Содержимое файла *\_1\_Class\_Draw*:

#pragma once

#include<SFML/Graphics.hpp>

class Draw

{

public:

virtual void draw(bool, sf::RenderWindow& )=0;

};

Содержимое файла *\_1\_Class\_GhostManager.h*:

#pragma once

#include<array>

#include"Ghost.h"

#include"\_1Class\_SimpleObject.h"

class GhostManager:public Draw

{

private:

//The ghosts will switch between the scatter mode and the chase mode before permanently chasing Pacman.

unsigned char current\_wave; // 0 1 2 3 4 5 6 7

unsigned short wave\_timer;

std::array<Ghost, 4> ghosts;

public:

GhostManager();

void update(unsigned char i\_level, std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH>& i\_map, Pacman& i\_pacman);

void draw(bool i\_flash, sf::RenderWindow& i\_window) override;

void reset(unsigned char i\_level, const std::array<SimpleObject::Position, 4>& i\_ghost\_positions);

};

Содержимое файла *\_1\_Class\_SimpleObject*:h

#pragma once

#include<SFML/Graphics.hpp>

class SimpleObject

{

public:

class Position

{

public:

Position() :x(0), y(0) {}

bool operator==(const Position& i\_position)

{

return this->x == i\_position.x && this->y == i\_position.y;

}

short x;

short y;

};

Position get\_position();

void set\_position(unsigned short x, unsigned short y);

protected:

Position position;

};

Содержимое файла *\_1\_Class\_MovingObject*:h

#pragma once

#include"\_1Class\_SimpleObject.h"

#include"\_1\_Class\_Draw.h"

class MovingObject:public SimpleObject,public Draw

{

public:

unsigned char get\_direction();

void set\_direction(unsigned char direction);

unsigned short get\_animation\_timer();

void set\_animation\_timer(unsigned short i\_animation\_timer);

protected:

unsigned char direction;

unsigned short animation\_timer;

};

Содержимое файла *AttentionWindow*:h

*#pragma once*

*#include<SFML/Graphics.hpp>*

*void AttentionWindow(sf::RenderWindow& Play, sf::RenderWindow& MENU, sf::Font& FontForOutput);*

Содержимое файла *Class\_queue*:h

#pragma once

#include<iostream>

using namespace std;

template<class T>

class Myqueue

{

public:

Myqueue();

void push(T variable);

void pop();

T& front();

T& back();

T& peek(int id);

void print();

bool isEmpty();

void clean();

int amount();

void pop\_back();

~Myqueue();

private:

struct Node

{

Node(T var);

Node\* next;

Node\* previous;

T data;

};

struct head\_and\_tail

{

head\_and\_tail();

Node\* head;

Node\* tail;

};

head\_and\_tail h\_t;

};

template<class T>

Myqueue<T>::Myqueue()

{

}

template<class T>

void Myqueue<T>::push(T variable)

{

if (h\_t.head == nullptr && h\_t.tail == nullptr)

{

h\_t.head = new Node(variable);

h\_t.tail = h\_t.head;

}

else

{

Node\* temp;

temp = h\_t.tail;

h\_t.tail = new Node(variable);

h\_t.tail->next = temp;

temp->previous = h\_t.tail;

}

}

template<class T>

void Myqueue<T>::pop()

{

if (h\_t.head == nullptr && h\_t.tail == nullptr)

{

std::cout << "Нечего удалять!" << std::endl;

return;

}

else

if (h\_t.head == h\_t.tail)

{

delete h\_t.head;

h\_t.head = h\_t.tail = nullptr;

}

else

{

Node\* temp;

temp = (h\_t.head)->previous;

delete h\_t.head;

h\_t.head = temp;

}

}

template<class T>

T& Myqueue<T>::front()

{

if (h\_t.head == nullptr)

{ }

//throw Exception("fail in method front");

return (h\_t.head)->data;

}

template<class T>

T& Myqueue<T>::back()

{

if (h\_t.tail == nullptr)

{ }

//throw Exception("fail in method back");

return (h\_t.tail)->data;

}

template<class T>

T& Myqueue<T>::peek(int id)

{

Node\* temp;

temp = h\_t.head;

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

{

if (temp == nullptr)

{

std::cout << "Элемента под номером: " << id << " нету!" << std::endl;

//throw Exception("fail in method peek");

}

temp = temp->previous;

}

if (temp == nullptr)

{

std::cout << "Элемента под номером: " << id << " нету!" << std::endl;

//throw Exception("fail in method peek");

}

return temp->data;

}

template<class T>

void Myqueue<T>::print()

{

while (h\_t.head != nullptr)

{

std::cout << h\_t.head->data << std::endl;

pop();

}

}

template<class T>

Myqueue<T>::Node::Node(T var)

{

data = var;

next = previous = nullptr;

}

template<class T>

Myqueue<T>::head\_and\_tail::head\_and\_tail()

{

head = tail = nullptr;

}

template<class T>

bool Myqueue<T>::isEmpty()

{

if (h\_t.head == nullptr)

{

return true;

}

else

{

return false;

}

}

template<class T>

void Myqueue<T>::clean()

{

while (h\_t.head != nullptr)

{

pop();

}

}

template<class T>

int Myqueue<T>::amount()

{

int count = 0;

Node\* temp;

temp = h\_t.head;

while (temp != nullptr)

{

count++;

temp = temp->previous;

}

return count;

}

template<class T>

Myqueue<T>::~Myqueue()

{

clean();

}

template<class T>

void Myqueue<T>::pop\_back()

{

if (h\_t.head == nullptr && h\_t.tail == nullptr)

{

std::cout << "Нечего удалять!" << std::endl;

return;

}

else

if (h\_t.head == h\_t.tail)

{

delete h\_t.tail;

h\_t.head = h\_t.tail = nullptr;

}

else

{

Node\* temp;

temp = (h\_t.tail)->next;

delete h\_t.tail;

h\_t.tail = temp;

}

}

Содержимое файла *ConvertSketch*:h

#pragma once

#include <array>

#include <string>

#include <SFML/Graphics.hpp>

#include "Global.h"

#include "Pacman.h"

#include "\_1Class\_SimpleObject.h"

std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH> convert\_sketch(const std::array<std::string, MAP\_HEIGHT>& i\_map\_sketch, std::array<SimpleObject::Position, 4>& i\_ghost\_positions, Pacman& i\_pacman);

Содержимое файла *DrawMap*:h

#pragma once

#include <array>

#include <SFML/Graphics.hpp>

#include "Global.h"

void draw\_map(const std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH>& i\_map, sf::RenderWin

Содержимое файла *FillFileAfterRemove*:h

#pragma once

#include<fstream>

#include"Class\_queue.h"

#include"MenuRecords.h"

#include"Global.h"

void FillFileAfterRemove(fstream& Records, MenuRecords& MenRec);

Содержимое файла *GameCycle*:h

#pragma once

#include<chrono>

#include<SFML/Graphics.hpp>

#include"Global.h"

#include<array>

#include<thread>

#include<ctime>

#include<iostream>

#include<Windows.h>

#include<fstream>

#include"DrawMap.h"

#include"MenuRecords.h"

#include<iostream>

#include"Pacman.h"

#include"Ghost.h"

#include"\_1\_Class\_GhostManager.h"

#include"\_1Class\_SimpleObject.h"

#include"FillFileAfterRemove.h"

#include"AttentionWindow.h"

#include"ConvertSketch.h"

#include"MainMenu.h"

void GameCycle(sf::RenderWindow& MENU, sf::Font& FontForOutput, sf::Text& TextForOutput);

Содержимое файла *Ghost*:h

#pragma once

#include"1\_Class\_MovingObject.h"

#include"Pacman.h"

#include<fstream>

#include"Class\_queue.h"

#include"Global.h"

#include"IsInFileNewRecord.h"

//0 - Red

//1 - Pink

//2 - Blue

//3 - Orange

class Ghost:public MovingObject

{

public:

Ghost(unsigned char i\_id);

void switch\_mode();

void reset(const Position& i\_home, const Position& i\_home\_exit);

void update(unsigned char i\_level, std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH>& i\_map, Ghost& i\_ghost\_0, Pacman& i\_pacman);

void update\_target(unsigned char i\_pacman\_direction, const Position& i\_ghost\_0\_position, const Position& i\_pacman\_position);

float get\_target\_distance(unsigned char i\_direction);

bool pacman\_collision(const Position& i\_pacman\_position);

void draw(bool i\_flash, sf::RenderWindow& i\_window)override;

private:

bool movement\_mode;

unsigned char frightened\_mode;

unsigned char id;

bool use\_door;

unsigned char frightened\_speed\_timer;

Position home;

Position home\_exit;

Position target;

};

Содержимое файла *Global*:h

#pragma once

#include<iostream>

const std::string EMPTY\_RECORD = "- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -";

const int AMOUNTRECORDS = 5;

constexpr unsigned char CELL\_SIZE = 16;

constexpr unsigned char FONT\_HEIGHT = 16;

constexpr unsigned char GHOST\_1\_CHASE = 2;

constexpr unsigned char GHOST\_2\_CHASE = 1;

constexpr unsigned char GHOST\_3\_CHASE = 4;

constexpr unsigned char GHOST\_ANIMATION\_FRAMES = 6;

constexpr unsigned char GHOST\_ANIMATION\_SPEED = 4;

constexpr unsigned char GHOST\_ESCAPE\_SPEED = 4;

constexpr unsigned char GHOST\_FRIGHTENED\_SPEED = 3;

constexpr unsigned char GHOST\_SPEED = 1;

constexpr unsigned char MAP\_HEIGHT = 21;

constexpr unsigned char MAP\_WIDTH = 21;

constexpr unsigned char PACMAN\_ANIMATION\_FRAMES = 6;

constexpr unsigned char PACMAN\_ANIMATION\_SPEED = 4;

constexpr unsigned char PACMAN\_DEATH\_FRAMES = 12;

constexpr unsigned char PACMAN\_SPEED = 2;

constexpr unsigned char SCREEN\_RESIZE = 2;

constexpr unsigned short CHASE\_DURATION = 1024;

constexpr unsigned short ENERGIZER\_DURATION = 512;

constexpr unsigned short FRAME\_DURATION = 20000;

constexpr unsigned short GHOST\_FLASH\_START = 64;

constexpr unsigned short LONG\_SCATTER\_DURATION = 512;

constexpr unsigned short SHORT\_SCATTER\_DURATION = 256;

enum Cell

{

Door,

Empty,

Energizer,

Pellet,

Wall

};

enum CollectPellets

{

No,

Yes

};

enum I\_Use\_Door

{

No\_,

Yes\_

};

Содержимое файла *IsInFileNewRecord*:h

#pragma once

#include"fstream"

#include"ctime"

#include<iostream>

#include<string>

#include"Global.h"

#include"Class\_queue.h"

void IsInFileNewRecord(int i\_level);

Содержимое файла *MainMenu*:h

#pragma once

#include<SFML/Graphics.hpp>

#include<iostream>

#define Max\_main\_menu 3

class MainMenu

{

public:

MainMenu(float width, float height);

void draw(sf::RenderWindow& window);

void MoveUp();

void MoveDown();

int MainMenuPressed()

{

return MainMenuSelected;

}

~MainMenu();

private:

int MainMenuSelected;

sf::Font font;

sf::Text mainMenu[Max\_main\_menu];

};

Содержимое файла *MapCollision*:h

#pragma once

#include <array>

#include <cmath>

#include "Global.h"

bool map\_collision(bool i\_collect\_pellets, bool i\_use\_door, short i\_x, short i\_y, std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH>& i\_map);

Содержимое файла *MenuRecords*:h

#pragma once

#include<SFML/Graphics.hpp>

#include<iostream>

#include"Global.h"

#define Max\_menu\_records 5

class MenuRecords

{

public:

MenuRecords(std::string str\_1 = EMPTY\_RECORD, std::string str\_2 = EMPTY\_RECORD, std::string str\_3 = EMPTY\_RECORD, std::string str\_4 = EMPTY\_RECORD, std::string str\_5 = EMPTY\_RECORD);

void draw(sf::RenderWindow& window);

void MoveUp();

void MoveDown();

void analyze(std::string);

void deleteRecord();

std::string\* getRecords();

int MenuRecordsPressed();

~MenuRecords();

static int position;

private:

//static int position;

int MenuRecordsSelected;

sf::Font font;

std::string infoRecords[Max\_menu\_records] = { EMPTY\_RECORD, EMPTY\_RECORD ,EMPTY\_RECORD ,EMPTY\_RECORD ,EMPTY\_RECORD };

sf::Text textRecords[Max\_menu\_records];

};

Содержимое файла *Pacman*:h

#pragma once

#include"1\_Class\_MovingObject.h"

#include"MapCollision.h"

class Pacman:public MovingObject

{

public:

Pacman();

bool get\_dead();

void set\_dead(bool i\_dead);

unsigned short get\_energizer\_timer();

void set\_energizer\_timer(unsigned short energ\_tim);

bool get\_animation\_over();

void set\_animation\_over(bool an\_tim);

void reset();

void draw(bool i\_victory, sf::RenderWindow& i\_window) override;

void update(unsigned char i\_level, std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH>& i\_map);

private:

bool dead;

unsigned short energizer\_timer;

bool animation\_over;

};

Содержимое файла 1\_Class\_GhostManager*.cpp:*

#include"\_1\_Class\_GhostManager.h"

GhostManager::GhostManager():

current\_wave(0),

wave\_timer(LONG\_SCATTER\_DURATION),

ghosts({ Ghost(0), Ghost(1), Ghost(2), Ghost(3) })

{

}

void GhostManager::update(unsigned char i\_level, std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH>& i\_map, Pacman& i\_pacman)

{

//если съел, то то просто обработка призраков

//если нет, то контролирующиеся волны

if (0 == i\_pacman.get\_energizer\_timer()) //We won't update the wave timer when Pacman is energized.

{

if (0 == wave\_timer)

{

if (7 > current\_wave)

{

current\_wave++;

for (Ghost& ghost : ghosts)

{ // cur mov\_m

ghost.switch\_mode(); // 0->1 0->1 2n+1 || 1

}

}

//I took the rules from the website.

if (1 == current\_wave % 2) // на 1 3 5 7 волнах

{

wave\_timer = CHASE\_DURATION; // 1024

}

else if (2 == current\_wave) // на 2

{

wave\_timer = static\_cast<unsigned short>(LONG\_SCATTER\_DURATION / pow(2, i\_level));// 512/2^level

}

else //4 6

{

wave\_timer = static\_cast<unsigned short>(SHORT\_SCATTER\_DURATION / pow(2, i\_level));// 256/2^level

}

}

else

{

wave\_timer--;

}

}

for (Ghost& ghost : ghosts)

{

if (1 == i\_pacman.get\_dead()) { continue; }

ghost.update(i\_level, i\_map, ghosts[0], i\_pacman);

}

}

void GhostManager::draw(bool i\_flash, sf::RenderWindow& i\_window)

{

for (Ghost& ghost : ghosts)

{

ghost.draw(i\_flash, i\_window);

}

}

void GhostManager::reset(unsigned char i\_level, const std::array<SimpleObject::Position, 4>& i\_ghost\_positions)

{

current\_wave = 0;

//This is how we're increasing the difficulty.

wave\_timer = static\_cast<unsigned short>(LONG\_SCATTER\_DURATION / pow(2, i\_level));

for (unsigned char a = 0; a < 4; a++)

{

ghosts[a].set\_position(i\_ghost\_positions[a].x, i\_ghost\_positions[a].y);

}

for (Ghost& ghost : ghosts)

{

//We use the blue ghost to get the location of the house and the red ghost to get the location of the exit.

ghost.reset(ghosts[2].get\_position(), ghosts[0].get\_position());

}

}

Содержимое файла 1\_Class\_MovingObject*.cpp:*

#include"1\_Class\_MovingObject.h"

unsigned char MovingObject::get\_direction()

{

return direction;

}

void MovingObject::set\_direction(unsigned char direction)

{

this->direction = direction;

}

unsigned short MovingObject::get\_animation\_timer()

{

return animation\_timer;

}

void MovingObject::set\_animation\_timer(unsigned short animation\_timer)

{

this->animation\_timer = animation\_timer;

}

Содержимое файла 1\_Class\_SimpleObject*.cpp:*

#include"\_1Class\_SimpleObject.h"

SimpleObject::Position SimpleObject::get\_position()

{

return position;

}

void SimpleObject::set\_position(unsigned short x, unsigned short y)

{

position.x = x; position.y = y;

}

Содержимое файла AttentionWindow*.cpp:*

#include"AttentionWindow.h"

void AttentionWindow(sf::RenderWindow& Play, sf::RenderWindow& MENU, sf::Font& FontForOutput )

{

sf::RenderWindow ATTENTION(sf::VideoMode(400, 80), "ATTENTION", sf::Style::Titlebar);

sf::Text TextForOutput[3];

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

{

TextForOutput[i].setFont(FontForOutput);

TextForOutput[i].setFillColor(sf::Color::White);

TextForOutput[i].setCharacterSize(15);

}

TextForOutput[0].setString("If you quit, you will lose a possible record");

TextForOutput[1].setString("Press 'SPACE' to exit");

TextForOutput[2].setString("Press any other button to continue");

//sf::Font FontForOutput;

//if (!FontForOutput.loadFromFile("Resources/Font/ALGER.TTF"));

TextForOutput[0].setPosition(5, 15);

TextForOutput[1].setPosition(5, 30);

TextForOutput[2].setPosition(5, 45);

while (ATTENTION.isOpen())

{

sf::Event eventA;

if (ATTENTION.pollEvent(eventA))

{

if (eventA.type == sf::Event::KeyPressed)

{

if (eventA.key.code == sf::Keyboard::Space)

{

ATTENTION.close();

Play.close();

MENU.setVisible(true);

}

if (eventA.key.code != sf::Keyboard::Space)

{

ATTENTION.close();

}

}

}

ATTENTION.clear();

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

{

ATTENTION.draw(TextForOutput[i]);

}

ATTENTION.display();

}

}

Содержимое файла Class\_queue*.cpp:*

#include "Class\_queue.h"

Содержимое файла ConvertSketch*.cpp:*

#include "ConvertSketch.h"

std::array<std::array<Cell, MAP\_WIDTH>, MAP\_HEIGHT> convert\_sketch(const std::array<std::string, MAP\_HEIGHT>& i\_map\_sketch, std::array<SimpleObject::Position, 4>& i\_ghost\_positions, Pacman& i\_pacman)

{

std::array<std::array<Cell, MAP\_WIDTH>, MAP\_HEIGHT> output\_map{};

for (unsigned char a = 0; a < MAP\_HEIGHT; a++)

{

for (unsigned char b = 0; b < MAP\_WIDTH; b++)

{

output\_map[b][a] = Cell::Empty;

switch (i\_map\_sketch[a][b])

{

case '#':

{

output\_map[b][a] = Cell::Wall;

break;

}

case '=':

{

output\_map[b][a] = Cell::Door;

break;

}

case '.':

{

output\_map[b][a] = Cell::Pellet;

break;

}

//Red ghost

case '0':

{

i\_ghost\_positions[0].x = CELL\_SIZE \* b;

i\_ghost\_positions[0].y = CELL\_SIZE \* a;

break;

}

//Pink ghost

case '1':

{

i\_ghost\_positions[1].x = CELL\_SIZE \* b;

i\_ghost\_positions[1].y = CELL\_SIZE \* a;

break;

}

//Blue (cyan) ghost

case '2':

{

i\_ghost\_positions[2].x = CELL\_SIZE \* b;

i\_ghost\_positions[2].y = CELL\_SIZE \* a;

break;

}

//Orange ghost

case '3':

{

i\_ghost\_positions[3].x = CELL\_SIZE \* b;

i\_ghost\_positions[3].y = CELL\_SIZE \* a;

break;

}

//Pacman!

case 'P':

{

i\_pacman.set\_position(CELL\_SIZE \* b, CELL\_SIZE \* a);

break;

}

case 'o':

{

output\_map[b][a] = Cell::Energizer;

}

}

}

}

return output\_map;

}

Содержимое файлаDrawMap*.cpp:*

#include"DrawMap.h"

void draw\_map(const std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH>& i\_map, sf::RenderWindow& i\_window)

{

sf::Sprite sprite;

sf::Texture texture;

texture.loadFromFile("Resources/Images/Map" + std::to\_string(CELL\_SIZE) + ".png");

sprite.setTexture(texture);

for (unsigned char a = 0; a < MAP\_WIDTH; a++)

{

for (unsigned char b = 0; b < MAP\_HEIGHT; b++)

{

sprite.setPosition(static\_cast<float>(CELL\_SIZE \* a), static\_cast<float>(CELL\_SIZE \* b));

switch (i\_map[a][b])

{

case Cell::Door:

{

sprite.setTextureRect(sf::IntRect(2 \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));// for default the whole texture is used by the rectangle that will be repsesented

i\_window.draw(sprite);

break;

}

case Cell::Energizer:

{

sprite.setTextureRect(sf::IntRect(CELL\_SIZE, CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));

i\_window.draw(sprite);

break;

}

case Cell::Pellet:

{

sprite.setTextureRect(sf::IntRect(0, CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));

i\_window.draw(sprite);

break;

}

case Cell::Wall:

{

bool down = 0;

bool left = 0;

bool right = 0;

bool up = 0;

if (b < MAP\_HEIGHT - 1)

{

if (Cell::Wall == i\_map[a][1 + b])

{

down = 1;

}

}

if (0 < a)

{

if (Cell::Wall == i\_map[a - 1][b])

{

left = 1;

}

}

else

{

left = 1;

}

if (a < MAP\_WIDTH - 1)

{

if (Cell::Wall == i\_map[1 + a][b])

{

right = 1;

}

}

else

{

right = 1;

}

if (0 < b)

{

if (Cell::Wall == i\_map[a][b - 1])

{

up = 1;

}

}

//--------------------------------------------< DISTRIBUTIVE PROPERTY! >----------------------------

sprite.setTextureRect(sf::IntRect(CELL\_SIZE \* (down + 2 \* (left + 2 \* (right + 2 \* up))), 0, CELL\_SIZE, CELL\_SIZE));

i\_window.draw(sprite);

}

}

}

}

}

Содержимое файла FillFileAfterRemove*.cpp:*

#include"FillFileAfterRemove.h"

void FillFileAfterRemove(fstream& Records, MenuRecords& MenRec)

{

Records.open("Records/RECORDS.txt", std::fstream::out);

Records.seekp(0, std::ios::beg);

std::string\* tempRecords;

tempRecords = MenRec.getRecords();

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

{

if (tempRecords[i].compare(EMPTY\_RECORD) == 0)

{

break;

}

Records << tempRecords[i] << '\n';

}

}

Содержимое файла GameCycle*.cpp:*

#include"GameCycle.h"

void GameCycle(sf::RenderWindow& MENU, sf::Font& FontForOutput, sf::Text& TextForOutput)

{

bool IsPressToStart = false;

sf::RenderWindow Play(sf::VideoMode(CELL\_SIZE \* MAP\_WIDTH \* SCREEN\_RESIZE, (FONT\_HEIGHT + CELL\_SIZE \* MAP\_HEIGHT) \* SCREEN\_RESIZE), "Pac-Man", sf::Style::Close);

Play.setView(sf::View(sf::FloatRect(0, 0, CELL\_SIZE \* MAP\_WIDTH, FONT\_HEIGHT + CELL\_SIZE \* MAP\_HEIGHT)));

bool game\_won = 0;

unsigned char level = 0;

std::array<std::string, MAP\_HEIGHT> map\_sketch = {

" ################### ", // -1|0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 | 21

" #........#........# ", // | |

" #o##.###.#.###.##o# ",

" #.................# ",

" #.##.#.#####.#.##.# ",

" #....#...#...#....# ",

" ####.### # ###.#### ",

" #.# 0 #.# ",

"#####.# ##=## #.#####",

" . #123# . ",

"#####.# ##### #.#####",

" #.# #.# ",

" ####.# ##### #.#### ",

" #........#........# ",

" #.##.###.#.###.##.# ",

" #o.#.....P.....#.o# ",

" ##.#.#.#####.#.#.## ",

" #....#...#...#....# ",

" #.######.#.######.# ",

" #.................# ",

" ################### "

};

std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH> map{};

std::array<SimpleObject::Position, 4> ghost\_positions;

GhostManager ghost\_manager;

Pacman pacman;

map = convert\_sketch(map\_sketch, ghost\_positions, pacman);// возвращает карту+ возвращает позицИИ призраков+ устанавливает позицИЮ пакмана

ghost\_manager.reset(level, ghost\_positions);// (таймер на разбегание,cur\_w=0 | (LONG\_SCATTER\_DURATION / pow(2, i\_level))) ||+ у призрака movement\_mode=0

//присвоить позиции призракам, установить цель

srand(static\_cast<unsigned>(time(0)));

std::chrono::time\_point<std::chrono::steady\_clock> previous\_time = std::chrono::steady\_clock::now();

unsigned delta\_time;

sf::Event eventP;

while (Play.isOpen())

{

if (0 == game\_won && 0 == pacman.get\_dead())

{

game\_won = 1; // состояние того, что мы еще в процессе победы СОЖРАЛИ ЛИ ВСЕ ТАБЛЕТКИ

pacman.update(level, map); // level влияет на время действия таблетки

ghost\_manager.update(level, map, pacman);

if (1 == pacman.get\_dead())

{

game\_won = 0;

goto escapePelletChecking;

}

//We're checking every cell in the map.

for (const std::array<Cell, MAP\_HEIGHT>& column : map)

{

for (const Cell& cell : column)

{

if (Cell::Pellet == cell) //And if at least one of them has a pellet.

{

game\_won = 0; //The game is not yet won.

break;

}

}

if (0 == game\_won)

{

break;

}

}

escapePelletChecking:

if (1 == game\_won)

{

pacman.set\_animation\_timer(0);

}

}

else if (1 == sf::Keyboard::isKeyPressed(sf::Keyboard::Enter)) //Restarting the game.

{

game\_won = 0;

if (1 == pacman.get\_dead())

{

level = 0;

}

else

{

level++;

}

map = convert\_sketch(map\_sketch, ghost\_positions, pacman);

ghost\_manager.reset(level, ghost\_positions);

pacman.reset();

IsPressToStart = false;

}

{

Play.clear();

if (0 == game\_won && 0 == pacman.get\_dead())

{

draw\_map(map, Play);

ghost\_manager.draw(GHOST\_FLASH\_START >= pacman.get\_energizer\_timer(), Play);

TextForOutput.setString("Level : " + std::to\_string(level));

TextForOutput.setPosition(0, CELL\_SIZE \* MAP\_HEIGHT);

Play.draw(TextForOutput);

}

pacman.draw(game\_won, Play);

if (1 == pacman.get\_animation\_over())

{

if (1 == game\_won)

{

TextForOutput.setString("Next Level!");

TextForOutput.setPosition((0.5f \* (CELL\_SIZE \* MAP\_WIDTH)) - 3 \* CELL\_SIZE, 0.5f \* CELL\_SIZE \* MAP\_HEIGHT);

Play.draw(TextForOutput);

}

else

{

TextForOutput.setString("Game over");

TextForOutput.setPosition((0.5f \* (CELL\_SIZE \* MAP\_WIDTH)) - 3 \* CELL\_SIZE, 0.5f \* CELL\_SIZE \* MAP\_HEIGHT);

Play.draw(TextForOutput);

}

}

Play.display();

if (!IsPressToStart)

{

sf::Event waitToStart;

Play.pollEvent(waitToStart);

while (waitToStart.type != sf::Event::KeyPressed)

{

Play.pollEvent(waitToStart);

}

IsPressToStart = true;

}

}

delta\_time = std::chrono::duration\_cast<std::chrono::microseconds>(std::chrono::steady\_clock::now() - previous\_time).count();

unsigned timeLeft = (delta\_time <= FRAME\_DURATION ? FRAME\_DURATION - delta\_time : 0);

std::this\_thread::sleep\_for(std::chrono::microseconds(timeLeft));

previous\_time = std::chrono::steady\_clock::now();

if (Play.pollEvent(eventP))// pool опрос

{

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

{

AttentionWindow(Play, MENU, FontForOutput);

}

if (eventP.type == sf::Event::KeyPressed)

{

if (eventP.key.code == sf::Keyboard::Escape)

{

AttentionWindow(Play, MENU, FontForOutput);

}

}

}

}

}

Содержимое файла Ghost*.cpp:*

#include"Ghost.h"

Ghost::Ghost(unsigned char i\_id):id(i\_id)

{

}

void Ghost::switch\_mode()

{

movement\_mode = 1 - movement\_mode;

}

void Ghost::reset(const Position& i\_home, const Position& i\_home\_exit)

{

movement\_mode = 0;

use\_door = 0 < id;

direction = 0;

frightened\_mode = 0;

frightened\_speed\_timer = 0;

animation\_timer = 0;

home = i\_home;

home\_exit = i\_home\_exit;

target = i\_home\_exit;

}

void Ghost::update(unsigned char i\_level, std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH>& i\_map, Ghost& i\_ghost\_0, Pacman& i\_pacman)

{

//========================================================================================

// состояние(ключевые моменты для перехода состояний) |

//выбрать цель, |

//коллизия со стеной, |

//выбор кротчайшего пути при (0,2 режимах), |

//выбор рандомного пути при режиме (1), |

//возможность попасть в тунель, |

//взаимодействие с пакманом |

//========================================================================================

bool move = 0;

//If this is greater than 1, that means that the gohst has reached the intersection.(перекресток)

//We don't consider the way back as an available way.

unsigned char available\_ways = 0;

unsigned char speed = GHOST\_SPEED;

std::array<bool, 4> walls{};

//Here the gohst starts and stops being frightened.

if (0 == frightened\_mode && i\_pacman.get\_energizer\_timer() == ENERGIZER\_DURATION / pow(2, i\_level))

{ // не были напуганы(0) и съел таблетку

frightened\_speed\_timer = GHOST\_FRIGHTENED\_SPEED;

frightened\_mode = 1;

}

else if (1 == frightened\_mode && 0 == i\_pacman.get\_energizer\_timer()) // были напуганы(1) и не глотал

{

frightened\_mode = 0;

}

// (0 4 8 12 16)

if (2 == frightened\_mode && 0 == position.x % GHOST\_ESCAPE\_SPEED && 0 == position.y % GHOST\_ESCAPE\_SPEED)

//

//if (2 == frightened\_mode) // застрянут вдоль или поперек в режиме 2

{ // в ужасе(2) и их съел пакман

speed = GHOST\_ESCAPE\_SPEED;

}

update\_target(i\_pacman.get\_direction(), i\_ghost\_0.get\_position(), i\_pacman.get\_position());

//This is so clean! I could spend hours staring at it.

walls[0] = map\_collision(I\_Use\_Door::No\_, use\_door, speed + position.x, position.y, i\_map);

walls[1] = map\_collision(I\_Use\_Door::No\_, use\_door, position.x, position.y - speed, i\_map);

walls[2] = map\_collision(I\_Use\_Door::No\_, use\_door, position.x - speed, position.y, i\_map);

walls[3] = map\_collision(I\_Use\_Door::No\_, use\_door, position.x, speed + position.y, i\_map);

if (1 != frightened\_mode) // 0 2

{

//I used 4 because using a number between 0 and 3 will make the gohst move in a direction it can't move.

unsigned char optimal\_direction = 4;

//The gohst can move.

move = 1;

for (unsigned char a = 0; a < 4; a++)

{

//Gohsts can't turn back! (Unless they really have to) // если справа есть стена, а призрак идет влево, то зачем тратить ресурсы на обработку коллизии

if (a == (2 + direction) % 4) //a(w)| 0 1 2 3

{ // --------------------------------------------

continue; //(d)0| 2 2 (2) 2

} // 1| 3 3 3 (3)

else if (0 == walls[a]) // 2| (0) 0 0 0

{ // 3| 1 (1) 1 1

if (4 == optimal\_direction)// пока нету выбора

{

optimal\_direction = a;

}

available\_ways++;

if (get\_target\_distance(a) < get\_target\_distance(optimal\_direction))

{

//The optimal direction is the direction that's closest to the target.

optimal\_direction = a;

}

}

}

if (1 < available\_ways)

{

direction = optimal\_direction;

}

else // такого в данной карте нету, но это на случай, если зашел в тупик

{

if (4 == optimal\_direction)

{

direction = (2 + direction) % 4; // такого в данной карте нету, но это на случай, если зашел в тупик

}

else

{

direction = optimal\_direction;

}

}

}

else // режим 1(съел таблетку)

{

unsigned char random\_direction = rand() % 4; // 0 1 2 3

if (0 == frightened\_speed\_timer)

{

move = 1;

frightened\_speed\_timer = GHOST\_FRIGHTENED\_SPEED;

for (unsigned char a = 0; a < 4; a++)

{

//They can't turn back even if they're frightened.

if (a == (2 + direction) % 4)

{

continue;

}

else if (0 == walls[a])

{

available\_ways++;

}

}

if (0 < available\_ways)

{

while (1 == walls[random\_direction] || random\_direction == (2 + direction) % 4)

{

//We keep picking a random direction until we can use it.

random\_direction = rand() % 4;

}

direction = random\_direction;

}

else

{

//If there's no other way, it turns back.

direction = (2 + direction) % 4; // опять же, такого нету в данной игре, так как нету тупиков

}

}

else

{

frightened\_speed\_timer--;

}

}

//If the gohst can move, we move it.

if (1 == move)

{

switch (direction)

{

case 0:

{

position.x += speed;

break;

}

case 1:

{

position.y -= speed;

break;

}

case 2:

{

position.x -= speed;

break;

}

case 3:

{

position.y += speed;

}

}

if (-CELL\_SIZE >= position.x)

{

position.x = CELL\_SIZE \* MAP\_WIDTH - speed;

}

else if (position.x >= CELL\_SIZE \* MAP\_WIDTH)

{

position.x = speed - CELL\_SIZE;

}

}

if (1 == pacman\_collision(i\_pacman.get\_position()))

{

if (0 == frightened\_mode) //When the gohst is not frightened and collides with Pacman, we kill Pacman.

{

i\_pacman.set\_dead(1);

IsInFileNewRecord(i\_level);

}

else //Otherwise, the gohst starts running towards the house.

{

use\_door = 1;

frightened\_mode = 2;

target = home;

}

}

}

void Ghost::update\_target(unsigned char i\_pacman\_direction, const Position& i\_ghost\_0\_position, const Position& i\_pacman\_position)

{

if (1 == use\_door) //If the gohst can use the door.

{

if (position == target)

{

if (home\_exit == target) //If the gohst has reached the exit.

{

use\_door = 0; //It can no longer use the door.

}

else if (home == target) //If the gohst has reached its home.

{

frightened\_mode = 0; //It stops being frightened.

target = home\_exit; //And starts leaving the house.

}

}

}

else

{

if (0 == movement\_mode) //The scatter mode

{

//Each gohst goes to the corner it's assigned to.

switch (id)

{

case 0:

{

target.x = CELL\_SIZE \* (MAP\_WIDTH - 1); target.y = 0;

break;

}

case 1:

{

target.x = 0; target.y = 0;

break;

}

case 2:

{

target.x = CELL\_SIZE \* (MAP\_WIDTH - 1); target.y=CELL\_SIZE\* (MAP\_HEIGHT - 1);

break;

}

case 3:

{

target.x = 0; target.y= CELL\_SIZE\* (MAP\_HEIGHT - 1);

}

}

}

else //The chase mode

{

switch (id)

{

case 0: //The red gohst will chase Pacman.

{

target = i\_pacman\_position;

break;

}

case 1: //The pink gohst will chase the 4th cell in front of Pacman.

{

target = i\_pacman\_position;

switch (i\_pacman\_direction)

{

case 0:

{

target.x += CELL\_SIZE \* GHOST\_1\_CHASE;

break;

}

case 1:

{

target.y -= CELL\_SIZE \* GHOST\_1\_CHASE;

break;

}

case 2:

{

target.x -= CELL\_SIZE \* GHOST\_1\_CHASE;

break;

}

case 3:

{

target.y += CELL\_SIZE \* GHOST\_1\_CHASE;

break;

}

}

break;

}

case 2: //The blue gohst.

{

target = i\_pacman\_position;

//Getting the second cell in front of Pacman.

switch (i\_pacman\_direction)

{

case 0:

{

target.x += CELL\_SIZE \* GHOST\_2\_CHASE;

break;

}

case 1:

{

target.y -= CELL\_SIZE \* GHOST\_2\_CHASE;

break;

}

case 2:

{

target.x -= CELL\_SIZE \* GHOST\_2\_CHASE;

break;

}

case 3:

{

target.y += CELL\_SIZE \* GHOST\_2\_CHASE;

}

}

target.x += target.x - i\_ghost\_0\_position.x;

target.y += target.y - i\_ghost\_0\_position.y;

break;

}

case 3: //The orange gohst will chase Pacman until it gets close to him. Then it'll switch to the scatter mode.

{

if (CELL\_SIZE \* GHOST\_3\_CHASE <= sqrt(pow(position.x - i\_pacman\_position.x, 2) + pow(position.y - i\_pacman\_position.y, 2)))

{

target = i\_pacman\_position;

}

else

{

target.x = 0; target.y = CELL\_SIZE \* (MAP\_HEIGHT - 1);

}

}

}

}

}

}

float Ghost::get\_target\_distance(unsigned char i\_direction)

{

short x = position.x;

short y = position.y;

//We'll imaginarily move the gohst in a given direction and calculate the distance to the target.

switch (i\_direction)

{

case 0:

{

x += GHOST\_SPEED;

break;

}

case 1:

{

y -= GHOST\_SPEED;

break;

}

case 2:

{

x -= GHOST\_SPEED;

break;

}

case 3:

{

y += GHOST\_SPEED;

}

}

return static\_cast<float>(sqrt(pow(x - target.x, 2) + pow(y - target.y, 2)));

}

bool Ghost::pacman\_collision(const Position& i\_pacman\_position)

{

if (pow(position.x - i\_pacman\_position.x, 2) < pow(CELL\_SIZE, 2) && pow(position.y - i\_pacman\_position.y, 2) < pow(CELL\_SIZE, 2))

{

return 1;

}

return 0;

}

void Ghost::draw(bool i\_flash, sf::RenderWindow& i\_window)

{

//Current frame of the animation.

unsigned char body\_frame = static\_cast<unsigned char>(floor(animation\_timer / static\_cast<float>(GHOST\_ANIMATION\_SPEED)));

sf::Sprite body;

sf::Sprite face;

sf::Texture texture;

texture.loadFromFile("Resources/Images/Ghost" + std::to\_string(CELL\_SIZE) + ".png");

body.setTexture(texture);

body.setPosition(position.x, position.y);

body.setTextureRect(sf::IntRect(CELL\_SIZE \* body\_frame, 0, CELL\_SIZE, CELL\_SIZE));

face.setTexture(texture);

face.setPosition(position.x, position.y);

//The "I'm not frightened" look.

if (0 == frightened\_mode)

{

switch (id)

{

case 0:

{

//Red color

body.setColor(sf::Color(255, 0, 0));

break;

}

case 1:

{

//Pink color

body.setColor(sf::Color(255, 182, 255));

break;

}

case 2:

{

//Cyan color (I still don't understand why they called it blue)

body.setColor(sf::Color(0, 255, 255));

break;

}

case 3:

{

//Orange color

body.setColor(sf::Color(255, 182, 85));

}

}

face.setTextureRect(sf::IntRect(CELL\_SIZE \* direction, CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));

i\_window.draw(body);

}

else if (1 == frightened\_mode)

{

body.setColor(sf::Color(36, 36, 255));

face.setTextureRect(sf::IntRect(4 \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));

if (1 == i\_flash && 0 == body\_frame % 2)

{

body.setColor(sf::Color(255, 255, 255));

face.setColor(sf::Color(255, 0, 0));

}

else

{

body.setColor(sf::Color(36, 36, 255));

face.setColor(sf::Color(255, 255, 255));

}

i\_window.draw(body);

}

else

{

face.setTextureRect(sf::IntRect(CELL\_SIZE \* direction, 2 \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE));

}

i\_window.draw(face);

//--------------------------------------< This is to prevent overflowing. >-

animation\_timer = (1 + animation\_timer) % (GHOST\_ANIMATION\_FRAMES \* GHOST\_ANIMATION\_SPEED);

}

Содержимое файла IsInDileNewRecord*.cpp:*

*#include"IsInFileNewRecord.h"*

*void IsInFileNewRecord(int i\_level)*

*{*

*std::fstream Records;*

*Records.open("Records/RECORDS.txt", std::fstream::in);*

*std::time\_t t = std::time(nullptr);*

*std::tm\* now = std::localtime(&t);*

*char buffer[128];*

*strftime(buffer, sizeof(buffer), "%m-%d-%Y %X", now);*

*char temp;*

*temp = buffer[3];*

*buffer[3] = buffer[0];*

*buffer[0] = temp;*

*temp = buffer[4];*

*buffer[4] = buffer[1];*

*buffer[1] = temp;*

*std::string PossibleNewRecord = "Reached level: " + std::to\_string(i\_level) + ": " + buffer;*

*//PossibleNewRecord = "Reached level "+i\_level+ ": ";*

*////if there is not the file with records*

*if (!Records.is\_open())*

*{*

*Records.close();*

*Records.open("Records/RECORDS.txt", std::fstream::out);*

*Records.seekp(0, std::ios::beg);*

*Records << PossibleNewRecord << '\n';*

*Records.close();*

*return;*

*}*

*///////////////////////////////////////////*

*else*

*{*

*Myqueue<std::string> queueOfRecords;*

*std::string Line;*

*int count = 0;*

*bool isPossibleRecordThere = false;*

*while (!Records.eof())*

*{*

*Line = "";*

*std::getline(Records, Line, '\n');*

*if (Line.compare("") == 0) { continue; }*

*queueOfRecords.push(Line);*

*}*

*Records.close();*

*Records.open("Records/RECORDS.txt", std::fstream::out);*

*Records.seekp(0, std::ios::beg);*

*std::string ToFile;*

*while (!queueOfRecords.isEmpty())*

*{*

*ToFile = queueOfRecords.front();*

*if (ToFile.compare(PossibleNewRecord) > 0)*

*{*

*Records << ToFile << '\n';*

*count++;*

*queueOfRecords.pop();*

*}*

*else*

*{*

*Records << PossibleNewRecord << '\n';*

*count++;*

*isPossibleRecordThere= true;*

*break;*

*}*

*}*

*if(isPossibleRecordThere)*

*while(count<AMOUNTRECORDS && !queueOfRecords.isEmpty() )*

*{*

*ToFile = queueOfRecords.front();*

*Records << ToFile << '\n';*

*count++;*

*queueOfRecords.pop();*

*}*

*else*

*{*

*if (count == 0 || ( count>0 && count<AMOUNTRECORDS))*

*{*

*Records << PossibleNewRecord << '\n';*

*}*

*}*

*Records.close();*

*return;*

*}*

*}*

Содержимое файла Main*.cpp:*

*#include<chrono>*

*#include<SFML/Graphics.hpp>*

*#include"Global.h"*

*#include<array>*

*#include<thread>*

*#include<ctime>*

*#include<iostream>*

*#include<Windows.h>*

*#include<fstream>*

*#include"DrawMap.h"*

*#include"MenuRecords.h"*

*#include<iostream>*

*#include"Pacman.h"*

*#include"Ghost.h"*

*#include"\_1\_Class\_GhostManager.h"*

*#include"\_1Class\_SimpleObject.h"*

*#include"FillFileAfterRemove.h"*

*#include"AttentionWindow.h"*

*#include"ConvertSketch.h"*

*#include"MainMenu.h"*

*#include"GameCycle.h"*

*//std::string EMPTY\_RECORD = "- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -";*

*void main()*

*{*

*//sf::VideoMode(CELL\_SIZE \* MAP\_WIDTH \* SCREEN\_RESIZE, (FONT\_HEIGHT + CELL\_SIZE \* MAP\_HEIGHT) \* SCREEN\_RESIZE), "Pac-Man", sf::Style::Close*

*sf::RenderWindow MENU(sf::VideoMode(CELL\_SIZE \* MAP\_WIDTH \* SCREEN\_RESIZE, (FONT\_HEIGHT + CELL\_SIZE \* MAP\_HEIGHT) \* SCREEN\_RESIZE), "Main\_Menu", sf::Style::Default);*

*MainMenu mainMenu(MENU.getSize().x, MENU.getSize().y);*

*//set background*

*sf::RectangleShape background;*

*background.setSize(sf::Vector2f(CELL\_SIZE \* MAP\_WIDTH \* SCREEN\_RESIZE, (FONT\_HEIGHT + CELL\_SIZE \* MAP\_HEIGHT) \* SCREEN\_RESIZE));*

*sf::Texture MainTexture;*

*MainTexture.loadFromFile("Resources/Images/pacman\_background.jpg");*

*background.setTexture(&MainTexture);*

*//photo to Records*

*sf::RectangleShape Rbackground;*

*Rbackground.setSize(sf::Vector2f(CELL\_SIZE \* MAP\_WIDTH \* SCREEN\_RESIZE, (FONT\_HEIGHT + CELL\_SIZE \* MAP\_HEIGHT) \* SCREEN\_RESIZE));*

*sf::Texture RptionTexture;*

*RptionTexture.loadFromFile("Resources/Images/Records\_background.png");*

*Rbackground.setTexture(&RptionTexture);*

*sf::Text TextForOutput;*

*sf::Font FontForOutput;*

*if (!FontForOutput.loadFromFile("Resources/Font/ALGER.TTF"));*

*TextForOutput.setFont(FontForOutput);*

*TextForOutput.setFillColor(sf::Color::White);*

*TextForOutput.setCharacterSize(FONT\_HEIGHT);*

*while (MENU.isOpen())*

*{*

*sf::Event event;*

*if (MENU.pollEvent(event))*

*{*

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

*{*

*MENU.close();*

*}*

*if (event.type == sf::Event::KeyReleased)*

*{*

*if (event.key.code == sf::Keyboard::Up)*

*{*

*mainMenu.MoveUp();*

*}*

*if (event.key.code == sf::Keyboard::Down)*

*{*

*mainMenu.MoveDown();*

*}*

*if (event.key.code == sf::Keyboard::Enter)*

*{*

*int x = mainMenu.MainMenuPressed();*

*if (x == 0)*

*{*

*MENU.setVisible(false);*

*GameCycle(MENU, FontForOutput, TextForOutput);*

*}*

*if (x == 1)*

*{*

*MENU.setVisible(false);*

*MenuRecords MyListOfRecords;*

*std::fstream Records;*

*Records.open("Records/RECORDS.txt", std::fstream::in);*

*sf::RenderWindow RECORDS(sf::VideoMode(CELL\_SIZE \* MAP\_WIDTH \* SCREEN\_RESIZE, (FONT\_HEIGHT + CELL\_SIZE \* MAP\_HEIGHT) \* SCREEN\_RESIZE), "RECORDS");*

*if (!Records.is\_open())*

*{*

*Records.close();*

*while (RECORDS.isOpen())*

*{*

*sf::Event Revent;*

*if (RECORDS.pollEvent(Revent))*

*{*

*if (Revent.type == sf::Event::Closed)*

*{*

*RECORDS.close();*

*MENU.setVisible(true);*

*}*

*if (Revent.type == sf::Event::KeyPressed)*

*{*

*if (Revent.key.code == sf::Keyboard::Escape)*

*{*

*RECORDS.close();*

*MENU.setVisible(true);*

*}*

*if (Revent.key.code == sf::Keyboard::Up)*

*{*

*MyListOfRecords.MoveUp();*

*}*

*if (Revent.key.code == sf::Keyboard::Down)*

*{*

*MyListOfRecords.MoveDown();*

*}*

*if (Revent.key.code == sf::Keyboard::Delete)*

*{*

*MyListOfRecords.deleteRecord();*

*}*

*}*

*}*

*RECORDS.clear();*

*RECORDS.draw(Rbackground);*

*MyListOfRecords.draw(RECORDS);*

*RECORDS.display();*

*}*

*}*

*else*

*{*

*Records.seekp(0, ios::beg);*

*while (!Records.eof())*

*{*

*std::string temp = "";*

*std::getline(Records, temp, '\n');*

*if (temp.compare("") == 0) { continue; }*

*MyListOfRecords.analyze(temp);*

*}*

*MenuRecords::position = 0;*

*Records.close();*

*while (RECORDS.isOpen())*

*{*

*sf::Event Revent;*

*if (RECORDS.pollEvent(Revent))*

*{*

*if (Revent.type == sf::Event::Closed)*

*{*

*FillFileAfterRemove(Records, MyListOfRecords);*

*Records.close();*

*RECORDS.close();*

*MENU.setVisible(true);*

*}*

*if (Revent.type == sf::Event::KeyPressed)*

*{*

*if (Revent.key.code == sf::Keyboard::Escape)*

*{*

*FillFileAfterRemove(Records, MyListOfRecords);*

*Records.close();*

*RECORDS.close();*

*MENU.setVisible(true);*

*}*

*if (Revent.key.code == sf::Keyboard::Up)*

*{*

*MyListOfRecords.MoveUp();*

*}*

*if (Revent.key.code == sf::Keyboard::Down)*

*{*

*MyListOfRecords.MoveDown();*

*}*

*if (Revent.key.code == sf::Keyboard::Delete)*

*{*

*MyListOfRecords.deleteRecord();*

*}*

*}*

*}*

*RECORDS.clear();*

*RECORDS.draw(Rbackground);*

*MyListOfRecords.draw(RECORDS);*

*RECORDS.display();*

*}*

*}*

*}*

*if (x == 2)*

*{*

*MENU.close();*

*break;*

*}*

*}*

*}*

*MENU.clear();*

*MENU.draw(background);*

*mainMenu.draw(MENU);*

*MENU.display();*

*}*

*}*

*}*

Содержимое файла MainMenu*.cpp:*

*#include"MainMenu.h"*

*MainMenu::MainMenu(float width, float height)*

*{*

*if (!font.loadFromFile("Resources/Font/ALGER.TTF"))*

*std::cout << "No font is here" << std::endl;*

*//Play*

*mainMenu[0].setFont(font);*

*mainMenu[0].setFillColor(sf::Color::Blue);*

*mainMenu[0].setString("Play");*

*mainMenu[0].setCharacterSize(70);*

*mainMenu[0].setPosition(250, 200);*

*//Records*

*mainMenu[1].setFont(font);*

*mainMenu[1].setFillColor(sf::Color::White);*

*mainMenu[1].setString("Records");*

*mainMenu[1].setCharacterSize(70);*

*mainMenu[1].setPosition(250, 300);*

*//Exit*

*mainMenu[2].setFont(font);*

*mainMenu[2].setFillColor(sf::Color::White);*

*mainMenu[2].setString("Exit");*

*mainMenu[2].setCharacterSize(70);*

*mainMenu[2].setPosition(250, 400);*

*MainMenuSelected = 0;*

*}*

*MainMenu::~MainMenu()*

*{*

*}*

*void MainMenu::draw(sf::RenderWindow& window)*

*{*

*for (int i = 0; i < Max\_main\_menu; ++i)*

*window.draw(mainMenu[i]);*

*}*

*void MainMenu::MoveUp()*

*{*

*MainMenuSelected--;*

*mainMenu[MainMenuSelected + 1].setFillColor(sf::Color::White);*

*if (MainMenuSelected == -1)*

*MainMenuSelected = Max\_main\_menu - 1;*

*mainMenu[MainMenuSelected].setFillColor(sf::Color::Blue);*

*}*

*void MainMenu::MoveDown()*

*{*

*MainMenuSelected++;*

*mainMenu[MainMenuSelected - 1].setFillColor(sf::Color::White);*

*if (MainMenuSelected == Max\_main\_menu)*

*MainMenuSelected = 0;*

*mainMenu[MainMenuSelected].setFillColor(sf::Color::Blue);*

*}*

Содержимое файла MapCollision*.cpp:*

*#include"MapCollision.h"*

*bool map\_collision(bool i\_collect\_pellets, bool i\_use\_door, short i\_x, short i\_y, std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH>& i\_map)*

*{*

*bool output = 0;*

*//Getting the exact position.*

*float cell\_x = i\_x / static\_cast<float>(CELL\_SIZE);*

*float cell\_y = i\_y / static\_cast<float>(CELL\_SIZE);*

*//A ghost/Pacman can intersect 4 cells at most.*

*for (unsigned char a = 0; a < 4; a++)*

*{*

*short x = 0;*

*short y = 0;*

*switch (a)*

*{*

*case 0: //Top left cell*

*{*

*x = static\_cast<short>(floor(cell\_x)); //The floor() function in C++ returns the largest possible integer value which is less than or equal to the given argument.*

*//It is defined in the cmath header file.*

*y = static\_cast<short>(floor(cell\_y));*

*break;*

*}*

*case 1: //Top right cell*

*{*

*x = static\_cast<short>(ceil(cell\_x));//The ceil() function in C++ returns the smallest possible integer value which is greater than or equal to the given argument. It is defined in the cmath header file.*

*y = static\_cast<short>(floor(cell\_y));*

*break;*

*}*

*case 2: //Bottom left cell*

*{*

*x = static\_cast<short>(floor(cell\_x));*

*y = static\_cast<short>(ceil(cell\_y));*

*break;*

*}*

*case 3: //Bottom right cell*

*{*

*x = static\_cast<short>(ceil(cell\_x));*

*y = static\_cast<short>(ceil(cell\_y));*

*}*

*}*

*//Making sure that the position is inside the map.*

*if (0 <= x && 0 <= y && MAP\_HEIGHT > y && MAP\_WIDTH > x)// надо, так как если за границу карты заходить, то выход за массив будет*

*{*

*if (0 == i\_collect\_pellets) //Here we only care about the walls.*

*{*

*if (Cell::Wall == i\_map[x][y])*

*{*

*output = 1;*

*}*

*else if (0 == i\_use\_door && Cell::Door == i\_map[x][y])*

*{*

*output = 1;*

*}*

*}*

*else //Here we only care about the collectables.*

*{*

*if (Cell::Energizer == i\_map[x][y])*

*{*

*output = 1;*

*i\_map[x][y] = Cell::Empty;*

*}*

*else if (Cell::Pellet == i\_map[x][y])*

*{*

*i\_map[x][y] = Cell::Empty;*

*}*

*}*

*}*

*}*

*return output;*

*}*

Содержимое файла MenuRecords*.cpp:*

*#include"MenuRecords.h"*

*//std::string EMPTY\_RECORD = "- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -";*

*MenuRecords::MenuRecords(std::string str\_1, std::string str\_2, std::string str\_3, std::string str\_4, std::string str\_5)*

*{*

*if (!font.loadFromFile("Resources/Font/ALGER.TTF"))*

*std::cout << "No font is here" << std::endl;*

*textRecords[0].setFont(font);*

*textRecords[0].setFillColor(sf::Color::Blue);*

*textRecords[0].setString(str\_1);*

*textRecords[0].setCharacterSize(30);*

*textRecords[0].setPosition(10, 100);*

*textRecords[1].setFont(font);*

*textRecords[1].setFillColor(sf::Color::Black);*

*textRecords[1].setString(str\_2);*

*textRecords[1].setCharacterSize(30);*

*textRecords[1].setPosition(10, 200);*

*textRecords[2].setFont(font);*

*textRecords[2].setFillColor(sf::Color::Black);*

*textRecords[2].setString(str\_2);*

*textRecords[2].setCharacterSize(30);*

*textRecords[2].setPosition(10, 300);*

*textRecords[3].setFont(font);*

*textRecords[3].setFillColor(sf::Color::Black);*

*textRecords[3].setString(str\_2);*

*textRecords[3].setCharacterSize(30);*

*textRecords[3].setPosition(10, 400);*

*textRecords[4].setFont(font);*

*textRecords[4].setFillColor(sf::Color::Black);*

*textRecords[4].setString(str\_2);*

*textRecords[4].setCharacterSize(30);*

*textRecords[4].setPosition(10, 500);*

*MenuRecordsSelected = 0;*

*}*

*void MenuRecords::analyze(std::string other)*

*{*

*//static int position;*

*textRecords[position].setString(other);*

*infoRecords[position] = other;*

*position++;*

*}*

*void MenuRecords::deleteRecord()*

*{*

*//textRecords[MenuRecordsPressed()].setString("-------------------------------------");*

*if (infoRecords[MenuRecordsPressed()].compare(EMPTY\_RECORD) == 0)*

*{*

*return;*

*}*

*else*

*{*

*for (int i = MenuRecordsPressed() + 1; ; i++)*

*{*

*if (i < Max\_menu\_records)*

*{*

*infoRecords[i - 1] = infoRecords[i];*

*textRecords[i - 1].setString(infoRecords[i - 1]);*

*}*

*else*

*{*

*infoRecords[i - 1] = EMPTY\_RECORD;*

*textRecords[i - 1].setString(infoRecords[i - 1]);*

*break;*

*}*

*}*

*}*

*}*

*std::string\* MenuRecords::getRecords()*

*{*

*return infoRecords;*

*}*

*MenuRecords::~MenuRecords()*

*{*

*}*

*void MenuRecords::draw(sf::RenderWindow& window\_records)*

*{*

*for (int i = 0; i < Max\_menu\_records; ++i)*

*window\_records.draw(textRecords[i]);*

*}*

*void MenuRecords::MoveUp()*

*{*

*MenuRecordsSelected--;*

*textRecords[MenuRecordsSelected + 1].setFillColor(sf::Color::Black);*

*if (MenuRecordsSelected == -1)*

*MenuRecordsSelected = Max\_menu\_records - 1;*

*textRecords[MenuRecordsSelected].setFillColor(sf::Color::Blue);*

*}*

*void MenuRecords::MoveDown()*

*{*

*MenuRecordsSelected++;*

*textRecords[MenuRecordsSelected - 1].setFillColor(sf::Color::Black);*

*if (MenuRecordsSelected == Max\_menu\_records)*

*MenuRecordsSelected = 0;*

*textRecords[MenuRecordsSelected].setFillColor(sf::Color::Blue);*

*}*

*int MenuRecords::MenuRecordsPressed()*

*{*

*return MenuRecordsSelected;*

*}*

*int MenuRecords::position;*

Содержимое файла Pacman*.cpp:*

*#include"Pacman.h"*

*Pacman::Pacman()*

*{*

*animation\_over = 0;*

*dead = 0;*

*energizer\_timer = 0;*

*direction = 1;*

*position.x=0, position.y=0;*

*}*

*bool Pacman::get\_dead()*

*{*

*return dead;*

*}*

*void Pacman::set\_dead(bool i\_dead)*

*{*

*dead = i\_dead;*

*}*

*unsigned short Pacman::get\_energizer\_timer()*

*{*

*return energizer\_timer;*

*}*

*void Pacman::set\_energizer\_timer(unsigned short energ\_tim)*

*{*

*energizer\_timer = energ\_tim;*

*}*

*bool Pacman::get\_animation\_over()*

*{*

*return animation\_over;*

*}*

*void Pacman::set\_animation\_over(bool an\_tim)*

*{*

*animation\_over = an\_tim;*

*}*

*void Pacman::reset()*

*{*

*dead = 0;*

*animation\_over = 0;*

*energizer\_timer = 0;*

*direction = 1;*

*animation\_timer = 0;*

*}*

*void Pacman::draw(bool i\_victory, sf::RenderWindow& i\_window)*

*{*

*unsigned char frame = static\_cast<unsigned char>(floor(animation\_timer / static\_cast<float>(PACMAN\_ANIMATION\_SPEED)));// 0->24 /4*

*sf::Sprite sprite;*

*sf::Texture texture;*

*sprite.setPosition(position.x, position.y);*

*if (1 == dead || 1 == i\_victory)*

*{*

*if (animation\_timer < PACMAN\_DEATH\_FRAMES \* PACMAN\_ANIMATION\_SPEED) // 12\*4*

*{*

*animation\_timer++;*

*texture.loadFromFile("Resources/Images/PacmanDeath" + std::to\_string(CELL\_SIZE) + ".png");*

*sprite.setTexture(texture);*

*sprite.setTextureRect(sf::IntRect(CELL\_SIZE \* frame, 0, CELL\_SIZE, CELL\_SIZE));*

*i\_window.draw(sprite);*

*}*

*else*

*{*

*//You can only die once.*

*animation\_over = 1;*

*}*

*}*

*else*

*{*

*texture.loadFromFile("Resources/Images/Pacman" + std::to\_string(CELL\_SIZE) + ".png");*

*sprite.setTexture(texture);*

*sprite.setTextureRect(sf::IntRect(CELL\_SIZE \* frame, CELL\_SIZE \* direction, CELL\_SIZE, CELL\_SIZE));*

*i\_window.draw(sprite);*

*animation\_timer = (1 + animation\_timer) % (PACMAN\_ANIMATION\_FRAMES \* PACMAN\_ANIMATION\_SPEED); // 6\*4*

*}*

*}*

*void Pacman::update(unsigned char i\_level, std::array<std::array<Cell, MAP\_HEIGHT>, MAP\_WIDTH>& i\_map)*

*{*

*//анализ стен*

*//проверка кнопок*

*//изменение координат пакмана в зависимости от стен и направления*

*//проход по тунелю*

*//обработка таблеток и пилюли*

*//обработка таймера энергии*

*std::array<bool, 4> walls{};*

*walls[0] = map\_collision(CollectPellets::No, I\_Use\_Door::No\_, PACMAN\_SPEED + position.x, position.y, i\_map);*

*walls[1] = map\_collision(CollectPellets::No, I\_Use\_Door::No\_, position.x, position.y - PACMAN\_SPEED, i\_map);*

*walls[2] = map\_collision(CollectPellets::No, I\_Use\_Door::No\_, position.x - PACMAN\_SPEED, position.y, i\_map);*

*walls[3] = map\_collision(CollectPellets::No, I\_Use\_Door::No\_, position.x, PACMAN\_SPEED + position.y, i\_map);*

*if (1 == sf::Keyboard::isKeyPressed(sf::Keyboard::Right))*

*{*

*if (0 == walls[0]) //You can't turn in this direction if there's a wall there.*

*{*

*direction = 0;// pacman moves right for default (it implements in the pacman class)*

*}*

*}*

*if (1 == sf::Keyboard::isKeyPressed(sf::Keyboard::Up))*

*{*

*if (0 == walls[1])*

*{*

*direction = 1;*

*}*

*}*

*if (1 == sf::Keyboard::isKeyPressed(sf::Keyboard::Left))*

*{*

*if (0 == walls[2])*

*{*

*direction = 2;*

*}*

*}*

*if (1 == sf::Keyboard::isKeyPressed(sf::Keyboard::Down))*

*{*

*if (0 == walls[3])*

*{*

*direction = 3;*

*}*

*}*

*if (0 == walls[direction])//if not the button, then it's necessary check*

*{*

*switch (direction)*

*{*

*case 0:*

*{*

*position.x += PACMAN\_SPEED;*

*break;*

*}*

*case 1:*

*{*

*position.y -= PACMAN\_SPEED;*

*break;*

*}*

*case 2:*

*{*

*position.x -= PACMAN\_SPEED;*

*break;*

*}*

*case 3:*

*{*

*position.y += PACMAN\_SPEED;*

*break;*

*}*

*}*

*//}*

*//////////////////////////////////////////////////////////////////boarder/////////*

*if (-CELL\_SIZE >= position.x) // -1|0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 | 21*

*// | |*

*{*

*position.x = CELL\_SIZE \* MAP\_WIDTH - PACMAN\_SPEED;*

*}*

*else if (CELL\_SIZE \* MAP\_WIDTH <= position.x)*

*{*

*position.x = PACMAN\_SPEED - CELL\_SIZE;*

*}*

*/////////////////////////////////////////////////////////////////////////////////////*

*if (1 == map\_collision(1, 0, position.x, position.y, i\_map)) //When Pacman eats an energizer...*

*{*

*//He becomes energized!*

*energizer\_timer = static\_cast<unsigned short>(ENERGIZER\_DURATION / pow(2, i\_level));*

*}*

*else*

*{*

*energizer\_timer = std::max(0, energizer\_timer - 1);*

*//energizer\_timer = 0;*

*/\*template<class T>*

*const T& max(const T & a, const T & b)*

*{*

*return (a < b) ? b : a;*

*}\*/*

*}*

*}*

*}*