Содержание

ВВЕДЕНИЕ..........................................................................................................................4

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ............................................................................6

2 ПОСТАНОВКА ЗАДАЧИ..............................................................................................13

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА......................................................... 14

3.1 Создание игрового поля.......................................................................................14

3.2 Создание «класса» игрока...................................................................................14

3.3 Движения и коллизии главного персонажа........................................................15

3.4 Создание «класса» Menu..................................................................................... 18

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.............................................................................22

4.1 Начало работы......................................................................................................22

4.2 Управление........................................................................................................... 24

4.3 Игровой процесс....................................................................................................24

ЗАКЛЮЧЕНИЕ..................................................................................................................25

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ………………………………………..26

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ.................................................27

ПРИЛОЖЕНИЕ Б. БЛОК-СХЕМА МЕТОДА UPDATE КЛАССА KUB

# 

# ВВЕДЕНИЕ

**Платформер** (англ. platformer) — жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня. Прыжки по платформам могут быть частью игрового процесса игр других жанров. Например, основой игры может быть стрельба, но на многих уровнях важно ловко прыгать по платформам. Если элементы платформера и другого жанра используются в равной мере (особенно если есть элемент квеста или ролевой игры), игру часто называют «Action/Adventure», то есть «активные приключения».

Противники (называемые «монстрами» независимо от внешнего вида), всегда многочисленные и разнородные, обладают примитивным искусственным интеллектом, стремясь максимально приблизиться к игроку, либо не обладают им вовсе, совершая повторяющиеся действия. Соприкосновение с противником обычно отнимает жизненные силы у героя или вовсе убивает его. Иногда противник может быть нейтрализован либо прыжком ему на голову, либо из оружия, если им обладает герой.

Игры подобного жанра характеризуются нереалистичностью, рисованной мультяшной графикой. Героями таких игр обычно бывают мифические существа (к примеру: драконы, гоблины) или антропоморфные животные.

**Виды платформеров**:

Платформеры различаются по нескольким признакам:

По виду графики — двухмерные (2D) и трёхмерные (3D).

По свободе перемещения — линейные и свободные. В линейных персонаж должен пройти определённый путь с начала до конца уровня. В свободных платформерах персонаж может перемещаться по уровню без ограничений, и для победы часто нужно посетить разные места в любой последовательности.

По глубине — в некоторых платформерах персонаж не может отдаляться от экрана и всё время находится на одной заданной линии, а в других может углубляться и приближаться. Иногда такое перемещение полностью свободно, иногда это переключение между двумя-тремя линиями глубины. Трёхмерные платформеры, без глубины называют «2.5D».

**Эра двухмерной графики**

В 1985 году фирма Nintendo выпустила для приставки Nintendo Entertainment System революционный платформер Super Mario Bros. Игра была наполнена большими и сложными уровнями, и стала примером для последующих создателей игр, и даже сегодня многие люди считают ее одной из самых лучших видеоигр. Игра имела фантастическую популярность и продавалась огромными тиражами. Для многих людей она стала первым в их жизни платформером, а главный герой Mario стал символом фирмы Nintendo.

**Трехмерная эра**

YoFrankie! — платформер с трёхмерной графикой

Термин «трехмерный платформер» может обозначать или геймплей, включающий все три измерения, или использование трехмерных полигонов в реальном времени для отрисовки уровней и героев, или и то и другое. Самым ярким примером является Crash Bandicoot. Эта игра оставалось верной традиции двумерных платформеров и в ней использовались довольно плоские уровни, в конце которых располагалась игровая цель.

**Изометрические игры**

Вероятно, изометрические платформеры являются поджанром и двух-, и трехмерных платформеров. Они отображают трехмерную сцену с помощью двумерной графики, которая отображает мир с жестко ориентированной камеры без учета перспективы. И хотя изометрические платформеры не были первыми изометрическими играми, ранними примерами изометрических платформеров являются игра 1983 года CongoBongo, в мире аркадных автоматов и 3DAntAttack для ZXSpectrum, перенесенная в следующем году на Commodore 64.

**1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

Наиболее популярные игры:

1. SuperMarioBros
2. MarkoftheNinja
3. SuperMeatBoy
4. Braid Rayman
5. Origins
6. Limbo
7. Geometry Dash

**SuperMarioBros** - культовая видеоигра, выпущенная в 1985 году компанией Nintendo. Занесена в «Книгу рекордов Гиннесса» как самая продаваемая игра в истории.

Главными героями игры являются водопроводчик Марио и его брат Луиджи (в качестве игрового персонажа для второго игрока). Цель игры — пройти через Грибное королевство, ускользая или уничтожая солдат черепашьего Короля Купы (также известного как Боузер), чтобы спасти захваченную им в плен Принцессу.

**Механика игры:**

Марио (или Луиджи) атакует противников, прыгая на них сверху (такой способ убивает грибы Гумба и временно нейтрализует черепах Купа, заставляя их прятаться в свои панцири) или ударяя по платформе, на которой находится противник, снизу. «Испугавшихся» черепах можно использовать как оружие против других врагов, подтолкнув панцирь впереди себя: разогнавшись, он сметает всех на своем пути, но когда встречает преграду, меняет направление и может ударить самого Марио. Некоторые из труб являются проходом в бонусный уровень, подземелье, в котором можно найти небольшое количество монет, а заодно сократить путь до выхода с уровня. Также встречаются ростки, по которым Марио поднимается на облака, бонусный уровень, на котором большое количество монет и нет противников.

По пути Марио собирает монеты и бонусы, ударяя по блокам со знаком вопроса, а также выискивая секретные хранилища монет в кирпичных стенах. При наборе ста монет Марио получает дополнительную жизнь. За поверженных врагов начисляются очки, которые не приносят какой-либо практической пользы, а служат лишь для отражения мастерства игрока.

Среди бонусов может встретиться оранжевый гриб, взяв который, персонаж увеличивается в размерах, превращаясь в Супер Марио (англ. SuperMario). Если взять после этого цветок, то Супер Марио становится Огненным Марио (англ. FireMario) и получает возможность стрелять огненными шарами, то есть поражать противника на расстоянии.

Если Супер Марио или Огненный Марио дотрагиваются до противника, то они возвращаются в стадию обычного Марио. Если же противник ранит Марио или кончается время, выделенное для прохождения уровня, то игрок теряет одну жизнь и игра начинается заново (либо с начала уровня, либо с его середины).

Ещё один встречающийся в игре бонус — звезда. Заполучив её, Марио становится на некоторое время неуязвимым для врагов: противники погибают от одного его прикосновения. На некоторых уровнях встречаются также бонусы в виде зелёного гриба «1-up», который добавляет Марио одну жизнь (гриб «1-up» движется, и для того, чтобы его поймать, игроку приходится проявить сноровку).

Игра состоит из восьми миров по четыре уровня в каждом. В конце каждого четвёртого уровня, на мосту через озеро с лавой, Марио встречается с Королём Купой (англ. KingKoopa) — огромным черепаховидным драконом. В игре существуют три секретных перехода, называемые Зоной Перемещения (англ. WarpZone) и позволяющих быстро переместиться в другие миры.

Игра без минусов, которая на долгие годы задала высокую планку платформерам, и до сегодняшнего дня вполне играбельна.

**Современные платформеры:**

**MarkoftheNinja**

MarkoftheNinja - двухмерная видеоигра в жанре стелс-экшен, разработанная канадской студией KleiEntertainment для Xbox 360 и ПК, выпущенная в сервисах XboxLiveArcade и Steam.

Геймплей игры основан на аккуратном прохождении, избегании контактов с противниками и обезвреживании ловушек. Главный герой — ниндзя — умеет прятаться в вентиляции, канализационных люках и за предметами интерьера.

Основное оружие — меч-ниндзято, которым можно выполнить бесшумное убийство ничего не подозревающей жертвы и бамбуковые дротики, предназначенные для отвлечения и уничтожения источников света. Несмотря на наличие смертельного оружия, игра в равной степени ориентирована как на устранение всей охраны уровня, так и на бескровное прохождение с помощью отвлечения, манипулирования, а то и вообще проскальзывания мимо противников. В процессе прохождения покупается и открывается дополнительное снаряжение атакующего и отвлекающего действия. Практически за каждое действие (уничтожение или обход противника, прятание тела, выключение питания ловушек и так далее) начисляются очки, на которые можно открыть новые приёмы, приобрести новое или улучшить старое снаряжение.

Помимо этого, в MarkoftheNinja большую роль играют источники света и звука. Охранники совершенно слепы в темноте и не увидят вас даже на расстоянии пары метров перед собой. Безопасное передвижение по уровням обеспечивается исключительно перебежками из одного темного угла в другой. В лучших традициях жанра некоторые источники света можно тушить самому броском дротика. Правда, в этом случае создается ненужный шум, оповещающий охранников о вашем присутствии. Все действия игрока имеют определенный радиус звучания. Наш ниндзя может перемещаться медленно, но тихо или же бегать сломя голову, но при этом ставя на уши всю охрану. Однако иногда прохождению уровня способствует привлечение внимания. Громко потопав у какой-нибудь двери, можно заставить стоящего у нее часового отойти от своего поста, а в это время проскользнуть мимо него.

В общем, прохождение миссий в стелс-режиме требует серьезного планирования всех своих действий.

Очень красивая система боя, с замедлением времени.

Достоинства: стильная графика; продуманный стелс; вариативность прохождения.

Недостатки: простоватый сюжет.

**SuperMeatBoy**

SuperMeatBoy — компьютернаяинди-игра в жанре платформера, разработанная командой TeamMeat, состоящей из Эдмунда МакМиллена (англ.EdmundMcMillen) и Томми Рефенеса (англ.TommyRefenes).Выпущена 20 октября 2010 года для XboxLiveArcade, 30 октября для Windows через Steam и 2 декабря через Direct2Drive.

SuperMeatBoy рассказывает историю Мясного Парня, спасающего свою подругу, Пластырь-Девочку, от зловещего Доктора Зародыша

Игра получила положительные отзывы критиков, которые отметили ретро-стиль игры, а также её высокую сложность. Помимо этого, она получила различные награды, среди которых «Самая сложная игра 2010 года» и ещё множество разных наград, имела колоссальный коммерческий успех (“платиновый” статус).

Игроку необходимо, управляя главным героем и используя способность скользить по стенам, провести его через уровни, содержащие различные ловушки, такие как циркулярные пилы, иглы и соль, и достигнуть BandageGirl. Количество жизней неограничено, а после смерти MeatBoy сразу появляется в начале уровня. После прохождения уровня игроку показывают повтор, в котором одновременно отображаются последние 40 попыток прохождения (что очень сильно понравилось игрокам).

Игра состоит из пяти основных глав, по 20 уровней в каждой. Игрок должен пройти как минимум 17 из них (в любом порядке) для того чтобы открыть босса, после прохождения которого открывается следующая глава. Всего в игре доступно около 350 уровней.

Во время прохождения уровня игрок может обнаружить синий портал — warp-зону (англ. warpzone), при вхождении в которую игрок попадает в мини-игру, стилизованную под одну из ретро-игр (что тоже является очень интересным шагом разработчиков).

Наряду с MeatBoy в игре присутствуют разблокируемые персонажи из различных видеоигр. Каждый из них имеет свои уникальные способности, сходные со способностями своих аналогов в оригинальных играх.

В SuperMeatBoy игрок, выполняя второстепенные задания, может зарабатывать достижения.

Достоинства: хороший баланс сложности; здоровый «хардкор»; обилие дополнительных вкусностей; обаятельная стилистика.

Недостатки: иногда все-таки слишком трудно — лучше заранее купите дешевую клавиатуру, дабы пинать ее по комнате, матерясь и размахивая руками. Пока играешь время летит слишком быстро.

**Braid**

Braid («хитросплетение») — видеоигра в жанре платформер-головоломка, разработанная Джонатаном Блоу. Вышла в свет в 2008 году для приставки Xbox 360, распространяемая через сервис XboxLiveArcade примерно за 15 € (1200 баллов Microsoft). Версия для Windows вышла 10 апреля 2009 года, цифровая копия стоит примерно 15 $ и доступна для покупки через GreenHouse, Impulse, Steam и GamersGate. 20 мая также вышла версия для MacOSX и Linux. 12 ноября вышла на PlayStation 3.

Загадочная история, положенная в основу сюжета, рассказывает о том, как главный герой, Тим, пытается спасти принцессу от монстра. Подсказки, дающиеся в сюжете, позволяют давать истории разные метафорические интерпретации: начиная от простой разлуки с возлюбленной и заканчивая разработкой ядерной бомбы.

В поисках принцессы Тим путешествует по шести фантастическим мирам. Каждый мир соответствует комнате в доме Тима. На декорациях видны невзгоды, через которые прошёл Тим: женитьба на другой, война, учёба, алкоголизм.

Изначально мы видим простой платформер, мало чем отличающийся от SuperMarioBros.: Тим может ходить влево-вправо, лазать по сеткам и прыгать монстрам на головы. Но как только Тим случайно умрёт, провалившись в яму или столкнувшись с монстром, мы узнаём главное отличие Braid: время можно перематывать назад и вперёд (в версии для PC — клавишей ⇧Shift ). На этой возможности управлять временем и основан геймплей.

В каждом мире своя особенная механика что делает игру ещё ярче.

Игроку часто попадаются кусочки мозаики, однако на что они нужны — пока что непонятно: миры открываются, когда Тим проходит от входа до выхода, что совсем несложно.

Достоинства: занимательные головоломки; графика.

Недостатки: нулевая реиграбельность.

**Limbo**

Limbo (Лимб) — мультиплатформенная компьютерная игра в жанрах пазл-платформер с элементами survivalhorror, разработанная независимой датской студией Playdead. Проект был анонсирован в 2010 году для приставки Xbox 360 и выпущен 21 июля 2010 года. В течение последующих трех лет игра была портирована на такие платформы как Mac OS X, MicrosoftWindows, PlayStation 3, Linux, Vita и iOS.

Технически проект является 2D сайд-скроллером, в котором используется физический движок Box2D, позволяющий управлять объектами окружающей среды и персонажем.

Игра получила множество призов на игровых фестивалях и конференциях, в том числе за лучшую визуальную стилистику, техническое исполнение, звук, а также номинировалась на премию «Игра года». Проект был высоко оценён журналистами и игровыми обозревателями. Большинству рецензентов понравилось графическое исполнение, простое управление и внутриигровые загадки. Впрочем, сюжет был воспринят неоднозначно: некоторые критики находили в нём скрытый смысл, другие же считали его чересчур обрывочным и не дающим цельного представления об игре. Многие обозреватели также негативно восприняли короткую продолжительность проекта. Всего было продано более миллиона копий Limbo, принёсших разработчикам восемь миллионов долларов дохода.

Протагонистом игры является безымянный мальчик, который пробуждается посреди леса «на краю ада». Герой ищет свою пропавшую сестру и по ходу повествования встречает нескольких человек, которые либо нападают на него, либо убегают прочь, либо уже мертвы к моменту их обнаружения. Во время своего путешествия он видит маленькую девочку, окутанную сиянием, которая исчезает прежде, чем он успевает до неё добраться. Лес сменяется городскими постройками. Разбив стеклянный блок и покинув заброшенный завод, главный герой падает на землю. Пройдя по поляне, он подходит к сидящей девочке, которая испуганно встаёт. Этой сценой игра заканчивается.

Согласно жанровой принадлежности, игра является 2D платформером. Как и в большинстве проектов данного типа, игровой процесс строится на физическом взаимодействии главного героя с окружением. Игрок может идти вправо, влево, прыгать, взбираться на небольшие уступы, карабкаться по верёвкам или лестницам, а также толкать и тянуть предметы. Игра выполнена в стиле минимализма: всё окружение изображено в чёрно-белых, реже серых тонах, а окружающие звуки создают ощущение преследования и тревоги.

Тьма скрывает много опасностей: в ней могут находиться как медвежьи капканы, так и различные монстры, в том числе гигантский паук. Враждебными считаются черви, впивающиеся в голову главного героя и заставляющие его идти в одну сторону до тех пор, пока герой не выйдет на освещённое место и не избавится от червя, или пока носитель не погибнет. Кроме того, встречаются и агрессивно настроенные дети, которые могут бросать камни, горящие шины или метать дротики из духовых трубок.

Во второй половине проекта в геймплее игры преобладают механические ловушки, включающие в себя головоломки с электричеством, гравитацией и различными механизмами. Большинство ловушек незаметны до тех пор, пока они не сработают, зачастую демонстрируя гибель протагониста. Погибнуть можно от падения с большой высоты, от удара предметом, утопления, электрического разряда, пулемётных выстрелов и прочего. Данный способ прохождения именуется разработчиками как метод «проб и смертей» (англ. Trialanddeath). Каждый вид смерти сопровождается реалистичной анимацией отрывающихся конечностей, брызг крови и прочих особенностей расчленения, хотя дополнительно подключаемый фильтр вместо отображения характерных черт летального исхода в определённый момент затемняет экран[18][19]. Система автосохранения возвращает протагониста в ближайшее безопасное место[20]. Игровые достижения включают в себя обнаружение на локациях всех спрятанных яиц и завершение игры с пятью и менее количеством смертей.

Достоинства: Очень интересные головоломки; Мрачная графика вместе с музыкой создают глубокую атмосферу.

Недостатки: Иногда чересчур жестокая и не подходит для детей.

**RaymanOrigins**

RaymanOrigins — игра-платформер из игровой серии Rayman, рассказывающая о приключениях Рэймана и Глобокса. Впервые о предстоящей игре рассказали профильные игровые издания, информация была подтверждена официальным трейлером на E3 2010.

В игре присутствуют отсылки сразу к первой (Rayman) и двум последним (Rayman 2:TheGreatEscape, Rayman 3: HoodlumHavoc) каноническим играм, игра является сиквелом первой части. Мы узнаём, как появился Рэйман, Глобокс, также сюжет рассказывает о происхождении Мистера Дарка, откуда появились Малютки, как были пойманы Электуны, кто такой БабблДример, он же Полокус из Rayman 2, как созданная вселенная достигла кризиса, и как Рэймен спасает ее.

В RaymanOrigins под контроль игрока представлены Рэйман и Глобокс и двое Тинси - играть может до 4 игроков. Игра идёт в режиме 1080p при 60 кадрах в секунду. Игра была выпущена на Xbox 360, PlayStation 3 и Wii 15 ноября 2011 года. Позже она была выпущена также на MicrosoftWindows, PlayStationVita и Nintendo 3DS.

Всю историю виртуального мира можно будет узнать из вступительного ролика, откуда становится ясно, что произошло. В один день дружная компания спала и так храпела, что мертвецы-бабушки очень разозлились. Они напали на Рэймана, Глобокса, Малюток и Пузыря-Сновидца. Кошмары одержали верх над героями. Но Рэйман исправил положение, и теперь он должен собрать электунов и спасти Луговину Снов! А в этом ему помогают его друзья: Глобокс, Малютки, муха Мёрфи, москиты и нимфа Бетилла. Игру по сюжету можно разделить на три части: 1. Спасение нимф. 2. Спасение королей и уничтожение Небесного Города Волшебника. 3. Прохождение Земли Живых Мертвецов.

Путь Рэймана пролегает через множество красивых и интересных уровней:через Тарабарские джунгли, пустыню Диджириду, Лакомую землю, Море прозорливости и Хмурные облака. Главной задачей героя в первой половине является нахождение и освобождение фей, а во второй - королей этих земель. На уровне герой может подобрать множество бонусов: монетки, электуны и маленькие светлячки – люмы.

Достоинства: динамичный и затягивающий геймплей; дизайн уровней; в меру высокая сложность.

Недостатки: система сохранений; отсутствие нормального кооператива.

**Geometry Dash**

Geometry Dash - это увлекательный геометрический платформер, в котором вам предстоит поиграть за квадратного персонажа. Задача игрока заключается в том, чтобы проходить сложные и заумные уровни, полные самых разных препятствий и ловушек. Управление в игре настолько простое, насколько это вообще возможно. Сам персонаж двигается по уровню самостоятельно, пользователю остается только вовремя подпрыгивать, касаясь любого места на экране. На словах это кажется довольно просто, но на деле - отнюдь. Geometry Dash требует от игрока не только внимательности, но и сноровки. Ошибившись в прыжке всего на долю секунды вы, с большой вероятностью, отправите своего персонажа в пропасть или ловушку. В общем, мы можем смело говорить о том, что перед нами действительно сложная и требовательная игра.

   Основная цель Geometry Dash заключается в том, чтобы продвинуться по предложенному уровню как можно дальше. Игра будет измерять пройденное вами расстояние, после чего фиксировать его в таблице результатов. Отдельно стоит похвалить музыкальное сопровождение, которое очень точно подчеркивает общую атмосферу и делает игровой процесс еще боле динамичным и захватывающим.

**2 ПОСТАНОВКА ЗАДАЧИ**

После анализа достоинств и недостатков вышеперечесленных игр было принято решения о создании платформера, который не будет загружен лишними элементами, отвлекающими от основного игрового процесса.

Была поставлена задача разработать платформер. Его суть такова - перепрыгивать кубиком через препятствия, забираясь на блоки, и ни в коем случае нельзя ошибаться - придется начать с самого начала! Персонаж - кубик - будет передвигаться по горизонтали с постоянной скоростью, а также по вертикали (кнопками «W» или «Space»). На карте будут присутствовать различные препятствия. Для прохождения уровня необходимо будет добраться до конца уровня, ни разу не задев препятствие.

**3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**3.1 Создание игрового поля**

Игровое поле состоит из спрайтов размером 32х32 пикселя. Поле считывается из файла Level1.tmx (Level2.tmx) и происходит отрисовка карты на форме. Код считывания объектов карты из файла и отрисовки карты на форме находится в файле level.cpp (Приложение А, страница 33).

Также, перед прорисовкой карты, отрисовывается задний фон.

Все игровое поле вместе с информацией, расположенной на нем, и объектом игрока отрисовывается в методе draw класса Kub:

void Kub::draw(RenderWindow &window, Level &lvl, Text text, Sprite background, float time, Text textLvl)

{

window.draw(sprite);

lvl.Draw(window);

window.display();

window.draw(sprite);

update(time);

window.clear();

window.draw(background);

std::ostringstream playerScoreString, levelNumString;

levelNumString << levelNum;

playerScoreString << playerDeath;

text.setString("Attempt " + playerScoreString.str());

textLvl.setString("Level " + levelNumString.str());

window.draw(text);

window.draw(textLvl);

}

**3.2 Создание «класса» игрока**

Для удобства создания игры было принято решение использовать разработку методами ООП (объектно – ориентированное программирование). Для этого были созданы различные классы, например, Kub, Menu и др. Класс главного персонажа:

class Kub

{

public:

Kub(String F, Level &lev, float X, float Y, float W, float H);

~Kub();

void update(float time);

float getplayercoordinateX();

float getplayercoordinateY();

void control(float time);

void checkCollision(float Dx, float Dy);

FloatRect getRect();

Object checkFinish(Object player, Object finish, Level &lvl, sf::RenderWindow &window, sf::RenderWindow &windowMenu);

void draw(RenderWindow &window, Level &lvl, Text text, Sprite background, float time, Text textLvl);

private:

float x, y = 0;

int levelNum = 1;

std::vector<Object> obj;

float w, h, dx, dy;

bool onGround, life = true;

int playerDeath, upDown = 1;

Music music;

String File;

Image image;

Texture texture;

Sprite sprite;

};

В данном классе представлены поля, необходимые для передвижения главного персонажа по игровому полю, жизнь, количество набранных очков и др. Методы данного класса реализуют передвижение главного персонажа по игровому полю и столкновения с другими объектами.

**3.3 Движения и коллизии главного персонажа**

Движение главного персонажа осуществляется сменой его координат.

Метод control класса Kub определяет, нажата или не нажата какая-нибудь из клавиш, с помощью которых осуществляется управление персонажем. Этот метод вызывается из метода update.

void Kub::control(float time)

{

if (((Keyboard::isKeyPressed(Keyboard::W)) || (Keyboard::isKeyPressed(Keyboard::Space))) && onGround)

{

dy = -0.7 \* upDown;

dx = 0.22;

onGround = false;

}

}

Если нажата клавиша W или клавиша Space, тогда метод onGround становится равным False, что означает, что игрок подпрыгнул, и изменяются значения ускорений, за которые отвечают поля dx и dy класса Kub. После чего продолжает обрабатываться метод update, в котором идет изменение координат x и y игрока.

void Kub::update(float time)

{

control(time);

x += dx\*time;

checkCollision(dx, 0);

if (!life)

{

playerDeath++;

life = true;

}

y += dy\*time;

sprite.setPosition(x, y);

if (!onGround)

{

dy = dy + 0.0028\*time\* upDown;

}

onGround = false;

checkCollision(0, dy);

}





Блок-схема 1 – «Движение и проверка столкновений игрока»

Во время выполнение процедуры update идет проверка на столкновения с другими объектами (проверка на коллизии). Эти проверки выполняют процедуры checkCollision и checkFinish.

void Kub::checkCollision(float Dx, float Dy)

{

for (int i = 0; i < obj.size(); i++)

if (getRect().intersects(obj[i].rect))

{

if (obj[i].name == "solid")

{

if ((Dy \* upDown) > 0)

{

y = obj[i].rect.top - h \* upDown;

dy = 0;

onGround = true;

}

if (Dx > 0)

{

x = obj[i].rect.left - w;

x = 32;

y = 250;

life = false;

music.stop();

music.play();

upDown = 1;

}

}

else

{

if (obj[i].name == "death")

{

x = obj[i].rect.left - w; x = 32; y = 250;

life = false;

music.stop();

music.play();

upDown = 1;

}

else

{

if (obj[i].name == "down")

{

upDown \*= -1;

x = 512; y = 416;

}

}

}

}

}

В процедуре checkCollision идет проверка на столкновения объекта игрока с объектами, расположенными на игровом поле. Различные объекты на поле имеют различные свойства, по которым определяется, что происходит с объектом игрока. Так же дальнейшее поведение объекта игрока с объектами поля зависит и от того, как именно они столкнулись, т.е. игрок напрыгнул на объект либо просто врезался в него сбоку.

Object Kub::checkFinish(Object player, Object finish, Level &lvl, sf::RenderWindow &window, sf::RenderWindow &windowMenu)

{

String levelFile, musicFile;

if (x > finish.rect.left)

{

levelNum++;

if (levelNum < 3)

{

lvl.ClearObjects();

lvl.ClearLayers();

levelFile = "Levels/level" + std::to\_string(levelNum);

levelFile += ".tmx";

lvl.LoadFromFile(levelFile);

player = lvl.GetObj("player");

finish = lvl.GetObj("finish");

musicFile = "music/musicLevel" + std::to\_string(levelNum) + ".ogg";

music.openFromFile(musicFile);

music.play();

x = player.rect.left;

y = player.rect.top;

playerDeath = 0;

upDown = 1;

obj.clear();

obj = lvl.GetAllObjects();

}

else

{

window.close();

windowMenu.setVisible(true);

}

}

return finish;

}

Как говорилось раньше, для того, чтобы пройти уровень необходимо добраться до его конца. В процедуре checkFinish идет проверка, не добрался ли игрок до объекта, отвечающего за завершение уровня. Если условие истинно, тогда начинается загрузка следующего уровня с его объектами, музыка уровня, а также количество попыток игрока пройти предыдущий уровень обнуляется со сменой уровня. После того, как игрок пройдет второй уровень, он автоматически вернется в главное меню, где сможет заново начать игру с самого начала.

**3.4 Создание «класса» Menu**

Для отображения меню в игре был создан класс Menu со своими методами и полями:

class Menu

{

public:

Menu(float width, float height, sf::String FileName, int MAX\_NUMBER\_OF\_ITEMS);

~Menu();

void draw(sf::RenderWindow &window, int MAX\_NUMBER\_OF\_ITEMS);

void MoveUp();

void MoveDown(int MAX\_NUMBER\_OF\_ITEMS);

int GetPressedItem() { return selectedItemIndex; }

private:

int selectedItemIndex;

sf::Image image[3];

sf::Texture texture[3];

sf::Sprite sprite[3];

};

В данном классе задаются размеры игрового меню, количество кнопок.

Отрисовка кнопок меню осуществляется с помощью метода draw данного класса:

void Menu::draw(sf::RenderWindow &window, int MAX\_NUMBER\_OF\_ITEMS)

{

for (int i = 0; i < MAX\_NUMBER\_OF\_ITEMS; i++)

window.draw(sprite[i]);

}



Блок-схема 2 – «Отрисовка кнопок Меню»

Контролем кнопками управляют методы MoveUp, MoveDown и GetPressedItem.

Методы MoveUp и MoveDown отвечают за подсветку выбранной кнопки:

void Menu::MoveUp()

{

sf::Color color;

if (selectedItemIndex - 1 >= 0)

{

color = sprite[selectedItemIndex - 1].getColor();

sprite[selectedItemIndex].setColor(color);

selectedItemIndex--;

sprite[selectedItemIndex].setColor(sf::Color::Red);

}

}



Блок-схема 3 – «Перемещение курсора вверх»

void Menu::MoveDown(int MAX\_NUMBER\_OF\_ITEMS)

{

sf::Color color;

if (selectedItemIndex + 1 < MAX\_NUMBER\_OF\_ITEMS)

{

color = sprite[selectedItemIndex + 1].getColor();

sprite[selectedItemIndex].setColor(color);

selectedItemIndex++;

sprite[selectedItemIndex].setColor(sf::Color::Red);

}

}



Блок-схема 4 – «Перемещение курсора вниз»

Выбранная кнопка подсвечивается красным цветом.

Метод GetPressedItem возвращает номер выбранной кнопки. Этот метод вызывается по нажатии клавиши Enter, чтобы определить, какая кнопка была нажата и что выполнять в дальнейшем.

# 

# **4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

# **4.1 Начало работы**

После запуска приложения открывается окно главного меню (Рисунок 2).



Рисунок 2 – «Главное меню»

В этом окне имеется три кнопки: «Start», «Help», «Exit».

При нажатии на кнопку «Exit» программа закрывается. Если нажать на кнопку «Help», то появиться вспомогательное окно, в котором будет указана Ваша задача и управление (Рисунок 3).

# **C:\Users\Alex\Desktop\2 курс\ЯП\kurs_proekt_2\help.png**

Рисунок 3 – «Вспомогательное окно»

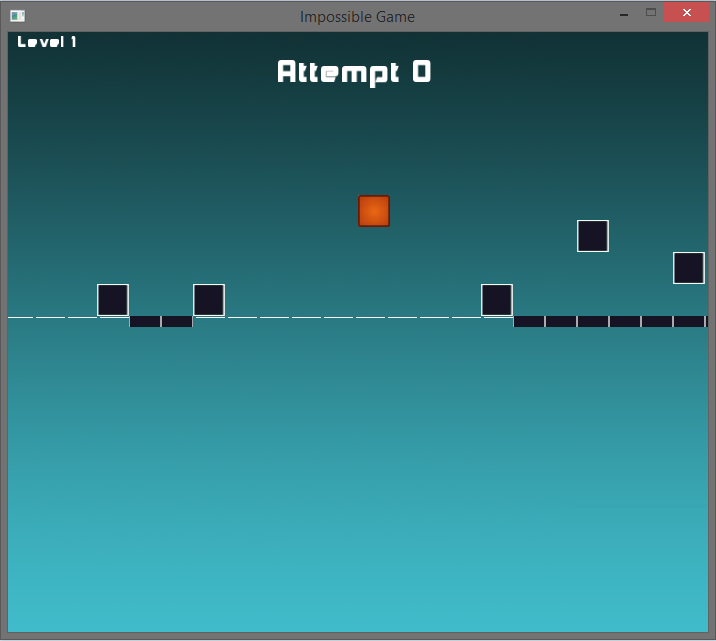
При нажатии кнопки «Start» открывается игровое поле (Рисунок 4).

Рисунок 4 – «Игровое поле приложения»

На игровом поле (Рисунок 4) сверху отображены количество попыток (Attempt) и номер уровня (Level 1).

**4.2 Управление**

Клавиша W/Space – прыжок главного персонажа;

**4.3 Игровой процесс**

Необходимо перепрыгивать кубиком через препятствия, забираясь на блоки, и ни в коем случае нельзя ошибаться – придется начать с самого начала! Следующий уровень открывается после прохождения предыдущего.

# **ЗАКЛЮЧЕНИЕ**

Была разработана игра Impossible Game, жанра платформер, в которой игрок сможет передвигаться по платформам с помощью клавиатуры.

Приложение было реализовано с использованием кроссплатформенной мультимедиа библиотеки SFML. При разработке активно использовалось программное средство контроля версий – GitHub.

Для повышения качества разрабатываемого продукта использовался ручной метод тестирования.

Игровое приложение обладает интуитивным управлением, что упрощает его использование для рядового пользователя. Со временем данное приложение будет совершенствоваться: будет добавлена таблица рекордов, будут добавлены новые типы препятствий, ловушки и различные бонусы.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Г. Шилдт «Полный справочник по С++»;
2. ГОСТ 19.701–90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – Введ. 1992–01–01. – М. : Изд-во стандартов, 1991.
3. <http://ru.stackoverflow.com/>
4. Б. Эккель, Ч. Эллисон «Философия С++. Том 1»
5. Р. Лафоре «Объектно – ориентированное программирование в С++»;
6. <https://code-live.ru/tag/cpp-manual/>
7. Чистый код / Роберт Мартин . Питер, 2014
8. <http://www.sfml-dev.org/tutorials/2.3/>
9. http://www.programmersclub.ru/main/
10. cyberforum.ru

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД**

**main.cpp**

#include "Menu.h"

#include "PlayWindow.h"

void main()

{

HWND hWnd = GetConsoleWindow();

ShowWindow(hWnd, SW\_HIDE);

window.setVisible(false);

sf::Font font;

font.loadFromFile("akashi.ttf");

sf::Text text("", font, 50);

text.setColor(sf::Color::White);

text.setStyle(sf::Text::Bold);

sf::RenderWindow windowMenu(sf::VideoMode(600, 600), "Impossible Game", sf::Style::Close); //800 640

Menu menu(windowMenu.getSize().x, windowMenu.getSize().y, "images/btn", 3);

while (windowMenu.isOpen())

{

sf::Event event;

while (windowMenu.pollEvent(event))

{

switch(event.type)

{

case sf::Event::KeyReleased:

switch (event.key.code)

{

break;

case sf::Keyboard::Up:

menu.MoveUp();

break;

case sf::Keyboard::Down:

menu.MoveDown(3);

break;

case sf::Keyboard::Return:

switch (menu.GetPressedItem())

{

case 0:

CreatePlayWindow(windowMenu);

break;

case 1:

{

sf::Text textH("", font, 50);

textH.setColor(sf::Color::White);

textH.setStyle(sf::Text::Bold);

Texture t;

t.loadFromFile("images/help.png");

Sprite help;

help.setTexture(t);

help.setPosition(10, 250);

sf::RenderWindow helpWindow(sf::VideoMode(500, 500), "Impossible Game", sf::Style::None);

while (helpWindow.isOpen())

{

windowMenu.setVisible(false);

sf::Event eventHelp;

while (helpWindow.pollEvent(eventHelp))

{

if ((eventHelp.type == sf::Event::Closed) || (eventHelp.key.code == sf::Keyboard::Escape))

{

helpWindow.close();

windowMenu.setVisible(true);

}

}

helpWindow.clear();

helpWindow.draw(help);

textH.setCharacterSize(50);

textH.setString("Help");

textH.setPosition(helpWindow.getSize().x - 480, 0);

helpWindow.draw(textH);

textH.setCharacterSize(20);

textH.setColor(Color(133, 133, 133));

textH.setString("The bottom line is that - jump over the \ncube through obstacles, \nclimb on the blocks, and in any case \nnot mistaken - \nwill start from the beginning!");

textH.setPosition(10, 80);

helpWindow.draw(textH);

textH.setCharacterSize(15);

textH.setColor(Color::White);

textH.setString("Press ESC to return to the menu");

textH.setPosition(10, helpWindow.getSize().y - 50);

helpWindow.draw(textH);

helpWindow.display();

}

}

break;

case 2:

windowMenu.close();

break;

}

break;

}

break;

case sf::Event::Closed:

windowMenu.close();

break;

}

}

windowMenu.clear();

menu.draw(windowMenu, 3);

text.setCharacterSize(50);

text.setString("Impossible Game");

text.setPosition(windowMenu.getSize().x - 520, 0);

windowMenu.draw(text);

text.setCharacterSize(15);

text.setString("Created by Alex");

text.setPosition(10, window.getSize().y - 30);

windowMenu.draw(text);

windowMenu.display();

}

}

**view.h**

#include <SFML/Graphics.hpp>

namespace

{

sf::View view;

}

void getplayercoordinateforview(float x, float y);

**level.h**

#ifndef LEVEL\_H

#define LEVEL\_H

#include <string>

#include <vector>

#include <map>

#include <SFML/Graphics.hpp>

#include <iostream>

#include "TinyXML/tinyxml.h"

struct Object

{

int GetPropertyInt(std::string name);//номер свойства объекта в нашем списке

float GetPropertyFloat(std::string name);

std::string GetPropertyString(std::string name);

std::string name;//объявили переменную name типа string

std::string type;//а здесь переменную type типа string

sf::Rect<float> rect;//тип Rect с нецелыми значениями

std::map<std::string, std::string> properties;//создаём ассоциатиный массив. ключ - строковый тип, значение - строковый

sf::Sprite sprite;//объявили спрайт

};

struct Layer//слои

{

int opacity;//непрозрачность слоя

std::vector<sf::Sprite> tiles;//закидываем в вектор тайлы

};

class Level//главный класс - уровень

{

public:

bool LoadFromFile(std::string filename);//возвращает false если не получилось загрузить

Object GetObj(std::string name);

std::vector<Object> GetObjects(std::string name);//выдаем объект в наш уровень

std::vector<Object> GetAllObjects();//выдаем все объекты в наш уровень

void Draw(sf::RenderWindow &window);//рисуем в окно

sf::Vector2i GetTileSize();//получаем размер тайла

void ClearObjects();

void ClearLayers();

private:

int width, height, tileWidth, tileHeight;//в tmx файле width height в начале,затем размер тайла

int firstTileID;//получаем айди первого тайла

sf::Rect<float> drawingBounds;//размер части карты которую рисуем

sf::Texture tilesetImage;//текстура карты

std::vector<Object> objects;//массив типа Объекты, который мы создали

std::vector<Layer> layers;

};

#endif

**Menu.h**

#pragma once

#include <SFML\Graphics.hpp>

class Menu

{

public:

Menu(float width, float height, sf::String FileName, int MAX\_NUMBER\_OF\_ITEMS);

~Menu();

void draw(sf::RenderWindow &window, int MAX\_NUMBER\_OF\_ITEMS);

void MoveUp();

void MoveDown(int MAX\_NUMBER\_OF\_ITEMS);

int GetPressedItem() { return selectedItemIndex; }

private:

int selectedItemIndex;

sf::Image image[3];

sf::Texture texture[3];

sf::Sprite sprite[3];

};

**Kub.h**

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/Audio.hpp>

#include <iostream>

#include "view.h"

#include "level.h"

#include <vector>

#include <sstream>

#include <Windows.h>

using namespace sf;

class Kub

{

public:

Kub(String F, Level &lev, float X, float Y, float W, float H);

~Kub();

void update(float time);

float getplayercoordinateX();

float getplayercoordinateY();

void control(float time);

void checkCollision(float Dx, float Dy);

FloatRect getRect();

Object checkFinish(Object player, Object finish, Level &lvl, sf::RenderWindow &window, sf::RenderWindow &windowMenu);

void draw(RenderWindow &window, Level &lvl, Text text, Sprite background, float time, Text textLvl);

private:

float x, y = 0;

int levelNum = 1;

std::vector<Object> obj;

float w, h, dx, dy;

bool onGround, life = true;

int playerDeath, upDown = 1;

Music music;

String File;

Image image;

Texture texture;

Sprite sprite;

};

**PlayWindow.h**

#include "Kub.h"

namespace

{

sf::RenderWindow window(sf::VideoMode(700, 600), "Impossible Game", sf::Style::Close);

}

void CreatePlayWindow(sf::RenderWindow &menuWindow)

{

HWND hWnd = GetConsoleWindow();

ShowWindow(hWnd, SW\_HIDE);

String levelFile;

Level lvl;

lvl.LoadFromFile("Levels/level1.tmx");

Font font;

font.loadFromFile("akashi.ttf");

Text text("", font, 30);

text.setColor(Color::White);

text.setStyle(sf::Text::Bold);

Text textLvl("", font, 30);

textLvl.setColor(Color::White);

textLvl.setStyle(sf::Text::Bold);

if (!window.isOpen())

window.create(sf::VideoMode(700, 600), "Impossible Game", sf::Style::Close);

else

window.setVisible(true);

view.reset(sf::FloatRect(0, 0, 700, 600));

Texture t;

t.loadFromFile("images/background.jpg");

Sprite background;

background.setTexture(t);

background.setTextureRect(sf::IntRect(0, 0, 14000, 1000));

Object player = lvl.GetObj("player");

Object finish = lvl.GetObj("finish");

Kub kub("kub.jpg", lvl, player.rect.left, player.rect.top, 32, 32);

Clock clock;

while (window.isOpen())

{

menuWindow.setVisible(false);

float time = clock.getElapsedTime().asMicroseconds();

clock.restart();

time = time / 800;

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

{

menuWindow.setVisible(true);

window.close();

}

}

finish = kub.checkFinish(player, finish, lvl, window, menuWindow);

float x = kub.getplayercoordinateX();

float y = kub.getplayercoordinateY();

float tempX = x; float tempY = y;

if (y > 300) tempY = 400;

else

if (y < 500) tempY = 400;

if (x < 400) tempX = 400;

else

if (x > 2700) tempX = 2700;

view.setCenter(tempX, tempY);

window.setView(view);

text.setPosition(view.getCenter().x - 80, view.getCenter().y - 280);

textLvl.setPosition(view.getCenter().x - 60, view.getCenter().y + 250);

kub.draw(window, lvl, text, background, time, textLvl);

}

}

**view.cpp**

#include "view.h"

void getplayercoordinateforview(float x, float y)

{

float tempX = x; float tempY = y;

if (y > 300) tempY = 400;

else

if (y < 500) tempY = 400;

if (x < 300) tempX = 400;

else

if (x > 2000) tempX = 2000;

view.setCenter(tempX, tempY);

}

**level.cpp**

#include "level.h"

int Object::GetPropertyInt(std::string name)//возвращаем номер свойства в нашем списке

{

return atoi(properties[name].c\_str());

}

float Object::GetPropertyFloat(std::string name)

{

return strtod(properties[name].c\_str(), NULL);

}

std::string Object::GetPropertyString(std::string name)//получить имя в виде строки.вроде понятно

{

return properties[name];

}

bool Level::LoadFromFile(std::string filename)//двоеточия-обращение к методам класса вне класса

{

TiXmlDocument levelFile(filename.c\_str());//загружаем файл в TiXmlDocument

// загружаем XML-карту

if (!levelFile.LoadFile())//если не удалось загрузить карту

{

std::cout << "Loading level \"" << filename << "\" failed." << std::endl;//выдаем ошибку

return false;

}

// работаем с контейнером map

TiXmlElement \*map;

map = levelFile.FirstChildElement("map");

// пример карты: <map version="1.0" orientation="orthogonal"

// width="10" height="10" tilewidth="34" tileheight="34">

width = atoi(map->Attribute("width"));//извлекаем из нашей карты ее свойства

height = atoi(map->Attribute("height"));//те свойства, которые задавали при работе в

tileWidth = atoi(map->Attribute("tilewidth"));//тайлмап редакторе

tileHeight = atoi(map->Attribute("tileheight"));

// Берем описание тайлсета и идентификатор первого тайла

TiXmlElement \*tilesetElement;

tilesetElement = map->FirstChildElement("tileset");

firstTileID = atoi(tilesetElement->Attribute("firstgid"));

// source - путь до картинки в контейнере image

TiXmlElement \*image;

image = tilesetElement->FirstChildElement("image");

std::string imagepath = image->Attribute("source");

// пытаемся загрузить тайлсет

sf::Image img;

if (!img.loadFromFile(imagepath))

{

std::cout << "Failed to load tile sheet." << std::endl;//если не удалось загрузить тайлсет-выводим ошибку в консоль

return false;

}

img.createMaskFromColor(sf::Color(255, 255, 255));//для маски цвета.сейчас нет маски

tilesetImage.loadFromImage(img);

tilesetImage.setSmooth(false);//сглаживание

// получаем количество столбцов и строк тайлсета

int columns = tilesetImage.getSize().x / tileWidth;

int rows = tilesetImage.getSize().y / tileHeight;

// вектор из прямоугольников изображений (TextureRect)

std::vector<sf::Rect<int>> subRects;

for (int y = 0; y < rows; y++)

for (int x = 0; x < columns; x++)

{

sf::Rect<int> rect;

rect.top = y \* tileHeight;

rect.height = tileHeight;

rect.left = x \* tileWidth;

rect.width = tileWidth;

subRects.push\_back(rect);

}

// работа со слоями

TiXmlElement \*layerElement;

layerElement = map->FirstChildElement("layer");

while (layerElement)

{

Layer layer;

// если присутствует opacity, то задаем прозрачность слоя, иначе он полностью непрозрачен

if (layerElement->Attribute("opacity") != NULL)

{

float opacity = strtod(layerElement->Attribute("opacity"), NULL);

layer.opacity = 255 \* opacity;

}

else

{

layer.opacity = 255;

}

//  контейнер <data>

TiXmlElement \*layerDataElement;

layerDataElement = layerElement->FirstChildElement("data");

if (layerDataElement == NULL)

{

std::cout << "Bad map. No layer information found." << std::endl;

}

//  контейнер <tile> - описание тайлов каждого слоя

TiXmlElement \*tileElement;

tileElement = layerDataElement->FirstChildElement("tile");

if (tileElement == NULL)

{

std::cout << "Bad map. No tile information found." << std::endl;

return false;

}

int x = 0;

int y = 0;

while (tileElement)

{

int tileGID = atoi(tileElement->Attribute("gid"));

int subRectToUse = tileGID - firstTileID;

// Устанавливаем TextureRect каждого тайла

if (subRectToUse >= 0)

{

sf::Sprite sprite;

sprite.setTexture(tilesetImage);

sprite.setTextureRect(subRects[subRectToUse]);

sprite.setPosition(x \* tileWidth, y \* tileHeight);

sprite.setColor(sf::Color(255, 255, 255, layer.opacity));

layer.tiles.push\_back(sprite);//закидываем в слой спрайты тайлов

}

tileElement = tileElement->NextSiblingElement("tile");

x++;

if (x >= width)

{

x = 0;

y++;

if (y >= height)

y = 0;

}

}

layers.push\_back(layer);

layerElement = layerElement->NextSiblingElement("layer");

}

// работа с объектами

TiXmlElement \*objectGroupElement;

// если есть слои объектов

if (map->FirstChildElement("objectgroup") != NULL)

{

objectGroupElement = map->FirstChildElement("objectgroup");

while (objectGroupElement)

{

//  контейнер <object>

TiXmlElement \*objectElement;

objectElement = objectGroupElement->FirstChildElement("object");

while (objectElement)

{

// получаем все данные - тип, имя, позиция, и тд

std::string objectType;

if (objectElement->Attribute("type") != NULL)

{

objectType = objectElement->Attribute("type");

}

std::string objectName;

if (objectElement->Attribute("name") != NULL)

{

objectName = objectElement->Attribute("name");

}

int x = atoi(objectElement->Attribute("x"));

int y = atoi(objectElement->Attribute("y"));

int width, height;

sf::Sprite sprite;

sprite.setTexture(tilesetImage);

sprite.setTextureRect(sf::Rect<int>(0, 0, 0, 0));

sprite.setPosition(x, y);

if (objectElement->Attribute("width") != NULL)

{

width = atoi(objectElement->Attribute("width"));

height = atoi(objectElement->Attribute("height"));

}

else

{

width = subRects[atoi(objectElement->Attribute("gid")) - firstTileID].width;

height = subRects[atoi(objectElement->Attribute("gid")) - firstTileID].height;

sprite.setTextureRect(subRects[atoi(objectElement->Attribute("gid")) - firstTileID]);

}

// экземпляр объекта

Object object;

object.name = objectName;

object.type = objectType;

object.sprite = sprite;

sf::Rect <float> objectRect;

objectRect.top = y;

objectRect.left = x;

objectRect.height = height;

objectRect.width = width;

object.rect = objectRect;

// "переменные" объекта

TiXmlElement \*properties;

properties = objectElement->FirstChildElement("properties");

if (properties != NULL)

{

TiXmlElement \*prop;

prop = properties->FirstChildElement("property");

if (prop != NULL)

{

while (prop)

{

std::string propertyName = prop->Attribute("name");

std::string propertyValue = prop->Attribute("value");

object.properties[propertyName] = propertyValue;

prop = prop->NextSiblingElement("property");

}

}

}

objects.push\_back(object);

objectElement = objectElement->NextSiblingElement("object");

}

objectGroupElement = objectGroupElement->NextSiblingElement("objectgroup");

}

}

else

{

std::cout << "No object layers found..." << std::endl;

}

return true;

}

Object Level::GetObj(std::string name)

{

// только первый объект с заданным именем

for (int i = 0; i < objects.size(); i++)

if (objects[i].name == name)

return objects[i];

}

std::vector<Object> Level::GetObjects(std::string name)

{

// все объекты с заданным именем

std::vector<Object> vec;

for (int i = 0; i < objects.size(); i++)

if (objects[i].name == name)

vec.push\_back(objects[i]);

return vec;

}

std::vector<Object> Level::GetAllObjects()

{

return objects;

};

void Level::ClearObjects()

{

objects.clear();

};

void Level::ClearLayers()

{

layers.clear();

};

sf::Vector2i Level::GetTileSize()

{

return sf::Vector2i(tileWidth, tileHeight);

}

void Level::Draw(sf::RenderWindow &window)

{

// рисуем все тайлы (объекты не рисуем!)

for (int layer = 0; layer < layers.size(); layer++)

for (int tile = 0; tile < layers[layer].tiles.size(); tile++)

window.draw(layers[layer].tiles[tile]);

}

**Kub.cpp**

#include "Kub.h"

#include "Menu.h"

Kub::Kub(String F, Level &lev, float X, float Y, float W, float H)

{

music.openFromFile("music/musicLevel1.ogg");

music.play();

obj = lev.GetAllObjects();

playerDeath = 0;

dx = 0.22;

dy = 0;

onGround = false;

life = true;

File = F;

w = W;

h = H;

image.loadFromFile("images/" + File);

image.createMaskFromColor(Color(255, 255, 255));

texture.loadFromImage(image);

sprite.setTexture(texture);

sprite.setTextureRect(IntRect(0, 0, 32, 32));

x = X;

y = Y;

}

Kub::~Kub()

{

}

void Kub::update(float time)

{

control(time);

x += dx\*time;

checkCollision(dx, 0);

if (!life)

{

playerDeath++;

life = true;

}

y += dy\*time;

sprite.setPosition(x, y);

if (!onGround)

{

dy = dy + 0.0028\*time\* upDown;

}

onGround = false;

checkCollision(0, dy);

}

float Kub::getplayercoordinateX()

{

return x;

}

float Kub::getplayercoordinateY()

{

return y;

}

void Kub::control(float time)

{

if (((Keyboard::isKeyPressed(Keyboard::W)) || (Keyboard::isKeyPressed(Keyboard::Space))) && onGround)

{

dy = -0.7 \* upDown;

dx = 0.22;

onGround = false;

}

}

void Kub::checkCollision(float Dx, float Dy)

{

for (int i = 0; i < obj.size(); i++)

if (getRect().intersects(obj[i].rect))

{

if (obj[i].name == "solid")

{

if ((Dy \* upDown) > 0)

{

y = obj[i].rect.top - h \* upDown;

dy = 0;

onGround = true;

}

if (Dx > 0)

{

x = obj[i].rect.left - w;

x = 32;

y = 250;

life = false;

music.stop();

music.play();

upDown = 1;

}

}

else

{

if (obj[i].name == "death")

{

x = obj[i].rect.left - w; x = 32; y = 250;

life = false;

music.stop();

music.play();

upDown = 1;

}

else

{

if (obj[i].name == "down")

{

upDown \*= -1;

x = 512; y = 416;

}

}

}

}

}

FloatRect Kub::getRect()

{

return FloatRect(x, y, w, h);

}

Object Kub::checkFinish(Object player, Object finish, Level &lvl, sf::RenderWindow &window, sf::RenderWindow &windowMenu)

{

String levelFile, musicFile;

if (x > finish.rect.left)

{

levelNum++;

if (levelNum < 3)

{

lvl.ClearObjects();

lvl.ClearLayers();

levelFile = "Levels/level" + std::to\_string(levelNum);

levelFile += ".tmx";

lvl.LoadFromFile(levelFile);

player = lvl.GetObj("player");

finish = lvl.GetObj("finish");

musicFile = "music/musicLevel" + std::to\_string(levelNum) + ".ogg";

music.openFromFile(musicFile);

music.play();

x = player.rect.left;

y = player.rect.top;

playerDeath = 0;

upDown = 1;

obj.clear();

obj = lvl.GetAllObjects();

}

else

{

window.close();

windowMenu.setVisible(true);

}

}

return finish;

}

void Kub::draw(RenderWindow &window, Level &lvl, Text text, Sprite background, float time, Text textLvl)

{

window.draw(sprite);

lvl.Draw(window);

window.display();

window.draw(sprite);

update(time);

window.clear();

window.draw(background);

std::ostringstream playerScoreString, levelNumString;

levelNumString << levelNum;

playerScoreString << playerDeath;

text.setString("Attempt " + playerScoreString.str());

textLvl.setString("Level " + levelNumString.str());

window.draw(text);

window.draw(textLvl);

}

**Menu.cpp**

#include "Menu.h"

Menu::Menu(float width, float heigth, sf::String FileName, int MAX\_NUMBER\_OF\_ITEMS)

{

sf::String filePicture;

for (int i = 0; i < MAX\_NUMBER\_OF\_ITEMS; i++)

{

filePicture = FileName;

filePicture += std::to\_string(i + 1) + ".png";

image[i].loadFromFile(filePicture);

texture[i].loadFromImage(image[i]);

sprite[i].setTexture(texture[i]);

sprite[i].setPosition(sf::Vector2f(width - width / 1.5, heigth / (MAX\_NUMBER\_OF\_ITEMS + 1) \* (i + 0.8)));

}

sprite[0].setColor(sf::Color::Red);

selectedItemIndex = 0;

}

Menu::~Menu()

{

}

void Menu::draw(sf::RenderWindow &window, int MAX\_NUMBER\_OF\_ITEMS)

{

for (int i = 0; i < MAX\_NUMBER\_OF\_ITEMS; i++)

{

window.draw(sprite[i]);

}

}

void Menu::MoveUp()

{

sf::Color color;

if (selectedItemIndex - 1 >= 0)

{

color = sprite[selectedItemIndex - 1].getColor();

sprite[selectedItemIndex].setColor(color);

selectedItemIndex--;

sprite[selectedItemIndex].setColor(sf::Color::Red);

}

}

void Menu::MoveDown(int MAX\_NUMBER\_OF\_ITEMS)

{

sf::Color color;

if (selectedItemIndex + 1 < MAX\_NUMBER\_OF\_ITEMS)

{

color = sprite[selectedItemIndex + 1].getColor();

sprite[selectedItemIndex].setColor(color);

selectedItemIndex++;

sprite[selectedItemIndex].setColor(sf::Color::Red);

}

}