**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**МЕХАНИКО МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**

**Кафедра веб-технологий и компьютерного моделирования**

**СОЗДАНИЕ КОНСОЛЬНОЙ ИГРЫ**

Курсовая работа

Драгалева Евгения Анатольевича

студента 2 курса,  
специальность «математика и информационные технологии (направление математическое и программное обеспечение мобильных устройств)»

Научный руководитель:

Старший преподаватель,

Д. Ю. Дедков

Минск 2017

Содержание

[Введение 3](#_Toc482892319)

[Постановка задачи 5](#_Toc482892320)

[Проектирование 6](#_Toc482892321)

[Возможные ошибки 10](#_Toc482892322)

[Заключение 11](#_Toc482892323)

[Приложения 12](#_Toc482892324)

[Использованная литература 32](#_Toc482892325)

# Введение

В наши дни уже не осталось людей которые не знают и не слышали о компьютерах, ведь это мощный инструмент в значительной мере упрощающий современную жизнь. Но что такое компьютер без всего программного обеспечения установленного на нем? Бессмысленные куски железяк. Таким образом б**о**льшую часть практической ценности несут программы установленные на компьютере. Они нужны для того что бы решать различные задачи и возвращать некоторый результат, который получить без помощи компьютера было бы довольно сложно, а порой даже невозможно. Но что если вас интересует не результат, а процесс, в ходе которого этот результат был получен. Ведь порой от самого процесса можно получить даже больше удовольствия чем от результата. Что же получается, может быть и такое, что мотивом некоторой деятельности является не результат, а сам ход его получения. Такой вид непродуктивной (зачастую) деятельности называется *игровым процессом* или *игрой*, а соответственно программы, которые служат для организации игрового процесса называются *компьютерными играми.*

Первые компьютерные игровые программы начали появляться еще в начале второй половины прошлого века. Spacewar! созданная в 1962 году была первой игрой написанной для мини-компьютера PDP-1. В 1970-х появились первые аркадные игровые автоматы, что вывело создание игр на коммерческий уровень. Одновременно с этим в научной среде создавались игры для больших университетских мейнфреймов. Создание одной из таких игр привела в итоге к появлению UNIX: в 1969 году Кеном Томпсоном была написана игра Space Travel, сперва для системы Multics, а позже для GECOS (ОС ориентированные на работу с мейнфреймами), но версия на GECOS не удовлетворила программистов, и ее портировали на PDP-7. Процесс портации был довольно сложным: разработчикам пришлось написать систему отладки кода, пакеты для вывода символов и работы с числами с плавающей запятой на ассемблере. Но в процессе портации был написан код который позже был использован при разработке UNIX.

Таким образом что же может быть интереснее чем компьютерные игры? Конечно же их создание, ведь как уже было отмечено выше процесс может принести больше удовольствия чем его итог. Написание игр предоставляет большое поле для экспериментов, поэтому также является отличным способом проверить свои навыки программирования и углубить познания языка программирования или технологии. Полученные в ходе разработки знания и опыт могут быть позже использованы при написании прикладных программ.

Я выбрал для написания аналог игры-кликера Timberman. Во-первых мне показалась довольно интересной идея реализовать ее в консоли с использованием псевдографики, а во-вторых хотелось сделать что-то необычное, отличающееся от стандартных игр, которые выбирают в таких случаях (Змейка, PacMan, Пятнашки и подобных).

# Постановка задачи

Создание консольной игры "Илья Муромец vs Змей Горыныч". Цель игры набрать как можно больше очков, срубая шею Змею Горынычу, один удар по шее - одно очко. Что бы сделать удар по шее нужно нажать на стрелку влево или вправо, при этом Илья Муромец рубит шею находясь соответственно слева или справа от нее. Процесс осложняют шипы на шее, от которых нужно уворачиваться переходя с одной стороны на другую, и таймер отсчитывающий время до того, как Змей Горыныч повергнет Илью Муромца, который увеличивается на единицу времени после каждого удара по шее. Единица времени уменьшается, когда игрок достигает определенного кол-ва очков. Игра заканчивается, когда Илья Муромец ударился о шип или был повержен Змеем Горынычем.

Лучший результат отображается в заголовке консольного окна. Если набранное в последней сессии число очков больше, то оно выводится в заголовок консоли. Перед выходом из игры это число сохраняется в файл и при следующем запуске будет считано из него.

Также необходим конфигурационный файл для настроек игровых параметров

Управление:

|  |  |  |
| --- | --- | --- |
|  | Игровая сессия | GAME OVER |
| Esc | Закончить сессию | Выйти из игры |
| → | Ударить справа |  |
| ← | Ударить слева |  |
| Space |  | Начать заново |

Timberman (http://store.steampowered.com/app/398710/) - аналог игры для понятия механики (аналогия: шея Змея Горыныча - ствол дерева, дровосек – Илья Муромец, ветки - шипы на шее Змея Горыныча)

# Проектирование

Были выделены следующие сущности-объекты:

* **NeckSegment** – фрагмент шеи Змея Горыныча, хранит в себе положение «шипа» на фрагменте.
* **Neck** – шея Змея Горыныча, состоящая из множества **NeckSegment**
* **Timer** – инструмент для отсчета времени до того как Змей Горыныч повергнет Илью Муромца
* **Drawer** – инструмент для отрисовки графики в консоли
* **Session** – игровая сессия, объект который содержит в себе игровую логику.
* **Application** – само приложение, содержит в себе набор инструментов для первоначальной инициализации объектов, запуска игровых сессий, сохранения результатов

Больше информации в диаграмме классов (*Приложение 1*)

**Neck** и **NeckSegment**.

Класс *Neck* было решено реализовать в виде «замкнутой очереди» из пяти *NeckSegment* с указателем front на тот фрагмент, удар по которому будет нанесен с следующим нажатием стрелки. Этот и три следующих фрагмента отображаются в консоли. Когда Илья Муромец совершает удар срабатывает метод *cyclePop*(), он в свою очередь вызывает функцию *setSegmentFrame*() для случайной генерации *NeckSegment*, на который указывает *front*, и перемещает *front* на следующий *NeckSegment*, для этого в *NeckSegment* хранится указатель на следующий фрагмент.

Чтобы в игре не было безвыходных ситуаций, (когда удар с любой стороны приводит к концу игры) *NeckSegment* генерируются так, что «шипы» могут появляться только на четных *NeckSegment*, для этого в *NeckSegment* так же хранится четность фрагмента.

Для отрисовки этого объекта в консоли необходимы методы для получения информации о расположении шипов на видимых фрагментах шеи, и для получения информации о положении шипа у основания (т.е. там где совершается удар) этими методами являются *getSpikeSeq*() *getCurrentSpikeSide*() соответственно.

**Timer**

Таймер как сущность представляет собой набор из 22 тиков одинаковой длительности, которая уменьшается при достижении игроком определенного количества очков. При этом таймер запускает счетчик времени для каждого тика отдельно, конкретно в момент его наступления. Функция отсчета тика *runTick*() спроектирована так, что если во время отсчета времени была нажата управляющая клавиша (Esc, →, ←) то отсчет прерывается и функция счетчика возвращает код нажатой клавиши, если же ни одна из управляющих клавиш не была нажата функция возвращает 0. Полученный результат далее обрабатывается функцией *run*() класса *Session* вызвавшей отсчет тика. Далее в результате обработки полученного значения к количеству тиков либо прибавится либо отнимется 1 для этого в классе предназначены методы *addTick*() и *substractTick*(). В ходе игры пользователь будет достигать определенного количества очков после которых время тика будет изменяться (по умолчанию уменьшаться) для этого используется метод *decreaseTick*(). Такие параметры как начальная длительность тика, минимальная длительность тика, максимальное число тиков и коэффициент на который умножается длина тика при достижении некоторого количества очков можно настроить в конфигурационном файле *settings.cfg.* Для получения текущего количества тиков и текущей длины тика в классе есть методы *getTicks*() и *getTickLength*()

**Drawer**

Класс *Drawer* предназначен для отображения событий приложения в консоль. Он хранит в себе элементы сцены и функции для их отрисовки. Средствами этого класса задаются размер консоли, ее заголовок (методом *updateConsoleTitle*()), отрисовывается графика. Отрисовка графики в консоль реализована методом двойной буферизации. Используется два консольных буфера, сначала сцена отрисовывается в скрытом консольном буфере, потом буферы меняются местами (меняются значения указателей на буферы) и следующая сцена рисуется уже в старом буфере. Эта операция повторяется каждый раз при выводе нового кадра. Использование этого метода обосновано тем, что элементы сцены отрисовываются по очереди, и если не использовать такой подход, а рисовать новую сцену в видимый экранный буфер, пользователь будет видеть процесс отрисовки – элементы сцены будут появляться на экране поэтапно, что может раздражать. Когда необходимо отрисовать сцену вызываются методы для отрисовки элементов сцены. Они задают координаты точки, в которую отрисовывается элемент и вызывают метод *writeToConsole*() который записывает полученный элемент в точку с полученными координатами в скрытый экранный буфер. После того как в скрытом экране буфера отрисуются все элементы сцены вызывается метод *updateScreen*(), меняет значения указателей на срытый и видимый буферы, и делает активным новый видимый буфер.

**Session**

Этот класс запускает каждую игровую сессию методом run(), который содержит в себе игровую логику. Кратко ее можно представить в виде диаграммы в *Приложении 2*. Класс содержит метод *chopResult*(), который выполняет все проверки и обновление окружения в зависимости от того какая стрелка была нажата, методы *run*(), для запуска сессии, и *checkDecreaseTickScorePoint*() который проверяет достиг ли игрок количества очков, когда нужно уменьшить длину тика.

**Application**

Класс *Application* представляет собой само приложение. Объект этого класса должен существовать в единственном экземпляре, поэтому реализован в виде синглтона. Он содержит в себе методы *readHighScore*(), *saveHighScore*() для работы с игровым рекордом: соответственно, для чтения и записи его в бинарном виде в файл. Для хранения рекорда был выбран именно бинарный вид, чтобы пользователю было сложнее поставить себе рекорд обманным путем. Метод *init*() этого класса выполняет первоначальную инициализацию классов параметры которых настраиваются в конфигурационном файле. Для этого в классах *Timer*, *Session* и *Drawer* есть статические инициализирующие методы

Для работы с конфигурационным файлом в этом классе есть отдельные методы. С помощью этого файла пользователь может облегчить, либо усложнить для себе игру настроив некоторые параметры классов:

1. *Timer*:  
   - минимальная длина тика;  
   - начальная длина тика;  
   - коэффициент уменьшения тика;  
   - максимальное количество тиков;
2. *Session*:

- точки уменьшения длительности тика;

1. *Drawer*:   
   -время задержки отрисовки сцен (если необходимо);

Содержание конфигурационного файла *settings.cfg*

DRAWER\_UPDATE\_SCREEN\_DELAY : 60

SESSION\_DECR\_POINT1 : 10

SESSION\_DECR\_POINT2 : 32

SESSION\_DECR\_POINT3 : 64

SESSION\_DECR\_POINT4 : 102

SESSION\_DECR\_POINT5 : 204

SESSION\_DECR\_POINT6 : 350

SESSION\_DECR\_POINT7 : 650

TIMER\_START\_TICKS\_LENGTH : 1.4

TIMER\_MIN\_TICK\_LENGTH : 0.002

TIMER\_DECREASE\_COEF : 0.3

TIMER\_MAX\_TICKS\_AMOUNT : 22

Для реализации приложения был выбран язык C++ так как он предоставляет средства настройки консоли cmd и работы с буферами консольного окна Windows, для этого была использована библиотека windows.h. Так же этот язык был выбран для того чтобы лучше изучить его.

# Возможные ошибки

Ошибки, которые могут произойти в ходе выполнения программы:

1. Нарушение структуры конфигурационного файла;

Вывод программы:  
*Config file structure error. The game will run with standard settings  
Press any key to continue*Программа продолжает работать с стандартными настройками

1. Ошибка открытия\чтения конфигурационного файла;

Вывод программы:  
*Error opening\reading config file, or empty. The game will run with standard settings  
Press any key to continue* Программа продолжает работать с стандартными настройками

1. Конфигурационный файл пуст;

Вывод программы:  
*Error opening\reading config file, or empty. The game will run with standard settings  
Press any key to continue*Программа продолжает работать с стандартными настройками

1. Ошибка открытия\нахождения bin файлов псевдографики;

Вывод программы:  
*Error, bin file not found or empty <*путь к файлу*>. You can download any missing files from github:  
https://github.com/evg-dragalev/coursework\_2crs/tree/master/dev/im\_vs\_zg/im\_vs\_zg/win\_bins  
Press any button to quit*Программа завершается с кодом 3

1. bin файл псевдографики пуст;

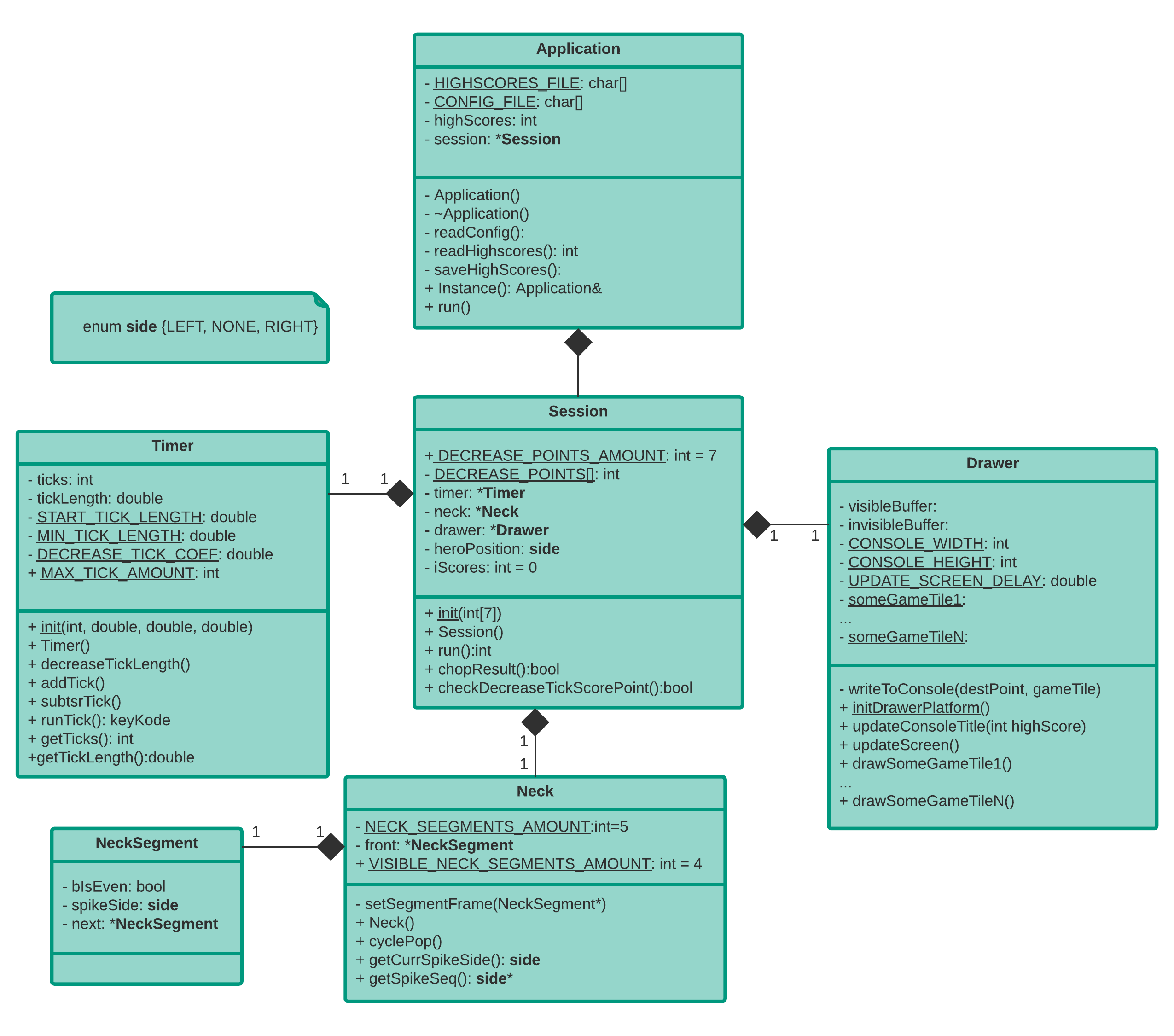
Вывод программы:  
*Error, bin file not found or empty <*путь к файлу*>. You can download any missing files from github:  
https://github.com/evg-dragalev/coursework\_2crs/tree/master/dev/im\_vs\_zg/im\_vs\_zg/win\_bins  
Press any button to quit*Программа завершается с кодом 3

# Заключение

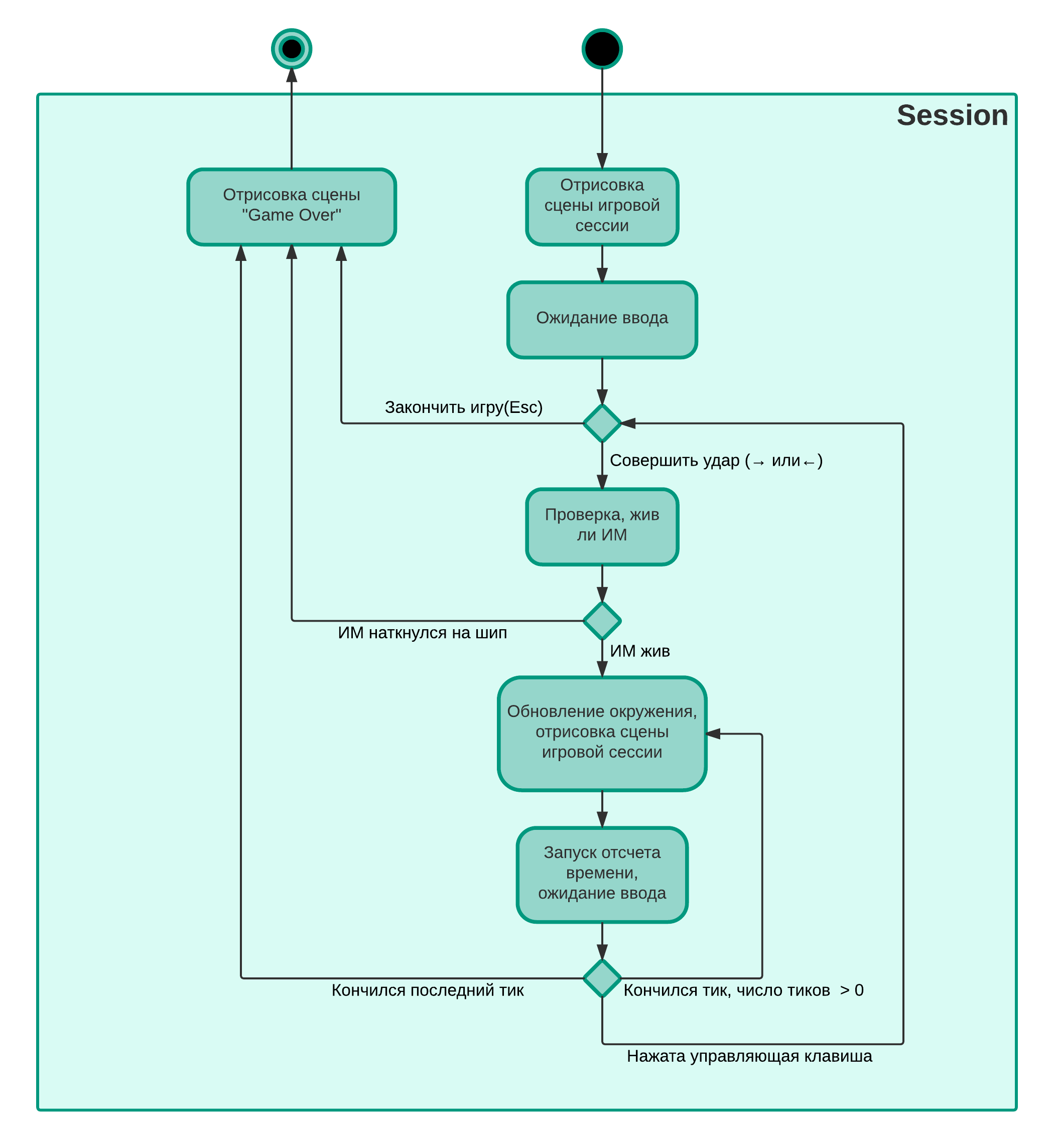
В ходе работы над курсовым проектом было создано консольное приложение: игра «Илья Муромец vs Змей Горыныч». Приложение использует бинарные файлы для чтения графики, конфигурируется с помощью соответствующего файла, сохраняет лучший результат в файл. Работая над проектом, я ознакомился с технологией двойной буферизации и набором низкоуровневых функций для работы с консолью Windows.

# Приложения

1. Диаграмма классов



1. Игровая логика (Session)



1. Реализация

main.cpp

#include "app.h"

int main() {

Application& app = Application::Instance();

app.run();

return 0;

}

headers.h

#pragma once

#include <iostream>

#include <fstream>

#include <conio.h>

#include <time.h>

#include <windows.h>

#include <string.h>

#include <string>

#include <map>

#include "timer.h"

#include "neck.h"

#include "session.h"

enum side { LEFT, NONE, RIGHT };

const int SPACEBAR\_KEYCODE = 32;

const int LEFTARROW\_KEYCODE = 75;

const int RIGHTARROW\_KEYCODE = 77;

const int ESCAPE\_KEYCODE = 27;

app.h

#include "headers.h"

#pragma once

class Application {

static const char HIGHSCORES\_FILE[];

static const char CONFIG\_FILE[];

int iHighScores;

Session\* session;

Application();

~Application();

Application(const Application&) = delete;

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

std::map<std::string, double> readConfig();

double getMapValue(std::map<std::string, double> cfgMap, std::string valName);

int readHighScores();

void saveHighScores(int hs);

public:

static Application& Instance();

void run();

};

app.cpp

#include "app.h"

using namespace std;

const char Application::HIGHSCORES\_FILE[] = "hs.sc";

const char Application::CONFIG\_FILE[] = "settings.cfg";

Application& Application::Instance() {

static Application app;

return app;

}

Application::Application() {

iHighScores = readHighScores();

map <string, double> cfgMap = readConfig();

Timer::init(

(int)getMapValue(cfgMap, "TIMER\_MAX\_TICKS\_AMOUNT"),

getMapValue(cfgMap, "TIMER\_START\_TICKS\_LENGTH"),

getMapValue(cfgMap, "TIMER\_MIN\_TICK\_LENGTH"),

getMapValue(cfgMap, "TIMER\_DECREASE\_COEF")

);

Drawer::initDrawerPlatform(

getMapValue(cfgMap, "DRAWER\_UPDATE\_SCREEN\_DELAY")

);

int decrPoints[Session::DECREASE\_POINTS\_AMOUNT] = {

getMapValue(cfgMap, "SESSION\_DECR\_POINT1"),

getMapValue(cfgMap, "SESSION\_DECR\_POINT2"),

getMapValue(cfgMap, "SESSION\_DECR\_POINT3"),

getMapValue(cfgMap, "SESSION\_DECR\_POINT4"),

getMapValue(cfgMap, "SESSION\_DECR\_POINT5"),

getMapValue(cfgMap, "SESSION\_DECR\_POINT6"),

getMapValue(cfgMap, "SESSION\_DECR\_POINT7"),

};

Session::init(decrPoints);

}

map<string, double> Application::readConfig() {

map <string, double> cfgMap;

ifstream cfgIn(CONFIG\_FILE, ios::in);

try {

if (!cfgIn || cfgIn.eof()) throw "Error opening\reading config file, or empty";

int cycleLimit = 0;

char tmp;

string cfgVarName;

double cfgVarValue;

while (!cfgIn.eof()) {

if (cycleLimit++ > 20) throw "Config file structure error";

cfgIn >> cfgVarName >> tmp >> cfgVarValue;

cfgMap.insert(pair<string, double>(cfgVarName, cfgVarValue));

}

}

catch (const ifstream::failure&) {

cout << "Error reading/finding config file. The game will run with standart settings\n"

<< "Press any key to continue\n";

cfgMap.clear();

\_getch();

return cfgMap;

}

catch (const char\* err) {

cout << err << ". The game will run with standart settings\n"

<< "Press any key to continue\n";

cfgMap.clear();

\_getch();

return cfgMap;

}

return cfgMap;

}

double Application::getMapValue(map<string, double> cfgMap, string valName) {

if (cfgMap.find(valName) != cfgMap.end()) {

return cfgMap.at(valName);

}

else {

return 0;

}

}

int Application::readHighScores() {

int hs = 0;

ifstream hsin(HIGHSCORES\_FILE, ios::binary);

if (hsin || !hsin.eof()) {

hsin >> hs;

}

hsin.close();

return hs;

}

void Application::saveHighScores(int hs) {

ofstream hsout(HIGHSCORES\_FILE, ios::binary);

hsout << hs;

hsout.close();

}

void Application::run() {

int iCurrScores;

int iKeyCode = SPACEBAR\_KEYCODE;

while (iKeyCode != ESCAPE\_KEYCODE) {

Drawer::updateConsoleTitle(iHighScores);

if (iKeyCode == SPACEBAR\_KEYCODE) {

session = new Session();

iCurrScores = session->run();

if (iCurrScores > iHighScores) {

iHighScores = iCurrScores;

}

session->~Session();

}

iKeyCode = ((iKeyCode = \_getch()) == 224) ? getch() : iKeyCode;

}

}

Application::~Application() {

saveHighScores(iHighScores);

}

session.h

#pragma once

#include "headers.h"

#include "drawer.h"

class Session {

public:

static const int DECREASE\_POINTS\_AMOUNT = 7;

static void init(int dp[DECREASE\_POINTS\_AMOUNT]);

Session();

//Вернет количество набранных очков

int run();

bool chopResult(side chopSide);

bool checkDecreaseTickScorePoint() {

for (int i = 0; i < DECREASE\_POINTS\_AMOUNT; i++) {

if (DECREASE\_POINT[i] == iScores) return true;

}

return false;

};

private:

static int DECREASE\_POINT[DECREASE\_POINTS\_AMOUNT];

Drawer\* drawer;

Neck\* neck;

Timer\* timer;

side heroPosition;

int iScores;

};

session.cpp

#include "headers.h"

int Session::DECREASE\_POINT[DECREASE\_POINTS\_AMOUNT] = { 10, 32, 64, 102, 204, 350, 650 };

void Session::init(int dp[DECREASE\_POINTS\_AMOUNT]) {

for (int i = 0; i < 7; i++) {

DECREASE\_POINT[i] = (dp[i]>0) ? dp[i]%1000 : DECREASE\_POINT[i];

}

}

Session::Session() {

drawer = new Drawer;

neck = new Neck;

timer = new Timer;

heroPosition = LEFT;

iScores = 0;

}

bool Session::chopResult(side chopSide) {

heroPosition = chopSide;

side spikeSide = neck->getCurrSpikeSide();

if (spikeSide == heroPosition) {

drawer->drawTimer(timer->getTicks());

drawer->drawNeck(neck->getSpikeSeq());

drawer->drawScores(iScores);

drawer->drawGrave(spikeSide, heroPosition);

drawer->updateScreen();

return false;

}

else {

iScores++;

timer->addTick();

if (checkDecreaseTickScorePoint()) {

timer->decreaseTickLength();

}

drawer->drawTimer(timer->getTicks());

drawer->drawNeck(neck->getSpikeSeq());

drawer->drawScores(iScores);

drawer->drawChop(spikeSide, heroPosition);

drawer->updateScreen();

neck->cyclePop();

spikeSide = neck->getCurrSpikeSide();

if (spikeSide == heroPosition) {

drawer->drawTimer(timer->getTicks());

drawer->drawNeck(neck->getSpikeSeq());

drawer->drawScores(iScores);

drawer->drawGrave(spikeSide, heroPosition);

drawer->updateScreen();

return false;

}

else {

drawer->drawTimer(timer->getTicks());

drawer->drawNeck(neck->getSpikeSeq());

drawer->drawScores(iScores);

drawer->drawHero(heroPosition);

drawer->updateScreen();

return true;

}

}

}

int Session::run() {

drawer->drawTimer(timer->getTicks());

drawer->drawNeck(neck->getSpikeSeq());

drawer->drawScores(iScores);

drawer->drawHero(heroPosition);

drawer->updateScreen();

int iKeyCode;

bool bGameOver = false;

//Первое нажатие

while (true) {

if (\_kbhit()) {

iKeyCode = ((iKeyCode = \_getch()) == 224) ? getch() : iKeyCode;

if (iKeyCode == ESCAPE\_KEYCODE || iKeyCode == RIGHTARROW\_KEYCODE || iKeyCode == LEFTARROW\_KEYCODE) break;

iKeyCode = 0;

}

}

while (!bGameOver) {

switch (iKeyCode) {

case 0: //ничего не нажато, "тик" прошел

if (timer->getTicks() == 0) {

bGameOver = true;

}

else {

timer->substrTick();

drawer->drawTimer(timer->getTicks());

drawer->drawNeck(neck->getSpikeSeq());

drawer->drawScores(iScores);

drawer->drawHero(heroPosition);

drawer->updateScreen();

}

break;

case ESCAPE\_KEYCODE:

bGameOver = true;

break;

case RIGHTARROW\_KEYCODE:

bGameOver = !chopResult(RIGHT);

break;

case LEFTARROW\_KEYCODE:

bGameOver = !chopResult(LEFT);

break;

default:

break;

}

if (!bGameOver) iKeyCode = timer->runTick(timer->getTickLength());

}

drawer->drawTimer(timer->getTicks());

drawer->drawNeck(neck->getSpikeSeq());

drawer->drawScores(iScores);

drawer->drawGrave(neck->getCurrSpikeSide(), heroPosition);

drawer->drawGameOver(iScores);

drawer->updateScreen();

return iScores;

}

timer.h

#pragma once

class Timer {

int\* piTicks;

double\* pdTickLength;

static int MAX\_TICKS\_AMOUNT;

static double START\_TICKS\_LENGTH;

static double MIN\_TICK\_LENGTH;

static double DECREASE\_COEF;

public:

static void init(int maxTicksAmount, double startTicksLength, double minTickLength, double decreaseCoef);

Timer();

void decreaseTickLength();

void addTick();

void substrTick();

int runTick(double dTick);

int getTicks() {

return \*piTicks;

};

double getTickLength() {

return \*pdTickLength;

}

};

timer.cpp

#include "headers.h"

int Timer::MAX\_TICKS\_AMOUNT = 22;

double Timer::MIN\_TICK\_LENGTH = 0.002;

double Timer::START\_TICKS\_LENGTH = 1.4;

double Timer::DECREASE\_COEF = 0.43;

void Timer::init(int maxTicksAmount, double startTicksLength, double minTickLength, double decreaseCoef) {

MAX\_TICKS\_AMOUNT = (maxTicksAmount >= 2 && maxTicksAmount <= MAX\_TICKS\_AMOUNT) ? maxTicksAmount : MAX\_TICKS\_AMOUNT;

MIN\_TICK\_LENGTH = (minTickLength > 0) ? minTickLength : MIN\_TICK\_LENGTH;

START\_TICKS\_LENGTH = (startTicksLength > 0) ? startTicksLength : START\_TICKS\_LENGTH;

DECREASE\_COEF = (decreaseCoef > 0) ? decreaseCoef : DECREASE\_COEF;

}

Timer::Timer() {

piTicks = new int(MAX\_TICKS\_AMOUNT/2);

pdTickLength = new double(START\_TICKS\_LENGTH);

}

void Timer::decreaseTickLength() {

\*pdTickLength = (\*pdTickLength < MIN\_TICK\_LENGTH) ? \*pdTickLength : (\*pdTickLength \* DECREASE\_COEF);

}

void Timer::addTick() {

\*piTicks += (\*piTicks < MAX\_TICKS\_AMOUNT) ? 1 : 0;

}

void Timer::substrTick() {

\*piTicks -= (\*piTicks > 0) ? 1 : 0;

}

int Timer::runTick(double dTick) {

int keyCode = 0;

clock\_t endTickTime = clock() + dTick \* CLOCKS\_PER\_SEC;

while (clock() < endTickTime) {

if (\_kbhit()) {

keyCode = ((keyCode = \_getch()) == 224) ? getch() : keyCode;

if (keyCode == ESCAPE\_KEYCODE || keyCode == RIGHTARROW\_KEYCODE || keyCode == LEFTARROW\_KEYCODE) break;

keyCode = 0;

}

}

return keyCode;

}

neck.h

#include "headers.h"

#pragma once

struct NeckSegment {

bool bIsEven;

enum side spikeSide; // {LEFT, NONE, RIGHT}

NeckSegment\* next;

};

class Neck {

static const int NECK\_SEGM\_AMOUNT;

NeckSegment\* front;

void setSegmentFrame(NeckSegment\* a);

public:

static const int VISIBLE\_NECK\_SEGM\_AMOUNT;

Neck();

void cyclePop();

side getCurrSpikeSide();

side\* getSpikeSeq();

};

neck.cpp

#include "headers.h"

const int Neck::NECK\_SEGM\_AMOUNT = 5;

const int Neck::VISIBLE\_NECK\_SEGM\_AMOUNT = 4;

void Neck::setSegmentFrame(NeckSegment\* a) {

if (a->bIsEven) {

srand(time(0));

int r = rand() % 9;

switch (r) {

case 1:

case 2:

case 3:

case 4:

a->spikeSide = LEFT;

break;

case 0:

a->spikeSide = NONE;

break;

case 5:

case 6:

case 7:

case 8:

a->spikeSide = RIGHT;

break;

default:

std::cerr << "[neck.setSegmentFrame]: Something went wrong";

break;

}

}

else {

a->spikeSide = NONE;

}

}

Neck::Neck() {

front = new NeckSegment;

front->bIsEven = false;

setSegmentFrame(front);

NeckSegment\* pointer = front;

for (int i = 0; i < NECK\_SEGM\_AMOUNT; i++) {

pointer->next = new NeckSegment(); //выделение памяти

pointer->next->bIsEven = !pointer->bIsEven; //инициализация сегмента

setSegmentFrame(pointer->next); //инициализация сегмента

pointer = pointer->next;

}

pointer->next = front;

}

void Neck::cyclePop(){

if (front->bIsEven) {

setSegmentFrame(front);

front = front->next;

}

else {

front = front->next;

}

}

side Neck::getCurrSpikeSide() {

return front->spikeSide;

}

side\* Neck::getSpikeSeq() {

side\* spikeSeq = new side[VISIBLE\_NECK\_SEGM\_AMOUNT];

NeckSegment \*p = front;

for (int i = VISIBLE\_NECK\_SEGM\_AMOUNT - 1; i >= 0; i--) {

spikeSeq[i] = p->spikeSide;

p = p->next;

}

return spikeSeq;

}

drawer.h

#pragma once

#include "headers.h"

class Drawer {

public:

static void initDrawerPlatform(double updateScreenDelay);

static void updateConsoleTitle(int iHightScores);

static void readTile(char\* str, std::streamsize count, char filepath[]);

Drawer();

~Drawer();

void drawFrame();

void drawHero(side heroSide);

void drawGrave(side neckCurrSeg, side heroSide);

void drawNeck(side\* neckSpikesSeq);

void drawTimer(int timeTicks);

void drawGameOver(int scores);

void drawChop(side neckCurrSeg, side heroPosition);

void drawScores(int scores);

void updateScreen();

private:

static const int CONSOLE\_WIDTH = 87; // <=999

static const int CONSOLE\_HEIGHT = 35;// <=999

//Для двойной буферизации

PHANDLE phVisibleBuffer;

PHANDLE phInvisibleBuffer;

static HANDLE hStdout;

static HANDLE hNewScreenBuffer;

static double UPDATE\_SCREEN\_DELAY;

//Тайлы

static CHAR\_INFO chiCrashFromLeft[133]; //7\*19

static CHAR\_INFO chiCrashFromLeftSpike[147]; //7\*21

static CHAR\_INFO chiCrashFromRight[133]; //7\*19

static CHAR\_INFO chiCrashFromRightSpike[147]; //7\*21

static CHAR\_INFO chiDigitsArr[10][6]; //2\*3

static CHAR\_INFO chiFrame[2924]; //34\*86

static CHAR\_INFO chiGameOver[1121]; //19\*59

static CHAR\_INFO chiHeroAtcBottom[15]; //15

static CHAR\_INFO chiHeroAtcHead[5]; //5

static CHAR\_INFO chiHeroDead[66]; //6\*11

static CHAR\_INFO chiHeroDeadL[66]; //6\*11

static CHAR\_INFO chiHeroDeadR[66]; //6\*11

static CHAR\_INFO chiHeroLeft[153]; //9\*17

static CHAR\_INFO chiHeroLeftAtc[224]; //8\*28

static CHAR\_INFO chiHeroRight[153]; //9\*17

static CHAR\_INFO chiHeroRightAtc[224]; //8\*28

static CHAR\_INFO chiNeckFrames[3][416]; //8\*52

static CHAR\_INFO chiScoresFrame[60]; //5\*12

static CHAR\_INFO chiScoresFrameGameOver[60]; //5\*12

static CHAR\_INFO chiTimeStringFull[22]; //22

static CHAR\_INFO chiTimeStringEmpty[22]; //22

void writeToConsole(COORD coordDestPoint, COORD coordBufSize, CHAR\_INFO chiTile[]) { //inline для быстродействия

SMALL\_RECT srctOutRect;

COORD coordBufCoord;

srctOutRect.Top = coordDestPoint.Y;

srctOutRect.Left = coordDestPoint.X;

srctOutRect.Bottom = srctOutRect.Top + coordBufSize.Y - 1;

srctOutRect.Right = srctOutRect.Left + coordBufSize.X - 1;

coordBufCoord.X = 0;

coordBufCoord.Y = 0;

if (!WriteConsoleOutput(

\*phInvisibleBuffer,

chiTile,

coordBufSize,

coordBufCoord,

&srctOutRect)) {

SetConsoleCursorPosition(hStdout, { 0, CONSOLE\_HEIGHT });

std::cout << "[Drawer::writeToConsole()]:WriteConsoleOutput Err Code" << GetLastError();

}

}

};

drawer.cpp

#include <stdio.h>

#include "headers.h"

double Drawer::UPDATE\_SCREEN\_DELAY = 60;

HANDLE Drawer::hStdout;

HANDLE Drawer::hNewScreenBuffer;

CHAR\_INFO Drawer::chiCrashFromLeft[133]; //7\*19

CHAR\_INFO Drawer::chiCrashFromLeftSpike[147]; //7\*21

CHAR\_INFO Drawer::chiCrashFromRight[133]; //7\*19

CHAR\_INFO Drawer::chiCrashFromRightSpike[147]; //7\*21

CHAR\_INFO Drawer::chiDigitsArr[10][6]; //2\*3

CHAR\_INFO Drawer::chiFrame[2924]; //34\*86

CHAR\_INFO Drawer::chiGameOver[1121]; //19\*59

CHAR\_INFO Drawer::chiHeroAtcBottom[15]; //15

CHAR\_INFO Drawer::chiHeroAtcHead[5]; //5

CHAR\_INFO Drawer::chiHeroDead[66]; //6\*11

CHAR\_INFO Drawer::chiHeroDeadL[66]; //6\*11

CHAR\_INFO Drawer::chiHeroDeadR[66]; //6\*11

CHAR\_INFO Drawer::chiHeroLeft[153]; //9\*17

CHAR\_INFO Drawer::chiHeroLeftAtc[224]; //8\*28

CHAR\_INFO Drawer::chiHeroRight[153]; //9\*17

CHAR\_INFO Drawer::chiHeroRightAtc[224]; //8\*28

CHAR\_INFO Drawer::chiNeckFrames[3][416]; //8\*52

CHAR\_INFO Drawer::chiScoresFrame[60]; //5\*12

CHAR\_INFO Drawer::chiScoresFrameGameOver[60]; //5\*12

CHAR\_INFO Drawer::chiTimeStringFull[22]; //22

CHAR\_INFO Drawer::chiTimeStringEmpty[22]; //22

using namespace std;

void Drawer::readTile(char\* str, streamsize count, char filepath[]){

try {

ifstream fin(filepath, ios::in | ios::binary);

fin.seekg(0, ios::end);

if (!fin || (long)fin.tellg() == 0) throw "Error, bin file not found or empty";

fin.seekg(0, ios::beg);

fin.read(str, count);

fin.close();

}

catch (const iostream::failure&) {

cout << "Error reading/finding bin files. Make sure you have win\_bins catalogue in game directory, or download it from github: " << endl

<< "https://github.com/evg-dragalev/coursework\_2crs/tree/master/dev/im\_vs\_zg/im\_vs\_zg/win\_bins " << endl

<< "Press any button to quit";

\_getch();

exit(2);

}

catch (const char\* err) {

cout << err << " " <<filepath << ". You can download any missing files from github: " << endl

<< "https://github.com/evg-dragalev/coursework\_2crs/tree/master/dev/im\_vs\_zg/im\_vs\_zg/win\_bins " << endl

<< "Press any button to quit";

\_getch();

exit(3);

}

}

void Drawer::initDrawerPlatform(double updateScreenDelay) {

UPDATE\_SCREEN\_DELAY = (updateScreenDelay > 0) ? updateScreenDelay : UPDATE\_SCREEN\_DELAY;

char modConCommand[30];

snprintf(modConCommand, 30, "MODE CON: COLS=%d LINES=%d", CONSOLE\_WIDTH, CONSOLE\_HEIGHT);

system(modConCommand);

//Чтение тайлов

readTile((char\*)chiCrashFromLeft, sizeof(chiCrashFromLeft), "win\_bins/crashFromLeft.bin");

readTile((char\*)chiCrashFromLeftSpike, sizeof(chiCrashFromLeftSpike), "win\_bins/crashFromLeftSpike.bin");

readTile((char\*)chiCrashFromRight, sizeof(chiCrashFromRight), "win\_bins/crashFromRight.bin");

readTile((char\*)chiCrashFromRightSpike, sizeof(chiCrashFromRightSpike), "win\_bins/crashFromRightSpike.bin");

readTile((char\*)chiDigitsArr, sizeof(chiDigitsArr), "win\_bins/digitsArr.bin");

readTile((char\*)chiFrame, sizeof(chiFrame), "win\_bins/frame.bin");

readTile((char\*)chiGameOver, sizeof(chiGameOver), "win\_bins/gameOver.bin");

readTile((char\*)chiHeroAtcBottom, sizeof(chiHeroAtcBottom), "win\_bins/heroAtcBottom.bin");

readTile((char\*)chiHeroAtcHead, sizeof(chiHeroAtcHead), "win\_bins/heroAtcHead.bin");

readTile((char\*)chiHeroDead, sizeof(chiHeroDead), "win\_bins/heroDead.bin");

readTile((char\*)chiHeroDeadL, sizeof(chiHeroDeadL), "win\_bins/heroDeadL.bin");

readTile((char\*)chiHeroDeadR, sizeof(chiHeroDeadR), "win\_bins/heroDeadR.bin");

readTile((char\*)chiHeroLeft, sizeof(chiHeroLeft), "win\_bins/heroLeft.bin");

readTile((char\*)chiHeroLeftAtc, sizeof(chiHeroLeftAtc), "win\_bins/heroLeftAtc.bin");

readTile((char\*)chiHeroRight, sizeof(chiHeroRight), "win\_bins/heroRight.bin");

readTile((char\*)chiHeroRightAtc, sizeof(chiHeroRightAtc), "win\_bins/heroRightAtc.bin");

readTile((char\*)chiNeckFrames, sizeof(chiNeckFrames), "win\_bins/neckFrames.bin");

readTile((char\*)chiScoresFrame, sizeof(chiScoresFrame), "win\_bins/scoresFrame.bin");

readTile((char\*)chiScoresFrameGameOver, sizeof(chiScoresFrameGameOver), "win\_bins/scoresFrameGameOver.bin");

readTile((char\*)chiTimeStringFull, sizeof(chiTimeStringFull), "win\_bins/timeStringFull.bin");

readTile((char\*)chiTimeStringEmpty, sizeof(chiTimeStringEmpty), "win\_bins/timeStringEmpty.bin");

//Определения буферов для двойной буферизации

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

hNewScreenBuffer = CreateConsoleScreenBuffer(

GENERIC\_READ | // доступ к чтению/записи

GENERIC\_WRITE,

0, // совместно не используется

NULL, // атрибутов защиты нет

CONSOLE\_TEXTMODE\_BUFFER, // должен быть TEXTMODE

NULL); // зарезервирован, должен быть NULL

if (hStdout == INVALID\_HANDLE\_VALUE ||

hNewScreenBuffer == INVALID\_HANDLE\_VALUE){

SetConsoleCursorPosition(hStdout, { 0, CONSOLE\_HEIGHT });

cout << "[Drawer::initDrawerPlatform()]: CreateConsoleScreenBuffer Err Code - "<< GetLastError();

}

}

void Drawer::updateConsoleTitle(int iHighScore) {

wchar\_t wcConsoleTitle[50];

swprintf(wcConsoleTitle, L"Ilya Muromets VS Zmey Gorynych. Highscore: %d", iHighScore);

SetConsoleTitle(wcConsoleTitle);

}

Drawer::Drawer() {

phInvisibleBuffer = &hStdout;

drawFrame();

phInvisibleBuffer = &hNewScreenBuffer;

drawFrame();

phVisibleBuffer = &hStdout;

SetConsoleCursorPosition(hStdout, { 0, CONSOLE\_HEIGHT });

}

void Drawer::drawFrame() {

COORD coordBufSize;

COORD coordDestPoint;

coordBufSize.X = 86;

coordBufSize.Y = 34;

coordDestPoint.X = 0;

coordDestPoint.Y = 0;

writeToConsole(coordDestPoint, coordBufSize, chiFrame);

}

void Drawer::drawScores(int scores) {

COORD coordBufSize;

COORD coordDestPoint;

coordBufSize.X = 12;

coordBufSize.Y = 5;

coordDestPoint.X = 36;

coordDestPoint.Y = 4;

writeToConsole(coordDestPoint, coordBufSize, chiScoresFrame);

scores = scores % 1000;

coordBufSize.X = 2;

coordBufSize.Y = 3;

coordDestPoint.X = 38;

coordDestPoint.Y = 5;

writeToConsole(coordDestPoint, coordBufSize, chiDigitsArr[scores/100]);

coordDestPoint.X = 41;

// .Y = 5

writeToConsole(coordDestPoint, coordBufSize, chiDigitsArr[scores/10%10]);

coordDestPoint.X = 44;

// .Y = 5

writeToConsole(coordDestPoint, coordBufSize, chiDigitsArr[scores%10]);

}

void Drawer::drawHero(side heroSide) {

COORD coordBufSize;

COORD coordDestPoint;

coordBufSize.X = 17;

coordBufSize.Y = 9;

if (heroSide == LEFT) {

coordDestPoint.X = 15;

coordDestPoint.Y = 24;

writeToConsole(coordDestPoint, coordBufSize, chiHeroLeft);

}

if (heroSide == RIGHT) {

coordDestPoint.X = 52;

coordDestPoint.Y = 24;

writeToConsole(coordDestPoint, coordBufSize, chiHeroRight);

}

}

void Drawer::drawGrave(side neckCurrSeg, side heroSide) {

COORD coordBufSize;

COORD coordDestPoint;

coordBufSize.X = 52;

coordBufSize.Y = 8;

coordDestPoint.X = 16;

coordDestPoint.Y = 25;

writeToConsole(coordDestPoint, coordBufSize, chiNeckFrames[neckCurrSeg]);

coordBufSize.X = 15;

coordBufSize.Y = 1;

coordDestPoint.X = 16;

coordDestPoint.Y = 24;

writeToConsole(coordDestPoint, coordBufSize, chiHeroAtcBottom);

coordDestPoint.X = 52;

writeToConsole(coordDestPoint, coordBufSize, chiHeroAtcBottom);

coordBufSize.X = 11;

coordBufSize.Y = 6;

if (heroSide == LEFT) {

coordDestPoint.X = 21;

coordDestPoint.Y = 27;

if (heroSide == neckCurrSeg) {

writeToConsole(coordDestPoint, coordBufSize, chiHeroDeadL);

}

else {

writeToConsole(coordDestPoint, coordBufSize, chiHeroDead);

}

}

if (heroSide == RIGHT) {

coordDestPoint.X = 52;

coordDestPoint.Y = 27;

if (heroSide == neckCurrSeg) {

writeToConsole(coordDestPoint, coordBufSize, chiHeroDeadR);

}

else {

writeToConsole(coordDestPoint, coordBufSize, chiHeroDead);

}

}

}

void Drawer::drawNeck(side\* neckSpikesSeq) {

COORD coordBufSize;

COORD coordDestPoint;

coordBufSize.X = 52;

coordBufSize.Y = 8;

coordDestPoint.X = 16;

for (int i = 0; i < Neck::VISIBLE\_NECK\_SEGM\_AMOUNT; i++) {

coordDestPoint.Y = 1 + i \* 8;

writeToConsole(coordDestPoint, coordBufSize, chiNeckFrames[neckSpikesSeq[i]]);

}

}

void Drawer::drawTimer(int timerTicks) {

COORD coordBufSize;

COORD coordDestPoint;

coordDestPoint.X = 55;

coordDestPoint.Y = 0;

coordBufSize.X = 22;

coordBufSize.Y = 1;

writeToConsole(coordDestPoint, coordBufSize, chiTimeStringEmpty);

coordBufSize.X = timerTicks;

writeToConsole(coordDestPoint, coordBufSize, chiTimeStringFull);

}

void Drawer::drawGameOver(int scores) {

COORD coordBufSize;

COORD coordDestPoint;

coordBufSize.X = 59;

coordBufSize.Y = 19;

coordDestPoint.X = 14;

coordDestPoint.Y = 4;

writeToConsole(coordDestPoint, coordBufSize, chiGameOver);

coordBufSize.X = 12;

coordBufSize.Y = 5;

coordDestPoint.X = 36;

coordDestPoint.Y = 22;

writeToConsole(coordDestPoint, coordBufSize, chiScoresFrameGameOver);

scores = scores % 1000;

coordBufSize.X = 2;

coordBufSize.Y = 3;

coordDestPoint.X = 38;

coordDestPoint.Y = 23;

writeToConsole(coordDestPoint, coordBufSize, chiDigitsArr[scores / 100]);

coordDestPoint.X = 41;

// .Y = 23

writeToConsole(coordDestPoint, coordBufSize, chiDigitsArr[scores / 10 % 10]);

coordDestPoint.X = 44;

// .Y = 23

writeToConsole(coordDestPoint, coordBufSize, chiDigitsArr[scores % 10]);

}

void Drawer::drawChop(side neckCurrSeg, side heroPosition) {

COORD coordBufSize;

COORD coordDestPoint;

if (heroPosition == LEFT) {

coordBufSize.X = 5;

coordBufSize.Y = 1;

coordDestPoint.X = 23;

coordDestPoint.Y = 23;

writeToConsole(coordDestPoint, coordBufSize, chiHeroAtcHead);

coordBufSize.X = 28;

coordBufSize.Y = 8;

coordDestPoint.X = 16;

coordDestPoint.Y = 24;

writeToConsole(coordDestPoint, coordBufSize, chiHeroLeftAtc);

coordBufSize.X = 15;

coordBufSize.Y = 1;

coordDestPoint.X = 16;

coordDestPoint.Y = 32;

writeToConsole(coordDestPoint, coordBufSize, chiHeroAtcBottom);

if (neckCurrSeg == RIGHT) {

coordBufSize.X = 21;

coordBufSize.Y = 7;

coordDestPoint.X = 44;

coordDestPoint.Y = 25;

writeToConsole(coordDestPoint, coordBufSize, chiCrashFromLeftSpike);

}

else {

coordBufSize.X = 19;

coordBufSize.Y = 7;

coordDestPoint.X = 44;

coordDestPoint.Y = 25;

writeToConsole(coordDestPoint, coordBufSize, chiCrashFromLeft);

}

}

if (heroPosition == RIGHT) {

coordBufSize.X = 5;

coordBufSize.Y = 1;

coordDestPoint.X = 56;

coordDestPoint.Y = 23;

writeToConsole(coordDestPoint, coordBufSize, chiHeroAtcHead);

coordBufSize.X = 28;

coordBufSize.Y = 8;

coordDestPoint.X = 40;

coordDestPoint.Y = 24;

writeToConsole(coordDestPoint, coordBufSize, chiHeroRightAtc);

coordBufSize.X = 15;

coordBufSize.Y = 1;

coordDestPoint.X = 52;

coordDestPoint.Y = 32;

writeToConsole(coordDestPoint, coordBufSize, chiHeroAtcBottom);

if (neckCurrSeg == LEFT) {

coordBufSize.X = 21;

coordBufSize.Y = 7;

coordDestPoint.X = 19;

coordDestPoint.Y = 25;

writeToConsole(coordDestPoint, coordBufSize, chiCrashFromRightSpike);

}

else {

coordBufSize.X = 19;

coordBufSize.Y = 7;

coordDestPoint.X = 21;

coordDestPoint.Y = 25;

writeToConsole(coordDestPoint, coordBufSize, chiCrashFromRight);

}

}

}

void Drawer::updateScreen() {

PHANDLE phBuf;

phBuf = phVisibleBuffer;

phVisibleBuffer = phInvisibleBuffer;

phInvisibleBuffer = phBuf;

Sleep(UPDATE\_SCREEN\_DELAY);

if (!SetConsoleActiveScreenBuffer(\*phVisibleBuffer)) {

SetConsoleCursorPosition(hStdout, { 0, CONSOLE\_HEIGHT });

cout << "[Drawer::updateScreen()]:SetConsoleActiveScreenBuffer Err";

}

}

Drawer::~Drawer() {

if (!SetConsoleActiveScreenBuffer(hStdout)) {

SetConsoleCursorPosition(hStdout, { 0, CONSOLE\_HEIGHT });

cout << "[Drawer::~Drawer()]:SetConsoleActiveScreenBuffer Err";

}

}

# Использованная литература

1. Абстракция данных и решение задач на C++. Стены и зеркала, 3-е издание. : Пер. с англ. — М.: Издательский дом "Вильямс", 2003. — 848 с : ил. — Парал. тит. англ.
2. Windows Dev Center. Console Functions © Microsoft 2017 – Режим доступа: <https://msdn.microsoft.com/en-us/library/windows/desktop/ms682073(v=vs.85).aspx> . – Дата доступа: 14.05.2017.
3. Windows Dev Center. Low-Level Console Output Functions © Microsoft 2017 – Режим доступа: <https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms684206(v=vs.85).aspx> . – Дата доступа: 14.05.2017.
4. Три возраста паттерна синглтон. – Режим доступа: <https://habrahabr.ru/post/147373/> . – Дата доступа: 14.05.2017