Курсовая работа на тему: «разработка игры Tower Defense»

Выполнил студент группы КЗ-33Б Галустов Г.Ю.

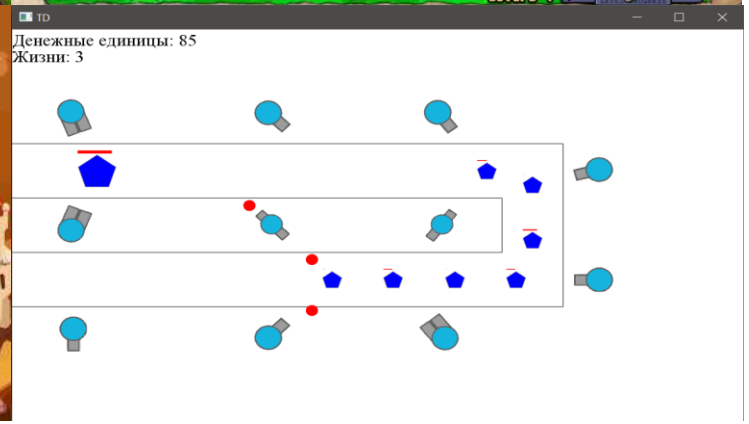
Цель и задачи курсовой работы

Целью данной курсовой работы является разработка игры жанра Tower Defense на языке C++ с использование методологии объектно-ориентированного программирования.

В соответствии с поставленной целью необходимо решить следующие задачи:

- Сделать обзор проекта и сравнить его с аналогичными программами.
- Разработать теоретическую базу для решения проблем проекта, которая должна включать в себя математические модели и алгоритмы.
- Найти, необходимые для реализации программного решения, сторонние библиотеки и кратко описать их.
- Реализовать программное решение с учетом продуманных технических и архитектурных решений, а в частности: использовать модульный подход, методологию объектно-ориентированного программирования, шаблоны проектирования и модульное тестирование.
- Описать структуру полученного проекта и использованные технические приемы.
- Составить инструкцию для пользователя

Обзор проекта



Математическая модель движения врага

$$\vec{v} = \frac{\vec{d}}{|\vec{d}|} * s$$

$\vec{v}$ – вектор движения в сторону точки с определенной скоростью;

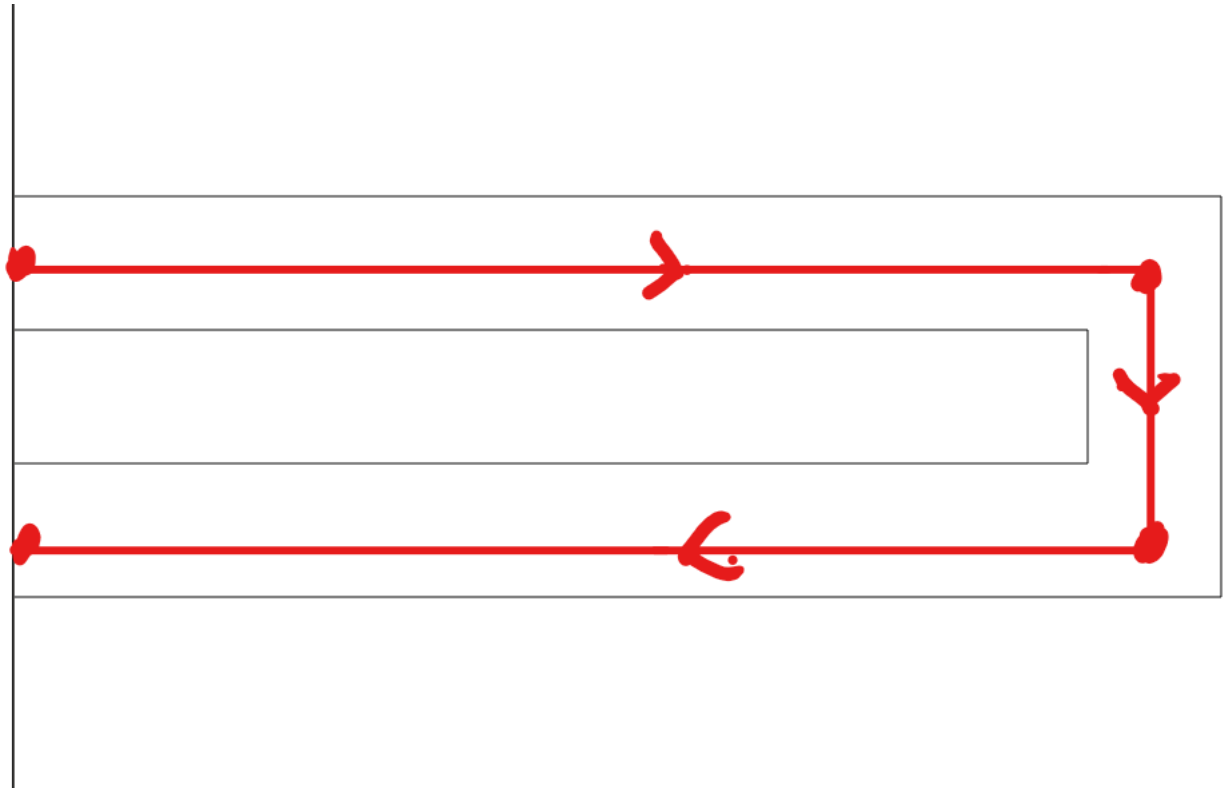
$\vec{d}$ – разность между позицией точки назначения и отправной точкой;

s – скорость движения;

$$\overrightarrow{w} = \begin{cases} \vec{v}, \overrightarrow{|v|} \leq \overrightarrow{|d|} \\ \vec{d}, \overrightarrow{|v|} > \overrightarrow{|d|} \end{cases}$$

$\overrightarrow{w}$ – вектор движения до точки с определенной скоростью;

Алгоритм движения врага



Реализация

```
void Enemy::move(float modifier)
{
	if ((int)path.size() - 1 < currPath) return;
	Vector2f distance = path[currPath] - shape.getPosition();
	Vector2f movement = normalize(path[currPath] - path[currPath - 1]) * speed *
modifier;
	if (len(movement) >= len(distance))
	{
		movement = distance;
		currPath++;
	}
	shape.move(movement);
	healthBar.move(movement);
}
```

Математическая модель для расчета столкновений

Столкновение игровых объектов можно рассчитывать, как пересечение геометрических фигур, покрывающих по размеру эти объекты. В этом случае удобно использовать либо прямоугольники, либо круги.

Определить пересекаются ли круг и прямоугольник можно, зная расстояние от центра до каждой из сторон прямоугольника. Формула для определения кратчайшего расстояния от точки до отрезка:

$$\vec{d} = \vec{v} + max(0, min(1, (\vec{p} - \vec{v}) * (\vec{w} - \vec{v}) \, / lenSqr)) * (\vec{w} - \vec{v})$$

где $\vec{d}$ – расстояние от точки до отрезка;

$\vec{v}$ – первая координата отрезка;

$\vec{w}$ – вторая координата отрезка;

$\vec{p}$ – координата точки;

lenSqr – квадрат длины отрезка;

Два круга пересекаются, если расстояние между их центрами меньше или равно сумме радиусов кругов:

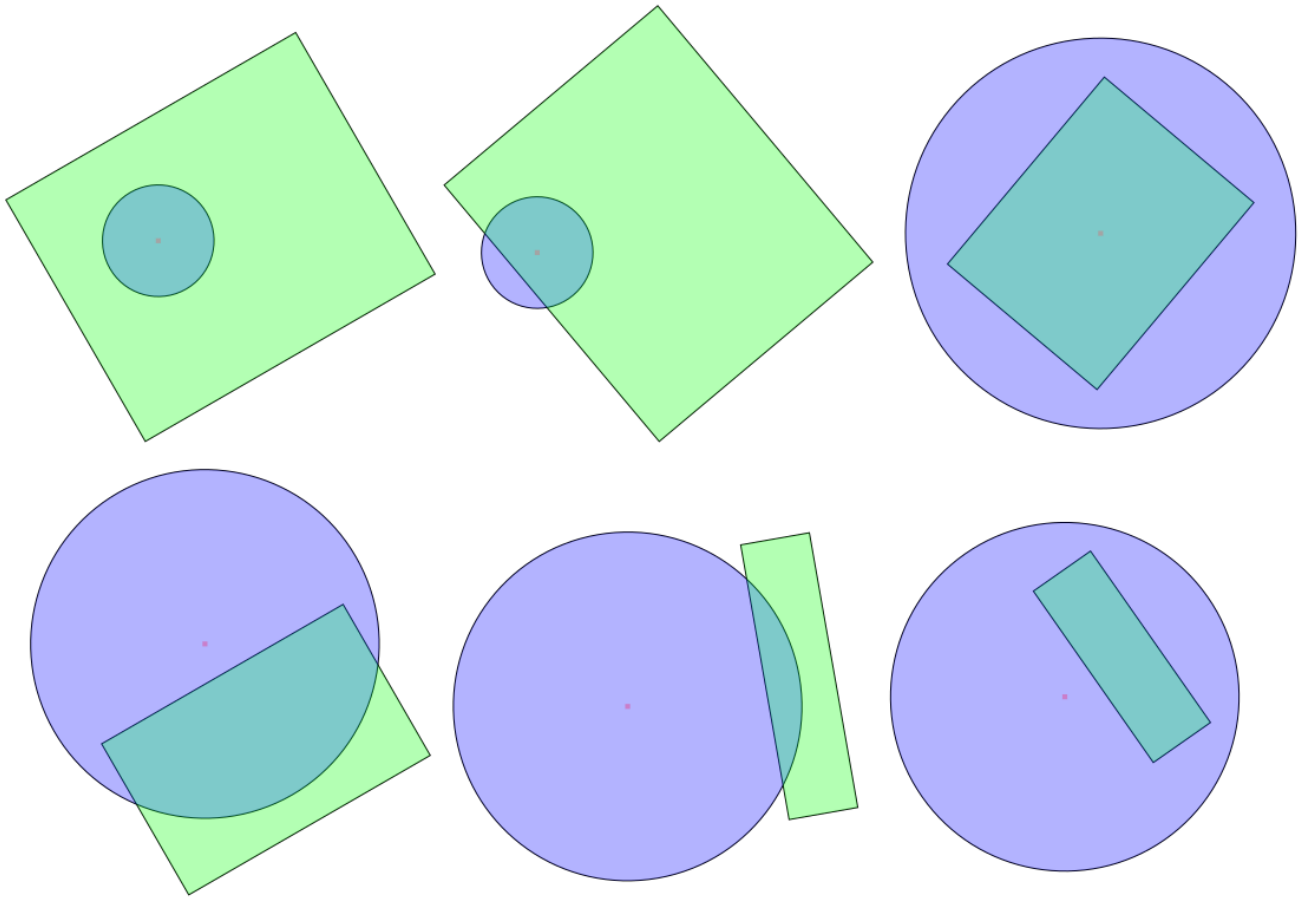
$$d \leq r_1 + r_2$$

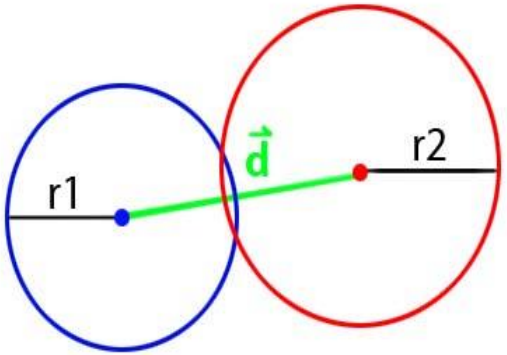
где d – расстояние между центрами кругов;

r_1 – радиус первого круга;

r_2 – радиус второго круга;

Алгоритмы расчета столкновений





Реализация расчета столкновения круга и прямоугольника

```
// Вовзращает дистанцию от точки до отрезка.
float distance(const Vector2f& p, const Vector2f (&lineSegment)[2])
{
	Vector2f p1 = lineSegment[0];
	Vector2f p2 = lineSegment[1];
	float lengthSqr = pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2);
	if (lengthSqr == 0) return len(p - p1);
	float t = dot(p - p1, p2 - p1) / lengthSqr;
	t = max(0.f, min(1.f, t));
	Vector2f projection = p1 + t * (p2 - p1);
	return len(p - projection);
}
```

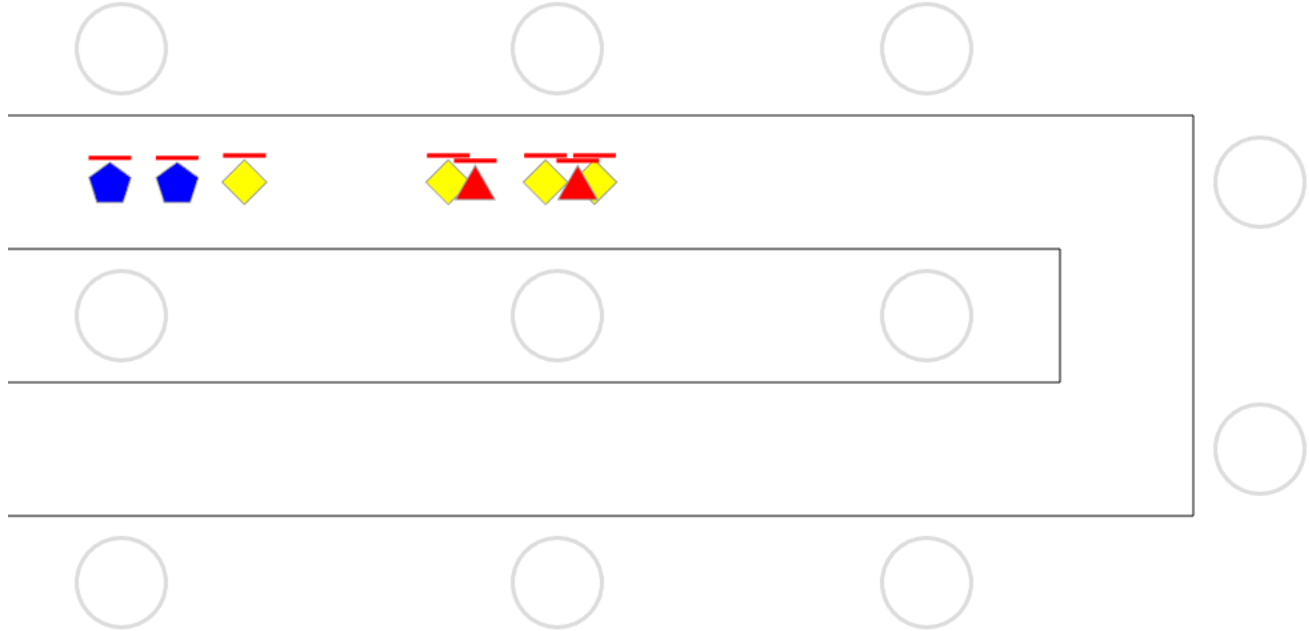
Реализация расчета столкновения круга и прямоугольника

```
// Возвращает истину, если круг и отрезок пересекаются.
bool intersects(const Vector2f& circlePos, float radius, const Vector2f (&lineSegment)[2])
{
	return distance(circlePos, lineSegment) <= radius;
}
// Возвращает истину, если круг и прямоугольник пересекаются.
bool intersects(const Vector2f& circlePos, float radius, const FloatRect& rect)
{
	Vector2f rectP1 = Vector2f(rect.left, rect.top);
	Vector2f rectP2 = Vector2f(rect.left + rect.width, rect.top);
	Vector2f rectP3 = Vector2f(rect.left + rect.width, rect.top + rect.height);
	Vector2f rectP4 = Vector2f(rect.left, rect.top + rect.height);
	return rect.contains(circlePos) ||
	intersects(circlePos, radius, { rectP1, rectP2 }) ||
	intersects(circlePos, radius, { rectP2, rectP3 }) ||
	intersects(circlePos, radius, { rectP3, rectP4 }) ||
	intersects(circlePos, radius, { rectP4, rectP1 });
}
```

Реализация расчета столкновения двух кругов

```
// Возвращает истину если два круга пересекаются.
bool intersects(const Vector2f& circlePos1, float
radius1, const Vector2f& circlePos2, float radius2)
{
    return len(circlePos2 - circlePos1) <= radius1 +
        radius2;
}
```

Алгоритм генерации врагов

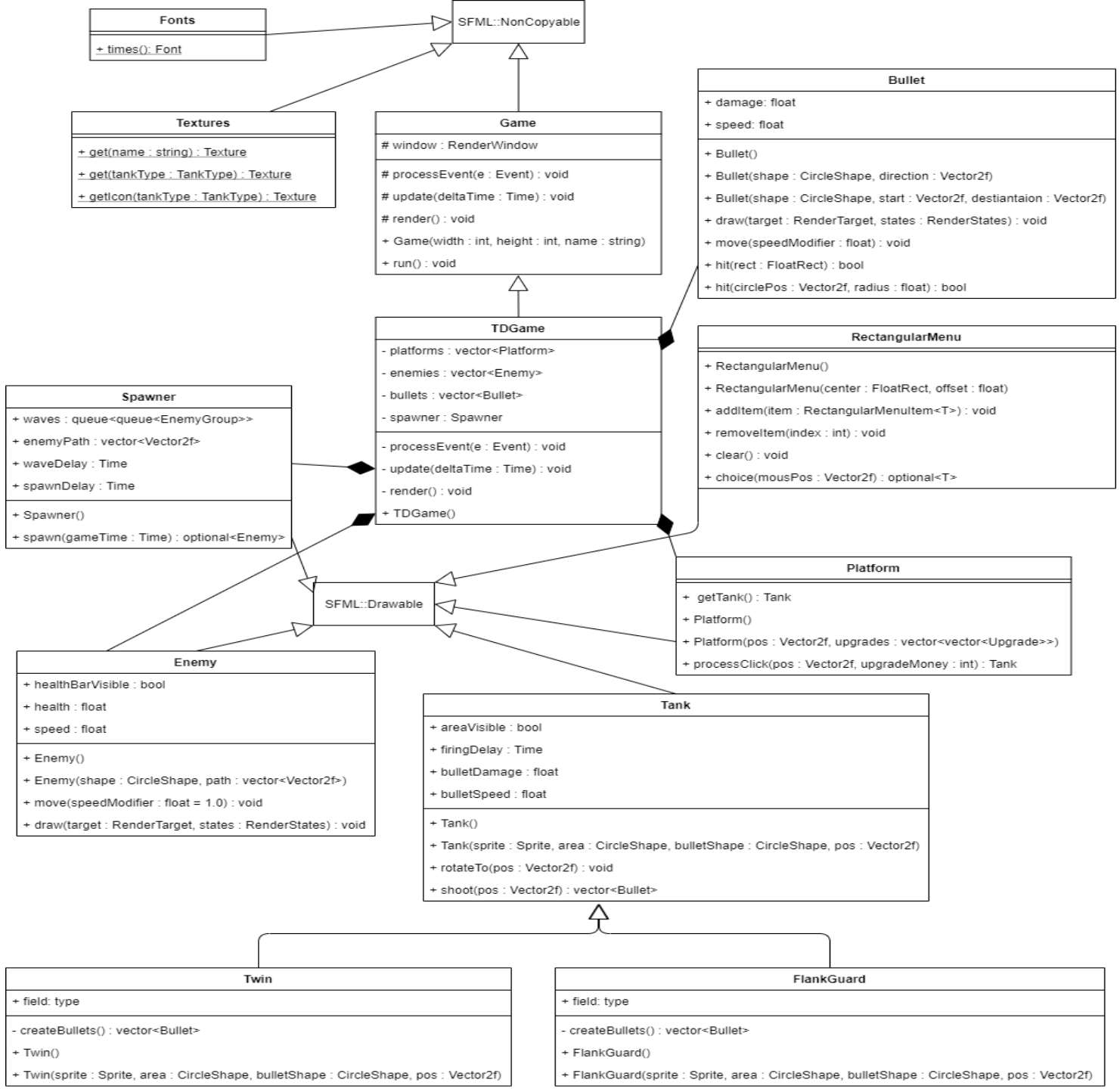


Реализация

```
optional<Enemy> Spawner::spawn(const Time& time)
{
            if (waves.empty()) return nullopt;
            Time delay = waveEnded ? waveDelay : spawnDelay;
            if (time - lastSpawn < delay) return nullopt;
            if (waves.front().empty())
            {
                        waves.pop();
                        waveEnded = true;
                        return spawn(time);
            }
            if (waves.front().front().count <= 0)
            {
                        waves.front().pop();
                        return spawn(time);
            }

            waves.front().front().count -= 1;
            lastSpawn = time;
            if (waveEnded) waveEnded = false;
            return createEnemy(waves.front().front().type);
}
```

Структура программы и использующиеся библиотеки


Fonts
+ times(): Font
SFML::NonCopyable
Textures
+ get(name : string) : Texture
+ get(tankType : TankType) : Texture
+ getIcon(tankType : TankType) : Texture
Game
window : RenderWindow
processEvent(e : Event) : void
update(deltaTime : Time) : void
render() : void
+ Game(width : int, height : int, name : string)
+ run() : void
Bullet
+ damage: float
+ speed: float
+ Bullet()
+ Bullet(shape : CircleShape, direction : Vector2f)
+ Bullet(shape : CircleShape, start : Vector2f, destiantaion : Vector2f)
+ draw(target : RenderTarget, states : RenderStates) : void
+ move(speedModifier : float) : void
+ hit(rect : FloatRect) : bool
+ hit(circlePos : Vector2f, radius : float) : bool
TDGame
- platforms : vector<Platform>
- enemies : vector<Enemy>
- bullets : vector<Bullet>
- spawner : Spawner
- processEvent(e : Event) : void
- update(deltaTime : Time) : void
- render() : void
+ TDGame()
RectangularMenu
+ RectangularMenu()
+ RectangularMenu(center : FloatRect, offset : float)
+ addItem(item : RectangularMenuItem<T>) : void
+ removeItem(index : int) : void
+ clear() : void
+ choice(mousPos : Vector2f) : optional<T>
Spawner
+ waves : queue<queue<EnemyGroup>>
+ enemyPath : vector<Vector2f>
+ waveDelay : Time
+ spawnDelay : Time
+ Spawner()
+ spawn(gameTime : Time) : optional<Enemy>
SFML::Drawable
Platform
+ getTank() : Tank
+ Platform()
+ Platform(pos : Vector2f, upgrades : vector<vector<Upgrade>>)
+ processClick(pos : Vector2f, upgradeMoney : int) : Tank
Enemy
+ healthBarVisible : bool
+ health : float
+ speed : float
+ Enemy()
+ Enemy(shape : CircleShape, path : vector<Vector2f>)
+ move(speedModifier : float = 1.0) : void
+ draw(target : RenderTarget, states : RenderStates) : void
Tank
+ areaVisible : bool
+ firingDelay : Time
+ bulletDamage : float
+ bulletSpeed : float
+ Tank()
+ Tank(sprite : Sprite, area : CircleShape, bulletShape : CircleShape, pos : Vector2f)
+ rotateTo(pos : Vector2f) : void
+ shoot(pos : Vector2f) : vector<Bullet>
Twin
+ field: type
- createBullets() : vector<Bullet>
+ Twin()
+ Twin(sprite : Sprite, area : CircleShape, bulletShape : CircleShape, pos : Vector2f)
FlankGuard
+ field: type
- createBullets() : vector<Bullet>
+ FlankGuard()
+ FlankGuard(sprite : Sprite, area : CircleShape, bulletShape : CircleShape, pos : Vector2f)

Паттерн шаблонный метод

```
vector<Bullet> Tank::shoot(const Vector2f& pos, const Time& currTime)
{
	vector<Bullet> bullets;
	if (currTime - lastShot < firingDelay) return bullets;
	lastShot = currTime;
	bullets = createBullets(pos);
	for (int i = 0; i < bullets.size(); i++)
	{
		bullets[i].setSpeed(bulletSpeed);
		bullets[i].setDamage(bulletDamage);
	}
	return bullets;
}
```

Паттерн шаблонный метод

```
vector<Bullet> Twin::createBullets(const Vector2f& destination)
{
	vector<Bullet> bullets;
	Vector2f offsetFromGun;
	offsetFromGun = Vector2f(-bulletShape.getRadius(), 0);
	bulletShape.setPosition(getBulletSpawnPos(bulletShape.getLocalBounds(), offsetFromGun));
	Bullet b1(bulletShape, bulletShape.getPosition(), destination);
	offsetFromGun = Vector2f(bulletShape.getRadius(), 0);
	bulletShape.setPosition(getBulletSpawnPos(bulletShape.getLocalBounds(), offsetFromGun));
	Bullet b2(bulletShape, bulletShape.getPosition(), destination);
	bullets.push_back(b1);
	bullets.push_back(b2);
	return bullets;
}
```

Паттерн одиночка и ленивая инициализация

```
const Texture& Textures::get(const String& name)
{
	static map<String, Texture> textures;
	if (!textures.contains(name))
	{
		Texture t;
		if (!t.loadFromFile(imagesFolder + name + imageExtension))
			cout << loadFailMsg << name.toAnsiString() << endl;
		textures[name] = t;
	}
	return textures[name];
}
```

Пример модульного тестирования

```
TEST_METHOD(testCircleAndLineIntersection)
{
	Vector2f circlePos(50, 50);
	float radius = 10.f;
	Vector2f line1[2] = { Vector2f(0, 0), Vector2f(100, 100) };
	Vector2f line2[2] = { Vector2f(40, 0), Vector2f(40, 100) };
	Vector2f line3[2] = { Vector2f(0, 0), Vector2f(0, 100) };
	Assert::IsTrue(intersects(circlePos, radius, line1));
	Assert::IsTrue(intersects(circlePos, radius, line2));
	Assert::IsFalse(intersects(circlePos, radius, line3));
}
```

Работа программы

Заключение