Converts the class to the json file

File.WriteAllText (Application.persistentDataPath + "/gamesettings.json", jsonData); // Save the jsonData (class \_gameConfig) in persistenceData

}// END

// Load settings JSON file

public void loadConfig(){

// Load the json file in the path where it is located and store it in the variable \_gameConfig

if (File.Exists (Application.persistentDataPath + "/gamesettings.json")) {

\_gameConfig = JsonUtility.FromJson<GameConfig> (File.ReadAllText (Application.persistentDataPath + "/gamesettings.json"));

} else {

Debug.LogWarning ("MENU\_KIT: gamesettings.json does not exists!");

}

setValues (); // Call the method

}// END

// Save all values in \_gameConfig(Class GameConfig)

public void saveGameConfig(){

// GRAPHICS

\_gameConfig.displayMode = dropdowns [0].value;

\_gameConfig.targetDisplay = dropdowns [1].value;

\_gameConfig.resulationId = dropdowns [2].value;

\_gameConfig.graphicsQuality = dropdowns [3].value;

\_gameConfig.antialiasing = dropdowns [4].value;

\_gameConfig.vsync = dropdowns [5].value;

// AUDIO

\_gameConfig.masterVolume = sliders [0].value;

\_gameConfig.musicVolume = sliders [1].value;

\_gameConfig.effectsVolume = sliders [2].value;

\_gameConfig.voiceVolume = sliders [3].value;

\_gameConfig.micVolume = sliders [4].value;

\_gameConfig.soundBackground = toggles[0].isOn;

// GAME

\_gameConfig.horizontalSensitivy = sliders [5].value;

\_gameConfig.verticalSensitivy = sliders [6].value;

\_gameConfig.difficuly = dropdowns[6].value;

\_gameConfig.language = dropdowns [7].value;

\_gameConfig.tips = toggles[1].isOn;

\_gameConfig.forward = \_menuControl.forwardDefaultKey;

\_gameConfig.back = \_menuControl.backDefaultKey;

\_gameConfig.left = \_menuControl.leftDefaultKey;

\_gameConfig.right = \_menuControl.rightDefaultKey;

\_gameConfig.crouch = \_menuControl.crouchDefaultKey;

\_gameConfig.jump = \_menuControl.jumpDefaultKey;

}// END

// Sets the values according to the variable \_gameConfig(class GameConfig)

public void setValues(){

// GRAPHICS

dropdowns [0].value = \_gameConfig.displayMode;

dropdowns [1].value = \_gameConfig.targetDisplay;

string x = resolutions [\_gameConfig.resulationId].x.ToString ();

string y = resolutions [\_gameConfig.resulationId].y.ToString ();

dropdowns [2].value = \_gameConfig.resulationId;

dropdowns [2].captionText.text = x + " x " + y;

dropdowns [3].value = \_gameConfig.graphicsQuality;

dropdowns [4].value = \_gameConfig.antialiasing;

dropdowns [5].value = \_gameConfig.vsync;

// AUDIO

sliders [0].value = \_gameConfig.masterVolume;

sliders [1].value = \_gameConfig.musicVolume;

sliders [2].value = \_gameConfig.effectsVolume;

sliders [3].value = \_gameConfig.voiceVolume;

sliders [4].value = \_gameConfig.micVolume;

toggles[0].isOn = \_gameConfig.soundBackground;

// GAME

sliders [5].value = \_gameConfig.horizontalSensitivy;

sliders [6].value = \_gameConfig.verticalSensitivy;

dropdowns [6].value = \_gameConfig.difficuly;

dropdowns [7].value = \_gameConfig.language;

toggles [1].isOn = \_gameConfig.tips;

\_menuControl.forwardDefaultKey = \_gameConfig.forward;

\_menuControl.backDefaultKey = \_gameConfig.back;

\_menuControl.leftDefaultKey = \_gameConfig.left;

\_menuControl.rightDefaultKey = \_gameConfig.right;

\_menuControl.crouchDefaultKey = \_gameConfig.crouch;

\_menuControl.jumpDefaultKey = \_gameConfig.jump;

Debug.Log ("SET VALUES!");

}// END

//------------------------------------------------------------------------END METHODS JSON--------------------------------------------------------------------------\\

//------------------------------------------------------------------------START METHODS SET CONFIG IN GAME----------------------------------------------------------\\

// Change settings in game

public void changeSettingsGame(){

InitSettings.Invoke ();

/\*

\* if(\_gameConfig.soundBackground == false)

{

\_menuControl.backgroundAudio.Stop ();

}

else

{

if (!\_menuControl.backgroundAudio.isPlaying)

{

\_menuControl.backgroundAudio.Play();

}

}

\*/

}// END

// Changes screen mode in game

public void changeDisplayMode(){

if (dropdowns [0].value == 0) {

Screen.fullScreen = true;

//Debug.Log ("Screen.fullScreen: true");

} else if (dropdowns [0].value == 1) {

Screen.fullScreen = false;

//Debug.Log ("Screen.fullScreen: false");

}

//Debug.Log (Screen.fullScreen);

//Debug.Log ("Dropdown[0] Value: "+dropdowns[0].value);

}// END

// Change the target display in the game

public void changeTargetDisplay(){

if (Camera.current != null) {

Camera.current.targetDisplay = \_gameConfig.targetDisplay;

}

}// END

// Changes resolution in game

public void changeResolution(){

// Sets the resolution according to the resolutions variable and the index defined in the dropdown

Screen.SetResolution ((int)resolutions[\_gameConfig.resulationId].x, (int)resolutions[dropdowns [2].value].y, Screen.fullScreen);

// Update Canvas Scaler with the new resolution

//canvasScaler.referenceResolution = new Vector2 (resolutions[\_gameConfig.resulationId].x, resolutions[dropdowns [2].value].y);

}// END

// Changes game quality in game

public void changeGraphicsQuality(){

QualitySettings.masterTextureLimit = \_gameConfig.graphicsQuality; // Sets the quality according to \_gameConfig (class GameConfig)

}// END

// Change antialiasing in game

public void changeAntiAliasing(){

QualitySettings.antiAliasing = \_gameConfig.antialiasing; // Sets the antialiasing according to \_gameConfig (class GameConfig)

}// END

// Change VSYNC in game

public void changeVSYNC(){

QualitySettings.vSyncCount = \_gameConfig.vsync; // Sets the vsync to \_gameConfig (class GameConfig)

}// END

public void ChangeMasterVolume(){

AudioListener.volume = sliders[0].value;

}

public void ChangeMusicVolume(){

\_menuControl.backgroundAudio.volume = sliders [1].value;

//Debug.Log (\_menuControl.backgroundAudio.volume);

//Debug.Log (sliders[1].name);

}

public void ChangeSoundBackground(){

Application.runInBackground = toggles [0].isOn;

}

public void ChangeLanguage(){

\_languageControl.SetLanguageInGame ();

}

//------------------------------------------------------------------------END METHODS SET CONFIG IN GAME-----------------------------------------------------------\\

}

**Скрипт для конфигураций:**

using UnityEngine;

public class GameConfig

{

// VIDEO \\

public int displayMode = 0;

public int targetDisplay = 0;

public int resulationId = 0;

public int graphicsQuality = 0;

public int antialiasing = 0;

public int vsync = 0;

public bool toggleTest = false;

// AUDIO \\

public float masterVolume = 1f;

public float musicVolume = 0.5f;

public float effectsVolume = 0.5f;

public float voiceVolume = 0.5f;

public float micVolume = 0.5f;

public bool soundBackground = true;

// GAME \\

public float horizontalSensitivy = 1f;

public float verticalSensitivy = 1f;

public int difficuly = 0;

public int language = 0;

public bool tips = true;

// INPUT \\

public string forward = "W";

public string back = "S";

public string left = "A";

public string right = "D";

public string crouch = "LeftControl";

public string jump = "Space";

}

# Приложение Д. Тестирование по критериям черного ящика

**Таблица Д.1. Чёрный ящик**

| **№ Теста** | **Действие** | **Ожидаемый результат** | **Реальный результат** |
| --- | --- | --- | --- |
| **Т1** | Нажатие кнопки «Продолжить» без сохранения | Кнопка не нажимается | Кнопка не нажимается |
| **Т2** | Нажатие кнопки «Продолжить» с сохранением | Загрузка сохраненного уровня | Загрузка сохраненного уровня |
| **Т3** | Нажатие кнопки «Новая игра» | Загрузка первого уровня | Загрузка первого уровня |
| **Т4** | Нажатие кнопки «Настройки» | Открытие панели настроек | Открытие панели настроек |
| **Т5** | Нажатие кнопки «Выход» | Сообщение о выходе | Сообщение о выходе |
| **T6** | Изменение настроек аудио и кнопка «Принять изменения» | Изменения принимаются | Изменения принимаются |
| **Т7** | Изменение настроек видео и кнопка «Принять изменения» | Изменения принимаются | Изменения принимаются |
| **Т8** | Изменение настроек игры и кнопка «Принять изменения» | Изменения принимаются | Изменения принимаются |
| **T9** | Изменение настроек аудио без кнопки «Принять изменения» | Изменения не принимаются | Изменения принимаются |
| **T10** | Изменение настроек видео и кнопки «Принять изменения» | Изменения не принимаются | Изменения не принимаются |
| **T11** | Изменение настроек игры и кнопки «Принять изменения» | Изменения не принимаются | Изменения не принимаются |
| **Т12** | Движение персонажем через стены | Персонаж упирается в стену | Персонаж упирается в стену |
| **T13** | Беспорядочное нажатие на все кнопки | Сообщение об ошибке | Залипание клавиш |
| **T14** | Подключение геймпада | С него можно управлять | С него можно управлять |
| **Т15** | Закрытие через диспетчер задач | Сохранение было произведено | Сохранение было произведено |

# Приложение Е. Документация пользователя

## Пользовательская документация

Данная игра является курсовой работой студента первого курса федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский университет «Высшая школа экономики» факультета экономики, менеджмента и бизнес-информатики направления программная инженерия группы ПИ-18-2 Чепокова Е.С

Документация описана по ГОСТ РД 50-34.698-90.

### **Введение**

Компьютерная игра «Maze» – offline игра для платформ Windows и MacOS. Игра используется в развлекательных целях

Помимо данной пользовательской документации не предусмотрено эксплуатационной документации, необходимой к ознакомлению.

### **Назначение и условия применения**

Данный вариант игры подготовлен для выявления ошибок и их последующего исправления. Документация написана с целю информирования о способах установить игру и способах связаться с разработчиком.

### **Подготовка к работе**

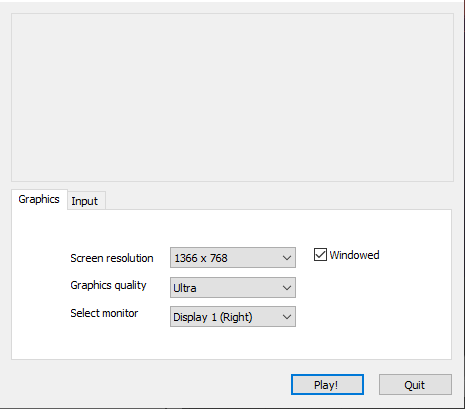
1. Скачивание игры

Скачивание файла с игрой осуществляется на сайте игры [bloodiesproject.gq](http://bloodiesproject.gq). Для скачивания нужно нажать на кнопку скачать. После чего вас направит на сайт Яндекс диска, откуда и можно скачать сжатый файл расширения .zip с игрой.

1. Установка

Клиент для Windows представляет собой в zip-архив. Распакуйте клиент средствами операционной системы или утилитами архивирования.

### **Начало игры**

1. Для запуска игры используйте исполняемый файл Maze.exe, расположенный в папке игры.
2. После запуска последует данное окно (рисунок Е.1), в нем нужно выбрать разрешение и монитор для отображения
3. После запуска, в главном меню выбираете кнопку «Новая игра» и начинаете играть

**Рисунок Е.1. Конфигурация игры**

### **Для экстренных ситуаций**

Действия по восстановлению программы при обнаружении ошибок в данных

При неуспешной попытке выполнить какую-либо функцию рекомендуется перезапустить приложение.

При любых аварийных ситуациях писать разработчику, с описанием проблемы в любой форме на почту или в сообщения «Вконтакте»:

[bloodiesco@yandex.ru](mailto:bloodiesco@yandex.ru)

<https://vk.com/elikch>