Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Петрозаводский государственный университет»

Физико-технический институт

Кафедра информационно-измерительных систем и физической электроники

КОМАНДНАЯ РАЗРАБОТКА ПРОГРАММНОГО ПРИЛОЖЕНИЯ «STAR WARS»

КУРСОВОЙ ПРОЕКТ

по дисциплине

«Технология программирования»

Автор работы:

студент группы 21316

С. Н. Турцевич

«\_\_\_» \_\_\_\_\_\_\_\_ 2017 г.

Научный руководитель:

канд.физ.-мат.наук, доцент

А. В. Бульба

«\_\_\_» \_\_\_\_\_\_\_\_ 2017 г.

Петрозаводск 2017

Оглавление

[1. Цель работы 3](#_Toc502100504)

[2. Сюжета игры 4](#_Toc502100505)

[3. Программная реализация 5](#_Toc502100506)

[Заключение 6](#_Toc502100507)

[Приложение А 7](#_Toc502100508)

[Приложение Б 8](#_Toc502100509)

1. Цель работы

Создать простую 2D игру с использованием SFML. В игре должны быть: главный герой, враги и бонусы в виде дополнительных жизней и брони. На экране должно быть постоянно не более четырех врагов. Для маневрирования главного героя, враги должны стрелять с таким интервалом времени, чтобы между их выстрелами можно было пролететь. В верхней части экрана постоянно должны находиться количество жизней и очков, набранных за попадания. Так же при использовании брони должна выводиться информация, сколько она еще продержится.

1. Сюжета игры

(не более 0.5 страницы).

1. Программная реализация

Для создания игры использовалась среда разработки Microsoft Visual C++ 2010 Express, соответственно языком программирования был C++.

Были созданы следующие заголовочные файлы:

* bonus.h – файл с определением класса бонусов (Bonus);
* bullet.h – файл с определением класса пуль (Bullet);
* enemy.h – файл с определением класса врагов (Enemy);
* entity.h – файл с определением главного родительского класса (Entity);
* hero.h – файл с определением класса героя (Hero);
* globals.h – файл с определением глобальных констант и объявлением глобальных переменных.

И также следующие файлы исходного кода:

* bonus.cpp – файл с определением методов класса Bonus;
* bullet.cpp – файл с определением методов класса Bullet;
* enemy.cpp – файл с определением методов класса Enemy;
* entity.cpp – файл с определением методов класса Entity;
* globals.cpp – файл с определением глобальных переменных;
* hero.cpp – файл с определением методов класса Hero;
* main.cpp – главный файл исходного кода.

В главном файле исходного кода реализована стрельба героя по нажатию клавиши «пробел», вызывается функция класса героя, с помощью которой, по нажатию клавиш «S», «W», «A», «D» и стрелочек, осуществляется передвижение героя. Так же вызываются функции автоматического движения врагов, пуль и бонусов из соответствующих классов. Главное в данной игре выжить и набрать как можно больше очков, для этого нужно попадать по врагам, для этого реализована обработка попаданий пуль во врагов и в героя. Для того чтобы продержаться в живых дольше, были созданы бонусы дополнительных жизней и броня, их нужно собирать чтобы они начали действовать, за это отвечает обработка сбора бонусов. Во время игры не все наши выстрелы могут попасть во врага, и наоборот, поэтому пули улетают за границу видимой области, а так как каждая пуля — это отдельный объект, занимающий память в компьютере, которая имеет свойство заканчиваться, чтобы этого избежать организовано удаление улетевших и попавших пуль. После того как мы убили врага он умирает, но данный объект не удаляется, а просто появляется в новом месте через некоторое время из чего складывается ощущение что появился еще один новый враг. В процессе игры мы, естественно, хотим знать счет и сколько жизней у нас осталось, для этого в верхней части экрана рисуется эта информация. Тоже самое и когда у нас есть броня, хочется знать сколько попаданий еще она выдержит, информация об этом так же выводится сверху экрана. Как только мы наигрались и выключаем игру в памяти компьютера еще хранятся некоторые данные, от них нужно избавиться, чтобы не засорять память, для этого организована ее очистка.

Заключение

При разработке данной игры, была использована система контроля версий GitHub. Все прецеденты реализованы. Наблюдается один недочет: при появлении нового врага, его изображение может наложиться на изображение другого врага. Был использован принцип раздельной компиляции. Так же реализована очистка памяти при выходе из игры. Отсутствуют неиспользуемые переменные. Все алгоритмы неизбыточны. Диаграммы [см. Приложение А]. Цель достигнута.

Приложение А

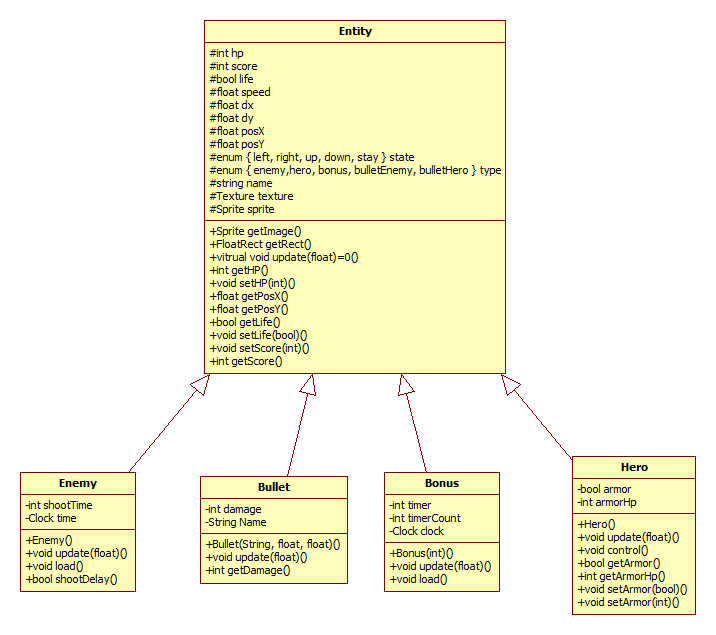


Рисунок 1. Диаграмма классов.

Приложение Б

Исходный код:

|  |
| --- |
| // bonus.h  #include "entity.h"  #ifndef \_\_BONUS\_H\_\_  #define \_\_BONUS\_H\_\_  class Bonus : public Entity  {  private:  int timer, timerCount;  Clock clock;  public:  Bonus(int);  void update(float);  void load();  };  #endif |
| // bullet.h  #include "entity.h"  #ifndef \_\_BULLET\_H\_\_  #define \_\_BULLET\_H\_\_  class Bullet : public Entity  {  private:  int damage;  String Name;  public:  Bullet(String, float, float);  void update(float);  int getDamage();  };  #endif |
| // enemy.h  #include "entity.h"  #ifndef \_\_ENEMY\_H\_\_  #define \_\_ENEMY\_H\_\_  class Enemy : public Entity  {  private:  int shootTime;  Clock time;  public:  Enemy();  void update(float);  void load();  bool shootDelay();  };  #endif |
| // entity.h  #include "SFML/Graphics.hpp"  #include "globals.h"  using namespace sf;  #ifndef \_\_ENTITY\_H\_\_  #define \_\_ENTITY\_H\_\_  class Entity  {  protected:  int hp, score;  bool life;  float speed, dx, dy, posX, posY;  enum { left, right, up, down, stay } state;  enum { enemy, hero, bonus, bulletEnemy, bulletHero } type;  std::string name;  Texture texture;  Sprite sprite;  public:  Sprite getImage();  FloatRect getRect();  virtual void update(float) = 0;  int getHP();  void setHP(int);  float getPosX();  float getPosY();  bool getLife();  void setLife(bool);  void setScore(int);  int getScore();  };  #endif |
| // hero.h  #include "entity.h"  #ifndef \_\_HERO\_H\_\_  #define \_\_HERO\_H\_\_  class Hero : public Entity  {  private:  bool armor;  int armorHp;  public:  Hero();  void update(float);  void control();  bool getArmor();  int getArmorHp();  void setArmor(bool);  void setArmor(int);  };  #endif |
| // globals.h  #ifndef \_\_GLOBALS\_H\_\_  #define \_\_GLOBALS\_H\_\_    const int weigth = 800;  const int height = 600;  const int sizeEnemy = 50;  extern int sizeHero;  const int sizeBullet = 5;  const int sizeBonus = 25;  const int delay = 2400;    #endif |
| // bonus.cpp  #include "bonus.h"  Bonus::Bonus(int t)  {  speed = 0.08;  type = bonus;  life = false;  if (t == 0)  {  timerCount = 10 + rand() % 20;  hp = 75;  texture.loadFromFile("images/health.png");  }  if (t == 1)  {  timerCount = 20 + rand() % 35;  texture.loadFromFile("images/energy.png");  }  sprite.setTexture(texture);  load();  sprite.setPosition(posX, posY);  }  void Bonus::load()  {  posX = rand() % (weigth - sizeBonus);  posY = 0;  }  void Bonus::update(float time)  {  if (life)  {  dy = speed;  clock.restart();  if (posY >= 0)  posY += dy\*time;    sprite.setPosition(posX, posY);  if (posY >= (height - sizeBonus))  {  life = false;  }  }  else  {  load();  timer = clock.getElapsedTime().asSeconds();  if (timer >= timerCount)  {  life = true;  }  }  } |
| // bullet.cpp  #include "bullet.h"  Bullet::Bullet(String name, float x, float y)  {  Name = name;  damage = 50;  posX = x;  posY = y;  if (Name == "enemy")  {  type = bulletEnemy;  speed = 0.1;  texture.loadFromFile("images/bulletEnemy.png");  sprite.setTexture(texture);  }  if (Name == "hero")  {  type = bulletHero;  speed = 0.5;  texture.loadFromFile("images/bulletHero.png");  sprite.setTexture(texture);  }  }  int Bullet::getDamage()  {  return damage;  }  void Bullet::update(float time)  {  if (life)  {  if (Name == "enemy")  {  dy = speed;  posY += dy\*time;  sprite.setPosition(posX + sizeEnemy / 2 - sizeBullet / 2, posY + sizeEnemy);  if (posY >= height)  {  life = false;  }  }  if (Name == "hero")  {  dy = -speed;  posY += dy\*time;  sprite.setPosition(posX + sizeHero / 2 - sizeBullet / 2, posY - (4 \* sizeBullet));  if (posY <= 0)  {  life = false;  }  }  }  } |
| // enemy.cpp  #include "enemy.h"  Enemy::Enemy()  {  hp = 100;  speed = 0.02;  type = enemy;  shootTime = 2500;  texture.loadFromFile("images/enemy.png");  sprite.setTexture(texture);  load();  load();  sprite.setPosition(posX, posY);  }  void Enemy::load()  {  posX = (rand() % 13) \* 60 + 10; // Выбирается позиция от 10 до 790 с шагом 60  posY = 0;  }  void Enemy::update(float time)  {  if (life)  {  dy = speed;  if ((posY >= 0) && (posY < (height - sizeEnemy)))  posY += dy\*time;  if (posY >= (height - sizeEnemy))  {  load();  }    sprite.setPosition(posX, posY);  if (hp <= 0)  {  life = false;  load();  }  }  }  bool Enemy::shootDelay()  {  shootTime = time.getElapsedTime().asMilliseconds();  if (shootTime >= delay)  {  time.restart();  return true;  }  else  return false;  } |
| // entity.cpp  #include "entity.h"  Sprite Entity::getImage()  {  return sprite;  }  bool Entity::getLife()  {  return life;  }  void Entity::setLife(bool l)  {  life = l;  if (l)  {  hp = 100;  }  }  void Entity::setScore(int s)  {  score += s;  }  int Entity::getScore()  {  return score;  }  int Entity::getHP()  {  return hp;  }  void Entity::setHP(int h)  {  hp += h;  }  float Entity::getPosX()  {  return posX;  }  float Entity::getPosY()  {  return posY;  }  FloatRect Entity::getRect()  {  if (type == enemy)  {  FloatRect FR(posX - sizeEnemy / 2, posY, sizeEnemy, sizeEnemy);  return FR;  }  if (type == hero)  {  if (sizeHero == 50)  {  FloatRect FR(posX - sizeHero / 2, posY, sizeHero, sizeHero);  return FR;  }  else  {  FloatRect FR(posX - sizeHero / 2 + 10, posY, sizeHero, sizeHero);  return FR;  }  }  if (type == bonus)  {  FloatRect FR(posX - sizeBonus / 2, posY, sizeBonus, sizeBonus);  return FR;  }  if (type == bulletEnemy)  {  FloatRect FR(posX, posY + 10 \* sizeBullet, sizeBullet, 4 \* sizeBullet);  return FR;  }  if (type == bulletHero)  {  FloatRect FR(posX, posY, sizeBullet, 4 \* sizeBullet);  return FR;  }    } |
| // hero.cpp  #include "hero.h"  Hero::Hero()  {  sizeHero = 50;  state = stay;  hp = 100;  posX = (weigth - sizeHero) / 2;  posY = height - sizeHero;  speed = 0.3;  type = hero;  score = 0;  armor = false;  texture.loadFromFile("images/hero.png");  sprite.setTexture(texture);  sprite.setPosition(posX, posY);  }  void Hero::update(float time)  {  if (life)  {  control();  switch (state)  {  case right:  dx = speed;  break;  case left:  dx = -speed;  break;  case up:  dy = -speed;  break;  case down:  dy = speed;  break;  case stay:  dx = 0;  dy = 0;  break;  default:  break;  }  if ((posX > 0) && (dx < 0))  posX += dx\*time;  if ((posX < (weigth - sizeHero)) && (dx > 0))  posX += dx\*time;  if ((posY > (2 \* height / 3)) && (dy < 0))  posY += dy\*time;  if ((posY < (height - sizeHero)) && (dy > 0))  posY += dy\*time;  state = stay;  sprite.setPosition(posX, posY);  if (hp <= 0)  {  hp = 0;  life = false;  }  if (armor)  {  texture.loadFromFile("images/armor1.png");  sprite.setTexture(texture, true);  sizeHero = 60;  }  else  {  texture.loadFromFile("images/hero.png");  sprite.setTexture(texture, true);  sizeHero = 50;  }  if (armorHp <= 0)  {  armorHp = 0;  armor = false;  }  }  }  void Hero::control()  {  if ((Keyboard::isKeyPressed(Keyboard::Left)) || (Keyboard::isKeyPressed(Keyboard::A)))  {  state = left;  }    if ((Keyboard::isKeyPressed(Keyboard::Right)) || (Keyboard::isKeyPressed(Keyboard::D)))  {  state = right;  }    if ((Keyboard::isKeyPressed(Keyboard::Up)) || (Keyboard::isKeyPressed(Keyboard::W)))  {  state = up;  }    if ((Keyboard::isKeyPressed(Keyboard::Down)) || (Keyboard::isKeyPressed(Keyboard::S)))  {  state = down;  }  }  bool Hero::getArmor()  {  return armor;  }  int Hero::getArmorHp()  {  return armorHp;  }  void Hero::setArmor(bool a)  {  if (!getArmor())  {  posX -= 5;  posY -= 10;  }  armor = a;  armorHp = 300;  }  void Hero::setArmor(int a)  {  armorHp += a;  } |
| // globals.cpp  #include "globals.h"  int sizeHero = 0; |
| // main.cpp  #include "hero.h"  #include "enemy.h"  #include "bullet.h"  #include "bonus.h"  #include <iostream>  #include <sstream>  #include <list>  int main()  {  Hero hero;  Bonus HP(0);  Bonus Armor(1);  int enemyCount = 4;  int enemyInGame = 0;  sf::VideoMode desktop = sf::VideoMode::getDesktopMode();  sf::RenderWindow window(sf::VideoMode(weigth, height, desktop.bitsPerPixel), "Star Wars");  Font font;  font.loadFromFile("zealot.ttf");  Text textHp("", font, 20);  textHp.setColor(Color::Red);  Text textScore("", font, 20);  textScore.setColor(Color::Red);  Text textLose("", font, 50);  textLose.setColor(Color::Red);  Text textArmor("", font, 20);  textArmor.setColor(Color::Green);  std::list<Enemy\*> enemies;  std::list<Enemy\*>::iterator itEnemy;  std::list<Bullet\*> bullHero;  std::list<Bullet\*> bullEnemy;  std::list<Bullet\*>::iterator itBull;  Clock clock;  Clock clock2;  Clock clock3;  int shootTime = 0;  int time2 = 0;  for (int i = 0; i < enemyCount; i++)  {  enemies.push\_back(new Enemy());  enemyInGame++;  }  while (window.isOpen())  {  float time = clock.getElapsedTime().asMicroseconds();  clock.restart();  time = time / 800;  sf::Event event;  while (window.pollEvent(event))  {  if (event.type == sf::Event::Closed)  window.close();  // Выстрел по нажатию пробела  shootTime = clock2.getElapsedTime().asMilliseconds();  if (event.type == sf::Event::KeyPressed)  {  if ((event.key.code == sf::Keyboard::Space) && (hero.getLife()) && (shootTime >= delay / 12))  {  clock2.restart();  bullHero.push\_back(new Bullet("hero", hero.getPosX(), hero.getPosY()));  }  }  }  // Оживляем врагов  for (itEnemy = enemies.begin(); itEnemy != enemies.end(); itEnemy++)  {  (\*itEnemy)->update(time);  }  // Оживляем пули героя  for (itBull = bullHero.begin(); itBull != bullHero.end(); itBull++)  {  (\*itBull)->update(time);  }  // Удаляем пули героя, если они улетели за карту или попали во врага  for (itBull = bullHero.begin(); itBull != bullHero.end();)  {  if ((\*itBull)->getLife() == false)  {  itBull = bullHero.erase(itBull);  }  else  {  itBull++;  }  }  // Выстрелы врага  for (itEnemy = enemies.begin(); itEnemy != enemies.end(); itEnemy++)  {  if ((\*itEnemy)->getLife())  {  if ((\*itEnemy)->shootDelay())  {  bullEnemy.push\_back(new Bullet("enemy", (\*itEnemy)->getPosX(), (\*itEnemy)->getPosY()));  }  // Проверяем попадание во врага  for (itBull = bullHero.begin(); itBull != bullHero.end(); itBull++)  {  if ((\*itEnemy)->getRect().intersects((\*itBull)->getRect()))  {  clock3.restart();  (\*itEnemy)->setHP(-(\*itBull)->getDamage());  (\*itBull)->setLife(false);  hero.setScore(100);  }  }  }  else  {  time2 = clock3.getElapsedTime().asMilliseconds();  }  }  // После убийства одного врага, добавляем "нового"  for (itEnemy = enemies.begin(); itEnemy != enemies.end(); itEnemy++)  {  if (!((\*itEnemy)->getLife()) && (time2 >= 1000))  {  clock3.restart();  (\*itEnemy)->setLife(true);  }  }  // Оживляем пули врага  for (itBull = bullEnemy.begin(); itBull != bullEnemy.end(); itBull++)  {  (\*itBull)->update(time);  }  // Обработка попадания в героя  if (hero.getLife())  {  for (itBull = bullEnemy.begin(); itBull != bullEnemy.end(); itBull++)  {  if (hero.getRect().intersects((\*itBull)->getRect()))  {  if (hero.getArmor())  {  hero.setArmor(-(\*itBull)->getDamage());  }  else  {  hero.setHP(-(\*itBull)->getDamage());  }  (\*itBull)->setLife(false);  }  }  }  // Обработка сбора бонусов  if (hero.getLife())  {  if (hero.getRect().intersects(HP.getRect()))  {  hero.setHP(HP.getHP());  HP.setLife(false);  }  if (hero.getRect().intersects(Armor.getRect()))  {  Armor.setLife(false);  hero.setArmor(true);  }  }  // Удаляем пули врага, если они улетели за карту или попали в героя  for (itBull = bullEnemy.begin(); itBull != bullEnemy.end();)  {  if ((\*itBull)->getLife() == false)  {  itBull = bullEnemy.erase(itBull);  }  else  {  itBull++;  }  }    // Оживляем бонус доп. жизней  HP.update(time);  // Оживляем бонус брони  Armor.update(time);  // Оживляем героя  hero.update(time);    window.clear();  // Рисуем героя  if (hero.getLife())  {  window.draw(hero.getImage());  }  // Рисуем бонус доп. жизни  if (HP.getLife())  {  window.draw(HP.getImage());  }  // Рисуем бонус броню  if (Armor.getLife())  {  window.draw(Armor.getImage());  }  // Рисуем врагов  for (itEnemy = enemies.begin(); itEnemy != enemies.end(); itEnemy++)  {  if ((\*itEnemy)->getLife())  {  window.draw((\*itEnemy)->getImage());  }  }  // Рисуем пули героя  for (itBull = bullHero.begin(); itBull != bullHero.end(); itBull++)  {  if ((\*itBull)->getLife())  window.draw((\*itBull)->getImage());  }  // Рисуем пули врагов  for (itBull = bullEnemy.begin(); itBull != bullEnemy.end(); itBull++)  {  if ((\*itBull)->getLife())  window.draw((\*itBull)->getImage());  }  // Если герой умер или враги смогли пролететь, то вы проиграли  for (itEnemy = enemies.begin(); itEnemy != enemies.end(); itEnemy++)  {  if (((\*itEnemy)->getPosY() >= (height - sizeEnemy - 5)) || !(hero.getLife()))  {  hero.setLife(false);  textLose.setString("You are lose!");  textLose.setPosition(150, height / 2 - 50);  window.draw(textLose);  }  }  // Вывод количества брони, если она есть  if (hero.getArmor())  {  std::ostringstream playerArmorString;  playerArmorString << hero.getArmorHp();  textArmor.setString("Armor: " + playerArmorString.str());  textArmor.setPosition(350, 0);  window.draw(textArmor);  }    // Вывод жизней и очков на экран  std::ostringstream playerHPString;  std::ostringstream playerScoreString;  playerHPString << hero.getHP();  playerScoreString << hero.getScore();  textHp.setString("Health: " + playerHPString.str());  textHp.setPosition(600, 0);  textScore.setString("Score: " + playerScoreString.str());  textScore.setPosition(10, 0);  window.draw(textHp);  window.draw(textScore);  window.display();  }  // Очистка памяти  for (itBull = bullHero.begin(); itBull != bullHero.end();)  {  itBull = bullHero.erase(itBull);  }  for (itBull = bullEnemy.begin(); itBull != bullEnemy.end();)  {  itBull = bullEnemy.erase(itBull);  }  for (itEnemy = enemies.begin(); itEnemy != enemies.end();)  {  itEnemy = enemies.erase(itEnemy);  }  return 0;  } |