Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Петрозаводский государственный университет»

Физико-технический институт

Кафедра информационно-измерительных систем и физической электроники

РАЗРАБОТКА КОМПЬЮТЕРНОЙ ИГРЫ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C++ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ SFML

Курсовая работа

Автор работы:

студент группы 21412

А.А.Койвестойнен

«19» декабря 2022 г.

Принял:

канд. физ.-мат. наук, доцент

А. В. Бульба

«20» декабря 2022 г.

Петрозаводск 2022

Содержание

Оглавление

ВВЕДЕНИЕ 3
История коммитов на GitHab 4
Перечень индивидуальных работ 6
ЗАКЛЮЧЕНИЕ 7
ПРИЛОЖЕНИЕ А 8
ПРИЛОЖЕНИЕ Б 9
ПРИЛОЖЕНИЕ В 10

ВВЕДЕНИЕ

Целью выполнения данной курсовой работы является реализация простой 2D-игры на языке программирования C++ с использованием библиотеки SFML (Simple and Fast Multimedia Library — простая и быстрая мультимедийная библиотека).

Было решено написать современную версию игры “Tank 1990”.

В программе должен быть реализован класс предок, класс-игрок, класс-враг, класс-пуля. В игре объект-игрок в одном экземпляре, объекты-враги создаются и уничтожаются по ходу игры, объекты-пули создаются и уничтожаются по ходу игры. Пересечение участников игры постоянно проверяется возможностями SFML.

При разработке программы использовано наследование, контейнеры, итераторы, раздельная компиляция. В общем отчете присутствует UML-диаграмма классов (class diagram) и диаграмма вариантов использования (use case diagram) разработанной программы.Промежуточные результаты работы команды сохранялись на GitHub.

История коммитов на GitHab

Адрес проекта на GitHab: https://github.com/DyachenkoAnna/game.git

Ниже приведен список моих коммитов:

Author: Anna <koyvestoynen@bk.ru>
Date: Wed Dec 14 19:23:52 2022 +0300

Добавлено описание класса Entity.h

Author: Anna <koyvestoynen@bk.ru>
Date: Thu Dec 15 00:54:05 2022 +0300

Добавление класса игрока-главного героя

Author: Anna <koyvestoynen@bk.ru>
Date: Fri Dec 16 02:06:34 2022 +0300

Добавление конструктора, метода определения движения игрока и метода определения столкновения с границами карты

Author: Anna <koyvestoynen@bk.ru>
Date: Sun Dec 18 14:56:12 2022 +0300

Обновлен класс Player

Author: Anna <koyvestoynen@bk.ru>
Date: Sun Dec 18 15:01:15 2022 +0300

Добавлены методы для движения главного героя, а также метод для анимации: движение колес, поворот пушки

Author: Anna <koyvestoynen@bk.ru>
Date: Sun Dec 18 15:11:37 2022 +0300

Обновление класса героя для отображения отрисовки колес и пушки

Author: Anna <koyvestoynen@bk.ru>
Date: Sun Dec 18 15:13:32 2022 +0300

Добавлен метод для отрисовки колес и пушки

Author: Anna <koyvestoynen@bk.ru>
Date: Sun Dec 18 18:09:21 2022 +0300

Обновление класса героя,дописано возращение позиции спрайта героя и конца пушки

Author: Anna <koyvestoynen@bk.ru>
Date: Sun Dec 18 18:14:52 2022 +0300

    Дописаны методы для окрашивания при нанесении урона и положения пушки

Author: Anna <koyvestoynen@bk.ru>
Date: Sun Dec 18 18:22:40 2022 +0300

    Добавлены комментарии

Author: Anna <koyvestoynen@bk.ru>
Date: Tue Dec 20 22:52:44 2022 +0300

    Исправлено положение вылетающего снаряда, добавлен учет координат центра танка для корректного наведения пушки по курсору

Перечень индивидуальных работ

В начале работы было принято решение о совместной работе над всеми необходимыми файлами, все члены команды предлагали свои идеи и способы усовершенствования уже принятых решений.
Конкретно мной велись работы по созданию таких файлов, как Entity.h, Player.h и Player.cpp.
Заголовочный файл Entity.h - это основной класс, наследуется к остальным объектам.
Поля:
- Текстура, Спрайт
- Строка отличия для схожих объектов - имя
- Булевые переменные живой - нет, двигается - нет
- Целые числа жизни и две переменные под размер спрайта
- Дробные числа под скорость/перемещение + таймер

Заголовочный файл Player.h - класс игрока, наследуется от класса Entity.
Переменные от родителя:
float dx, dy, x, y, speed, moveTimer;
int w, h, health;
bool life, isMove;
sf::Texture texture;
sf::Sprite sprite;
sf::String name;

Исходный файл Player.cpp содержит в себе методы для работы с главным героем, а именно конструктор, метод для определения движения героя, проверку столкновения с краями карты, анимацию движения колес и т.п., метод для движения главного игрока, положения пушки, отрисовку самого героя и окрашивание героя в красный цвет при нанесении урона.
В начале игры у главного героя есть максимальное число жизней, при попадании по нему врагами часть жизней отнимается, ее можно восполнить, встав на точку оздоровления (так называемый хил поинт).
Для того чтобы был шанс спрятаться от врага, на карте установлены специальные ящики.
При нанесении урона врагами по главному герою предусмотрена анимация получения урона, а именно окрашивание танка в красный цвет.
Как только у главного героя заканчиваются жизни, он умирает, игра заканчивается и на экран выводится итоговый счет.

Помимо работы с классом игрока совместно с Сулаковой С.В. было разработано руководство пользователя. Для удобства полное руководство пользователя прикрепили отдельным файлом в репозитории, а сокращенную версию добавили в общий отчет.
В приложении А, Б, и В приведены тексты файлов, над которыми я работала.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной курсовой работы создана игра на языке программирования C++, которую можно считать современной версией игры “Tank 1990”.

В процессе работы применялась система контроля версий Git. Реализованы все прецеденты. Сбои/зависания программы во время тестирования и использования не наблюдаются. Программа написана с учетом принципа раздельной компиляции. Реализовано освобождение памяти для класса Engine (в нем освобождается вся используемая память). Нет неиспользуемых методов, атрибутов и переменных. Все конструкции в программе используются для работы с классами. Нет конструкций, без которых можно было обойтись. К отчету приложены диаграммы классов и вариантов использования. Поставленная цель достигнута.

ПРИЛОЖЕНИЕ А

Entity.h

```
#ifndef __Entity__
#define __Entity__

#include <SFML/Graphics.hpp>

class Entity {
public:
	sf::Texture texture;
	sf::Sprite sprite;
	sf::FloatRect GetRect() {return sf::FloatRect(x,y,w,h);};//прямоугольник для поиска пересечения объектов
	Entity(sf::Image& image, float X, float Y, int W, int H, sf::String Name) {
		x = X; y = Y; w = W; h = H; name = Name; moveTimer = 0;
		speed = 0; health = 100; dx = 0; dy = 0;//dx и dy - на сколько менять позицию
		life = true; isMove = false;
		texture.loadFromImage(image);//установка текстуры
		sprite.setTexture(texture);//вырезка спрайта из текстуры
		sprite.setOrigin(w / 2, h / 2);//координаты в центре спрайта
	}
	bool isAlive() { return life; };//возвращает позицию спрайта
protected:

	float dx, dy, x, y, speed, moveTimer;
	int w, h, health;
	bool life, isMove;
	sf::String name;//враги по именам
	enum { left, right, up, down, stay, fly } state; //тип пересечение - состояние объекта
};

#endif
```

ПРИЛОЖЕНИЕ Б

Player.h

```
#ifndef __Player_h__
#define __Player_h__

#include "Entity.h"
#include "Constants.h"
/*
Класс игрок. Главный герой

Переменные от родителя:
float dx, dy, x, y, speed, moveTimer;
int w, h, health;
bool life, isMove;
sf::Texture texture;
sf::Sprite sprite;
sf::String name;
*/

class Player :protected Entity {
public:
	Player(sf::Image& image, float X, float Y, int W, int H, sf::String Name);
	bool isAlive() { return life; };
	sf::FloatRect GetRect() { return sf::FloatRect(x+5, y+5, w-5, h-5); };
	float GetRotation() { return gunrotation; };//берем направление пушки
	int Getscore() { return score; };
	void Addscore(int SCR) { score += SCR; };
	int Gethealth() { return health; };
	int update(float time, sf::String TileMap[HEIGHT_MAP], sf::Event event);// Жизнь объекта,
ф-я вызывается в основной программе
	void struck(int damage);
	sf::Vector2f GetXY() { return sf::Vector2f(x + w / 2, y + h / 2); }; //
Возвращает позицию спрайта героя
	sf::Vector2f GetgunXY();// возвращает позицию конца пушки
	void draw(sf::RenderTarget& target);
private:
	// Будет вызываться внутри
	int control(sf::Event event);
	int checkCollisionWithMap(float, float, sf::String TileMap[HEIGHT_MAP]);//Проверка
столкновений с картой
	sf::Sprite wheelR;//правые колеса
	sf::Sprite wheelL;//левые
	sf::Sprite gun;//пушка
	sf::Sprite frame;//рамка
	float gunrotation;
	int score;
	float struckTimer;//для анимации нанесения урона
	void animation();
};
#endif
```

ПРИЛОЖЕНИЕ В

Player.cpp

```
#include "Player.h"
#include "Constants.h"

// Конструктор
Player::Player(sf::Image &image, float X, float Y,int W,int H, sf::String
Name):Entity(image,X,Y,W,H,Name)
{
	score = 0;
	struckTimer = 0;
    state = up;
	frame.setTexture(texture);
	wheelL.setTexture(texture);
	wheelR.setTexture(texture);
	gun.setTexture(texture);

	wheelL.setScale(0.9, 1);
	wheelR.setScale(0.9, 1);
	frame.setScale(0.9, 0.9);
	gun.setScale(0.9, 0.9);
	//размеры
	wheelL.setTextureRect(sf::IntRect(1, 80, 19, 77));
	wheelL.setOrigin(w / 2 + 10, h / 2);

	wheelR.setTextureRect(sf::IntRect(20, 81, 19, 77));
	wheelR.setOrigin(- w / 2 +10, h / 2);

	frame.setTextureRect(sf::IntRect(84,80,70,80));
	frame.setOrigin(w / 2, h / 2);

	gun.setTextureRect(sf::IntRect(181, 81, 36, 78));
	gun.setOrigin(18, 58);
	//позиция-центр спрайтов
	frame.setPosition(x, y);
	wheelL.setPosition(x, y);
	wheelR.setPosition(x, y);
	gun.setPosition(x, y);
	gunrotation = 0;
}

 //Для определения движения
int  Player::control(sf::Event event)
{
	state = stay;
	sf::Vector2i DXY = sf::Mouse::getPosition();
	//sf::Vector2i DXY = sf::Vector2i(event.mouseMove.x, event.mouseMove.y);
	gunrotation = atan2(-y + DXY.y - 40, -x + DXY.x - 35) * 180 / 3.14159265 +
90;

	if (sf::Keyboard::isKeyPressed) {//если нажата клавиша
		if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
			state = left; speed = 0.1;
		}
		else if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
```

```
                    state = right; speed = 0.1;
              }
              else if ((sf::Keyboard::isKeyPressed(sf::Keyboard::W))) {
                    state = up; speed = 0.085;
              }
              else if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
                    state = down; speed = 0.085;
              }
        }

        return 0;
}

int  Player::checkCollisionWithMap(float Dx, float Dy, sf::String TileMap[HEIGHT_MAP])
{
        for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты
              for (int j = x / 32; j < (x + w) / 32; j++)
              {
                    if (TileMap[i][j] == '0' || TileMap[i][j] == 'b')
                    {
                          if (Dy > 0) { y = i * 32 - h;  dy = 0; }//столкновение с
нижним краем карты
                          if (Dy < 0) { y = i * 32 + 32;  dy = 0; }//с верхним
краем карты
                          if (Dx > 0) { x = j * 32 - w; }//с правым краем карты
                          if (Dx < 0) { x = j * 32 + 32; }// с левым краем карты
                    }
                    if (TileMap[i][j] == 'h' && state == stay && moveTimer > 1000)
                    {
                          health += 10;
                          //Если встали и стоим на хил поинте, то прибавляем
здоровье
                          moveTimer = 500;
                          if (health > 500)
                          {
                                health = 500;
                          }
                    }
              }

        return 0;
}

//Анимация
void  Player::animation()
{
        if (state != stay)
        {
              if (moveTimer < 100)
              {
                    wheelL.setTextureRect(sf::IntRect(1, 80, 19, 77));
                    wheelR.setTextureRect(sf::IntRect(20, 81, 19, 77));
              }
              else if (moveTimer < 200)
              {
                    wheelL.setTextureRect(sf::IntRect(40, 82, 19, 77));
                    wheelR.setTextureRect(sf::IntRect(59, 82, 19, 77));
              }
              else
```

```
            {
                  moveTimer = 0;
            }
            //Движение колес
      }
      gun.setRotation(gunrotation);
      switch (state)
      {
      case Entity::left:
            frame.setRotation(-90);
            wheelL.setRotation(-90);
            wheelR.setRotation(-90);
            break;
      case Entity::right:
            frame.setRotation(90);
            wheelL.setRotation(90);
            wheelR.setRotation(90);
            break;
      case Entity::up:
            frame.setRotation(0);
            wheelL.setRotation(0);
            wheelR.setRotation(0);
            break;
      case Entity::down:
            frame.setRotation(180);
            wheelL.setRotation(180);
            wheelR.setRotation(180);
            break;
      default:
            break;
      }
      //повернули спрайты по направлению движения
      if (struckTimer > 0)
      {
            gun.setColor(sf::Color::Red);
            frame.setColor(sf::Color::Red);
            wheelL.setColor(sf::Color::Red);
            wheelR.setColor(sf::Color::Red);
      }
      else
      {
            gun.setColor(sf::Color::White);
            frame.setColor(sf::Color::White);
            wheelL.setColor(sf::Color::White);
            wheelR.setColor(sf::Color::White);
      }//Красим при уроне
      return;
}

//Движение главного игрока
int  Player::update(float time, sf::String TileMap[HEIGHT_MAP], sf::Event event)
{
      if (!life)
      {
            return 0;
      }
      if (struckTimer > 0)
      {
            struckTimer -= time;
```

```
	}
	moveTimer += time;
	control(event);
	animation();
	switch (state)//тут делаются различные действия в зависимости от состояния
	{
	case right: dx = speed; dy = 0; break;//состояние идти вправо
	case left: dx = -speed; dy = 0; break;//состояние идти влево
	case up: dx = 0; dy = -speed; break;// состояние поднятия наверх
	case down: dx = 0; dy = speed; break;// состояние во время спуска
	case stay: dx = 0; dy = 0; break;//стоим
	}
	x += dx * time;
	checkCollisionWithMap(dx, 0, TileMap);//обрабатываем столкновение по X
	y += dy * time;
	checkCollisionWithMap(0, dy, TileMap);//обрабатываем столкновение по Y
	frame.setPosition(x + w / 2, y + h / 2); //задаем позицию спрайта
	wheelL.setPosition(x + w / 2, y + h / 2);
	wheelR.setPosition(x + w / 2, y + h / 2);
	gun.setPosition(x + w / 2, y + h / 2);
	if (health <= 0) { life = false; }
	speed = 0;
	return 0;
}

//Положение пушки
sf::Vector2f Player::GetgunXY()
{
	sf::Vector2f buf = gun.getPosition();
	//std::cout << "угол: " << gunrotation << std::endl;
	if (gunrotation >= -45 && gunrotation < 45)
	{
	    //верхний угол
	      buf.x += 80 * cos((gunrotation - 90) / 180 * 3.14159265) + (70/4);
	      buf.y += 80 * sin((gunrotation - 90) / 180 * 3.14159265) - (80/4);
	}
	else if (gunrotation >= 45 && gunrotation < 135)
	{
	    //правый
	      buf.x += 80 * cos((gunrotation - 90) / 180 * 3.14159265) - (80/4);
	      buf.y += 80 * sin((gunrotation - 90) / 180 * 3.14159265) + (80/4);
	}
	else if (gunrotation >= 135 && gunrotation < 225)
	{
	  //нижний
	      buf.x += 80 * cos((gunrotation - 90) / 180 * 3.14159265) - (80/2);
	      buf.y += 80 * sin((gunrotation - 90) / 180 * 3.14159265) - (70/4);
	}
	else if ((gunrotation >= 225 && gunrotation <= 270)||(gunrotation >= -90
&& gunrotation < -45))
	{
	  //левый
	      buf.x += 80 * cos((gunrotation - 90) / 180 * 3.14159265);
	      buf.y += 80 * sin((gunrotation - 90) / 180 * 3.14159265) - (80/2);
	}

	return buf;
}
```

```
//Окрашивание при нанесении урона
void  Player::struck(int damage)
{
	health -= damage;
	struckTimer = 200;//на 2мс в красный цвет
}

//отрисовка
void  Player::draw(sf::RenderTarget& target)
{
	target.draw(wheelL);
	target.draw(wheelR);
	target.draw(frame);
	target.draw(gun);

}
```