Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра прикладной математики и кибернетики

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование»

Тема: «Морской бой»

Выполнил: студент 2 курса группы ИП-811 Мироненко К. А

Проверил: доцент кафедры ПМиК Ситняковская Е. И.

Оглавление

[1. Постановка задачи 3](#_Toc30598306)

[1.1 Общие положения 3](#_Toc30598307)

[1.2 Задание в соответствии с вариантом 3](#_Toc30598308)

[2. Описание иерархии объектов и методов объектов 4](#_Toc30598309)

[3. Описание алгоритма основной программы 10](#_Toc30598310)

[4. Примеры работы программы 11](#_Toc30598311)

[5. Выводы 14](#_Toc30598312)

[6. Список используемых источников 15](#_Toc30598313)

[*Приложение* Листинг 16](#_Toc30598314)

# Постановка задачи

## Общие положения

Программа должна быть написана с использованием **объектно-ориентированных технологий**. **Описания** объектов и методов **необходимо оформить в отдельном модуле**. Допускаются и поощряются различные творческие модификации, по существу, расширяющие и развивающие реализацию задания.

**Необходимый минимум содержания работы**:

* Инкапсуляция (все поля данных не доступны из внешних функций)
* Наследование (минимум 3 класса, один из которых - абстрактный)
* Полиморфизм
* Конструкторы, Перегрузка конструкторов
* Списки инициализации

Желательно: минимум + 2 технологии ООП (статические элементы, дружественные функции, классы, виртуальные функции, шаблоны, множественное наследование, массивы указателей на объекты, конструкторы копирования, параметры по умолчанию, использование объектов в качестве аргументов или возвращаемых значений)

## Задание в соответствии с вариантом

Реализовать игру «морской бой» (графический режим)

# Описание иерархии объектов и методов объектов

***Класс GlobalVars*** *(Singleton pattern)* – класс, хранящий глобальные переменные и гарантирующий, что в однопоточном приложении будет единственный экземпляр класса, и предоставляющий глобальную точку доступа к этому экземпляру.

**Public**

**Поля:**

* bool debug – логическая переменная, отвечающая за вкл/откл debug-режима;
* uint8\_t gameDifficulty – игровая сложность (возможны значения 0,1,2);

**Методы:**

* static GlobalVars& Instance() – метод, возвращающий единственный экземпляр класса;

***Класс Assets*** *(Singleton pattern)* – класс, хранящий ресурсы и гарантирующий, что в однопоточном приложении будет единственный экземпляр класса, и предоставляющий глобальную точку доступа к этому экземпляру.

**Public**

**Поля:**

* sf::Texture boardTexture – переменная, хранящая текстуры игрового поля;
* sf::Texture shipDeskTexture – переменная, хранящая текстуры устанавливаемого корабля;
* sf::Texture menuBackGround – переменная, хранящая текстуры устанавливаемого заднего фона окна с меню;
* sf::Font fontIron – переменная, хранящая шрифт Iron;
* sf::Font fontMagnum – переменная, хранящая шрифт Magnum;

**Методы:**

* static GlobalVars& Instance() – метод, возвращающий экземпляр класса;
* void Assets::Load() – метод, инициализирующий поля (загружающий ресурсы в память);

***Класс FieldCell*** – класс, отвечающий за ячейку на игровом поле.

**Public**

**Методы:**

* cellEnum getState() – метод, возвращающий информацию о текущем состоянии ячейки (возможно: empty, miss, hit, kill, ship);
* void setState(cellEnum state) – метод, устанавливающий состояние ячейки (возможно: empty, miss, hit, kill, ship);
* sf::Sprite& getSprite() – метод, возвращающий текущий спрайт ячейки;
* void setSprite(cellEnum state) – метод, задающий спрайт ячейке;

**Private**

**Поля:**

* cellEnum cellState – состояние ячейки (возможно: empty, miss, hit, kill, ship);
* sf::Sprite sprite – текущий спрайт ячейки;

***Класс Ship*** – класс, отвечающий за корабль на игровом поле.

**Public**

**Методы:**

* void setSize(uint8\_t size) – метод, задающий размер корабля;
* uint8\_t getSize() – метод, возвращающий кол-во палуб корабля;
* void setOrientation(shipOrientationEnum orientation) – метод, устанавливающий ориентацию корабля;
* shipOrientationEnum getOrientation() – метод, возвращающий ориентацию корабля;
* void changeOrientation() – метод, меняющий ориентацию корабля на противоположную;
* void setCoordinates(uint8\_t x, uint8\_t y) – метод, устанавливающий координаты корабля;
* coordinates \*getCoordinates() – метод, возвращающий координаты корабля;
* bool hasCoordinates(int8\_t x, int8\_t y) – метод, проверяющий наличие корабля в соответствующих координатах;
* void setSinkingPart(uint8\_t x, uint8\_t y) – метод, “топящий” соответвующую часть корабля;
* int8\_t findUnSunkenPart() – метод, возвращающий индекс не затонувшей части корабля
* bool isSunken() – метод, проверяющий: “Затонул ли корабль?”

**Private**

**Поля:**

* uint8\_t shipSize – кол-во палуб (1-4);
* shipOrientationEnum shipOrientation – ориентация корабля (h(0), v(1));
* bool \*sunkenPart – bool массив (размером shipSize) затонувших палуб;
* coordinates \*position – массив (размером shipSize), зранящий координаты палуб;

**Методы:**

* void generateArrays() – метод, генерирующий(перегенирирующий) массивы sunkenPart и position ;

***Класс Board*** – класс, отвечающий за игровое поле.

**Public**

**Методы:**

* uint8\_t getSize() – метод, возвращающий размер поля;
* FieldCell\*\* getGrid() – метод, возвращающий игровую сетку (квадратную матрицу ячеек);;
* Ship\*getShips() – метод, возвращающий корабли игрового поля (массив кораблей);

**Private**

**Поля:**

* uint8\_t size – размер игрового поля;
* FieldCell\*\* grid – сетка игрового поля (квадратная матрица ячеек);
* Ship\* ships– корабли игрового поля (массив кораблей);

***Класс WindowInterface (abstract)*** – абстрактный класс отрисовываемого окна.

**Public**

**Методы:**

* virtual int draw(sf::RenderWindow& window) = 0; – виртуальный метод, отвечающий за отрисовку элементов окна;

***Класс Application*** – класс, отвечающий за запуск приложения и смену окон.

**Public**

**Методы:**

* void run() – метод, запускающий приложение и вызывающий текущее окно;

***Класс MainWindow : public WindowInterface*** – класс, отвечающий за окно “Главное меню”.

**Public**

**Методы:**

* int draw(sf::RenderWindow& window) override – метод, отвечающий за отрисовку элементов, а также логику окна;

**Private**

**Поля:**

* sf::Sprite background – фон окна;
* sf::Text menuText[3] – массив, содержащий элементы меню;
* sf::Text difficultyText[3] – массив со сложностями игры;

**Методы:**

* void hoverButtonsMenu(sf::Vector2i mousePosition) – метод, отвечающий за графическое выделение элемента меню при наведении на него;
* void hoverButtonsDifficulty(sf::Vector2i mousePosition) – метод, отвечающий за графическое выделение одной из сложностей при наведении на нее;

***Класс InfoWindow : public WindowInterface*** – класс, отвечающий за окно “Инфо”.

**Public**

**Методы:**

* int draw(sf::RenderWindow& window) override – метод, отвечающий за отрисовку элементов, а также логику окна;

**Private**

**Поля:**

* sf::Color backgroundColor – фон окна;
* sf::Text information[5] – массив строк, содержащих информацию;

***Класс GameWindow : public WindowInterface*** – класс, отвечающий за окно c игрой;

**Public**

**Методы:**

* int draw(sf::RenderWindow& window) override – метод, отвечающий за отрисовку элементов, а также логику окна;

**Private**

**Поля:**

* sf::Color backgroundColor – фон окна;
* sf::Clock blinkTimer – таймер, используемый при мерцании текста;
* sf::Text moveText – строка, хранящая информацию о текущем ходе(“Ход игрока” или “Ход компьютера”);
* sf::Text playersText[2] – массив строк хранящий подписи игровых полей;
* Board \*playerBoard, \*computerBoard – игровые поля игрока и компьютера, соответственно;
* gameEnum gameState – состояние игры на данный момент;

**Методы:**

* void currentMoveText() – метод, меняющий строку(moveText) и ее свойства в зависимости от текущего состояния игры(gameState);
* void drawBoards(sf::RenderWindow &window) – метод, отрисовывающий игровые поля;
* void MouseDrawShip(sf::RenderWindow &window) – метод, отвечающий за отрисовку корабля на курсоре при расстановке кораблей пользователем;
* void setShipOnField(sf::Vector2i mousePosition) – метод, проверяющий корректность установки, а также устанавливающий корабль, если положение корректно (при расстановке кораблей пользователем);
* void placementShipsByComputer() – метод, генерирующий расстановку кораблей компьютером;
* int8\_t playerMakeShot(sf::Vector2i mousePosition) – метод, отвечающий за ход игрока;
* int8\_t computerMakeShot(sf::Clock &thinkingTimer) – метод, отвечающий за ход компьютера;
* void checkSunkenShips() – метод, проверяющий затонувшие корабли и в зависимости от этого меняющий состояние ячейки поля;
* bool isShipsCrossing(Ship &first, Ship &second) –метод проверяющие два корабля на соприкосновение (используется при расстановке кораблей);
* bool isEndgame() – метод, проверяющий завершение игры;

# Описание алгоритма основной программы

Работа программы начинается с главного меню. В главном меню пользователь может начать игру – кнопка “Начать игру”, просмотреть блок с информацией – кнопка “Инфо”, либо выйти из игры – кнопка “Выход”. Также пользователю доступен выбор сложности игры (блок в нижнем правом углу).

Перемещения по пунктам меню осуществляется с помощью мышки, нажатием лев. кн. мыши осуществляется соответствующий выбор пункта меню.

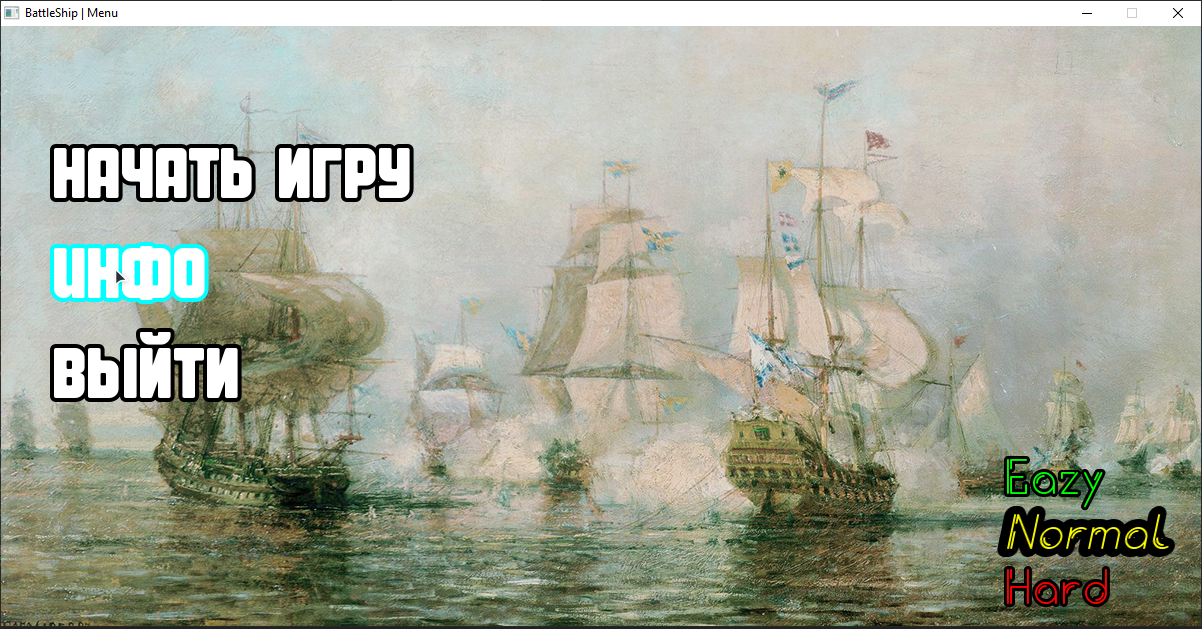
При выборе “Инфо” окно, содержащее меню, сменяется окном с информацией (InfoWindow). Выход из окна осуществляется нажатием клавиши “Esc”.

В случае когда игрок начинает игру окно сменяется окном с игрой(GameWindow). Далее, выполняется соответствующая состоянию (gameState) логика игры “Морской бой”. Выход из окна осуществляется нажатием клавиши “Esc”.

При выборе пользователем пункта главного меню “Выход” прекращается отрисовка графического окна и происходит последующее закрытие программы.

# Примеры работы программы

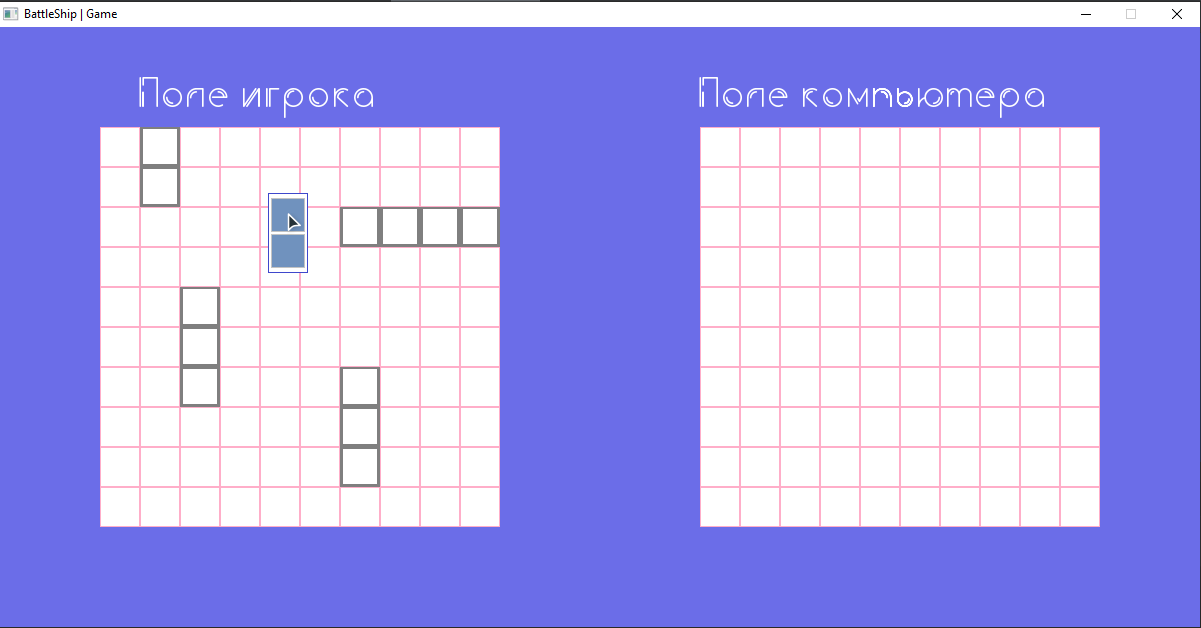
**Окно с меню**

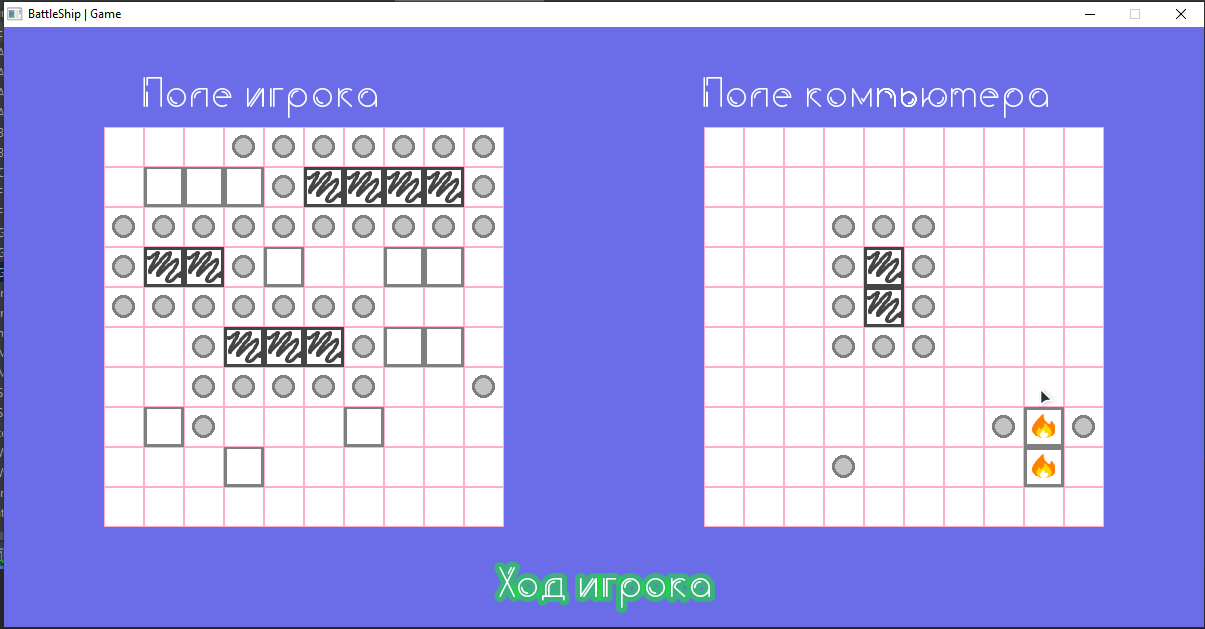


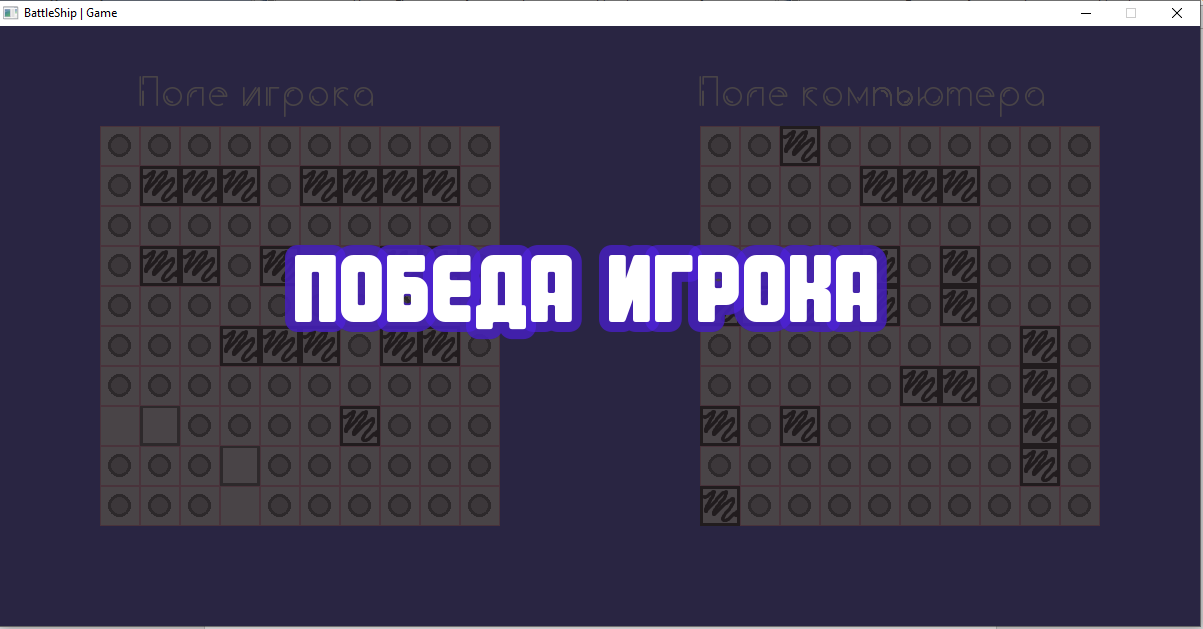
**Окно с информацией**



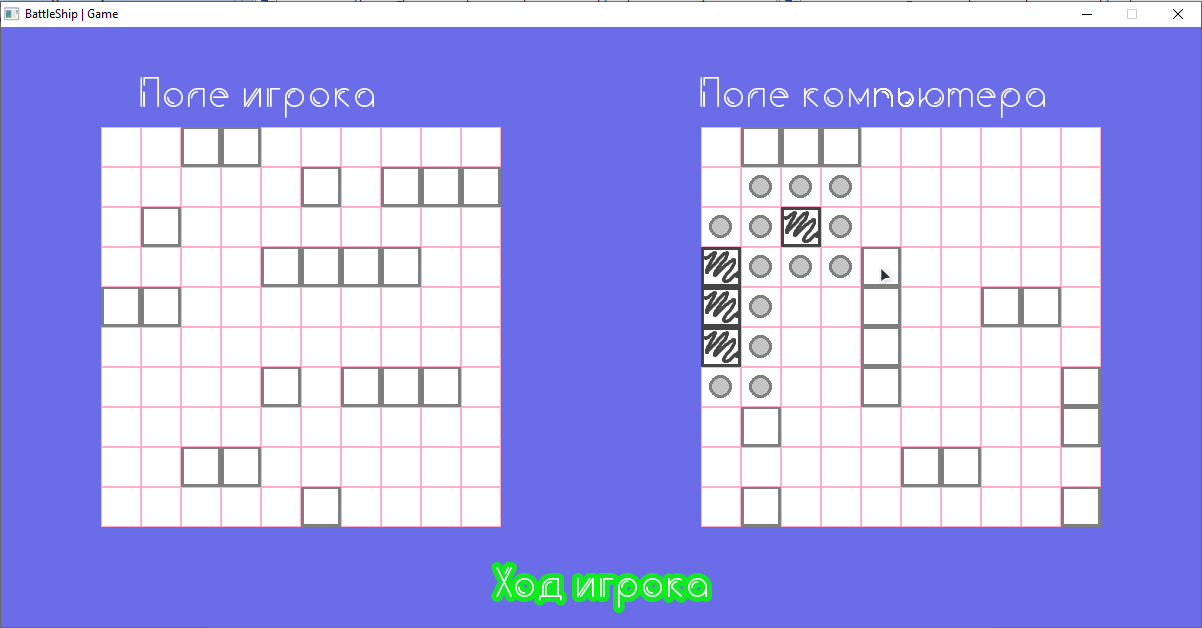
**Игровое окно**







**Работа debug-режима**



# Выводы

Благодаря этому проекту я на практике применил такую методологию программирования, как объектно-ориентированное программирование, а также закрепил знания, полученные в ходе данного курса.

# Список используемых источников

1. Порождающие шаблоны проектирования [Электронный ресурс] // Википедия : Свободная энциклопедия. URL : https://en.wikipedia.org/wiki/Creational\_pattern (дата обращения: 15.01.20)
2. Singleton pattern (шаблон проектирования) [Электронный ресурс] // Википедия : Свободная энциклопедия. URL : https://en.wikipedia.org/wiki/Singleton\_pattern (дата обращения: 15.01.20)
3. Documentation of SFML 2.5.1 [Электронный ресурс] // SFML : Simple and Fast Multimedia Library. URL : https://www.sfml-dev.org/documentation/2.5.1/ (дата обращения: 15.01.20)

# *Приложение* Листинг

**main.сpp**

#include "Application.hpp"

#include "Assets.hpp"

#include "GlobalVars.hpp"

#include <cstring>

#include <ctime>

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

{

srand(time(0)) ;

if (argc > 1)

if (strcmp(argv[1], "-d") == 0) {

GlobalVars::Instance().debug = true ;

}

}

Assets::Instance().Load() ; // Загружаем ресурсы

Application\* app = new Application ;

app->run() ;

return 0 ;

}

**GlobalVars.hpp**

#ifndef GLOBALVARS\_HPP

#define GLOBALVARS\_HPP

#include <cstdint>

class GlobalVars{

public:

bool debug = false ;

uint8\_t gameDifficulty = 1 ;

public:

static GlobalVars& Instance()

{

static GlobalVars s ;

return s ;

}

private:

GlobalVars() {} ;

~GlobalVars() {} ;

GlobalVars(GlobalVars const&) = delete ;

GlobalVars& operator= (GlobalVars const&) = delete ;

};

#endif // GLOBALVARS\_HPP

**Assets.hpp**

#ifndef ASSETS\_HPP

#define ASSETS\_HPP

#include <SFML/Graphics.hpp>

class Assets

{

public:

sf::Texture boardTexture;

sf::Texture shipDeskTexture;

sf::Texture menuBackGround;

sf::Font fontIron;

sf::Font fontMagnum;

public:

static Assets& Instance()

{

static Assets s;

return s;

}

void Load();

private:

Assets() {};

~Assets() {};

Assets(Assets const&) = delete;

Assets& operator= (Assets const&) = delete;

};

#endif // ASSETS\_HPP

**Assets.сpp**

#include "Assets.hpp"

void Assets::Load()

{

if (!boardTexture.loadFromFile("texture/boardTextures.png")) throw;

if (!shipDeskTexture.loadFromFile("texture/shipDeckTextures.png")) throw;

if (!menuBackGround.loadFromFile("texture/menuBackground.png")) throw;

if (!fontIron.loadFromFile("texture/Iron.ttf")) throw;

if (!fontMagnum.loadFromFile("texture/Magnum.ttf")) throw;

}

**FieldCell.hpp**

#ifndef FIELDCELL\_HPP

#define FIELDCELL\_HPP

#include <SFML/Graphics.hpp>

#include "Assets.hpp"

enum cellEnum:uint8\_t {empty, miss, hit, kill, ship} ;

class FieldCell { // Ячейка поля

/\* Поля \*/

private:

cellEnum cellState ; // Состояние ячейки

sf::Sprite sprite ;

/\* Конструкторы и деструкторы \*/

public:

FieldCell() ;

explicit FieldCell(cellEnum state) ;

public:

/\* Методы \*/

cellEnum getState() ;

void setState(cellEnum state) ;

sf::Sprite& getSprite() ;

void setSprite(cellEnum state) ;

};

#endif // FIELDCELL\_HPP

**FieldCell.cpp**

#include "FieldCell.hpp"

FieldCell::FieldCell() : cellState(cellEnum::empty) {

sprite.setTexture(Assets::Instance().boardTexture);

}

FieldCell::FieldCell(cellEnum state): cellState(state)

{

sprite.setTexture(Assets::Instance().boardTexture);

}

cellEnum FieldCell::getState()

{

return this->cellState ;

}

void FieldCell::setState(cellEnum state)

{

this->cellState = state ;

}

sf::Sprite& FieldCell::getSprite()

{

return this->sprite ;

}

void FieldCell::setSprite(cellEnum state)

{

switch (state){

case empty:

sprite.setTextureRect(sf::IntRect(0, 0, 40, 40));

break;

case ship:

sprite.setTextureRect(sf::IntRect(160, 0, 40, 40));

break;

case miss:

sprite.setTextureRect(sf::IntRect(40, 0, 40, 40));

break;

case hit:

sprite.setTextureRect(sf::IntRect(80, 0, 40, 40));

break;

case kill:

sprite.setTextureRect(sf::IntRect(120, 0, 40, 40));

break;

}

}

**Ship.hpp**

#ifndef SHIP\_HPP

#define SHIP\_HPP

#include <SFML/Graphics.hpp>

#include <iostream>

#include "GlobalVars.hpp"

enum shipOrientationEnum:bool {h, v} ;

struct coordinates

{

int8\_t x ;

int8\_t y ;

} ;

class Ship {

/\* Поля \*/

private:

uint8\_t shipSize ; // кол-во палуб

shipOrientationEnum shipOrientation ; // ориентация корабля

bool \*sunkenPart; // bool массив затонувших палуб

coordinates \*position ; // координаты палуб

/\* Конструкторы и деструкторы \*/

public:

Ship() ;

~Ship() ;

/\* Методы \*/

private:

void generateArrays() ;

public:

void setSize(uint8\_t size) ;

uint8\_t getSize() ;

void setOrientation(shipOrientationEnum orientation) ;

shipOrientationEnum getOrientation() ;

void changeOrientation() ;

void setCoordinates(uint8\_t x, uint8\_t y) ;

coordinates \*getCoordinates() ;

bool hasCoordinates(int8\_t x, int8\_t y) ;

void setSinkingPart(uint8\_t x, uint8\_t y) ;

int8\_t findUnSunkenPart() ;

bool isSunken() ;

};

#endif // SHIP\_HPP

**Ship.cpp**

#include "Ship.hpp"

Ship::Ship()

{

shipSize = 1 ;

shipOrientation = h ;

sunkenPart = nullptr ;

position = nullptr ;

generateArrays() ;

}

Ship::~Ship()

{

delete sunkenPart;

sunkenPart = nullptr;

delete position;

position = nullptr;

}

void Ship::generateArrays()

{

delete sunkenPart;

delete position ;

sunkenPart = new bool[shipSize]{false} ;

position = new coordinates[shipSize] ;

for (uint8\_t i = 0 ; i < shipSize ; i++)

position[i].x = position[i].y = 0 ;

}

void Ship::setSize(uint8\_t size)

{

if ((size >= 1) && (size <= 4))

shipSize = size ;

else

{

std::cout << "Error[Incorrect ship size]\n" ;

throw ;

}

generateArrays() ;

}

uint8\_t Ship::getSize()

{

return shipSize ;

}

void Ship::setOrientation(shipOrientationEnum orientation)

{

shipOrientation = orientation ;

}

shipOrientationEnum Ship::getOrientation()

{

return shipOrientation ;

}

void Ship::changeOrientation()

{

(shipOrientation == h) ? (shipOrientation = v) : (shipOrientation = h);

}

void Ship::setCoordinates(uint8\_t x, uint8\_t y)

{

if (GlobalVars::Instance().debug)

std::cout << (int)shipSize << "-ship : " ;

for (uint8\_t i = 0 ; i < shipSize ; i++)

{

if (shipOrientation == h)

{

position[i].x = x++ ;

position[i].y = y ;

}

else

{

position[i].x = x ;

position[i].y = y++ ;

}

if (GlobalVars::Instance().debug)

std::cout << (int)position[i].x + 1 << "," << (int)position[i].y + 1 << " " ;

}

if (GlobalVars::Instance().debug)

std::cout << "\n" ;

}

coordinates\* Ship::getCoordinates()

{

return position ;

}

bool Ship::hasCoordinates(int8\_t x, int8\_t y)

{

for (uint8\_t i = 0 ; i < shipSize ; i++)

if (position[i].x == x and position[i].y == y)

return true ;

return false ;

}

void Ship::setSinkingPart(uint8\_t x, uint8\_t y)

{

for (uint8\_t i = 0 ; i < shipSize ; i++)

if (position[i].x == x and position[i].y == y)

sunkenPart[i] = true ;

}

int8\_t Ship::findUnSunkenPart()

{

for (uint8\_t i = 0 ; i < shipSize ; i++)

if (!sunkenPart[i])

return i ;

return -1 ;

}

bool Ship::isSunken()

{

for (uint8\_t i = 0 ; i < shipSize ; i++)

if (!sunkenPart[i])

return false ;

return true ;

}

**Board.hpp**

#ifndef BOARD\_HPP

#define BOARD\_HPP

#include "FieldCell.hpp"

#include "Ship.hpp"

class Board {

/\* Поля \*/

private:

uint8\_t size ;

FieldCell\*\* grid ;

Ship\* ships ;

/\* Конструкторы и деструкторы \*/

public:

Board() ;

~Board() ;

/\* Методы \*/

public:

uint8\_t getSize() ;

FieldCell\*\* getGrid() ;

Ship\*getShips() ;

} ;

#endif // BOARD\_HPP

**Board.cpp**

#include "Board.hpp"

Board::Board()

{

size = 10 ;

grid = new FieldCell\*[size] ;

for (uint8\_t i = 0 ; i < size ; ++i)

grid[i] = new FieldCell[size];

ships = new Ship[size];

}

Board::~Board()

{

for (uint8\_t i = 0 ; i < size ; ++i)

delete grid[i] ;

delete[] grid ;

grid = nullptr ;

delete[] ships ;

ships = nullptr ;

}

uint8\_t Board::getSize()

{

return size ;

}

FieldCell\*\* Board::getGrid()

{

return grid ;

}

Ship\* Board::getShips()

{

return ships ;

}

**Application.hpp**

#ifndef APPLICATION\_HPP

#define APPLICATION\_HPP

#include <SFML/Graphics.hpp>

#include "WindowInterface.hpp"

#include "MainWindow.hpp"

#include "GameWindow.hpp"

#include "InfoWindow.hpp"

class Application {

/\* Методы \*/

public:

void run() ;

} ;

#endif // APPLICATION\_HPP

**Application.cpp**

#include "Application.hpp"

enum stateEnum:int8\_t {end = -1, winMain, winGame, winInfo} ;

void Application::run()

{

sf::RenderWindow window (sf::VideoMode(1200, 600), "BattleShip", sf::Style::Close) ;

window.setFramerateLimit(60) ;

WindowInterface\* currentWindow = nullptr ;

int8\_t gameState = winMain ;

while (gameState != stateEnum::end)

{

switch (gameState)

{

case winMain:

currentWindow = new MainWindow ;

gameState = currentWindow -> draw(window) ;

delete currentWindow ;

break;

case winGame:

currentWindow = new GameWindow ;

gameState = currentWindow -> draw(window) ;

delete currentWindow ;

break;

case winInfo:

currentWindow = new InfoWindow ;

gameState = currentWindow->draw(window) ;

delete currentWindow ;

break;

default:

gameState = stateEnum::end ;

break;

}

}

if (window.isOpen())

{

window.close() ;

}

}

**WindowInterface.hpp**

#ifndef WINDOWINTERFACE\_HPP

#define WINDOWINTERFACE\_HPP

#include "SFML/Graphics.hpp"

class WindowInterface

{

public:

WindowInterface();

virtual ~WindowInterface();

virtual int draw(sf::RenderWindow& window) = 0;

};

#endif // WINDOWINTERFACE\_HPP

**WindowInterface.cpp**

#include "WindowInterface.hpp"

WindowInterface::WindowInterface(){}

WindowInterface::~WindowInterface(){}

**MainWindow.hpp**

#ifndef MAINWINDOW\_HPP

#define MAINWINDOW\_HPP

#include "WindowInterface.hpp"

#include "Assets.hpp"

#include "GlobalVars.hpp"

class MainWindow : public WindowInterface {

/\* Поля \*/

private:

sf::Sprite background ;

sf::Text menuText[3] ;

sf::Text difficultyText[3] ;

/\* Конструкторы и деструкторы \*/

public:

MainWindow() ;

int draw(sf::RenderWindow& window) override;

/\* Методы \*/

public:

int draw(sf::RenderWindow& window) override;

private:

void hoverButtonsMenu(sf::Vector2i mousePosition) ;

void hoverButtonsDifficulty(sf::Vector2i mousePosition) ;

} ;

#endif // MAINWINDOW\_HPP

**MainWindow.cpp**

#include "MainWindow.hpp"

MainWindow::MainWindow()

{

background.setTexture(Assets::Instance().menuBackGround) ;

background.setPosition(0, 0) ;

// Элементы меню

for (uint8\_t i = 0 ; i < 3 ; i++)

{

menuText[i].setFont(Assets::Instance().fontMagnum) ;

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

menuText[i].setOutlineColor(sf::Color::Black) ;

menuText[i].setOutlineThickness(4.f) ;

menuText[i].setCharacterSize(72) ;

menuText[i].setPosition(50, 100 + 100 \* i) ;

}

menuText[0].setString(L"Начать игру") ;

menuText[1].setString(L"Инфо") ;

menuText[2].setString(L"Выйти") ;

// Сложности

for (uint8\_t i = 0 ; i < 3 ; i++)

{

difficultyText[i].setFont(Assets::Instance().fontIron) ;

difficultyText[i].setOutlineColor(sf::Color::Black) ;

difficultyText[i].setOutlineThickness(3.f) ;

difficultyText[i].setCharacterSize(58) ;

difficultyText[i].setPosition(1000, 410 + 55 \* i) ;

}

difficultyText[0].setString(L"Eazy") ;

difficultyText[0].setFillColor(sf::Color::Green) ;

difficultyText[1].setString(L"Normal") ;

difficultyText[1].setFillColor(sf::Color::Yellow) ;

difficultyText[2].setString(L"Hard") ;

difficultyText[2].setFillColor(sf::Color::Red) ;

}

int MainWindow::draw(sf::RenderWindow& window)

{

window.setTitle("BattleShip | Menu") ;

sf::Event event ;

sf::Vector2i mousePosition ;

while (true)

{

while(window.pollEvent(event))

{

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

or

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

and

(event.key.code == sf::Keyboard::Escape))

)

return -1 ;

if (event.type == sf::Event::MouseButtonReleased

and

event.mouseButton.button == sf::Mouse::Left)

{

if (sf::IntRect(menuText[0].getGlobalBounds()).contains(sf::Mouse::getPosition(window)))

return 1 ;

else if (sf::IntRect(menuText[1].getGlobalBounds()).contains(sf::Mouse::getPosition(window)))

return 2 ;

else if (sf::IntRect(menuText[2].getGlobalBounds()).contains(sf::Mouse::getPosition(window)))

return -1 ;

for (uint8\_t i = 0 ; i < 3 ; i++)

{

if (sf::IntRect(difficultyText[i].getGlobalBounds()).contains(sf::Mouse::getPosition(window)))

GlobalVars::Instance().gameDifficulty = i ;

}

}

}

mousePosition = sf::Mouse::getPosition(window) ;

hoverButtonsMenu(mousePosition) ;

hoverButtonsDifficulty(mousePosition) ;

window.clear() ;

window.draw(background) ;

for (sf::Text &textMenu : menuText)

window.draw(textMenu) ;

for (sf::Text &textDiff : difficultyText)

window.draw(textDiff) ;

window.display() ;

}

}

void MainWindow::hoverButtonsMenu(sf::Vector2i mousePosition)

{

for (sf::Text &textMenu : menuText)

{

if (sf::IntRect(textMenu.getGlobalBounds()).contains(mousePosition)){

textMenu.setOutlineColor(sf::Color::Cyan) ;

}

else{

textMenu.setOutlineColor(sf::Color::Black) ;

}

}

}

void MainWindow::hoverButtonsDifficulty(sf::Vector2i mousePosition)

{

for (uint8\_t i = 0 ; i < 3 ; i++)

if (GlobalVars::Instance().gameDifficulty == i){

difficultyText[i].setOutlineThickness(8.f) ;

difficultyText[i].setStyle(sf::Text::Italic) ;

}

else{

difficultyText[i].setOutlineColor(sf::Color::Black) ;

difficultyText[i].setOutlineThickness(3.f) ;

difficultyText[i].setStyle(sf::Text::Regular) ;

}

for (sf::Text &textDiff : difficultyText)

{

if (sf::IntRect(textDiff.getGlobalBounds()).contains(mousePosition))

textDiff.setOutlineColor(sf::Color::Magenta) ;

else

textDiff.setOutlineColor(sf::Color::Black) ;

}

}

**InfoWindow.hpp**

#ifndef INFOWINDOW\_HPP

#define INFOWINDOW\_HPP

#include "WindowInterface.hpp"

#include "Assets.hpp"

class InfoWindow : public WindowInterface {

/\* Поля \*/

private:

sf::Color backgroundColor ;

sf::Text information[5] ;

/\* Конструкторы и деструкторы \*/

public:

InfoWindow() ;

/\* Методы \*/

public:

int draw(sf::RenderWindow &window) override ;

} ;

#endif // INFOWINDOW\_HPP

**InfoWindow.cpp**

#include "InfoWindow.hpp"

InfoWindow::InfoWindow()

{

backgroundColor = sf::Color(102, 165, 235);

for (sf::Text &t : information)

{

t.setFont(Assets::Instance().fontMagnum) ;

t.setFillColor(sf::Color::White) ;

t.setOutlineColor(sf::Color::Black) ;

t.setOutlineThickness(2.f) ;

t.setCharacterSize(84) ;

}

information[0].setString(L"курсовой проект по ооп") ;

information[0].setPosition(175, 25) ;

information[1].setString(L"\"морской бой\"") ;

information[1].setPosition(345, 115) ;

information[2].setString(L"группа ип-811") ;

information[2].setPosition(355, 280) ;

information[3].setString(L"мироненко кирилл") ;

information[3].setPosition(275, 350) ;

information[4].setString(L"2019 - 2020 уч. год") ;

information[4].setPosition(260, 500) ;

}

int InfoWindow::draw(sf::RenderWindow &window)

{

window.setTitle("BattleShip | Information") ;

sf::Event event ;

while (true)

{

while(window.pollEvent(event))

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

return -1 ;

else if ((event.type == sf::Event::KeyReleased) and (event.key.code == sf::Keyboard::Escape))

return 0 ;

window.clear(backgroundColor) ;

for (sf::Text &text : information)

window.draw(text) ;

window.display() ;

}

}

**GameWindow.hpp**

#ifndef GAMEWINDOW\_HPP

#define GAMEWINDOW\_HPP

#include <SFML/Graphics.hpp>

#include <iostream>

#include "WindowInterface.hpp"

#include "Board.hpp"

#include "Assets.hpp"

#include "GlobalVars.hpp"

enum gameEnum:uint8\_t { playerPlacesShips,

computerPlacesShips,

playerShot,

computerShot,

playerEndGame,

computerEndGame

} ;

class GameWindow : public WindowInterface {

/\* Поля \*/

private:

sf::Color backgroundColor ;

sf::Clock blinkTimer ;

sf::Text moveText;

sf::Text playersText[2] ;

Board \*playerBoard, \*computerBoard ;

gameEnum gameState ; // состояние игры на данный момент

/\* Конструкторы и деструкторы \*/

public:

GameWindow() ;

~GameWindow() ;

/\* Методы \*/

private:

void currentMoveText() ;

void drawBoards(sf::RenderWindow &window) ;

void MouseDrawShip(sf::RenderWindow &window) ;

void setShipOnField(sf::Vector2i mousePosition) ;

void placementShipsByComputer() ;

int8\_t playerMakeShot(sf::Vector2i mousePosition) ;

int8\_t computerMakeShot(sf::Clock &thinkingTimer) ;

void checkSunkenShips() ; // Проверка затонувших

bool isShipsCrossing(Ship &first, Ship &second) ; // Проверка на соприкосновение

bool isEndgame() ;

public:

int draw(sf::RenderWindow &window) override ;

};

#endif // GAMEWINDOW\_HPP

**GameWindow.cpp**

#include "GameWindow.hpp"

#define BLINK\_PERIOD 1.f

#define NUM\_SHIPS 10

static uint8\_t currentShip = 0 ; // Текущий корабль (Используется при заполнении массива кораблей)

static shipOrientationEnum currentShipOrientation = h ; // Текущий положение корабля (Используется при заполнении массива кораблей)

GameWindow::GameWindow()

{

currentShip = 0 ;

currentShipOrientation = h ;

backgroundColor = sf::Color(107,109,232) ;

gameState = playerPlacesShips ;

moveText.setFont(Assets::Instance().fontIron) ;

moveText.setCharacterSize(50) ;

moveText.setOutlineThickness(5.f) ;

for (sf::Text &text : playersText)

{

text.setFont(Assets::Instance().fontIron) ;

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

text.setOutlineColor(sf::Color::Red) ;

text.setOutlineThickness(0.f) ;

text.setCharacterSize(50) ;

}

playersText[0].setString(L"Поле игрока") ;

playersText[0].setPosition(135, 30) ;

playersText[1].setString(L"Поле компьютера") ;

playersText[1].setPosition(695, 30) ;

playerBoard = new Board() ;

computerBoard = new Board() ;

}

GameWindow::~GameWindow()

{

delete playerBoard ;

playerBoard = nullptr ;

delete computerBoard ;

computerBoard = nullptr ;

}

void GameWindow::currentMoveText()

{

if (blinkTimer.getElapsedTime().asSeconds() >= BLINK\_PERIOD)

blinkTimer.restart() ;

if (gameState == playerShot)

{

moveText.setString(L"Ход игрока");

moveText.setPosition(490, 520) ;

moveText.setOutlineColor(sf::Color(0, 255, 0, (255 \* blinkTimer.getElapsedTime().asSeconds() / BLINK\_PERIOD))) ;

}

if (gameState == computerShot)

{

moveText.setString(L"Ход компьютера");

moveText.setPosition(425, 520) ;

moveText.setOutlineColor(sf::Color(255, 0, 0, (255 \* blinkTimer.getElapsedTime().asSeconds() / BLINK\_PERIOD))) ;

}

if (gameState == playerEndGame)

{

moveText.setString(L"победа игрока");

moveText.setFont(Assets::Instance().fontMagnum);

moveText.setCharacterSize(96);

moveText.setPosition(290, 200) ;

moveText.setOutlineThickness(10.f);

moveText.setOutlineColor(sf::Color(80, 35, 224, (255 \* blinkTimer.getElapsedTime().asSeconds() / BLINK\_PERIOD \* 2))) ;

}

if (gameState == computerEndGame)

{

moveText.setString(L"победа компьютера");

moveText.setFont(Assets::Instance().fontMagnum);

moveText.setCharacterSize(96);

moveText.setPosition(180, 200) ;

moveText.setOutlineThickness(10.f);

moveText.setOutlineColor(sf::Color(239, 0, 150, (255 \* blinkTimer.getElapsedTime().asSeconds() / BLINK\_PERIOD \* 2))) ;

}

}

void GameWindow::drawBoards(sf::RenderWindow &window)

{

for (uint8\_t x = 0 ; x < playerBoard->getSize() ; x++)

for (uint8\_t y = 0 ; y < playerBoard->getSize() ; y++){

playerBoard->getGrid()[x][y].setSprite(playerBoard->getGrid()[x][y].getState()) ;

playerBoard->getGrid()[x][y].getSprite().setPosition(100 + 40 \* x, 100 + 40 \* y) ;

window.draw(playerBoard->getGrid()[x][y].getSprite()) ;

}

for (uint8\_t x = 0 ; x < computerBoard->getSize() ; x++)

for (uint8\_t y = 0 ; y < computerBoard->getSize() ; y++)

{

if (computerBoard->getGrid()[x][y].getState() == ship)

{

if (GlobalVars::Instance().debug)

computerBoard->getGrid()[x][y].setSprite(ship) ;

else

computerBoard->getGrid()[x][y].setSprite(empty) ;

}

else

computerBoard->getGrid()[x][y].setSprite(computerBoard->getGrid()[x][y].getState()) ;

computerBoard->getGrid()[x][y].getSprite().setPosition(700 + 40 \* x, 100 + 40 \* y) ;

window.draw(computerBoard->getGrid()[x][y].getSprite()) ;

}

}

void GameWindow::MouseDrawShip(sf::RenderWindow &window)

{

sf::Sprite shipSprite(Assets::Instance().shipDeskTexture) ;

sf::Vector2i mousePosition = sf::Mouse::getPosition(window) ;

shipSprite.setOrigin(20, 20) ;

if (currentShip == 0)

{

shipSprite.setTextureRect(sf::IntRect(240, 0, 160, 40)) ;

playerBoard->getShips()[currentShip].setSize(4) ;

}

if (1 <= currentShip and currentShip <= 2)

{

shipSprite.setTextureRect(sf::IntRect(120, 0, 120, 40)) ;

playerBoard->getShips()[currentShip].setSize(3) ;

}

if (3 <= currentShip and currentShip <= 5)

{

shipSprite.setTextureRect(sf::IntRect(40, 0, 80, 40)) ;

playerBoard->getShips()[currentShip].setSize(2) ;

}

if (6 <= currentShip and currentShip <= 9)

{

shipSprite.setTextureRect(sf::IntRect(0, 0, 40, 40)) ;

playerBoard->getShips()[currentShip].setSize(1) ;

}

shipSprite.setPosition((sf::Vector2f) mousePosition) ;

shipSprite.setRotation(currentShipOrientation == v ? 90 : 0) ;

window.draw(shipSprite) ;

}

void GameWindow::setShipOnField(sf::Vector2i mousePosition)

{

for (uint8\_t x = 0 ; x < playerBoard->getSize() ; x++)

{

for (uint8\_t y = 0 ; y < playerBoard->getSize() ; y++)

{

if (sf::IntRect(playerBoard->getGrid()[x][y].getSprite().getGlobalBounds()).contains(mousePosition))

{

playerBoard->getShips()[currentShip].setOrientation(currentShipOrientation) ;

playerBoard->getShips()[currentShip].setCoordinates(x, y) ;

if ((currentShipOrientation == h &&

x + playerBoard->getShips()[currentShip].getSize() > playerBoard->getSize())

or

(currentShipOrientation == v &&

y + playerBoard->getShips()[currentShip].getSize() > playerBoard->getSize()))

return ; // Если корабль выходит за границу сетки

for (uint8\_t count = 0 ; count < currentShip ; count++)

if (isShipsCrossing(playerBoard->getShips()[count], playerBoard->getShips()[currentShip]))

return ; // Если корабль (соприкасается с др.кораблем | пересекает др. корабль)

if (currentShipOrientation == h)

{

for (uint8\_t tmp = x; tmp < x + playerBoard->getShips()[currentShip].getSize() ; tmp++)

{

playerBoard->getGrid()[tmp][y].setState(cellEnum::ship) ;

}

}

else

{

for (uint8\_t tmp = y; tmp < y + playerBoard->getShips()[currentShip].getSize() ; tmp++)

{

playerBoard->getGrid()[x][tmp].setState(cellEnum::ship) ;

}

}

currentShip++ ;

if (currentShip == NUM\_SHIPS)

{

gameState = computerPlacesShips ;

currentShip = 0 ;

}

return ;

}

}

}

}

void GameWindow::placementShipsByComputer()

{

for (uint8\_t i = 0 ; i < NUM\_SHIPS ; i++)

{

if (i == 0)

computerBoard->getShips()[i].setSize(4) ;

else if (1 <= i and i <= 2)

computerBoard->getShips()[i].setSize(3) ;

else if (3 <= i and i <= 5)

computerBoard->getShips()[i].setSize(2) ;

else

computerBoard->getShips()[i].setSize(1) ;

uint8\_t x = 0, y = 0 ;

bool generated = false ;

do

{

x = rand() % computerBoard->getSize() ;

y = rand() % computerBoard->getSize() ;

if (rand() % 2)

computerBoard->getShips()[i].changeOrientation() ;

if ((computerBoard->getShips()[i].getOrientation() == h) and (x + computerBoard->getShips()[i].getSize() <= computerBoard->getSize())

or

(computerBoard->getShips()[i].getOrientation() == v) and (y + computerBoard->getShips()[i].getSize() <= computerBoard->getSize()))

{

computerBoard->getShips()[i].setCoordinates(x, y) ;

bool correct = true ;

for (uint8\_t ship = 0 ; ship < i ; ship++)

if (isShipsCrossing(computerBoard->getShips()[ship], computerBoard->getShips()[i]))

{

correct = false ;

break ;

}

if (correct)

{

if(computerBoard->getShips()[i].getOrientation() == h)

{

for (uint8\_t p\_x = x ; p\_x < x + computerBoard->getShips()[i].getSize() ; p\_x++)

computerBoard->getGrid()[p\_x][y].setState(cellEnum::ship) ;

}

else

{

for (uint8\_t p\_y = y ; p\_y < y + computerBoard->getShips()[i].getSize() ; p\_y++)

computerBoard->getGrid()[x][p\_y].setState(cellEnum::ship) ;

}

generated = true ;

}

}

} while (!generated);

}

}

int8\_t GameWindow::playerMakeShot(sf::Vector2i mousePosition)

{

for (uint8\_t x = 0 ; x < computerBoard->getSize() ; x++)

{

for (uint8\_t y = 0 ; y < computerBoard->getSize() ; y++)

{

if (sf::IntRect(computerBoard->getGrid()[x][y].getSprite().getGlobalBounds()).contains(mousePosition))

{

if (computerBoard->getGrid()[x][y].getState() == empty)

{

computerBoard->getGrid()[x][y].setState(cellEnum::miss) ;

return 0 ;

}

else if (computerBoard->getGrid()[x][y].getState() == ship)

{

for (uint8\_t ship = 0 ; ship < NUM\_SHIPS ; ship++)

if (computerBoard->getShips()[ship].hasCoordinates(x, y))

{

computerBoard->getShips()[ship].setSinkingPart(x, y) ;

break ;

}

computerBoard->getGrid()[x][y].setState(cellEnum::hit) ;

return 1 ;

}

else

return -1 ;

}

}

}

return -1 ;

}

int8\_t GameWindow::computerMakeShot(sf::Clock &thinkingTimer)

{

static Ship \*focusShip = nullptr ;

float waitTime ;

if (GlobalVars::Instance().debug)

waitTime = 0.0f ;

else

waitTime = rand()\*2.0f/RAND\_MAX+1.f;

if (thinkingTimer.getElapsedTime().asSeconds() < waitTime)

return -1 ;

uint8\_t x = 0, y = 0 ;

bool lucky ;

if (GlobalVars::Instance().gameDifficulty == 0)

lucky = true ;

else if (GlobalVars::Instance().gameDifficulty == 1)

lucky = rand() % 2 ;

else if (GlobalVars::Instance().gameDifficulty == 2)

lucky = false ;

if (focusShip != nullptr)

{

int8\_t shootIndex = focusShip->findUnSunkenPart() ;

if (shootIndex >= 0)

{

coordinates \*pos = focusShip->getCoordinates() ;

uint8\_t p\_x = pos[shootIndex].x ;

uint8\_t p\_y = pos[shootIndex].y ;

if (lucky)

{

if (focusShip->getOrientation() == h)

(p\_y < 5) ? p\_y += 1: p\_y -= 1 ;

else

(p\_x < 5) ? p\_x += 1: p\_x -= 1 ;

}

focusShip->setSinkingPart(p\_x, p\_y) ;

if (playerBoard->getGrid()[p\_x][p\_y].getState() == ship)

playerBoard->getGrid()[p\_x][p\_y].setState(cellEnum::hit) ;

else

playerBoard->getGrid()[p\_x][p\_y].setState(cellEnum::miss) ;

thinkingTimer.restart() ;

}

else

focusShip = nullptr ;

if (!lucky)

return 1 ;

else

return 0 ;

}

do

{

x = rand() % playerBoard->getSize() ;

y = rand() % playerBoard->getSize() ;

if (playerBoard->getGrid()[x][y].getState() == empty)

{

playerBoard->getGrid()[x][y].setState(cellEnum::miss) ;

thinkingTimer.restart() ;

return 0 ;

}

else if (playerBoard->getGrid()[x][y].getState() == ship)

{

for (uint8\_t i = 0 ; i < NUM\_SHIPS ; i++)

{

if (playerBoard->getShips()[i].hasCoordinates(x, y))

{

playerBoard->getShips()[i].setSinkingPart(x, y) ;

if (!playerBoard->getShips()[i].isSunken() and (focusShip == nullptr))

focusShip = &playerBoard->getShips()[i] ;

break ;

}

}

playerBoard->getGrid()[x][y].setState(cellEnum::hit) ;

thinkingTimer.restart() ;

return 1 ;

}

} while (true) ;

}

void GameWindow::checkSunkenShips()

{

coordinates \*pos = nullptr ;

coordinates check[8] ;

for (uint8\_t i = 0 ; i < NUM\_SHIPS ; i++)

{

if (playerBoard->getShips()[i].isSunken())

{

pos = playerBoard->getShips()[i].getCoordinates() ;

for (uint8\_t partShip = 0 ; partShip < playerBoard->getShips()[i].getSize() ; partShip++)

{

for (coordinates &f : check)

f = pos[partShip] ;

check[0].x -= 1 ; // л (по час)

check[1].x -= 1 ; check[1].y -= 1 ;

check[2].y -= 1 ;

check[3].x += 1 ; check[3].y -= 1 ;

check[4].x += 1 ;

check[5].x += 1 ; check[5].y += 1 ;

check[6].y += 1 ;

check[7].x -= 1 ; check[7].y += 1 ;

for (coordinates &f : check)

if(((f.x > -1) and (f.x < computerBoard->getSize())) and ((f.y > -1) and (f.y < computerBoard->getSize())) and (playerBoard->getGrid()[f.x][f.y].getState() == cellEnum::empty))

playerBoard->getGrid()[f.x][f.y].setState(cellEnum::miss) ;

playerBoard->getGrid()[pos[partShip].x][pos[partShip].y].setState(cellEnum::kill) ;

}

}

if (computerBoard->getShips()[i].isSunken())

{

pos = computerBoard->getShips()[i].getCoordinates() ;

for (uint8\_t partShip = 0 ; partShip < computerBoard->getShips()[i].getSize() ; partShip++)

{

for (coordinates &f : check)

f = pos[partShip] ;

check[0].x -= 1 ; // л (по час)

check[1].x -= 1 ; check[1].y -= 1 ;

check[2].y -= 1 ;

check[3].x += 1 ; check[3].y -= 1 ;

check[4].x += 1 ;

check[5].x += 1 ; check[5].y += 1 ;

check[6].y += 1 ;

check[7].x -= 1 ; check[7].y += 1 ;

for (coordinates &f : check)

if(((f.x > -1) and (f.x < computerBoard->getSize())) and ((f.y > -1) and (f.y < computerBoard->getSize())) and (computerBoard->getGrid()[f.x][f.y].getState() == cellEnum::empty))

computerBoard->getGrid()[f.x][f.y].setState(cellEnum::miss) ;

computerBoard->getGrid()[pos[partShip].x][pos[partShip].y].setState(cellEnum::kill) ;

}

}

}

pos = nullptr ;

}

bool GameWindow::isShipsCrossing(Ship &first, Ship &second)

{

coordinates \*pos ;

pos = second.getCoordinates() ;

// Пересечение кораблей

for (uint8\_t z = 0 ; z < second.getSize() ; z++)

if (first.hasCoordinates(pos[z].x, pos[z].y))

return true ;

// Соприкосновение кораблей

coordinates check[8] ;

for (uint8\_t z = 0 ; z < second.getSize() ; z++)

{

for (coordinates &f : check)

f = pos[z] ;

check[0].x -= 1 ; // л (по час)

check[1].x -= 1 ; check[1].y -= 1 ;

check[2].y -= 1 ;

check[3].x += 1 ; check[3].y -= 1 ;

check[4].x += 1 ;

check[5].x += 1 ; check[5].y += 1 ;

check[6].y += 1 ;

check[7].x -= 1 ; check[7].y += 1 ;

for (coordinates &ch : check)

if (first.hasCoordinates(ch.x, ch.y))

return true ;

}

pos = nullptr ;

return false ;

}

bool GameWindow::isEndgame()

{

bool isEndgamePlayer = true;

bool isEndgameComputer = true;

for (uint8\_t x = 0; x < NUM\_SHIPS; x++)

{

if (!playerBoard->getShips()[x].isSunken())

isEndgamePlayer = false;

if (!computerBoard->getShips()[x].isSunken())

isEndgameComputer = false;

}

return !(!isEndgamePlayer and !isEndgameComputer);

}

int GameWindow::draw(sf::RenderWindow &window)

{

window.setTitle("BattleShip | Game") ;

sf::Clock thinkTime ;

sf::RectangleShape zRectangle(sf::Vector2f(window.getSize().x, window.getSize().y));

zRectangle.setPosition(sf::Vector2f(0, 0));

zRectangle.setFillColor(sf::Color(23, 17, 20, 200));

blinkTimer.restart() ;

sf::Event event ;

while (true)

{

window.clear(backgroundColor) ; // Очистка экрана

for (sf::Text &text : playersText) // Отрисовка подписей полей

window.draw(text) ;

drawBoards(window) ; // Отрисовка досок

currentMoveText() ; // Задание свойств

window.draw(moveText);

if (gameState == playerPlacesShips)

{

MouseDrawShip(window) ;

}

while (window.pollEvent(event))

{

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

return -1 ;

if (event.type == sf::Event::KeyReleased and event.key.code == sf::Keyboard::Escape)

return 0 ;

if (gameState == playerPlacesShips)

{

if (event.type == sf::Event::MouseButtonReleased and event.mouseButton.button == sf::Mouse::Right)

currentShipOrientation == h ? currentShipOrientation = v : currentShipOrientation = h ;

if (event.type == sf::Event::MouseButtonReleased and event.mouseButton.button == sf::Mouse::Left)

setShipOnField(sf::Mouse::getPosition(window)) ;

}

if (gameState == playerShot)

{

if (event.type == sf::Event::MouseButtonReleased and event.mouseButton.button == sf::Mouse::Left)

{

int8\_t result = playerMakeShot(sf::Mouse::getPosition(window)) ;

if (result >= 0)

{

checkSunkenShips() ;

if (result == 0)

{

gameState = computerShot ;

thinkTime.restart() ;

}

if (isEndgame())

gameState = playerEndGame;

}

}

}

}

if (gameState == computerPlacesShips)

{

placementShipsByComputer() ;

gameState = playerShot ;

}

if (gameState == computerShot)

{

int8\_t result = computerMakeShot(thinkTime) ;

if (result >= 0)

{

checkSunkenShips() ;

if (result == 0)

gameState = playerShot ;

if (isEndgame())

gameState = computerEndGame;

}

}

if (gameState == playerEndGame)

{

window.draw(zRectangle) ; // Затемняющая маска

window.draw( moveText);

}

if (gameState == computerEndGame)

{

window.draw(zRectangle) ; // Затемняющая маска

window.draw( moveText);

}

window.display() ;

}

}