Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные Системы и Сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

«Сетевая игра «Крокодил»»

БГУИР КП 1-40 01 01  121  ПЗ

Студент: гр. 451006 Поняков Д.С.

Руководитель: асс. Перцев И.Ю.

Минск 2016

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc453322062)

[1 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc453322063)

[1.1 Описание предметной области 4](#_Toc453322064)

[1.2 Классификация компьютерных игр 5](#_Toc453322065)

[1.3 Игровой цикл 7](#_Toc453322066)

[1.4 Цель и задачи 8](#_Toc453322067)

[1.4.1 Цель 8](#_Toc453322068)

[1.4.2 Задачи 9](#_Toc453322069)

[2 РАЗРАБОТКА АЛГОРИТМА 10](#_Toc453322070)

[2.1 Алгоритм работы сети 10](#_Toc453322071)

[2.2 Диграмма экранов (UI/UX) 10](#_Toc453322072)

[2.3 Диаграмма базовых классов для управления игровым процессом 10](#_Toc453322073)

[2.4 Разработка игровых механизмов 11](#_Toc453322074)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 12](#_Toc453322075)

[3.1 Протокол взаимодействия пользователей 13](#_Toc453322076)

[4ТЕХНИЧЕСКИЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ 13](#_Toc453322077)

[4.1 Асинхронный обработчик сообщений 14](#_Toc453322078)

[4.2 Обновление состояния игровых объектов 14](#_Toc453322079)

[4.3 Контроль состояния игры 14](#_Toc453322080)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 16](#_Toc453322081)

[5.1 Главное меню 16](#_Toc453322082)

[5.2 Настройки игры 16](#_Toc453322083)

[5.3 Окна авторизации и регистрации 17](#_Toc453322084)

[5.4 Окно выбора партнера 18](#_Toc453322085)

[5.5 Меню выбора уровней 19](#_Toc453322086)

[5.6 Игровой процесс 19](#_Toc453322087)

[ЗАКЛЮЧЕНИЕ 21](#_Toc453322088)

[СПИСОК ЛИТЕРАТУРЫ 22](#_Toc453322089)

[ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ 23](#_Toc453322090)

# ВВЕДЕНИЕ

Компьютерные игры – это одна из составляющих нашей повседневной жизни. В 21 веке компьютерные развлечения стали популярными среди всех возрастов: начиная с детей и заканчивая людьми пенсионного возраста. Для одних – это работа (разработчики), для других – это способ отвлечься и отдохнуть, а для третьих – это большая возможность раздвинуть границы реальности!

В последнее время принято считать, что компьютерные игры – это своего рода вредная зависимость, которая стоит на ряду с алкоголизмом и наркоманией. На самом деле всё гораздо проще. Всё зависит от времени, проведённого за игрой: во всём должна быть «золотая середина». Но не хотелось бы начинать заезженную тему о компьютерной зависимости, а наоборот, рассказать о возможностях, которые нам дают компьютерные игры.

Рассмотрим самые полезные грани этого развлечения.

Во-первых, логические компьютерные игры развивают умственные навыки как ребёнка, так и взрослого, умение складывать цепочки событий, принимать обдуманные решения, иметь в голове разные способы решения ситуаций. К таким играм можно отнести «Судоку», «Пазлы», «Майджонг» и т.д.

Во-вторых, это яркий способ изучения различных областей науки и языков. Не зря на различных курсах быстрого обучения прибегают к игровому методу, т.е. изучение посредством игры.

В-третьих, это реальная возможность людям с ограниченными возможностями почувствовать себя на некоторое время такими как все. В этом помогают различного рода симуляторы. Они раздвигают границы возможностей и помогают хоть и виртуально, но всё же окунуться в жизнь среднестатистического человека.

И конечно же невозможно забыть о том, что любимые игры нам приносят удовольствие, помогают расслабиться, скоротать время, почувствовать дух соперничества, а также забыть о повседневных заботах хоть на некоторое время.

Особое внимание хотелось бы уделить играм, которые сплачивают людей, помогают научиться работать в команде и чувствовать ответственность перед другим. Такого рода игр очень мало в данное время. Чтобы найти хотя бы одну, пришлось потратить уйму времени. И так возникла идея написать нечто своё, нечто новое и очень увлекательное.

Данная курсовая работа решит проблему недостатка игр в команде, а также могла бы быть хорошим тренингом для достижения сплочённости в коллективе.

Процесс игры имеет ряд особенностей. Выделим основные из них, а также задачи, которые возлагаются на данную игру:

* Для прохождения уровней необходимо два игрока.
* Умение работать сообща.
* Быстро организовывать план действий.
* Ловкость рук
* Скорость реакции

# 1 ПОСТАНОВКА ЗАДАЧИ

## 1.1 Описание предметной области

Суть многопользовательских игр состоит в том, что в игре принимает участие несколько человек. Классифицировать многопользовательские игры можно по разным критериям: по технической реализации, по связи и по организации связи.

Крокодил – компьютерная многопользовательская игра, рассчитанная на двух людей, работающих в команде.

Прежде чем приступить к игре, игроки должны пройти процесс регистрации, выбрать себе игровое имя и пароль от своей учетной записи. Затем игрок может выбрать себе наиболее понравившуюся раскраску крокодила – персонажа, за которого ему предстоит играть.

В игре не предусмотрена возможность прохождения без активного подключения к интернету, так как учетные данные пользователей хранятся на сервере в интернете, там же и расположены данные о сессиях активных пользователях, информация о игровых сессиях двух конкретных игроков, и их общие достижения в виде сохранения игрового процесса.

Можно выделить основной перечень целей, которых игроки должны стремиться достигнуть в процессе игры:

* Умение работать в команде
* Умение нести ответственность за свои действия
* Умение быстро организовывать план действий
* Тренировать моторику пальцев
* Умение приспосабливаться к новой физике
* Расчёт уровня опасности и учёт времени

После того, как пользователь прошел регистрацию, он может начать игру и просмотреть список всех пользователей, а также выделить только тех пользователей, кто на данный момент в сети, чтобы подключиться к ним и начать совместную игру.

## 1.2 Классификация компьютерных игр

* Многопользовательская игра на одном компьютере (Splitscreen)

При таком типе игры пользователи играют на одном устройстве, при этом реализация может различаться.

Это могут быть несколько игровых окон на несколько игроков, либо оба игрока играют на одном экране, используя разную раскладку клавиатуры, или, что гораздо более удобно, джойстики.

В другом случае, это может быть пошаговая игра с поочередным переключением активного пользователя.

* Сетевая игра

Такой тип игры гораздо более удобен, так как игрокам не приходится делить общее физическое пространство, они могут быть далеко друг от друга, и связываться друг с другом через удаленные сервера, по локальной сети, беспроводные интерфейсы типа Bluetooth и т.д.

**По правилам:**

* По одной «жизни»

В этом режиме игроки совсем не взаимодействуют друг с другом, каждый ведёт свою игру; когда игрок теряет «жизнь», управление передаётся напарнику;

* Кооперативная игра

Цель игроков общая: вместе победить компьютерных соперников. Этот тип игры появился намного раньше сетевых игр;

* Личные соревновательные игры

Интересы игроков пересекаются, и каждый игрок старается победить остальных;

* Командные соревновательные игры

Игроки объединяются в команды, соревнующиеся друг с другом;

* Игры с полной свободой

Объединение в команды и война друг с другом полностью лежат на совести игроков;

**По организации связи:**

* «Равный с равным»

В этом режиме связи нет чётко выделенного главного компьютера; от каждого игрока информация передаётся всем остальным компьютерам. Один из компьютеров обычно является ведущим, его роль ограничивается заданием темпа игрыи управлением игрой (смена уровня, изменение настроек игры).

* Звездообразная связь

Архитектура напоминает «равный с равным», однако вся связь ведётся через один центральный компьютер. Эта связь является переходным между «равный с равным» и «клиент-сервер»;

* Клиент-сервер

Один из компьютеров (сервер) содержит полную и достоверную информацию об игровом мире. Остальным компьютерам (клиентам) передаётся лишь та доля информации, которая позволяет вести игру и адекватно отображать игровой мир;

* Многосерверная модель

Многосерверная модель разработана для того, чтобы уменьшить проблему задержек, присущую обычной клиент-серверной модели. В этом режиме каждый компьютер является и клиентом, и сервером. Например, как только игрок Б попадает в кадр игроку А, тот может запросить, чтобы компьютер Б передавал своё положение напрямую на А, в обход сервера;

## 1.3 Игровой цикл

Игровой цикл компьютерной игры включает следующие этапы:

* Инициализация. На этапе инициализации проекта проводится инициализация графической и звуковых подсистем, системы ввода данных;
* Загрузка ресурсов. Здесь загружаются игровые ресурсы, такие, как текстуры и двумерные модели, звуки, шрифты;
* Начало игрового цикла. Основные действия, которые выполняются при работе игры, организованы в виде игрового цикла. Фактически, игровой цикл – это обязательный элемент любой игры;
* Проверка критерия прекращения игры. Игровой цикл продолжается до тех пор, пока игра не будет прекращена. Например, критерием останова может быть истечение времени, выделенного игроку на выполнение игровой задачи, набор определенного количества очков, какое-то событие, произошедшее в процессе игры, принудительная остановка игры, если пользователь решил прекратить играть и так далее;
* Конец игрового цикла. Игровой цикл заканчивается после выполнения условия останова игры;
* Освобождение ресурсов. На последнем этапе работы игры проводится освобождение системных ресурсов, занятых игрой и выход из игры;

## 1.4 Цель и задачи

### 1.4.1 Цель

Цель данной курсовой работы – написание программного средства на платформе .NET версии 2.0 (так же известной как Mono) и языке C#, с использованием технологий Unity3D и серверной части на Apache, с установленными модулями PHP 5.3 и MySQL, предназначенной для реализации сетевого платформера.

Серверная часть должна хранить информацию о пользователях, командах (парах игроков), сохранения команд и список активных сессий.

### 1.4.2 Задачи

Данное приложение должно выполнять следующие задачи:

* Подключение двух игроков друг к другу посредством сервера. Сервер используется для хранения данных о сессиях пользователей;
* Реализовывать игровой процесс, при условии наличия необходимого количества игроков;
* При выходе одного из игроков производить завершение игры.
* Обеспечивать желание играть в игру продолжительное время

# 2 РАЗРАБОТКА АЛГОРИТМА

## 2.1 Алгоритм работы сети

Для реализации сетевой составляющей нашего программного средства, нам необходимо реализовать игровой сервер, который отвечал бы за обмен данными между игроками и контроль подключенных игроков.

Прежде всего, необходимо контролировать наличия сервера по определённому IP-адресу и при его отсутствии выводить соответствующее сообщение, например “Нет сервера с таким IP-адресом”. В противном случае мы производим соединение клиента с нашим сервером, который в свою очередь смотрит, достаточно ли игроков для начала игрового процесса.

При достаточном количестве игроков сервер передаёт игрокам информацию о порядке ходов (в нашем случае, чей ход первый), и является посредником для передачи данных от игрока, чей сейчас ход, к другим игрокам, предоставляя правильное отображение игровой обстановки.

## 2.2 Диграмма экранов (UI/UX)

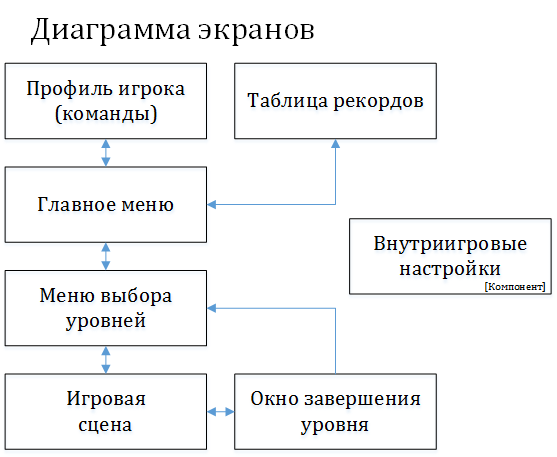


Рисунок 2.1 – Диаграмма экранов

## 2.3 Диаграмма базовых классов для управления игровым процессом

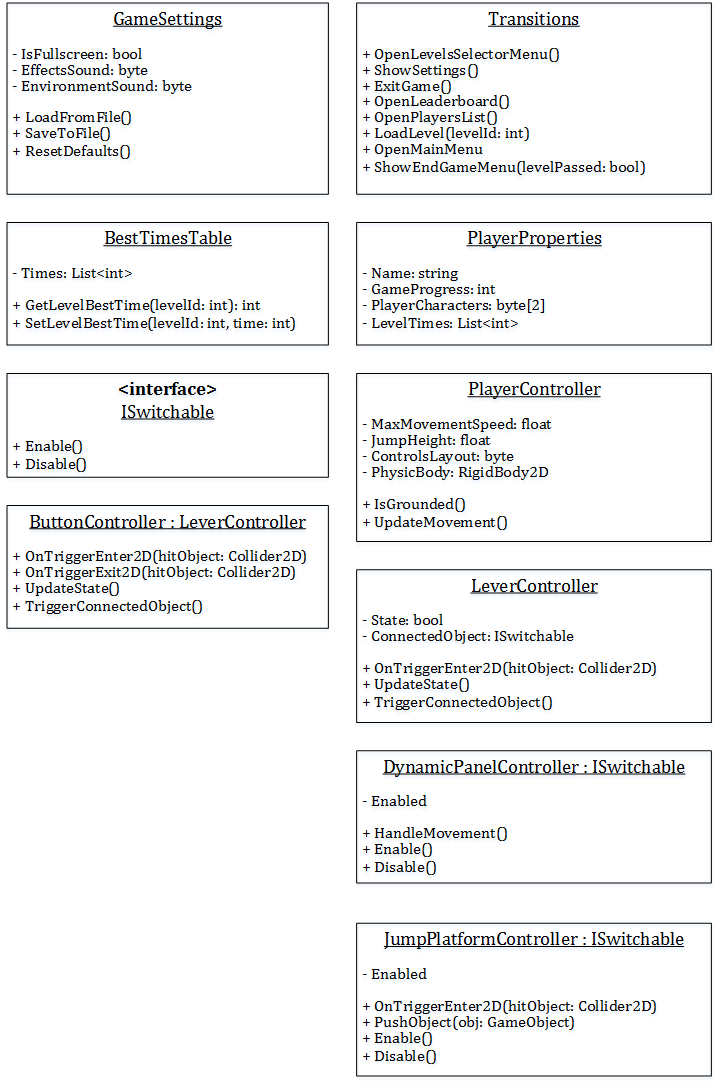


Рисунок 2.2 – Диаграмма классов

## 2.4 Разработка игровых механизмов

В игре присутствует множество механизмов, призванных запутать игрока, заставить его принять неверное решение, и разнообразить игру.

Среди механизмов можно выделить:

* Кнопки – интерактивные элементы, имеющие два состояния, при этом чтобы сохранить состояние включенности, на кнопке должен стоять по крайней мере один игрок
* Рычаги – интерактивные элементы, которые под воздействием физики могут переходить в крайние положения и вызывать срабатывание других механизмов и ловушек
* Двигающиеся панели. В игре присутствуют панели, которые приводятся в движение сами по себе, либо под действием других механизмов. Обычно с помощью панелей игроки могут доставлять себя на верхние уровни друг с помощью друга
* Двери, которые могут блокировать некоторые секции уровня.
* Панели на перемычке, которые приводятся в действие наступанием на них персонажей. Обычно под такими перемычками находятся ядовитые лужи
* Ядовитые лужи – это бассейны, в которых могут утонуть определенные персонажи, таким образом один бассейн опасен для одного игрока, а для второго – безопасен, и игрокам постоянно приходится преодолевать эти бассейны с помощью партнера
* Порталы, которые связаны между собой. Попав в один портал, персонаж с этой же скоростью, однако в другом направлении вылетит из другого портала. Обратное перемещение невозможно
* Платформы для прыжков, которые отражают скорость прыгающего на них персонажа в обратном направлении, при этом увеличивая модуль скорости

# 3 **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

## 3.1 Протокол взаимодействия пользователей

Для полноценной работы программного средства посредством сети был разработан протокол взаимодействия, использующий TCP/IP протокол.

Все команды реализованы в текстовом виде.

Код команды отделяется от данных команды символом « | »

Поля данных отделяются символом « $ »

В таблице 3.1 приведены все команды протокола и краткая информация о них

Таблица 3.1 – Набор команд протокола

|  |  |  |
| --- | --- | --- |
| **Команда** | **Назначение** | **Данные** |
| PartnerConnectRequested | Запрос на подключение партнера | ID игрока |
| AcceptConnectRequest | Подтверждение подключения | Нет |
| PauseRequest | Вызов меню паузы | Нет |
| ExitRequest | Уведомление о выходе | Нет |
| ChatMessage | Сообщение в чате | Текст сообщения |
| ReadyToStartLevel | Уведомление о готовности к началу уровня | ID уровня |
| OpenSwitchLevelScene | Вход в меню выбора уровней | Нет |
| PlayerControllerUpdate | Сетевое управление персонажем | Позиция и скорость игрока, время замера |
| ObjectReplication | Сетевое управление объектами | Специфичные поля объектов |
| StartLevel | Уведомление о начале уровня | Нет |
| LevelFailed | Уведомление о смерти партнера | Нет |

# 4ТЕХНИЧЕСКИЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ

## 4.1 Асинхронный обработчик сообщений

После установки TCP соединения с партнером, игрок должен иметь возможность постоянно получать от него сообщения, которые будут обрабатываться обработчиком TCP протокола. Если принимать сообщения в основном потоке программы, то блокирующие методы приема сообщений будут останавливать программу, и она станет неактивной, что недопустимо.

Для обхода этой особенности было принято решение использовать отдельный поток для асинхронного приема сообщений с помощью интерфейса, предоставляемого модулем System.Net.Sockets платформы .NET.

Асинхронный прием сообщений был помещен в класс-обертку AsyncMessageReceiver, который принимает данные от физического устройства, производит разбиение общих данных на код команды и прикрепленные данные согласно протоколу, отправляет их на обработчик протокола согласно паттерну программирования «Конвейер».

## 4.2 Обновление состояния игровых объектов

Для обновления состояния игровых объектов используется механизм сетевой репликации. Суть метода состоит в том, что у игровых объектов выделяется набор свойств, которые невозможно синхронизировать на основании других свойств. Следовательно, их нужно получить по сети от партнера.

Для этого набора свойств создаются слушатели, которые при изменении состояния свойства формируют сетевое сообщение протокола, и прикрепляют обновленное состояние к данным сообщения, и немедленно отправляют данные партнеру по сети.

Каждый игровой объект при проектировании уровней наделяется сетевым идентификатором, и регистрируется в состоянии игрового мира. При получении сообщения об обновлении игрового объекта протоколом, он переправляет данные на менеджер игрового состояния, который является контекстом выдачи сетевых идентификаторов.

## 4.3 Контроль состояния игры

Для того, чтобы определить, закончена игра или нет, необходимо контролировать несколько вещей. Во-первых, необходимо контролировать состояние соединение сервера с другими игроками. При отключение соперника, происходит автоматическое оповещение второго игрока об этом, путём отправки ему соответствующего пакета. Игрок, получив его, видит соответствующее сообщение, при закрытии которого происходит завершение данного сеанса игры. Все клиенты закрываются, так же, как и сервер.

Если вдруг произойдёт так, что сервер отключится, то произойдёт обработка исключительной ситуации, при которой, как и в первом случае, клиент увидит сообщение, при закрытии которого, произойдёт завершение текущего состояния игры.

Если с соединение никаких проблем не возникает и игровой процесс идёт гладко, рано или поздно настанет момент, когда один из игроков выиграет другого. Это контролируется путём проверки после каждой атаки списка юнитов, принадлежащей второй стороне. Если у второй стороны все юниты имеют значение null (они мертвы), то тогда он получает сообщение о том, что он проиграл игру и у него происходит завершение игры. Первая же сторона получает сообщение о победе и так же завершает игру.

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 5.1 Главное меню

При входе в игру пользователь попадает в главное меню, в котором ему предлагается пройти авторизацию либо регистрацию в сети посредством нажатия кнопок «Log In» и «Register» соответственно (см рис.5.1).

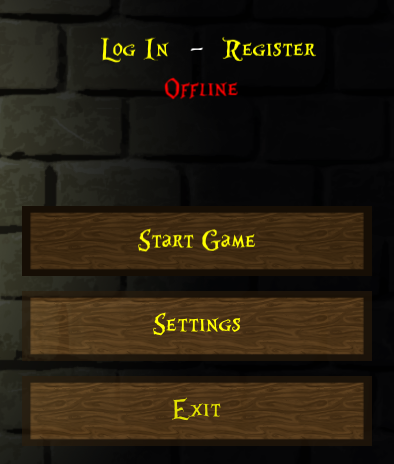


Рисунок 5.1 – Кнопки главного меню игры

## 5.2 Настройки игры

При нажатии на кнопку «Settings» пользователь попадает в меню настроек, где ему будет предложено включить или отключить звук, изменить громкость и вернуться к стандартным настройкам (см. рис. 5.2).

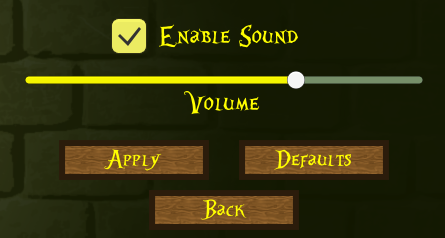


Рисунок 5.2 – Окно настроек игры

Настройки игры сохраняются на компьютере игрока и не записываются в сетевое хранилище, поэтому для загрузки пользовательских настроек не нужно активное интернет соединение.

## 5.3 Окна авторизации и регистрации

С помощью окна авторизации вы можете авторизироваться в системе с помощью своих учетных данных – логина и пароля.

Если ваши данные не верны, то поля ввода сбросятся и вы сможете попробовать снова.



Рисунок 5.3 – Окно авторизации пользователя

Если же вы в первый раз вошли в игру, вы можете зарегистрироваться. Если вы укажете логин или пароль, не соответствующий правилам, вы увидите окно с текстом соответствующей ошибки.

Правила корректности логина и пароля:

* Длина не менее 4 символов и не более 32 символов
* Допустимы символы латинского алфавита и цифры

Так же вы можете выбрать визуальную цветовую схему вашего персонажа. Доступны следующие цветовые схемы: зеленая, синяя, красная и фиолетовая.

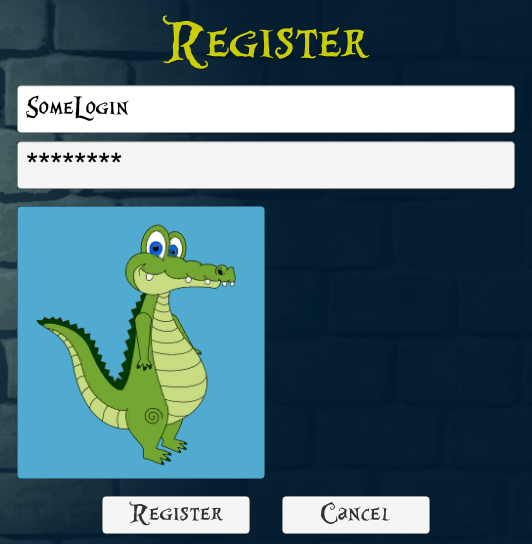


Рисунок 5.4 – Окно регистрации пользователя

## 5.4 Окно выбора партнера

Как только вы авторизировались в системе, вы можете перейти к этапу выбора партнера (Matchmaking).

С сервера будут загружены данные о всех игроках и их текущем сетевом статусе (Online / Offline).

С помощью кнопки «Refresh» можно обновить эту информацию.

При нажатии на кнопку, соответствующую персонажу, вы отошлете ему запрос на совместную игру. Когда игрок пришлет подтверждение, вы перейдете в меню выбора уровней.



Рисунок 5.5 – Окно выбора партнера

## 5.5 Меню выбора уровней

В меню выбора уровней вы увидите несколько панелей, соответствующих игровым уровням. На каждой панели вы увидите общую геометрию уровня со всеми сущностями и механизмами, название уровня, лучшее время и кол-во звезд за уровень.

Если уровень еще не открыт, то он будет блокирован иконкой замка.



Рисунок 5.6 – Фрагмент меню выбора уровней

## 5.6 Игровой процесс

Когда загрузится уровень, игрокам будет предложено вместе решать всевозможные головоломки разной степени сложности.

На уровнях будет представлено множество ловушек, как общего назначения, которые убивают персонажа, так и частных, которые убивают только конкретного персонажа.

Бывают также пассивные ловушки в виде неразрешимых игровых ситуаций. Если игроки неправильно оценят решение головоломки и окажутся в положении, когда прохождение уровня невозможно, необходимо вручную перезапускать уровень и проходить его заново.

Внутри уровней сохранений не предусмотрено, поэтому уровень защитывается как пройденный только в том случае, если он пройден от начала до конца без смерти любого из игроков.

На уровнях встречаются различные механизмы, которые позволяют игрокам помогать друг другу избегать ловушек, проникать в недоступные поодиночке места. Игра заставляет игроков работать сообща, ждать друг друга, и оптимально рассчитывать время простоя в определенных местах и тщательно выбирать маршрут достижения выхода с уровня.

Игрокам предстоит уворачиваться от падающих с потолка сосулек, обходить стороной лужи чужого цвета, активно управлять физикой в игре, использовать рычаги, кнопки, платформы для прыжков, порталы, избегать застревания партнера беспомощным в изолированной зоне и других ситуациях.



Рисунок 5.7 – Фрагмент уровня

Общей целью каждого уровня является сбор всех драгоценных камней на уровне и передвижение к заветной дверце, которая ведет на выход из уровня.

Пока игроки не собрали все драгоценные камни, дверь блокируется враждебной фиолетовой тучкой. Как только игроки собирают все камни, тучка растворяется и открывает проход, после чего оба игрока должны подойти к двери, тогда уровень будет считаться пройденным.

Для каждого уровня определено свое время для получения 3, 2 и 1 звезд за него. За прохождение уровня игрок будет награжден как минимум 1 звездой вне зависимости от времени.



Рисунок 5.8 – Тучка на двери выхода

# ЗАКЛЮЧЕНИЕ

В результате разработки было спроектирована и выполнена полноценная компьютерная многопользовательская игра, отвечающая всем первоначально поставленным требованиям.

Цели и задачи были достигнуты практически полностью.

Проектирование и создание архитектуры игры потребовало глубокого понимания предмета и исследования других Open Source технологий, в частности Unreal Engine 4, и попытки адаптации сетевой модели, использованной в этой технологии.

В процессе проектирования я столкнулся с серьезными проблемами организации сетевой связи типа «равный с равным» в игре в режиме реального времени, и на личном опыте ознакомился со всеми достоинствами и недостатками данной модели.

Была проведена тщательная работа по проектированию увлекательного игрового дизайна и дизайна уровней, созданию уникальной концепции, красивой анимационной графики, которая была создана вручную с помощью Corel Draw X7 и Adobe Photoshop CS5.

В результате проделанной работы я готов представить игру с обучающими уровнями, и одним уровнем, объединяющим все механизмы, созданные в игре. На основании отзывов людей, принявших участие в тестировании игры были точно определены точки максимального удовольствия в игровом процессе, и были приняты все отзывы, как отрицательные, так и положительные. Это дает задатки для создания новых улучшенных версий игры, которые будут намного более увлекательными.

# СПИСОК ЛИТЕРАТУРЫ

1. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# [Текст] / Д. Рихтер.
2. Видеоматериалы по технологии Unity3D 5 [Электронный ресурс] – Электронные данные. – Режим доступа: https://unity3d.com/learn/tutorials.

# ПРИЛОЖЕНИЕ А.

## КОД ПРОГРАММЫ

Класс управления веб-сервисами PHP

WebServices.cs

using System.Net;

using System.Collections.Generic;

using System.Text;

using System.IO;

using System;

using UnityEngine;

public sealed class WebServices

{

const string ServicesPath = "http://nexonlab.hol.es/game\_server/";

const int DefaultTimeout = 5000;

public static HttpWebRequest GeneratePostRequest(string ServiceName, Dictionary<string, string> Params)

{

HttpWebRequest Request = WebRequest.Create(string.Format("{0}{1}.php", ServicesPath, ServiceName)) as HttpWebRequest;

Request.Method = "POST";

// Convert dictionary values to post format string

int ParamsWritten = 0;

string Data = string.Empty;

foreach (var Pair in Params)

{

Data += string.Format("{0}{1}={2}", ((ParamsWritten > 0) ? "&" : ""), Pair.Key, Pair.Value);

ParamsWritten++;

}

byte[] DataBytes = Encoding.UTF8.GetBytes(Data);

Request.ContentType = "application/x-www-form-urlencoded";

Request.ContentLength = DataBytes.Length;

Request.ReadWriteTimeout = DefaultTimeout;

// Write request data

Stream RequestStream = Request.GetRequestStream();

RequestStream.Write(DataBytes, 0, DataBytes.Length);

RequestStream.Close();

return Request;

}

public static string GetResponseString(HttpWebRequest Request)

{

try

{

HttpWebResponse Response = Request.GetResponse() as HttpWebResponse;

return new StreamReader(Response.GetResponseStream()).ReadToEnd().Trim();

}

catch (Exception)

{

Debug.LogError("Failed to send web service request!");

return null;

}

}

public static byte[] GetResponseData(HttpWebRequest Request)

{

try

{

HttpWebResponse Response = Request.GetResponse() as HttpWebResponse;

return new BinaryReader(Response.GetResponseStream()).ReadBytes ((int)Response.ContentLength);

}

catch (Exception)

{

Debug.LogError("Failed to send web service request!");

return null;

}

}

}

Класс асинхронной обработки сообщений партнера

AsyncMessageProcessor.cs

using System;

using System.Net.Sockets;

using System.Text;

using UnityEngine;

using Nexon;

public class AsyncMessageProcessor

{

private const int BufferSize = 512;

private TcpClient ClientSocket;

private NetworkStream DataStream;

private byte[] buffer;

public AsyncMessageProcessor(TcpClient Client)

{

ClientSocket = Client;

}

public void StartMessageReceiveLoop()

{

if (ClientSocket != null)

{

DataStream = ClientSocket.GetStream();

buffer = new byte[BufferSize];

DataStream.BeginRead(buffer, 0, buffer.Length, MessageReceiveLoop, null);

}

}

private void MessageReceiveLoop(IAsyncResult Result)

{

int BytesRead = DataStream.EndRead(Result);

if (BytesRead == 0)

{

ClientSocket.Close();

Debug.LogError("Partner disconnected");

}

else

{

TCPMessage ReceivedMessage = new TCPMessage(Encoding.UTF8.GetString(buffer));

TCPProtocol.Instance.ProcessCommand(ReceivedMessage.Command, ReceivedMessage.AttachedData);

DataStream.BeginRead(buffer, 0, buffer.Length, MessageReceiveLoop, null);

}

}

}

Обработчик сообщений протокола

using Nexon;

using System.Collections.Generic;

using UnityEngine;

public sealed class TCPProtocol

{

private delegate void Processor(string Data);

private Dictionary<TCPCommand, Processor> Processors;

private static int MessageIndex;

private TCPProtocol()

{

RegisterCommandProcessors();

}

public void ProcessCommand(TCPCommand Command, string Data)

{

if (Processors.ContainsKey(Command))

{

Processors[Command](Data);

}

}

private void RegisterCommandProcessors()

{

Processors = new Dictionary<TCPCommand, Processor>();

Processors.Add(TCPCommand.PartnerConnectRequested, ProcessPartnerConnectRequest);

Processors.Add(TCPCommand.OpenSwitchLevelScene, ProcessOpenSwitchLevelScene);

Processors.Add(TCPCommand.AcceptConnectRequest, ProcessAcceptConnectRequest);

Processors.Add(TCPCommand.StartLevel, ProcessStartLevel);

Processors.Add(TCPCommand.PlayerControllerUpdate, ProcessPlayerControllerUpdate);

Processors.Add(TCPCommand.LevelFailed, ProcessLevelFailed);

Processors.Add(TCPCommand.ObjectReplication, ProcessObjectReplication);

}

private void ProcessPartnerConnectRequest(string Data)

{

Debug.LogError("<- PartnerConnectRequest");

// Resolve attached player id

int PartnerId;

if (int.TryParse(Data, out PartnerId))

{

OnlineGameState GameState = OnlineGameState.Instance;

GameState.ConnectPartner(PartnerId, true);

GameState.Player.OrderInTeam = 1;

// When partner registered, send confirmation message

TCPMessage ConfirmationMessage = new TCPMessage(TCPCommand.AcceptConnectRequest);

GameState.SendTCPMessageToPartner(ConfirmationMessage);

Transition.Instance.SwitchToLevelSelectionScene();

}

}

private void ProcessOpenSwitchLevelScene(string Data)

{

Debug.LogError("<- OpenSwitchLevelScene");

Transition.Instance.SwitchToLevelSelectionScene();

}

private void ProcessAcceptConnectRequest(string Data)

{

Debug.LogError("<- AcceptConnectRequest");

// When we know, that our partner confirmed connection request,

// We can switch to level selection scene

Transition.Instance.SwitchToLevelSelectionScene();

}

private void ProcessStartLevel(string Data)

{

Debug.LogError("<- StartLevel");

Transition.Instance.GoToLevel(Data);

}

private void ProcessPlayerControllerUpdate(string Data)

{

Debug.LogError("<- PlayerControllerUpdate" + MessageIndex.ToString());

MessageIndex++;

// Parse received update package

string[] ParsedUpdate = Data.Split('$');

Vector3 position = new Vector3(float.Parse(ParsedUpdate[0]), float.Parse(ParsedUpdate[1]), 0.0f);

Vector2 velocity = new Vector2(float.Parse(ParsedUpdate[2]), float.Parse(ParsedUpdate[3]));

OnlineGameState GameState = OnlineGameState.Instance;

GameState.UpdatePartnerPlayerState(position, velocity);

}

private void ProcessLevelFailed(string Data)

{

Debug.LogError("<- LevelFailed");

GameNetworkContext.Instance.NetworkLevelFailed = true;

}

private void ProcessObjectReplication(string Data)

{

Debug.LogError("<- ObjectReplication");

string[] ReplicationValues = Data.Split('$');

int NetworkId = int.Parse(ReplicationValues[0]);

GameNetworkContext.Instance.ProcessReplicationMessage(NetworkId, ReplicationValues[1], ReplicationValues[2]);

}

}

Класс управления игровым уровнем

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

using Nexon;

public class LevelController : MonoBehaviour

{

enum LevelState

{

InProgress,

Paused,

Failed,

Completed

}

public GameObject gameFailedPanel;

public GameObject gameCompletedPanel;

public int[] LevelGemsToGather;

public GameObject FinalCloud;

public Text TimeText;

public AudioSource LevelThemeSound;

public AudioSource FailSound;

public int [] LevelStarsMapping;

public GameObject FinalStarsImage;

public GameObject [] TeamSpawnPoints;

public GameObject[] TeamGemsSpawnPoints;

public int[] TeamGemsOwner;

private LevelState levelState;

public static PlayerController[] PlayerControllers = new PlayerController [2];

private int [] LevelGems;

private float LevelTime;

private void Start()

{

LevelManager.CreateInstance();

controllerSample = this;

levelState = LevelState.InProgress;

gameFailedPanel.SetActive(false);

gameCompletedPanel.SetActive(false);

LevelGems = new int [2];

SpawnPlayers();

SpawnGems();

}

private void SpawnPlayers()

{

int [] PlayerPrefabsId = new int[2];

OnlineGameState GameState = OnlineGameState.Instance;

if (GameState.Player != null && GameState.Partner != null)

{

int PlayerOrder = GameState.Player.OrderInTeam;

int PartnerOrder = (PlayerOrder == 0) ? 1 : 0;

PlayerPrefabsId[PlayerOrder] = GameState.Player.CrocoIndex;

PlayerPrefabsId[PartnerOrder] = GameState.Partner.CrocoIndex;

}

else

{

PlayerPrefabsId[0] = 0;

PlayerPrefabsId[1] = 1;

}

for (int i = 0; i < 2; i++)

{

GameObject PlayerObject = Instantiate(PlayerVisualInfoStorage.GetPlayerPrefab(PlayerPrefabsId[i]));

PlayerObject.transform.position = TeamSpawnPoints[i].transform.position;

PlayerController controller = PlayerObject.GetComponent<PlayerController>() as PlayerController;

controller.Team = PlayerPrefabsId[i];

if (GameState.DataLoaded)

{

controller.DisableMovement = (i != GameState.Player.OrderInTeam);

controller.Owner = (i == GameState.Player.OrderInTeam) ? GameState.Player : GameState.Partner;

if (i != GameState.Player.OrderInTeam)

{

controller.SetNetworkController();

}

}

PlayerControllers[i] = controller;

}

}

private void SpawnGems()

{

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

{

int GemTeam = TeamGemsOwner[i];

GameObject Gem = Instantiate(PlayerVisualInfoStorage.GetGemPrefab(GemTeam));

PickupBehaviour gemController = Gem.GetComponent<PickupBehaviour>();

gemController.Team = GemTeam;

Gem.transform.position = TeamGemsSpawnPoints[i].transform.position;

}

}

public void RestartLevel()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().name);

}

public void SetLevelFailed(bool FromReplication = false)

{

levelState = LevelState.Failed;

gameFailedPanel.SetActive(true);

DeactivatePlayerContollers();

if (GameSettings.ActiveSettings.EnableSound)

{

FailSound.volume = GameSettings.ActiveSettings.SoundVolume;

FailSound.mute = false;

FailSound.Play();

}

if (!FromReplication)

{

TCPMessage FailMessage = new TCPMessage(TCPCommand.LevelFailed);

OnlineGameState.Instance.SendTCPMessageToPartner(FailMessage);

}

}

public void SetLevelCompleted()

{

levelState = LevelState.Completed;

gameCompletedPanel.SetActive(true);

DeactivatePlayerContollers();

int StarsCount = GetStarsCount();

FinalStarsImage.GetComponent<Image>().sprite = StarImagesStorage.GetStarImage(StarsCount);

// Update level stats

OnlineGameState GameState = OnlineGameState.Instance;

LevelStats LevelStats = GameState.Team.Save.LevelsInfo[Transition.CurrentLevelId];

if (!LevelStats.Passed)

{

LevelStats.Minutes = (int)LevelTime / 60;

LevelStats.Seconds = (int)LevelTime % 60;

LevelStats.Passed = true;

}

else if ((int)LevelTime < LevelStats.Minutes \* 60 + LevelStats.Seconds)

{

LevelStats.Minutes = (int)LevelTime / 60;

LevelStats.Seconds = (int)LevelTime % 60;

}

LevelStats.Stars = StarsCount;

GameState.Team.Save.LevelsInfo[Transition.CurrentLevelId] = LevelStats;

// Unlock next level

if (LevelManager.Instance.LevelsCount > Transition.CurrentLevelId + 1)

{

GameState.Team.Save.LevelsInfo[Transition.CurrentLevelId + 1].Available = true;

}

GameState.Team.Save.SaveToServer(GameState.Team.Id);

}

public void AddGem(int team)

{

LevelGems[team]++;

int GemsFoundFactor = 0;

const int TeamsCount = 2;

for (int i = 0; i < TeamsCount; i++)

{

if (LevelGems[i] >= LevelGemsToGather[i])

{

GemsFoundFactor++;

}

}

// If all teams gathered their gems - remove cloud from exit door

if (GemsFoundFactor == TeamsCount)

{

DestroyFinalCloud();

}

}

public PlayerController GetPlayerController(int team)

{

return PlayerControllers[team];

}

public void DestroyFinalCloud()

{

if (FinalCloud != null)

{

CloudController controller = FinalCloud.GetComponent<CloudController>();

if (controller != null)

{

controller.FadeOut();

}

else

{

Destroy(FinalCloud);

}

}

}

public void DeactivatePlayerContollers()

{

foreach (var controller in PlayerControllers)

{

controller.DisableMovement = true;

}

}

public int GetStarsCount()

{

if (levelState == LevelState.Completed)

{

int FinalTime = (int)LevelTime;

if (FinalTime <= LevelStarsMapping[0])

{

return 3;

}

else if (FinalTime <= LevelStarsMapping[1])

{

return 2;

}

return 1;

}

return 0;

}

private void Update()

{

if (Input.GetKeyDown(KeyCode.E))

{

for (int i = 0; i < 10; i++)

{

AddGem(0);

AddGem(1);

}

}

if (Input.GetKeyDown(KeyCode.R))

{

LevelTime += 10.0f;

}

if (Input.GetKeyDown(KeyCode.T))

{

SetLevelCompleted();

}

if (levelState == LevelState.InProgress)

{

LevelTime += Time.deltaTime;

if (TimeText != null)

{

int seconds = (int) (LevelTime);

TimeText.text = string.Format("{0}:{1:00}", seconds / 60,

seconds % 60);

}

}

}

}