# ВВЕДЕНИЕ

Изучение компьютерных игр в современных социальных и гуманитарных науках по-прежнему остаётся относительно новым направлением. Игры являются настоящим культурным явлением, став важной составляющей жизни современного человека, которая приобрела исключительную популярность в последние десятилетия.

Среди самых популярных игровых приложений всегда выделяются сетевые игры. Сетевые приложения требуют использования более одного устройства, связь между которыми может быть обеспечена различными способами соединения. Эти игровые приложения позволяют пользователям играть друг с другом с помощью интернет-соединения. Популярность игр этого типа и повсеместное распространение Интернета свидетельствуют о необходимости разработки таких сетевых приложений.

Таким образом, онлайн-игры оказали настолько значительное влияние на общество, что в информационных технологиях наметилась устойчивая тенденция к геймификации неигрового прикладного программного обеспечения. Компании тратят много ресурсов на разработку крупных игровых проектов и их продвижение.

1. ИГРОВЫЕ ПРИЛОЖЕНИЯ И СРЕДСТВА ИХ РАЗРАБОТКИ

## История развития многопользовательских игр

Первые многопользовательские игры появились еще в 40-х годах прошлого века. На тот момент, чтобы поиграть с друзьями на разных компьютерах, необходимо было соединить их вместе с помощью сетевого кабеля. Интернет официально появился только в 1983 году, но еще до этого программисты со всего мира предпринимали попытки объединить пользователей ПК одной сетью.

Начиналась история мультиплеера с максимально простых текстовых проектов, которые давали возможность нескольким пользователям подключиться к чату и писать определенные команды. При этом по сети могло играть ограниченное количество людей, и в какой-то момент те самые игры 40-х годов так и остались забавой для «гаражных» кодеров, которые делали их для себя и своих товарищей. До начала 90-х годов многопользовательские игры больше никак не развивались.

Как ни странно, но именно *Bioware* создала первую в мире игру с полноценным многопользовательским режимом (раньше студия вообще делала много чего хорошего). Ею стала *Neverwinter* *Nights*, которая вышла в 1991 году. Проект объединил в себе вид от третьего лица, ключевые элементы ролевых игр тех времен, а также полноценный хостинг, который и обеспечил геймерам наличие сетевого режима.

В проекте было несколько серверов, на каждом из которых могло играть по 96 игроков и 1 мастер подземелья. *Neverwinter* *Nights* полностью строилась на правилах *Dungeons* & *Dragons*, поэтому сразу же стала понятной для огромного количества людей по всему миру. Даже поклонники «настолок» оценили старания *Bioware* и большинство из них от привычных посиделок в закрытых клубах с друзьями перешли в игру, в которую можно было играть, не выходя из дома. На тот момент единственным минусом проекта было отсутствие глобального чата. Группы игроков могли лишь переписываться в своих отдельных чатах, которые они создавали перед входом в игровую сессию. Тем не менее, полноценный мультиплеер в *Neverwinter* *Nights* все-таки был, и это стало отправной точкой для сетевого режима в видеоиграх.

Спустя 3 года после первых попыток сделать сетевой режим в видеоигре, выходит первый *Doom*. Сразу становится ясно, что для совместной игры по локальной сети создатели позаимствовали идеи *Bioware*. Тем не менее, их проект был совершенно в другом жанре, так что разработчикам хотелось сделать доступ к мультиплееру более удобным и понятным.

В 1996 году выходит масштабное обновление для *Doom*, которое привносит в него полноценный многопользовательский режим. Причем сделан он был в привычном для современных игроков виде. Разработчики использовали сервис под названием *DWANGO*, который разработали программисты из Хьюстона. Основной задачей *DWANGO* был подбор игроков. То есть несколько игроков заходили в мультиплеер, запускали подбор соперников и просто ждали, пока сторонний сервис свяжет их друг с другом. Даже на то время система работала достаточно быстро и через несколько лет перекочевала в *Doom* *II*, а затем в *Duke* *Nukem* 3*D* и другие проекты.

После успешной реализации многопользовательского режима в *Doom*, многие игроделы начали задумываться о том, чтобы сделать проект, который будет полностью сетевым. Пока одни думали, другие взяли и выпустили *Meridian* 59. Эта первая ролевая игра, которая была полностью сетевой и при этом обладала продвинутой графикой, естественно, на то время. До выхода *Meridian* 59 многопользовательские *RPG* были в основном текстовыми, и только парочка проектов содержала в себе простенькие спрайты и какие-то зарисовки.

В свою очередь, *Meridian* 59, которую выпустили в 1996 году, предлагала вид от первого лица, в ней были полностью отрисованные локации, пользовательский интерфейс, инвентарь главного героя, карта и многое другое, что уже давно стало классикой жанра. Многие популярные сегодня *MMORPG* позаимствовали у *Meridian* 59 кучу механик. Например, именно здесь впервые появилась внутриигровая почтовая система, залы гильдий, голосование внутри группы игроков, кастомизация предметов и выделенные каналы связи с конкретными игроками.

*Meridian* 59 стала легендарной игрой, но в середине 90-х геймеры уже успели устать от бесконечного потока ролевых игр. Очень вовремя, а именно под конец 1996 года, выходит дополнение под названием *Gold* для стратегии *Command* & *Conquer*. Это обновление привносит в игру полноценный многопользовательский режим, который постепенно начинают заимствовать другие разработчики стратегий. Например, уже в 1997 году *Microsoft* подсмотрела, как реализован сетевой режим у конкурентов, и быстро прикрутила его к своей *Age* *of* *Empires*.

Следом за *Microsoft* волну многопользовательских режимов в одиночных играх подхватила и *Activision*. Все в том же 1997 году издается *NetStorm*: *Islands* *At* *War*, игру, которую разрабатывала студия *Titanic* *Entertainment*. Сетевой режим был рассчитан одновременно на 8 игроков. При этом проект привлекал внимание тем, что не был похож на классические игры *RTS* того времени. Здесь игрок управлял всего несколькими движущимися юнитами, которые защищали его личные острова под предводительством жрецов. Для победы необходимо было захватить всех жрецов оппонента. Многие любители достойных старых стратегий даже сегодня считают, что *NetStorm*: *Islands* *At* *War* опередил свое время, и не только за счет мультиплеера.

До конца 1997 года многие студии экспериментировали с добавлением сетевого режима в свои проекты, но удачно получалось далеко не у всех. Тем не менее, попытки продолжались и в итоге привели к тому, что на свет появилась первая полноценная *MMORPG* под названием *Ultima* *Online*. На самом деле, серия *Ultimate* выходила еще с 1981 года, но разработчики никак не решались полностью перевести ее в интернет, хотя игра как будто была создана для этого.

Успех проекта был просто ошеломительным. Многие поклонники *RPG* играют в *Ultima* *Online* даже сегодня, потому что создатели продолжают ее поддерживать и даже выпускают масштабные обновления. К слову, это первая многопользовательская игра, для которой делали крупные аддоны, как для *World* *of* *Warcraft*. Также именно здесь появилась привычная для нас система *PvP*, и основное внимание уделяли именно ей.

Благодаря всем инновациям, которые *Ultima* *Online* привнесла в многопользовательские игры, в 1998 году появилась первая часть *Lineage*. Создатели особо не рассчитывали на успех, но в итоге, в первые дни собрали более 3 миллионов подписчиков. Изюминкой «Линейки» стала система формирования групп, в которые объединялись игроки и затем вступали в масштабные войны. Вместе с этим группы могли обзавестись собственным замком, а потом периодически защищать его в кровавых осадах. Масштабы были просто запредельными, и именно *Lineage* подарила игровой индустрии несколько ключевых механик, которые можно наблюдать в современных *MMORPG*.

Вплоть до 1998 года игры с сетевыми режимами делало лишь небольшое количество известных, на то время, разработчиков. Крупные игроки на рынке уже поняли, что за мультиплеером будущее и просто начали развивать свои сервисы для совместной игры в интернете. Студии поменьше пытались придумать собственные системы, чтобы прикрутить к играм сетевой режим, но получалось это далеко не у всех. В те времена создавать что-то новое было довольно сложно, и никто из игроделов не собирался делиться друг с другом своими наработками.

По сути, последнюю инновацию по части мультилпеера в индустрию привнесла *Blizzard*. На то время студия была крепким середняком, но после выхода *StarCraft* и запуска клиента *Battle*.*net*, разработчики мгновенно залетели на место «короля» игровой индустрии.

Инновация *Battle*.*net* заключалась в том, что теперь у простого геймера была удобная программа, которая предоставляла доступ к сетевому режиму. При этом через нее он мог общаться с друзьями, искать себе напарников для игровых сессий, и все это было максимально удобно. А самое главное, что для доступа к мультиплееру достаточно было зарегистрироваться в *Battle*.*net*, и больше никаких «танцев с бубном», как в других играх, где сетевой режим был в качестве дополнения. Примеру *Blizzard* последовали многие студии и постепенно начали выпускать свои собственные сервисы, чтобы геймеры могли играть друг с другом без лишних проблем.

## Классификации многопользовательских игр

Классификация многопользовательских игр по технической реализации.

Многопользовательская игра на одном компьютере:

* игроки участвуют в игре одновременно, в спортивных симуляторах это естественный режим, в других играх, например, гонках, экран делится пополам, и такой режим называется *splitscreen*;
* игроки участвуют в игре поочерёдно (англ. *hotseat*), используется в некоторых пошаговых стратегиях, например, *Heroes* *of* *Might* *and* *Magic*, в симуляторах Формулы-1 *Grand* *Prix* 2, 3 и 4 также есть режим поочерёдной игры, когда один игрок ведёт свой болид, а болидами остальных управляет компьютер, при этом несколько раз за гонку управление передаётся от одного игрока другому;
* один компьютер – несколько терминалов, на данный момент применяется в основном в игровых автоматах.

Игра по сети. Несколько компьютеров соединены в вычислительную сеть:

* через последовательные или параллельные порты;
* через модем;
* через локальную сеть и интернет, по протоколам *IPX* или *TCP*/*IP*. (3*D*-шутеры, стратегии в реальном времени);
* через собственный клиент: *Ultima* *Online*, Аллоды Онлайн;
* браузерные игры: *Tale*, Бойцовский клуб, *Magic*, *Livegames*;
* через электронную почту. См. *PBEM*;
* через специальный веб-сайт: Мафия;
* через *IRC* (например, викторины);
* через *Bluetooth*, *IrDA*, *SMS* (игры для мобильных телефонов).

Некоторые игры (в основном для игровых приставок) поддерживают сетевую игру, в которой на каждой машине могут играть несколько игроков в режиме *splitscreen*.

Классификация многопользовательских игр по правилам:

* по одной «жизни». Этот режим применялся в некоторых играх для первых ПК (например, *ZX* *Spectrum*). В этом режиме игроки совсем не взаимодействуют друг с другом, каждый ведёт свою игру; когда игрок теряет «жизнь», управление передаётся напарнику, например, *Digger*;
* 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, когда объединение в команды и война друг с другом полностью лежат на совести игроков. Разумеется, предательства не возбраняются, например, *MMORPG*.

Классификация многопользовательских игр по оргранизации связи.

«Равный с равным» (англ. *peer*-*to*-*peer*). В этом режиме связи нет чётко выделенного главного компьютера; от каждого игрока информация передаётся всем остальным компьютерам. Каждый из компьютеров имеет достоверную информацию об игровом мире. Один из компьютеров обычно является ведущим, его роль ограничивается заданием темпа игры и управлением игрой (смена уровня, изменение настроек игры). Если ведущий выходит, роль ведущего может взять на себя любой другой компьютер. В этом режиме обычно работают стратегии в реальном времени.

Достоинства: простота реализации; при архитектуре, когда передаются команды управления, игровой мир может быть сколь угодно сложным, но трафик зависит лишь от количества игроков; минимальная нагрузка на ведущего (равняющаяся нагрузке на ведомых); минимальные задержки передачи; при выходе ведущего любой компьютер может взять на себя обязанности ведущего.

Недостатки: взломав игру, можно следить за остальными игроками; большой трафик при большом количестве игроков; практически невозможно реализовать противодействие некачественной связи; проблемы с входом в начатую игру; требуется установленный канал связи каждого с каждым; высокая нагрузка на ведомые машины.

Применение: первые сетевые игры (например, *Doom*), игры для небольшого количества игроков, игры на слабых машинах (телефонах/карманных приставках), стратегии в реальном времени.

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

Достоинства: простота реализации; малая нагрузка на сеть; можно реализовать противодействие задержкам, передачу ведущего на другой компьютер и вход в начатую игру; легко реализуется Кооперативная игра.

Недостатки: взломав игру, всё ещё можно следить за остальными игроками; всё ещё высокая вычислительная нагрузка на ведомые машины; более высокие задержки передачи.

Применение: аналогично одноранговым сетям.

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.

Достоинства: минимальная нагрузка на клиенты; можно установить выделенный сервер (в остальных моделях это бесполезно); наибольшая стойкость к читерству; естественный выбор, когда игровой мир очень велик, но каждый отдельный пользователь имеет дело с небольшой его частью; часто при изменениях в серверной части пользователям не нужно обновлять клиенты.

Недостатки: сложная реализация; большая нагрузка на сеть, когда игровой мир насыщен активными объектами (поэтому кооперативная игра плохо реализуется); миграция сервера на другой компьютер практически невозможна; высокая задержка между нажатием клавиши и действием.

Применение: большинство современных игр (за исключением стратегий, на которых выгодно применять одноранговую или звездообразную связь).

Многосерверная модель. Современные локальные сети мало похожи на сети времён *Doom* – в них намного больше коллизий, а значит, выше пинг. Многосерверная модель разработана для того, чтобы уменьшить проблему задержек, присущую обычной клиент-серверной модели. В этом режиме каждый компьютер является и клиентом, и сервером [1, с. 69]. 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.

Достоинства: минимальные задержки на современных ЛВС.

Недостатки: более сложная реализация; недостаточное противодействие читерству; при онлайн-игре устанавливается обычный клиент-серверный режим.

Применение: эта модель рассчитана на игры, в которых сотые доли секунды решают всё. Применяется редко; предположительно используется в *Splinter* *Cell* и *Soldier* *of* *Fortune*.

Клиент-серверное взаимодействие в многопользовательских сетевых приложениях

Передача данных по сети с одного устройства на другое проводится по ряду определенных действий, определяемых сетевыми протоколами. Для облегчения разработки протоколов были созданы сетевые модели – схемы, определяющие общие принципы работы набора сетевых протоколов и способы их взаимодействия друг с другом для осуществления передачи данных по сети. Модель разделяется на уровни, в которых протоколы используют протоколы нижестоящего уровня.

Главной задачей при разработке многопользовательских игр является синхронизация состояний всех пользователей на сервере. Для обеспечения синхронизации клиентам необходимо отправлять серверу обновления с фиксированным интервалом времени. Обновления содержат введенные пользователям данные. После получения данных от всех пользователей, следующим тактом сервер обновляет состояния всех клиентов. На рисунке 1.1 представлено взаимодействие сервера с несколькими клиентами.



Рисунок 1.1 – Взаимодействие сервера с несколькими клиентами

Отправка данных с фиксированным интервалом времени в лучшем случае имеет высокую скорость обработки данных и возможность обслуживания большего количества пользователей. Однако, если имеется игрок с плохим интернет-соединением, то замедляется обновления у всех пользователей. Решением данной проблемы является использование концепции прогнозирования. Прогнозирование происходит на стороне клиента, когда вместо ожидания ответа от сервера программа прогнозирует исход действий, отправленных серверу [2]. И когда клиент получит данные от сервера, он подменяет их на устройстве. Таким образом, данная концепция минимизирует количество явных зависаний у всех клиентов при низкопроизводительном интернет-соединении.

Применение протокола *HTTP* в сетевых коммуникациях

*Hypertext* *Transfer* *Protocol* (*HTTP*) – широко распространенный протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов, содержащих ссылки и позволяющих организовать переход к другим документам по этим ссылкам. *HTTP*, в соответствии со спецификацией сетевой модели *OSI*, является протоколом прикладного уровня для передачи гипертекста. Актуальной на данный момент версией протокола является *HTTP* 2.1, и описывается она со спецификации *RFC* 2616 [3, с. 21].

Протокол *HTTP* предполагает клиент-серверную структуру взаимодействия устройств. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает запрос, формирует ответ и передает данные обратно клиенту. Затем клиентское приложение может отправлять другие запросы, которые будут обработаны сервером аналогичным образом. Традиционной задачей, решаемой с помощью протокола *HTTP*, является обмен данных между приложением пользователя, осуществляющим доступ к интернет-ресурсам, и сервером с этими ресурсами.

Соединения по протоколу *HTTP* инициализируются агентом пользователя и состоят из запроса, который применяется к ресурсу на удаленном сервере. В простом случае, запрос выполняется посредством единоразового соединения между агентом пользователя и сервером, и впоследствии могут использоваться другие типы запросов для разных операций.

Сетевое взаимодействие по протоколу *HTTP* происходит с помощью запросов. Каждый запрос имеет следующую структуру:

* строка запроса, в которой указывается версия протокола и метод запроса;
* заголовки, разделенные между собой символом конца строки, в которых передаются параметры для успешного соединения;
* пустую строку для отделения служебной информации от тела сообщения;
* необязательно тело сообщения.

Для *HTTP* операции над данными задаются с помощью методов запроса [4, с. 35]. Методы *GET* и *HEAD* являются методами запроса информации, которая располагается на сервере и доступна по унифицированному указателю ресурса. Метод *GET* является наиболее распространенным методом получения документов для визуализации страницы интернет-браузера. Также результат метода может быть файлом с сервера, результатом выполнения программы или операции, выходная информация аппаратного устройства и др. Отличием метода *HEAD* от метода *GET* является пустое тело ответа от сервера. Метод *HEAD* используется для получения информации заголовка файла или ресурса. Информация заголовка должна быть такой же, как и для метода *GET*. *C* помощью запроса с методом *HEAD* клиент может получать информацию о времени изменения документа, размере документа, типе документа или типе сервера.

Методы *POST* и *PUT* позволяют посылать и изменять данные на сервере соответственно. Метод *POST* служит для добавления нового ресурса и вызывает изменения на сервере. Метод *PUT* заменяет ресурсы на сервере данными из запроса. Разница между методами *PUT* и *POST* в том, что использование одинаковых запросов с методом *PUT* дадут одинаковый результат, а при использовании запроса с методом *POST* может иметь непредвиденные повторные выполнения операций.

Метод *DELETE* удаляет указанный в запросе ресурс. Таким образом, методы запросов в полной мере позволяют реализовать базовые функции *Create*-*Read*-*Update*-*Delete* (*CRUD*) для работы с любимыми хранилищами данных. В случае запросов, хранилищем данных будет выступать сервер, а клиент будет использовать эти функции.

На рисунке 1.2 представлено взаимодействие сервера с клиентом по протоколу *HTTP*.



Рисунок 1.2 – Взаимодействие сервера с клиентом по протоколу *HTTP*

## Графическая библиотека *DirectX*

*DirectX* является мощным программным интерфейсом, который позволяет разработчикам писать быстродействующие графические приложения, используя определённый стандартом набор библиотек и интерфейсов платформы *Windows*.

*DirectX*, как правило, используется для создания игр. Многие игровые движки рендеринга более высокого уровня построены на основе *Direct*3*D*. Тем не менее, и другие отрасли нуждаются в интерактивной визуализации 3*D* графики, например, для обучения в медицинской сфере, изучениях в научной сфере, а также при проектировании деталей или целой архитектуры зданий.

Программы все чаще начинают использовать преимущественно графический процессор, а именно блок обработки графики, для уменьшения нагрузки на центральный процессор.

*DirectX* является библиотекой низкого уровня, так как *API* напрямую управляет базовым графическим устройством, что позволяет создавать приложения в единственном экземпляре, которые могут быть запущены на любых видеокартах конечных пользователей [5, с. 18]. Работа с *DirectX* заключается в создании структуры, описывающей сцену в двумерном или трёхмерной пространстве, и передаче этой структуры на конвейер *DirectX*. Конвейер включает в себя 3 этапа: преобразование и освещение, который относятся к стадии геометрический преобразований, и растеризации, выполняющейся на стадии рендеринга. После прохождения данных через конвейер мы получаем изображение на экране.

Для работы с объектом используются указатели на их интерфейс, через которые и вызываются методы. Аналогичным образом происходит и создание объектов, посредством вызова метода через интерфейс.

*DirectX* основан на *Component* *Object* *Model* (*COM*) – объектно-ориентированной модели программирования, использующей несколько технологий. *COM* позволяет работать с объектно-ориентированной моделью любого языка, который его поддерживает [6, с. 29].

Функции многопоточности позволяют разработчикам программного приложения масштабировать сложные многоядерные архитектуры. Функции замещения поддерживают адаптивное масштабирование контента в различных разрешениях изображения, и увеличение производительности графического процессора, ориентируясь на вычислительную мощность, где это наиболее выгодно. А все общие вычисления учитывают новые подходы и алгоритмы для использования дополнительной вычислительной мощности. При этом каждый алгоритм и каждый метод имеют несколько способов реализации, которые необходимо учитывать.

1. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРОВОГО ПРИЛОЖЕНИЯ «КЛАД»

## Программные средства и паттерны проектирования «Фабричный метод» и «Декоратор» для реализации игрового приложения

Для разработки программного обеспечения необходима среда разработки *Visual* *Studio* 2019 и компоненты, позволяющие работать на платформе .*NET* и создавать проекты библиотеки классов и построений клиентских приложений *Windows* *Presentation* *Foundation* (*WPF*). Для работы с *DirectX* с помощью диспетчера пакетов *NuGet* необходимо установить библиотеку *SharpDX* – ядро для всех сборок *SharpDX*, а также библиотеки *SharpDX*.*Desktop*, *SharpDX*.*Direct*2*D*1, *SharpDX*.*DirectInput*, *SharpDX*.*DXGI*, *SharpDX*.*Mathematics*, позволяющие работать со средствами *DirectX*. Для создания спрайтовой графики в форматах .*bmp* и .*png* подойдет программное обеспечение *Adobe* *Photoshop* *CC* 2017. Для возможности расширения функционала разрабатываемого приложения необходимо использовать паттерны (шаблоны) проектирования.

Паттерн проектирования «Фабричный метод» является порождающим паттерном, определяющий интерфейс для создания объекта, но оставляет подклассам задачу решения о создании конкретного класса [7, с. 112]. Данный паттерн использует механизм полиморфизма, при котором классы всех конечных типов наследуются от одного базового класса.

Преимущества и недостатки паттерна «Фабричный метод».

Классы фабрик избавляют программиста от необходимости встраивать в код зависящие от приложения классы. Взаимодействие происходит только с интерфейсом класса, поэтому он может работать с любыми реализациями конкретных классов.

Непосредственное создание не является гибким решением, в то время как создание объектов с помощью паттерна внутри класса. Класс фабрики создает в подклассах операции-зацепки для предоставления расширенной версии объектов.

Клиентом для создания лишь одного нового объекта, наследуемого от интерфейса, необходимо создавать и наследник класса фабрики. В данном случае, порождение наследников оправдывается, если клиенту в любом случае придётся создавать наследников фабрики, иначе клиенту придётся работать с дополнительным уровнем наследников.

В игре «Клад» паттерн применим к классам, которые динамически добавляются и удаляются со сцены, что требует постоянного создания новых объектов. Такими классами в игре является подбираемое оружие, оружие в руках персонажа, пули и эффекты, накладываемые на персонажей. Поэтому в реализации необходимо создать абстрактные классы фабрик и их наследников с реализацией создания каждого объекта.

Такая структура позволит вносить изменения в проект без возникновения проблем.

На рисунке 2.1 представлена схема реализации паттерна «Фабричный метод».

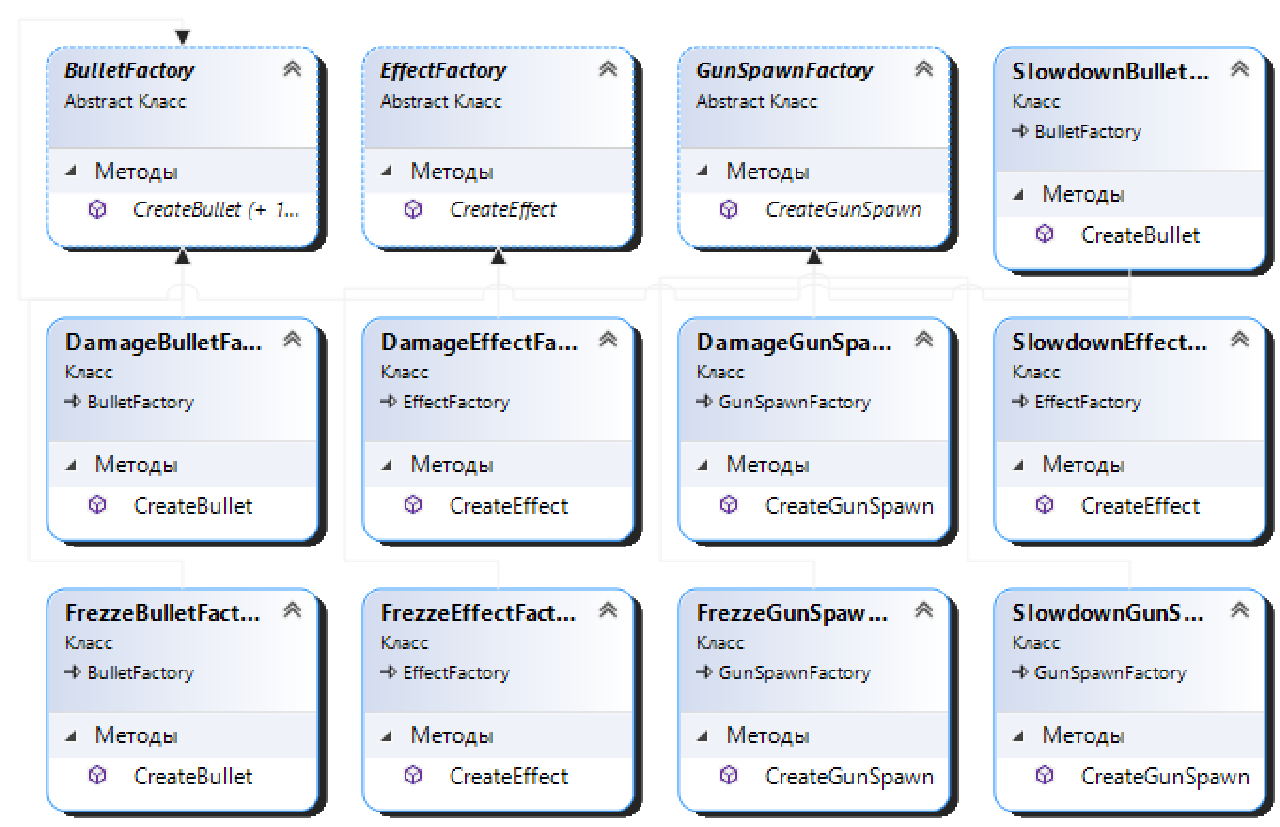


Рисунок 2.1 – Схема реализации паттерна «Фабричный метод»

Шаблон проектирования «Декоратор» позволяет динамически добавлять новые обязанности объекту и является гибкой альтернативой порождению классов с целью расширения функциональности [8, с. 174]. Для реализации паттерна необходимо создать класс, описывающий компонент, который декорируется, а затем класс, декорирующий этот компонент. Класс-декоратор реализует интерфейс компонента и позволяет замещать его. При этом класс декоратор не только реализует интерфейс компонента, но позволяет его дополнить необходимым функционалом.

Преимущества и недостатки паттерна «Декоратор».

Большая гибкость по сравнению со статическим наследованием. Добавление объекту новых обязанностей считается более гибким решением, чем в случае множественного наследования. Также паттерн позволяет динамически добавлять или удалять функциональность у объектов во время выполнения. При использовании множественного наследования приходиться создавать новый класс для каждой новой функциональности, что приводит к увеличению числа классов в системе, и как результат, к сложности системы. А при использовании «Декоратора» появляется возможность использовать несколько обязанностей в одном классе.

Паттерн позволяет избежать перегруженных функциями классов, от которых наследуются потомки, а также необходимости расширения этих классов в дальнейшей разработке. Паттерн позволяет добавлять новый функционал при необходимости. Поэтому не приходиться предусматривать все возможности в одном сложном, допускающем разностороннее применение классе – вместо этого определяется простой класс и декорируется его функциональность. Как результат, класс не имеет лишней функциональности, а при необходимости, «Декоратор» позволит добавить новую функциональность, даже если она и не предполагалась.

Стоит учитывать, что декорируемый компонент не идентичен исходному компоненту, и при разработке это стоит учитывать.

В игре «Клад» часто необходимо расширять функциональность объектов оружия, пуль и эффектов, в частности параметров этих объектов. Для каждого объекта потребуется создать класс базового декоратора. Он предоставит механизм подключения компонента и обеспечит переадресацию всех методов и свойств. Затем с помощью наследования создать классы конкретных декораторов, реализуя в них только методы и свойства с изменением функциональности.

Такая структура позволит получать различные объекты с различными параметрами путем декорирования. На рисунке 2.2 представлена схема реализации паттерна «Декоратор».

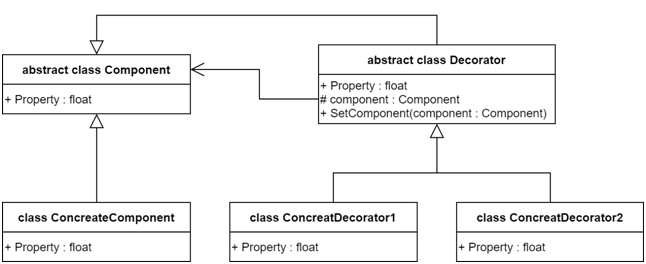


Рисунок 2.2 – Схема реализации паттерна «Декоратор»

## Игровой движок как основной компонент игрового приложения

Для реализации игровой логики требуется игровой движок – проект, разделяющий центральные компоненты игры (например, создание окна отображения и его настройки, создание ввода с клавиатуры, создание счетчика времени и игровых объектов, имеющих определённый набор компонентов для отображения и корректного поведения на сцене) от других логических частей игры [7, с. 121]. Игровой движок должен в полной мере поддерживать разработку основных механик игр жанра аркада. Проект определяется библиотекой классов и не имеет графической компонентов для визуальной разработки. В данном проекте требуются технологии перемещения по горизонтали, вертикали и диагонали, отображения векторной графики, анимация векторной графики, распознавание столкновений твердых тел и «триггеров».

Игровой движок предполагает определенный алгоритм работы. В частном случае алгоритм состоит из подготовки ресурсов для корректной работы приложения, циклического рендеринга объектов на протяжении всей игры и освобождения ресурсов перед закрытием. На рисунке 2.3 представлен алгоритм работы игрового движка приложения.

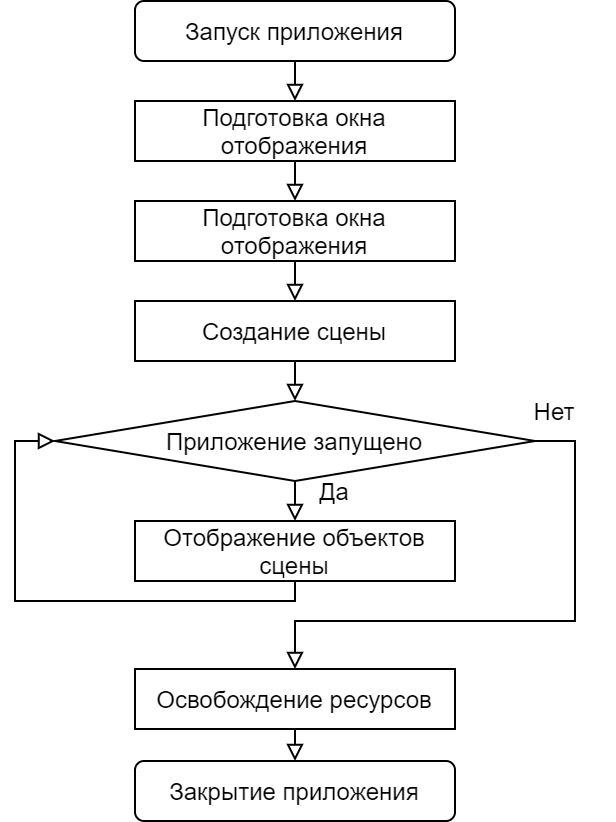


Рисунок 2.3 – Алгоритм работы игрового движка приложения

## Сетевое взаимодействие в игровом приложении

Проект игровой логики состоит из множества игровых объектов со специальными компонентами, независимыми друг от друга. Компоненты могут выполнять любые действия и расчеты каждый кадр. Их использование позволяет создавать игровые сущности без переиспользования кода и с уменьшением вероятности ошибки при тестировании. Сценарии игровой логики – специальные компоненты, наследуясь от которых можно реализовать любые игровые механики. У игрового объекта может быть ноль и больше таких сценариев. Таким образом, компоненты позволяют создавать любые игровые сущности. Например, это может быть игрок с компонентами отображения графики, функциями перемещения и логикой взаимодействия с предметами на уровне, или пустой объект со сценарием генерации объектов в игре. Ключевым элементом в проекте является сцена, которая создает и обновляет игровые объекты в игре, а также динамически добавляет и удаляет их.

Классы для сетевого взаимодействия хранят следующие данные:

* *bulletNetworkData* – класс для хранения направления, места появления пули, силы выстрела и тэга игрока, который стреляет;
* *networkData* – класс для хранения данных об игроках и призах;
* *iNetworkHandler* – интерфейс сетевого взаимодействия;
* *server* – сущность сервера, содержащая методы для обмена данными;
* *client* – сущность клиента, содержащая методы для обмена данными.

На рисунке 2.4 представлены классы, отвечающие за сетевое взаимодействие в игровом приложении.

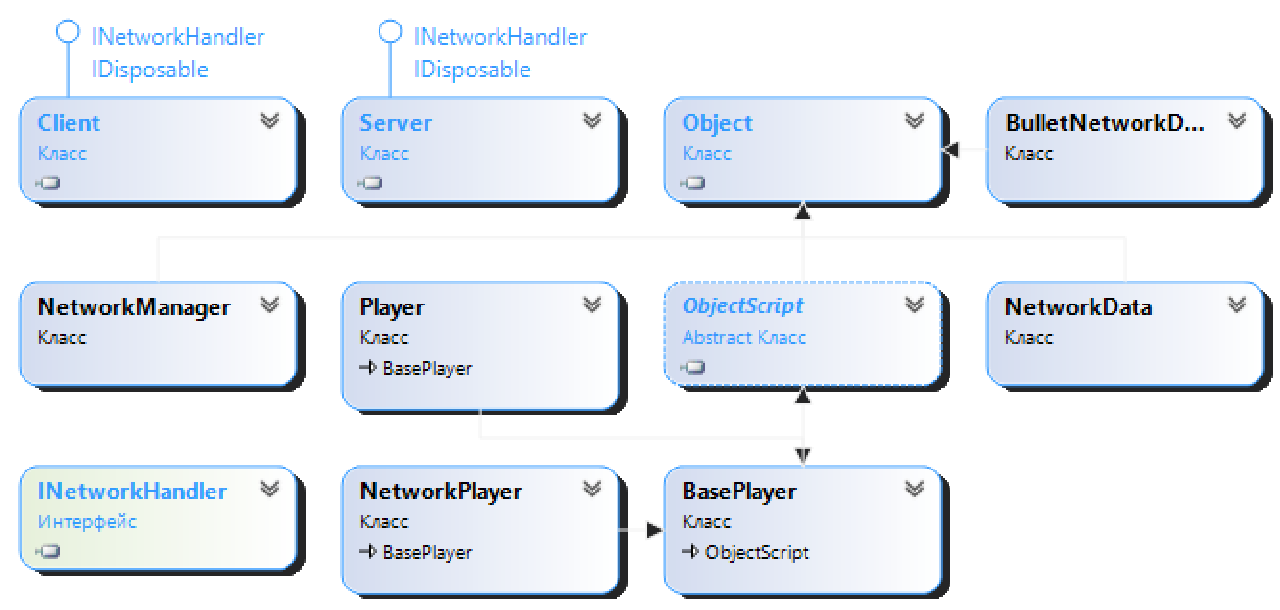


Рисунок 2.4 – Классы, отвечающие за сетевое взаимодействие в игровом приложении

Проект игровой логики коммуникации приложений по сети использует проект сетевого взаимодействия, который позволяет принимать и создавать подключения, отправлять и принимать данные, выполняться в разных потоках и обрабатывать ошибки. Так как игровой процесс предполагает игру для двоих пользователей, в проекте необходима архитектура для взаимодействия игроков в одноранговой сети, то есть, данные между устройствами будут передаваться напрямую, без участия общего сервера. Преимуществами реализации *peer*-*to*-*peer* соединения являются независимость от удаленного сервера и способность обработки большого объема данных, так как нагрузка будет распределяться на ресурсы собственного устройства. В качестве передаваемой информации проект сетевого взаимодействия использует данные в формате *JSON*, сериализуя и десериализуя их при отправке и получении соответственно.

Средство разработки *SharpDX* позволяет создавать собственное окно отображения. Однако для вывода статистики потребуется интегрировать создаваемое окно рендеринга в *WPF* приложение и создать элементы для отображения данных из проекта игровой логики. Элементы выводят количество собранных монет, подобранное оружие, эффект, действующий на персонажа и победителя игры. Для оповещения об изменении состояния игровых объектов необходимо использовать делегаты. Также, на *WPF* придётся возложить создание экземпляров игрового движка и игровой логики и высвобождение ресурсов у этих элементов, если это необходимо.

Класс *MazeScene* при игре по сети обновляет состояние игроков при вызове события *OnSceneDrawed*, которое вызывается в конце каждого кадра после отрисовки и обновления всех игровых объектов. Класс сцены взаимодействует с сетевыми сущностями клиента, описанной классом *Client* (Приложение А, код программы *Client*.*cs*), и сервера, описанной классом *Server* (Приложение А, код программы *Server*.*cs*) по интерфейсу *INetworkHandler* (Приложение А, код программы *INetworkHandler*.*cs*). Интерфейс предоставляет обобщенный метод *UpdateData* для обновления данных по сети, и событие *OnDataGot*, вызываемое при получении данных по сети. При создании экземпляра сервера создается экземпляр класса *HttpListener* для прослушивания подключений и обработки запросов. При вызове метода *UpdateData* сервер ожидает запрос и при получении его забирает данные другого игрока из тела запроса, если они существуют, а затем в ответ записывает данные игрока на этом устройстве. При создании экземпляра клиента создает экземпляр класса *HttpClient* для отправки запросов. При вызове метода *UpdateData* клиент отправляет *POST* запрос, включая данные игрока с устройства в тело запроса, и ожидает ответ с данными другого игрока.

Синхронизация игрового времени происходит при получении информации по сети. Пользовательские приложения приостанавливают работу и ожидают данные друг от друга. В случае с приложением, выступающим сервером, они ожидает запрос на получение данных от клиента. Ожидание приостанавливает основной поток выполнения игрового приложения. В случае с приложением, выступающим клиентом, он сначала отправляет свои данные, а затем ожидает ответ от сервера. Так же как и у сервера, ожидание блокирует основной поток клиентского приложения до тех пор, пока не будут получены данные. Таким образом, пока оба пользователя не получат данные друг от друга, их приложения не продолжат выполнение.

Получение данных в игре происходит в конце визуализации каждого кадра, поэтому интервал между обменом данных привязывается к циклу обновления приложения и одинаковый у обоих пользователей. После того, как каждое пользовательское приложение получит данные от другого устройства, продолжается выполнение основного потока, применяются полученная по сети информацию к игровым объектам, и начинается визуализация нового кадра. В результате, первый и второй игрок имеют общее игровое время, привязанное к циклу обновления приложения и ожиданию получения данных.

На рисунке 2.5 представлена схема сетевой архитектуры приложения.

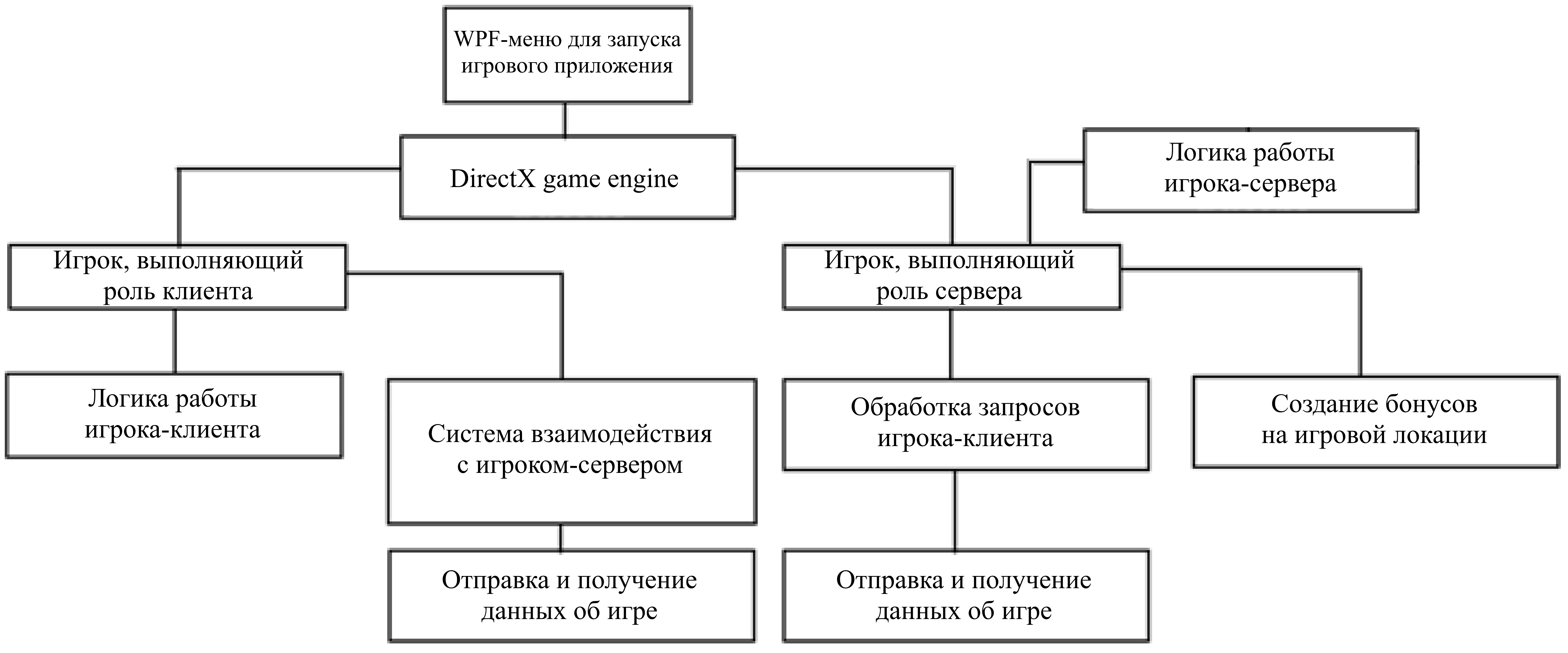


Рисунок 2.5 – Схема сетевой архитектуры приложения

## Программная реализация игрового приложения

Класс *RenderingApplication* (Приложение А, код программы *RenderingApplication*.*cs*) является центрального компонента проекта *EngineLibrary*. В конструкторе данного класса создается экземпляр класса *RenderForm*, представляющий собой окно рендеринга, и инициализация свойств: название окна, размер окна, возможность изменять размер окна, режим отображения окна верхнего уровня. Также в конструкторе создается объект класса *RenderingSystem* (Приложение А, код программы *RenderingSystem*.*cs*), отвечающий за инициализацию необходимых настроек и создания целевого окна рендеринга. Данный класс имеет статические методы *LoadBitmap* и *LoadAnimation*, загружающие изображение и последовательность изображений соответственно.

Для корректного запуска игрового приложения требуется установить наследуемые от абстрактного класса *Scene* (Приложение А, код программы *Scene*.*cs*) объекты. Сцена устанавливается в методе *SetScene*, который запускает создание сцены в инициализированном целевом окне рендеринга. Класс *Scene* имеет три списка игровых объектов: текущие игровые объекты рендеринга, объекты для добавления в следующем кадре на сцену и объекты для удаления в следующем кадре со сцены, и имеет следующие методы:

* *createGameObjectsOnScene* – абстрактный метод создания игровых объектов перед запуском рендеринга;
* *drawScene* – метод рендеринга игровых объектов на сцене;
* *addRenderGameObjects* и *RemoveRenderGameObjects* – методы для добавления и удаления игровых объектов в следующем кадре;
* *endScene* – виртуальный метод, оповещающий об окончании рендеринга сцены.

Обработка пользовательского ввода осуществляется в классе *Input* (Приложение А, код программы *Input*.*cs*), имеющий методы для проверки нажатия клавиши и получения положительных и отрицательных значений стандартных осей перемещения. Данный класс обращается к *InputHandler* (Приложение А, код программы *InputHandler*.*cs*), который отвечает за обновления состояния клавиатуры и мыши и считывания с них нажатых клавиш с помощью средств *SharpDX*.

Абстрактный класс *ObjectScript* (Приложение А, код программы *ObjectScript*.*cs*) представляет собой сценарий выполнения игрового объекта на сцене. Имеет абстрактные методы *Start*, инициализирующий игровой объект при создании, и *Update*, описывающее поведение этого объекта в каждом кадре.

Таким образом, проект *EngineLibrary* позволяет создавать сцену и игровые объекты на ней, а для реализации игровой логики используется компонент *ObjectScript*.

Проект игровой логики содержит реализацию конкретной игры «Клад» на основе спроектированного игрового движка. Главным элементом является класс *MazeScene* (Приложение А, код программы *MazeScene*.*cs*), описывающий сцену с лабиринтом. Данный класс реализует паттерн «одиночка», гарантируя, что экземпляр класса единственный и позволяет взять ссылку на сцену в любом месте проекта.

Генерация призов происходит в классе *SpawnManager* (Приложение А, код программы *SpawnManager*.*cs*), наследуемый от *ObjectScript*, и хранится в пустом игровом объекте. Создание разных экземпляров оружия реализуется с помощью шаблона проектирования «Фабричный метод».

Класс *Player* (Приложение А, код программы *Player*.*cs*), наследуемый от *ObjectScript*, позволяет управлять игровым объектом персонажа. Данный также содержит и позволяет изменять количество собранных сокровищ. В методе *Update* вычисляется передвижение игрового объекта персонажа по сцене, а также просчитывается столкновение со стенами, взаимодействие с лестницами и платформами.

Игрок может стрелять только в том случае, если подобрал оружие. Различные типы оружия реализуются с помощью паттерна «Декоратор», изменяя свойства оружия (Приложение И). Абстрактный класс *Gun* (Приложение А, код программы *Gun*.*cs*) имеет содержит необходимую для всех наследуемых оружий логику стрельбы, перезарядки и исчезновения по истечению времени в методе *Update*. При этом данный класс устанавливает абстрактные методы *LoadAnimation* и *SpawnBullet*, требующие реализацию у наследуемых классов. Абстрактный класс *GunDecorator* (Приложение А, код программы *GunDecorator*.*cs*) декорирует свойства *UseTime* и *ReloadTime*, а декорируемый класс задается в методе *SetDecoratedGun*. Класс *DamageGun* (Приложение А, код программы *DamageGun*.*cs*) реализует абстрактный класс *Gun* и декорируется классами *SlowdownGun* (Приложение А, код программы *SlowdownGun*.*cs*) и *FrezzeGun* (Приложение А, код программы *FrezzeGun*.*cs*).

Пули создаются в зависимости от типа оружия. Создание пуль реализуется с помощью паттерна проектирования «Фабричный метод». Различные типы пуль осуществляется аналогичным образом, что и оружие. Абстрактный класс *Bullet* (Приложение А, код программы *Bullet*.*cs*) задает передвижение и просчитывает столкновение пули в методе *Update*. При столкновении с игровым объектом стены вызывается метод его разрушения и позволяет проходить сквозь них, а при столкновении с персонажем вызывается метод *PlayerInteraction*, реализуемый в классах наследниках. Абстрактный класс *BulletDecorator* (Приложение А, код программы *BulletDecorator*.*cs*) декорирует свойство *Speed*, а декорируемый класс устанавливается в методе *SetDecoratedBullet*. Класс *DamageBullet* (Приложение А, код программы *DamageBullet*.*cs*) реализует абстрактный класс *Bullet* и декорируется классами *SlowdownBullet* (Приложение А, код программы *SlowdownBullet*.*cs*) и *FrezzeBullet* (Приложение А, код программы *FrezzeBullet*.*cs*).

Каждая пуля создает конкретный эффект, действующей на персонажа. С помощью шаблона проектирования «Фабричный метод» осуществляется создание эффектов, а с помощью паттерна «Декоратор» реализуются различные типы эффектов. Абстрактный класс *Effect* (Приложение А, код программы *SlowdownBullet*.*cs*) описывает общую логику игрового объекта в методе *Update*, деактивируя эффект по истечению его времени действия. Класс имеет абстрактные методы *ActivateEffect*, активирующий эффект на определённом персонаже, и *DeactivateEffect*, которые отключает его действие с персонажа и удаляет со сцены. Абстрактный метод *BehaviorOnScene* расширяет логику поведения игрового объекта на сцене. Абстрактный класс *EffectDecorator* (Приложение А, код программы *SlowdownBullet*.*cs*) декорирует свойство *EffectTime*, а декорируемый класс устанавливается в методе *SetDecoratedEffect*. Класс *DamageEffect* (Приложение А, код программы *SlowdownBullet*.*cs*) реализует абстрактный класс *Effect*, уничтожая персонажа, который возрождается спустя время, и декорируется классами *SlowdownGun* (Приложение А, код программы *SlowdownBullet*.*cs*) и *FrezzeGun* (Приложение А, код программы *SlowdownBullet*.*cs*), замедляющий и замораживающий игрока соответственно.

Оповещения игрового приложения о собранных монетах, подбираемого оружия, действующего эффекта и конца игры осуществляется делегатами, созданными в статическом классе *GameEvents* (Приложение А, код программы *GameEvents*.*cs*). Таким образом, мы можем добавить собственные методы отображения игровой статистики в делегаты, вызов, который осуществляется в проекте игровой логики. Делегаты обеспечивают возможность последовательного вызова нескольких методов, а также вызова как статических, так и экземплярных методов [9, с. 434].

1. ВЕРИФИКАЦИЯ И АПРОБАЦИЯ ИГРОВОГО ПРИЛОЖЕНИЯ «КЛАД»

## Принцип работы игрового приложения

При запуске игрового приложения открывается проект *WPF* с пользовательским интерфейсом (Приложение Д, рисунок Д.1). В классе *GameWindow* инициализируются компоненты интерфейса и создаются экземпляры классов игрового движка *RenderingApplication*. Методы класса для отображения статистики подписываются на делегаты проекта игровой логики. Элементы *Image* и *TextBlock* осуществляют вывод статистики и помощи по управлению в игре, а элементы *Button* переключают панели взаимодействия. Экземпляр окна рендеринга *RenderForm* из класса *RenderingApplication* транслируется в пользовательский интерфейс с помощью элемента *WindowsFormsHost*, который позволяет использовать элементы *Windows* *Forms* в *WPF* [10, с. 98].

При нажатии на кнопку «*Single*-*player*» интерфейс сменяется на экран с описанием клавиш управления и кнопкой «*Play*» (Приложение Д, рисунок Д.2), при нажатии на которую создается экземпляр *MazeScene* для игры на одном экране. Конструкторы игровых персонажей инициализируются соответствующим управлением и сценариями игровой логики. При Нажатии на клавишу «*Play* *LAN*» интерфейс сменяется на экран выбора создания игры или присоединения к существующей (Приложение Д, рисунок Д.3). Если пользователь нажимает на кнопку «*Create* *Game*», его приложения создает сервер для получения данных и ожидает подключения. Другой игрок может подключиться к нему, нажав кнопку «*Connect* *to* *the* *Game*» и подтвердив *IP*-адрес игрока, создавшего сеанс. Когда сервер принял подключения клиента, в обоих приложения создается экземпляр *MazeScene* для игры по сети. В таком случае, один из конструкторов персонажей инициализируется альтернативным сценарием игрока, который обновляется по сети. После создается экземпляр класса *NetworkManager* (Приложение А, код программы *NetworkManager*.*cs*), который подписывает классы на запись и получения сетевых данных.

Затем создание игры для обоих случаев происходит по одинаковому алгоритму – создаются игровые персонажи и уровень. Лабиринт создается из случайно выбранного *bmp* файла, который содержит структура лабиринта, где цвет каждого пикселя обозначает элемент игровой сцены. Создание игровых объектов лабиринта происходит в экземпляре класса *MazeElementsFactory* (Приложение А, код программы *MazeElementsFactory*.*cs*). Также сцена сохраняет пустые клетки в список для последующей генерации призов в лабиринте и имеет методы добавления и удаления игровых объектов со сцены. При удалении бонуса со сцены, его позиция снова добавляется в список генерации призов, а при удалении сокровища со сцены, проверяется окончание игры.

После инициализации сцены запускается зацикленная инфраструктуру рендеринга, которая формирует каждый кадр игрового окна соответствующим методом *RenderCallback*. Данный метод позволяет игровому приложению каждый кадр обновлять состояние компонентов клавиатуры и времени, а затем выводить изображение текущего кадра с помощью рендеринга игровых объектов сцены. Игровые объекты требуется проинициализировать нужными для его цели компонентами, после чего игровой объект каждый кадр выводится на экран методом *Draw*, пока игровой объект существует на сцене. Рендеринг объекта заключается в создании матрицы перемещения, вращения и размера и рисования изображения на основе это матрицы. Затем перед началом вызова следующего *RenderCallback* записываются и обновляются сетевые данные для игроков и появившихся пуль. По сети передаются данных о позиции игрока, его повороте и состоянии движения, а также позиции появления, направления и типе пули, создаваемой на сцене.

Статический класс *Time* (Приложение А, код программы *Time*.*cs*) позволяет осуществлять работу с временем. Данный класс хранит текущее время *CurrentTime* с начала игры и время *DeltaTime*, которое потребовалось для рендеринга последнего кадра. Каждый вызов *RenderCallback* обновляет значение свойств *CurrentTime* и *DeltaTime*. Использование *DeltaTime* как множителя при вычитании или прибавлении значений за определённую единицу времени делает приложение независимым от частоты кадров.

Основной механикой игры является сбор монет. Класс *Coin* (Приложение А, код программы *Coin*.*cs*), наследуемый от *ObjectScript*, в методе *Update* проверяет столкновение с игровыми персонажами, после чего игровой объект удаляется из списка рендеринга, а значение собранных монет у игрока увеличивается на единицу и оповещает систему об изменении с помощью соответствующего делегата.

При генерации бонусов проверяется возможность их создания в зависимости от пройденного времени с последней генерации, и если это время прошло, то на сцене создается игровой объект с оружием, описываемый классом *GunSpawn* (Приложение А, код программы *GunSpawn*.*cs*). Данный класс хранит экземпляр оружия, которое подбирается игроком при столкновении с бонусом. Выбор игрового бонуса осуществляется случайным образом и конкретными шансами для каждого оружия.

Если игрок подобрал оружие, он имеет возможность стрелять по нажатию клавиши стрельбы на клавиатуре. Игровой объект персонажа устанавливается родительским объектом и позволяет передвигать оружие по сцене относительно персонажа. При стрельбе генерируются пули, которые перемещаются по горизонтали и имеют методы проверки столкновения с игровыми объектами. Если объектом оказывается персонаж игрока, то на него накладывается соответствующий пули эффект. Если же объектом оказалась разрушаемая стена, описанная классом *BreakWall* (Приложение А, код программы *BreakWall*.*cs*), то у неё вызывается метод *DestroyWall*, который запускает проигрывание анимации разрушения, а после чего стена становится не активной на определённое время. В любом случае, при столкновении с объектом, игровой объект пули убирается из списка рендеринга объектов сцены.

При завершении игры закрывается элемент управления *WindowsFormsHost* с окном рендеринга, отключается рендеринг игровых объектов на сцене и выводится результаты игры. При закрытии приложения или непредвиденной ошибки закрывается интернет-соединение, высвобождаются ресурсы проекта игрового движка.

## Результаты тестирования игрового приложения

Проект *TestGameApp* позволяет протестировать разработанные игровые библиотеки. Для тестирования используются статические методы класса *Assert* и классы модульного тестирования, которые находится в пространстве имен *Microsoft*.*VisualStudio*.*Test*-*Tools*.*UnitTesting*. Выделяется следующий набор классов модульных тестов, позволяющий протестировать игровое приложение:

* *testMovement* (Приложение А, код программы *TestMovement*.*cs*), тестирующий класс *TransformComponent*, в частности изменение его свойства *Position*, отвечающее за расположение игровых объектов на сцене, с помощью метода *AreEqual*, проверяющий равенство двух значений;
* *testCollision* (Приложение А, код программы *TestCollision*.*cs*), позволяющий протестировать класс *ColliderComponent*, который отслеживает столкновение игровых объектов, с помощью методов *IsTrue* и *IsFalse*, проверяющие булевое значение;
* *testDecoratedGun* (Приложение А, код программы *TestDecoratedGun*.*cs*), содержащий методы для тестирования декорируемых классов *SlowdownGun* и *FrezzeGun*, описывающие поведение игровых объектов оружия, с помощью метода *AreEqual*, проверяющий равенство двух значений;
* *testDecoratedBullet* (Приложение А, код программы *TestDecoratedBullet*.*cs*), содержащий методы для тестирования декорируемых классов *SlowdownBullet* и *FrezzeBullet*, описывающие поведение игровых объектов пуль, с помощью метода *AreEqual*, проверяющий равенство двух значений;
* *testDecoratedEffect* (Приложение А, код программы *TestDecoratedEffect*.*cs*), содержащий методы для тестирования декорируемых классов *SlowdownEffect* и *FrezzeEffect*, описывающие поведение игровых объектов эффектов, накладываемых на персонажей, с помощью метода *AreEqual*, проверяющий равенство двух значений;
* *testNetworkHandlers* (Приложение А, код программы *TestNetwork*-*Handlers*.*cs*), содержащий метод для тестирования сетевой коммуникации между классами *Server* и *Client*, выполняющие обмен данными и записывающие количество подключений, проверяемых с помощью метода *AreEqual*.

Данный проект позволяет проверить, корректно ли работают разработанные классы и их методы и свойства.

На рисунке 3.1 представлены результаты тестирования классов разработанного игрового приложения.



Рисунок 3.1 – Результаты тестирования классов разработанного игрового приложения

Также в процессе тестирования анализируется сетевое взаимодействие между устройствами и объем передаваемых данных. В результате определяется минимальная пропускная способность сети, приблизительно равная 150 килобитам в секунду. На рисунке 3.2 представлена иллюстрация с пропускной способностью сети при работе приложения.

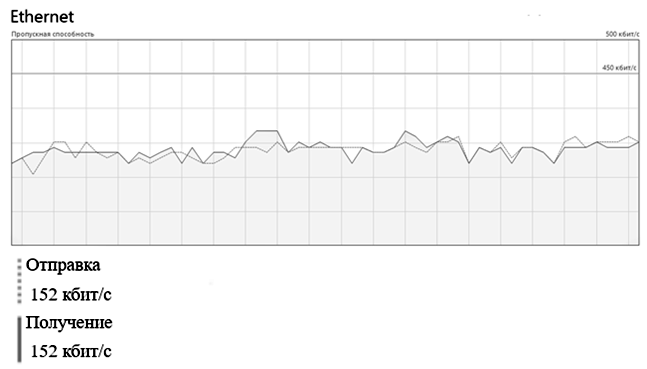


Рисунок 3.2 – Пропускная способность сети при работе приложения

Также было осуществлено нагрузочное тестирование приложения с помощью *Apache* *JMeter*.

Результаты прохождения нагрузочных тестов представлены на рисунке 3.3.

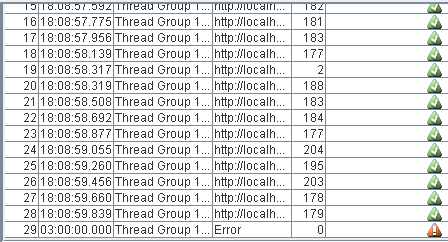


Рисунок 3.3 – Результаты прохождения нагрузочных тестов

На рисунке 3.4 представлен график времени отклика приложения.

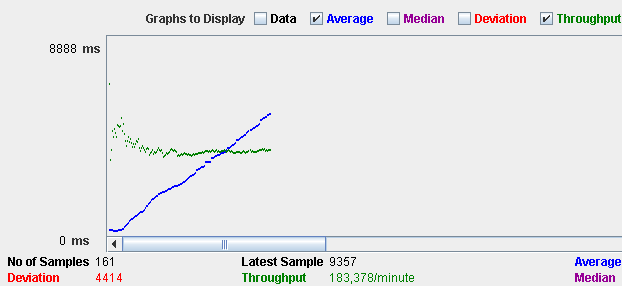


Рисунок 3.4 – График времени отклика приложения

## Результаты верификации игрового приложения

При запуске *WPF* проекта открывается интерфейс приложения (Приложение Д, рисунок Д.1). После начала игры, объекты игроков появляются на сцене в местах, заданных в хранимой структуре лабиринта формата .*bmp*. На старте игрового приложения игроки не имеют возможности стрелять, пока не подберут оружие. Оружие в лабиринте генерируется случайным образом (Приложение Д, рисунок Д.4).

В таблице 3.1 представлены управления игровыми персонажами при игре на одном экране.

Таблица 3.1 – Управление игровыми персонажами при игре на одном экране

|  |  |  |
| --- | --- | --- |
| Действие | Красный персонаж | Синий персонаж |
| 1 | 2 | 3 |
| Передвижение по горизонтали (влево/вправо) | *A*/*D* | *Left*/*Right* |
| Передвижение по вертикали | По лестнице (вверх/вниз) | |
| *W*/*S* | *Up*/*Down* |
| На платформе (вниз) | |
| *S* | *Down* |
| Стрельба (при наличии оружия) | *Space* | *NumLock* *Enter* |

При игре по сети обоим игрокам необходимо подключение к интернету или находиться одной локальной сети. Для игрока, создающего сеанс игры, необходимо запускать приложение от имени администратора, так как класс *HttpListener* запрашивает доступ к сетевым ресурсам компьютера. Для подключения другому игроку необходимо ввести *IP*­-адрес устройства игрока, создающего сеанс игры. При успешном подключении другого пользователя у обоих игроков открывается игровая сцена и начинается игровой цикл. В таблице 3.2 представлено управление игровыми персонажами при игре по сети.

Таблица 3.2 – Управление игровыми персонажами при игре по сети

|  |  |
| --- | --- |
| Действие | Персонаж |
| Передвижение по горизонтали (влево/вправо) | *A*/*D* |
| Передвижение по вертикали | По лестнице (вверх/вниз) |
| *W*/*S* |
| На платформе (вниз) |
| *S* |
| Стрельба (при наличии оружия) | *Space* |

Первый игрок управляет синим персонажем, а второй игрок – красным. Каждый игровой объект персонажа имеет анимации покоя и бега в обоих направлениях. Цель каждого игрока заключается в сборе всех монет в лабиринте. В процессе игры можно передвигаться по горизонтали и вертикали при столкновении с лестницей, а также падать вниз через платформы. Треснувшие стены можно сломать, если в них выстрелить, после чего в том месте можно пройти пока стена снова не появилась. При столкновении с монетой она исчезает и добавляется в счет игрока, подобравшего монету.

В процессе игры можно подбирать бонусное оружие. Каждое оружие имеет уникальный тип эффекта, накладываемый на персонажа. В игре реализовано 3 типа оружия:

* тип урона с шансом появления 20 процентов, который при попадании уничтожает игрового персонажа и он теряет половину собранных сокровищ, эти сокровища может получить любой из персонажей при столкновении с наградой, уничтоженный персонаж появляется на стартовой позиции спустя 5 секунд;
* тип замедления с шансом появления 50 процентов, который при попадании замедляет скорость передвижения игрока, персонаж передвигается в два раза медленнее в течение 10 секунд, эффект не складывается при повторном попадании;
* тип заморозки с шансом появления 30 процентов, который при попадании останавливает игрока, персонаж не может двигаться в течение семи с половиной секунд, эффект также не складывается при повторном попадании.

На рисунке 3.5 представлены генерируемые в лабиринте призы с оружием. На рисунке 3.6 представлены действующие эффекты, создаваемые пулями.



Рисунок 3.5 – Генерируемые в лабиринте призы с оружием

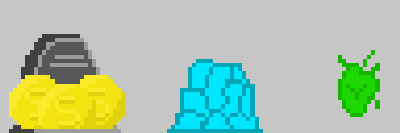


Рисунок 3.6 – Эффекты, создаваемые пулями

Информационная статистика выводится в *WPF* приложение с помощью делегатов. При столкновении с монетой изменяется значение строкового свойства *Text* элемента *TextBlock*, связанного с конкретным игроком. При столкновении с подбираемым бонусом оружия в свойство *Source* компонента *Image* подгружается изображение из *WPF*-проекта в соответствии с подобранным оружием для определённого игрока. По истечению времени использования оружие оно пропадает, как и возможность стрелять из него, а элемент *Image* скрывается в пользовательском интерфейсе. При попадании пули в игрового персонажа на него накладывается эффект, а название типа эффекта заполняется в свойство *Text* элемента *TextBlock*. После окончания действия эффекта в поле *Text* устанавливается пустая строка. На рисунке 3.7 представлена статистика каждого игрока из пользовательского интерфейса во время игры.

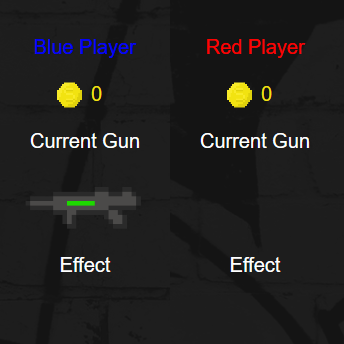


Рисунок 3.7 – Статистика каждого игрока из пользовательского интерфейса во время игры

На протяжении всей игры данные обновляются постоянно каждый кадр. В случае возникновения ошибки передачи данных или долгом ожидании ответа, приложение продолжает свою работу и производит попытку обновления данных в следующем кадре. В случае потери соединения или возникновения непредвиденной ошибки, игровое приложение закрывается у обоих пользователей.

В приложение загружено 5 разных уровней, которые выбираются случайным образом. Игроки не могут выработать конкретную стратегию для победы, так как они не знают, какой запускается уровень. Разнообразие в игру вносят бонусы различного типа, которые позволяют помешать другому игроку в сборе монет. Однако можно сосредоточиться только на монетах и постараться собрать их быстрее противника. Поэтому игра не кажется скучной из-за необходимости каждый раз стараться победить в новых условиях. Также на вовлеченность в игровой процесс может повлиять соревновательные отношения между игроками.

Игра заканчивается при сборе всех монет на игровом уровне. Победитель становится игрок, собравший большее количество монет. Окно рендеринга закрывается, а в окне *WPF* приложения показывается победитель.

На рисунке 3.8 представлен пользовательский интерфейс приложения после окончания игры.

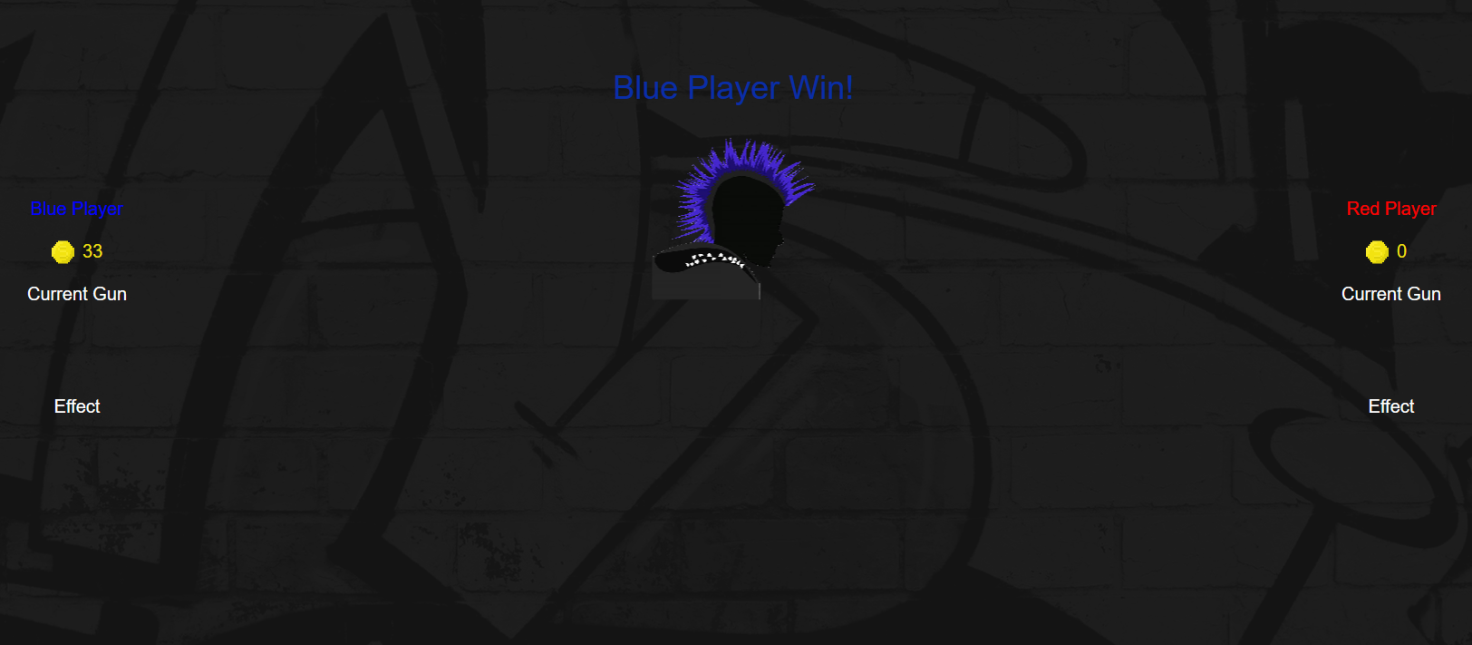


Рисунок 3.8 – Пользовательский интерфейс приложения после окончания игры

После завершения игры пользователи на информационных панелях могут узнать свою статистику на момент окончания игры и посмотреть количество собранных за игру монет. Пользователи могут закрыть приложение и при необходимости запустить его снова, чтобы заново сыграть в игру.

# ЗАКЛЮЧЕНИЕ

Результатом проделанной работы является игровое приложение жанра аркада для игры двух пользователей на одном экране или по сети интернет. Каждый игрок управляет с клавиатуры персонажем в лабиринте, и их задача заключается в сборе всех монет на уровне.

В ходе разработки игрового приложения «Клад» был проведён аналитический обзор игрового жанра «Аркада», особенностей разработки многопользовательских игр и их средств разработки. В процессе анализа рассматривалась проблема синхронизации игрового времени между пользователями и определялись способы её решения. Затем была определена архитектура, алгоритм работы и способ сетевой коммуникации в игровом приложении. Так как игровое приложение предусмотрено для двух игроков, в качестве архитектуры сетевого взаимодействия было принято решение использовать реализацию *peer*-*to*-*peer* соединения с помощью протокола *HTTP*. Основным этапом является разработка проекта сетевого взаимодействия, реализующая обмен данными пользователей по сети с решением проблемы синхронизации игрового времени путём управления общим временем пользователей, а также реализацией обработки сетевых ошибок и предотвращением конфликтов, связанных с подключением других пользователей к существующей игровой сессии. Разработанные классы протестированы модульными тестами, выявленные по результатам тестирования ошибки устранены, что гарантирует корректную работу приложения и ликвидирует возможность получения ошибки пользователем. Также проводилась оценка сетевой коммуникации, определялась минимальная пропускная способность. Конечным этапом стало создание пользовательского интерфейса, который реализует подключение, запускает приложение и получает игровую статистику.

В разработанном приложении создаётся карта, по которой перемещаются два пользователя, попутно собирая сокровища и бонусы. Игра интуитивно понятная и имеет низкий порог вхождения. Приложение прошло опытную эксплуатацию с положительными результатами, представленными в Приложении Е.

Курсовая работа проверена системой «Антиплагиат». Оригинальность составила 94,56 процентов. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».

# Список использованных источников

1. *Rouse*, *R*., *Game* *Design*: *Theory* *and* *Practice*. / *R*. *Rouse*. – *Texas*: *Los* *Rios* *Boulevard* *Plano*, 2016. – 723 *p*.
2. Синхронизация данных в мультиплеерных играх [Электронный ресурс]. – Режим доступа: *https*://*habr*.*com*/*ru*/*post*/328702/ – Дата доступа: 27.11.2022.
3. *Pollard*, *B*., *HTTP*/2 *in* *Action*. / *B*. *Pollard*. – *NY*: *Manning*, 2019. – 416 *p*.
4. Тимачев, С., Обзор *HTTP*. / С. Тимачев; под общей редакцией М. Коломыцев. – М.: Диалог-Мифи, 2003. – 96 с.
5. Жарков, В., Основы программирования игр и приложения на *Visual* *C*# 2008 и *DirectX* 10 для настольных компьютеров и ноутбуков. / В. Жарков. – М.: Жарков Пресс, 2007. – 908 с.
6. Миллер, Т., *DirectX* 9 с управляемым кодом. Программирование игр и графика. / Т. Миллер. – М.: Издательский дом КомБук, 2005. – 401 с.
7. Гамма, Э., Приёмы объектно-ориентированного программирования. Паттерны проектирования. / Э. Гамма. – СПб.: Питер, 2015. – 368 с.
8. *Gregory*, *J*., *Game* *Engine* *Architecture*. *Second* *edition*. / *J*. *Gregory*. – *Florida*: *CRC* *Press*, 2018. – 1018 *p*.
9. Рихтер, Дж., *CLR* *via* *C*#. Программирование на платформе *Microsoft* .*NET* *Framework* 4.5 на языке *C*#. 4-е издание. / Джон Рихтер. – СПб.: Питер, 2019. – 896 с.
10. Мак-Дональд, М., *WPF*: *Windows* *Presentation* *Foundation* в .*NET* 4.5 с примерами на *C*# 5.0 для профессионалов. / М. Мак-Дональд. – М.: Вильямс, 2019. – 1024 с.

# ПРИЛОЖЕНИЕ А

(обязательное)

**Листинг программы «Клад»**

**Код программы для *RenderingApplication*.*cs*:**

using System;

using System.Drawing;

using SharpDX.Windows;

using SharpDX.Direct2D1;

namespace EngineLibrary.EngineComponents

{

/// <summary>

/// Класс отрисовки и обновление всех процессов

/// </summary>

public class RenderingApplication : IDisposable

{

private const float unitsPerHeight = 15.0f;

private const int widthOfApplication = 1280;

private const int heightOfApplication = 720;

private WindowRenderTarget renderTarget;

private RenderingSystem rendering;

private InputHandler input;

private Scene scene;

/// <summary>

/// Форма отрисовки

/// </summary>

public RenderForm RenderForm { get; private set; }

/// <summary>

/// Конструктор класса, инциализирующий его компоненты

/// </summary>

public RenderingApplication()

{

RenderForm = new RenderForm("Game");

RenderForm.ClientSize = new Size(widthOfApplication, heightOfApplication);

RenderForm.TopLevel = false;

RenderForm.AllowUserResizing = false;

RenderForm.FormBorderStyle = System.Windows.Forms.FormBorderStyle.None;

rendering = new RenderingSystem(RenderForm);

renderTarget = rendering.RenderTarget;

input = new InputHandler(RenderForm);

Input.SetupInputHandler(input);

}

/// <summary>

/// Установка игровой сцены

/// </summary>

/// <param name="targetScene"></param>

public void SetScene(Scene targetScene)

{

scene = targetScene;

scene.InitializeScene(renderTarget, RenderForm.ClientSize.Height / unitsPerHeight);

}

/// <summary>

/// Запуск приложения

/// </summary>

public void Run()

{

RenderLoop.Run(RenderForm, RenderCallback);

}

/// <summary>

/// Формирование каждого кадра

/// </summary>

private void RenderCallback()

{

input.UpdateKeyboardState();

Time.UpdateTime();

renderTarget.BeginDraw();

renderTarget.Clear(SharpDX.Color.Black);

scene.DrawScene();

renderTarget.EndDraw();

if (!scene.IsDrawScene)

{

Dispose();

}

}

/// <summary>

/// Освобождение ресурсов

/// </summary>

public void Dispose()

{

RenderForm.Dispose();

rendering.Dispose();

input.Dispose();

}

}

}

**Код программы для *RenderingSystem*.*cs*:**

using System;

using System.Collections.Generic;

using SharpDX;

using SharpDX.Windows;

using SharpDX.Direct2D1;

using SharpDX.WIC;

using Direct2D1 = SharpDX.Direct2D1;

using WIC = SharpDX.WIC;

using RenderTargetFactory = SharpDX.Direct2D1.Factory;

namespace EngineLibrary.EngineComponents

{

/// <summary>

/// Класс создания системы отрисовки

/// </summary>

public class RenderingSystem : IDisposable

{

private RenderTargetFactory renderTargetFactory;

/// <summary>

/// Окно отрисовки

/// </summary>

public WindowRenderTarget RenderTarget { get => renderTarget; }

private WindowRenderTarget renderTarget;

private static ImagingFactory imageFactory;

private static WindowRenderTarget staticRenderTarget;

/// <summary>

/// Коснтруктор, создаюший систему отрисовки в форме

/// </summary>

/// <param name="form">Форма для отрисовки</param>

public RenderingSystem(RenderForm form)

{

renderTargetFactory = new RenderTargetFactory();

imageFactory = new ImagingFactory();

RenderTargetProperties renderProperties = new RenderTargetProperties()

{

DpiX = 0,

DpiY = 0,

MinLevel = FeatureLevel.Level\_10,

PixelFormat = new Direct2D1.PixelFormat(SharpDX.DXGI.Format.B8G8R8A8\_UNorm, AlphaMode.Premultiplied),

Type = RenderTargetType.Hardware,

Usage = RenderTargetUsage.None

};

HwndRenderTargetProperties winProperties = new HwndRenderTargetProperties()

{

Hwnd = form.Handle,

PixelSize = new Size2(form.ClientSize.Width, form.ClientSize.Height),

PresentOptions = PresentOptions.None

};

renderTarget = new WindowRenderTarget(renderTargetFactory, renderProperties, winProperties);

staticRenderTarget = renderTarget;

}

/// <summary>

/// Загрузка изображения из файла

/// </summary>

/// <param name="imageFileName">Путь к файлу</param>

/// <returns>Изображение</returns>

public static Direct2D1.Bitmap LoadBitmap(string imageFileName)

{

BitmapDecoder decoder = new BitmapDecoder(imageFactory, imageFileName, DecodeOptions.CacheOnDemand);

BitmapFrameDecode frame = decoder.GetFrame(0);

FormatConverter converter = new FormatConverter(imageFactory);

converter.Initialize(frame, WIC.PixelFormat.Format32bppPRGBA, BitmapDitherType.Ordered4x4, null, 0.0, BitmapPaletteType.Custom);

Direct2D1.Bitmap bitmap = Direct2D1.Bitmap.FromWicBitmap(staticRenderTarget, converter);

Utilities.Dispose(ref converter);

Utilities.Dispose(ref frame);

Utilities.Dispose(ref decoder);

return bitmap;

}

/// <summary>

/// Загрузка последовательности изображений для анимации

/// </summary>

/// <param name="pathToFiles">Путь к файлам</param>

/// <param name="countOfBitmaps">Количество файлов для загрузки</param>

/// <returns>Список изображений</returns>

public static List<Direct2D1.Bitmap> LoadAnimation(string pathToFiles, int countOfBitmaps)

{

List<Direct2D1.Bitmap> bitmaps = new List<Direct2D1.Bitmap>();

for(int i = 1; i <= countOfBitmaps; i++)

{

string fileName = pathToFiles + i + ".png";

bitmaps.Add(LoadBitmap(fileName));

}

return bitmaps;

}

/// <summary>

/// Освобождение ресурсов

/// </summary>

public void Dispose()

{

Utilities.Dispose(ref renderTarget);

Utilities.Dispose(ref imageFactory);

Utilities.Dispose(ref renderTargetFactory);

}

}

}

**Код программы для *Scene*.*cs*:**

using System;

using System.Collections.Generic;

using EngineLibrary.ObjectComponents;

using SharpDX.Direct2D1;

namespace EngineLibrary.EngineComponents

{

/// <summary>

/// Абстрактный класс сцены

/// </summary>

public abstract class Scene

{

private WindowRenderTarget renderTarget;

private float worldScale;

/// <summary>

/// Событие окончания отрисовки сцены

/// </summary>

protected event Action OnSceneDrawed;

/// <summary>

/// Список текущих игровых объектов для отрисовки

/// </summary>

public List<GameObject> gameObjects = new List<GameObject>();

/// <summary>

/// Список игровых объектов для добалвения в список отрисовки

/// </summary>

protected List<GameObject> gameObjectsToAdd = new List<GameObject>();

/// <summary>

/// Список игровых объектов для удаления их списка отрисовки

/// </summary>

protected List<GameObject> gameObjectsToRemove = new List<GameObject>();

/// <summary>

/// Состояния отрисовки сцены

/// </summary>

public bool IsDrawScene = true;

/// <summary>

/// Инициализация сцены и игровых объектов

/// </summary>

/// <param name="target">Окно отрисовки</param>

/// <param name="scale">Относительный размер объектов</param>

public void InitializeScene(WindowRenderTarget target, float scale)

{

renderTarget = target;

worldScale = scale;

CreateGameObjectsOnScene();

InitializeEngineSettingsToGameObjects(gameObjects);

}

/// <summary>

/// Установка опций рендеринга игровым объектам

/// </summary>

private void InitializeEngineSettingsToGameObjects(List<GameObject> gameObjects)

{

foreach (GameObject gameObject in gameObjects)

gameObject.InitalizeEngineSettings(renderTarget, worldScale);

}

/// <summary>

/// Создание игровых объектов на сцене

/// </summary>

protected abstract void CreateGameObjectsOnScene();

/// <summary>

/// Отрисовка сцены (игровых объектов)

/// </summary>

public void DrawScene()

{

foreach (GameObject gameObject in gameObjects)

gameObject.Draw();

AddRenderGameObjects();

RemoveRenderGameObjects();

if (!IsDrawScene)

gameObjects.Clear();

OnSceneDrawed?.Invoke();

}

/// <summary>

/// Добавление игрового объекта для отрисовки

/// </summary>

private void AddRenderGameObjects()

{

InitializeEngineSettingsToGameObjects(gameObjectsToAdd);

gameObjects.AddRange(gameObjectsToAdd);

gameObjectsToAdd.Clear();

}

/// <summary>

/// Удаление игрового объекта из отрисовки

/// </summary>

private void RemoveRenderGameObjects()

{

foreach(GameObject removeGameObject in gameObjectsToRemove)

{

gameObjects.Remove(removeGameObject);

}

gameObjectsToRemove.Clear();

}

/// <summary>

/// Поведения при окончании сцены

/// </summary>

protected virtual void EndScene()

{

IsDrawScene = false;

}

}

}

**Код программы для *InputHandler*.*cs*:**

using System;

using SharpDX;

using SharpDX.DirectInput;

using SharpDX.Windows;

namespace EngineLibrary.EngineComponents

{

/// <summary>

/// Обработчик ввода с клавиатуры

/// </summary>

public class InputHandler : IDisposable

{

private DirectInput directInput;

private Keyboard keyboard;

/// <summary>

/// Состояние клавиатуры

/// </summary>

public KeyboardState KeyboardState { get => keyboardState; }

private KeyboardState keyboardState;

/// <summary>

/// Удалось ли обновить состоние клавиатуры

/// </summary>

public bool KeyboardUpdated { get => keyboardUpdated; }

private bool keyboardUpdated;

private bool keyboardAcquired;

/// <summary>

/// Конструктор класса, инциализирующий компоненты обработчика

/// </summary>

/// <param name="renderForm">Form отрисовки</param>

public InputHandler(RenderForm renderForm)

{

directInput = new DirectInput();

keyboard = new Keyboard(directInput);

keyboard.Properties.BufferSize = 16;

AcquireKeyboard();

keyboardState = new KeyboardState();

}

/// <summary>

/// Попытка получить доступ к клавиатуре

/// </summary>

private void AcquireKeyboard()

{

try

{

keyboard.Acquire();

keyboardAcquired = true;

}

catch (SharpDXException e)

{

if (e.ResultCode.Failure)

keyboardAcquired = false;

}

}

/// <summary>

/// Обновление состония клавиатуры

/// </summary>

public void UpdateKeyboardState()

{

if (!keyboardAcquired) AcquireKeyboard();

ResultDescriptor resultCode = ResultCode.Ok;

try

{

keyboard.GetCurrentState(ref keyboardState);

keyboardUpdated = true;

}

catch (SharpDXException e)

{

resultCode = e.Descriptor;

keyboardUpdated = false;

}

if (resultCode == ResultCode.InputLost || resultCode == ResultCode.NotAcquired)

keyboardAcquired = false;

}

/// <summary>

/// Освобождение ресурсов

/// </summary>

public void Dispose()

{

Utilities.Dispose(ref keyboard);

Utilities.Dispose(ref directInput);

}

}

}

**Код программы для *Input*.*cs*:**

using SharpDX.DirectInput;

namespace EngineLibrary.EngineComponents

{

/// <summary>

/// Класс, позволяющий управлять вводом с клавиатуры

/// </summary>

public static class Input

{

private static InputHandler inputHandler;

/// <summary>

/// Установка обработчика ввода с клавиатуры

/// </summary>

/// <param name="input">Обработчик ввода с клавиатуры</param>

public static void SetupInputHandler(InputHandler input)

{

inputHandler = input;

}

/// <summary>

/// Метод, возращающий значение ввода основных осей направления

/// </summary>

/// <param name="axis">Ось направления</param>

/// <returns>Положительное или отрицательное значение оси</returns>

public static int GetAxis(AxisOfInput axis)

{

if (inputHandler == null) return 0;

int move = 0;

if (inputHandler.KeyboardUpdated)

{

switch(axis)

{

case AxisOfInput.Horizontal:

if (inputHandler.KeyboardState.IsPressed(Key.D)) move++;

if (inputHandler.KeyboardState.IsPressed(Key.A)) move--;

break;

case AxisOfInput.Vertical:

if (inputHandler.KeyboardState.IsPressed(Key.W)) move++;

if (inputHandler.KeyboardState.IsPressed(Key.S)) move--;

break;

case AxisOfInput.AlternativeHorizontal:

if (inputHandler.KeyboardState.IsPressed(Key.Right)) move++;

if (inputHandler.KeyboardState.IsPressed(Key.Left)) move--;

break;

case AxisOfInput.AlternativeVertical:

if (inputHandler.KeyboardState.IsPressed(Key.Up)) move++;

if (inputHandler.KeyboardState.IsPressed(Key.Down)) move--;

break;

}

}

return move;

}

/// <summary>

/// Метод, возращающий реакцию на нажатие клавиши ввода

/// </summary>

/// <param name="key">Клавиша ввода</param>

/// <returns>Реакция true или false</returns>

public static bool GetButtonDawn(Key key)

{

if (inputHandler == null) return false;

if (inputHandler.KeyboardUpdated)

{

return inputHandler.KeyboardState.IsPressed(key);

}

return false;

}

}

/// <summary>

/// Ось направления ввода

/// </summary>

public enum AxisOfInput

{

/// <summary>

/// Горизонтальная ось

/// </summary>

Horizontal = 0,

/// <summary>

/// Вертикальная ось

/// </summary>

Vertical = 1,

/// <summary>

/// Альтернативная горизонтальная ось

/// </summary>

AlternativeHorizontal = 2,

/// <summary>

/// Альтернативная вертикальная ось

/// </summary>

AlternativeVertical = 3,

}

}

**Код программы для *Time*.*cs*:**

using System;

using System.Diagnostics;

namespace EngineLibrary.EngineComponents

{

/// <summary>

/// Статический класс управления временем

/// </summary>

public static class Time

{

private static Stopwatch watch;

private static long previousTicks;

/// <summary>

/// Текущее время с запуска приложения

/// </summary>

public static float CurrentTime { get; private set; }

/// <summary>

/// Разница во времени между кадрами

/// </summary>

public static float DeltaTime { get; private set; }

/// <summary>

/// Конструктори статического класса

/// </summary>

static Time()

{

watch = new Stopwatch();

Reset();

}

/// <summary>

/// Обновление подсчитанных значений

/// </summary>

public static void UpdateTime()

{

long ticks = watch.Elapsed.Ticks;

CurrentTime = (float)ticks / TimeSpan.TicksPerSecond;

DeltaTime = (float)(ticks - previousTicks) / TimeSpan.TicksPerSecond;

previousTicks = ticks;

}

/// <summary>

/// Сброс таймера и счетчика

/// </summary>

public static void Reset()

{

watch.Reset();

watch.Start();

previousTicks = watch.Elapsed.Ticks;

}

}

}

**Код программы для *GameObject*.*cs*:**

using SharpDX;

using SharpDX.Direct2D1;

namespace EngineLibrary.ObjectComponents

{

/// <summary>

/// Игровой объект

/// </summary>

public class GameObject

{

private WindowRenderTarget renderTarget;

private float worldScale;

/// <summary>

/// Родительский игровой объект

/// </summary>

public GameObject ParentGameObject { get; set; }

/// <summary>

/// Тэг игрового объекта

/// </summary>

public string GameObjectTag { get; set; }

/// <summary>

/// Активность игрового объекта

/// </summary>

public bool IsActive { get; set; } = true;

/// <summary>

/// Компонент перемещения

/// </summary>

public TransformComponent Transform { get; private set; }

/// <summary>

/// Компонент спрайта

/// </summary>

public SpriteComponent Sprite { get; private set; }

/// <summary>

/// Компонент твердого тела

/// </summary>

public ColliderComponent Collider { get; private set; }

/// <summary>

/// Сценарий выполения

/// </summary>

public ObjectScript Script { get; private set; }

/// <summary>

/// Инциализация опций рендеринга игрового объекта

/// </summary>

/// <param name="target">Окно отрисовки</param>

/// <param name="scale">Относительный размер</param>

internal void InitalizeEngineSettings(WindowRenderTarget target, float scale)

{

renderTarget = target;

worldScale = scale;

}

/// <summary>

/// Инциализация компонетов игрового объекта

/// </summary>

/// <param name="component">Компонент игровго объекта</param>

public void InitializeObjectComponent(object component)

{

switch(component)

{

case TransformComponent transformComponent:

Transform = transformComponent;

break;

case SpriteComponent spriteComponent:

Sprite = spriteComponent;

break;

case ColliderComponent colliderComponent:

Collider = colliderComponent;

break;

}

}

/// <summary>

/// Инициализация сценария поведения объекта

/// </summary>

/// <param name="objectScript">Сценарий игрового объекта</param>

public void InitializeObjectScript(ObjectScript objectScript)

{

Script = objectScript;

Script.Initialize(this);

Script.Start();

}

/// <summary>

/// Метод отрисовки игрового объекта

/// </summary>

public void Draw()

{

if (Collider != null)

Collider.IsInactive = !IsActive;

if (Script != null)

Script.Update();

if (!IsActive || (ParentGameObject != null && !ParentGameObject.IsActive)) return;

if (Sprite == null) return;

Vector2 translation = new Vector2(0,0);

if (ParentGameObject != null)

{

translation.X = ParentGameObject.Transform.Position.X \* ParentGameObject.Transform.Scale.Width;

translation.Y = ParentGameObject.Transform.Position.Y \* ParentGameObject.Transform.Scale.Height;

}

translation.X += Transform.Position.X \* Transform.Scale.Width;

translation.Y += Transform.Position.Y \* Transform.Scale.Height;

Matrix3x2 matrix = Matrix3x2.Rotation(0, translation);

matrix \*= Matrix3x2.Scaling(Transform.Scale.Width \* worldScale / Sprite.WidthOfSprite, Transform.Scale.Height \* worldScale / Sprite.HeightOfSprite, translation);

matrix \*= Matrix3x2.Translation(translation \* worldScale);

renderTarget.Transform = matrix;

Sprite.PlayAnimation();

if (Sprite.Bitmap != null)

renderTarget.DrawBitmap(Sprite.Bitmap, 1f, BitmapInterpolationMode.Linear);

}

}

}

**Код программы для *GameObject*.*cs*:**

using SharpDX;

using SharpDX.Direct2D1;

namespace EngineLibrary.ObjectComponents

{

/// <summary>

/// Игровой объект

/// </summary>

public class GameObject

{

private WindowRenderTarget renderTarget;

private float worldScale;

/// <summary>

/// Родительский игровой объект

/// </summary>

public GameObject ParentGameObject { get; set; }

/// <summary>

/// Тэг игрового объекта

/// </summary>

public string GameObjectTag { get; set; }

/// <summary>

/// Активность игрового объекта

/// </summary>

public bool IsActive { get; set; } = true;

/// <summary>

/// Компонент перемещения

/// </summary>

public TransformComponent Transform { get; private set; }

/// <summary>

/// Компонент спрайта

/// </summary>

public SpriteComponent Sprite { get; private set; }

/// <summary>

/// Компонент твердого тела

/// </summary>

public ColliderComponent Collider { get; private set; }

/// <summary>

/// Сценарий выполения

/// </summary>

public ObjectScript Script { get; private set; }

/// <summary>

/// Инциализация опций рендеринга игрового объекта

/// </summary>

/// <param name="target">Окно отрисовки</param>

/// <param name="scale">Относительный размер</param>

internal void InitalizeEngineSettings(WindowRenderTarget target, float scale)

{

renderTarget = target;

worldScale = scale;

}

/// <summary>

/// Инциализация компонетов игрового объекта

/// </summary>

/// <param name="component">Компонент игровго объекта</param>

public void InitializeObjectComponent(object component)

{

switch(component)

{

case TransformComponent transformComponent:

Transform = transformComponent;

break;

case SpriteComponent spriteComponent:

Sprite = spriteComponent;

break;

case ColliderComponent colliderComponent:

Collider = colliderComponent;

break;

}

}

/// <summary>

/// Инициализация сценария поведения объекта

/// </summary>

/// <param name="objectScript">Сценарий игрового объекта</param>

public void InitializeObjectScript(ObjectScript objectScript)

{

Script = objectScript;

Script.Initialize(this);

Script.Start();

}

/// <summary>

/// Метод отрисовки игрового объекта

/// </summary>

public void Draw()

{

if (Collider != null)

Collider.IsInactive = !IsActive;

if (Script != null)

Script.Update();

if (!IsActive || (ParentGameObject != null && !ParentGameObject.IsActive)) return;

if (Sprite == null) return;

Vector2 translation = new Vector2(0,0);

if (ParentGameObject != null)

{

translation.X = ParentGameObject.Transform.Position.X \* ParentGameObject.Transform.Scale.Width;

translation.Y = ParentGameObject.Transform.Position.Y \* ParentGameObject.Transform.Scale.Height;

}

translation.X += Transform.Position.X \* Transform.Scale.Width;

translation.Y += Transform.Position.Y \* Transform.Scale.Height;

Matrix3x2 matrix = Matrix3x2.Rotation(0, translation);

matrix \*= Matrix3x2.Scaling(Transform.Scale.Width \* worldScale / Sprite.WidthOfSprite, Transform.Scale.Height \* worldScale / Sprite.HeightOfSprite, translation);

matrix \*= Matrix3x2.Translation(translation \* worldScale);

renderTarget.Transform = matrix;

Sprite.PlayAnimation();

if (Sprite.Bitmap != null)

renderTarget.DrawBitmap(Sprite.Bitmap, 1f, BitmapInterpolationMode.Linear);

}

}

}

**Код программы для *TransformComponent*.*cs*:**

using SharpDX;

namespace EngineLibrary.ObjectComponents

{

/// <summary>

/// Компонент перемещения игрового объекта

/// </summary>

public class TransformComponent

{

private const float accelerationOfGravity = 0.0981f;

/// <summary>

/// Позиция игрового объекта

/// </summary>

public Vector2 Position { get => position; set => position = value; }

private Vector2 position;

/// <summary>

/// Размер игрового объекта

/// </summary>

public Size2F Scale { get => scale; set => scale = value; }

private Size2F scale;

/// <summary>

/// Использование гравитации

/// </summary>

public bool IsUseGravitation { get; set; } = true;

private Vector2 movementInCurrentFrame;

/// <summary>

/// Конструктор компонента

/// </summary>

/// <param name="position">начальная позиция</param>

/// <param name="scale">Начальный размер</param>

public TransformComponent(Vector2 position, Size2F scale)

{

movementInCurrentFrame = Vector2.Zero;

this.position = position;

this.scale = scale;

}

/// <summary>

/// Перемещение объкта

/// </summary>

/// <param name="movement">Вектор перемещения</param>

public void SetMovement(Vector2 movement)

{

movementInCurrentFrame = movement;

position.X += movement.X;

position.Y -= movement.Y;

}

/// <summary>

/// Возврат позиция в этом кадре

/// </summary>

public void ResetMovement()

{

position.X -= movementInCurrentFrame.X;

if(!IsUseGravitation)

position.Y += movementInCurrentFrame.Y;

}

/// <summary>

/// Добавление силы тяжести

/// </summary>

public void AddGravitation()

{

if(IsUseGravitation)

position.Y += accelerationOfGravity;

}

/// <summary>

/// Возрат силы тяжести

/// </summary>

public void ResetGravitation()

{

if (IsUseGravitation)

position.Y -= accelerationOfGravity;

}

}

}

**Код программы для *SpriteComponent*.*cs*:**

using System.Collections.Generic;

using SharpDX.Direct2D1;

namespace EngineLibrary.ObjectComponents

{

/// <summary>

/// Класс спрайта игровйо объекта

/// </summary>

public class SpriteComponent

{

/// <summary>

/// Изображение игрового объекта

/// </summary>

public Bitmap Bitmap { get; set; }

/// <summary>

/// Ширина спрайта

/// </summary>

public float WidthOfSprite { get; private set; }

/// <summary>

/// Высота спрайта

/// </summary>

public float HeightOfSprite { get; private set; }

/// <summary>

/// Инверсия по оси X

/// </summary>

public bool IsFlipX { get; set; }

private Dictionary<string, Animation> animations;

private string currentAnimation = "";

/// <summary>

/// Конструктор класса спрайта

/// </summary>

/// <param name="sprite">Изображение</param>

public SpriteComponent(Bitmap sprite)

{

Bitmap = sprite;

WidthOfSprite = sprite.Size.Width;

HeightOfSprite = sprite.Size.Height;

animations = new Dictionary<string, Animation>();

animations.Add("inactive", new Animation(null, 1f, true));

}

/// <summary>

/// Добавление анимации

/// </summary>

/// <param name="name">Ключ анимации</param>

/// <param name="animation">Анимация</param>

public void AddAnimation(string name, Animation animation)

{

animations.Add(name, animation);

}

/// <summary>

/// Проигрывание анимации

/// </summary>

public void PlayAnimation()

{

if (currentAnimation == "") return;

Bitmap = animations[currentAnimation].GetSpriteFromAnimation();

}

/// <summary>

/// Установка текущей анимации

/// </summary>

/// <param name="name">Ключ анимации</param>

public void SetAnimation(string name)

{

if (currentAnimation == name) return;

if(currentAnimation != "")

animations[currentAnimation].ResetAnimation();

currentAnimation = name;

}

}

}

**Код программы для *Animation*.*cs*:**

using System.Collections.Generic;

using EngineLibrary.EngineComponents;

using SharpDX.Direct2D1;

namespace EngineLibrary.ObjectComponents

{

/// <summary>

/// Класс анимации

/// </summary>

public class Animation

{

private List<Bitmap> sprites;

private int currentIndexInAnimation;

private float changeTime;

private float deltaTimeAnimation;

private bool isLoop;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="sprites">Список спрайтов</param>

/// <param name="changeTime">Время смены каждого спрайта</param>

/// <param name="isLoop">Зацикленностб анимации</param>

public Animation(List<Bitmap> sprites, float changeTime, bool isLoop)

{

this.sprites = sprites;

this.changeTime = changeTime;

this.isLoop = isLoop;

deltaTimeAnimation = changeTime;

currentIndexInAnimation = 0;

}

/// <summary>

/// Сброс анимации

/// </summary>

public void ResetAnimation()

{

currentIndexInAnimation = 0;

}

/// <summary>

/// Возращает текущее изображение анимации

/// </summary>

/// <returns>Текущее изображение в анимации</returns>

public Bitmap GetSpriteFromAnimation()

{

if (sprites == null)

return null;

if (changeTime < Time.CurrentTime)

{

currentIndexInAnimation++;

changeTime = Time.CurrentTime + deltaTimeAnimation;

}

if (currentIndexInAnimation >= sprites.Count)

if (!isLoop)

return null;

else

currentIndexInAnimation = 0;

return sprites[currentIndexInAnimation];

}

}

}

**Код программы для *ColliderComponent*.*cs*:**

using SharpDX;

using System.Collections.Generic;

namespace EngineLibrary.ObjectComponents

{

/// <summary>

/// Класс компонента, описывающий твердое тело

/// </summary>

public class ColliderComponent

{

private static List<GameObject> collidersOfGameObjects;

private Vector2[] boundCorners;

private GameObject gameObject;

private Size2F colliderScale;

private Vector2 offsetCollider;

/// <summary>

/// Неактивность элемента

/// </summary>

public bool IsInactive { get; set; }

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="gameObject">Игровой объект, которому принадлежит компонент</param>

/// <param name="scale">Размер коллайдера</param>

/// <param name="offset">Cмещения коллайдера от центра</param>

public ColliderComponent(GameObject gameObject, Size2F scale, Vector2 offset = new Vector2())

{

if (collidersOfGameObjects == null)

collidersOfGameObjects = new List<GameObject>();

collidersOfGameObjects.Add(gameObject);

this.gameObject = gameObject;

boundCorners = new Vector2[4];

colliderScale = scale;

offsetCollider = offset;

IsInactive = false;

}

/// <summary>

/// Обновление границ твердого тела

/// </summary>

private void UpdateBounds()

{

Vector2 position = gameObject.Transform.Position;

float offsetWidth = colliderScale.Width / 2;

float offsetHeight = colliderScale.Height / 2;

boundCorners[0] = new Vector2(position.X + offsetCollider.X - offsetWidth, position.Y + offsetCollider.Y - offsetHeight);

boundCorners[1] = new Vector2(position.X + offsetCollider.X - offsetWidth, position.Y + offsetCollider.Y + offsetHeight);

boundCorners[2] = new Vector2(position.X + offsetCollider.X + offsetWidth, position.Y + offsetCollider.Y + offsetHeight);

boundCorners[3] = new Vector2(position.X + offsetCollider.X + offsetWidth, position.Y + offsetCollider.Y - offsetHeight);

}

/// <summary>

/// Проверка на пересечние компонента твердого тела с другими компонентами твердого тела, имеющие тег у игрового объекта

/// </summary>

/// <param name="tagNames">Теги игровых объектов, с которыми ожидается столкновение</param>

/// <returns>Реакция на проверку</returns>

public bool CheckIntersection(params string[] tagNames)

{

foreach (GameObject otherGameObject in collidersOfGameObjects)

{

if (otherGameObject == gameObject || otherGameObject.Collider.IsInactive) continue;

bool hasTag = false;

for (int i = 0; i < tagNames.Length && !hasTag; i++)

{

hasTag = otherGameObject.GameObjectTag == tagNames[i];

}

if(hasTag)

{

if (CheckGameObjectIntersection(otherGameObject))

{

return true;

}

}

}

return false;

}

/// <summary>

/// Проверка на пересечние компонента твердого тела с другими компонентами твердого тела, имеющие тег у игрового объекта

/// </summary>

/// <param name="intersecredGameObject">Пересаемый объект</param>

/// <param name="tagNames">Теги игровых объектов, с которыми ожидается столкновение</param>

/// <returns>Реакция на проверку</returns>

public bool CheckIntersection(out GameObject intersecredGameObject,params string[] tagNames)

{

foreach (GameObject otherGameObject in collidersOfGameObjects)

{

if (otherGameObject == gameObject || otherGameObject.Collider.IsInactive) continue;

bool hasTag = false;

for (int i = 0; i < tagNames.Length && !hasTag; i++)

{

hasTag = otherGameObject.GameObjectTag == tagNames[i];

}

if (hasTag)

{

if (CheckGameObjectIntersection(otherGameObject))

{

intersecredGameObject = otherGameObject;

return true;

}

}

}

intersecredGameObject = null;

return false;

}

/// <summary>

/// Проверка на пересечение компонента твердого тела с другими компонентами твердого тела, имеющие конкретный сценарий выполнения T

/// </summary>

/// <typeparam name="T">Конкретный сценарий выполнения</typeparam>

/// <param name="objectScript">Сценарий выполнения игрового объекта, с которым столкнулся</param>

/// <returns>Реакция на столкновение</returns>

public bool CheckIntersection<T>(out T objectScript)

where T : ObjectScript

{

foreach (GameObject otherGameObject in collidersOfGameObjects)

{

if (otherGameObject == gameObject || otherGameObject.Script == null || otherGameObject.Collider.IsInactive) continue;

if (otherGameObject.Script is T)

{

if (CheckGameObjectIntersection(otherGameObject))

{

objectScript = (T)otherGameObject.Script;

return true;

}

}

}

objectScript = null;

return false;

}

/// <summary>

/// Проверка на пересечение компонента твердого тела с другим компонентам твердого тела

/// </summary>

/// <param name="otherGameObject">Игровой объект с компонетом твердого тела</param>

/// <returns>Реакция на столкновение</returns>

private bool CheckGameObjectIntersection(GameObject otherGameObject)

{

UpdateBounds();

otherGameObject.Collider.UpdateBounds();

ColliderComponent collider = otherGameObject.Collider;

int count = boundCorners.Length + collider.boundCorners.Length;

Vector2[] allCorners = new Vector2[count];

boundCorners.CopyTo(allCorners, 0);

collider.boundCorners.CopyTo(allCorners, boundCorners.Length);

Vector2 normal;

bool isInteresect = false;

for (int i = 0; i < count && !isInteresect; i++)

{

normal = GetNormal(allCorners, i);

Vector2 currentProjection = GetProjection(normal);

Vector2 otherProjection = collider.GetProjection(normal);

if (currentProjection.X < otherProjection.Y || otherProjection.X < currentProjection.Y)

{

return false;

}

}

return true;

}

/// <summary>

/// Создание нормали

/// </summary>

/// <param name="corners">Углы двух компонетов</param>

/// <param name="index">Номер угла</param>

/// <returns>Нормаль</returns>

private Vector2 GetNormal(Vector2[] corners, int index)

{

int next = index + 1;

next = next == corners.Length ? 0 : next;

Vector2 firstPoint = corners[index];

Vector2 secondPoint = corners[next];

Vector2 edge = new Vector2(secondPoint.X - firstPoint.X, secondPoint.Y - firstPoint.Y);

return new Vector2(-edge.Y, edge.X);

}

/// <summary>

/// Создание проекции

/// </summary>

/// <param name="normal">Нормаль</param>

/// <returns>Проекцию</returns>

private Vector2 GetProjection(Vector2 normal)

{

Vector2 result = new Vector2();

bool isNull = true;

foreach(Vector2 current in boundCorners)

{

float projection = normal.X \* current.X + normal.Y \* current.Y;

if(isNull)

{

result = new Vector2(projection, projection);

isNull = false;

}

if (projection > result.X)

result.X = projection;

if (projection < result.Y)

result.Y = projection;

}

return result;

}

}

}

**Код программы для *ObjectScript*.*cs*:**

namespace EngineLibrary.ObjectComponents

{

/// <summary>

/// Абстрактный класс сценария поведения игрового объекта

/// </summary>

public abstract class ObjectScript

{

/// <summary>

/// Игровой объект, которым управляет сценарий

/// </summary>

protected GameObject gameObject;

/// <summary>

/// Инициализация сценария

/// </summary>

/// <param name="gameObject">Игровой объект, который выполняет сценарий</param>

public void Initialize(GameObject gameObject)

{

this.gameObject = gameObject;

}

/// <summary>

/// Поведение на момент создание игрового объекта

/// </summary>

public abstract void Start();

/// <summary>

/// Обновление игрового объекта

/// </summary>

public abstract void Update();

}

}

**Код программы для *MazeScene*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using SharpDX;

using System;

using System.Drawing;

using System.Collections.Generic;

using GameLibrary.Game;

using GameLibrary.Maze.Network;

using NetworkLibrary;

using SharpDX.DirectInput;

namespace GameLibrary.Maze

{

/// <summary>

/// Класс лабиринта

/// </summary>

public class MazeScene : Scene

{

/// <summary>

/// Рандом, используемый в игре

/// </summary>

public static Random GameRandom { get; private set; }

/// <summary>

/// Статическая ссылка на класс

/// </summary>

public static MazeScene instance = null;

/// <summary>

/// Фабрика создания элементов лабиринта

/// </summary>

public MazeElementsFactory ElementsFactory { get; private set; }

/// <summary>

/// Конструктор синего игрока

/// </summary>

public PlayerConstructor FirstPlayerConstructor { get; private set; }

/// <summary>

/// Конструктор красного игрока

/// </summary>

public PlayerConstructor SecondPlayerConstructor { get; private set; }

private NetworkManager networkManager;

private List<Vector2> emptyBlocks = new List<Vector2>();

private int countOfcoins = 0;

/// <summary>

/// Конструктор класса для создания игры на одном экране

/// </summary>

public MazeScene()

{

GameRandom = new Random();

FirstPlayerConstructor = new PlayerConstructor("Blue Player",

new Player(new PlayerControl(AxisOfInput.Horizontal, AxisOfInput.Vertical, Key.Space)));

SecondPlayerConstructor = new PlayerConstructor("Red Player",

new Player(new PlayerControl(AxisOfInput.AlternativeHorizontal, AxisOfInput.AlternativeVertical,

Key.NumberPadEnter)));

}

/// <summary>

/// Конструктор класса для создания игры по сети

/// </summary>

/// <param name="handler">Обработчик сетевого взаимодействия</param>

/// <param name="seed">Сид генерации</param>

/// <param name="hostPlayer">Тег игрока, играющего на этом компьютере</param>

/// <param name="networkPlayer">Тег игрока, играющего по интернету</param>

public MazeScene(INetworkHandler handler, int seed, string hostPlayer, string networkPlayer)

{

GameRandom = new Random(seed);

var playerScript = new Player(new PlayerControl(AxisOfInput.Horizontal, AxisOfInput.Vertical, Key.Space));

var networkPlayerScript = new NetworkPlayer();

if (hostPlayer == "Blue Player")

{

FirstPlayerConstructor = new PlayerConstructor(hostPlayer, playerScript);

SecondPlayerConstructor = new PlayerConstructor(networkPlayer, networkPlayerScript);

}

else

{

SecondPlayerConstructor = new PlayerConstructor(hostPlayer, playerScript);

FirstPlayerConstructor = new PlayerConstructor(networkPlayer, networkPlayerScript);

}

networkManager = new NetworkManager(handler);

networkManager.OnWriteData += playerScript.WriteNetworkData;

networkManager.OnUpdateData += networkPlayerScript.UpdateNetworkData;

OnSceneDrawed += networkManager.UpdateData;

}

/// <summary>

/// Обновление пули в сети

/// </summary>

/// <param name="data">Сетевые данные о пули</param>

public void UpdateBullet(BulletNetworkData data)

{

networkManager.CurrentPlayerNetworkData.BulletData = data;

}

/// <summary>

/// Создание игровых объектов на сцене

/// </summary>

protected override void CreateGameObjectsOnScene()

{

if (instance == null)

instance = this;

ElementsFactory = new MazeElementsFactory();

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(new Vector2(0f, 0f), new Size2F(27, 15)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/Фон.png")));

gameObject.GameObjectTag = "Background";

gameObjects.Add(gameObject);

gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(new Vector2(0f, 0f), new Size2F(1, 1)));

gameObject.GameObjectTag = "GameManager";

gameObject.InitializeObjectScript(new SpawnManager());

gameObjects.Add(gameObject);

CreateMaze();

GameObject player = FirstPlayerConstructor.CreatePlayer();

gameObjects.Add(player);

gameObjects.Add(FirstPlayerConstructor.CreateArms());

player = SecondPlayerConstructor.CreatePlayer();

gameObjects.Add(player);

gameObjects.Add(SecondPlayerConstructor.CreateArms());

}

/// <summary>

/// Метод создания лабиринта

/// </summary>

public void CreateMaze()

{

string text = "Resources/Mazes/Maze " + GameRandom.Next(1, 6) + ".bmp";

Bitmap bitmap = new Bitmap(text);

for (int i = 0; i < bitmap.Height; i++)

{

for (int j = 0; j < bitmap.Width; j++)

{

System.Drawing.Color color = bitmap.GetPixel(j, i);

GameObject gameObject = null;

if (color.R == 0 && color.G == 0 && color.B == 0)

gameObject = ElementsFactory.CreateMazeElement(new Vector2(j, i), "Wall");

else if (color.R == 255 && color.G == 0 && color.B == 0)

gameObject = ElementsFactory.CreateMazeElement(new Vector2(j, i), "BreakWall");

else if (color.R == 0 && color.G == 255 && color.B == 0)

gameObject = ElementsFactory.CreateMazeElement(new Vector2(j, i), "Platform");

else if (color.R == 0 && color.G == 0 && color.B == 255)

gameObject = ElementsFactory.CreateMazeElement(new Vector2(j, i), "Stair");

else if (color.R == 255 && color.G == 255 && color.B == 0)

{

gameObject = ElementsFactory.CreateCoin(new Vector2(j, i));

countOfcoins++;

}

else if (color.R == 125 && color.G == 0 && color.B == 0)

SecondPlayerConstructor.StartPosition = new Vector2(j, i);

else if (color.R == 0 && color.G == 0 && color.B == 125)

FirstPlayerConstructor.StartPosition = new Vector2(j, i);

else

emptyBlocks.Add(new Vector2(j, i));

if (gameObject != null)

gameObjects.Add(gameObject);

}

}

}

/// <summary>

/// Добавление объекта в лист отрисовки

/// </summary>

/// <param name="gameObject">Игровой объект</param>

public void AddObjectOnScene(GameObject gameObject)

{

gameObjectsToAdd.Add(gameObject);

}

/// <summary>

/// Удаления объекта из листа отрисовки

/// </summary>

/// <param name="gameObject">Игровой объект</param>

public void RemoveObjectFromScene(GameObject gameObject)

{

if (gameObject.GameObjectTag == "Spawn")

emptyBlocks.Add(gameObject.Transform.Position);

gameObjectsToRemove.Add(gameObject);

if (gameObject.GameObjectTag == "Coin")

{

countOfcoins--;

if(countOfcoins == 0)

EndScene();

}

}

/// <summary>

/// Рандомное место в лабиринте

/// </summary>

/// <returns>Позицию</returns>

public Vector2 GetRandomPosition()

{

int index = GameRandom.Next(0, emptyBlocks.Count);

Vector2 position = emptyBlocks[index];

emptyBlocks.Remove(position);

return position;

}

/// <summary>

/// Поведение при завершении сцены

/// </summary>

protected override void EndScene()

{

base.EndScene();

string winPlayer = "";

int firstPlayerCountCoins = ((BasePlayer) FirstPlayerConstructor.PlayerGameObject.Script).Coins;

int secondPlayerCountCoins = ((BasePlayer) SecondPlayerConstructor.PlayerGameObject.Script).Coins;

if(firstPlayerCountCoins < secondPlayerCountCoins)

winPlayer = SecondPlayerConstructor.PlayerTag;

else

winPlayer = FirstPlayerConstructor.PlayerTag;

GameEvents.EndGame?.Invoke(winPlayer);

}

}

}

**Код программы для *MazeElementsFactory*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using SharpDX;

namespace GameLibrary.Maze

{

/// <summary>

/// Класс фабрики создания элементов лабиринта

/// </summary>

public class MazeElementsFactory

{

/// <summary>

/// Создает элемент лабиринта

/// </summary>

/// <param name="position">Позиция объекта на сцене</param>

/// <param name="TagName">Тег игрового объекта</param>

/// <returns>Созданный игровой объект</returns>

public GameObject CreateMazeElement(Vector2 position, string TagName)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(position, new Size2F(1f, 1f)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/MazeElements/" + TagName + ".png")));

if (TagName == "Platform")

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(1f, 0.1f), new Vector2(0, -0.5f)));

else

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(1f, 1f)));

gameObject.GameObjectTag = TagName;

if (TagName == "BreakWall")

gameObject.InitializeObjectScript(new BreakWall());

return gameObject;

}

/// <summary>

/// Создание монет в лабиринте

/// </summary>

/// <param name="position">Позиция объекта на сцене</param>

/// <returns>Игровой объект</returns>

public GameObject CreateCoin(Vector2 position)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(position, new Size2F(1f, 1f)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/MazeElements/Treasure.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(1f, 1f)));

gameObject.GameObjectTag = "Coin";

gameObject.InitializeObjectScript(new Coin());

return gameObject;

}

}

}

**Код программы для *SpawnManager*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Guns.SpawnFactories;

using SharpDX;

using System;

namespace GameLibrary.Maze

{

/// <summary>

/// Класс игрового менеджера

/// </summary>

public class SpawnManager : ObjectScript

{

private const float timeToSpawn = 5f;

private float currentTimeToSpawn;

private MazeScene maze;

private GunSpawnFactory spawnFactory;

/// <summary>

/// Поведение на момент создание игрового объекта

/// </summary>

public override void Start()

{

maze = MazeScene.instance;

currentTimeToSpawn = Time.CurrentTime;

}

/// <summary>

/// Обновление игрового объекта

/// </summary>

public override void Update()

{

if (currentTimeToSpawn < Time.CurrentTime)

{

Random random = new Random();

int chance = random.Next(0, 101);

Vector2 position = maze.GetRandomPosition();

if (chance <= 20)

{

spawnFactory = new DamageGunSpawnFactory();

}

else if (chance > 20 && chance <= 50)

{

spawnFactory = new FrezzeGunSpawnFactory();

}

else

{

spawnFactory = new SlowdownGunSpawnFactory();

}

maze.AddObjectOnScene(spawnFactory.CreateGunSpawn(position));

currentTimeToSpawn += timeToSpawn;

}

}

}

}

**Код программы для *BreakWall*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

namespace GameLibrary.Maze

{

/// <summary>

/// Класс ломающейся стены

/// </summary>

public class BreakWall : ObjectScript

{

private const float timeToRespawn = 2f;

private float respawnTime;

/// <summary>

/// Поведение на момент создание игрового объекта

/// </summary>

public override void Start()

{

Animation animation = new Animation(RenderingSystem.LoadAnimation("Resources/MazeElements/BreakWall ", 1), 10f, true);

gameObject.Sprite.AddAnimation("idleWall", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/MazeElements/BreakWall ", 3), 0.2f, false);

gameObject.Sprite.AddAnimation("breakWall", animation);

gameObject.Sprite.SetAnimation("idleWall");

}

/// <summary>

/// Обновление игрового объекта

/// </summary>

public override void Update()

{

if(gameObject.Sprite.Bitmap == null && gameObject.IsActive)

{

gameObject.IsActive = false;

respawnTime = Time.CurrentTime + timeToRespawn;

}

if(!gameObject.IsActive)

{

if(respawnTime < Time.CurrentTime)

{

gameObject.IsActive = true;

if (!gameObject.Collider.CheckIntersection("Blue Player","Red Player"))

{

gameObject.Sprite.SetAnimation("idleWall");

}

else

{

gameObject.IsActive = false;

}

}

}

}

/// <summary>

/// Уничтожении стены

/// </summary>

public void DestroyWall()

{

gameObject.Sprite.SetAnimation("breakWall");

}

}

}

**Код программы для *Coin*.*cs*:**

using EngineLibrary.ObjectComponents;

using GameLibrary.Game;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GameLibrary.Maze

{

/// <summary>

/// Класс монеты

/// </summary>

public class Coin : ObjectScript

{

private MazeScene maze;

/// <summary>

/// Поведение на момент создание игрового объекта

/// </summary>

public override void Start()

{

maze = MazeScene.instance;

}

/// <summary>

/// Обновление игрового объекта

/// </summary>

public override void Update()

{

if(gameObject.Collider.CheckIntersection(out Player player))

{

player.ChangeCoinsValue(1);

maze.RemoveObjectFromScene(gameObject);

}

}

}

}

**Код программы для *GunSpawn*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Guns;

namespace GameLibrary.Maze

{

/// <summary>

/// Класс подбираемого оружия в лабиринте

/// </summary>

public class GunSpawn : ObjectScript

{

private MazeScene maze;

private Gun dropOutGun;

private float cuurentTimeToDisappear;

private string nameOfgun;

/// <summary>

/// Инициализация места подбираемого оружия

/// </summary>

/// <param name="name">Название оружия</param>

/// <param name="gun">Подбираемое оружие</param>

/// <param name="disappearTime">Время исчезнование места</param>

public void InitializeGunSpawn(string name, Gun gun, float disappearTime)

{

dropOutGun = gun;

nameOfgun = name;

cuurentTimeToDisappear = Time.CurrentTime + disappearTime;

}

/// <summary>

/// Поведение на момент создание игрового объекта

/// </summary>

public override void Start()

{

maze = MazeScene.instance;

Animation animation = new Animation(RenderingSystem.LoadAnimation("Resources/MazeElements/Spawn Guns/" + nameOfgun + " spawn ", 2), 0.3f, true);

gameObject.Sprite.AddAnimation("idle", animation);

gameObject.Sprite.SetAnimation("idle");

}

/// <summary>

/// Обновление игрового объекта

/// </summary>

public override void Update()

{

if(cuurentTimeToDisappear < Time.CurrentTime)

{

maze.RemoveObjectFromScene(gameObject);

}

if(gameObject.Collider.CheckIntersection(out GameObject player,"Blue Player","Red Player"))

{

if(player.GameObjectTag == "Blue Player")

maze.AddObjectOnScene(maze.BluePlayerFactory.CreateGun(dropOutGun, nameOfgun));

else if (player.GameObjectTag == "Red Player")

maze.AddObjectOnScene(maze.RedPlayerFactory.CreateGun(dropOutGun, nameOfgun));

maze.RemoveObjectFromScene(gameObject);

}

}

}

}

**Код программы для *BasePlayer*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Maze;

using GameLibrary.Maze.Network;

namespace GameLibrary.Game

{

/// <summary>

/// Базовый класс игрока

/// </summary>

public class BasePlayer : ObjectScript

{

/// <summary>

/// Собарнные монеты

/// </summary>

public int Coins { get; private set; } = 0;

protected bool isMove;

private GameObject childGameObject;

/// <summary>

/// Изменение значени монет

/// </summary>

/// <param name="value">Значение, которое прибавляется к текущему значению монет</param>

public void ChangeCoinsValue(int value)

{

Coins += value;

GameEvents.ChangeCoins?.Invoke(gameObject.GameObjectTag, Coins);

}

/// <summary>

/// Установка дочернего объекта

/// </summary>

/// <param name="gameObject">Дочерний объект</param>

public void SetChildGameObject(GameObject gameObject)

{

if (childGameObject != null)

MazeScene.instance.RemoveObjectFromScene(childGameObject);

childGameObject = gameObject;

}

/// <summary>

/// Запись сетевых данных об игроке

/// </summary>

/// <param name="manager">Менеджер сетевого взаимодействия</param>

public void WriteNetworkData(NetworkManager manager)

{

manager.CurrentPlayerNetworkData.PlayerPosition = gameObject.Transform.Position;

manager.CurrentPlayerNetworkData.IsPlayerSpriteFlip = gameObject.Sprite.IsFlipX;

manager.CurrentPlayerNetworkData.IsPlayerMove = isMove;

}

/// <summary>

/// Обновление данные сетевого игрока

/// </summary>

/// <param name="manager">Менеджер сетевого взаимодействия</param>

public void UpdateNetworkData(NetworkManager manager)

{

gameObject.Transform.Position = manager.NetworkPlayerNetworkData.PlayerPosition;

gameObject.Sprite.IsFlipX = manager.NetworkPlayerNetworkData.IsPlayerSpriteFlip;

isMove = manager.NetworkPlayerNetworkData.IsPlayerMove;

UpdatePlayerFlip();

}

/// <summary>

/// Поведение на момент создание игрового объекта

/// </summary>

public override void Start()

{

Animation animation;

if (gameObject.GameObjectTag == "Blue Player")

{

animation = new Animation(RenderingSystem.LoadAnimation("Resources/Blue Player/left idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/Blue Player/left run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/Blue Player/right idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleRight", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/Blue Player/right run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runRight", animation);

}

else

{

animation = new Animation(RenderingSystem.LoadAnimation("Resources/Red Player/left idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/Red Player/left run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/Red Player/right idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleRight", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/Red Player/right run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runRight", animation);

}

gameObject.Sprite.SetAnimation("idleLeft");

}

/// <summary>

/// Обновление игрового объекта

/// </summary>

public override void Update()

{

UpdatePlayerFlip();

}

/// <summary>

/// Обновление вращения игрока

/// </summary>

private void UpdatePlayerFlip()

{

if (isMove)

{

if (childGameObject != null)

childGameObject.Sprite.IsFlipX = gameObject.Sprite.IsFlipX;

if (childGameObject != null && childGameObject.Sprite.IsFlipX)

childGameObject.Sprite.SetAnimation("runLeft");

else if (childGameObject != null)

childGameObject.Sprite.SetAnimation("runRight");

if (gameObject.Sprite.IsFlipX)

gameObject.Sprite.SetAnimation("runLeft");

else

gameObject.Sprite.SetAnimation("runRight");

}

else

{

if(childGameObject != null)

childGameObject.Sprite.IsFlipX = gameObject.Sprite.IsFlipX;

if (childGameObject != null && childGameObject.Sprite.IsFlipX)

childGameObject.Sprite.SetAnimation("idleLeft");

else if (childGameObject != null)

childGameObject.Sprite.SetAnimation("idleRight");

if (gameObject.Sprite.IsFlipX)

gameObject.Sprite.SetAnimation("idleLeft");

else

gameObject.Sprite.SetAnimation("idleRight");

}

}

}

}

**Код программы для *Player*.*cs*:**

using EngineLibrary.EngineComponents;

using SharpDX;

namespace GameLibrary.Game

{

/// <summary>

/// Класс, описывающий сценарий поведения игрока

/// </summary>

public class Player : BasePlayer

{

/// <summary>

/// Управление игрока

/// </summary>

public PlayerControl Control { get; private set; }

/// <summary>

/// Скорость игрока

/// </summary>

public float Speed { get; set; } = 5;

/// <summary>

/// Возможность двигаться у игрока

/// </summary>

public bool IsCanMove { get; set; } = true;

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="control">Управления игрока</param>

public Player(PlayerControl control)

{

Control = control;

}

/// <summary>

/// Обновление игрового объекта

/// </summary>

public override void Update()

{

if (gameObject.IsActive && IsCanMove)

Move();

base.Update();

}

/// <summary>

/// Метод движения игрока

/// </summary>

private void Move()

{

int directionX = 0, directionY = 0;

directionX = Input.GetAxis(Control.HorizontalAxis);

if (gameObject.Collider.CheckIntersection("Stair"))

{

directionY = Input.GetAxis(Control.VerticalAxis);

gameObject.Transform.IsUseGravitation = gameObject.Collider.CheckIntersection("Wall");

}

else

{

gameObject.Transform.IsUseGravitation = true;

}

Vector2 direction;

if (directionX == 0)

{

direction = new Vector2(0, directionY);

}

else

{

gameObject.Sprite.IsFlipX = directionX < 0;

direction = new Vector2(directionX, 0);

}

isMove = directionX != 0;

Vector2 movement = direction \* Speed \* Time.DeltaTime;

gameObject.Transform.SetMovement(movement);

DetectCollision();

}

/// <summary>

/// Распознавание столкновений и реакция на них

/// </summary>

private void DetectCollision()

{

if (gameObject.Collider.CheckIntersection("Wall","BreakWall"))

{

gameObject.Transform.ResetMovement();

}

if (gameObject.GameObjectTag == "Blue Player" || gameObject.GameObjectTag == "Red Player")

gameObject.Transform.AddGravitation();

string tag = (Input.GetAxis(Control.VerticalAxis) == -1) ? "" : "Platform";

if (gameObject.Collider.CheckIntersection("Wall", tag))

{

gameObject.Transform.ResetGravitation();

}

}

}

}

**Код программы для *NetworkPlayer*.*cs*:**

namespace GameLibrary.Game

{

/// <summary>

/// Класс, описывающий сценарий поведения игрока, играющего по интернету

/// </summary>

public class NetworkPlayer : BasePlayer

{

}

}

**Код программы для *PlayerControl*.*cs*:**

using EngineLibrary.EngineComponents;

using SharpDX.DirectInput;

namespace GameLibrary.Game

{

/// <summary>

/// Структура игрового управления персонажа

/// </summary>

public struct PlayerControl

{

/// <summary>

/// Горизонтальная ось передвижения

/// </summary>

public AxisOfInput HorizontalAxis { get; private set; }

/// <summary>

/// Вертикальная ось передвижения

/// </summary>

public AxisOfInput VerticalAxis { get; private set; }

/// <summary>

/// Кнопка стрельбы

/// </summary>

public Key ShootKey { get; private set; }

/// <summary>

/// Конструктор структуры

/// </summary>

/// <param name="horizontalAxis">Горизонтальная ось передвижения</param>

/// <param name="verticalAxis"> Вертикальная ось передвижения</param>

/// <param name="shootKey">Кнопка стрельбы</param>

public PlayerControl(AxisOfInput horizontalAxis, AxisOfInput verticalAxis, Key shootKey)

{

HorizontalAxis = horizontalAxis;

VerticalAxis = verticalAxis;

ShootKey = shootKey;

}

}

}

**Код программы для *PlayerConstructor*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Guns;

using SharpDX;

namespace GameLibrary.Game

{

/// <summary>

/// Класс фабрики создания персонажа

/// </summary>

public class PlayerConstructor

{

/// <summary>

/// Игровой Объект игрока

/// </summary>

public GameObject PlayerGameObject { get; private set; }

/// <summary>

/// Тег плеера

/// </summary>

public string PlayerTag { get; private set; }

/// <summary>

/// Начальная позиция игрока в лабиринте

/// </summary>

public Vector2 StartPosition { get; set; }

private BasePlayer playerScript;

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="tag">Тег игрока</param>

/// <param name="playerScript">Сценарий поведения игрока</param>

public PlayerConstructor(string tag, BasePlayer playerScript)

{

PlayerTag = tag;

this.playerScript = playerScript;

}

/// <summary>

/// Создание игрового объекта персонажа

/// </summary>

/// <returns>Игровой объект</returns>

public GameObject CreatePlayer()

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(StartPosition, new Size2F(1f, 1f)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/" + PlayerTag + "/left idle 1.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(0.5f, 0.5f), new Vector2(0, 0.2f)));

gameObject.GameObjectTag = PlayerTag;

gameObject.InitializeObjectScript(playerScript);

PlayerGameObject = gameObject;

return gameObject;

}

/// <summary>

/// Создание игрового объектиа оружия

/// </summary>

/// <returns>Игровой объект</returns>

public GameObject CreateGun(Gun gun, string nameOfgun)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(new Vector2(0f, 0f), new Size2F(1, 1)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/" + PlayerTag + "/" + nameOfgun + " Gun/" + nameOfgun + " left idle 1.png")));

gameObject.GameObjectTag = "Gun";

gameObject.ParentGameObject = PlayerGameObject;

gun.PlayerFactory = this;

gameObject.InitializeObjectScript(gun);

playerScript.SetChildGameObject(gameObject);

GameEvents.ChangeGun?.Invoke(PlayerTag, nameOfgun);

return gameObject;

}

/// <summary>

/// Создание игрового объекта рук

/// </summary>

/// <returns>Игровой объект</returns>

public GameObject CreateArms()

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(new Vector2(0f, 0f), new Size2F(1, 1)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/" + PlayerTag + "/Arms/arms left idle 1.png")));

gameObject.GameObjectTag = "Arms";

Animation animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerTag + "/Arms/arms left idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerTag + "/Arms/arms left run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerTag + "/Arms/arms right idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleRight", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerTag + "/Arms/arms right run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runRight", animation);

gameObject.Sprite.SetAnimation("idleLeft");

playerScript.SetChildGameObject(gameObject);

gameObject.ParentGameObject = PlayerGameObject;

GameEvents.ChangeGun?.Invoke(PlayerTag, "");

return gameObject;

}

}

}

**Код программы для *Gun*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Game;

using GameLibrary.Maze;

using SharpDX;

namespace GameLibrary.Guns

{

/// <summary>

/// Абстрактный класс оружия

/// </summary>

public abstract class Gun : ObjectScript

{

/// <summary>

/// Время использования оружия

/// </summary>

public abstract float UseTime { get; }

/// <summary>

/// Время перезарядки оружия

/// </summary>

public abstract float ReloadTime { get; }

/// <summary>

/// Фабрика персонажа, к которому прикреплено оружие

/// </summary>

public PlayerConstructor PlayerFactory { get; set; }

/// <summary>

/// Экземпляр сцены игры

/// </summary>

protected MazeScene maze;

private Player playerScript;

private float currentReloadTime;

private float currentUseTime;

/// <summary>

/// Поведение на момент создание игрового объекта

/// </summary>

public override void Start()

{

playerScript = ((Player)gameObject.ParentGameObject.Script);

maze = MazeScene.instance;

currentReloadTime = Time.CurrentTime + ReloadTime;

currentUseTime = Time.CurrentTime + UseTime;

LoadAnimation();

}

/// <summary>

/// Загрузка анимации оружия

/// </summary>

protected abstract void LoadAnimation();

/// <summary>

/// Обновление игрового объекта

/// </summary>

public override void Update()

{

if (playerScript.IsCanMove && Input.GetButtonDawn(playerScript.Control.ShootKey) && currentReloadTime < Time.CurrentTime)

{

Vector2 bulletSpawnPosition = gameObject.ParentGameObject.Transform.Position;

Vector2 bulletDirection = new Vector2(gameObject.Sprite.IsFlipX ? -1 : 1, 0);

SpawnBullet(bulletSpawnPosition, bulletDirection);

currentReloadTime = Time.CurrentTime + ReloadTime;

}

if (currentUseTime < Time.CurrentTime)

{

maze.AddObjectOnScene(PlayerFactory.CreateArms());

maze.RemoveObjectFromScene(gameObject);

}

}

/// <summary>

/// Создание пули из фабрики

/// </summary>

/// <param name="position">Позиция создания</param>

/// <param name="direction">Направление пули</param>

protected abstract void SpawnBullet(Vector2 position, Vector2 direction);

}

}

**Код программы для *GunDecorator*.*cs*:**

using SharpDX;

namespace GameLibrary.Guns

{

/// <summary>

/// Абстрактный класс декоратора оружия

/// </summary>

public abstract class GunDecorator : Gun

{

/// <summary>

/// Время использования оружия

/// </summary>

public override float UseTime => decoratedGun.UseTime;

/// <summary>

/// Время перезарядки оружия

/// </summary>

public override float ReloadTime => decoratedGun.ReloadTime;

/// <summary>

/// Декорируемое оружие

/// </summary>

protected Gun decoratedGun;

/// <summary>

/// Установление декорируемого оружия

/// </summary>

/// <param name="gun">Оружие</param>

public void SetDecoratedGun(Gun gun)

{

decoratedGun = gun;

}

/// <summary>

/// Загрузка анимации оружия

/// </summary>

protected override void LoadAnimation() { }

/// <summary>

/// Создание пули из фабрики

/// </summary>

/// <param name="position">Позиция создания</param>

/// <param name="direction">Направление пули</param>

protected override void SpawnBullet(Vector2 position, Vector2 direction) { }

}

}

**Код программы для *DamageGun*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Bullets.BulletFactories;

using SharpDX;

namespace GameLibrary.Guns

{

/// <summary>

/// Класс оружия урона

/// </summary>

public class DamageGun : Gun

{

/// <summary>

/// Время использования оружия

/// </summary>

public override float UseTime => 10f;

/// <summary>

/// Время перезарядки оружия

/// </summary>

public override float ReloadTime => 2f;

/// <summary>

/// Загрузка анимации оружия

/// </summary>

protected override void LoadAnimation()

{

Animation animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Damage Gun/damage left idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Damage Gun/damage left run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Damage Gun/damage right idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleRight", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Damage Gun/damage right run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runRight", animation);

gameObject.Sprite.SetAnimation("idleLeft");

}

/// <summary>

/// Создание пули из фабрики

/// </summary>

/// <param name="position">Позиция создания</param>

/// <param name="direction">Направление пули</param>

protected override void SpawnBullet(Vector2 position, Vector2 direction)

{

DamageBulletFactory factory = new DamageBulletFactory();

maze.AddObjectOnScene(factory.CreateBullet(position, direction, gameObject.ParentGameObject.GameObjectTag));

}

}

}

**Код программы для *SlowdownGun*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Bullets.BulletFactories;

using SharpDX;

namespace GameLibrary.Guns

{

/// <summary>

/// Класс замедляющего оружия

/// </summary>

public class SlowdownGun : GunDecorator

{

/// <summary>

/// Время использования оружия

/// </summary>

public override float UseTime => decoratedGun.UseTime + 10f;

/// <summary>

/// Время перезарядки оружия

/// </summary>

public override float ReloadTime => decoratedGun.ReloadTime - 1f;

/// <summary>

/// Загрузка анимации оружия

/// </summary>

protected override void LoadAnimation()

{

Animation animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Slowdown Gun/slowdown left idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Slowdown Gun/slowdown left run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Slowdown Gun/slowdown right idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleRight", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Slowdown Gun/slowdown right run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runRight", animation);

gameObject.Sprite.SetAnimation("idleLeft");

}

/// <summary>

/// Создание пули из фабрики

/// </summary>

/// <param name="position">Позиция создания</param>

/// <param name="direction">Направление пули</param>

protected override void SpawnBullet(Vector2 position, Vector2 direction)

{

SlowdownBulletFactory factory = new SlowdownBulletFactory();

maze.AddObjectOnScene(factory.CreateBullet(position, direction, gameObject.ParentGameObject.GameObjectTag));

}

}

}

**Код программы для *FrezzeGun*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Bullets.BulletFactories;

using SharpDX;

namespace GameLibrary.Guns

{

/// <summary>

/// Класс оружия заморозки

/// </summary>

public class FrezzeGun : GunDecorator

{

/// <summary>

/// Время использования оружия

/// </summary>

public override float UseTime => decoratedGun.UseTime + 5f;

/// <summary>

/// Время перезарядки оружия

/// </summary>

public override float ReloadTime => decoratedGun.ReloadTime - 0.5f;

/// <summary>

/// Загрузка анимации оружия

/// </summary>

protected override void LoadAnimation()

{

Animation animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Frezze Gun/frezze left idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Frezze Gun/frezze left run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Frezze Gun/frezze right idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleRight", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/" + PlayerFactory.PlayerTag + "/Frezze Gun/frezze right run ", 4), 0.2f, true);

gameObject.Sprite.AddAnimation("runRight", animation);

gameObject.Sprite.SetAnimation("idleLeft");

}

/// <summary>

/// Создание пули из фабрики

/// </summary>

/// <param name="position">Позиция создания пули</param>

/// <param name="direction">Направление пули</param>

protected override void SpawnBullet(Vector2 position, Vector2 direction)

{

FrezzeBulletFactory factory = new FrezzeBulletFactory();

maze.AddObjectOnScene(factory.CreateBullet(position, direction, gameObject.ParentGameObject.GameObjectTag));

}

}

}

**Код программы для *GunSpawnFactory*.*cs*:**

using EngineLibrary.ObjectComponents;

using SharpDX;

namespace GameLibrary.Guns.SpawnFactories

{

/// <summary>

/// Фбстрактная фабрика создания места подбираемого оружия

/// </summary>

public abstract class GunSpawnFactory

{

/// <summary>

/// Создание места подбираемого оружия

/// </summary>

/// <param name="position">Позиция появления</param>

/// <returns>Игровой объект</returns>

public abstract GameObject CreateGunSpawn(Vector2 position);

}

}

**Код программы для *DamageGunSpawnFactory*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Maze;

using SharpDX;

namespace GameLibrary.Guns.SpawnFactories

{

/// <summary>

/// Фабрика создания места подбираемого оружия урона

/// </summary>

public class DamageGunSpawnFactory : GunSpawnFactory

{

/// <summary>

/// Создание места подбираемого оружия с уроном

/// </summary>

/// <param name="position">Позиция появления</param>

/// <returns>Игровой объект</returns>

public override GameObject CreateGunSpawn(Vector2 position)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(position, new Size2F(1, 1)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/MazeElements/Spawn Guns/damage spawn 1.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(0.8f, 0.8f)));

gameObject.GameObjectTag = "Spawn";

GunSpawn gunSpawn = new GunSpawn();

DamageGun damageGun = new DamageGun();

gunSpawn.InitializeGunSpawn("damage", damageGun, 10f);

gameObject.InitializeObjectScript(gunSpawn);

return gameObject;

}

}

}

**Код программы для *SlowdownGunSpawnFactory*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Maze;

using SharpDX;

namespace GameLibrary.Guns.SpawnFactories

{

/// <summary>

/// Фабрика создания места подбираемого оружия замедления

/// </summary>

public class SlowdownGunSpawnFactory : GunSpawnFactory

{

/// <summary>

/// Создание места подбираемого оружия замедления

/// </summary>

/// <param name="position">Позиция появления</param>

/// <returns>Игровой объект</returns>

public override GameObject CreateGunSpawn(Vector2 position)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(position, new Size2F(1, 1)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/MazeElements/Spawn Guns/slowdown spawn 1.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(0.8f, 0.8f)));

gameObject.GameObjectTag = "Spawn";

GunSpawn gunSpawn = new GunSpawn();

DamageGun damageGun = new DamageGun();

SlowdownGun slowdownGun = new SlowdownGun();

slowdownGun.SetDecoratedGun(damageGun);

gunSpawn.InitializeGunSpawn("slowdown", slowdownGun, 20f);

gameObject.InitializeObjectScript(gunSpawn);

return gameObject;

}

}

}

**Код программы для *FrezzeGunSpawnFactory*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Maze;

using SharpDX;

namespace GameLibrary.Guns.SpawnFactories

{

/// <summary>

/// Фабрика создания места подбираемого оружия заморозки

/// </summary>

public class FrezzeGunSpawnFactory : GunSpawnFactory

{

/// <summary>

/// Создание места подбираемого оружия с заморозкой

/// </summary>

/// <param name="position">Позиция появления</param>

/// <returns>Игровой объект</returns>

public override GameObject CreateGunSpawn(Vector2 position)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(position, new Size2F(1, 1)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/MazeElements/Spawn Guns/slowdown spawn 1.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(0.8f, 0.8f)));

gameObject.GameObjectTag = "Spawn";

GunSpawn gunSpawn = new GunSpawn();

DamageGun damageGun = new DamageGun();

FrezzeGun frezzeGun = new FrezzeGun();

frezzeGun.SetDecoratedGun(damageGun);

gunSpawn.InitializeGunSpawn("frezze", frezzeGun, 20f);

gameObject.InitializeObjectScript(gunSpawn);

return gameObject;

}

}

}

**Код программы для *Bullet*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Maze;

using SharpDX;

namespace GameLibrary.Bullets

{

/// <summary>

/// Абстрактный класс пули

/// </summary>

public abstract class Bullet : ObjectScript

{

/// <summary>

/// Скорость пули

/// </summary>

public abstract float Speed { get; }

/// <summary>

/// Экземпляр сцены игры

/// </summary>

protected MazeScene maze;

private Vector2 flyDirection;

private string interactionTag;

/// <summary>

/// Установление направления полета пули

/// </summary>

/// <param name="direction">Вектор направления</param>

/// <param name="tag">Тег игрового объекта, создающий пулю</param>

public void SetSettings(Vector2 direction, string tag)

{

flyDirection = direction;

if (tag == "Blue Player")

interactionTag = "Red Player";

else

interactionTag = "Blue Player";

}

/// <summary>

/// Поведение на момент создание игрового объекта

/// </summary>

public override void Start()

{

maze = MazeScene.instance;

}

/// <summary>

/// Обновление игрового объекта

/// </summary>

public override void Update()

{

Vector2 movement = flyDirection \* Speed \* Time.DeltaTime;

gameObject.Transform.SetMovement(movement);

if (gameObject.Collider.CheckIntersection("Wall"))

maze.RemoveObjectFromScene(gameObject);

if (gameObject.Collider.CheckIntersection(out BreakWall wall))

{

maze.RemoveObjectFromScene(gameObject);

wall.DestroyWall();

}

if (gameObject.Collider.CheckIntersection(out GameObject playerGameObject, interactionTag))

{

PlayerInteraction(playerGameObject);

maze.RemoveObjectFromScene(gameObject);

}

}

/// <summary>

/// Взаимодействие с игроком

/// </summary>

public abstract void PlayerInteraction(GameObject playerGameObject);

}

}

**Код программы для *BulletDecorator*.*cs*:**

using EngineLibrary.ObjectComponents;

namespace GameLibrary.Bullets

{

/// <summary>

/// Абстрактный класс декоратора пули

/// </summary>

public class BulletDecorator : Bullet

{

/// <summary>

/// Скорость пули

/// </summary>

public override float Speed => decoratedBullet.Speed;

/// <summary>

/// Декорируемая пуля

/// </summary>

protected Bullet decoratedBullet;

/// <summary>

/// Установление декорируемой пули

/// </summary>

/// <param name="bullet">Пуля</param>

public void SetDecoratedBullet(Bullet bullet)

{

decoratedBullet = bullet;

}

/// <summary>

/// Взаимодействие с игроком

/// </summary>

public override void PlayerInteraction(GameObject playerGameObject) { }

}

}

**Код программы для *DamageBullet*.*cs*:**

using EngineLibrary.ObjectComponents;

using GameLibrary.Effects.EffectFactories;

namespace GameLibrary.Bullets

{

/// <summary>

/// Класс пули урона

/// </summary>

public class DamageBullet : Bullet

{

/// <summary>

/// Скорость пули

/// </summary>

public override float Speed => 10f;

/// <summary>

/// Взаимодействие с игроком

/// </summary>

public override void PlayerInteraction(GameObject playerGameObject)

{

DamageEffectFactory factory = new DamageEffectFactory();

maze.AddObjectOnScene(factory.CreateEffect(playerGameObject));

}

}

}

**Код программы для *SlowdownBullet*.*cs*:**

using EngineLibrary.ObjectComponents;

using GameLibrary.Effects;

using GameLibrary.Effects.EffectFactories;

namespace GameLibrary.Bullets

{

/// <summary>

/// Класс замедляющей пули, декорируемый от пули урона

/// </summary>

public class SlowdownBullet : BulletDecorator

{

/// <summary>

/// Скорость полёта

/// </summary>

public override float Speed => decoratedBullet.Speed + 10f;

/// <summary>

/// Взаимодействие с игроком

/// </summary>

/// <param name="playerGameObject">Игровой объект игрока</param>

public override void PlayerInteraction(GameObject playerGameObject)

{

if (playerGameObject.GameObjectTag == "Blue Player" && SlowdownEffect.BluePlayerEffect != null) return;

if (playerGameObject.GameObjectTag == "Red Player" && SlowdownEffect.RedPlayerEffect != null) return;

SlowdownEffectFactory factory = new SlowdownEffectFactory();

maze.AddObjectOnScene(factory.CreateEffect(playerGameObject));

}

}

}

**Код программы для *FrezzeBullet*.*cs*:**

using EngineLibrary.ObjectComponents;

using GameLibrary.Effects;

using GameLibrary.Effects.EffectFactories;

namespace GameLibrary.Bullets

{

/// <summary>

/// Класс замораживающей пули, декорируемый от пули урона

/// </summary>

public class FrezzeBullet : BulletDecorator

{

/// <summary>

/// Скорость полёта

/// </summary>

public override float Speed => decoratedBullet.Speed + 5f;

/// <summary>

/// Взаимодействие с игроком

/// </summary>

/// <param name="playerGameObject">Игровой объект игрока</param>

public override void PlayerInteraction(GameObject playerGameObject)

{

if (playerGameObject.GameObjectTag == "Blue Player" && FrezzeEffect.BluePlayerEffect != null) return;

if (playerGameObject.GameObjectTag == "Red Player" && FrezzeEffect.RedPlayerEffect != null) return;

FrezzeEffectFactory factory = new FrezzeEffectFactory();

maze.AddObjectOnScene(factory.CreateEffect(playerGameObject));

}

}

}

**Код программы для *BulletFactory*.*cs*:**

using EngineLibrary.ObjectComponents;

using GameLibrary.Effects.EffectFactories;

using SharpDX;

namespace GameLibrary.Bullets.BulletFactories

{

/// <summary>

/// Абстрактный класс фабрики создания пуль

/// </summary>

public abstract class BulletFactory

{

/// <summary>

/// Создание игрового объекта пули

/// </summary>

/// <param name="position">Позиция появления пули</param>

/// <param name="direction">Направление пули</param>

/// <param name="tag">Тег игрового объекта, создающий пулю</param>

/// <returns>Игровой объект</returns>

public abstract GameObject CreateBullet(Vector2 position, Vector2 direction, string tag);

/// <summary>

/// Создание игрового объекта пули

/// </summary>

/// <param name="type">Тип пули</param>

/// <param name="position">Позиция появления пули</param>

/// <param name="direction">Направление пули</param>

/// <param name="tag">Тег игрового объекта, создающий пулю</param>

/// <returns>Игровой объект</returns>

public static GameObject CreateBullet(BulletType type, Vector2 position, Vector2 direction, string tag)

{

switch (type)

{

case BulletType.Damage:

return new DamageBulletFactory().CreateBullet(position, direction, tag);

case BulletType.Slowdown:

return new SlowdownBulletFactory().CreateBullet(position, direction, tag);

case BulletType.Frezze:

return new FrezzeBulletFactory().CreateBullet(position, direction, tag);

default:

return null;

}

}

}

}

**Код программы для *DamageBulletFactory*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using SharpDX;

namespace GameLibrary.Bullets.BulletFactories

{

/// <summary>

/// Класс фабрики создания пули урона

/// </summary>

public class DamageBulletFactory : BulletFactory

{

/// <summary>

/// Создание игрового объекта пули, которая убивает

/// </summary>

/// <param name="position">Позиция появления пули</param>

/// <param name="direction">Направление пули</param>

/// <param name="tag">Тег игрового объекта, создающий пулю</param>

/// <returns>Игровой объект</returns>

public override GameObject CreateBullet(Vector2 position, Vector2 direction, string tag)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(position, new Size2F(1, 1)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/Bullets/damage bullet.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(0.4f, 0.4f)));

gameObject.GameObjectTag = "Bullet";

DamageBullet bullet = new DamageBullet();

bullet.SetSettings(direction, tag);

gameObject.InitializeObjectScript(bullet);

return gameObject;

}

}

}

**Код программы для *SlowdownBulletFactory*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using SharpDX;

namespace GameLibrary.Bullets.BulletFactories

{

/// <summary>

/// Класс фабрики создания замедляющей пули

/// </summary>

public class SlowdownBulletFactory : BulletFactory

{

/// <summary>

/// Создание игрового объекта пули, которая замедляет

/// </summary>

/// <param name="position">Позиция появления пули</param>

/// <param name="direction">Направление пули</param>

/// /// <param name="tag">Тег игрового объекта, создающий пулю</param>

/// <returns>Игровой объект</returns>

public override GameObject CreateBullet(Vector2 position, Vector2 direction, string tag)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(position, new Size2F(1, 1)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/Bullets/slowdown bullet.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(0.4f, 0.4f)));

gameObject.GameObjectTag = "Bullet";

DamageBullet bullet = new DamageBullet();

SlowdownBullet slowdownBullet = new SlowdownBullet();

slowdownBullet.SetDecoratedBullet(bullet);

slowdownBullet.SetSettings(direction, tag);

gameObject.InitializeObjectScript(slowdownBullet);

return gameObject;

}

}

}

**Код программы для *FrezzeBulletFactory*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using SharpDX;

namespace GameLibrary.Bullets.BulletFactories

{

/// <summary>

/// Класс фабрики создания замораживающей пули

/// </summary>

public class FrezzeBulletFactory : BulletFactory

{

/// <summary>

/// Создание игрового объекта пули, которая замораживает

/// </summary>

/// <param name="position">Позиция появления пули</param>

/// <param name="direction">Направление пули</param>

/// /// <param name="tag">Тег игрового объекта, создающий пулю</param>

/// <returns>Игровой объект</returns>

public override GameObject CreateBullet(Vector2 position, Vector2 direction, string tag)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(position, new Size2F(1, 1)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/Bullets/frezze bullet.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(0.4f, 0.4f)));

gameObject.GameObjectTag = "Bullet";

DamageBullet bullet = new DamageBullet();

FrezzeBullet frezzeBullet = new FrezzeBullet();

frezzeBullet.SetDecoratedBullet(bullet);

frezzeBullet.SetSettings(direction, tag);

gameObject.InitializeObjectScript(frezzeBullet);

return gameObject;

}

}

}

**Код программы для *Effect*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Game;

using GameLibrary.Maze;

namespace GameLibrary.Effects

{

/// <summary>

/// Асбтрактный класс эффекта

/// </summary>

public abstract class Effect : ObjectScript

{

/// <summary>

/// Время дейтсвия эффекта

/// </summary>

public abstract float EffectTime { get; }

/// <summary>

/// Экземпляр сцены игры

/// </summary>

protected MazeScene maze;

/// <summary>

/// Ссылка на сценарий игрока, на которого действует эффект

/// </summary>

protected Player playerScript;

/// <summary>

/// Ссылка на игровой объект, на которого действует эффект

/// </summary>

protected GameObject playerGameObject;

private float currentEffectTime;

/// <summary>

/// Поведение на момент создание игрового объекта

/// </summary>

public override void Start()

{

maze = MazeScene.instance;

currentEffectTime = Time.CurrentTime + EffectTime;

Initialize();

}

/// <summary>

/// Обновление игрового объекта

/// </summary>

public override void Update()

{

if (currentEffectTime < Time.CurrentTime)

DeactivateEffect();

BehaviorOnScene();

}

/// <summary>

/// Активация эффекта

/// </summary>

/// <param name="player">Игровой объект, на который будет наложен эффект</param>

public abstract void ActivateEffect(GameObject player);

/// <summary>

/// Инициализация эффекта

/// </summary>

protected abstract void Initialize();

/// <summary>

/// Деактивация эффекта

/// </summary>

protected abstract void DeactivateEffect();

/// <summary>

/// Поведение на сцене

/// </summary>

protected abstract void BehaviorOnScene();

}

}

**Код программы для *EffectDecorator*.*cs*:**

namespace GameLibrary.Effects

{

/// <summary>

/// Абстрактный класс декоратора эффекта

/// </summary>

public abstract class EffectDecorator : Effect

{

/// <summary>

/// Время дейтсвия эффекта

/// </summary>

public override float EffectTime => decoratedEffect.EffectTime;

/// <summary>

/// Дерорируемый эффект

/// </summary>

protected DamageEffect decoratedEffect;

/// <summary>

/// Установка декорируемого эффекта

/// </summary>

/// <param name="effect">Декорируемый эффект</param>

public void SetDecoratedEffect(DamageEffect effect)

{

decoratedEffect = effect;

}

/// <summary>

/// Инициализация эффекта

/// </summary>

protected override void BehaviorOnScene() { }

/// <summary>

/// Деактивация эффекта

/// </summary>

protected override void DeactivateEffect() { }

/// <summary>

/// Поведение на сцене

/// </summary>

protected override void Initialize() { }

}

}

**Код программы для *DamageEffect*.*cs*:**

using EngineLibrary.ObjectComponents;

using GameLibrary.Game;

namespace GameLibrary.Effects

{

/// <summary>

/// Эффект после урона

/// </summary>

public class DamageEffect : Effect

{

/// <summary>

/// Время дейтсвия эффекта

/// </summary>

public override float EffectTime => 5f;

private int storedCoins;

/// <summary>

/// Активация эффекта

/// </summary>

/// <param name="player">Игровой объект, на который будет наложен эффект</param>

public override void ActivateEffect(GameObject player)

{

playerGameObject = player;

playerGameObject.IsActive = false;

storedCoins = (playerGameObject.Script as Player).Coins / 2;

(playerGameObject.Script as Player).ChangeCoinsValue(-storedCoins);

GameEvents.ChangeEffect?.Invoke(playerGameObject.GameObjectTag, "Death");

}

/// <summary>

/// Инициализация эффекта

/// </summary>

protected override void Initialize() { }

/// <summary>

/// Деактивация эффекта

/// </summary>

protected override void DeactivateEffect()

{

if (!playerGameObject.IsActive)

{

if (playerGameObject.GameObjectTag == "Blue Player")

playerGameObject.Transform.Position = maze.BluePlayerFactory.StartPosition;

else

playerGameObject.Transform.Position = maze.RedPlayerFactory.StartPosition;

playerGameObject.IsActive = true;

}

GameEvents.ChangeEffect?.Invoke(playerGameObject.GameObjectTag, "");

}

/// <summary>

/// Поведение на сцене

/// </summary>

protected override void BehaviorOnScene()

{

if (gameObject.Sprite.Bitmap == null)

{

gameObject.Sprite.SetAnimation("idle");

}

if (gameObject.Collider.CheckIntersection(out Player player) && gameObject.IsActive)

{

player.ChangeCoinsValue(storedCoins);

gameObject.IsActive = false;

}

if(playerGameObject.IsActive && !gameObject.IsActive)

{

maze.RemoveObjectFromScene(gameObject);

}

}

}

}

**Код программы для *SlowdownEffect*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Game;

namespace GameLibrary.Effects

{

/// <summary>

/// Класс эффекта замедления

/// </summary>

public class SlowdownEffect : EffectDecorator

{

/// <summary>

/// Экземпляр текущего эффекта на сцене для синего игрока

/// </summary>

public static SlowdownEffect BluePlayerEffect { get; private set; } = null;

/// <summary>

/// Экземпляр текущего эффекта на сцене для красного игрока

/// </summary>

public static SlowdownEffect RedPlayerEffect { get; private set; } = null;

/// <summary>

/// Время дейтсвия эффекта

/// </summary>

public override float EffectTime => decoratedEffect.EffectTime + 5f;

private float trueSpeed;

/// <summary>

/// Активация эффекта

/// </summary>

/// <param name="player">Игровой объект, на который будет наложен эффект</param>

public override void ActivateEffect(GameObject player)

{

playerGameObject = player;

trueSpeed = (playerGameObject.Script as Player).Speed;

(playerGameObject.Script as Player).Speed = trueSpeed / 2;

if (playerGameObject.GameObjectTag == "Blue Player")

BluePlayerEffect = this;

else

RedPlayerEffect = this;

GameEvents.ChangeEffect?.Invoke(playerGameObject.GameObjectTag, "Slowdown");

}

/// <summary>

/// Поведение на сцене

/// </summary>

protected override void BehaviorOnScene()

{

if (playerGameObject.Sprite.IsFlipX)

gameObject.Sprite.SetAnimation("idleLeft");

else

gameObject.Sprite.SetAnimation("idleRight");

}

/// <summary>

/// Деактивация эффекта

/// </summary>

protected override void DeactivateEffect()

{

(playerGameObject.Script as Player).Speed = trueSpeed;

if (playerGameObject.GameObjectTag == "Blue Player")

BluePlayerEffect = null;

else

RedPlayerEffect = null;

maze.RemoveObjectFromScene(gameObject);

GameEvents.ChangeEffect?.Invoke(playerGameObject.GameObjectTag, "");

}

/// <summary>

/// Инициализация эффекта

/// </summary>

protected override void Initialize()

{

Animation animation = new Animation(RenderingSystem.LoadAnimation("Resources/MazeElements/Effects/slowdown left idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleLeft", animation);

animation = new Animation(RenderingSystem.LoadAnimation("Resources/MazeElements/Effects/slowdown right idle ", 2), 0.2f, true);

gameObject.Sprite.AddAnimation("idleRight", animation);

if(playerGameObject.Sprite.IsFlipX)

gameObject.Sprite.SetAnimation("idleLeft");

else

gameObject.Sprite.SetAnimation("idleRight");

}

}

}

**Код программы для *FrezzeEffect*.*cs*:**

using System.Collections.Generic;

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using GameLibrary.Game;

using SharpDX.Direct2D1;

namespace GameLibrary.Effects

{

/// <summary>

/// Эффект заморозки

/// </summary>

public class FrezzeEffect : EffectDecorator

{

/// <summary>

/// Экземпляр текущего эффекта на сцене для синего игрока

/// </summary>

public static FrezzeEffect BluePlayerEffect { get; private set; } = null;

/// <summary>

/// Экземпляр текущего эффекта на сцене для красного игрока

/// </summary>

public static FrezzeEffect RedPlayerEffect { get; private set; } = null;

/// <summary>

/// Время дейтсвия эффекта

/// </summary>

public override float EffectTime => decoratedEffect.EffectTime + 2f;

/// <summary>

/// Активация эффекта

/// </summary>

/// <param name="player">Игровой объект, на который будет наложен эффект</param>

public override void ActivateEffect(GameObject player)

{

playerGameObject = player;

(playerGameObject.Script as Player).IsCanMove = false;

if (playerGameObject.GameObjectTag == "Blue Player")

BluePlayerEffect = this;

else

RedPlayerEffect = this;

GameEvents.ChangeEffect?.Invoke(playerGameObject.GameObjectTag, "Frezze");

}

/// <summary>

/// Поведение на сцене

/// </summary>

protected override void BehaviorOnScene() { }

/// <summary>

/// Деактивация эффекта

/// </summary>

protected override void DeactivateEffect()

{

(playerGameObject.Script as Player).IsCanMove = true;

if (playerGameObject.GameObjectTag == "Blue Player")

BluePlayerEffect = null;

else

RedPlayerEffect = null;

maze.RemoveObjectFromScene(gameObject);

GameEvents.ChangeEffect?.Invoke(playerGameObject.GameObjectTag, "");

}

/// <summary>

/// Инициализация эффекта

/// </summary>

protected override void Initialize()

{

Animation animation = new Animation(new List<Bitmap>() { RenderingSystem.LoadBitmap("Resources/MazeElements/Effects/frezze left idle.png") }, 1f, true);

gameObject.Sprite.AddAnimation("idleLeft", animation);

animation = new Animation(new List<Bitmap>() { RenderingSystem.LoadBitmap("Resources/MazeElements/Effects/frezze right idle.png") }, 1f, true);

gameObject.Sprite.AddAnimation("idleRight", animation);

if (playerGameObject.Sprite.IsFlipX)

gameObject.Sprite.SetAnimation("idleLeft");

else

gameObject.Sprite.SetAnimation("idleRight");

}

}

}

**Код программы для *EffectFactory*.*cs*:**

using EngineLibrary.ObjectComponents;

namespace GameLibrary.Effects.EffectFactories

{

/// <summary>

/// Абстрактная фабрика создания эффектов

/// </summary>

public abstract class EffectFactory

{

/// <summary>

/// Создание эффекта

/// </summary>

/// <param name="player">Игровой объект игрока</param>

/// <returns>Игровой объект</returns>

public abstract GameObject CreateEffect(GameObject player);

}

}

**Код программы для *DamageEffectFactory*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using SharpDX;

namespace GameLibrary.Effects.EffectFactories

{

/// <summary>

/// Фабрика создания эффекта смерти

/// </summary>

public class DamageEffectFactory : EffectFactory

{

/// <summary>

/// Создание могилы с монетами

/// </summary>

/// <param name="player">Игровой объект игрока</param>

/// <returns>Игровой объект</returns>

public override GameObject CreateEffect(GameObject player)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(player.Transform.Position, new Size2F(1f, 1f)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/MazeElements/Effects/damage idle 1.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(0.8f, 0.8f)));

gameObject.GameObjectTag = "Effect";

DamageEffect damageEffect = new DamageEffect();

damageEffect.ActivateEffect(player);

gameObject.InitializeObjectScript(damageEffect);

return gameObject;

}

}

}

**Код программы для *SlowdownEffectFactory*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using SharpDX;

namespace GameLibrary.Effects.EffectFactories

{

/// <summary>

/// Фабрика создания эффекта замедления

/// </summary>

public class SlowdownEffectFactory : EffectFactory

{

/// <summary>

/// Создание замедляющего эффекта

/// </summary>

/// <param name="player">Игровой объект игрока</param>

/// <returns>Игровой объект</returns>

public override GameObject CreateEffect(GameObject player)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(new Vector2(0, 0), new Size2F(1f, 1f)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/MazeElements/Effects/slowdown left idle 1.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(0.8f, 0.8f)));

gameObject.GameObjectTag = "Effect";

gameObject.ParentGameObject = player;

DamageEffect damageEffect = new DamageEffect();

SlowdownEffect slowdownEffect = new SlowdownEffect();

slowdownEffect.SetDecoratedEffect(damageEffect);

slowdownEffect.ActivateEffect(player);

gameObject.InitializeObjectScript(slowdownEffect);

return gameObject;

}

}

}

**Код программы для *FrezzeEffectFactory*.*cs*:**

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using SharpDX;

namespace GameLibrary.Effects.EffectFactories

{

/// <summary>

/// Фабрика создания эффекта заморозки

/// </summary>

public class FrezzeEffectFactory : EffectFactory

{

/// <summary>

/// Создание замораживающего эффекта

/// </summary>

/// <param name="player">Игровой объект игрока</param>

/// <returns>Игровой объект</returns>

public override GameObject CreateEffect(GameObject player)

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(new Vector2(0, 0), new Size2F(1f, 1f)));

gameObject.InitializeObjectComponent(new SpriteComponent(RenderingSystem.LoadBitmap("Resources/MazeElements/Effects/frezze left idle.png")));

gameObject.InitializeObjectComponent(new ColliderComponent(gameObject, new Size2F(0.8f, 0.8f)));

gameObject.GameObjectTag = "Effect";

gameObject.ParentGameObject = player;

DamageEffect damageEffect = new DamageEffect();

FrezzeEffect frezzeEffect = new FrezzeEffect();

frezzeEffect.SetDecoratedEffect(damageEffect);

frezzeEffect.ActivateEffect(player);

gameObject.InitializeObjectScript(frezzeEffect);

return gameObject;

}

}

}

**Код программы для *GameEvents*.*cs*:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GameLibrary.Game

{

/// <summary>

/// Статический класс событий игры

/// </summary>

public static class GameEvents

{

/// <summary>

/// Делегат события изменения количества монет

/// </summary>

/// <param name="tagPlayer">Тег игрового объекта игрока</param>

/// <param name="value">Значение собранных монет</param>

public delegate void CoinsDelegate(string tagPlayer, int value);

/// <summary>

/// Событие изменения количества монет

/// </summary>

///

public static CoinsDelegate ChangeCoins { get; set; }

/// <summary>

/// Делегат события окончания игры

/// </summary>

public delegate void EndGameDelegate(string winPlayer);

/// <summary>

/// Событие окончания игры

/// </summary>

public static EndGameDelegate EndGame { get; set; }

/// <summary>

/// Делегат события получения эффекта игроком

/// </summary>

/// <param name="tagPlayer">Тег игрового объекта игрока</param>

/// <param name="effectName">Название эффекта</param>

public delegate void EffectDelegate(string tagPlayer, string effectName);

/// <summary>

/// Событие получения эффекта игроком

/// </summary>

public static EffectDelegate ChangeEffect { get; set; }

/// <summary>

/// Делегат события получения оружия игроком

/// </summary>

/// <param name="tagPlayer">Тег игрового объекта игрока</param>

/// <param name="gunName">Название оружия</param>

public delegate void GunDelegate(string tagPlayer, string gunName);

/// <summary>

/// Событие получения оружия игроком

/// </summary>

public static GunDelegate ChangeGun { get; set; }

}

}

**Код программы для *BulletNetworkData*.*cs*:**

using SharpDX;

namespace GameLibrary.Maze.Network

{

/// <summary>

/// Игровые данные, отправляемые по сети

/// </summary>

public class BulletNetworkData

{

/// <summary>

/// Индефикатор пули

/// </summary>

public int TypeId { get; set; }

/// <summary>

/// Позиция появления

/// </summary>

public Vector2 SpawnPosition { get; set; }

/// <summary>

/// Направление пули

/// </summary>

public Vector2 Direction { get; set; }

/// <summary>

/// Тег игрока, создавшего пулю

/// </summary>

public string Tag { get; set; }

}

}

**Код программы для *NetworkData*.*cs*:**

using SharpDX;

namespace GameLibrary.Maze.Network

{

/// <summary>

/// Игровые данные, отправляемые по сети

/// </summary>

public class NetworkData

{

/// <summary>

/// Позиция игрока

/// </summary>

public Vector2 PlayerPosition { get; set; }

/// <summary>

/// Повернут ли спрайт игрока

/// </summary>

public bool IsPlayerSpriteFlip { get; set; }

/// <summary>

/// Передвигался ли игрок

/// </summary>

public bool IsPlayerMove { get; set; }

/// <summary>

/// Данные о созданной пуле

/// </summary>

public BulletNetworkData BulletData { get; set; }

}

}

**Код программы для *NetworkManager*.*cs*:**

using System;

using System.Diagnostics;

using System.Threading.Tasks;

using GameLibrary.Bullets;

using GameLibrary.Bullets.BulletFactories;

using NetworkLibrary;

namespace GameLibrary.Maze.Network

{

/// <summary>

/// Менеджер сетевого взаимодействия

/// </summary>

public class NetworkManager

{

/// <summary>

/// Событие записи данных

/// </summary>

public event Action<NetworkManager> OnWriteData;

/// <summary>

/// Событие обновления данных

/// </summary>

public event Action<NetworkManager> OnUpdateData;

private INetworkHandler networkHandler;

/// <summary>

/// Данные текущего игрока

/// </summary>

public NetworkData CurrentPlayerNetworkData { get; private set; }

/// <summary>

/// Данные игрока по сети

/// </summary>

public NetworkData NetworkPlayerNetworkData { get; private set; }

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="networkHandler">Обработчик сетевого взаимодействия</param>

public NetworkManager(INetworkHandler networkHandler)

{

this.networkHandler = networkHandler;

CurrentPlayerNetworkData = new NetworkData();

NetworkPlayerNetworkData = new NetworkData();

networkHandler.OnDataGot += OnDataGot;

}

/// <summary>

/// Обновление данных

/// </summary>

public void UpdateData()

{

OnWriteData?.Invoke(this);

networkHandler.UpdateData(CurrentPlayerNetworkData);

}

/// <summary>

/// Событие, когда данные по сети получены

/// </summary>

/// <param name="data">Данные</param>

private void OnDataGot(object data)

{

NetworkPlayerNetworkData = (NetworkData)data;

var bulletData = NetworkPlayerNetworkData.BulletData;

if (bulletData != null)

{

MazeScene.instance.AddObjectOnScene(

BulletFactory.CreateBullet((BulletType) bulletData.TypeId, bulletData.SpawnPosition, bulletData.Direction, bulletData.Tag));

}

CurrentPlayerNetworkData.BulletData = null;

NetworkPlayerNetworkData.BulletData = null;

OnUpdateData?.Invoke(this);

}

}

}

**Код программы для *INetworkHandler*.*cs*:**

using System;

using System.Threading.Tasks;

namespace NetworkLibrary

{

/// <summary>

/// Интерфейс объекта для сетевого взаимодействия

/// </summary>

public interface INetworkHandler

{

/// <summary>

/// Событие получения данных по сети

/// </summary>

event Action<object> OnDataGot;

/// <summary>

/// Обновление данных

/// </summary>

/// <param name="obj">Объект для передачи по сети</param>

/// <typeparam name="T">Тип объекта для передачи по сети</typeparam>

Task UpdateData<T>(T obj);

/// <summary>

/// Очистка случшителей событий

/// </summary>

void ClearListeners();

}

}

**Код программы для *Server*.*cs*:**

using System;

using System.IO;

using System.Net;

using System.Threading.Tasks;

using Newtonsoft.Json;

namespace NetworkLibrary

{

/// <summary>

/// Класс сетевого взаимодействия для сервера

/// </summary>

public class Server : INetworkHandler, IDisposable

{

private HttpListener \_listener;

/// <summary>

/// Событие получения данных по сети

/// </summary>

public event Action<object> OnDataGot;

/// <summary>

/// Конструктор класса

/// </summary>

public Server()

{

\_listener = new HttpListener();

\_listener.Prefixes.Add($"http://\*:{8000}/");

\_listener.Start();

}

/// <summary>

/// Обновление данных

/// </summary>

/// <param name="obj">Объект для передачи по сети</param>

/// <typeparam name="T">Тип объекта для передачи по сети</typeparam>

public async Task UpdateData<T>(T obj)

{

try

{

HttpListenerContext context = await \_listener.GetContextAsync();

var result = HandleRequest<T>(context.Request);

SendResponse(context.Response, obj);

OnDataGot?.Invoke(result);

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

/// <summary>

/// Отправка ответа клиенту

/// </summary>

/// <param name="response">Объекта ответа</param>

/// <param name="obj">Объект для передачи по сети</param>

/// <typeparam name="T">Тип объекта для передачи по сети</typeparam>

private void SendResponse<T>(HttpListenerResponse response, T obj)

{

Stream output = null;

try

{

output = response.OutputStream;

string responseStr = JsonConvert.SerializeObject(obj);

byte[] buffer = System.Text.Encoding.UTF8.GetBytes(responseStr);

response.ContentLength64 = buffer.Length;

output.Write(buffer, 0, buffer.Length);

}

catch (Exception e)

{

Console.WriteLine(e);

throw e;

}

finally

{

output?.Close();

}

}

/// <summary>

/// Обработка ответа от клиента

/// </summary>

/// <param name="request">Объекта ответа</param>

/// <typeparam name="T">Тип получаемого объекта по сети</typeparam>

/// <returns>Полученный по сети объекта</returns>

private T HandleRequest<T>(HttpListenerRequest request)

{

if (!request.HasEntityBody)

{

Console.WriteLine("No client data was sent with the request.");

return default;

}

Stream stream = null;

StreamReader reader = null;

try

{

stream = request.InputStream;

reader = new StreamReader(stream, request.ContentEncoding);

var resultText = reader.ReadToEnd();

return JsonConvert.DeserializeObject<T>(resultText);

}

catch (Exception e)

{

Console.WriteLine(e);

throw e;

}

finally

{

stream?.Close();

reader?.Close();

}

}

/// <summary>

/// Очистка случшителей событий

/// </summary>

public void ClearListeners()

{

OnDataGot = null;

}

/// <summary>

/// Освобождение ресурсов

/// </summary>

public void Dispose()

{

\_listener.Close();

}

}

}

**Код программы для *Client*.*cs*:**

using System;

using System.Net.Http;

using System.Text;

using System.Threading.Tasks;

using Newtonsoft.Json;

namespace NetworkLibrary

{

/// <summary>

/// Класс сетевого взаимодействия для клиента

/// </summary>

public class Client : INetworkHandler, IDisposable

{

private HttpClient \_client;

private string \_connectUri;

/// <summary>

/// Событие получения данных по сети

/// </summary>

public event Action<object> OnDataGot;

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="serverAddress">Адрес подклбчения к серверу</param>

public Client(string serverAddress)

{

\_client = new HttpClient(new HttpClientHandler());

\_connectUri = $"http://{serverAddress}:8000/";

}

/// <summary>

/// Обновление данных

/// </summary>

/// <param name="obj">Объект для передачи по сети</param>

/// <typeparam name="T">Тип объекта для передачи по сети</typeparam>

public async Task UpdateData<T>(T obj)

{

try

{

var json = JsonConvert.SerializeObject(obj);

var data = new StringContent(json, Encoding.UTF8, "application/json");

\_client.DefaultRequestHeaders.Accept.Clear();

var response = await \_client.PostAsync(\_connectUri, data);

string resultText = response.Content.ReadAsStringAsync().Result;

var result = JsonConvert.DeserializeObject<T>(resultText);

OnDataGot?.Invoke(result);

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

/// <summary>

/// Получение данных

/// </summary>

/// <typeparam name="T">Тип получаемых данных</typeparam>

public async Task GetData<T>()

{

try

{

var response = await \_client.GetAsync(\_connectUri);

var resultText = response.Content.ReadAsStringAsync().Result;

Console.WriteLine(resultText);

var result = JsonConvert.DeserializeObject<T>(resultText);

OnDataGot?.Invoke(result);

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

/// <summary>

/// Очистка случшителей событий

/// </summary>

public void ClearListeners()

{

OnDataGot = null;

}

/// <summary>

/// Освобождение ресурсов

/// </summary>

public void Dispose()

{

\_client?.Dispose();

}

}

}

**Код программы для *GameWindow*.*xaml*.*cs*:**

using EngineLibrary.EngineComponents;

using GameLibrary.Game;

using GameLibrary.Maze;

using NetworkLibrary;

using System;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

namespace GameApplication

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class GameWindow : Window

{

private RenderingApplication application;

/// <summary>

/// Конструктор класса

/// </summary>

public GameWindow()

{

InitializeComponent();

application = new RenderingApplication();

GameEvents.ChangeCoins += ChangeCoins;

GameEvents.EndGame += EndGame;

GameEvents.ChangeEffect += ChangeEffect;

GameEvents.ChangeGun += ChangeGun;

PreviewKeyDown += GameWindowPreviewKeyDown;

}

/// <summary>

/// Обработка клавиш навигаций в окне

/// </summary>

/// <param name="sender">Объект-отпарвитель</param>

/// <param name="e">Аргументы</param>

static void GameWindowPreviewKeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Left || e.Key == Key.Right || e.Key == Key.Up || e.Key == Key.Down || e.Key == Key.Tab)

{

e.Handled = true;

}

}

/// <summary>

/// Событие закрытия окна

/// </summary>

/// <param name="sender">Объект-отпарвитель</param>

/// <param name="e">Аргументы</param>

private void WindowClosing(object sender, System.ComponentModel.CancelEventArgs e)

{

if(application != null)

application.Dispose();

}

/// <summary>

/// Соыбтия нажатия кнопки OneScreenButtonMenu

/// </summary>

/// <param name="sender">Объект-отпарвитель</param>

/// <param name="e">Аргументы</param>

private void OneScreenButtonMenu\_Click(object sender, RoutedEventArgs e)

{

MainScreen.Visibility = Visibility.Hidden;

OneScreenGamePanel.Visibility = Visibility.Visible;

}

/// <summary>

/// Соыбтия нажатия кнопки LANButtonMenu

/// </summary>

/// <param name="sender">Объект-отпарвитель</param>

/// <param name="e">Аргументы</param>

private void LANButtonMenu\_Click(object sender, RoutedEventArgs e)

{

MainScreen.Visibility = Visibility.Hidden;

LANGamePanel.Visibility = Visibility.Visible;

}

/// <summary>

/// Соыбтия нажатия кнопки LANCreateGameButtonPlay

/// </summary>

/// <param name="sender">Объект-отпарвитель</param>

/// <param name="e">Аргументы</param>

private void LANCreateGameButtonPlay\_Click(object sender, RoutedEventArgs e)

{

LANButtonsPanel.Visibility = Visibility.Hidden;

WaitConnectionText.Visibility = Visibility.Visible;

var seed = new Random().Next();

var server = new Server();

server.OnDataGot += (o) => OnClientServed(server, seed);

server.UpdateData(seed);

}

/// <summary>

/// Событие обслуживания подключений сервером

/// </summary>

/// <param name="handler">Интернет обработчик</param>

/// <param name="seed">Сид генерации</param>

private void OnClientServed(INetworkHandler handler, int seed)

{

handler.ClearListeners();

StartGame(new MazeScene(handler, seed, "Blue Player", "Red Player"));

}

/// <summary>

/// Соыбтия нажатия кнопки LANConnectGameButtonPlay

/// </summary>

/// <param name="sender">Объект-отпарвитель</param>

/// <param name="e">Аргументы</param>

private void LANConnectGameButtonPlay\_Click(object sender, RoutedEventArgs e)

{

LANButtonsPanel.Visibility = Visibility.Hidden;

EnterIpPanel.Visibility = Visibility.Visible;

}

/// <summary>

/// Соыбтия нажатия кнопки LANConnectGameButtonEnterIp

/// </summary>

/// <param name="sender">Объект-отпарвитель</param>

/// <param name="e">Аргументы</param>

private void LANConnectGameButtonEnterIp\_Click(object sender, RoutedEventArgs e)

{

EnterIpPanel.Visibility = Visibility.Hidden;

WaitConnectionText.Visibility = Visibility.Visible;

var address = IpInput.Text;

Client client = new Client(address);

client.OnDataGot += (o) => OnServerResponsed(client, o);

client.GetData<int>();

}

/// <summary>

/// Событие результат подключения клиента

/// </summary>

/// <param name="handler">Интернет обработчик</param>

/// <param name="obj">Полученные данные</param>

private void OnServerResponsed(INetworkHandler handler, object obj)

{

var seed = (int) obj;

handler.ClearListeners();

StartGame(new MazeScene(handler, seed, "Red Player", "Blue Player"));

}

/// <summary>

/// Соыбтия нажатия кнопки LANConnectGameButtonPlay

/// </summary>

/// <param name="sender">Объект-отпарвитель</param>

/// <param name="e">Аргументы</param>

private void OneScreenButtonPlay\_Click(object sender, RoutedEventArgs e)

{

StartGame(new MazeScene());

}

/// <summary>

/// Запуск игры

/// </summary>

/// <param name="scene">Сцена игрового приложения</param>

private void StartGame(MazeScene scene)

{

MainScreen.Visibility = Visibility.Hidden;

OneScreenGamePanel.Visibility = Visibility.Hidden;

LANGamePanel.Visibility = Visibility.Hidden;

BluePlayerPanel.Visibility = Visibility.Visible;

RedPlayerPanel.Visibility = Visibility.Visible;

formhost.Child = application.RenderForm;

application.SetScene(scene);

application.Run();

}

/// <summary>

/// Окончание игры

/// </summary>

/// <param name="winPlayer">Тег игрока</param>

private void EndGame(string winPlayer)

{

formhost.Visibility = Visibility.Hidden;

BPGun.Visibility = Visibility.Hidden;

RPGun.Visibility = Visibility.Hidden;

BPEffectText.Text = "";

RPEffectText.Text = "";

if (winPlayer == "Blue Player")

WinPlayerText.Foreground = new SolidColorBrush(Color.FromRgb(9, 46, 168));

else

WinPlayerText.Foreground = new SolidColorBrush(Color.FromRgb(179, 22, 22));

WinPlayerText.Text = winPlayer + " Win!";

Uri resourceLocater = new Uri("/Images/"+ winPlayer + ".png", UriKind.Relative);

BitmapImage bitmap = new BitmapImage(resourceLocater);

WinPlayerImage.Source = bitmap;

WinPanel.Visibility = Visibility.Visible;

GameEvents.EndGame -= EndGame;

GameEvents.ChangeCoins -= ChangeCoins;

}

/// <summary>

/// Событие смены количества монет игрока

/// </summary>

/// <param name="player">Тег игрока</param>

/// <param name="value">Значение монет</param>

private void ChangeCoins(string player, int value)

{

if (player == "Blue Player")

BluePlayerCoins.Text = value.ToString();

else

RedPlayerCoins.Text = value.ToString();

}

/// <summary>

/// Событие смены эффекта на игроке

/// </summary>

/// <param name="player">Тег игрока</param>

/// <param name="effect">Накладываемый эффект</param>

private void ChangeEffect(string player, string effect)

{

TextBlock textBlock;

if (player == "Blue Player")

textBlock = BPEffectText;

else

textBlock = RPEffectText;

switch (effect)

{

case "Death":

textBlock.Foreground = new SolidColorBrush(Color.FromRgb(246, 242, 0));

break;

case "Slowdown":

textBlock.Foreground = new SolidColorBrush(Color.FromRgb(29, 216, 0));

break;

case "Frezze":

textBlock.Foreground = new SolidColorBrush(Color.FromRgb(0, 230, 255));

break;

}

textBlock.Text = effect;

}

/// <summary>

/// Событие смены оружия игрока

/// </summary>

/// <param name="player">Тег игрока</param>

/// <param name="gun">Добавляемое оружие</param>

private void ChangeGun(string player, string gun)

{

Image image;

if (player == "Blue Player")

image = BPGun;

else

image = RPGun;

if (gun == "")

{

image.Visibility = Visibility.Hidden;

}

else

{

Uri resourceLocater = new Uri("/Images/Guns/" + gun + ".png", UriKind.Relative);

BitmapImage bitmap = new BitmapImage(resourceLocater);

image.Source = bitmap;

image.Visibility = Visibility.Visible;

}

}

/// <summary>

/// Событие смены фокуса в окне рендеринга

/// </summary>

/// <param name="sender">Объект-отпарвитель</param>

/// <param name="e">Аргументы</param>

private void formhost\_IsKeyboardFocusedChanged(object sender, DependencyPropertyChangedEventArgs e)

{

application.IsFocused = (bool)e.NewValue;

}

}

}

**Код программы для *TestCollision*.*cs*:**

using System;

using EngineLibrary.EngineComponents;

using EngineLibrary.ObjectComponents;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using SharpDX;

namespace TestGameApp

{

[TestClass]

public class TestCollision

{

[TestMethod]

public void TestCollider()

{

GameObject firstObject = new GameObject();

firstObject.InitializeObjectComponent(new TransformComponent(new Vector2(1f, 1f), new Size2F(1, 1)));

firstObject.InitializeObjectComponent(new ColliderComponent(firstObject, new Size2F(1, 1)));

firstObject.GameObjectTag = "Test";

GameObject secondObject = new GameObject();

secondObject.InitializeObjectComponent(new TransformComponent(new Vector2(1f, 3f), new Size2F(1, 1)));

secondObject.InitializeObjectComponent(new ColliderComponent(secondObject, new Size2F(1, 1)));

secondObject.GameObjectTag = "Test";

Assert.IsFalse(firstObject.Collider.CheckIntersection("Test"));

secondObject.Transform.Position = new Vector2(1f, 1.5f);

Assert.IsTrue(firstObject.Collider.CheckIntersection("Test"));

secondObject.GameObjectTag = "";

Assert.IsFalse(firstObject.Collider.CheckIntersection("Test"));

}

}

}

**Код программы для *TestMovement*.*cs*:**

using System;

using System.Text;

using System.Collections.Generic;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using EngineLibrary.ObjectComponents;

using SharpDX;

namespace TestGameApp

{

[TestClass]

public class TestMovement

{

[TestMethod]

public void TestGameObjectMovement()

{

GameObject gameObject = new GameObject();

gameObject.InitializeObjectComponent(new TransformComponent(new Vector2(1f, 1f), new Size2F(1, 1)));

Vector2 offset = new Vector2(1f, 1f);

gameObject.Transform.SetMovement(offset);

Vector2 expected = new Vector2(2f, 0f);

Vector2 actual = gameObject.Transform.Position;

Assert.AreEqual(expected, actual);

}

}

}

**Код программы для *TestDecoratedGun*.*cs*:**

using System;

using GameLibrary.Guns;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace TestGameApp

{

[TestClass]

public class TestDecoratedGun

{

DamageGun damageGun;

[TestInitialize]

public void InitalizeGun()

{

damageGun = new DamageGun();

}

[TestMethod]

public void TestSlowdownGunDecorator()

{

SlowdownGun slowdownGun = new SlowdownGun();

slowdownGun.SetDecoratedGun(damageGun);

float expectedUseTime = 20f;

float actualUseTime = slowdownGun.UseTime;

Assert.AreEqual(expectedUseTime, actualUseTime);

float expectedReloadTime = 1f;

float actualRealodTime = slowdownGun.ReloadTime;

Assert.AreEqual(expectedReloadTime, actualRealodTime);

}

[TestMethod]

public void TestFrezzeGunDecorator()

{

FrezzeGun frezzeGun = new FrezzeGun();

frezzeGun.SetDecoratedGun(damageGun);

float expectedUseTime = 15f;

float actualUseTime = frezzeGun.UseTime;

Assert.AreEqual(expectedUseTime, actualUseTime);

float expectedReloadTime = 1.5f;

float actualRealodTime = frezzeGun.ReloadTime;

Assert.AreEqual(expectedReloadTime, actualRealodTime);

}

}

}

**Код программы для *TestDecoratedBullet*.*cs*:**

using System;

using GameLibrary.Bullets;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace TestGameApp

{

[TestClass]

public class TestDecoratedBullet

{

DamageBullet damageBullet;

[TestInitialize]

public void InitalizeBullet()

{

damageBullet = new DamageBullet();

}

[TestMethod]

public void TestSlowdownBulletDecorator()

{

SlowdownBullet slowdownBullet = new SlowdownBullet();

slowdownBullet.SetDecoratedBullet(damageBullet);

float expectedSpeed = 20f;

float actualSpeed = slowdownBullet.Speed;

Assert.AreEqual(expectedSpeed, actualSpeed);

}

[TestMethod]

public void TestFrezzeBulletDecorator()

{

FrezzeBullet frezzeBullet = new FrezzeBullet();

frezzeBullet.SetDecoratedBullet(damageBullet);

float expectedSpeed = 15f;

float actualSpeed = frezzeBullet.Speed;

Assert.AreEqual(expectedSpeed, actualSpeed);

}

}

}

**Код программы для *TestDecoratedEffect*.*cs*:**

using System;

using GameLibrary.Effects;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace TestGameApp

{

[TestClass]

public class TestDecoratedEffect

{

DamageEffect damageEffect;

[TestInitialize]

public void InitalizeGun()

{

damageEffect = new DamageEffect();

}

[TestMethod]

public void TestSlowdownEffectDecorator()

{

SlowdownEffect slowdownEffect = new SlowdownEffect();

slowdownEffect.SetDecoratedEffect(damageEffect);

float expectedSpeed = 10f;

float actualSpeed = slowdownEffect.EffectTime;

Assert.AreEqual(expectedSpeed, actualSpeed);

}

[TestMethod]

public void TestFrezzeEffectDecorator()

{

FrezzeEffect frezzeEffect = new FrezzeEffect();

frezzeEffect.SetDecoratedEffect(damageEffect);

float expectedSpeed = 7f;

float actualSpeed = frezzeEffect.EffectTime;

Assert.AreEqual(expectedSpeed, actualSpeed);

}

}

}

**Код программы для *TestNerworkHandlers*.*cs*:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using NetworkLibrary;

using System.Threading;

namespace TestGameApp

{

[TestClass]

public class TestNetworkHandlers

{

private class TestNetworkUnit

{

private INetworkHandler \_networkHandler;

public int Counter { get; private set; } = 0;

public TestNetworkUnit(INetworkHandler handler){

handler.OnDataGot += (o) => Counter++;

\_networkHandler = handler;

}

public void UpdateData()

{

\_networkHandler.UpdateData(Counter);

}

}

[TestMethod]

public void TestGameObjectMovement()

{

var servertHandler = new Server();

var clientHandler = new Client("localhost");

var serverkUnit = new TestNetworkUnit(servertHandler);

var clientkUnit = new TestNetworkUnit(clientHandler);

serverkUnit.UpdateData();

Thread.Sleep(200);

clientkUnit.UpdateData();

serverkUnit.UpdateData();

Thread.Sleep(500);

clientkUnit.UpdateData();

serverkUnit.UpdateData();

Thread.Sleep(1000);

clientkUnit.UpdateData();

Assert.AreEqual(serverkUnit.Counter, clientkUnit.Counter);

}

}

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Руководство пользователя**

1. Введение.

Разработанное программное приложение предназначено для запуска на ОС не ниже *Windows* 7. Игра обладает минимальным порогом вхождения для пользователей.

Игровое приложение обладает следующим функционалом:

* возможность игры двоих пользователей на одном экране или на разных устройствах по сети;
* игровой уровень в виде лабиринта с элементами для взаимодействия и бонусами;
* передвижение игроков;
* генерация призов.

Для использования программного приложения пользователь должен быть ознакомлен со следующей информацией:

* настоящим руководством пользователя;
* правилами использования ЭВМ.

1. Назначение и условия применения.

Разработанное программное приложение предназначено для игры двух игроков на одном экране или на рахных устройствах по сети. Разработанное игровое приложение предназначено для развития внимания и реакции, а также для развлечения. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

* центральный процессор *Intel* *Core* 2 *Duo* *c* тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и монитора *SVGA* с разрешением не менее 1640 на 750 пикселей;
* операционная система *Windows* 7 и выше;
* 100 Мб оперативной памяти;
* скорость интернет соединения не ниже 150 килобайт в секунду.

1. Подготовка к работе.

Приложение запускается путём открытия файла *GameApplication*.*exe* от имени администратора. Также на компьютере должны быть стабильное подключение к Интернету и установленые драйвера для видеокарты, сетевой карты. Если все инструкции соблюдены, и приложение не выдаёт никаких сообщений об ошибках, значит, программа работает исправно.

1. Описание операций.

При запуске приложения открывается интерфейс, который выводит изображение с информацией об управлении в игре и кнопкой запуска игры. Первый игрок передвигается с помощью клавиш *W*, *A*, *S*, *D* и стреляет при нажатии на *Space*. Второй игрок передвигается с помощью клавиш *Up*, *Left*, *Down*, *Right* и стреляет при нажатии на *NumLock* *Enter*. При игре по сети управление соотвествует управлению за первого игрока для игры на одном экране. На старте приложения игроки не имеют возможности стрелять, пока не подберут оружие. Оружие генерируется случайным образом в лабиринте.

Первый игрок управляет синим персонажем, а второй игрок – красным. Цель каждого игрока заключается в сборе всех монет в лабиринте. В процессе игры можно передвигаться по вдоль горизонтали и по вертикали при столкновении с лестницей, а также падать вниз через платформы. Треснувшие стены можно сломать, если в них выстрелить, после чего в том месте можно пройти пока стена снова не появилась. При столкновении с монетой она исчезает и добавляется в счет игрока, подобравшего монету.

В процессе игры можно подбирать бонусное оружие трёх типов:

* оружие с уроном;
* замедляющее оружие;
* замораживающее оружие.

Уровни выбираются случайным образом. Информация о собранных монетах, текущем оружии и действующем эффекте отображается в информационных панелях по краям игрового окна.

1. Аварийные ситуации

Чтобы избежать ошибок при использовании программы, необходимо соблюдать порядок действий и условия пользования, описанные в пункте 3, данного руководства пользователя.

В случае непредвиденного «зависания» программы рекомендуется завершить процесс в диспетчере задач и запустить снова.

1. Рекомендации по освоению.

Заранее изучить работу с клавиатурой персонального компьютера. Запомнить расположение клавиш, необходимых для игры.

# ПРИЛОЖЕНИЕ В

(обязательное)

**Руководство** **программиста**

1. Назначения и условия применения программы.

Разработанное игровое приложение предназначено для игры между двумя игроками на одном компьютере или на разных устройствах по сети. Разработанное игровое приложение предназначено для развития внимания и реакции, а также для развлечения. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

* центральный процессор *Intel* *Core* 2 *Duo* *c* тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и монитора *SVGA* с разрешением не менее 1640 на 750 пикселей;
* операционная система *Windows* 7 и выше;
* 100 Мб оперативной памяти;
* скорость интернет-соединения не ниже 150 килобайт в секунду.

1. Характеристики программы.

Приложение необходимо запускать от имени администратора. Для запуска решения необходима среда разработки *Visual* *Studio* с установленными фреймворком .*NET* и библиотекой *SharpDX*.

1. Обращение к программе.

Приложение запускается путём открытия файла *GameApplication*.*exe*, находящегося в папке *CourceGame*.

1. Входные и выходные данные.

В данной программе в качестве входных данных используется ввод с клавиатуры кнопок управления игровым процессом. В качестве выходных выступает окно отображения игры.

1. Сообщения.

При окончании игры выводится победивший игрок. В процессе работы приложения игровая статистика выводится в информационные лейблы каждого игрока.

**ПРИЛОЖЕНИЕ Г**

(обязательное)

**Руководство системного программиста**

1. Общие сведения о программе.

Разработанное игровое приложение предназначено для игры между двумя игроками на одном компьютере или на разных устройствах по сети. Разработанное игровое приложение предназначено для развития внимания и реакции, а также для развлечения. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

* центральный процессор *Intel* *Core* 2 *Duo* *c* тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и монитора *SVGA* с разрешением не менее 1640 на 750 пикселей;
* операционная система *Windows* 7 и выше;
* 100 Мб оперативной памяти;
* скорость интернет-соединения не ниже 150 килобайт в секунду.

1. Структура программы.

Игровое приложение логически можно разбить на несколько составляющих: игровой движок, содержащий средства работы с графикой, непосредственно логика игровых объектов и игрового процесса, проект сетевого взаимодействия и графический интерфейс пользователя.

1. Настройка программы.

Приложение необходимо запускать от имени администратора. Для запуска решения необходима среда разработки *Visual* *Studio* с установленными фреймворком .*NET* и библиотекой *SharpDX*.

1. Проверка программы.

Для тестирования программного средства были реализованы модульные тесты. Для запуска отладки в среде разработки *Visual* *Studio* необходимо запустить выполнение модульных тестов. По окончанию процесса отладки будет выдан отчет о результатах тестирования приложения.

1. Дополнительные возможности.

Приложение является узконаправленным и не имеет дополнительных возможностей.

1. Сообщение системному программисту.

Проект игрового движка можно использовать для разработки других игр. Проект игровой логики можно дополнить, добавив новые лабиринты, бонусы и элементы лабиринта.

**ПРИЛОЖЕНИЕ Д**

(справочное)

**Внешний вид окон интерфейса программы**

Приложение запускается в одном окне. Окно рендеринга библиотеки *SharpDX* воспроизводится через элемент управления *WPF*, информационная статистика выводится в *WPF*. На рисунке Д.1 представлен вид окна пользовательского интерфейса при запуске приложения. На рисунке Д.2 представлен вид окна пользовательского интерфейса при выборе игры на одном экране. На рисунке Д.3 представлен вид окна пользовательского интерфейса при выборе игры по сети. На рисунке Д.4 представлен вид окна пользовательского интерфейса в момент игры.

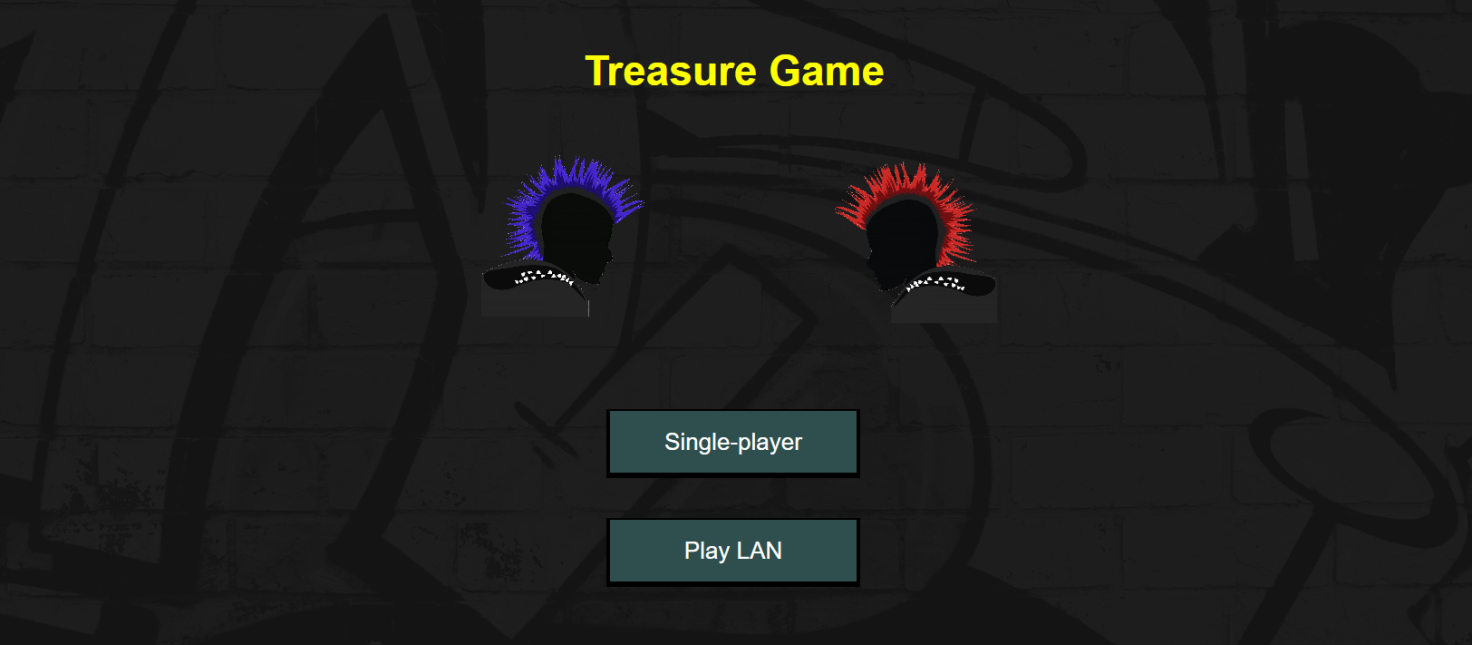


Рисунок Д.1 – Вид окна пользовательского интерфейса при запуске приложения

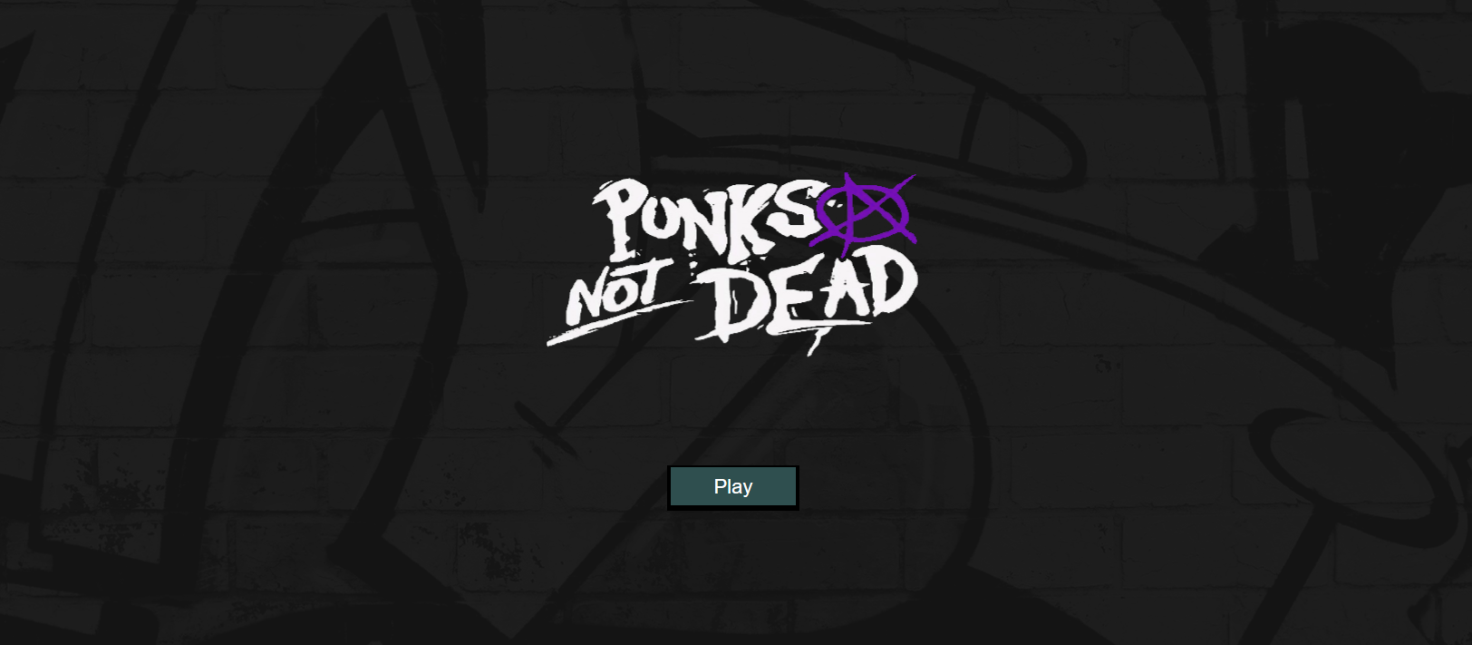


Рисунок Д.2 – Вид окна пользовательского интерфейса при выборе игры на одном экране

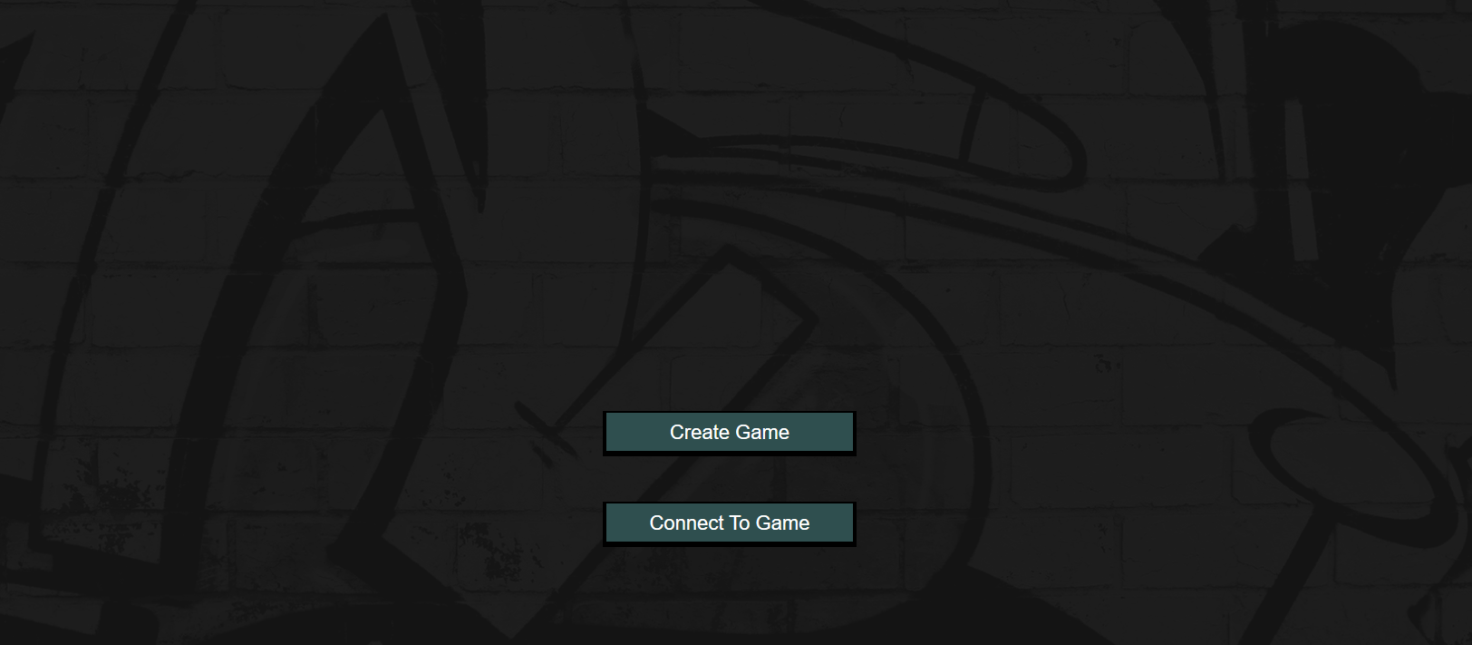


Рисунок Д.3 – Вид окна пользовательского интерфейса при выборе игры по сети

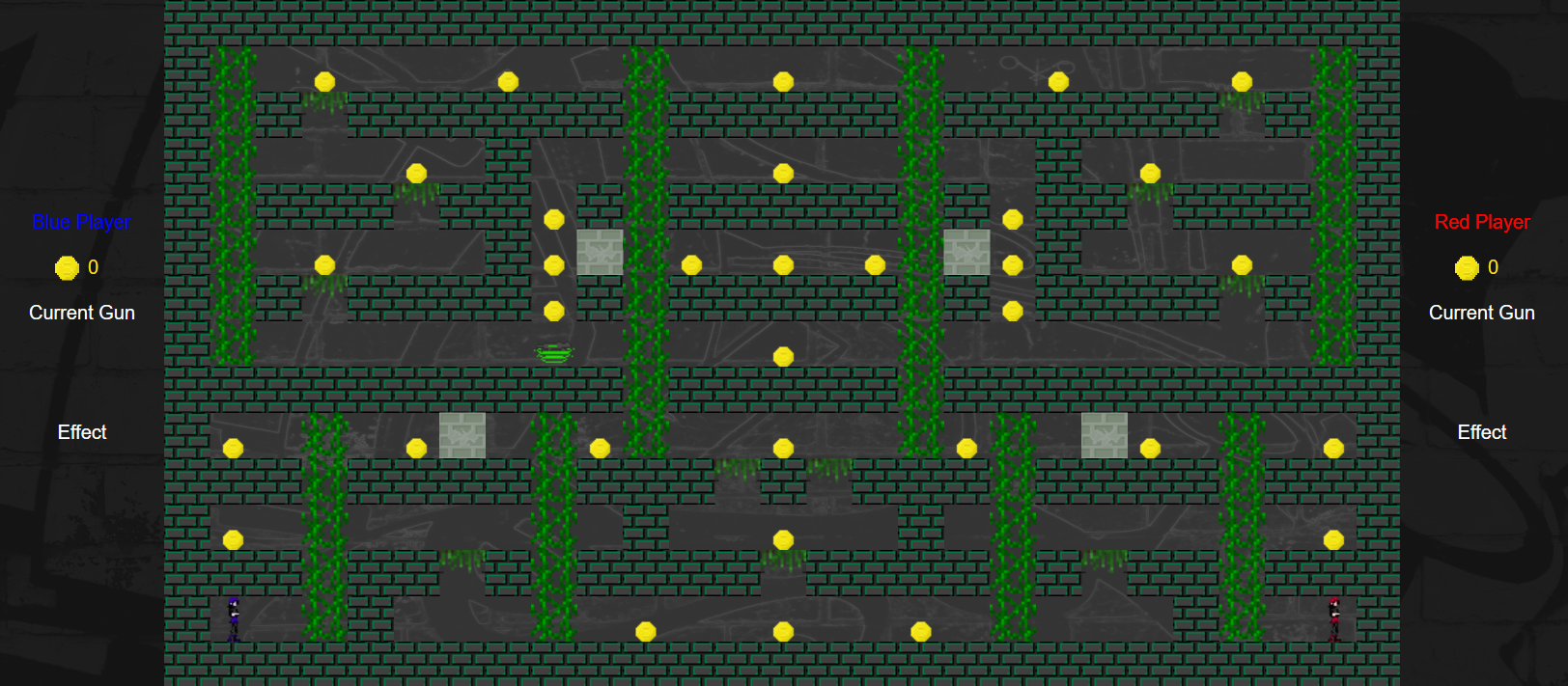


Рисунок Д.4 – Вид окна пользовательского интерфейса в момент игры

**ПРИЛОЖЕНИЕ Е**

(справочное)

**Результаты опытной эксплуатации**

Приложение функционирует при следующей минимальной конфигурации технических средств аппаратного обеспечения:

* центральный процессор *Intel* *Core* 2 *Duo* *c* тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и монитора *SVGA* с разрешением не менее 1640 на 750 пикселей;
* операционная система *Windows* 7 и выше;
* 100 Мб оперативной памяти;
* скорость интернет-соединения не ниже 150 килобайт в секунду.

Опытная эксплуатация проводилась в течение двух часов на персональных компьютерах разной конфигурации разными пользователями. В процессе эксплуатации проверялся запуск приложения, полное прохождение уровня двумя пользователями, синхронизация игрового времени, корректная передача и отображение данных, закрытие приложения.

Проводить работу над устранением проблем не требуется, так как в процессе эксплуатации не было выявлено ошибок и недостатков.

**ПРИЛОЖЕНИЕ Ж**

(обязательное)

**Схема использования паттерна «Декоратор»**