Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Конструирование программного обеспечения (КПО)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**ПРОГРАММНОЕ СРЕДСТВО**

**«Space Invaders»**

БГУИР КП 1-40 01 01 003 ПЗ

Студент гр. 351004 Бражалович А.И.

Руководитель асс. Шостак Е.В.

Минск 2024

СОДЕРЖАНИЕ

[Введение 5](#_Toc167417859)

[1 Анализ предметной области 7](#_Toc167417860)

[1.1 Обзор аналогов 7](#_Toc167417861)

[1.2 Постановка задачи 9](#_Toc167417862)

[2 Проектирование программного средства 10](#_Toc167417863)

[2.1 Структура программы 10](#_Toc167417864)

[2.2 Проектирование интерфейса программного средства 10](#_Toc167417865)

[2.3 Проектирование функционала программного средства 14](#_Toc167417866)

[3 Разработка программного средства 18](#_Toc167417867)

[3.1 Прорисовка материалов для игры 18](#_Toc167417868)

[3.2 Игровой процесс 20](#_Toc167417869)

[3.3 Работа со списком игроков 21](#_Toc167417874)

[4 Тестирование программного средства 24](#_Toc167417875)

[5 Руководство пользователя 25](#_Toc167417876)

[5.1 Интерфейс программного средства 25](#_Toc167417877)

[Заключение 30](#_Toc167417879)

[Список использованных источников 30](#_Toc167417880)

[Приложение А. Текст программы 31](#_Toc167417881)

# ВВЕДЕНИЕ

В последние десятилетия сфера информационных технологий претерпела множество изменений, и на сегодняшний день перед специалистами стоят задачи самого разного спектра: от разработки сложного технического обеспечения для крупных производственных организаций до создания простых аркадных мобильных приложений для развлечения среднестатистического пользователя. Именно в последней категории программных продуктов начала развиваться целая индустрия компьютерных игр. Сегодня рынок переполнен продуктами разного качества и жанров: платформеры, шутеры, приключения и головоломки.

Своему появлению компьютерные игры обязаны таким пионерам, как Ральф Бауэр (инженер, выдвинувший идею интерактивного телевидения в 1951 году), Александр Дуглас (разработчик «ОХО» – компьютерной реализации крестиков-ноликов как примера для диссертации на тему взаимодействия человека и компьютера в 1952 году) и Уильям Хигинботам (создатель первой многопользовательской игры «Tennis for Two» в 1958 году). Однако широкое распространение компьютерные игры получили только с выпуском первого компьютера серии PDP. Он получил название PDP-1, и только спустя два года, в 1962 году, для него была разработана первая компьютерная игра – «SpaceWar!».

Прообразами современных компьютерных игр всегда являлись вещи или события, уже существующие и перенесенные в цифровой формат. Так, жанр MMORPG стал продолжением настольной игры «Dungeons & Dragons», а аркадные шутеры, такие как «Space Invaders», были вдохновлены концепцией борьбы с нападающими волнами врагов, часто заимствованной из научно-фантастических фильмов и книг.

История создания и развития «Space Invaders» берет начало в 1978 году, когда японская компания Taito выпустила эту игру. Разработанная Томохиро Нисикадо, она быстро завоевала популярность благодаря своим простым, но увлекательным игровым механикам и незабываемому звуковому сопровождению. «Space Invaders» стала не просто игрой, а культурным феноменом, определившим направление развития аркадных игр. Основная идея заключалась в том, чтобы игрок отбивал волны инопланетных захватчиков, постепенно увеличивая сложность и напряжение.

Игра «Space Invaders» стала новаторской в своем роде. Она установила стандарт для аркадных шутеров, предложив игрокам постепенное увеличение сложности и требуя от них стратегического мышления и быстроты реакции. Простые, но затягивающие механики сделали игру популярной среди широкого круга игроков, от детей до взрослых, и привлекли внимание общественности к видеоиграм как к форме развлечения.

Почему сегодня «Space Invaders» остается популярной в этом современном, быстро развивающемся мире? Одна из возможных причин заключается в том, что эта игра обращается к базовому человеческому инстинкту соревновательности и стремления к совершенству. Есть что-то очень удовлетворяющее в уничтожении врагов и достижении новых уровней. Другая причина в том, что правила игры очень просты и понятны. Люди всех возрастов могут быстро научиться игре и начать наслаждаться захватывающим процессом. Глобальная популярность «Space Invaders» показывает, что игра легко понятна и привлекательна для людей по всему миру, так как основывается на простых и интуитивно понятных механиках.

«Space Invaders» также внесла значительный вклад в развитие игровой культуры и индустрии. Она стала вдохновением для множества последующих игр и способствовала развитию технологий и дизайнерских подходов в игровой индустрии. Многие современные шутеры и аркадные игры черпают вдохновение из оригинальных механик и концепций, заложенных в «Space Invaders».

Таким образом, целью данного проекта является создание игрового приложения, воссоздающего классический геймплей «Space Invaders» и адаптирующего его для современных пользователей. Этот проект направлен на оживление ностальгических воспоминаний о золотой эре аркадных игр, а также на демонстрацию того, как принципы и механики классики могут быть успешно адаптированы с использованием современных технологий и подходов в разработке игр.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

На данный момент существует большое множество вариаций игры «Space Invaders». В каждой вариации этой игры есть разные режимы и модификации.

Первой и самой популярной редакцией этой игры стала версия, выпущенная для игровых автоматов в 1978 году в Японии. Позже ее портировали на персональные компьютеры.

В портированных версиях сохранились дизайн и звуковое сопровождение. Также добавили кооперативный режим и новые механики атак.

Внешний вид данного приложения представлен на рисунке 1.1.

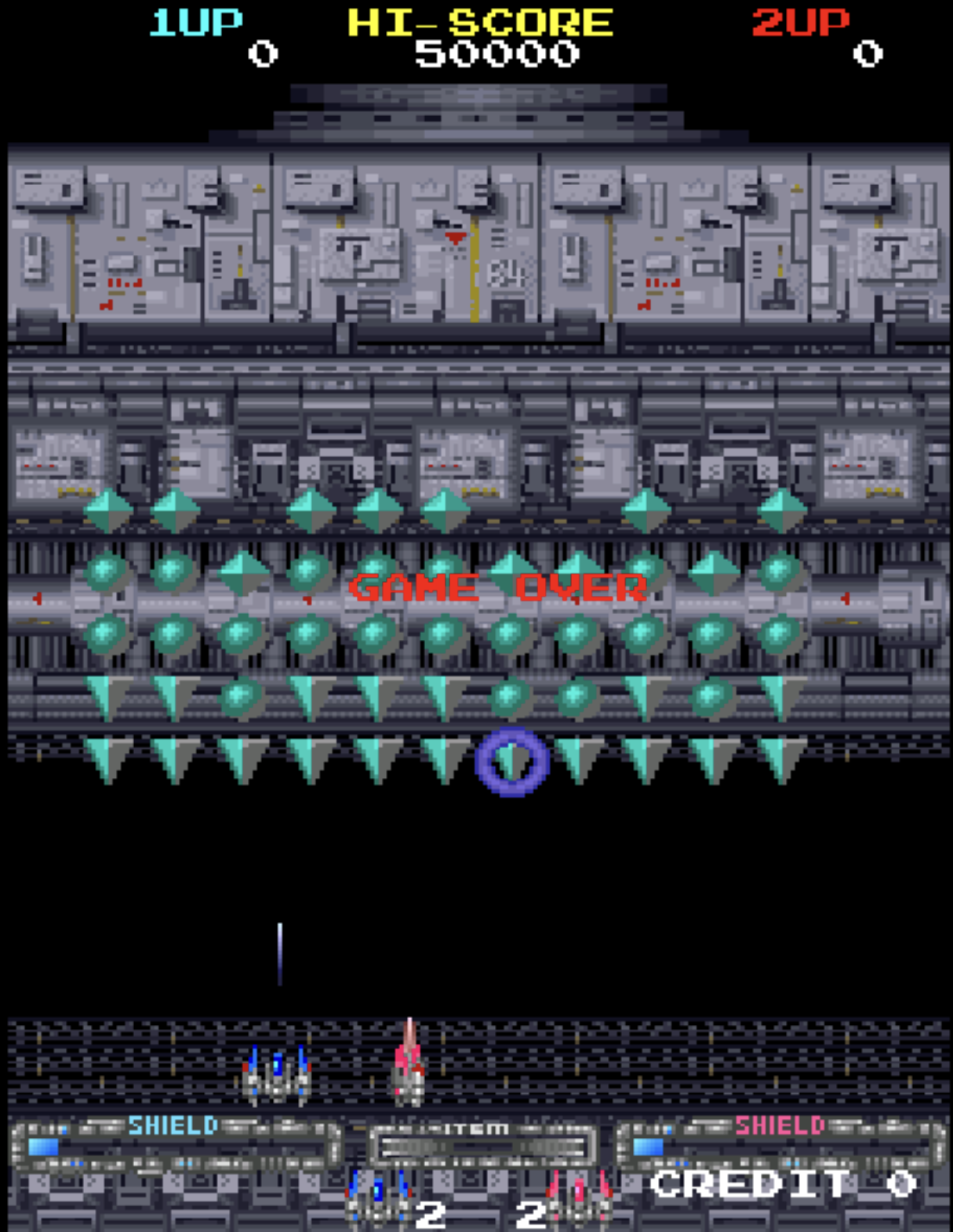


Рисунок 1.1 – Программное средство «Space Invaders Part IV»

Далее стоит рассмотреть реализацию «Space Invaders» от Google. В этой версии игры обычные барьеры заменены словом Google. Противники совершают большее количество атак. Также в этой версии есть обучение и несколько вариантов управления главной пушкой.

Внешний вид окна данного приложения представлен на рисунке 1.2.



Рисунок 1.2 – Программное средство «Space Invaders»

Еще более простую версию игры «Space Invades» от разработчика MaxMouse можно встретить в расширениях для браузера Chrome. Инопланетяне, как и сама главная пушка заменены на прямоугольники. Из недостатков можно заметить отсутствие специльных стен, как в оригинальной игре. На рисунке 1.3 представлен интерфейс игрового приложения.

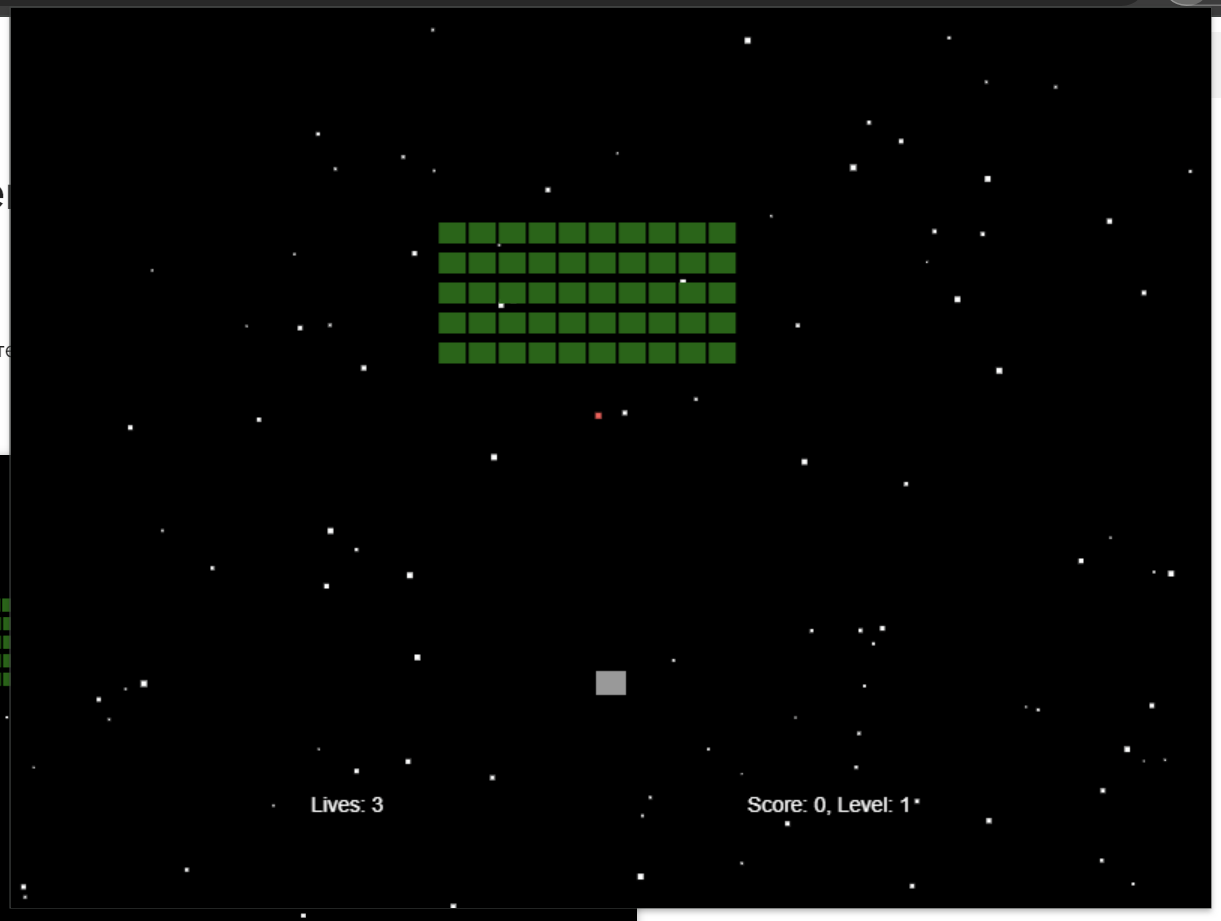


Рисунок 1.3 – Программное средство «Simple Space Invaders»

В рассмотренных аналогах можно увидеть разные механики игры, реализованные в разных стилях. Многие аналоги полностью повторяют оригинальное издание.

## Постановка задачи

В рамках данной курсовой работы планируется разработать игру «Space Invaders» для платформы Windows. После рассмотрения аналогов выявились некоторые недостатки, которые можно учесть в данной курсовой работе. На основе аналогов возможно добавление новых функций, не имеющихся в рассмотренных аналогах. Добавление новых функций отличных от похожих проектов привнесет разнообразие в игровой процесс. В процессе разработки должны быть реализованы базовые функции игры:

* выстрелы пушки;
* движение главной пушки и врагов;
* создание таблицы с рейтингом игроков.

Планируется работа с внешними устройствами:

* сохранение списка игроков в файл;
* добавление новых игроков в файл;
* удаление имеющихся игроков из файла;
* запуск списка игроков из файла.

Планируется добавление такого функционала, как:

* подсчет текущего числа очков;
* выстрелы врагов в главную пушку;
* механику жизней главной пушки.

Для разработки программного средства будет использоваться язык программирования С++ и среда разработки Clion 2024.2.1.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

При разработке приложения будет использовано одиннадцать модулей:

* gameWindowUnit – модуль, в котором происходит процесс игры;
* gameOverWindowUnit – модуль, обеспечивающий отрисовку окна окончания игры;
* TextureManagerClass – модуль, обеспечивающий удобную загрузку изображений;
* mainWindowUnit – модуль, обеспечивающий отображение главного окна приложения;
* CannonClass – модуль, содержащий реализацию логики работы пушки;
* choosePlayerWindow – модуль, обеспечивающий отображение окна взаимодействия со списком игроков;
* pauseWindowUnit – модуль, обеспечивающий отображение окна паузы приложения;
* newPlayerWindowUnit – модуль, обеспечивающий отображение окна добавления нового игрока;
* fileModule – модуль, обеспечивающий корректную работу с типизированными файлами и содержащий класс реализующий список;
* scoreBoardWindowUnit – модуль, обеспечивающий отрисоку таблицы рекордов игроков.
* DropBox – модуль, реализующий отображение и логику работы выпадающего списка.
* AlienClass – модуль, реализующий корректное создание, отображение и логику поведения врагов.
* RocketClass – модуль, содержащий реализацию снарядов главной пушки и врагов.

## 2.2 Проектирование интерфейса программного средства

При разработке программного средства за основу будет взят дизайн программного средства «Space Invaders».

2.2.1 Главное меню

Главное меню приложения будет состоять из четырех основных кнопок:

* первой должна располагаться StartButton, по нажатию которой пользователь переходит в модуль gameWindowUnit;
* второй расположится кнопка PlayerButton, позволяющая управлять списком игроков;
* третьей будет ScoreListButton, нажав на которую, пользователь может просмотреть список рекордов игроков;
* четвертой будет кнопка ExitButton, которая позволяет покинуть программное средство.

Макет главного меню приложения представлен на рисунке 2.1.

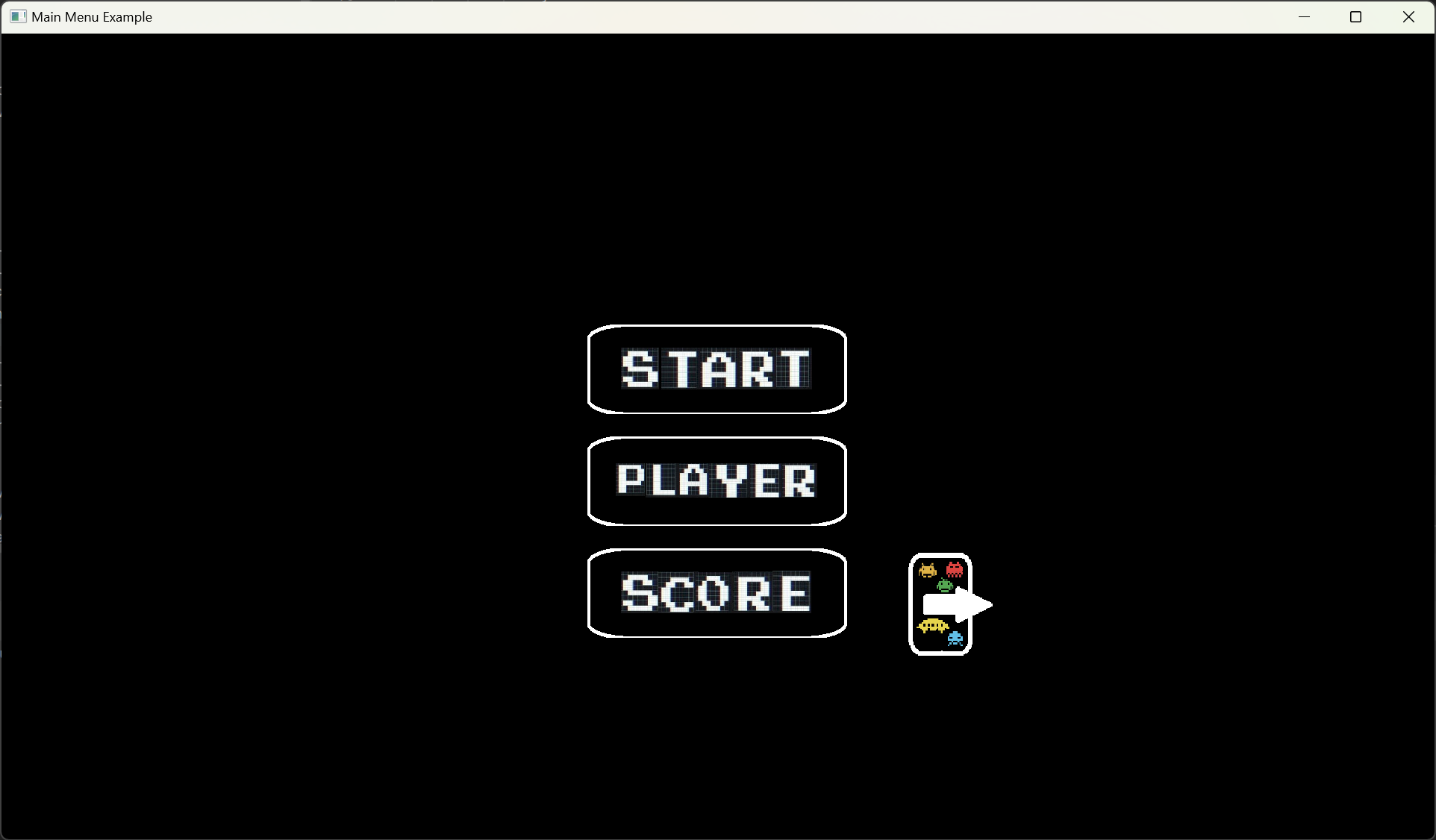




Рисунок 2.1 – Макет главного меню приложения

В главном меню будет предусмотрена возможность вызова контекстного меню с информацией о разработчике.

2.2.2 Окно «Player»

Окно игроков будет содержать управление списком игроков. Кнопки в данном окне будут реализованы с помощью RectangleShape. Для отображения игроков палнируется использовать DropBox. Макет окна игроков представлен на рисунке 2.2.

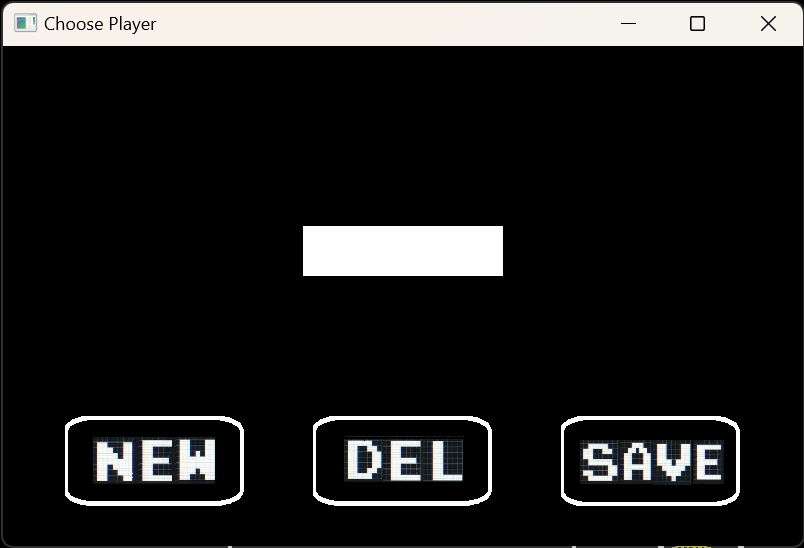


Рисунок 2.2 – Макет окна игроков

В окне игроков пользователю будет доступна возможность добавлять новых игроков и удалять уже имеющихся.

2.2.3 Окно добавления нового игрока

Окно добавления нового игрока будет содержать кнопку добавления и поле ввода имени игрока. Макет окна добавления нового игрока представлен на рисунке 2.3.



Рисунок 2.3 – Макет окна добавления нового игрока

Кнопка в окне добавления игрока будет реализована с помощью компонента RectangleShape.

2.2.4 Окно паузы

Окно паузы должно состоять из двух основных кнопок:

* первой должна располагаться кнопка, по нажатию которой пользователь возвращается к игре;
* второй будет кнопка «Exit», нажав на которую пользователь вернется в главное меню.

Макет окна паузы представлен на рисунке 2.4.

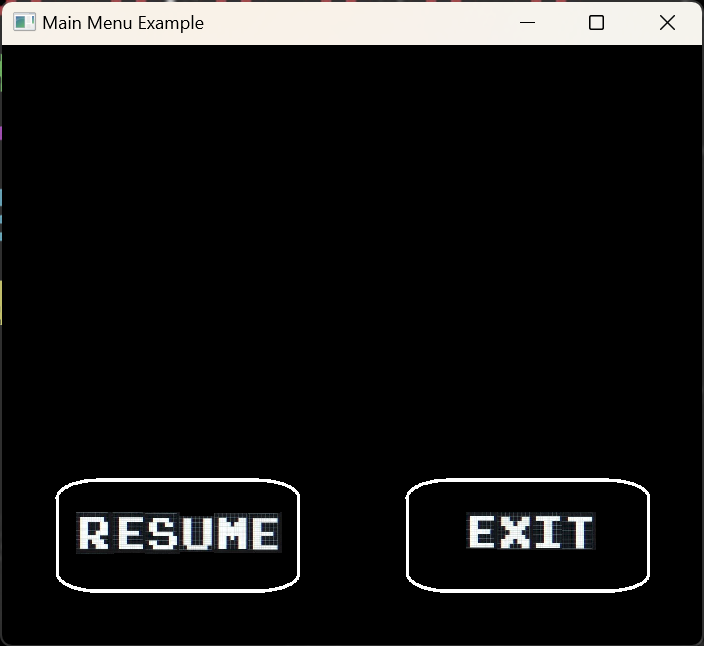


Рисунок 2.4 – Макет окна паузы

Для более удобного взаимодействия с программным средством планируется добавить вызов окна паузы с помощью нажатия на клавишу «Escape».

2.2.5 Окно окончания игры

Окно окончания игры будет содержать информацию о том, что игра закончилась. Макет окончания игры представлен на рисунке 2.5.

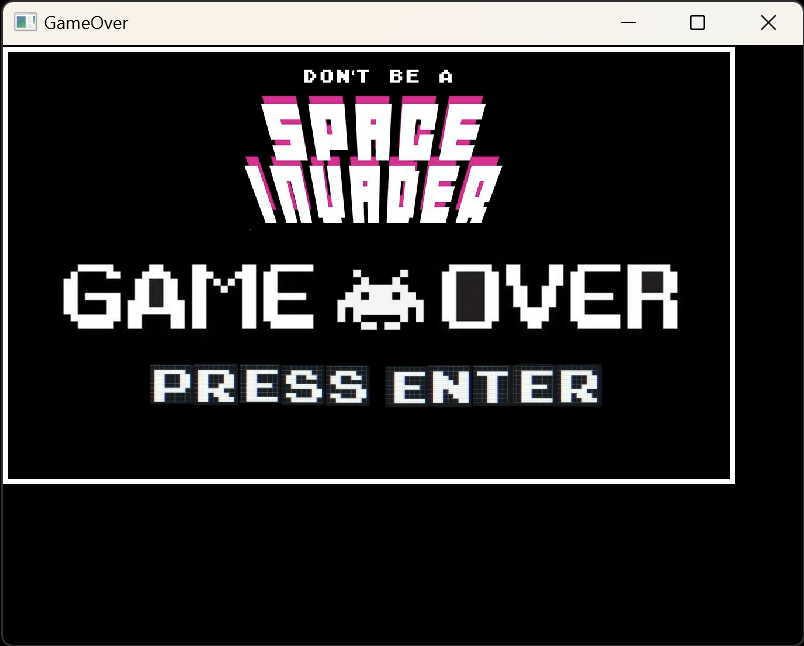


Рисунок 2.5 – Макет окна окончания игры

Для выхода из окна окончания игры будет ожидаться нажатие на клавишу Enter.

2.2.6 Окно игрового процесса

Окно игрового процесса будет состоять из множества элементов:

* информация о количестве жизней игрока;
* кнопка паузы;
* главная пушка и ее снаряды;
* враги;
* счет игрока.

На экране будут отображены все враги и главная пушка. Игровой процесс постоянно прогружает новые состояния врагов с помощью компонента Clock, отвечающего за вывод за определенные промежутки времени нового изображения. Макет окна игрового процесса представлен на рисунке 2.6.

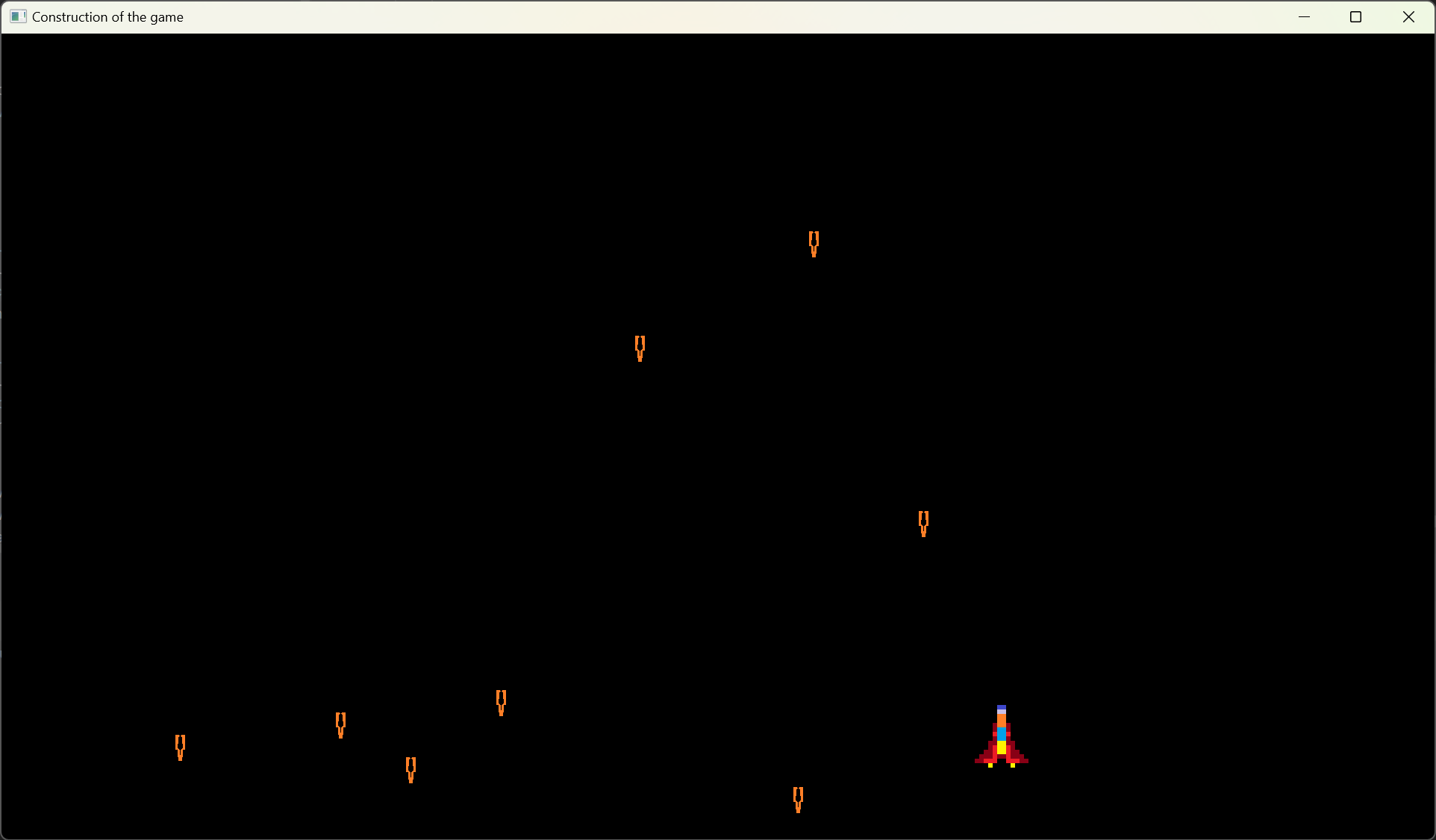


Рисунок 2.6 – Макет окна игрового процесса

Будет реализован выход в главное меню из окна игрового процесса.

2.2.7 Окно рекордов игроков

Окно рекордов игроков состоит из таблицы рекордов и кнопки подтверждения. Макет окна рекордов игроков представлен на рисунке 2.7.



Рисунок 2.7 – Макет окна рекордов игроков

В окне рекордов игроков будет предусмотрена возможность пролистывания таблицы.

## 2.3 Проектирование функционала программного средства

Разработка алгоритмов является ключевой фазой в проектировании программного средства. Игровое приложение «Space Invaders» должно предоставлять пользователю такой минимальный функуционал, как:

* отрисовка игрового поля, пушки и врагов;
* выстрелы пушки;
* движение врагов.

2.3.1 Отрисовка игрового поля, врагов и пушки

Отрисовка игрового поля, врагов и пушки в игре может быть реализована с использованием компонентов Sprite и Texture, которые представляют собой графические контейнеры. При отрисовке игрового поля необходимо учесть размеры врагов и пушки заранее.

Первоначально отрисовываются фон, враги, пушка и количество жизней игрока на игровом поле. Затем определяются границы врагов и главной пушки.

Во время старта таймера враги принимают начальное расположение на игровом поле. Далее происходит рассчет их границ.

После отрисовки врагов на игровом поле можно перейти к их передвижению. Враги двигаются по предопределенному маршруту, постепенно приближаясь к главной пушке. Враги могут быть представлены в виде графических объектов, размещенных на игровом поле в виде таблицы. Блок-схема алгоритма процедуры alienInitialization приведена на рисунке 2.8.



Рисунок 2.8 – Блок-схема процедуры alienInitialization

2.3.2 Выстрелы пушки

Выстрелы пушки основаны на простом перемещении снаряда по координатам. При передвижении пули с каждым перемещением проверяется пересечение пули с врагом или с границей игового поля. После пересечения пуля пропадает с игового поля. Блок-схема алгоритма процедуры deleteRocket приведена на рисунке 2.9.



Рисунок 2.9 – Блок-схема процедуры deleteRocket

После выполнения этой процедуры необходимо будет проверять пересечение с границами врага и границей игрового поля.

2.3.3 Движение врагов

Алгоритм движения врагов включает в себя обновление параметров позиции каждого врага по обеим координатам и состояния. На рисунке 2.10 приведена блок-схема алгоритма процедуры moveAliens.



Рисунок 2.10 – Блок-схема процедуры MoveAliens

Граммотное проектирование проекта поможет в будущем избежать ошибок при разработке.

1. **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**3.1 Прорисовка материалов для игры**

Одними из важных процедур программы являются те, что связаны с прорисовкой текущего состояния игрового поля, информационной части об игре. В основу работы с отображением были добавлены компоненты TImage и TBitMap.

**3.1.1** Прорисовка движения пушки

Прорисовка пушки игрока происходит после нажатия на стрелки на клавиатуре. Отрисовка пушки производится с помощью компонета Sprite. Код процедуры moveCannon приведен ниже.

void moveCannon(sf::Keybord) {

// Проверка на нажатие левой стрелки

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {

//Сдвиг пушки влево

cannon.moveLeft();

}

//Проверка на нажатие правой стрелки

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {

//Сдвиг пушки вправо

cannon.moveRight();

}

//Пересчёт времени от прошлого нажатия

float deltaTime = clock.restart().asSeconds();

//Проверка на пересечение левой границы игрового поля

if (cannonSprite.getPosition().x < 0) {

//Перемещение пушки в крайнее левое положение

cannonSprite.setPosition(0, cannonSprite.getPosition().y);

//Проверка на пересечение правой границы поля

} else if (cannonSprite.getPosition().x + cannonSprite.getGlobalBounds().width > 1900) {

//Перемещение пушки в крайнее правое положение

cannonSprite.setPosition(1900 - cannonSprite.getGlobalBounds().width, cannonSprite.getPosition().y);

}

}

**3.1.2** Прорисовка снарядов врагов

Снаряды отображаются по мере выстрелов врагов. Если количество снарядов превышает максимум, то враги перестают стрелять до момента уменьшения количества снарядов на экране. Код процедуры deleteAlienBullet приведен ниже.

void deleteAlienBullet(std::vector<RocketClass> &alienBullet) {

//Проходим по массиву ракет

alienBullet.erase(std::remove\_if(alienBullet.begin(), alienBullet.end(), [&cannon, lives](RocketClass& alien) {

//Проверяем на пересечение с главной пушкой

if (alien.getBounds().intersects(cannon.getBounds())) {

//Возвращаем истинну

return true;

} return false;

}),

alienBullet.end());

//Проходим по массиву врагов

for (auto itAlienRocket = alienBullet.begin(); itAlienRocket != alienBullet.end(); ) {

//Проверяем пересечение ракеты и пушки

if (itAlienRocket->getBounds().intersects(cannon.getBounds())) {

//Декрементируем колиество жизней

lives--;

//Проверяем на отсутсвие жизней

if (lives == 0) {

//Закрываем игровое окно

gameWindow.close();

//Открываем окно конца игры

gameOver();

//Обновляем рекорд

if (currentPlayer->data.score < score) {

currentPlayer->data.score = score;

}

}

//Очищаем ресурсы после столкновения ракеты

itAlienRocket = alienBullet.erase(itAlienRocket); // Удаление пули

} else {

//Идём далее по циклу

++itAlienRocket;

}

}

}

**3.1.3** Создание снарядов

С помощью конструктора RocketClass можно создать снаряд пушки или врага. Данная процедура реализована с помощью Sprite. Код конструктора RocketClass приведен ниже.

//Передаём начальные значиения и указываем скорость при создании

RocketClass::RocketClass(float startX, float startY, bool cannon) : speed(5.0f) {

//Если это пушка

if (cannon) {

//Загружаем текстуру ракеты пушки

rocketSprite.setTexture(TextureManager::getTexture("images\\BulletRocke t.png"));

//Устанавливаем соответсвующее соотношение сторон

rocketSprite.setScale(4, 4);

//Повышаем скорость

speed = 8.0f;

} else {

//Загружаем текстуру ракеты врага rocketSprite.setTexture(TextureManager::getTexture("images\\AlienBullet .png"));

}

//Устанавливаем начальное положение нового снаряда

rocketSprite.setPosition(startX, startY);

}

Очень важно при отрисовке вовремя изменять параметры состояния врагов, снарядов и пушки.

**3.2 Игровой процесс**

При разработке программного средства основной упор делался на игровую логику. Ниже будут рассмотрены основные подпрограммы для реализации игрового процесса.

**3.2.1** Высчитывание новых параметров для врагов

Данная функция довольна важна для динамики игры. Код функции calculateAlienBounds приведен ниже.

Sf::RectangleShape сalculateAlienBounds(AlienClass &alien) {

// Сохранение ширины врага

int alienWidth = alien.getSide().width;

// Сохранение высоты врага

int alienHeight = alien.getSide().height;

// Сохранение левой границы врага

int alienLeft = alien.getSide().left;

// Сохранение правой границы врага

int alienRight = alienLeft + alienWidth;

// Сохранение верхней границы врага

int alienTop = alien.getSide().top;

// Сохранение нижней границы врага

int alienBottom = alienTop + alienHeight;

// Рассчет границ врага

sf::RectangleShape alienBounds(alienLeft, alienRight, alienTop, alienBottom);

return alienBounds;

}

Данная процедура обходит всех врагов для пересчета границ после их передвижения.

**3.2.2** Высчитывание состояния врагов

Для корректной реализации логики игры нужно проверять состояние неуничтоженных врагов. Каждый враг, должен корректно обновлять состояние в зависимости от времени и положения. Код функции uptadeAlien приведен ниже.

void AlienClass::update(float deltaTime, std::vector<RocketClass> &alienBullets) {

//Пережвижение врага по заданному направлению

sprite.move(speed \* direction.x \* deltaTime, speed \* direction.y \* deltaTime);

//Обновляем счётчик времени посел передвижения

shootTimer += deltaTime;

//Проверяем превысило ли время заданный интевал

if (shootTimer >= shootInterval && canShoot) {

//Создаём снаряд врага с координатами его нижней границы

alienBullets.emplace\_back(sprite.getPosition().x + sprite.getGlobalBounds().width / 2, sprite.getPosition().y + sprite.getGlobalBounds().height, false);

//Обнуляем счётчик времени после выстрела

shootTimer = 0.0f;

//Задаём новое значение интервала

shootInterval = 1.0f + static\_cast<float>(rand()) / static\_cast<float>(RAND\_MAX / 2.0f);

}

}

**3.2.4** Уничтожение врагов

Процедура реализует уничтожение врагов, которые были поражены снарядом пушки. Код процедуры checkCannonRocket приведен ниже.

void checkCannonRocket() {

//Проходим по массиву рокет

for (auto itRocket = rockets.begin(); itRocket != rockets.end(); ) {

//Инициализируем переменную для проверки удалена ли ракета

bool rocketRemoved = false;

//Проходим по массиву врагов

for (auto itAlien = aliens.begin(); itAlien != aliens.end(); ) {

//Проверяем пересечение границ ракеты и врага

if (itRocket->getBounds().intersects(itAlien->getBounds())) {

//Удаляем врага

itAlien = aliens.erase(itAlien);

//Удаляем ракету

itRocket = rockets.erase(itRocket);

//Ракета удалена

rocketRemoved = true;

//Увеличиваем общий счёт

score += itAlien->getScore();

//Проверяем наличие врагов

if (aliens.empty()) {

//Создаём новых врагов

alienInitialization(aliens, alienSpeed, alienAcceleration, alienDirection);

}

//Выход из цикла

break;

}

}

}

} с



## 3.3 Работа со списком игроков

Работа со списком игроков включает в себя использование однонаправленного списка. Необходимо разработать процедуры добавления, удаления элементов списка, а также очищение списка.

**3.3.1** Добавление нового элемента в список

Игроки представляют собой однонаправленный список. Данная структура данных была выбрана по той причине, чтобы более комфортно хранить информацию об игроках. Для того чтобы обновлять данные, было принято добавлять информацию о новом игроке в конец списка. Код процедуры PlayerLinkedList::add приведен ниже.

void PlayerLinkedList::add(std::string &name, int score) {

// Создание нового узла

auto \*newPlayer = new player\_t{{name, score},nullptr };

// Список пуст

if (head == nullptr) {

//Инициализация списка

head = newPlayer

} else {

// Получаем указатель на конец списка

player\_t \*temp = head;

while (temp->next) {

temp = temp->next;

}

// Добавляем новый элемент в конец

temp->next = newPlayer;

}

}

* + 1. Удаление элемента из списка

Для удаления игрока из списка надо пройтись по всему списку и извлечь элемент. Код функции PlayerLinkedList::remove приведен ниже.

void PlayerLinkedList::remove(const std::string &name) {

//Список пуст

if (head == NULL) {

//Завершаем процедуру

return;

}

// Соответсвует ли текущееимя с головой списка

if (head->data.name == name) {

player\_t \*temp = head;

head = head->next;

delete temp;

return;

}

// Проходимся по списку и сравниваем игроков

player\_t \*temp = head;

while (temp->next && temp->next->data.name != name) {

temp = temp->next;

}

//Проверяем наличие следующего элемента в списке

if (temp->next) {

player\_t \*tempRemove = temp->next;

//Переустанавливаем указатель

temp->next = temp->next->next;

//Удаляем игрока

delete tempRemove;

}

}

**3.3.3** Очищение игроков

Важным аспектом использования динамической памяти является очищение. Код процедуры PlayerLinkedList::clear приведен ниже.

void PlayerLinkedList::clear() {

//Получаем указатель на первый элемент списка

player\_t\* currPlayer = head

while (head) {

// Поочередно удаляем элементы из списка

player\_t \*temp = head;

head = head->next;

delete temp;

}

// Разрываем указатель и обнуляем значения количества

head = nullptr;

}

**3.3.4** Сохранение списка в файл

Для удобного хранения и доступа к информации о игроках после закрытия программного средства используется бинарный файл. Код процедуры сохранения в файл PlayerLinkedList::saveToFile представлен ниже.

void PlayerLinkedList::saveToFile() const {

//Создание или перезапись файла для хранения списка игроков

std::ofstream file("playersData.bin", std::ios::binary);

//Получаем указатель на первый элеент списка

player\_t \*temp = head;

//Проходимся по списку игроков

while (temp) {

//Находим размер имени

size\_t nameSize = temp->data.name.size();

//Сохранияем информацию в файл

file.write(reinterpret\_cast<char\*>(&nameSize),sizeof(nameSize));

file.write(temp->data.name.c\_str(), nameSize);

file.write(reinterpret\_cast<char\*>(&temp->data.score), sizeof(temp- >data.score));

temp = temp->next;

}

//Закрытие файла

file.close();

}

Во время разработки игры необходимо корректно использовать аппаратные ресурсы и недопускать утечки памяти.

# 4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Основной проблемой была некорректное отображение снарядов на в игровом окне игроков. Проблема представлена на рисунке 4.1.



Рисунок 4.1 – Некорректное отображение снарядов врагов

Возникала проблема при рассчете интервала времени и врага имеющего возможность выстрела. Проблема была решена дополнительной фунцией рсчета времени в зависимости от нужного врага. Код процедуры расчета соостветсвующего времени.

void AlienClass::tryToShoot(std::vector<RocketClass> &bullets) {

//Проверяем возможность выстрела

if (shootTimer >= shootInterval && canShoot) {

//Создаем снаряд с нужными координатами

bullets.emplace\_back(sprite.getPosition().x + sprite.getGlobalBounds().width / 2, sprite.getPosition().y + sprite.getGlobalBounds().height, false);

//Обнуление счётчика

shootTimer = 0.0f;

//Пересчет нужного интервала для следующего выстрела

shootInterval = 1.0f + static\_cast<float>(rand()) / static\_cast<float>(RAND\_MAX / 2.0f);

}

}

Большинство проблем возникло из-за недочетов на стадии проектирования программного средства, на стадии тестирования приложения все проблемы были исправлены.

**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

**5.1 Интерфейс программного средства**

Внешний вид программных средств играет существенную роль в обеспечении удобства пользователей и качества самого средства.

**5.1.1** Главное меню

Главное меню приложения состоит из четырех основных кнопок:

* первой располагается кнопка, по нажатию которой пользователь переходит в окно игры;
* второй располагается кнопка «Player», позволяющая управлять списком игроков;
* третьей кнопкой является «Score», нажав на которую пользователь может просмотреть таблицу рекордов игроков;
* четвертой является кнопка «Выйти», которая позволяет покинуть программное средство.

Внешний вид главного меню приложения представлен на рисунке 5.1.

****

Рисунок 5.1 – Главное окно приложения

Выход из программного средства можно осуществить нажатием на клавишу «Escape», находясь в главном окне приложения.

5.1.2 Окно «Player»

Окно «Player» содержит управление списком игроков. При открытии этого окна можно добавлять, удалять или выбирать игрока. Окно «Player» представлено на рисунке 5.2.

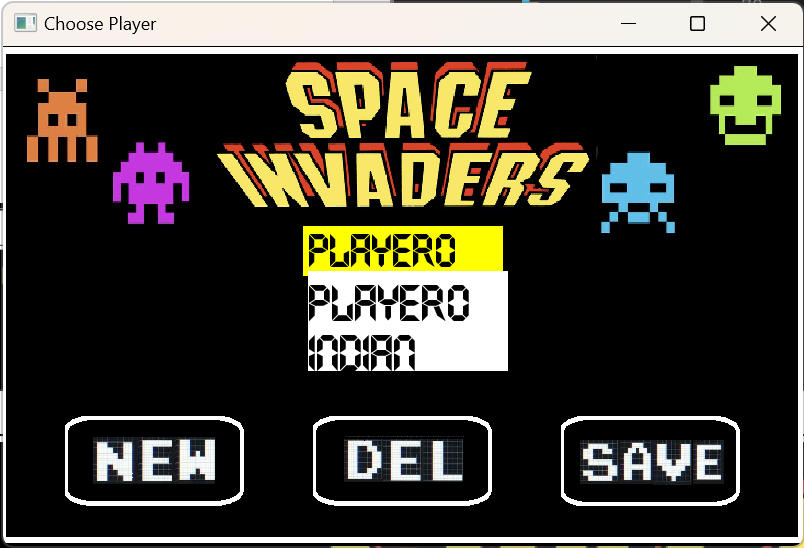


Рисунок 5.2 – Окно «Player»

Для добавления нового игрока, пользователю нужно нажать на кнопку «New». После добавления нового игрока его имя появится в выпадающем списке. При удалении игрока из списка, его информация удалится и в таблице рекордов.

5.1.3 Окно «Score»

Окно «Score» содержит таблицу рекордов игроков. Окно «Score» представлено на рисунке 5.3.



Рисунок 5.3 – Окно «Score»

Когда число игроков в таблице достигает значения большего 10, появляется полоса прокрутки. Новый рекорд заноситься в таблицу сразу после игры.

5.1.4 Окно паузы

Окно паузы состоит из двух основных кнопок:

* первой располагается кнопка, по нажатию которой пользователь возвращается к игре;
* второй кнопкой является «Exit», нажав на которую, пользователь вернется в главное меню.

Окно паузы представлено на рисунке 5.4.



Рисунок 5.4 – Окно паузы

Также для возврата в игру можно нажать клавишу «Escape» на клавиатуре.

5.1.5 Окно окончания игры

Окно окончания игры содержит сообщение о том, что игра закончена. Окно окончания игры представлено на рисунке 5.5.



Рисунок 5.5 – Окно окончания игры

Для подтверждения окончания игры пользователю необходимо нажать клавишу «Enter». После нажатия на «Enter», произойдет закрытие окна окончания игры и переход в главное меню.

5.1.6 Окно игрового процесса

Окно игрового процесса состоит из множества элементов:

* информацию о количестве жизней пушки;
* кнопка паузы;
* враги;
* снаряды врагов;
* пушка;
* снаряд пушки;
* счет игрока.

Окно игрового процесса представлено на рисунке 5.6.

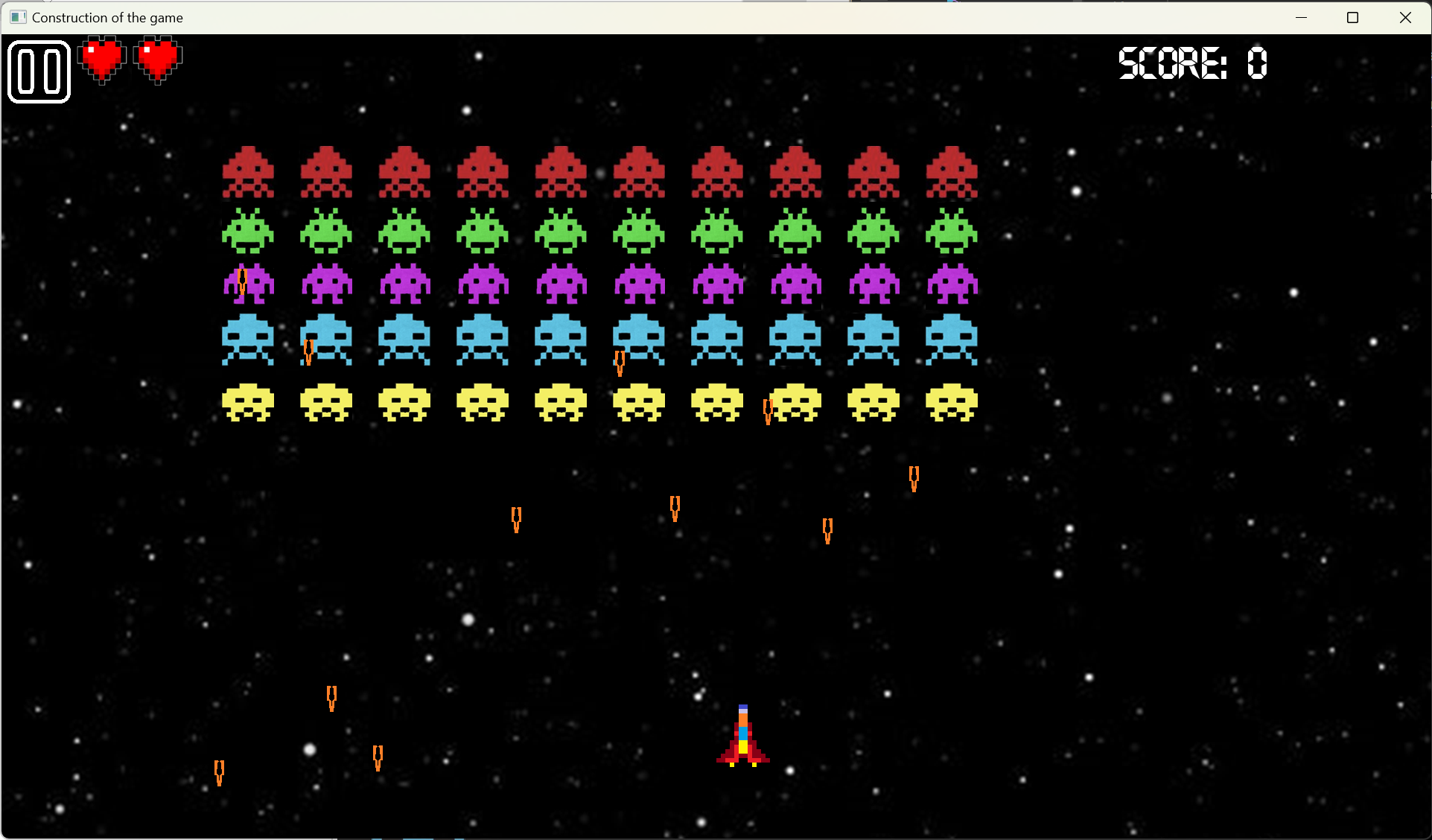


Рисунок 5.6 – Окно игрового процесса

В окне игрового процесса предусмотрен выход в главное меню, с помощью окна паузы. Счет обновляется после уничтожения каждого врага.

# ЗАКЛЮЧЕНИЕ

В современном мире компьютерные игры играют значительную роль в обеспечении отдыха и развлечения людей. В рамках данного курсового проекта было разработано программное средство «Space Invaders», которое предоставляет пользователям операционной системы Windows возможность провести время вместе в атмосфере ретро аркадных игр.

Во время разработки программного средства были успешно выполнены все поставленные задачи:

* работа с внешними устройствами;
* выстрелы пушки;
* механика жизней главной пушки;
* движение главной пушки и врагов;
* создание таблицы с рейтингом игроков.

Реализованы следующие функции:

* подсчет текущего числа очков;
* выстрелы врагов в главную пушку.

Для успешного достижения всех поставленных целей при разработке данного приложения было необходимо углубленно изучить объектно-ориентированные и формовые возможности языка программирования Delphi. Это позволило эффективно использовать функциональность данного языка при создании пользовательского интерфейса, обеспечить визуальное представление данных и взаимодействие с пользователем. Усвоение указанных аспектов языка позволило успешно реализовать все поставленные задачи в рамках разработки приложения.

Однако существуют пути для улучшения данного программного средства. Например, можно добавить звуковое сопровождение, которое поможет создать более приятную атмосферу во время игры. Также можно реализовать разные режимы игры с разными настройками, что добавит разнообразия и дополнительные вызовы в игровой процесс.

В целом «Space Invaders» представляют собой интересное программное средство, которое предоставляет возможность насладиться игрой в атмосфере ретро акрад. За счет разработанных компонентов пользователь может полностью погрузиться в увлекательную игру.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Таненбаум Э., Бос Х. Современные операционные системы. – СПб.: Питер, 2006. – 1104 с. – ISBN 5-94723-200-3.
2. Шилдт Г. C++: полное руководство. – М.: Диалектика-Вильямс, 2003. – 848 с. – ISBN 5-8459-0272-4.
3. Прата С. Язык программирования C++. Лекции и упражнения. – СПб.: БХВ-Петербург, 2008. – 720 с. – ISBN 978-5-9775-0192-7.
4. Страуструп Б. Язык программирования C++. – М.: Бином, 2014. – 496 с. – ISBN 978-5-9905833-1-6.
5. Бут С. Программирование в среде C++ Builder. – СПб.: Питер, 2005. – 624 с. – ISBN 5-94723-991-1.
6. Кузнецов С. Введение в программирование на языке C++ Builder. – М.: ДМК Пресс, 2004. – 320 с. – ISBN 5-94074-435-6.
7. Штерн Р., Симон Д. Мультимедийное программирование на C++ Builder 7. – СПб.: Питер, 2005. – 544 с. – ISBN 5-469-00072-4.

# ПРИЛОЖЕНИЕ А

Текст программы

#include "AlienClass.h"

#include "TextureManagerClass.h"

#include <iostream>

#include <ostream>

#include "RocketClass.h"

AlienClass::AlienClass(float x, float y, AlienColor color, int score\_) : speed(100.0f), direction(1.0f, 0.0f), shootTimer(0.0f), shootInterval(1.0f + static\_cast<float>(rand()) / (RAND\_MAX / 2.0f)) {

std::string texturePath;

switch (color) {

case AlienColor::RED: {

texturePath = "images/RedAlien.png";

canShoot = true;

break;

}

case AlienColor::GREEN: {

texturePath = "images/GreenAlien.png";

canShoot = false;

break;

}

case AlienColor::PURPLE: {

texturePath = "images/PurpleAlien.png";

canShoot = false;

break;

}

case AlienColor::BLUE: {

texturePath = "images/BlueAlien.png";

canShoot = false;

break;

}

case AlienColor::YELLOW: {

texturePath = "images/YellowAlien.png";

canShoot = false;

break;

}

}

try {

sprite.setTexture(TextureManager::getTexture(texturePath));

} catch(const std::exception& e) {

std::cerr << "Error: " << e.what() << std::endl; throw;

}

score = score\_;

sprite.setPosition(x, y);

}

void AlienClass::draw(sf::RenderWindow &window) {

window.draw(sprite);

}

void AlienClass::update(float deltaTime, std::vector<RocketClass> &alienBullets) {

sprite.move(speed \* direction.x \* deltaTime, speed \* direction.y \* deltaTime);

shootTimer += deltaTime;

tryToShoot(alienBullets);

}

sf::FloatRect AlienClass::getBounds() const {

return sprite.getGlobalBounds();

}

void AlienClass::setDirection(float dirX, float dirY) {

direction.x = dirX;

direction.y = dirY;

}

void AlienClass::setSpeed(float newSpeed) {

speed = newSpeed;

}

void AlienClass::setPosition(float x, float y) {

sprite.setPosition(x, y);

}

void AlienClass::tryToShoot(std::vector<RocketClass> &bullets) {

if (shootTimer >= shootInterval && canShoot) {

bullets.emplace\_back(sprite.getPosition().x + sprite.getGlobalBounds().width / 2, sprite.getPosition().y + sprite.getGlobalBounds().height, false);

shootTimer = 0.0f;

shootInterval = 1.0f + static\_cast<float>(rand()) / static\_cast<float>(RAND\_MAX / 2.0f);

}

}

int AlienClass::getScore() {

return score;

}

//

// Created by User on 16.12.2024.

//

#ifndef ALIENCLASS\_H

#define ALIENCLASS\_H

#include <SFML/Graphics/RenderWindow.hpp>

#include <SFML/Graphics/Sprite.hpp>

#include <SFML/Graphics/Texture.hpp>

#include "RocketClass.h"

#define ALIEN\_SHIFT 150

enum class AlienColor { RED, GREEN, PURPLE, BLUE, YELLOW};

class AlienClass {

public:

AlienClass(float x, float y, AlienColor color, int score\_);

void draw(sf::RenderWindow &window);

void update(float deltaTime, std::vector<RocketClass> &alienBullets);

sf::FloatRect getBounds() const;

void setPosition(float x, float y);

void tryToShoot(std::vector<RocketClass> &bullets);

int getScore();

void setDirection(float dirX, float dirY);

void setSpeed(float newSpeed);

private:

bool isKilled;

sf::Sprite sprite;

sf::Texture texture;

float speed;

bool canShoot;

float acceleration;

sf::Vector2f direction;

float shootTimer;

float shootInterval;

int score;

};

#endif //ALIENCLASS\_H

#include "CannonClass.h"

CannonClass::CannonClass(float x, float y) : speed(8.0f) {

cannonTexture.loadFromFile("images\\Cannon.png");

cannonSprite.setTexture(cannonTexture);

cannonSprite.setScale(6, 6);

cannonSprite.setPosition(x, y);

}

void CannonClass::update() {

if (cannonSprite.getPosition().x < 0) {

cannonSprite.setPosition(0, cannonSprite.getPosition().y);

} else if (cannonSprite.getPosition().x + cannonSprite.getGlobalBounds().width > 1900) {

cannonSprite.setPosition(1900 - cannonSprite.getGlobalBounds().width, cannonSprite.getPosition().y);

}

}

void CannonClass::draw(sf::RenderWindow &window) {

window.draw(cannonSprite);

}

void CannonClass::moveLeft() {

cannonSprite.move(-speed, 0);

}

void CannonClass::moveRight() {

cannonSprite.move(speed, 0);

}

sf::FloatRect CannonClass::getBounds() {

return cannonSprite.getGlobalBounds();

}

#ifndef CANNONCLASS\_H

#define CANNONCLASS\_H

#include <SFML/Graphics/RenderWindow.hpp>

#include <SFML/Graphics/Sprite.hpp>

#include <SFML/Graphics/Texture.hpp>

class CannonClass {

public:

CannonClass(float x, float y);

void update();

void draw(sf::RenderWindow &window);

sf::FloatRect getBounds();

void moveLeft();

void moveRight();

private:

sf::Texture cannonTexture;

sf::Sprite cannonSprite;

float speed;

};

#endif //CANNONCLASS\_H

#ifndef CHOOSEPLAYERWINDOW\_H

#define CHOOSEPLAYERWINDOW\_H

#include "fileModule.h"

void choosePlayer();

bool isMouseOverButtonC(const sf::RectangleShape& button, const sf::Vector2i& mousePos);

extern std::string currentPlayerName;

extern PlayerLinkedList::player\_t\* currentPlayer;

#endif //CHOOSEPLAYERWINDOW\_H

#include <iostream>

#include <SFML/Graphics.hpp>

#include <SFML/Window/Event.hpp>

#include <SFML/Window/VideoMode.hpp>

#include "choosePlayerWindow.h"

#include "DropBox.h"

#include "fileModule.h"

#include "newPlayerWindowUnit.h"

#define CHOOSE\_PLAYER\_WINDOW\_WIDTH 800

#define CHOOSE\_PLAYER\_WINDOW\_HEIGHT 500

#define CHOOSE\_PLAYER\_BUTTON\_WIDTH 180

#define CHOOSE\_PLAYER\_BUTTON\_HEIGHT 90

#define CHOOSE\_PLAYER\_SHIFT\_BUTTON 80

PlayerLinkedList::player\_t\* currentPlayer = nullptr;

std::string currentPlayerName = "";

bool isMouseOverButtonC(const sf::RectangleShape& button, const sf::Vector2i& mousePos) {

return button.getGlobalBounds().contains(static\_cast<sf::Vector2f>(mousePos));

}

void choosePlayer () {

sf::RenderWindow chooseWindow(sf::VideoMode(CHOOSE\_PLAYER\_WINDOW\_WIDTH, CHOOSE\_PLAYER\_WINDOW\_HEIGHT), "Choose Player");

sf::Vector2u chooseWindowSize = chooseWindow.getSize();

int currWindowWidth = chooseWindowSize.x;

int currWindowHeight = chooseWindowSize.y;

//BackGround

sf::Texture backgroundChoosePlayerTexture;

if (!backgroundChoosePlayerTexture.loadFromFile("images\\PlayerBackgroundImage.png")) {

std::cerr << "Failed to load background texture" << std::endl;

}

sf::Sprite backgroundChoosePlayerSprite(backgroundChoosePlayerTexture);

sf::Vector2u textureSize = backgroundChoosePlayerTexture.getSize();

float scaleX = static\_cast<float>(chooseWindowSize.x) / textureSize.x;

float scaleY = static\_cast<float>(chooseWindowSize.y) / textureSize.y;

backgroundChoosePlayerSprite.setScale(scaleX, scaleY);

//NewPlayer Button

sf::RectangleShape newPlayerButton(sf::Vector2f(CHOOSE\_PLAYER\_BUTTON\_WIDTH, CHOOSE\_PLAYER\_BUTTON\_HEIGHT));

sf::Texture newPlayerButtonTexture;

if (!newPlayerButtonTexture.loadFromFile("images\\NewPlayerButton.png")) {

std::cerr << "Failed to load scoreButton texture" << std::endl;

}

newPlayerButton.setPosition((currWindowWidth - newPlayerButton.getSize().x) / 2 - CHOOSE\_PLAYER\_SHIFT\_BUTTON \* 3.1f, currWindowHeight / 2 + CHOOSE\_PLAYER\_SHIFT\_BUTTON \* 1.5);

newPlayerButton.setTexture(&newPlayerButtonTexture, true);

//Delete PLayer

sf::RectangleShape deletePlayerButton(sf::Vector2f(CHOOSE\_PLAYER\_BUTTON\_WIDTH, CHOOSE\_PLAYER\_BUTTON\_HEIGHT));

sf::Texture deletePlayerButtonTexture;

if (!deletePlayerButtonTexture.loadFromFile("images\\DelPlayerButton.png")) {

std::cerr << "Failed to load scoreButton texture" << std::endl;

}

deletePlayerButton.setPosition((currWindowWidth - deletePlayerButton.getSize().x) / 2, currWindowHeight / 2 + CHOOSE\_PLAYER\_SHIFT\_BUTTON \* 1.5);

deletePlayerButton.setTexture(&deletePlayerButtonTexture, true);

//Save Button

sf::RectangleShape savePlayerButton(sf::Vector2f(CHOOSE\_PLAYER\_BUTTON\_WIDTH, CHOOSE\_PLAYER\_BUTTON\_HEIGHT));

sf::Texture savePlayerButtonTexture;

if (!savePlayerButtonTexture.loadFromFile("images\\SavePlayerButton.png")) {

std::cerr << "Failed to load scoreButton texture" << std::endl;

}

savePlayerButton.setPosition((currWindowWidth - newPlayerButton.getSize().x) / 2 + CHOOSE\_PLAYER\_SHIFT\_BUTTON \* 3.1f, currWindowHeight / 2 + CHOOSE\_PLAYER\_SHIFT\_BUTTON \* 1.5);

savePlayerButton.setTexture(&savePlayerButtonTexture, true);

//Initialize dropBox

sf::Font font;

if (!font.loadFromFile("fonts\\ofont.ru\_DS Crystal.ttf")) {

std::cerr << "Failed to load font" << std::endl;

}

players.loadFromFile();

DropBox dropBox(font, players.createNameVector(), 300, 180, 200, 50, 4);

while (chooseWindow.isOpen()) {

sf::Event event;

while (chooseWindow.pollEvent(event)) {

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

chooseWindow.close();

}

if (event.type == sf::Event::Resized) {

chooseWindowSize.x = CHOOSE\_PLAYER\_WINDOW\_WIDTH;

chooseWindowSize.y = CHOOSE\_PLAYER\_WINDOW\_HEIGHT;

chooseWindow.setSize(chooseWindowSize);

}

if (event.type == sf::Event::MouseButtonPressed) {

sf::Vector2i mousePos = sf::Mouse::getPosition(chooseWindow);

if (isMouseOverButtonC(newPlayerButton, mousePos)) {

newPlayerWin();

dropBox.update(players.createNameVector());

}

if (isMouseOverButtonC(deletePlayerButton, mousePos) && !dropBox.getSelectedItem().empty()) {

players.remove(dropBox.getSelectedItem());

players.saveToFile();

dropBox.update(players.createNameVector());

dropBox.clearMainField();

}

if (isMouseOverButtonC(savePlayerButton, mousePos) && !dropBox.getSelectedItem().empty()) {

currentPlayer = players.getInfo(dropBox.getSelectedItem());

currentPlayerName = dropBox.getSelectedItem();

chooseWindow.close();

}

}

dropBox.handleEvent(event);

dropBox.update(players.createNameVector());

}

chooseWindow.clear();

chooseWindow.draw(backgroundChoosePlayerSprite);

chooseWindow.draw(newPlayerButton);

if (!dropBox.getSelectedItem().empty()) {

chooseWindow.draw(deletePlayerButton);

chooseWindow.draw(savePlayerButton);

}

dropBox.draw(chooseWindow);

chooseWindow.display();

}

}

#include "DropBox.h"

#include <SFML/Window/Event.hpp>

DropBox::DropBox(sf::Font &fontI, const std::vector<std::string> &names, float x, float y, float width, float height, int visibleCount) : isOpen(false), selectedIndex(-1), scrollOffset(0) {

font = fontI;

mainRect.setSize(sf::Vector2f(width, height));

mainRect.setPosition(x, y);

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

mainText.setOutlineColor(sf::Color::Yellow);

mainText.setFont(font);

mainText.setCharacterSize(46);

mainText.setPosition(x + 5, y - 5);

mainText.setFillColor(sf::Color::Black);

for (const auto &name : names) {

sf::Text text;

text.setFont(font);

text.setString(name);

text.setCharacterSize(50);

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

itemsTexts.push\_back(text);

}

visibleItems = visibleCount;

updateItemsPosition();

}

void DropBox::draw(sf::RenderWindow &window) {

window.draw(mainRect);

window.draw(mainText);

if (isOpen) {

for (const auto &rect : itemsRects) {

window.draw(rect);

}

for (const auto &texts : itemsTexts) {

window.draw(texts);

}

if (selectedIndex >= 0 && selectedIndex < itemsTexts.size()) {

sf::RectangleShape rect(sf::Vector2f(mainRect.getSize().x, mainRect.getSize().y));

rect.setPosition(itemsTexts[selectedIndex].getPosition().x - 5, itemsTexts[selectedIndex].getPosition().y - 5);

rect.setFillColor(sf::Color::Green);

}

}

}

void DropBox::updateItemsPosition() {

itemsRects.clear();

for (size\_t i = 0; i < itemsTexts.size(); i++) {

sf::RectangleShape rect;

rect.setSize(mainRect.getSize());

rect.setPosition(mainText.getPosition().x, mainText.getPosition().y + (i + 1) \* mainRect.getSize().y);

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

rect.setOutlineColor(sf::Color::Yellow);

itemsRects.push\_back(rect);

itemsTexts[i].setPosition(mainRect.getPosition().x + 5, mainRect.getPosition().y + (i + 1) \* mainRect.getSize().y - 5);

}

}

void DropBox::handleEvent(sf::Event &event) {

if (event.type == sf::Event::MouseButtonPressed) {

float mouseX = static\_cast<float>(event.mouseButton.x);

float mouseY = static\_cast<float>(event.mouseButton.y);

if (mainRect.getGlobalBounds().contains(mouseX, mouseY)) {

isOpen = !isOpen;

} else if (isOpen) {

for (size\_t i = 0; i < itemsTexts.size(); ++i) {

if (itemsTexts[i].getGlobalBounds().contains(mouseX, mouseY)) {

selectedIndex = static\_cast<int>(i);

mainRect.setFillColor(sf::Color::Yellow);

mainText.setString(itemsTexts[i].getString());

isOpen = false;

break;

}

}

}

} else if (event.type == sf::Event::KeyPressed) {

if (isOpen) {

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

selectedIndex = std::max(0, selectedIndex - 1);

mainText.setString(itemsTexts[selectedIndex].getString());

updateItemsPosition();

} else if (event.key.code == sf::Keyboard::Down) {

selectedIndex = std::min(static\_cast<int>(itemsTexts.size()) - 1, selectedIndex + 1);

if (selectedIndex >= scrollOffset + visibleItems) {

scrollOffset = std::min(static\_cast<int>(itemsTexts.size()) - visibleItems, scrollOffset);

}

mainText.setString(itemsTexts[selectedIndex].getString());

updateItemsPosition();

} else if (event.key.code == sf::Keyboard::Enter) {

if (selectedIndex >= 0 && selectedIndex < itemsTexts.size()) {

mainRect.setFillColor(sf::Color::Yellow);

mainText.setString(itemsTexts[selectedIndex].getString());

isOpen = false;

}

}

}

}

}

void DropBox::clearMainField() {

mainText.setString("");

}

void DropBox::update(std::vector<std::string> names) {

itemsTexts.clear();

for (const auto &name : names) {

sf::Text text;

text.setFont(font);

text.setString(name);

text.setCharacterSize(50);

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

itemsTexts.push\_back(text);

}

updateItemsPosition();

}

std::string DropBox::getSelectedItem() const {

if (selectedIndex >= 0 && selectedIndex < itemsTexts.size()) {

return itemsTexts[selectedIndex].getString();

}

return "";

}

#ifndef DROPBOX\_H

#define DROPBOX\_H

#include <SFML/Graphics/RectangleShape.hpp>

#include <SFML/Graphics/RenderWindow.hpp>

#include <SFML/Graphics/Text.hpp>

#include "fileModule.h"

class DropBox {

public:

DropBox(sf::Font &font, const std::vector<std::string> &names, float x, float y, float width, float, int visibleCount);

void draw(sf::RenderWindow &window);

void handleEvent(sf::Event &event);

void update(std::vector<std::string> names);

std::string getSelectedItem() const;

void clearMainField();

private:

void updateItemsPosition();

bool isOpen;

int visibleItems;

int selectedIndex;

int scrollOffset;

sf::RectangleShape mainRect;

sf::Text mainText;

std::vector<sf::RectangleShape> itemsRects;

std::vector<sf::Text> itemsTexts;

sf::Font font;

};

#endif //DROPBOX\_H

#include "fileModule.h"

#include <string>

#include <fstream>

#include <vector>

PlayerLinkedList players;

PlayerLinkedList::PlayerLinkedList() : head(nullptr) {}

PlayerLinkedList::~PlayerLinkedList() {

while (head != nullptr) {

player\_t \*tmp = head;

head = head->next;

delete tmp;

}

}

void PlayerLinkedList::add(std::string &name, int score) {

auto \*newPlayer = new player\_t{{name, score},nullptr };

if (head == nullptr) {

head = newPlayer;

} else {

player\_t \*temp = head;

while (temp->next) {

temp = temp->next;

}

temp->next = newPlayer;

}

}

void PlayerLinkedList::remove(const std::string &name) {

if (head == NULL) {

return;

}

if (head->data.name == name) {

player\_t \*temp = head;

head = head->next;

delete temp;

return;

}

player\_t \*temp = head;

while (temp->next && temp->next->data.name != name) {

temp = temp->next;

}

if (temp->next) {

player\_t \*tempRemove = temp->next;

temp->next = temp->next->next;

delete tempRemove;

}

}

void PlayerLinkedList::saveToFile() const {

std::ofstream file("playersData.bin", std::ios::binary);

player\_t \*temp = head;

while (temp) {

size\_t nameSize = temp->data.name.size();

file.write(reinterpret\_cast<char\*>(&nameSize),sizeof(nameSize));

file.write(temp->data.name.c\_str(), nameSize);

file.write(reinterpret\_cast<char\*>(&temp->data.score), sizeof(temp->data.score));

temp = temp->next;

}

file.close();

}

void PlayerLinkedList::loadFromFile() {

clear();

std::ifstream file("playersData.bin", std::ios::binary);

while (file) {

size\_t nameSize;

file.read(reinterpret\_cast<char\*>(&nameSize), sizeof(nameSize));

if (file.eof()) {

break;

}

std::string name(nameSize, ' ');

file.read(&name[0], nameSize);

int score;

file.read(reinterpret\_cast<char\*>(&score), sizeof(score));

add(name, score);

}

file.close();

}

std::vector<std::string> PlayerLinkedList::createNameVector() const {

player\_t \*temp = head;

std::vector <std::string> nameVector;

while (temp) {

nameVector.push\_back(temp->data.name);

temp = temp->next;

}

return nameVector;

}

int PlayerLinkedList::getCount() const {

player\_t \*temp = head;

int count = 0;

while (temp) {

count++;

temp = temp->next;

}

return count;

}

void PlayerLinkedList::updateScore(std::string &name, int score) const {

player\_t \*temp = head;

while (temp->data.name != name && temp) {

temp = temp->next;

}

temp->data.score = score;

}

void PlayerLinkedList::clear() {

while (head) {

player\_t \*temp = head;

head = head->next;

delete temp;

}

head = nullptr;

}

PlayerLinkedList::player\_t\* PlayerLinkedList::getInfo(std::string name) {

player\_t \*temp = head;

while (temp && temp->data.name != name) {

temp = temp->next;

}

return temp;

}

//

// Created by User on 09.12.2024.

//

#ifndef FILEMODULE\_H

#define FILEMODULE\_H

#include <string>

#include <vector>

class PlayerLinkedList {

public:

struct player\_t {

struct playerData\_t {

std::string name;

int score;

};

playerData\_t data;

player\_t \*next;

};

player\_t \*head;

PlayerLinkedList();

~PlayerLinkedList();

void add(std::string &name, int score);

void remove(const std::string &name);

void saveToFile() const;

void loadFromFile();

std::vector<std::string> createNameVector() const;

int getCount() const;

void updateScore(std::string &name, int score) const;

void clear();

player\_t\* getInfo(std::string name);

};

extern PlayerLinkedList players;

#endif //FILEMODULE\_H

#include "gameOverWindowUnit.h"

#include <SFML/Graphics/RenderWindow.hpp>

#include <SFML/Graphics/Sprite.hpp>

#include <SFML/Graphics/Texture.hpp>

#include "TextureManagerClass.h"

#define GAME\_OVER\_WINDOW\_WIDTH 800

#define GAME\_OVER\_WINDOW\_HEIGHT 600

void gameOver() {

sf::RenderWindow gameOverWindow(sf::VideoMode(GAME\_OVER\_WINDOW\_WIDTH, GAME\_OVER\_WINDOW\_HEIGHT), "GameOver");

sf::Sprite backgroundSprite;

sf::Vector2u textureSize = TextureManager::getTexture("images\\GameOver.png").getSize();

sf::Vector2u windowSize = gameOverWindow.getSize();

float scaleX = static\_cast<float>(windowSize.x) / textureSize.x;

float scaleY = static\_cast<float>(windowSize.y) / textureSize.y;

backgroundSprite.setScale(scaleX, scaleY);

backgroundSprite.setTexture(TextureManager::getTexture("images\\GameOver.png"));

while (gameOverWindow.isOpen()) {

sf::Event event;

while (gameOverWindow.pollEvent(event)) {

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

gameOverWindow.close();

}

if (event.type == sf::Event::Resized) {

sf::Vector2u windowSize;

windowSize.x = GAME\_OVER\_WINDOW\_WIDTH;

windowSize.y = GAME\_OVER\_WINDOW\_HEIGHT;

gameOverWindow.setSize(windowSize);

}

if (event.type == sf::Event::KeyPressed) {

if (event.key.code == sf::Keyboard::Enter || event.key.code == sf::Keyboard::Escape) {

gameOverWindow.close();

}

}

}

gameOverWindow.clear();

gameOverWindow.draw(backgroundSprite);

gameOverWindow.display();

}

}

#ifndef GAMEOVERWINDOWUNIT\_H

#define GAMEOVERWINDOWUNIT\_H

void gameOver();

#endif //GAMEOVERWINDOWUNIT\_H

#include <SFML/Graphics.hpp>

#include <iostream>

#include "gameWindowUnit.h"

#include "choosePlayerWindow.h"

#include "RocketClass.h"

#include "CannonClass.h"

#include "AlienClass.h"

#include "TextureManagerClass.h"

#include "gameOverWindowUnit.h"

#include "pauseWindowUnit.h"

#define GAME\_WINDOW\_WIDTH 1920

#define GAME\_WINDOW\_HEIGHT 1080

#define PAUSE\_BUTTON\_WIDTH 100

void alienInitialization(std::vector<AlienClass>& aliens, float &alienSpeed, float &alienAcceleration, float &alienDirection) {

alienSpeed = 100.0f;

alienAcceleration = 10.0f;

alienDirection = 1.0f;

for (int i = 0; i < 10; i++) {

aliens.emplace\_back(ALIEN\_SHIFT \* i \* 0.7, ALIEN\_SHIFT, AlienColor::RED, 50);

aliens.emplace\_back(ALIEN\_SHIFT \* i \* 0.7, ALIEN\_SHIFT \* 1.5, AlienColor::GREEN, 40);

aliens.emplace\_back(ALIEN\_SHIFT \* i \* 0.7, ALIEN\_SHIFT \* 2, AlienColor::PURPLE, 30);

aliens.emplace\_back(ALIEN\_SHIFT \* i \* 0.7, ALIEN\_SHIFT \* 2.5, AlienColor::BLUE, 20);

aliens.emplace\_back(ALIEN\_SHIFT \* i \* 0.7, ALIEN\_SHIFT \* 3, AlienColor::YELLOW, 10);

}

}

void scoreInitialization(int &score, sf::Text &scoreText, sf::Font &scoreFont) {

score = 0;

scoreFont.loadFromFile("fonts\\ofont.ru\_DS Crystal.ttf");

scoreText.setFont(scoreFont);

scoreText.setString("Score: " + std::to\_string(score));

scoreText.setCharacterSize(60);

scoreText.setColor(sf::Color::White);

scoreText.setPosition(1500, 0);

}

void createButton(sf::RectangleShape &button, const float posX, const float posY, const std::string &texturePath) {

button.setPosition(posX, posY);

button.setTexture(&TextureManager::getTexture(texturePath), true);

}

void game() {

sf::RenderWindow gameWindow(sf::VideoMode(GAME\_WINDOW\_WIDTH, GAME\_WINDOW\_HEIGHT), "Construction of the game");

sf::Vector2u windowSize = gameWindow.getSize();

gameWindow.setFramerateLimit(120);

sf::Texture backgroundTexture;

if (!backgroundTexture.loadFromFile("images\\BackGroundImage.png")) {

std::cerr << "Failed to load background texture" << std::endl;

}

sf::Sprite backgroundSprite(backgroundTexture);

//Lives

sf::Texture heartTexture;

int lives = 3;

std::vector<sf::Sprite> hearts;

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

sf::Sprite heartSprite(TextureManager::getTexture("images\\Heart.png"));

heartSprite.setPosition(75 \* i + 100, 0);

hearts.push\_back(heartSprite);

}

//Score

int score;

sf::Text scoreText;

sf::Font font;

scoreInitialization(score, scoreText, font);

//Pause Button

sf::RectangleShape pauseButton(sf::Vector2f(PAUSE\_BUTTON\_WIDTH, PAUSE\_BUTTON\_WIDTH));

createButton(pauseButton, 0, 0, "images\\PauseButton.png");

//Rocket & Cannon initialization

CannonClass cannon(windowSize.x / 2, 900);

std::vector<RocketClass> rockets;

// Alien initialization

std::vector<RocketClass> alienBullet;

std::vector<AlienClass> aliens;

float alienSpeed;

float alienAcceleration;

float alienDirection;

alienInitialization(aliens, alienSpeed, alienAcceleration, alienDirection);

//Main Cycle

sf::Clock clock;

while (gameWindow.isOpen()) {

sf::Event event{};

while (gameWindow.pollEvent(event)) {

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

gameWindow.close();

}

if (event.type == sf::Event::Resized) {

windowSize.x = GAME\_WINDOW\_WIDTH;

windowSize.y = GAME\_WINDOW\_HEIGHT;

gameWindow.setSize(windowSize);

}

if (event.type == sf::Event::KeyPressed) {

if (event.key.code == sf::Keyboard::Space) {

if (rockets.empty()) {

rockets.emplace\_back(cannon.getBounds().left + cannon.getBounds().width / 2, cannon.getBounds().top, true);

}

}

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

pause(gameWindow);

clock.restart();

}

if (event.type == sf::Event::MouseButtonPressed) {

sf::Vector2i mousePos = sf::Mouse::getPosition(gameWindow);

if (isMouseOverButtonC(pauseButton, mousePos)) {

pause(gameWindow);

clock.restart();

}

}

}

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {

cannon.moveLeft();

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {

cannon.moveRight();

}

float deltaTime = clock.restart().asSeconds();

cannon.update();

for (auto& rocket : rockets) {

rocket.update(true);

}

for (auto& alien : aliens) {

alien.update(deltaTime, alienBullet);

}

for (auto& alienBullets : alienBullet) {

alienBullets.update(false);

}

//Check edges

bool hitEdge = false;

for (const auto& alien : aliens) {

if (alien.getBounds().left <= 0 || alien.getBounds().left + alien.getBounds().width >= gameWindow.getSize().x) {

hitEdge = true;

break;

}

}

if (hitEdge) {

for (auto& alien : aliens) {

alien.setDirection(alienDirection \* -1.0f, 0.0f);

}

for (auto& alien : aliens) {

alien.setPosition(alien.getBounds().left, alien.getBounds().top + 50.0f);

}

alienSpeed += alienAcceleration;

alienDirection \*= -1.0f;

for (auto& alien : aliens) {

alien.setSpeed(alienSpeed);

}

}

//

//Delete rockets (bounds)

rockets.erase(std::remove\_if(rockets.begin(), rockets.end(), [](RocketClass& rocket){

return rocket.getBounds().top < 0;

}), rockets.end());

//Delete alien Bullets (bounds)

alienBullet.erase(std::remove\_if(alienBullet.begin(), alienBullet.end(), [&gameWindow](RocketClass& alien){

return alien.getBounds().top > gameWindow.getSize().y;

}), alienBullet.end());

//Delete alien bullets (cannon)

for (auto itAlienRocket = alienBullet.begin(); itAlienRocket != alienBullet.end(); ){

if (itAlienRocket->getBounds().intersects(cannon.getBounds())) {

lives--;

if (lives == 0) {

gameWindow.close();

if (currentPlayer->data.score < score) {

currentPlayer->data.score = score;

}

gameOver();

std::cout << "score "<< currentPlayer->data.score << "\n" << currentPlayer->data.name << "\n";

}

hearts.pop\_back();

//explosionSound.play();

//explosions.emplace\_back(cannon.getBounds().left, cannon.getBounds().top, explosionTexture);

itAlienRocket = alienBullet.erase(itAlienRocket); // Удаление пули

} else {

++itAlienRocket;

}

}

alienBullet.erase(std::remove\_if(alienBullet.begin(), alienBullet.end(), [&cannon, lives](RocketClass& alien) {

if (alien.getBounds().intersects(cannon.getBounds())) {

// explosionSound.play();

// explosions.emplace\_back(cannon.getBounds().left, cannon.getBounds().top, explosionTexture);

return true;

} return false;

}),

alienBullet.end());

//Deleting rockets (aliens)

for (auto itRocket = rockets.begin(); itRocket != rockets.end(); ) {

bool rocketRemoved = false;

for (auto itAlien = aliens.begin(); itAlien != aliens.end(); ) {

if (itRocket->getBounds().intersects(itAlien->getBounds())) {

itAlien = aliens.erase(itAlien);

itRocket = rockets.erase(itRocket);

rocketRemoved = true;

score += itAlien->getScore();

scoreText.setString("Score: " + std::to\_string(score));

if (aliens.empty()) {

alienInitialization(aliens, alienSpeed, alienAcceleration, alienDirection);

}

break;

} else {

++itAlien;

}

}

if (!rocketRemoved) {

++itRocket;

}

}

gameWindow.clear();

gameWindow.draw(backgroundSprite);

cannon.draw(gameWindow);

for (auto& rocket : rockets) {

rocket.draw(gameWindow);

}

for (auto& alien : aliens) {

alien.draw(gameWindow);

}

for (auto& alien : alienBullet) {

alien.draw(gameWindow);

}

for (auto& heart : hearts) {

gameWindow.draw(heart);

}

gameWindow.draw(scoreText);

gameWindow.draw(pauseButton);

gameWindow.display();

}

}

#include <SFML/Graphics.hpp>

#ifndef GAMEWINDOWUNIT\_H

#define GAMEWINDOWUNIT\_H

void game();

void createButton(sf::RectangleShape &button, const float posX, const float posY, const std::string &texturePath);

#endif //GAMEWINDOWUNIT\_H

#include <SFML/Graphics.hpp>

#include <iostream>

#include "choosePlayerWindow.h"

#include "fileModule.h"

#include "gameWindowUnit.h"

#include "scoreBoardWindowUnit.h"

#define WINDOW\_WIDTH 1920

#define WINDOW\_HEIGHT 1080

#define BUTTON\_WIDTH 350

#define BUTTON\_HEIGHT 120

#define SHIFT\_BUTTON 150

// Функция для проверки, попадает ли мышь на кнопку

bool isMouseOverButton(const sf::RectangleShape& button, const sf::Vector2i& mousePos) {

return button.getGlobalBounds().contains(static\_cast<sf::Vector2f>(mousePos));

}

int main() {

sf::RenderWindow window(sf::VideoMode(WINDOW\_WIDTH, WINDOW\_HEIGHT), "Main Menu Example");

sf::Vector2u windowSize = window.getSize();

//Создание фонового изображения

sf::Texture backgroundTexture;

if (!backgroundTexture.loadFromFile("images\\StartMenuBackground.png")) {

std::cerr << "Failed to load background texture" << std::endl;

}

sf::Sprite backgroundSprite(backgroundTexture);

sf::Vector2u textureSize = backgroundTexture.getSize();

float scaleX = static\_cast<float>(windowSize.x) / textureSize.x;

float scaleY = static\_cast<float>(windowSize.y) / textureSize.y;

backgroundSprite.setScale(scaleX, scaleY);

// Создание кнопок

//Player Button

sf::RectangleShape playerButton(sf::Vector2f(BUTTON\_WIDTH, BUTTON\_HEIGHT));

createButton(playerButton, (windowSize.x - playerButton.getSize().x) / 2, windowSize.y / 2, "images\\playerButton.png");

//Start button

sf::RectangleShape startButton(sf::Vector2f(BUTTON\_WIDTH, BUTTON\_HEIGHT));

createButton(startButton,(windowSize.x - startButton.getSize().x) / 2, windowSize.y / 2 - SHIFT\_BUTTON, "images\\playButton.png");

//Score Button

sf::RectangleShape scoreButton(sf::Vector2f(BUTTON\_WIDTH, BUTTON\_HEIGHT));

createButton(scoreButton, (windowSize.x - scoreButton.getSize().x) / 2, windowSize.y / 2 + SHIFT\_BUTTON, "images\\scoreButton.png");

//Exit Button

sf::RectangleShape exitButton(sf::Vector2f(150, 150));

createButton(exitButton, (windowSize.x - exitButton.getSize().x) / 2 + SHIFT\_BUTTON \* 2, windowSize.y / 2 + SHIFT\_BUTTON,"images\\exitButton.png");

PlayerLinkedList::player\_t\* player = currentPlayer;

// Main cycle

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

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

window.close();

}

else if (event.type == sf::Event::Resized) {

windowSize.x = WINDOW\_WIDTH;

windowSize.y = WINDOW\_HEIGHT;

window.setSize(windowSize);

}

if (event.type == sf::Event::MouseButtonPressed) {

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

if (isMouseOverButton(startButton, mousePos)) {

currentPlayer = players.getInfo(currentPlayerName);

game();

players.saveToFile();

}

if (isMouseOverButton(exitButton, mousePos)) {

players.saveToFile();

window.close();

//Save to file

}

if (isMouseOverButton(playerButton, mousePos)) {

choosePlayer();

}

if (isMouseOverButton(scoreButton, mousePos)) {

players.loadFromFile();

scoreBoard(players);

}

}

window.clear();

window.draw(backgroundSprite);

if (currentPlayer) {

window.draw(startButton);

}

window.draw(playerButton);

window.draw(scoreButton);

window.draw(exitButton);

window.display();

}

}

return 0;

}

#include <SFML/Graphics.hpp>

#include "newPlayerWindowUnit.h"

#include <iostream>

#include "fileModule.h"

#define NEW\_PLAYER\_WINDOW\_WIDTH 650

#define NEW\_PLAYER\_WINDOW\_HEIGHT 400

#define NEW\_PLAYER\_OK\_BUTTON\_WIDTH 140

#define NEW\_PLAYER\_OK\_BUTTON\_HEIGHT 80

bool isMouseOverButtonOk(const sf::RectangleShape& button, const sf::Vector2i& mousePos) {

return button.getGlobalBounds().contains(static\_cast<sf::Vector2f>(mousePos));

}

void newPlayerWin() {

sf::RenderWindow newPlayerWindow(sf::VideoMode (NEW\_PLAYER\_WINDOW\_WIDTH, NEW\_PLAYER\_WINDOW\_HEIGHT ), "New Player");

int currWindowWidth = newPlayerWindow.getSize().x;

int currWindowHeight = newPlayerWindow.getSize().y;

//background image

sf::Vector2u newPlayerWindowSize = newPlayerWindow.getSize();

sf::Texture backgroundNewPlayerTexture;

if (!backgroundNewPlayerTexture.loadFromFile("images\\AddNewPlayerBackground.png")) {

std::cerr << "Failed to load background texture" << std::endl;

}

sf::Sprite backgroundNewPlayerSprite(backgroundNewPlayerTexture);

sf::Vector2u textureSize = backgroundNewPlayerTexture.getSize();

float scaleX = static\_cast<float>(newPlayerWindowSize.x) / textureSize.x;

float scaleY = static\_cast<float>(newPlayerWindowSize.y) / textureSize.y;

backgroundNewPlayerSprite.setScale(scaleX, scaleY);

//input Edit

sf::Font font;

if (!font.loadFromFile("fonts\\ofont.ru\_DS Crystal.ttf")) {

std::cerr << "Failed to load font" << std::endl;

}

sf::Text newNameEdit;

newNameEdit.setFont(font);

newNameEdit.setCharacterSize(65);

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

newNameEdit.setPosition(220, 190);

std::string newNameStr;

//ok button

sf::RectangleShape okButton(sf::Vector2f(NEW\_PLAYER\_OK\_BUTTON\_WIDTH, NEW\_PLAYER\_OK\_BUTTON\_HEIGHT));

sf::Texture okButtonTexture;

if (!okButtonTexture.loadFromFile("images\\OkButton.png")) {

std::cerr << "Failed to load scoreButton texture" << std::endl;

}

okButton.setPosition((currWindowWidth - okButton.getSize().x) / 2 + 200, currWindowHeight / 2 + 100);

okButton.setTexture(&okButtonTexture, true);

//main cycle

while (newPlayerWindow.isOpen()) {

sf::Event event;

while (newPlayerWindow.pollEvent(event)) {

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

newPlayerWindow.close();

}

if (event.type == sf::Event::Resized) {

sf::Vector2u windowSize;

windowSize.x = NEW\_PLAYER\_WINDOW\_WIDTH;

windowSize.y = NEW\_PLAYER\_WINDOW\_HEIGHT;

newPlayerWindow.setSize(windowSize);

}

if (event.type == sf::Event::TextEntered) {

if (event.text.unicode == '\b' && !newNameStr.empty()) {

newNameStr.pop\_back();

} else if (event.text.unicode < 128 && event.text.unicode != '\b') {

newNameStr += static\_cast<char>(event.text.unicode);

}

newNameEdit.setString(newNameStr);

}

if (event.type == sf::Event::MouseButtonPressed) {

sf::Vector2i mousePos = sf::Mouse::getPosition(newPlayerWindow);

if (isMouseOverButtonOk(okButton, mousePos) && !newNameStr.empty()) {

//add new player

players.add(newNameStr, 0);

players.saveToFile();

newPlayerWindow.close();

}

}

}

newPlayerWindow.clear();

newPlayerWindow.draw(backgroundNewPlayerSprite);

newPlayerWindow.draw(newNameEdit);

if (!newNameStr.empty()){

newPlayerWindow.draw(okButton);

}

newPlayerWindow.display();

}

}

#ifndef NEWPLAYERWINDOWUNIT\_H

#define NEWPLAYERWINDOWUNIT\_H

void newPlayerWin();

#endif //NEWPLAYERWINDOWUNIT\_H

#include "pauseWindowUnit.h"

#include "gameWindowUnit.h"

#include <SFML/Graphics/RenderWindow.hpp>

#include <SFML/Graphics/Texture.hpp>

#include <iostream>

#include <SFML/Graphics/RectangleShape.hpp>

#include <SFML/Graphics/Sprite.hpp>

#include <SFML/Window/Event.hpp>

#define PAUSE\_WINDOW\_WIDTH 700

#define PAUSE\_WINDOW\_HEIGHT 600

#define BUTTON\_WIDTH 250

#define BUTTON\_HEIGHT 125

bool isMouseOverButtonP(const sf::RectangleShape& button, const sf::Vector2i& mousePos) {

return button.getGlobalBounds().contains(static\_cast<sf::Vector2f>(mousePos));

}

void pause(sf::RenderWindow &gameWindow) {

sf::RenderWindow window(sf::VideoMode(PAUSE\_WINDOW\_WIDTH, PAUSE\_WINDOW\_HEIGHT), "Main Menu Example");

sf::Vector2u windowSize = window.getSize();

sf::Texture backgroundTexture;

if (!backgroundTexture.loadFromFile("images\\PauseBackGround.png")) {

std::cerr << "Failed to load background texture" << std::endl;

}

sf::Sprite backgroundSprite(backgroundTexture);

sf::Vector2u textureSize = backgroundTexture.getSize();

float scaleX = static\_cast<float>(windowSize.x) / textureSize.x;

float scaleY = static\_cast<float>(windowSize.y) / textureSize.y;

backgroundSprite.setScale(scaleX, scaleY);

sf::RectangleShape resumeButton(sf::Vector2f(BUTTON\_WIDTH, BUTTON\_HEIGHT));

createButton(resumeButton, 50, 430, "images\\ResumeButton.png");

sf::RectangleShape exitButton(sf::Vector2f(BUTTON\_WIDTH, BUTTON\_HEIGHT));

createButton(exitButton, windowSize.x - exitButton.getSize().x - 50, 430, "images\\ExitFromPauseButton.png");

while (window.isOpen()) {

sf::Event event{};

while (window.pollEvent(event)) {

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

window.close();

}

if (event.type == sf::Event::Resized) {

sf::Vector2u windowSize;

windowSize.x = PAUSE\_WINDOW\_WIDTH;

windowSize.y = PAUSE\_WINDOW\_HEIGHT;

window.setSize(windowSize);

}

if (event.type == sf::Event::KeyPressed) {

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

window.close();

}

}

if (event.type == sf::Event::MouseButtonPressed) {

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

if (isMouseOverButtonP(resumeButton, mousePos)) {

window.close();

}

if (isMouseOverButtonP(exitButton, mousePos)) {

window.close();

gameWindow.close();

}

// newPlayerWin();

// dropBox.update(players.createNameVector());

}

window.clear();

window.draw(backgroundSprite);

window.draw(resumeButton);

window.draw(exitButton);

window.display();

}

}

}

#ifndef PAUSEWINDOWUNIT\_H

#define PAUSEWINDOWUNIT\_H

#include <SFML/Graphics/RenderWindow.hpp>

void pause(sf::RenderWindow& gameWindow);

#endif //PAUSEWINDOWUNIT\_H

#include "RocketClass.h"

#include "TextureManagerClass.h"

RocketClass::RocketClass(float startX, float startY, bool cannon) : speed(5.0f) {

if (cannon) {

rocketSprite.setTexture(TextureManager::getTexture("images\\BulletRocket.png"));

rocketSprite.setScale(4, 4);

speed = 8.0f;

} else {

rocketSprite.setTexture(TextureManager::getTexture("images\\AlienBullet.png"));

}

rocketSprite.setPosition(startX, startY);

}

void RocketClass::draw(sf::RenderWindow &window) {

window.draw(rocketSprite);

}

void RocketClass::update(bool cannon) {

if (cannon) {

rocketSprite.move(0, -speed);

} else {

rocketSprite.move(0, speed);

}

}

sf::FloatRect RocketClass::getBounds() {

return rocketSprite.getGlobalBounds();

}

#ifndef ROCKETCLASS\_H

#define ROCKETCLASS\_H

#include <SFML\Graphics.hpp>

class RocketClass {

public:

RocketClass(float startX, float startY, bool cannon);

sf::FloatRect getBounds();

void draw(sf::RenderWindow &window);

void update(bool cannon);

private:

float speed;

sf::Texture rocketTexture;

sf::Sprite rocketSprite;

};

#endif //ROCKETCLASS\_H

#include "scoreBoardWindowUnit.h"

#include <SFML/Graphics/RenderWindow.hpp>

#include <SFML/System.hpp>

#include "fileModule.h"

#include "TextureManagerClass.h"

#define SCORE\_BOARD\_WINDOW\_WIDTH 800

#define SCORE\_BOARD\_WINDOW\_HEIGHT 900

void createScoreVector(std::vector<sf::Text> &scoreList, sf::Font &font, PlayerLinkedList &player) {

font.loadFromFile("fonts\\ofont.ru\_DS Crystal.ttf");

PlayerLinkedList::player\_t \*currPlayer = player.head;

while (currPlayer != nullptr) {

sf::Text scoreText;

scoreText.setFont(font);

scoreText.setCharacterSize(50);

scoreText.setString(currPlayer->data.name + ": " + std::to\_string(currPlayer->data.score));

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

scoreList.push\_back(scoreText);

currPlayer = currPlayer->next;

}

float startY = 330.0f;

float spacing = 60.0f;

for (size\_t i = 0; i < scoreList.size(); i++) {

scoreList[i].setPosition(sf::Vector2f(300, startY + i \* spacing));

}

}

void scoreBoard(PlayerLinkedList &players) {

sf::RenderWindow scoreBoardWindow(sf::VideoMode(SCORE\_BOARD\_WINDOW\_WIDTH, SCORE\_BOARD\_WINDOW\_HEIGHT), "ScoreBoard");

sf::Sprite backgroundSprite;

sf::Vector2u textureSize = TextureManager::getTexture("images\\ScoreBackGround.png").getSize();

sf::Vector2u windowSize = scoreBoardWindow.getSize();

float scaleX = static\_cast<float>(windowSize.x) / textureSize.x;

float scaleY = static\_cast<float>(windowSize.y) / textureSize.y;

backgroundSprite.setScale(scaleX, scaleY);

backgroundSprite.setTexture(TextureManager::getTexture("images\\ScoreBackGround.png"));

std::vector<sf::Text> scoreList;

sf::Font font;

createScoreVector(scoreList, font, players);

while (scoreBoardWindow.isOpen()) {

sf::Event event;

while (scoreBoardWindow.pollEvent(event)) {

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

scoreList.clear();

scoreBoardWindow.close();

}

if (event.type == sf::Event::Resized) {

sf::Vector2u windowSize;

windowSize.x = SCORE\_BOARD\_WINDOW\_WIDTH;

windowSize.y = SCORE\_BOARD\_WINDOW\_HEIGHT;

scoreBoardWindow.setSize(windowSize);

}

}

scoreBoardWindow.clear();

scoreBoardWindow.draw(backgroundSprite);

for (auto &scoreText : scoreList) {

scoreBoardWindow.draw(scoreText);

}

scoreBoardWindow.display();

}

}

#ifndef SCOREBOARDWINDOWUNIT\_H

#define SCOREBOARDWINDOWUNIT\_H

#include "fileModule.h";

void scoreBoard(PlayerLinkedList &players);

#endif //SCOREBOARDWINDOWUNIT\_H

#ifndef TEXTUREMANAGERCLASS\_H

#define TEXTUREMANAGERCLASS\_H

#include <SFML/Graphics.hpp>

#include <unordered\_map>

#include <string>

class TextureManager {

public:

static sf::Texture& getTexture(const std::string& filename) {

auto& textures = getInstance().textures;

auto it = textures.find(filename);

if (it == textures.end()) {

sf::Texture texture;

if (!texture.loadFromFile(filename)) {

throw std::runtime\_error("Failed to load texture: " + filename);

}

it = textures.emplace(filename, std::move(texture)).first;

}

return it->second;

}

private:

TextureManager() = default;

static TextureManager& getInstance() {

static TextureManager instance;

return instance;

}

std::unordered\_map<std::string, sf::Texture> textures;

};

#endif // TEXTUREMANAGERCLASS\_H