|  |  |
| --- | --- |
|  | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  федеральное государственное бюджетное образовательное учреждение высшего образования  «Балтийский государственный технический университет  «BOEHMEX» им. Д.Ф. Устинова» (БFTУ «BOEHMEX» им. Д.Ф. Устинова») |
| БГТУ.СМК-Ф-4.2-К5-01 |

Факультет Кафедра Дисциплина

И Информационные и управляющие системы

шифр наименование

И5 Информационные системы и программная инженерия

шифр наименование

Программирование на языке высокого уровня

КУРСОВАЯ РАБОТА

на тему

СОЗДАНИЕ КОМПЬЮТЕРНЫХ ИГР НА ЯЗЫКЕ С++ С ИСПОЛЬЗОВАНИЕМ SDL

Выполнил студент группы И407Б

Альков В.С.

Фамилия И.О.

РУКОВОДИТЕЛЬ

Васюков В.М.

Фамилия И.О. Подпись

Оценка

« »

Санкт-Петербург

2021 г.

## Содержание

Содержание 2

Перечень сокращений и обозначений 3

Введение 4

1. Постановка задачи 5

2. Описание программы 6

2. 1. Используемые файлы 6

2. 2. Диаграмма классов 6

2. 3. Описание файла классов 8

2. 4. Описание основного исходного файла 26

2. 5. Описание изображений 27

2. 6. Результаты работы программы 27

Заключение 31

Список использованных источников 32

## Перечень сокращений и обозначений

В отчете используются следующие сокращения и обозначения. SDL – Simple DirectMedia Layer – свободная кроссплатформенная мультимедийная библиотека, реализующая единый программный интерфейс к графической подсистеме, звуковым устройствам и средствам ввода для широкого спектра платформ.

## Введение

В процессе этой работы мы создадим игру, написанную на языке программирования C++, с использованием библиотеки SDL 2.0 и ее подключаемых библиотек.

## 1. Постановка задачи

Целью курсовой работы является создание компьютерной игры на языке C++ с использованием графического пользовательского интерфейса и библиотеки SDL 2.0.

Название игры: «Лабиринт».

Правила игры: необходимо найти сыр в лабиринте.

Игрок с помощью клавиш управления (стрелки) передвигается по карте. Если при движении игрока его координаты совпадут с координатами стен, то игрок не сможет пройти. Если координаты сыра совпадут с координатами игрока, то сыр считается найденным, а уровень пройденным.

Цель игры: найти сыр и пройти уровень как можно быстрее.

## 2. Описание программы

### **2. 1. Используемые файлы**

Проект состоит из следующих файлов:

- main.cpp – основной исходный файл;  
- classes.h – заголовочный файл с классами;  
- a\_Albionic.ttf – файл шрифта;  
- mouse.png – изображение игрока;  
- cheese.png – изображение сыра;  
- lab1.dat, lab2.dat, lab3.dat, lab4.dat, lab5.dat – файлы для построения карты;  
- results.dat – файл для хранения результатов;

К программе подключаются библиотеки SDL2, SDL2\_image, SDL2\_ttf.

### **2. 2. Диаграмма классов**

Диаграмма классов представлена на рисунках 1-3:

|  |
| --- |
| Untitled Diagram(3) |
| Рисунок 1 – Диаграмма классов(1) |

|  |  |
| --- | --- |
| 2021-06-29_085039 | |
| Рисунок 2 – Диаграмма классов(2) |

|  |
| --- |
| 2021-06-29_085125 |
| Рисунок 3 – Диаграмма классов(3) |

### **2. 3. Описание файла классов**

Ниже приведено содержимое файла classes.h. Все необходимые пояснения даны в комментариях к коду.

#ifndef CLASSES\_H

#define CLASSES\_H

#include <vector>

#include <algorithm>

#include <chrono>

#include <fstream>

static SDL\_Renderer\* renderer; /\*рендерер\*/

static SDL\_Window\* window; /\*окно\*/

static int windowHeight; /\*высота окна\*/

static int windowWidth; /\*ширина окна\*/

class Object /\*абстрактный класс для стен, сыра, камеры и игрока\*/

{

protected:

int x, y, h, w;

SDL\_Rect rect;

public:

Object(){};

virtual ~Object(){};

virtual void draw()=0;

virtual int getType() /\*тип объекта\*/

{

return 0;

}

void setWidth(int width) /\*установить ширину объекта\*/

{

rect.w=width;

}

void setHeight(int height) /\* установить высоту объекта \*/

{

rect.h=height;

}

void setXY(int posx, int posy) /\* установить координаты объекта \*/

{

rect.x=posx;

rect.y=posy;

}

void setXY2(int posx, int posy) /\* установить координаты объекта \*/

{

x=posx;

y=posy;

}

int getX() /\*получение координаты по X\*/

{

return rect.x;

}

int getX2() /\*получение координаты по X\*/

{

return x;

}

int getY() /\*получение координаты по Y\*/

{

return rect.y;;

}

int getY2() /\*получение координаты по Y\*/

{

return y;

}

int getW() /\*получение ширины объекта\*/

{

return rect.w;

}

int getH() /\*получение высоты объекта\*/

{

return rect.h;

}

virtual void move(int dx, int dy) /\*смещение координат объекта по оси X на dx, Y на dy\*/

{

rect.x+=dx;

rect.y+=dy;

};

virtual void addH(int dh) /\*увеличение высоты объекта на dh\*/

{

rect.h+=dh;

}

virtual void addW(int dw) /\*увеличение ширины объекта на dw\*/

{

rect.w+=dw;

}

};

class Wall : public Object /\*класс стены, производный от Object\*/

{

public:

Wall(){};

Wall(int posx, int posy, int width, int height)

{

rect.x = posx;

rect.y = posy;

rect.w = width;

rect.h = height;

}

~Wall(){};

void draw()

{

SDL\_SetRenderDrawColor(renderer, 0, 0, 0, 0); /\*устанавливаем цвет стен\*/

SDL\_RenderDrawRect(renderer, &rect);

}

virtual int getType() /\*получение типа\*/

{

return -1;

}

};

class HorisontalWall : public Wall /\*класс горизонтальной стены, производный от Wall\*/

{

public:

HorisontalWall(int z1, int z2, int z3, int z4): Wall(z1,z2,z3,z4){};

void setWidth(int w)

{

rect.h = w;

}

int getType()

{

return 0;

}

};

class VerticalWall : public Wall /\*класс вертикальной стены, производный от Wall\*/

{

public:

VerticalWall(int z1, int z2, int z3, int z4): Wall(z1,z2,z3,z4){};

int getType()

{

return 1;

}

};

class Player : public Object /\*класс игрока, производный от Object\*/

{

int speed;

std::vector<Object\*> obj; /\*объекты для обработки столкновений\*/

SDL\_Texture \*texture;

bool cheese; /\*x,y от object – координаты на карте\*/

public:

friend class Wall;

Player()

{

rect.h = 22\*windowWidth\*0.002;

rect.w = 12\*windowWidth\*0.002;

speed = 10\*windowWidth\*0.001;

cheese = false;

};

~Player()

{

SDL\_DestroyTexture(texture);

}

void setObjects(std::vector<Object\*> objects) /\*установка объектов\*/

{

obj = objects;

}

bool isCollisonWithWalls(int x1, int y1, int x2, int y2) /\*проверка столкновений\*/

{

int count = obj.size();

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

{

if(obj[i]->getType() != 3) /\*если объект не типа Spawn\*/

{

if(obj[i]->getY() + obj[i]->getH()< y1)

continue;

if(obj[i]->getX() + obj[i]->getW()< x1)

continue;

if(obj[i]->getY() > y2)

continue;

if(obj[i]->getX() >x2)

continue;

if(obj[i]->getType() == 2) /\*если объект типа Cheese\*/

cheese = true;

return true;

}

}

return false;

}

void move(int dx, int dy) /\*движение по карте\*/

{

if(!this->isCollisonWithWalls(x+dx\*speed, y+dy\*speed, x+rect.w+dx\*speed, y+rect.h+dy\*speed)) /\*проврка столкновения с объектами в точке, куда переместиться игрок на текущей скорости\*/

{

x+=speed\*dx;

y+=speed\*dy;

}

else if(!this->isCollisonWithWalls(x+dx, y+dy, x+rect.w+dx-1, y+rect.h+dy-1))

{

/\*на минимальной скорости, чтобы можно было подходить к стенам вплотную\*/

x+=dx;

y+=dy;

}

}

void draw()

{

SDL\_RenderCopy(renderer, texture, nullptr, &rect);

}

void setTexture(SDL\_Texture \*t) /\*установка текстуры\*/

{

texture = t;

}

bool cheeseCollected() /\*проверка собран ли сыр\*/

{

return cheese;

}

void reset() /\*сброс\*/

{

cheese = false;

}

};

class Camera :public Object /\*класс камеры, производный от Object\*/

{

SDL\_Texture \*texture; /\*храниться вся карта\*/

Object \*player;

public:

Camera()

{

texture = nullptr;

player = nullptr;

w=h=0;

rect = {0, 0, windowWidth, windowHeight}; /\*область экрана\*/

};

~Camera()

{

SDL\_DestroyTexture(texture);

texture = nullptr;

}

void draw()

{

move(); /\*передвигаем игрока в координатах экрана и камеру за ним, а потом рисуем\*/

SDL\_RenderCopy(renderer, texture, &rect, NULL);

}

void move() /\*движение камеры по карте вместе с игроком в координатах экрана\*/

{

rect.x = player->getX2() - windowWidth/2; /\*получаем координаты области на карте\*/

rect.y = player->getY2() - windowHeight/2; /\* чтобы игрок находился в центре экрана\*/

if(rect.x<0) /\*проверка выхода за границы экрана\*/

rect.x=0;

if(rect.y<0)

rect.y=0;

if(rect.x+windowWidth>w)

rect.x=w-windowWidth;

if(rect.y+windowHeight>h)

rect.y=h-windowHeight;

player->setXY(player->getX2() - rect.x, player->getY2()-rect.y); /\*устанавливаем координаты игрока на экране\*/

}

void setPlayer(Object \*p) /\*установка игрока\*/

{

player = p;

}

void setTexture(SDL\_Texture \*t) /\*установка текстуры\*/

{

texture = t;

SDL\_QueryTexture(t, NULL, NULL, &w, &h);

}

};

/\*ф-ии для добавления пространства между стенами для масштабируемости карты\*/

void addSpaceY(std::vector<Object\*> d, int offset, int size);

void addSpaceX(std::vector<Object\*> d, int offset, int size);

bool comp1(Object \*i, Object \*j);

bool comp2(Object \*i, Object \*j);

class Labyrinth /\*класс лабиринт\*/

{

int wallWidth, w,h, x, y;

std::vector<Object\*> objects;

public:

Labyrinth(std::vector<Object\*> obj)

{

w=h=0;

wallWidth = 2;

objects = obj;

};

~Labyrinth()

{

int size = objects.size();

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

delete objects[i];

objects.clear();

}

void addSpace(int ds) /\*добавление пространства между стенами\*/

{

addSpaceY(objects, ds, h+wallWidth);

addSpaceX(objects, ds, w+wallWidth);

}

void setWallWidth(int w) /\*установка ширины стен, для вертикальных ширина – ширина, горизонтальных - высота\*/

{

int size = objects.size();

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

if(!objects[i]->getType())

objects[i]->setHeight(w);

else

objects[i]->setWidth(w);

}

void resize(int width) /\*масштабирование лабиринта\*/

{

findHW();

addSpace(width);

setWallWidth(width/4);

findHW();

}

void findHW() /\*нахождение высоты и ширины лабиринта\*/

{

w=h=0;

int size = objects.size();

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

{

if(objects[i]->getW()>w)

w=objects[i]->getW();

if(objects[i]->getH()>h)

h=objects[i]->getH();

}

}

SDL\_Texture\* getTexture() /\*получение текстуры лабиринта, карты\*/

{

SDL\_Texture \*texture = SDL\_CreateTexture(renderer, SDL\_PIXELFORMAT\_RGBA8888,

SDL\_TEXTUREACCESS\_TARGET, w+windowWidth\*0.08, h+windowHeight\*0.08);

SDL\_SetRenderTarget(renderer, texture);

SDL\_SetRenderDrawColor(renderer, 255, 255, 255, 255);

SDL\_RenderClear(renderer);

this->draw();

SDL\_SetRenderTarget(renderer, NULL);

return texture;

};

void draw()

{

int size = objects.size();

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

objects[i]->draw();

}

void getPlayerSpawn(int\* sx, int\* sy) /\*получение начальных координат игрока на карте\*/

{

int size = objects.size();

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

if(objects[i]->getType() == 3)

{

\*sx = objects[i]->getX();

\*sy = objects[i]->getY();

}

}

void move(int dx, int dy) /\*передвижение лабиринта на dx, dy\*/

{

int size = objects.size();

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

objects[i]->move(dx, dy);

}

};

bool comp1(Object \*i, Object \*j)

{

return i->getY() < j->getY();

}

bool comp2(Object \*i, Object \*j)

{

return i->getX() < j->getX();

}

void addSpaceY(std::vector<Object\*> d, int offset, int size)

{

std::sort(d.begin(), d.end(), comp1);

int check[size];

int b, c, z, sizeD = d.size();

for(b=0; b<size; check[b]=0,b++);

for(b = 0; b<sizeD; b++)

if(d[b]->getType()==0)

{

check[d[b]->getY()] = 1;

break;

};

for(b=0; b<sizeD; b++)

if(z!=d[b]->getY())

{

check[d[b]->getY()] = 1;

z=d[b]->getY();

};

int y0 = d[0]->getY();

for(int i = 0, k=0; i<sizeD; i++)

{

if(y0 != d[i]->getY())

{

k++;

y0 = d[i]->getY();

}

if (d[i]->getType()==1)

{

c=0;

for(int j = d[i]->getY(), h=j+d[i]->getH(); j<h; c+=check[j], j++);

d[i]->addH(offset\*(c-1));

}

d[i]->move(0, k\*offset);

};

}

void addSpaceX(std::vector<Object\*> d, int offset, int size)

{

std::sort(d.begin(), d.end(), comp2);

int check[size], b, z, sizeD = d.size();

for(b=0; b<size; check[b]=0,b++);

for(b = 0; b<sizeD; b++)

if(d[b]->getType()==1)

{

z=d[b]->getX();

check[d[b]->getX()] = 1;

break;

};

for(; b<sizeD; b++)

if(z!=d[b]->getX())

{

check[d[b]->getX()] = 1;

z=d[b]->getX();

};

int x0 = d[0]->getX(), c;

for(int i = 0, k=0; i<sizeD; i++)

{

if(x0 != d[i]->getX())

{

k++;

x0 = d[i]->getX();

};

if (d[i]->getType() == 0)

{

c=0;

for(int j = d[i]->getX(), h=j+d[i]->getW(); j<h; c+=check[j], j++);

d[i]->addW(offset\*(c-1));

}

d[i]->move(k\*offset, 0);

};

}

class Cheese: public Object /\*класс сыр, производный от Object\*/

{

SDL\_Texture \*texture;

public:

Cheese(int posx, int posy, int width, int height)

{

rect.x = posx;

rect.y = posy;

rect.w = 12\*windowWidth\*0.01;

rect.h = 5\*windowWidth\*0.01;

texture = IMG\_LoadTexture(renderer,"resourse/cheese.png");

}

~Cheese()

{

SDL\_DestroyTexture(texture);

}

int getType() /\*тип объекта\*/

{

return 2;

}

void draw()

{

SDL\_RenderCopy(renderer, texture, nullptr, &rect);

}

};

class Spawn: public Object /\*класс спавн, производный от Object\*/

{

public:

Spawn(int posx, int posy, int width, int height)

{

rect.x = posx;

rect.y = posy;

rect.w = width;

rect.h = height;

}

~Spawn(){};

int getType() /\*тип объекта\*/

{

return 3;

}

void draw()

{

}

};

class Timer /\*класс таймер\*/

{

std::chrono::high\_resolution\_clock::time\_point t\_begin;

public:

Timer() /\*засекаем время\*/

{

t\_begin = std::chrono::high\_resolution\_clock::now();

}

~Timer(){};

void reset() /\*сброс таймера\*/

{

t\_begin = std::chrono::high\_resolution\_clock::now();

}

int getTime() /\*получение текущего отсчета\*/

{

return std::chrono::duration\_cast<std::chrono::seconds>(std::chrono::high\_resolution\_clock::now() - t\_begin).count();

}

};

TTF\_Font\* loadFont(std::string name, int size) /\*ф-ия загрузки шрифта\*/

{

return TTF\_OpenFont(name.c\_str(), size);

}

class Text /\*класс текстурного текста\*/

{

SDL\_Texture\* texture;

SDL\_Rect rect;

SDL\_Color color;

TTF\_Font\* font;

Uint8 alpha;

public:

Text()

{

texture = nullptr;

rect.x = rect.y = rect.w = rect.h = 0;

color.r = color.g = color.b = 0;

color.a = 255;

}

~Text()

{

clearText();

}

void draw()

{

SDL\_RenderCopy(renderer, texture, nullptr, &rect);

}

void setPos(int x, int y) /\*установка координат\*/

{

rect.x = x;

rect.y = y;

}

void setText(std::string text) /\*установка текста\*/

{

SDL\_Surface\* textSurface = TTF\_RenderUTF8\_Blended(font, text.c\_str(), color);

if (textSurface != nullptr)

{

texture = SDL\_CreateTextureFromSurface(renderer, textSurface);

if (texture != nullptr)

{

rect.w = textSurface->w;

rect.h = textSurface->h;

}

SDL\_FreeSurface(textSurface);

}

}

void clearText() /\*очистка текста\*/

{

SDL\_DestroyTexture(texture);

texture = nullptr;

rect.w = rect.h = 0;

}

void setFont(TTF\_Font\* f) /\*установка шрифта\*/

{

font = f;

}

void setColor(Uint8 r, Uint8 g, Uint8 b) /\*установка цвета текста\*/

{

color.r = r;

color.g = g;

color.b = b;

}

void setAlpha(Uint8 a) /\*установка прозрачности текста\*/

{

alpha = a;

color.a = alpha;

SDL\_SetTextureAlphaMod(texture, alpha);

}

int x() /\*получение x координаты\*/

{

return rect.x;

}

int y() /\*получение y координаты\*/

{

return rect.y;

}

int h() /\*получение высоты\*/

{

return rect.h;

}

int w() /\*получение ширины\*/

{

return rect.w;

}

};

class Level /\*класс уровень\*/

{

Labyrinth\* scene;

Player\* player;

SDL\_Texture\* texture;

Camera\* camera;

Timer timer;

Text textTimer;

int timeOfCompletion; /\*время выполнения\*/

bool complete; /\*пройден ли уровень\*/

bool quit; /\*флаг выхода\*/

TTF\_Font\* font;

public:

Level()

{

texture = nullptr;

camera = nullptr;

scene = nullptr;

font = loadFont("resourse/a\_Albionic.ttf", windowWidth\*0.03);

textTimer.setFont(font);

textTimer.setColor(0, 0, 0);

textTimer.setPos(0,0);

textTimer.setAlpha(255);

complete = quit = false;

}

~Level()

{

this->clear();

TTF\_CloseFont(font);

}

bool isQuitToMenu() /\*проверка выхода в меню\*/

{

return quit;

}

void handleKeyboardEvent(const SDL\_KeyboardEvent &event)/\*обработчик событий клавиатуры\*/

{

switch (event.state)

{

case SDL\_PRESSED: switch (event.keysym.sym)

{

case SDLK\_q: quit = true; break;

case SDLK\_UP: player->move(0, -1); break;

case SDLK\_DOWN: player->move(0, 1); break;

case SDLK\_LEFT: player->move(-1, 0); break;

case SDLK\_RIGHT: player->move(1, 0); break;

}

break;

}

if(player->cheeseCollected()) /\*если сыр собран, то уровень пройден\*/

{

timeOfCompletion = timer.getTime();

complete = true;

}

}

void draw()

{

camera->draw();

player->draw();

textTimer.setText("Время: "+std::to\_string(timer.getTime())); /\*таймер\*/

textTimer.draw();

textTimer.clearText();

}

void init(Labyrinth\* s) /\*инициализация уровня\*/

{

scene = s;

scene->resize(windowWidth \* 0.1); /\*устанавливаем размер карты\*/

texture = scene->getTexture(); /\*получаем текстуру карты\*/

camera->setTexture(texture);

int x, y; /\*получаем координаты игрока на карте\*/

scene->getPlayerSpawn(&x, &y);

player->setXY2(x,y); /\*устанавливаем координаты игрока\*/

timer.reset(); /\*сброс таймера\*/

}

void setPlayer(Player\* p) /\*установка игрока\*/

{

player = p;

}

void setCamera(Camera\* cam) /\*установка камеры\*/

{

camera = cam;

}

void clear() /\*очистка уровня\*/

{

complete = quit = false;

SDL\_DestroyTexture(texture);

player->reset();

delete scene;

texture = nullptr;

scene = nullptr;

}

bool completed() /\*проверка выполнения уровня\*/

{

return complete;

}

int getCompletionTime() /\*получение времени выполнения уровня\*/

{

return timeOfCompletion;

}

};

class Index /\*класс Индекс, вспомогательный для класса MainMenu, активный элемент меню\*/

{

int num;

public:

Index()

{

num = 0;

}

~Index(){};

int operator++()

{

num++;

if (num>2)

num=0;

return num;

}

int operator--()

{

num--;

if (num<0)

num = 2;

return num;

}

operator int()

{

return num;

}

};

class MainMenu /\*класс меню\*/

{

std::vector<Text\*> items, resultItems; /\*items – пункты меню, resultItems – эл-ты пункта “Результаты”\*/

Index currentItem;

int alpha\_selected, alpha\_unselected, w, h, state; /\*state – состояние меню\*/

bool exit, start;

public:

bool isExit(){return exit;} /\*проверка выхода\*/

bool isStart(){return start;} /\*проверка начала игры\*/

MainMenu()

{

w = h = 500;

alpha\_selected = 250;

alpha\_unselected = 100;

exit = start = false;

state = 0;

}

~MainMenu()

{

reset();

int size = items.size();

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

items[i]->clearText();

}

void draw()

{

int size1 = items.size(), size2 = resultItems.size();

switch(state) /\*state, 0 – главное меню, 1 – подменю с результатами\*/

{

case 0: for(int i=0; i<size1; i++)

items[i]->draw();

break;

case 1: for(int i=0; i<size2; i++)

resultItems[i]->draw();

break;

}

}

void initItems() /\*инициализация пунктов главного меню\*/

{

int windowXCenter=windowWidth/2, windowYCenter =windowHeight/2;

int fontSize=windowWidth\*0.064;

fontSize = fontSize>32 ? 32 : fontSize;

TTF\_Font\* font = loadFont("resourse/a\_Albionic.ttf", fontSize);

items.resize(3);

std::string itemsNames[3] ={"старт", "результаты", "выход"};

for(int i = 0; i<3; i++)

{

items[i] = new Text();

items[i]->setFont(font);

items[i]->setColor(0, 0, 0);

items[i]->setText(itemsNames[i]);

items[i]->setAlpha(alpha\_unselected);

}

items[0]->setAlpha(alpha\_selected);

int y = windowYCenter - 4\*items[0]->h(), x, size = items.size();

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

{

x = windowXCenter - items[i]->w() / 2;

items[i]->setPos(x, y);

y += items[i]->h() \* 2;

}

TTF\_CloseFont(font);

}

void initResultItems() /\*инициализация пунктов подменю с результатами\*/

{

int windowXCenter=windowWidth/2,windowYCenter=windowHeight/2,fontSize= windowWidth\*0.02;

int textWidth, textHeight;

fontSize = fontSize>32 ? 32 : fontSize;

TTF\_Font\* font = loadFont("resourse/a\_Albionic.ttf", fontSize);

Text\* current;

std::string itemsNames[5] ={"уровень 1","уровень 2","уровень 3","уровень 4","уровень 5"};

TTF\_SizeText(font,itemsNames[0].c\_str(),&textWidth,&textHeight);

int y = windowYCenter - textHeight\*10 , x = windowXCenter - textWidth\*2.4 , num[2];

for(int i = 0; i<5; i++)

{

current = new Text();

current->setFont(font);

current->setColor(0, 0, 0);

current->setText(itemsNames[i]);

current->setAlpha(alpha\_selected);

current->setPos(x, y);

x += current->w() \* 1.5;

resultItems.push\_back(current);

}

std::fstream f("resourse/results.dat", std::ios::binary | std::ios::in);

while(f.read((char\*)num, 4\*2))

{

if(x!= resultItems[num[0]]->x())

{

x = resultItems[num[0]]->x();

y = resultItems[num[0]]->y()+resultItems[num[0]]->h()\*2;

}

current = new Text();

current->setFont(font);

current->setColor(0, 0, 0);

current->setText(std::to\_string(num[1]));

current->setAlpha(alpha\_selected);

current->setPos(x+resultItems[num[0]]->w()/2-current->w()/2, y);

resultItems.push\_back(current);

y += current->h() \* 2;

}

f.close();

TTF\_CloseFont(font);

}

void HandleKeyboardEvent(const SDL\_KeyboardEvent &event)/\*обработчик событий клавиатуры\*/

{

switch(state)

{

case 0: switch (event.state) /\*главное меню\*/

{

case SDL\_PRESSED: break;

case SDL\_RELEASED:

switch (event.keysym.sym)

{

case SDLK\_ESCAPE: exit = true;

break;

/\*установка прозрачностей и смещение текущего пункта меню\*/

case SDLK\_DOWN:items[currentItem]->setAlpha(alpha\_unselected);

++currentItem;

items[currentItem]->setAlpha(alpha\_selected);

break;

case SDLK\_UP: items[currentItem]->setAlpha(alpha\_unselected);

--currentItem;

items[currentItem]->setAlpha(alpha\_selected);

break;

/\*при нажатии Enter в зависимости от текущего пункта ставим флаги\*/

case SDLK\_RETURN: switch (currentItem)

{

case 0: start = true; break;

case 1: state = 1; break;

case 2: exit = true; break;

}

break;

}

break;

}

break;

case 1: switch (event.state) /\*подменю результатов\*/

{

case SDL\_PRESSED: break;

case SDL\_RELEASED:

switch (event.keysym.sym)

{

case SDLK\_ESCAPE: state = 0;

break;

}

break;

};

break;

}

}

void setStart(bool p) /\*установка начала игры \*/

{

start = p;

}

void reset() /\*сброс меню\*/

{

int size = resultItems.size();

state = 0;

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

resultItems[i]->clearText();

resultItems.clear();

}

};

struct levelResult

{

int number, time;

};

bool sortLevelResult(struct levelResult i, struct levelResult j)

{

return i.time < j.time;

}

class Game /\*класс игра\*/

{

bool isRun, mapInitFlag;/\*isRun – флаг о работе игры, mapInitFlag – флаг загрузки карты\*/

int gameState, currentMap; /\*gameState – состояние игры, в меню – 0, в игре - 1\*/

SDL\_Event e; /\*currentMap - номер текущей карты\*/

MainMenu menu;

Level level;

Camera camera;

Player player;

/\*названий файлов с уровнями\*/

char mapNames[5][20]{"resourse/lab1.dat", "resourse/lab2.dat", "resourse/lab3.dat","resourse/lab4.dat", "resourse/lab5.dat"};

public:

Game()

{

currentMap = 0;

mapInitFlag = true;

SDL\_Texture \*t = IMG\_LoadTexture(renderer, "resourse/mouse.png");

player.setTexture(t);

camera.setPlayer(&player);

level.setPlayer(&player);

level.setCamera(&camera);

menu.initItems();

menu.initResultItems();

gameState = 0;

isRun = true;

};

~Game(){};

std::vector<Object\*> readMap(char\* filename) /\*чтение карты из файла\*/

{

/\*структура бинарного файла – тип объекта, x, y, ширина, высота, числа по 4 байта\*/

std::vector<Object\*> objects;

std::fstream f(filename, std::ios::binary| std::ios::in);

int num[5];

while(f.read((char\*)num, 4\*5))

{

switch(num[0]) /\*создаем объекты в зависимости от типа\*/

{

case 0: objects.push\_back(new HorisontalWall(num[1],num[2],num[3],num[4]));

break;

case 1: objects.push\_back(new VerticalWall(num[1],num[2],num[3],num[4]));

break;

case 2: objects.push\_back(new Cheese(num[1],num[2],num[3],num[4]));

break;

case 3: objects.push\_back(new Spawn(num[1],num[2],num[3],num[4]));

break;

}

}

f.close();

return objects;

}

void handleEvents() /\*обработчик событий\*/

{

if(SDL\_PollEvent(&e) !=0)

switch(e.type)

{

case SDL\_QUIT: isRun = false;

break;

case SDL\_KEYDOWN:

case SDL\_KEYUP:

{

handleKeyboardEvents(e.key);

break;

}

}

if(level.completed()) /\*если уровень пройден, то сохраняем результаты и переходим к следующему уровню\*/

{

struct levelResult temp;

std::vector<struct levelResult> resultsCurrentLevel, resultsOtherLevels;

temp.number = currentMap;

temp.time = level.getCompletionTime();

resultsCurrentLevel.push\_back(temp);

level.clear();

std::fstream f("resourse/results.dat", std::ios::binary| std::ios::in);

while(f.read((char\*)&temp, 2\*4))

{

if(temp.number == currentMap)

resultsCurrentLevel.push\_back(temp);

else

resultsOtherLevels.push\_back(temp);

}

f.close();

std::sort(resultsCurrentLevel.begin(), resultsCurrentLevel.end(), sortLevelResult);

int size1 = resultsCurrentLevel.size();

int size2 = resultsOtherLevels.size();

f.open("resourse/results.dat", std::ios::out | std::ios::binary | std::ios::trunc);

for(int i=0; i<10 && i<size1; i++)

f.write((char\*)&(resultsCurrentLevel[i]), 4\*2);

for(int i=0; i<size2; i++)

f.write((char\*)&(resultsOtherLevels[i]), 4\*2);

f.close();

resultsCurrentLevel.clear();

resultsOtherLevels.clear();

menu.reset();

mapInitFlag = true;

currentMap++;

}

if(level.isQuitToMenu() || currentMap > 4) /\*если в уровне нажата “q” или карт больше нет, то выходим в меню, и сбрасываем уровень\*/

{

mapInitFlag = true;

level.clear();

menu.reset();

menu.initResultItems();

menu.setStart(false);

currentMap = 0;

gameState = 0;

}

if(gameState && mapInitFlag) /\*если была запущена игра и флаг карты = 1 \*/

{

mapInitFlag = false;

std::vector<Object\*> objects = readMap(mapNames[currentMap]);

player.setObjects(objects);

level.init(new Labyrinth(objects));

}

}

void run() /\*запуск приложения\*/

{

while(isRun)

{

handleEvents(); /\*обрабатываем события\*/

gameState = menu.isStart();

isRun = !menu.isExit() && isRun;

draw();

SDL\_Delay(33); /\*кол-во кадров в секунду = 30\*/

}

}

void handleKeyboardEvents(const SDL\_KeyboardEvent& e) /\*обработчик событий клавиватуры\*/

{

switch(gameState) /\*зависимости от состояний игры, передаем событий нужному объекту\*/

{

case 0: menu.HandleKeyboardEvent(e); break;

case 1: level.handleKeyboardEvent(e); break;

}

}

void draw()

{

SDL\_SetRenderDrawColor(renderer, 255, 255, 255, 0); /\*очищаем рендер\*/

SDL\_RenderClear(renderer);

switch(gameState) /\*зависимости от состояний игры, обрисовываем нужный объект\*/

{

case 0: menu.draw(); break;

case 1: level.draw();break;

}

SDL\_RenderPresent(renderer);

}

};

#endif // CLASSES\_H

### **2. 4. Описание основного исходного файла**

Ниже приведен код файла main.cpp.

#include <iostream>

#include <SDL2/SDL.h>

#include <SDL2/SDL\_image.h>

#include <SDL2/SDL\_ttf.h>

#include "classes.h"

using namespace std;

#define FULLSCREEN 0

#define DEFAULTWIDTH 500

#define DEFAULTHEIGHT 500

int main(int argc, char \*argv[])

{

setlocale(LC\_ALL, "rus");

SDL\_Init(SDL\_INIT\_EVERYTHING);

SDL\_DisplayMode displayMode;

if (SDL\_GetDesktopDisplayMode(0, &displayMode) != 0)

{

SDL\_Log("SDL\_GetDesktopDisplayMode failed: %s", SDL\_GetError());

return 1;

}

/\*если запускается в оконном режиме, то получаем координаты, чтобы окно было по центру\*/

int windowPosX = displayMode.w/2-DEFAULTWIDTH/2,windowPosY = displayMode.h/2-DEFAULTHEIGHT/2;

/\*создаем окно\*/

window = SDL\_CreateWindow("Game", windowPosX\*!FULLSCREEN,

windowPosY\*!FULLSCREEN,

DEFAULTWIDTH\*!FULLSCREEN,

DEFAULTWIDTH\*!FULLSCREEN,

SDL\_WINDOW\_FULLSCREEN\_DESKTOP\*FULLSCREEN);

/\*проверка инициализации\*/

if (window == nullptr)

{

std::cout << "SDL\_CreateWindow Error: " << SDL\_GetError() << std::endl;

return 1;

}

renderer = SDL\_CreateRenderer(window, -1, 0);

if (renderer == nullptr)

{

std::cout << "SDL\_CreateRenderer Error: " << SDL\_GetError() << std::endl;

return 1;

}

if (TTF\_Init() != 0)

{

std::cout << "TTF\_Init: " << TTF\_GetError() << std::endl;

return 1;

}

IMG\_Init(IMG\_INIT\_PNG);

/\*получаем размер полотна окна\*/

SDL\_GL\_GetDrawableSize(window, &windowWidth, &windowHeight);

Game game;

game.run();

SDL\_DestroyRenderer(renderer);

SDL\_DestroyWindow(window);

SDL\_Quit();

return 1;

}

### **2. 5. Описание изображений**

|  |  |
| --- | --- |
| C:\Users\basic\AppData\Local\Microsoft\Windows\INetCache\Content.Word\cheese.png | C:\Users\basic\AppData\Local\Microsoft\Windows\INetCache\Content.Word\mouse — копия.png |
| Рисунок 4 – Сыр | Рисунок 5 – Игрок |

### **2. 6. Результаты работы программы**

На рисунках 5-8 предоставлено меню, взаимодействие происходит с помощью стрелок и клавиши «Enter».

|  |
| --- |
|  |
| Рисунок 6 – Главное меню, выбран пункт «Старт» |
|  |
| Рисунок 7 – Главное меню, выбран пункт «Результаты» |
|  |
| Рисунок 8 – Главное меню, выбран пункт «Выход» |

На рисунках 9-10 представлено подменю «Результаты», в нем можно просмотреть результаты прохождения уровней в секундах. 10 лучших результатов по каждому уровню хранятся в файле. На рисунке 9 ни один уровень пока не был пройден.

|  |
| --- |
|  |
| Рисунок 9 – Результаты |
|  |
| Рисунок 10 – Результаты |

На рисунках 11-12 показан процесс игры, после сбора сыра будет переход на следующий уровень, выход в меню будет произведен после прохождения всех уровней или нажатия клавиши «q».

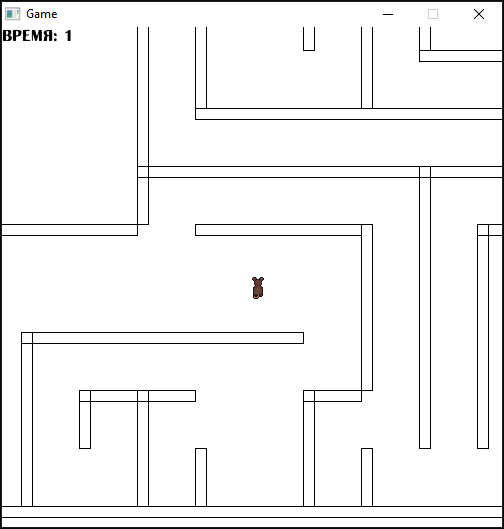


Рисунок 11 – Процесс игры

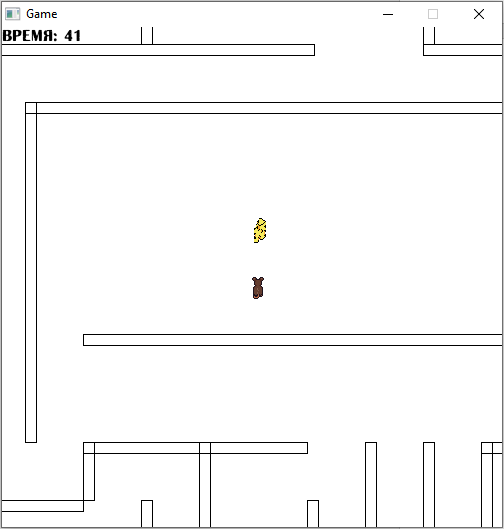


Рисунок 12 – Процесс игры

## Заключение

В результате проделанной работы были приобретены навыки написания программ на языке C++ с графическим пользовательским интерфейсом с использованием библиотеки SDL 2.0 и ее подключаемых библиотек.

Разработана компьютерная игра.

## Список использованных источников

1. <https://www.libsdl.org/>
2. <https://www.libsdl.org/projects/SDL_ttf/>
3. <https://www.libsdl.org/projects/SDL_image/>
4. <https://www.lazyfoo.net/tutorials/SDL>