Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра «Вычислительная техника и системы управления»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по дисциплине

«Языки программирования»

Тема: «Kosmolet»

Выполнил:

студент группы ВТ-122

Киселев А.В.

Проверил:

Шарафеддин М. А.

Владимир 2023

Содержание

Введение 3

1 Формулировка задачи 4

2 Проектирование программы 5

3 Код программы на языке *С++* 11

4 Результаты тестирования 22

Заключение 24

Введение

В данной курсовой работе было необходимо разработать симуляцию игры «Kosmolet».

В наше время игры давно стали частью жизни для многих. Поэтому их разработка так же важна, как съёмка кино или написание книг. Исходя из этого и была выбрана данная тема, как шанс посмотреть на создание таких цифровых развлечений.

Для написания программы был выбран объектно-ориентированный подход. В связи со своими особенностями объектно-ориентированное программирование имеет ряд преимуществ перед структурным программированием: использование одного и того же программного кода с разными данными. Классы позволяют создавать множество объектов, каждый из которых имеет собственные значения атрибутов. Объект можно представить как некий контейнер данных.

Цель работы.

Научится практически использовать навыки написания программ на языке программирования С++, которые будут основываться на принципах объектно-ориентированного программирования: разделение программы на классы и методы.

Показать практическую пользу применения объектно-ориентированного программирования на практике, удобность использования таких принципов как: полиморфизм, наследование и инкапсуляция.

1 Формулировка задачи

Тема проекта:

Реализация игры «Kosmolet». Данный проект представляет из себя игру, где мы управляем дроном белого цвета, находящимся в космосе, который, в свою очередь, сбивает вражеские дроны, которые имеют в тенденции красный цвет. С каждым сбитым вражеским дроном рекорд игрока увеличивается, а враги повышают скорость своего движения, в связи с этим, игра начинает усложнятся.

* 1. Анализ

Для реализации игры необходимо:

Создать класс для вооружения, создать для него форму и загрузить текстуру.

Создать класс для игрока(космического корабля), создать текстуру для него, настроить движение игрока, стрельбу (т.е. движение снаряда), сделать определенное количество «жизней», сделать накапливаемую статистику уничтоженных врагов за игру (т.е. рекорд).

Создать класс для врага (вражеский дрон), создать текстуру для него, настроить движение врага, сделать определенное количество «жизней», воспроизвести столкновение врага с игроком, в результате которого у игрока будут уменьшаться жизни, настроить скорость движения врагов, а также ее повышение

В интерфейсе воспроизвести методы для работы главного меню, имеющее кнопки: *play, help, exit;* меню для рестарта игра; воспроизвести классы и методы для отрисовки объектов программы.

Моей задачей будет реализовать работу и отношения данных классов при помощи графической библиотеки SFML.

2 Проектирование программы

2.1 Спецификации классов

В таблице 1 приведены поля и методы класса Textures.

Таблица 1

|  |  |  |
| --- | --- | --- |
| Поле / Метод | Возвращаемый тип | Использование |
| +playerTex | Texture | Текстура игрока |
| +enemyTex | Texture | Текстура врага |
| +bulletTex | Texture | Текстура пули |
| +loadTex | void | Загрузка текстуры |
| +createBackground | Sprite | Текстура заднего фона |

В таблице 2 приведены поля и методы класса Bullet.

Таблица 2

|  |  |  |
| --- | --- | --- |
| Поле / Метод | Возвращаемый тип | Использование |
| +shape | Sprite | Хранение текстуры пули |

В таблице 3 приведены поля и методы класса Player.

Таблица 3

|  |  |  |
| --- | --- | --- |
| Поле / Метод | Возвращаемый тип | Использование |
| +bulletTex | Bullet | Хранение текстуры пули |
| +hpText | Text | Объект жизней игрока |
| +shape | Sprite | Объект модели игрока |
| +texture | Texture | Объект текстуры игрока |
| +playerTex | Texture | Картинка игрока |
| +font | Font | Шрифт текстовых объектов игрока |
| +HP | int | Текущее значение жизни игрока |
| +HPmax | int | Максимальное значение жизней игрока |
| +score | int | Статистика, убитых игроком врагов |
| +shootTimer | int | Временной интервал между двумя выстрелами игрока (Частота стрельбы) |
| +bullets | vector | Множество пуль, которые выпускает игрок |
| +shoot | void | Функция выпуска пули игроком |
| +PlayerControl | void | Функция отвечающая за клавиши управления игроком |
| +handleWindowCollision | void | Отвечает за границы игрового поля, чтобы игрок их не пересекал |
| +incrementShootTimer | void | Отвечает за возобновление значения ShootTimer |
| +loadFont | void | Отвечает за загрузку определенного шрифта для текстовых объектов. |
| +text\_collision | void | Отвечает за границы игрового поля текстовых объектов игрока, чтобы они не пересекали поле, а также двигались относительно игрока |
| +checkAndShoot | void | Содержит в себе инструкцию по воспроизведению выстрела игроком по нажатию клавиши |

В таблице 4 приведены поля и методы класса Enemy.

Таблица 4

|  |  |  |
| --- | --- | --- |
| Поле / Метод | Возвращаемый тип | Использование |
| +shape | Sprite | Объект модели врага |
| +player | Player | Объект класса Player |
| +enemies | vector | Множественная реализация врагов в игре |
| +enemySpawnTimer | int | Временной интервал между появлением новых врагов (Частота появления врагов) |
| +enemy\_incrementTimer | void | Возобновление значения enemySpawnTimer |
| +speed | double | Скорость движения врагов |
| +moveAndHandleCollisions | void | Отвечает за столкновение игрока с противниками, получение урона в результате этого |
| +handleBulletCollision | void | Отвечает за столкновение пули с врагом, нанесение пулей урона врагу |
| +eraseEnemy | void | Удаление врага из его множества (вектора) |
| +DamageToPlayer | void | Нанесение урона игроку |
| +increaseSpeed | void | Увеличение скорости врага |
| +EnemySpawn | void | Появление врагов в левой части экрана |
| +resetEnemies | void | Удаление всех врагов с игрового поля |

В таблице 5 приведены поля и методы класса Drawing.

Таблица 5

|  |  |  |
| --- | --- | --- |
| Поле / Метод | Возвращаемый тип | Использование |
| +drawPlayerBullets | void | Отрисовка пуль игрока |
| +drawEnemies | void | Отрисовка врагов |

В таблице 6 приведены поля и методы класса Game.

Таблица 6

|  |  |  |
| --- | --- | --- |
| Поле / Метод | Возвращаемый тип | Использование |
| +player | Player | Объект класса Player |
| +enemy | Enemy | Объект класса Enemy |
| +drawMenu | void | Отрисовка главного меню |
| +showGameOverScreen | void | Отрисовка экрана проигрыша игрока для перезапуска игры |
| +update | void | Работа объектов всех классов друг с другом (работа игры) |

В таблице 7 приведены методы класса play\_game

|  |  |  |
| --- | --- | --- |
| Поле / Метод | Возвращаемый тип | Использование |
| +start\_game | int | Запуск игры |

Класс Textures:

Отвечает за загрузку данных, которые требуются для отрисовки моделей игрока, врагов, пули и заднего фона.

Класс Bullet:

Отвечает за создание пули внутри игрового поля, отрисовку пули.

Класс Player:

Отвечает за жизни, управление и загрузку текстуры игрока. Обновляет позицию игрока относительно игровой зоны, отвечает за выстрелы игроком во врагов, подсчет уничтоженных врагов в течении игры, текущее состояние жизней игрока.

Класс Enemy:

Отвечает за появление врагов внутри игровой зоны, их скорость движения, увеличение их скорости движения, наносимый урон игроку, в результате столкновения с ним.

Класс Drawing – отвечает за отрисовку пуль и вектора врагов на карте.

Класс Game:

Отвечает за весь игровой процесс, текущее состояние игрового процесса за работу между всеми созданными нами классами, отрисовку главного меню, а также меню проигрыша для дальнейшего рестарта игры.

Класс play\_game:

Отвечает за запуск игры.

2.2 Диаграмма классов

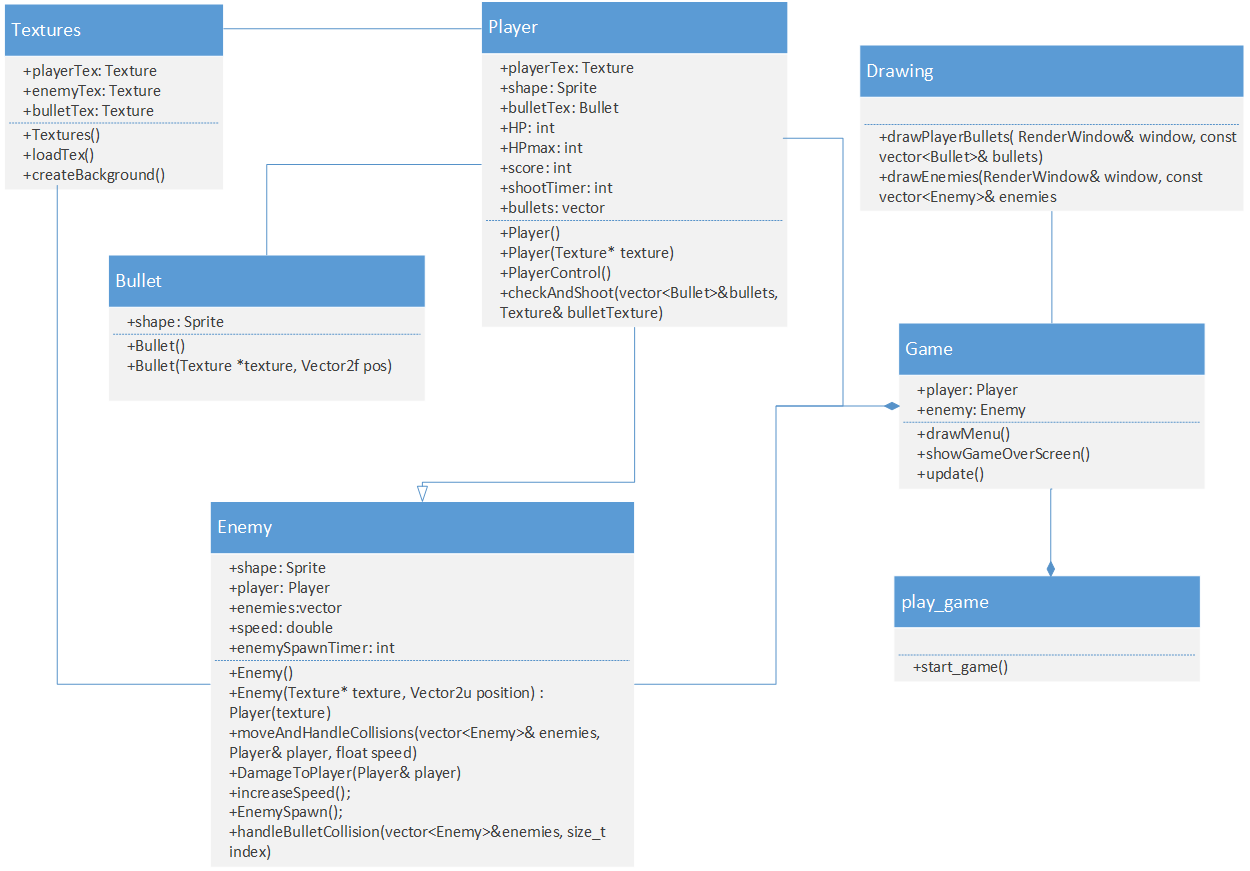


Рисунок 1 – Диаграмма классов

3 Код программы на языке *С++*

Текст программы по данному алгоритму находится в таблице 7.

Таблица 7 – Текст программы для решения индивидуального задания

|  |
| --- |
| Программа на языке С++ |
| #include <iostream>  #include <SFML\Graphics.hpp>  #include <SFML\Window.hpp>  #include <SFML\System.hpp>  #include <math.h>  #include <cstdlib>  #include <vector>  #include <locale.h>  #include <windows.h>  using namespace sf;  class Textures {  public:  Texture playerTex;  Texture enemyTex;  Texture bulletTex;  Textures() {  }  void loadTex() {  playerTex.loadFromFile("Textures/ship.png");  enemyTex.loadFromFile("Textures/enemy(2).png");  bulletTex.loadFromFile("Textures/missile.png");  }  static sf::Sprite createBackground(const std::string& filePath, int width, int height) {  sf::Texture texture;  if (!texture.loadFromFile(filePath)) {  // Обработка ошибки загрузки текстуры  }  sf::Sprite background(texture);  background.setScale(  static\_cast<float>(width) / texture.getSize().x,  static\_cast<float>(height) / texture.getSize().y  );  return background;  }  };  class Bullet  {  public:  Sprite shape;  Bullet() {  }  Bullet(Texture\* texture, Vector2f pos) {  this->shape.setTexture(\*texture);  this->shape.setScale(0.03f, 0.05f);  this->shape.setPosition(pos);  };  ~Bullet() {}  };  class Player  {  public:  Bullet\* bulletTex;  Text hpText;  Sprite shape;  Texture\* texture;  Texture playerTex;  Font font; // добавим шрифт в класс Enemy  int HP;  int HPmax;  int score = 0;  int shootTimer = 20;  int initialHP;  std::vector<Bullet> bullets;  Player() {  }  Player(Texture\* texture)  {  this->HPmax = 100;  this->HP = this->HPmax;  this->texture = texture;  this->shape.setTexture(\*texture);  this->shape.setScale(0.05f, 0.05f);  }  void shoot(std::vector<Bullet>& bullets, sf::Texture& bulletTex) {  sf::Vector2f bulletPos = shape.getPosition();  bulletPos.x += shape.getGlobalBounds().width / 2; // Позиция x находится посередине ширины объекта player  bulletPos.y += shape.getGlobalBounds().height / 2.7; // Позиция y находится посередине высоты объекта player  bullets.push\_back(Bullet(&bulletTex, bulletPos));  }  void PlayerControl() {  if (Keyboard::isKeyPressed(Keyboard::W))  shape.move(0.f, -10.f);  if (Keyboard::isKeyPressed(Keyboard::A))  shape.move(-10.f, 0.f);  if (Keyboard::isKeyPressed(Keyboard::S))  shape.move(0.f, 10.f);  if (Keyboard::isKeyPressed(Keyboard::D))  shape.move(10.f, 0.f);  }  void handleWindowCollision(const sf::Vector2u& windowSize) {  // Adjust player position based on window boundaries  if (shape.getPosition().x <= 0) // Left edge  shape.setPosition(0.f, shape.getPosition().y);  if (shape.getPosition().x >= windowSize.x - shape.getGlobalBounds().width) // Right edge  shape.setPosition(windowSize.x - shape.getGlobalBounds().width, shape.getPosition().y);  if (shape.getPosition().y <= 0) // Top edge  shape.setPosition(shape.getPosition().x, 0.f);  if (shape.getPosition().y >= windowSize.y - shape.getGlobalBounds().height) // Bottom edge  shape.setPosition(shape.getPosition().x, windowSize.y - shape.getGlobalBounds().height);  }  void incrementShootTimer() {  if (shootTimer < 13) {  shootTimer++;  }  }  void loadFont() {  if (!font.loadFromFile("Textures/Blackjac.ttf")) {  // Обработка ошибки загрузки шрифта  }  }    void text\_collision(Player player) {  hpText.setPosition(player.shape.getPosition().x, player.shape.getPosition().y - hpText.getGlobalBounds().height);  hpText.setString(std::to\_string(player.HP) + "/" + std::to\_string(player.HPmax));  }  void checkAndShoot(std::vector<Bullet>& bullets, sf::Texture& bulletTexture) {  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space) && shootTimer >= 13) {  shoot(bullets, bulletTexture);  shootTimer = 0; // Сбросить таймер  }  }  void reset() {  // Сброс параметров игрока до начального состояния  HP = initialHP; // Возвращаем здоровье игрока к начальному уровню  score = 0; // Сброс счетчика очков  // Другие действия по сбросу игры, если необходимо  }  ~Player() {}  };  class Enemy : public Player  {  public:  Sprite shape;  Player player;  std::vector<Enemy> enemies;  int enemySpawnTimer = 0;  double speed = -9;  int initialEnemyCount;  float initialSpeed;  Enemy() {  }  Enemy(Texture\* texture, Vector2u position) : Player(texture)  {  this->HPmax = 2;  this->HP = this->HPmax;  this->shape.setTexture(\*texture);  this->shape.setScale(0.2f, 0.2f);  this->shape.setPosition(position.x - this->shape.getGlobalBounds().width, rand() % (int)position.y - this->shape.getGlobalBounds().height);  }  void enemy\_incrementTimer() {  if (enemySpawnTimer < 19)  enemySpawnTimer++;  }  void moveAndHandleCollisions(std::vector<Enemy>& enemies, Player& player, float speed) {  for (size\_t i = 0; i < enemies.size(); i++) {  enemies[i].shape.move(speed, 0.f);  if (enemies[i].shape.getPosition().x <= 0 - enemies[i].shape.getGlobalBounds().width) {  eraseEnemy(enemies, i);  break;  }  if (enemies[i].shape.getGlobalBounds().intersects(player.shape.getGlobalBounds())) {  eraseEnemy(enemies, i);  DamageToPlayer(player);  increaseSpeed();  break;  }  }  }  void handleBulletCollision(std::vector<Enemy>& enemies, Player& player, size\_t bulletIndex) {  for (size\_t k = 0; k < enemies.size(); k++) {  if (player.bullets[bulletIndex].shape.getGlobalBounds().intersects(enemies[k].shape.getGlobalBounds())) {  if (enemies[k].HP <= 1) {  player.score += enemies[k].HPmax;  increaseSpeed(); // Уменьшить скорость  eraseEnemy(enemies, k);  }  else {  enemies[k].HP--;  }  player.bullets.erase(player.bullets.begin() + bulletIndex);  break;  }  }  }  void eraseEnemy(std::vector<Enemy>& enemies, size\_t index) {  enemies.erase(enemies.begin() + index);  }  void DamageToPlayer(Player& player) {  player.HP -= 10; // Игрок получает урон  }  void increaseSpeed() {  speed -= 0.1; // Увеличение скорости врага  }  void EnemySpawn(std::vector<Enemy>& enemies, sf::Texture& enemyTexture, const sf::Vector2u& windowSize) {  enemy\_incrementTimer(); // увеличиваем таймер  if (enemySpawnTimer >= 19) {  enemies.push\_back(Enemy(&enemyTexture, windowSize));  enemySpawnTimer = 0; // сбрасываем таймер  }  }  void resetEnemies() {  enemies.clear(); // очищаем вектор врагов  }  ~Enemy() {}  };  class Drawing {  public:  static void drawPlayerBullets(sf::RenderWindow& window, const std::vector<Bullet>& bullets) {  for (size\_t i = 0; i < bullets.size(); i++) {  window.draw(bullets[i].shape);  }  }  static void drawEnemies(sf::RenderWindow& window, const std::vector<Enemy>& enemies) {  for (size\_t i = 0; i < enemies.size(); i++) {  window.draw(enemies[i].shape);  }  }  };  class Game {  public:  Player player;  Enemy enemy;  void drawMenu(sf::RenderWindow& window) {  sf::Font font;  if (!font.loadFromFile("Textures/Blackjac.ttf")) {  // обработка ошибки загрузки шрифта  }  sf::Texture texture;  if (!texture.loadFromFile("Textures/Space.jpg")) {  // Обработка ошибки загрузки текстуры  }  // Создание объекта спрайта и установка текстуры  sf::Sprite background(texture);  sf::Text menuTitle("Kosmolet", font, 50); // создаем заголовок меню  menuTitle.setPosition(200, 100);  sf::Text playOption("Play", font, 30); // создаем опцию "Play"  playOption.setPosition(250, 200);  sf::Text helpOption("Help", font, 30); // создаем опцию "Help"  helpOption.setPosition(250, 250);  sf::Text exitOption("Exit", font, 30); // создаем опцию "Exit"  exitOption.setPosition(250, 300);  bool isMenuOpen = true;  int selectedOption = 0;  int helpm = 0;  while (isMenuOpen) {  window.clear();  // обработка событий  sf::Event event;  while (window.pollEvent(event)) {  if (event.type == sf::Event::Closed) {  window.close();  isMenuOpen = false;  }  if (event.type == sf::Event::KeyReleased) {  if (event.key.code == sf::Keyboard::W) {  selectedOption--;  if (selectedOption < 0) {  selectedOption = 2; // Учитывая новый пункт меню "Help"  }  }  if (event.key.code == sf::Keyboard::S) {  selectedOption = (selectedOption + 1) % 3; // Обеспечиваем зацикливание переключения опций  }  if (event.key.code == sf::Keyboard::Space) {  if (selectedOption == 0) {  // Действие для "Play"  std::cout << "Starting the game!" << std::endl;  isMenuOpen = false;  }  else if (selectedOption == 1) {  // Действие для "Help"  std::cout << "Entering Help section." << std::endl;  // Здесь вы можете добавить логику для перехода в окно с информацией об управлении  sf::RenderWindow helpWindow(sf::VideoMode(800, 600), "Controls Help");  // Отобразить текст с информацией об управлении  sf::Font font;  if (!font.loadFromFile("Textures/Blackjac.ttf.")) {  // Обработка ошибки загрузки шрифта  }  // Загрузка текстуры  sf::Texture texture;  if (!texture.loadFromFile("Textures/kik.jpg")) {  // обработка ошибки загрузки текстуры  }  // Создание спрайта с загруженной текстурой  sf::Sprite background(texture);  sf::Text controlInfo("BUTTONS:\n W - Up\n A - LEFT\n S - DOWN\n D - RIGHT\n Sp - SHOOT", font, 30);  sf::Text gameRools("Rools: \n You must shoot about 2 times\n to kill the enemy. With each \n killed enemie their speed will increase.\n Score increase with each killed.\n Good Game!", font, 30);  controlInfo.setPosition(300, 100); // Установите положение текста по вашему усмотрению  gameRools.setPosition(50, 300);  while (helpWindow.isOpen()) {  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {  helpWindow.close();  }  sf::Event helpEvent;  while (helpWindow.pollEvent(helpEvent)) {  if (helpEvent.type == sf::Event::Closed) {  helpWindow.close(); // Обработка закрытия окна через системное меню закрытия  }  }  helpWindow.clear();  helpWindow.draw(background); // Рисуем фоновое изображение  helpWindow.draw(controlInfo);  helpWindow.draw(gameRools);  helpWindow.display();  }  }  else if (selectedOption == 2) {  window.close(); // Close the window on "Exit"  isMenuOpen = false;  }  }  }  }  // отрисовка меню  playOption.setFillColor(sf::Color::White);  helpOption.setFillColor(sf::Color::White);  exitOption.setFillColor(sf::Color::White);  if (selectedOption == 0) {  playOption.setFillColor(sf::Color::Red);  }  else if (selectedOption == 1) {  helpOption.setFillColor(sf::Color::Red);  }  else {  exitOption.setFillColor(sf::Color::Red);  }  window.draw(background);  window.draw(menuTitle);  window.draw(playOption);  window.draw(helpOption);  window.draw(exitOption);  window.display();  }  }  void showGameOverScreen(sf::RenderWindow& window, int finalScore, bool& returnToMenu) {  //Создание шрифта  sf::Font font;  if (!font.loadFromFile("Textures/Blackjac.ttf")) {  // Обработка ошибки загрузки шрифта  }  // Загрузка текстуры  sf::Texture texture;  if (!texture.loadFromFile("Textures/kik.jpg")) {  // обработка ошибки загрузки текстуры  }  // Создание спрайта с загруженной текстурой  sf::Sprite background(texture);  sf::Text gameOverText("Game Over", font, 50);  gameOverText.setFillColor(sf::Color::Red);  gameOverText.setPosition(240, 200);  sf::Text scoreText("Your Score: " + std::to\_string(finalScore), font, 30);  scoreText.setFillColor(sf::Color::White);  scoreText.setPosition(265, 300);  sf::Text Instr("Press R to Continue ", font, 30);  Instr.setFillColor(sf::Color::Red);  Instr.setPosition(220, 400);  bool gameOver = true; // Flag to track if game over screen is shown  while (gameOver && window.isOpen()) {  sf::Event event;  while (window.pollEvent(event)) {  if (event.type == sf::Event::Closed) {  window.close();  returnToMenu = true; // Устанавливаем флаг возврата к меню  }  if (event.type == sf::Event::KeyReleased) {  if (event.key.code == sf::Keyboard::R) {  // Выходим из цикла, переключаем флаг game over в false  gameOver = false;  returnToMenu = true; // Устанавливаем флаг возврата к меню  if (returnToMenu = true) {  drawMenu(window);  }  }  }  }  window.clear();  window.draw(background); // Рисуем фоновое изображение  window.draw(gameOverText);  window.draw(scoreText);  window.draw(Instr);  window.display();  }  }  void update() {  srand(time(NULL));  setlocale(LC\_ALL, "RUS");  RenderWindow window(VideoMode(800, 600), "Kosmolet!", Style::Default);  window.setFramerateLimit(120);  sf::Image icon;  if (!icon.loadFromFile("Textures/ship.png")) {  // Обработка ошибки загрузки иконки  }  window.setIcon(icon.getSize().x, icon.getSize().y, icon.getPixelsPtr());  drawMenu(window);  //Init player textures  Textures func;  func.playerTex;  func.enemyTex;  func.bulletTex;  func.loadTex();  //Player init  Player player(&func.playerTex);  player.score;  player.shootTimer;  // Создание шрифта для игрока  player.font;  player.loadFont();  //UI init  Text scoreText;  scoreText.setFont(player.font);  scoreText.setCharacterSize(20);  scoreText.setFillColor(Color::White);  scoreText.setPosition(10.f, 10.f);  player.hpText;  player.hpText.setFont(player.font);  player.hpText.setCharacterSize(20);  player.hpText.setFillColor(Color::White);  player.hpText.setPosition(150.f, window.getSize().y / 2);  //Enemy init  Enemy enemy;  enemy.enemySpawnTimer;  enemy.speed;  enemy.enemies;  //wtf enemies  enemy.enemies.push\_back(Enemy(&func.enemyTex, window.getSize()));  bool isGameOver = false; // Новая переменная для определения, является ли игра оконченной  bool returnToMenu = false;  player.shape.setPosition(10, 200);  while (window.isOpen())  {  Event event;  while (window.pollEvent(event))  while (window.pollEvent(event))  {  if (event.type == Event::Closed)  window.close();  }  if (player.HP > 0)  {  //Player  player.PlayerControl(); // управление игроком  player.text\_collision(player);  //Collision with window  player.handleWindowCollision(window.getSize());  //Update Controls  player.incrementShootTimer();  player.checkAndShoot(player.bullets, func.bulletTex);  //Bullets  for (size\_t i = 0; i < player.bullets.size(); i++)  {  //Move  player.bullets[i].shape.move(20.f, 0.f);  //Out of window bounds  if (player.bullets[i].shape.getPosition().x > window.getSize().x)  {  player.bullets.erase(player.bullets.begin() + i);  break;  }  //Enemy collision  for (size\_t i = 0; i < player.bullets.size(); i++) {  enemy.handleBulletCollision(enemy.enemies, player, i);  }  }  //Enemy spawn  enemy.EnemySpawn(enemy.enemies, func.enemyTex, window.getSize());  //Enemy  enemy.moveAndHandleCollisions(enemy.enemies, player, enemy.speed); // Здесь enemySpeed - скорость движения врагов  //UI Update  scoreText.setString("Score: " + std::to\_string(player.score));  }  else if (player.HP <= 0) {  isGameOver = true; // Устанавливаем флаг, говорящий о том, что игра окончена  window.clear();  showGameOverScreen(window, player.score, returnToMenu); // Показываем экран окончания игры  player.HP += 100;  player.score = 0;  enemy.speed = -9;  player.shape.setPosition(10, 200);  enemy.resetEnemies();  player.bullets.clear();  }  // другие действия после окончания игры, если необходимо  //Draw =============================================================================== DRAW  // Загрузка изображения для фона  // Создаем объект для фона  sf::Texture texture;  if (!texture.loadFromFile("Textures/Space.jpg")) {  // Обработка ошибки загрузки текстуры  }  // Создание объекта спрайта и установка текстуры  sf::Sprite background(texture);  window.clear();  // Отрисовываем фон  window.draw(background);  //player  window.draw(player.shape); // Рисуем игрока  //bullets  Drawing::drawPlayerBullets(window, player.bullets); // Рисуем пули игрока  //enemy  Drawing::drawEnemies(window, enemy.enemies); // Отрисовка врагов  //UI  window.draw(scoreText);  window.draw(player.hpText);  window.display();  }  }  };  class play\_game {  public:  int start\_game() {  Game game;  game.update();  return 0;  }  };  int main() {  play\_game p;  p.start\_game();  return 0;  } |

1. Результаты тестирования

Результаты тестирования изображены на рисунках 2-5.

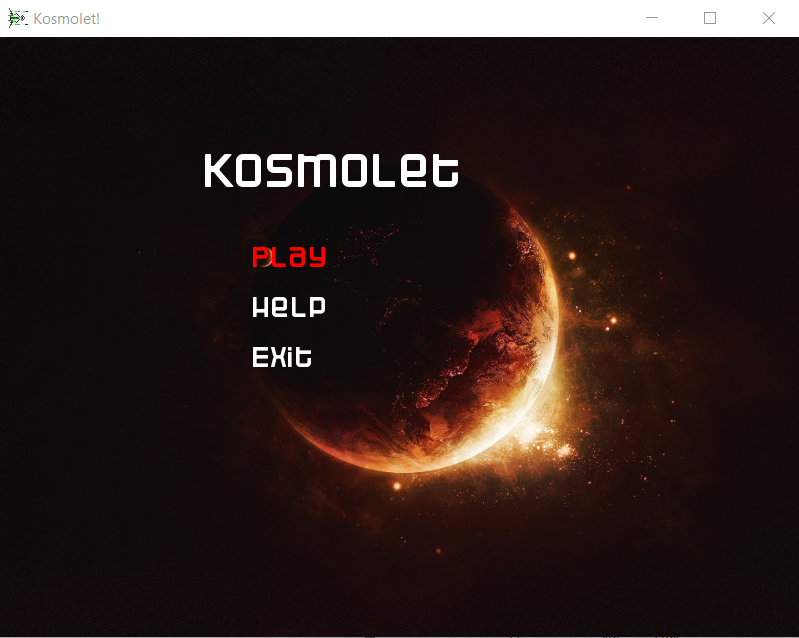


Рисунок 2 – Меню игры

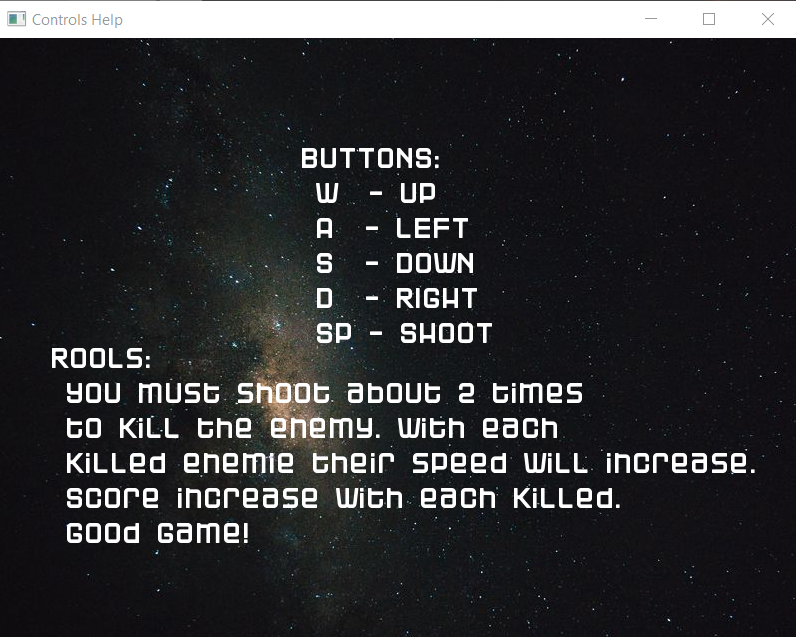


Рисунок 3 – Раздел Help

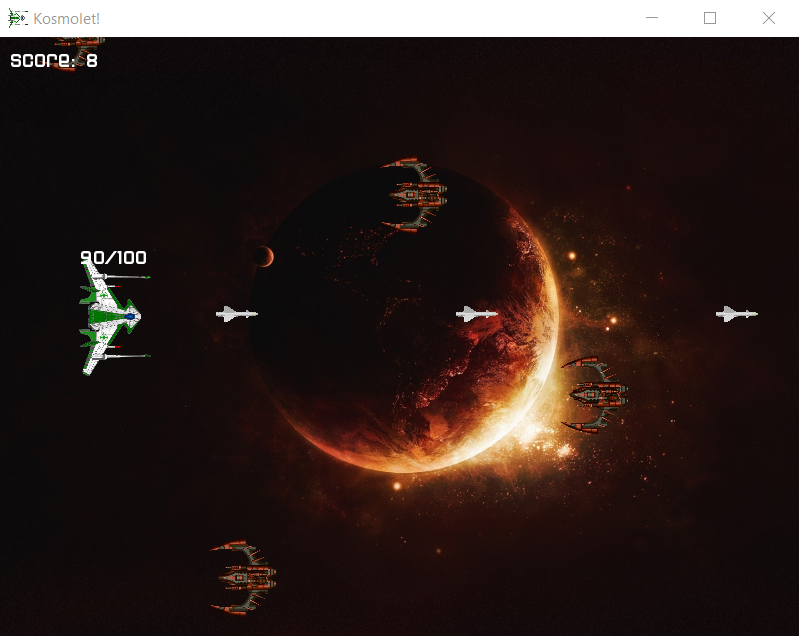


Рисунок 4 – Тестирование игры

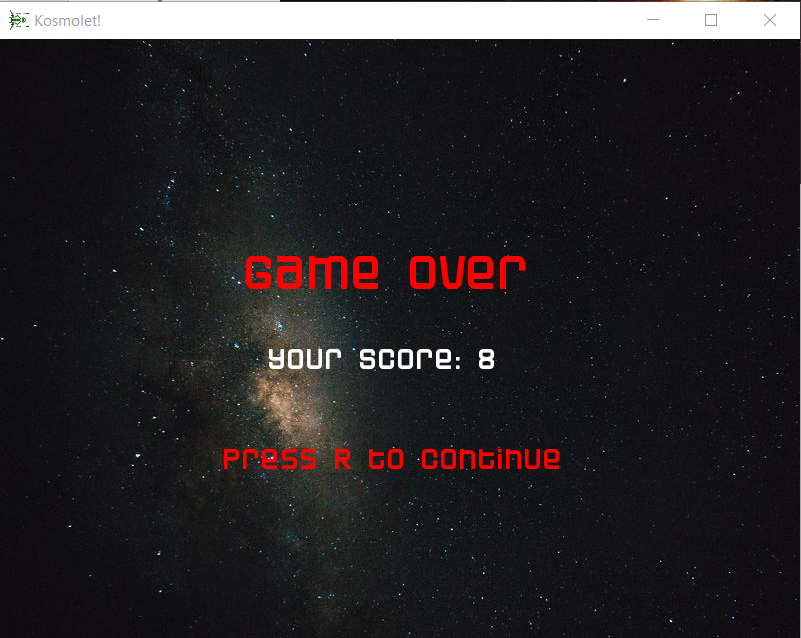


Рисунок 5 – Тестирование игры

Заключение

В ходе выполнения курсовой работы была написана программа, создающая симуляцию игры. Мной был сделан класс *Player*, который отвечает за отрисовку главного игрока, перемещение, а также стрельбу. Класс *Enemy*, который отвечает за отрисовку врага, его передвжиение, появление и атаку. В ходе выполнения курсовой работы оттачивались навыки создания классов и отношения между ними с использованием библиотеки *SFML*. Был получен большой опыт в проектировании игры, разработки и написания программы. Данная программа содержит 646 строк и имеет размер 190 КБ.

Вместе с написанием программы изучался курс программирования по *C*++ и оттачивались навыки работы с этим языком. Полученные навыки пригодятся в будущем как на *C++,* так и вообще в программировании по части проработки сложных алгоритмов.

В игру также можно внести следующие улучшения:

1. Добавить в игру «Босса»
2. Добавить возможность ответной стрельбы от врагов
3. Добавить игроку возможность восстанавливать свой уровень «здоровья».
4. Добавить возможность изменять виды «вооружения» главного персонажа.