Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по курсовому проекту

на тему

Компьютерная игра в жанре платформер “The Story Of A Rabbit”

Студент: гр. 553505 Станько В. В.

Руководитель: Сухарев А. А.

Минск 2016

**Содержание**

1. Введение……………………………………………………….3

1.1Цели работы ………………………………………………..3

1.2Постановка задач ………………………………………….3

2.Основная часть ………………………………………4

2.1Теоретические сведения …………………………….4

2.2Аналитический обзор ……………………………….4

2.2.1Пути реализации …………………………………….4

2.3Процесс разработки …………………………………8

2.3.1Алгоритм создания игры ……………………………8

2.3.2Дополнительные ресурсы …………………………12

2.3.3Программный код ………………………………….13

3.Заключение …………………………………………25

3.1Вывод ……………………………………………….26

4.Используемая литература и ресурсы……………....27

4.1Литература ……………………………………….…27

4.1Интернет-ресурсы …………………………….……27

**1.Введение**

Значение и популярность компьютерных игр в наше время довольно велики. Одним из самых первых жанров компьютерных игр в истории является именно платформер.

***1.1 Цели работы:***

1.Рассмотреть основные возможности языка С++.

2.Создание полноценной игры в жанре платформер.

3.Ознакомиться с работой библиотеки SFML.

***1.2 Постановка задач:***

1.Изучить литературу по основам языка С++

2.Найти и изучить специализированные ресурсы в Интернете по данной теме.

3.Написать программу для игры в жанре платформер с использованием возможностей библиотеки SFML.

1. **Основная часть**

**2.1Теоретические сведения**

Платфо́рмер (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. Коллекционные предметы, оружие собираются обычно простым прикосновением персонажа и для применения не требуют специальных действий со стороны игрока. Реже предметы собираются в «инвентарь» героя и применяются специальной командой (такое поведение более характерно для аркадных головоломок).

Противники (называемые «врагами»), всегда многочисленные и разнородные, обладают примитивным искусственным интеллектом, стремясь максимально приблизиться к игроку, либо не обладают им вовсе, перемещаясь по круговой дистанции или 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. Смерть живых существ обычно изображается упрощённо или символически (существо исчезает или проваливается вниз за пределы экрана).

Уровни, как правило, изобилуют секретами (скрытые проходы в стенах, высокие или труднодоступные места), нахождение которых существенно облегчает прохождение и подогревает интерес игрока.

Примерами таких игр являются Super Mario Bros., Sonic The Hedgehog.

Игры этого жанра появились и успели завоевать популярность еще в начале 1980-х годов, однако они актуальны и до сих пор.

**2.2 Аналитический** **обзор**

***2.2.1 Пути реализации***

Существует несколько путей реализации платформера:

1.Основанный на плитках

Движение персонажа ограничено плитками, так что невозможно стоять на стыке между двумя плитками. Могут быть использованы специальные анимации, чтобы создать иллюзию плавного движения, но, игрок всегда будет стоять прямо на какой-то плитке. Это самый простой способ реализации платформера, но это накладывает жесткие ограничения на управление персонажем, что делает подход непригодным для реализации игрового процесса привычных игр платформеров.



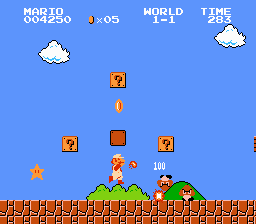
Пример платформера, основанного на плитках: Flashback



Prince of Persia, еще один пример игры в жанре платформер, основанный на плитках

2. **Основаный на плитках (со сглаживанием)**

В данном виде платформера взаимодействие с игровым миром также определяется по сетке ячеек, но персонажи имеют возможность свободно перемещаться по миру (обычно подразумевается разрешение 1px с округлением до целых). Это самый распространённый метод разработки платформеров. Он достаточно популярен и до сих пор благодаря своей относительной простоте, в связи с чем редактирование уровней игры представляет собой более лёгкую задачу, чем при работе с более продвинутыми типами. Это позволяет создавать наклонные платформы в уровне, а также задавать более плавную траекторию для прыжков.



Пример игры, основанной на плитках, но со сглаживанием: Super Mario Bros.

**3. Основанный на Битовой маске**

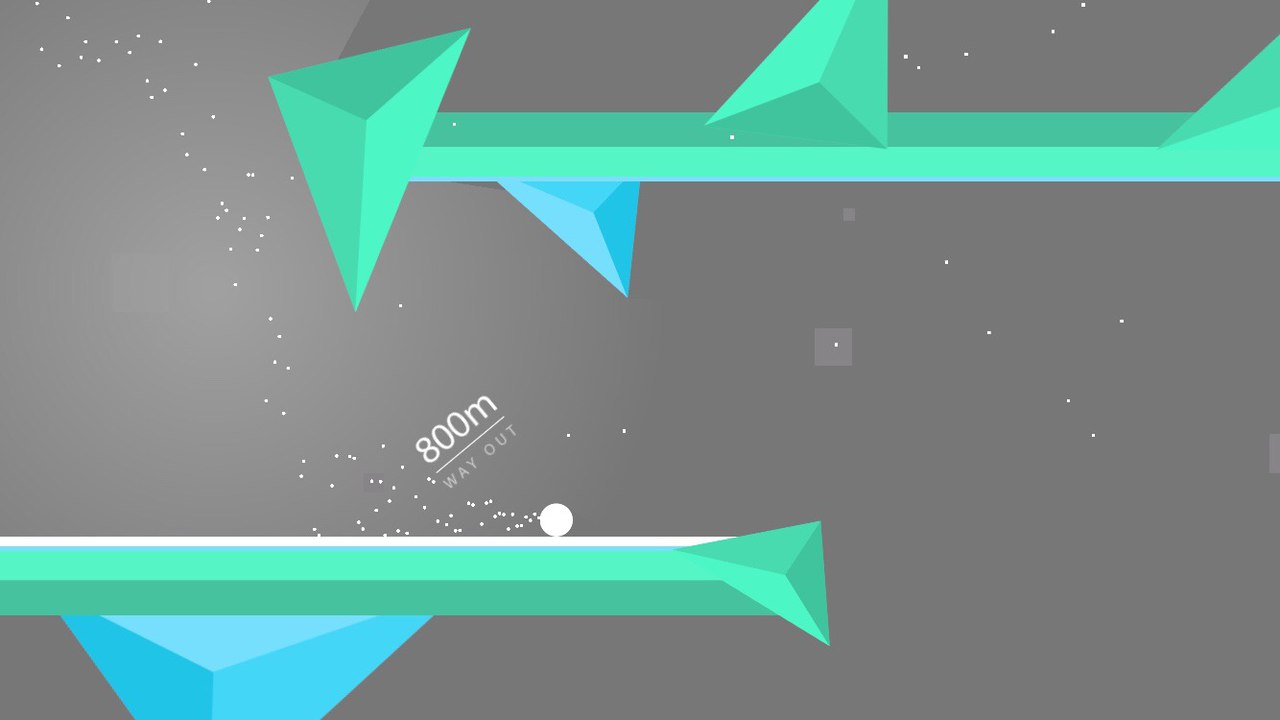
Данный вид платформера похож на предыдущий, только для определения столкновения используются не крупные плитки, а пиксели. Такая техника позволяет улучшить детализацию, но при этом увеличивается сложность исполнения и использование памяти. Часто для создания графики плитки не используются, поэтому для каждого уровня могут требоваться большие, замысловатые, индивидуальные изображения. В связи со всем этим, данная техника не является распространённой, но с её использованием можно добиться более качественных результатов, по сравнению с типами, основанными на ячеистом принципе. Она также подходит для создания динамичного окружения – в битовой маске можно «рисовать», и тем самым изменять модель уровня.

Пример игры, основанной на битовой маске: Игра Talbot’s Odyssey



**4. Основанный на векторах**

В этом способе создания используется векторный подход (линии и многоугольники) для определения границ области коллизии. Несмотря на сложность реализации, он набирает все большую популярность благодаря повсеместному использованию физических движков, таких как Box2D. Векторный способ обладает теми же преимуществами, что и битовая маска, но без большого расхода памяти и с использованием иного способа редактирования уровней.



Пример платформера на основе векторов: Parallels

Для создания игры в жанре платформер наиболее оптимальным будет использование способа, основанного на плитках, но со сглаживанием, поскольку он очень гибок, относительно прост в реализации, и предоставляет большую степень контроля, нежели другие три типа.

**2.3 Процесс разработки**

***2.3.1 Алгоритм создания игры***

Карта представляет собой сетку из плиток, каждая из которых хранит информацию о том, является ли она препятствием или нет, какое изображение используется, какие звуки шагов персонажа должны использоваться и так далее.

Информация о карте записывается и хранится следующим образом: создается Hitbox персонажа -  ограничивающий прямоугольник, выровненный по координатным осям (AABB, или, проще говоря, прямоугольник, который не вращается). Как правило, ограничивающий прямоугольник является целым кратным от размера ячейки. Во многих случаях, спрайт самого персонажа больше, чем логический хитбокс, так как это более приемлемо визуально. Игровой процесс таким образом становится «честнее», ведь для игрока лучше иметь возможность избежать удара противника, когда он по идее должен быть, чем получать повреждения, когда на экране снаряд проходит мимо. Если предположить, что на карте нет наклонных и односторонних платформ, алгоритм несложен:

1.          Последовательно разбить на шаги движение по осям X и Y. Затем, для каждой оси:

2.         Найти координату края, направленного вперёд (ведущего края). Например, при движении влево это координата Х левой стороны ограничивающего прямоугольника, при движении вправо - координата Х правой стороны, при движении вверх - координата Y верхней стороны и т. д.

3.         Определить, с какими линиями сетки пересекается ограничивающий прямоугольник – так можно получить минимальное и максимальное значение на противоположной оси (т.е. при движении по горизонтали находятся номера вертикальных ячеек, с которыми пересекаемся).

4.         Проследовать дальше по линии ячеек в направлении движения пока не найдётся ближайшее твердое препятствие. Затем пройтись по всем движущимся объектам, и определить ближайшее препятствие среди них на пути игрока.

5.         Минимальное из двух значений (расстояние до ближайшего препятствия и расстояние на которое необходимо было продвинуться изначально) и будет значением расстояния, на которое передвигается персонаж.

6.         Передвинуть игрока на новую позицию. С новой позиции, повторить шаги, и так далее.

Наклоны могут представлять определённую сложность, так как они по сути являются препятствиями, но при этом игрок всё же может частично входить в эти ячейки. Они также предполагают изменение Y координаты персонажа в зависимости от движения по оси X. Во избежание проблем, нужно сделать так, чтобы ячейка содержала параметр “floor y” каждой стороны.

Система, описанная ниже, позволяет вводить произвольные наклоны. Алгоритм столкновений для горизонтального движения изменяется следующим образом:

■       Здесь обязательно сначала смещаться по X, потом по Y.

■       Считается, что столкновение с ячейкой склона произошло только если его ближайший край - высокий. Это исключит «проваливание» персонажа через склон с противоположной стороны.

■       Может быть полезным запрещать склонам заканчиваться “на полпути”. Если этого не сделать, возможно столкновение с более сложными ситуациями, при которых игрок будет пытаться забраться с нижней стороны ячейки спуска. Эти ситуации решаются, например, путём предварительной обработки уровня с отметкой всех подобных конфликтных ячеек. При определении столкновения можно задать дополнительное условие: нижняя координата Y игрока должна быть больше (ниже на оси) чем выступающий край ячейки (tileСoord \* tileSize + floorY).

Для вертикального движения:

■       Если движение вниз по склону осуществляется за счёт гравитации, то необходимо убедиться в том, что минимальное гравитационное смещение совместимо со скоростью движения собственно по склону и по горизонтали. Если данные значения не контролировать, то игрок некоторое время после схода с выступа продолжит двигаться по горизонтали, пока гравитация его не спустит вниз. Это приведёт к тому, что игрок будет скакать по уклону, вместо того чтобы плавно спускаться по нему;

■       Как альтернатива гравитации, можно высчитать, на сколько пикселей выше уровня пола игрок находился до начала движения, и на сколько это значение изменилось после. Затем следует отрегулировать позицию игрока так, чтобы эти значения совпадали. Для вычислений можно использовать формулу из следующего пункта.

■       При движении вниз, вместо верхнего края плитки за границу соприкосновения должна быть принята нижняя координата плитки на данной вертикали. Для вычисления этой координаты, необходимо определить значение между [0, 1], представляющее собой позицию игрока вдоль ячейки (0 = лево, 1 = право), затем использовать её, чтобы линейно интерполировать значения floorY. Программный код будет иметь примерно следующий вид:    

float t = float(centerX - tileX) / tileSize;                    
float floorY = (1-t) \* leftFloorY + t \* rightFloorY;

■       При движении вниз, если есть несколько ячеек с общей координатой Y (например, персонаж находится между спуском и обычным твердой плиткой), то необходимо создать столкновение со склоном и игнорировать все прочие, даже если другие ячейки ближе. Это обеспечит должное поведение игрока возле краёв склонов: он будет «проваливаться» в фактически твёрдую ячейку из-за того, что тут начинается склон.

Лестницы могут показаться сложными для реализации, но они попросту представляют собой другое состояние персонажа: будучи на лестнице, игроком игнорируется практически вся система столкновений, и вместо неё вводится новый набор правил. Обычно, ширина лестницы - одна ячейка.

Переход в «лестничное» состояние возможен следующими путями:

■       Ограничивающий прямоугольник игрока пересекает лестницу на уровне пола или в полёте, игрок нажимает клавишу «вверх» (в некоторых играх переход возможен также по нажатию клавиши «вниз»);

■       Персонаж стоит на верхней «лестничной» ячейке (которая чаще всего по сути является односторонней платформой, чтобы по ней можно было пройтись сверху), игрок нажимает «вниз».

Таким образом, возникает эффект моментального совмещения x-координаты игрока с лестницей.  Для движения вниз с верхушки лестницы нужно переместить y-координату персонажа так, чтобы игрок оказался внутри лестницы. В некоторых играх для таких ситуаций вводится альтернативный ограничивающий прямоугольник, отслеживающий позицию игрока в пределах лестницы. Покинуть лестницу можно следующими способами:

■       Игрок достигает верха лестницы. Здесь обычно включается особая анимация, при которой игрок перемещается на несколько пикселей вверх по оси Y и оказывается над лестницей в стоячем положении;

■       Игрок движется влево или вправо. Если нет препятствий, игрок сходит с лестницы в соответствующем направлении;

■       Игрок прыгает. В некоторых играх позволяется отпустить лестницу таким образом.

Будучи на лестнице, действия игрока ограничены движением вверх-вниз, а также иногда возможно атаковать.

***2.3.2 Дополнительные ресурсы***

Для работы с изображениями и музыкой подходит довольно популярная библиотека SFML.

SFML (англ. *Simple and Fast Multimedia Library* — простая и быстрая мультимедийная библиотека) — свободная кроссплатформенная мультимедийная библиотека. Написана на C++, но доступна также для C, D, Java, Python, Ruby, OCaml, .Net и Go. Представляет собой объектно-ориентированный аналог SDL.

SFML содержит ряд модулей для простого программирования игр и мультимедиа приложений. В состав библиотеки входит несколько подключаемых модулей.

## **Модули**

**System** — управление временем и потоками, он является обязательным, так как все модули зависят от него.

**Window** — управление окнами и взаимодействием с пользователем.

**Graphics** — делает простым отображение графических примитивов и изображений, для своей работы требует модуль **Window**.

**Audio** — предоставляет интерфейс для управления звуком.

**Network** — для сетевых приложений.

Для того, чтобы написать платформер, необходимо подключить модули **System, Window, Graphics, Audio**.

Для хранения карты и анимации персонажа наиболее оптимальным будет использование формата XML, так как существуют программы редактора карт Tiled и редактора анимации SpriteDecomposer, которые используют именно формат XML (XML - расширяемый формат разметки).

Для работы с форматом XML оптимальнее всего будет использовать библиотеку TinyXML, которая разбирает документ XML, и строит из него Document Object Model (DOM), который может быть прочитан, модифицирован и сохранен. Оптимальнее это будет потому, что данная библиотека очень проста в подключении и использовании.

***2.3.3 Программный код***

Для простоты прочтения кода были созданы отдельно подключаемые модули:

1. ClAnimation.h – в данном модуле был создан класс, отвечающий за создание одиночной анимации и установку ее первоначальных параметров

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Audio.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

**using** **namespace** sf**;**

class Animation

**{**

public**:**

std**::**vector**<**IntRect**>** frames**,** frames\_flip**;**

float currentFrame**,** speed**;**

bool loop**,** flip**,** isPlaying**;** // loop показвает зациклена ли анимация. Например анимация взрыва должна проиграться один раз и остановиться, loop=false

Sprite sprite**;**

Animation**()**

void tick**(**float time**)**

**};**

1. ClAnimationManager.h – в данном модуле хранится вспомогательный класс AnimationManager, который хранит в себе массив анимаций

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Audio.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

#include "ClAnimation.h"

class AnimationManager

**{**

public**:**

std**::**string currentAnim**;**

std**::**map**<**std**::**string**,** Animation**>** animList**;**

AnimationManager**();**

**~**AnimationManager**();**

//создание анимаций вручную

void create**(**std**::**string name**,** Texture **&**texture**,** int x**,** int y**,** int w**,** int h**,** int count**,** float speed**,** int step**=**0**,** bool Loop**=true);**

//загрузка из файла XML

void loadFromXML**(**std**::**string fileName**,**Texture **&**t**);**

void set**(**std**::**string name**);**

void draw**(**RenderWindow **&**window**,**int x**=**0**,** int y**=**0**);**

void flip**(**bool b**=**1**)** ;

void tick**(**float time**);**

void pause**();**

void play**();**

void play**(**std**::**string name**);**

bool isPlaying**();**

float getH**()** ;

float getW**();**

**};**

1. ClLevel.h – в данном модуле располагаются элементы, необходимые для создания и отображения уровня

#pragma once

#include <string>

#include <vector>

#include <map>

#include <iostream>

#include <SFML/Graphics.hpp>

#include "Content\TinyXML\tinyxml.h"

**using** **namespace** sf**;**

struct Object

**{**

int GetPropertyInt**(**std**::**string name**);**

float GetPropertyFloat**(**std**::**string name**);**

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

std**::**string name**;**

std**::**string type**;**

sf**::**Rect**<**float**>** rect**;**

std**::**map**<**std**::**string**,** std**::**string**>** properties**;**

sf**::**Sprite sprite**;**

**};**

struct Layer

**{**

int opacity**;**

std**::**vector**<**sf**::**Sprite**>** tiles**;**

**};**

class Level

**{**

public**:**

bool LoadFromFile**(**std**::**string filename**);**

Object GetObject**(**std**::**string name**);**

std**::**vector**<**Object**>** GetObjects**(**std**::**string name**);**

std**::**vector**<**Object**>** GetAllObjects**();**

void Draw**(**sf**::**RenderWindow **&**window**);**

sf**::**Vector2i GetTileSize**();**

private**:**

int width**,** height**,** tileWidth**,** tileHeight**;**

int firstTileID**;**

sf**::**Rect**<**float**>** drawingBounds**;**

sf**::**Texture tilesetImage**;**

std**::**vector**<**Object**>** objects**;**

std**::**vector**<**Layer**>** layers**;**

**};**

1. ClEntity.h – модуль, в котором хранится базовый класс для всех динамических объектов

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Audio.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

#include "ClAnimation.h"

#include "ClAnimationManager.h"

#include "ClLevel.h"

class Entity

**{**

public**:**

float x**,**y**,**dx**,**dy**,**w**,**h**;**

AnimationManager anim**;**

std**::**vector**<**Object**>** obj**;**

bool life**,** dir**;**

float timer**,** timer\_end**;**

std**::**string Name**;**

int Health**;**

Entity**(**AnimationManager **&**A**,**int X**,** int Y**);**

virtual void update**(**float time**)** **=** 0**;**

void draw**(**RenderWindow **&**window**);**

FloatRect getRect**();**

void option**(**std**::**string NAME**,** float SPEED**=**0**,** int HEALTH**=**10**,** std**::**string FIRST\_ANIM**=**""**);**

**};**

1. ClEnemy.h – модуль, в котором располагается класс, в котором создается враг, а также обновляется состояние врага

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

#include "ClAnimation.h"

#include "ClAnimationManager.h"

#include "ClEntity.h"

#include "ClLevel.h"

**using** **namespace** sf**;**

class ENEMY**:** public Entity

**{**

public**:**

ENEMY**(**AnimationManager **&**a**,**Level **&**lev**,**int x**,**int y**):**Entity**(**a**,**x**,**y**);**

void update**(**float time**);**

**};**

1. ClBullet.h – модуль, в котором располагается класс, в котором создаются пули, а также обновляется их состояние

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Audio.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

#include "ClEntity.h"

#include "ClAnimationManager.h"

**using** **namespace** sf**;**

class Bullet**:**public Entity

**{**

public**:**

Bullet**(**AnimationManager **&**a**,** Level **&**lev**,**int x**,**int y**,**bool dir**):**Entity**(**a**,**x**,**y**);**

void update**(**float time**);**

**};**

1. ClPlayer.h – модуль, в котором располагается класс, в котором создается персонаж, а также располагаются обновление его состояния, обработка столкновений и таких событий как смерть персонажа или победа в игре

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Audio.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

#include "ClEntity.h"

#include "ClAnimation.h"

#include "ClAnimationManager.h"

#include "ClLevel.h"

**using** **namespace** sf**;**

class PLAYER**:** public Entity

**{**

public**:**

SoundBuffer jumpbuf**;**

Sound JuMp**;**

enum **{**stay**,**walk**,**jump**,**climb**}** STATE**;**

bool onLadder**,** shoot**,** hit**,** onGround**;**

std**::**map**<**std**::**string**,**bool**>** key**;**

int score**;**

bool fin**,** finish**;**

PLAYER**(**AnimationManager **&**a**,** Level **&**lev**,**int x**,**int y**):**Entity**(**a**,**x**,**y**);**

void Keyboard**();**

void update**(**float time**);**

void Collision**(**int num**);**

void Animation**(**float time**);**

void PlayerDied**(**RenderWindow **&**window**,**float time**);**

void PlayerWin**(**RenderWindow **&**window**,**float time**);**

**};**

1. ClHealthbar.h – модуль, в котором хранится класс, в котором создается строка здоровья игрока и обновляется ее состояние

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Audio.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

**using** **namespace** sf**;**

class HealthBar

**{**

public**:**

Texture t**;**

Sprite s**;**

int max**;**

RectangleShape bar**;**

bool health**;**

HealthBar**();**

void update**(**int k**);**

void draw**(**RenderWindow **&**window**);**

**};**

1. ClCoin.h - модуль, в котором создается подбираемый объект(монетка) и обновляется его состояние

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Audio.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

#include "ClAnimation.h"

#include "ClAnimationManager.h"

#include "ClEntity.h"

#include "ClLevel.h"

**using** **namespace** sf**;**

class COIN**:**public Entity

**{**

public**:**

COIN**(**AnimationManager **&**a**,** Level **&**lev**,** int x **,** int y**):**Entity**(**a**,**x**,**y**);**

void update **(**float time**);**

**};**

1. ClMenu.h – модуль, в котором располагается функция вызова меню игры

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Audio.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

#include "ClAnimation.h"

#include "ClAnimationManager.h"

#include "ClBullet.h"

#include "ClCoin.h"

#include "ClEnemy.h"

#include "ClEntity.h"

#include "ClHealthBar.h"

#include "ClLevel.h"

#include "ClMyText.h"

#include "ClPlayer.h"

**using** **namespace** sf**;**

bool menu**(**RenderWindow **&** window**)** **{**

Font font**;**

font**.**loadFromFile**(**"Content/whata2.ttf"**);**

Text menu1**,** menu2**,** menu3**;**

menu1**.**setString**(**"Start Game!"**);**

menu2**.**setString**(**"How To Play"**);**

menu3**.**setString**(**"Exit"**);**

menu1**.**setFont**(**font**);**

menu2**.**setFont**(**font**);**

menu3**.**setFont**(**font**);**

Texture menuBackground**,** aboutTexture**;**

aboutTexture**.**loadFromFile**(**"Content/info.jpg"**);**

menuBackground**.**loadFromFile**(**"Content/notr.jpg"**);**

Sprite menuBg**(**menuBackground**),** about**(**aboutTexture**);**

Music mus**;**

mus**.**openFromFile**(**"Content/track08.ogg"**);**

mus**.**play**();**

mus**.**setLoop**(true);**

bool isMenu **=** 1**;**

int menuNum **=** 0**;**

menu1**.**setPosition**(**100**,** 30**);**

menu2**.**setPosition**(**100**,** 90**);**

menu3**.**setPosition**(**100**,** 150**);**

menuBg**.**setPosition**(**0**,** 0**);**

**while** **(**isMenu**)**

**{**

**while** **(**window**.**isOpen**())**

**{** Event event**;**

**while** **(**window**.**pollEvent**(**event**))**

**{**

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

**{**

window**.**close**();**

isMenu **=** **false;**

**return** isMenu**;**

**}**

menu1**.**setColor**(**Color**::**White**);**

menu2**.**setColor**(**Color**::**White**);**

menu3**.**setColor**(**Color**::**White**);**

menuNum **=** 0**;**

window**.**clear**(**Color**(**129**,** 181**,** 221**));**

**if** **(**IntRect**(**100**,** 30**,** 300**,** 50**).**contains**(**Mouse**::**getPosition**(**window**)))** **{** menu1**.**setColor**(**Color**::**Magenta**);** menuNum **=** 1**;** **}**

**if** **(**IntRect**(**100**,** 90**,** 300**,** 50**).**contains**(**Mouse**::**getPosition**(**window**)))** **{** menu2**.**setColor**(**Color**::**Magenta**);** menuNum **=** 2**;** **}**

**if** **(**IntRect**(**100**,** 150**,** 300**,** 50**).**contains**(**Mouse**::**getPosition**(**window**)))** **{** menu3**.**setColor**(**Color**::**Magenta**);** menuNum **=** 3**;** **}**

**if** **(**Mouse**::**isButtonPressed**(**Mouse**::**Left**))**

**{**

**if** **(**menuNum **==** 1**)** **{**window**.**close**();** isMenu **=** **false;}**//если нажали первую кнопку, то выходим из меню

**if** **(**menuNum **==** 2**)** **{** window**.**draw**(**about**);** window**.**display**();** **while** **(!**Keyboard**::**isKeyPressed**(**Keyboard**::**Escape**));** **}**

**if** **(**menuNum **==** 3**)** **{** window**.**close**();** isMenu **=** **false;** **return** isMenu**;** **}**

**}**

**}**

window**.**draw**(**menuBg**);**

window**.**draw**(**menu1**);**

window**.**draw**(**menu2**);**

window**.**draw**(**menu3**);**

window**.**display**();**

**}** **}**

**}**

1. ClMyText.h – модуль, в котором хранится класс, в котором создаются текстовые объекты, такие, как отображение числа очков персонажа (количество собранных монеток), сообщение о прохождении уровня и т. д.

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Audio.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

**using** **namespace** sf**;**

class MyText

**{**

public**:**

Font font**;**

Text text**;**

MyText**();**

void drawDead **(**RenderWindow **&**window**);**

void drawWin **(**RenderWindow **&**window**);**

void draw **(**RenderWindow **&**window**,** int a**);**

**};**

1. ClGame.h – модуль, в котором контролируется нажатие клавиш, а также генерируются события столкновения, завершения уровня и т. д.

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include <SFML/Audio.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

#include <map>

#include <list>

#include <iostream>

#include <sstream>

#include "Content\TinyXML\tinyxml.h"

#include "ClAnimation.h"

#include "ClAnimationManager.h"

#include "ClBullet.h"

#include "ClCoin.h"

#include "ClEnemy.h"

#include "ClEntity.h"

#include "ClHealthBar.h"

#include "ClLevel.h"

#include "ClMenu.h"

#include "ClMyText.h"

#include "ClPlayer.h"

**using** **namespace** sf**;**

void ChangeLevel **(**Texture **&**bg **,**Level **&**lvl**,** int levelNum **);**

bool RunGame**(**RenderWindow **&**window**,** int **&** levNum**)**

**{**

View view**(** FloatRect**(**0**,** 0**,** 450**,** 280**)** **);**

Texture bg**;**

Level lvl**;**

ChangeLevel**(** bg**,**lvl**,**levNum**);**

Texture enemy\_t**,** moveplatform\_t**,** megaman\_t**,** bullet\_t/\* bg\*/**,** coin\_t**,** jump**,** walk**,** hit**,** stay**,** climb**,** attack**;**

enemy\_t**.**loadFromFile**(**"Content/enemy.png"**);**

megaman\_t**.**loadFromFile**(**"Content/11.png"**);**

bullet\_t**.**loadFromFile**(**"Content/bullet.png"**);**

coin\_t**.**loadFromFile**(**"Content/coin.png"**);**

AnimationManager anim**;**

anim**.**loadFromXML**(**"Content/13133.xml"**,**megaman\_t**);**

AnimationManager anim2**;**

anim2**.**create**(**"move"**,**bullet\_t**,**7**,**10**,**8**,**8**,**1**,**0**);**

anim2**.**create**(**"explode"**,**bullet\_t**,**27**,**7**,**18**,**18**,**4**,**0.01**,**29**,false);**

AnimationManager anim3**;**

anim3**.**create**(**"move"**,**enemy\_t**,**0**,**0**,**16**,**16**,**2**,**0.002**,**18**);**

anim3**.**create**(**"dead"**,**enemy\_t**,**58**,**0**,**16**,**16**,**1**,**0**);**

AnimationManager anim4**;**

anim4**.**create**(**"stay"**,** coin\_t**,** 0**,**0**,**31**,**31**,**1**,**0**);**

Music mus**;**

mus**.**openFromFile**(**"Content/track02.ogg"**);**

mus**.**play**();**

mus**.**setLoop**(true);**

Sprite background**(**bg**);**

background**.**setOrigin**(**bg**.**getSize**().**x**/**2**,**bg**.**getSize**().**y**/**2**);**

std**::**list**<**Entity**\*>** entities**;**

std**::**list**<**Entity**\*>::**iterator it**;**

std**::**vector**<**Object**>** e **=** lvl**.**GetObjects**(**"enemy"**);**

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

entities**.**push\_back**(new** ENEMY**(**anim3**,** lvl**,** e**[**i**].**rect**.**left**,** e**[**i**].**rect**.**top**)** **);**

e **=** lvl**.**GetObjects**(**"star"**);**

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

entities**.**push\_back**(new** COIN**(**anim4**,** lvl**,** e**[**i**].**rect**.**left**,** e**[**i**].**rect**.**top**)** **);**

Object pl **=** lvl**.**GetObject**(**"player"**);**

PLAYER Mario**(**anim**,** lvl**,** pl**.**rect**.**left**,** pl**.**rect**.**top**);**

MyText texscore**;**

HealthBar healthBar**;**

Clock clock**;**

**while** **(**window**.**isOpen**())**

**{**

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

clock**.**restart**();**

time **=** time**/**500**;**

**if** **(**time **>** 40**)** time **=** 40**;**

Event event**;**

**while** **(**window**.**pollEvent**(**event**))**

**{**

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

window**.**close**();**

**if** **(**event**.**type **==** Event**::**KeyPressed**)**

**if** **(**event**.**key**.**code**==**Keyboard**::**Space**)**

entities**.**push\_back**(new** Bullet**(**anim2**,**lvl**,**Mario**.**x**+**18**,**Mario**.**y**+**18**,**Mario**.**dir**)** **);**

**}**

**if** **(**Keyboard**::**isKeyPressed**(**Keyboard**::**Left**))** Mario**.**key**[**"L"**]=true;**

**if** **(**Keyboard**::**isKeyPressed**(**Keyboard**::**Right**))** Mario**.**key**[**"R"**]=true;**

**if** **(**Keyboard**::**isKeyPressed**(**Keyboard**::**Up**))** Mario**.**key**[**"Up"**]=true;**

**if** **(**Keyboard**::**isKeyPressed**(**Keyboard**::**Down**))** Mario**.**key**[**"Down"**]=true;**

**if** **(**Keyboard**::**isKeyPressed**(**Keyboard**::**Space**))** Mario**.**key**[**"Space"**]=true;**

**for(**it**=**entities**.**begin**();**it**!=**entities**.**end**();)**

**{**

Entity **\***b **=** **\***it**;**

b**->**update**(**time**);**

**if** **(**b**->**life**==false)** **{** it **=** entities**.**erase**(**it**);** **delete** b**;}**

**else** it**++;**

**}**

**if** **(**Keyboard**::**isKeyPressed**(**Keyboard**::**Tab**))** **return** **true;**

**if** **(**Keyboard**::**isKeyPressed**(**Keyboard**::**Escape**))** **return** **false;**

Mario**.**update**(**time**);**

healthBar**.**update**(**Mario**.**Health**);**

**for(**it**=**entities**.**begin**();**it**!=**entities**.**end**();**it**++)**

**{**

**if** **((\***it**)->**Name**==**"Enemy"**)**

**{**

Entity **\***enemy **=** **\***it**;**

**if** **(**enemy**->**Health**<=**0**)** **continue;**

**if** **(**Mario**.**getRect**().**intersects**(** enemy**->**getRect**()** **))**

**if** **(**Mario**.**dy**>**0**)** **{** enemy**->**dx**=**0**;** Mario**.**dy**=-**0.2**;** enemy**->**Health**=**0**;}**

**else** **if** **(!**Mario**.**hit**)** **{** Mario**.**Health**-=**5**;** Mario**.**hit**=true;**

**if** **(**Mario**.**dir**)** Mario**.**x**+=**10**;** **else** Mario**.**x**-=**10**;}**

**for** **(**std**::**list**<**Entity**\*>::**iterator it2**=**entities**.**begin**();** it2**!=**entities**.**end**();** it2**++)**

**{**

Entity **\***bullet **=** **\***it2**;**

**if** **(**bullet**->**Name**==**"Bullet"**)**

**if** **(** bullet**->**Health**>**0**)**

**if** **(**bullet**->**getRect**().**intersects**(** enemy**->**getRect**()** **)** **)**

**{** bullet**->**Health**=**0**;** enemy**->**Health**-=**5**;}**

**}**

**}**

**if** **((\***it**)->**Name**==**"Coin"**)**

**{**

Entity **\***coin **=** **\***it**;**

**if** **(**Mario**.**getRect**().**intersects**(** coin**->**getRect**()** **))**

**{** coin**->**Health**=**0**;**Mario**.**score**++;}**

**}**

**}**

view**.**setCenter**(** Mario**.**x**,**Mario**.**y**);**

window**.**setView**(**view**);**

background**.**setPosition**(**view**.**getCenter**());**

window**.**draw**(**background**);**

lvl**.**Draw**(**window**);**

**for(**it**=**entities**.**begin**();**it**!=**entities**.**end**();**it**++)**

**(\***it**)->**draw**(**window**);**

Mario**.**draw**(**window**);**

healthBar**.**draw**(**window**);**

**if** **(!**healthBar**.**health**)**

**{**

MyText text**;**

text**.**drawDead**(**window**);**

Mario**.**PlayerDied**(**window**,** time**);**

**}**

**if** **(**Mario**.**finish**)**

**{**

MyText text**;**

text**.**drawWin**(**window**);**

Mario**.**PlayerWin**(**window**,** time**);**

**};**

**if** **(**Mario**.**fin**)**

**{**

**return** **true;**

**};**

texscore**.**draw**(**window**,** Mario**.**score**);**

window**.**display**();**

**}**

**};**

void Gamerunning**(** RenderWindow **&**window**,**int **&** levelNum**);**

1. **Заключение**

Используя все вышеперечисленные модули, можно получить игру:

**

Скриншот из игры: меню игры, надпись при наведении на нее мышкой начинает окрашиваться в другой цвет

**

Скриншот из игры: персонаж и собираемые объекты(монетки), слева в верхнем углу можно увидеть строку здоровья и количество очков

**

Скриншот из игры: анимация передвижения персонажа и объект, при пересечении с которым игра переходит на новый уровень или происходит завершение игры

***3.1 Вывод:***

В ходе проделанной работы были изучены материалы по языку C/C++, библиотекам TinyXML, SFML.Кроме этого, мною также были изучены алгоритмы создания платформеров, основанных на плитках.

В ходе написания курсового проекта были затронуты такие темы, как обработка XML – документов, объектно-ориентированное программирование, создание простейшей физики объектов и создание покадровой анимации вручную.

Данный проект стал основой для изучения основ игровой концепции, а также довольно эффективным способом изучения C/C++.

1. **Используемая литература и ресурсы**

***4.1 Литература***

Б. Керниган, Д. Ритчи - Язык программирования Си

Герберт Шилдт - Полный справочник по C++

Tyler Sylvester – Designing games

* 1. ***Интернет-ресурсы:***

http://wikipedia.org

http://cppstudio.com

http://cpp.com.ru/

http://progressor-blog.ru/

https://code-live.ru

http://quiz.trinli.net/