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)