**Содержание**

[Введение 2](#_Toc29948424)

[1. Математические модели и алгоритмы 4](#_Toc29948425)

[1.1 Движение врага 5](#_Toc29948426)

[1.2 Расчет столкновений 6](#_Toc29948427)

[1.3 Генерация врагов 7](#_Toc29948428)

[2. Структура программы 8](#_Toc29948429)

[2.1 UML диаграмма 9](#_Toc29948430)

[2.2 Игровой цикл 10](#_Toc29948431)

[2.3 Защитные башни 11](#_Toc29948432)

[2.4 Враги 13](#_Toc29948433)

[2.5 Вспомогательные классы, структуры и перечисления 14](#_Toc29948434)

[2.6 Модуль Math 15](#_Toc29948435)

[2.7 Модульное тестирование 16](#_Toc29948436)

[Заключение 17](#_Toc29948437)

[Список литературы 18](#_Toc29948438)

Приложение А Основной код программы

Приложение Б Модульное тестирование

Приложение В Инструкция для пользователя

# **Введение**

Разработка игры жанра Tower Defense актуальна, т. к. множество игровых механизмов имеют применения в различных других сферах деятельности. Такой проект позволяет наглядно увидеть в действии различные математические модели и системы, а также архитектуру кода и программные решения для данных проблем.

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:

1. Сделать обзор проекта и сравнить его с аналогичными программами.
2. Разработать теоретическую базу для решения проблем проекта, которая должна включать в себя математические модели и алгоритмы.
3. Найти, необходимые для реализации программного решения, сторонние библиотеки и кратко описать их.
4. 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.
5. Описать структуру полученного проекта и использованные технические приемы.
6. Составить инструкцию для пользователя.

Tower Defense (с англ. — «Башенная защита»), сокращенно TD — название жанра компьютерных стратегических игр. Задача игрока в играх подобного жанра — расправиться с наступающими врагами, называемыми в некоторых играх «крипы», до того, как они пересекут карту, с помощью строительства башен, атакующих их, когда те проходят вблизи. Противники и башни обычно различаются по характеристикам и цене. Когда враги побеждены, игрок зарабатывает деньги или очки, которые используются для покупки или модернизации башен. Подбор вида башен и их расположение — неотъемлемая стратегия игры. Обычно враги пробегают через подобие лабиринта, что дает игроку возможность стратегического размещения башен, но также существуют известные версии игры, названные линейными TD, где используются прямые пути вместо лабиринтов. Один из самых ранних представителей TD — аркадная игра Rampart, перенесённая впоследствии на множество платформ. Игра стала причиной создания ряда карт игры «StarCraft», получивших название Turret Defense, которые в свою очередь вдохновили на создание карт Sunken и Hero Defense (Warcraft III). Такие модификации стали популярными среди игроков Warcraft III и Age of Empires II, затем укрепившись в виде отдельного жанра. Первым представителем отдельной игры в жанре Tower Defense стала компьютерная игра «Master of Defense», вышедшая 7 ноября 2005 года. Она стала очень популярна, получила награду «Стратегия года» от GameTunnel в 2006-м. Популярный образец жанра — GemCraft, флеш-игра, работающая в браузере. Жанр существует и на других платформах, таких, как мобильные телефоны (Town Defense, Tower Defense, Plants vs Zombies), PSP (VectorTD, Castle Rustle, Field Runner).

# **1. Математические модели и алгоритмы**

Для написания проекта используются различные математические модели и алгоритмы, которые делают возможным реализацию манипуляций с игровыми объектами в декартовой системе координат. Ключевым математическим объектом в такой системе является вектор. С помощью него можно задавать координаты игровых объектов, их направление, поэтому необходимо иметь возможность совершать широкое количество операций с ним.

Для оптимизации программы необходимо найти наиболее требовательные к производительности места и попытаться переосмыслить их логику, составив более удачный алгоритм. Этого можно достигнуть различными способами: применением алгоритма сортировки, использованием структур данных и т. д.

## **1.1 Движение врага**

Враг движется по карте, используя точки пути. Для движения от какой-либо точки к точке пути необходимо, в первую очередь, найти направление движения. Это можно сделать с помощью разности позиции точки пути и отправной точки. Затем, необходимо нормализовать полученный вектор, чтобы величина разности не влияла на скорость передвижения врага. Для регулировки скорости передвижения врага необходимо умножить результирующий вектор на число, равное скорости. Таким образом получена формула:

где – вектор движения в сторону точки с определенной скоростью;

– разность между позицией точки назначения и отправной точкой;

**–** скорость движения;

Однако, враг может двинуться дальше точки пути, если вектор движения слишком большой. Следовательно, необходимо ограничить движение врага до точки назначения:

где – вектор движения до точки с определенной скоростью;

Используя данную формулу, можно построить алгоритм движения врага по точкам пути. Данный алгоритм представлен в приложении А №1 в методе move.

## **1.2 Расчет 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:

где – расстояние от точки до отрезка;

– первая координата отрезка;

– вторая координата отрезка;

– координата точки;

lenSqr – квадрат длины отрезка;

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

где – расстояние между центрами кругов;

– радиус первого круга;

– радиус второго круга;

Реализацию данных формул можно найти в приложении A №10 в функциях distance и intersects.

## **1.3 Генерация врагов**

Генерация волн врагов является одной из самых ресурсоемких задач курсового проекта. Важно максимально оптимизировать этот процесс, при этом сохранить относительную простоту кода.

Прежде всего нужно продумать каким образом хранить последовательность волн врагов. Если хранить врагов напрямую в массивах, то это приведет к использованию большого количества памяти. Возможным решением данной проблемы является использование перечисляемого типа, который будет обозначать тип врага, которого надо сгенерировать. Далее, можно еще больше уменьшить количество занимаемой памяти, если хранить не последовательность типов врагов, а структуры с типом врага и количеством врагов данного типа, иначе говоря - группы врагов.

Следующим шагом необходимо продумать, как обходить массивы и генерировать врагов. Для данной задачи подходит такая структура данных, как очередь. Сначала проверяется пуста ли очередь с волнами. Если она не пуста, то проверяется очередь с группами врагов. Если она пуста, то очередь с волнами уменьшается на 1 элемент и алгоритм начинается заново. Если не пуста, то для первой группы врагов проверяется их количество. Если оно меньше или равно нулю, то очередь с группами врагов уменьшается на 1 и алгоритм начинается заново. Если больше нуля, то по первой группе врагов генерируется враг и количество врагов в группе уменьшается на 1.

Реализацию алгоритма можно найти в приложении А №5 в методе spawn.

# **2. Структура программы**

Для реализации программы преимущественно используется методология объектно-ориентированного программирования.

Для взаимодействия с различными мультимедийными компонентами на компьютерах используется готовая кроссплатформенная библиотека SFML, предоставляющая для этого простой программный интерфейс (API). Написана на C++, но также доступна для C, Crystal, D, Euphoria, Go, Java, Julia, .NET, Nim, OCaml, Python, Ruby и Rust. Библиотека SFML способна создавать окна и обрабатывать для них пользовательский ввод, а также создавать контексты OpenGL и управлять ими. Также SFML предоставляет графический модуль для простого аппаратного ускорения 2D-компьютерной графики, который включает в себя визуализацию текста с использованием FreeType, аудио-модуль, использующий OpenAL, и сетевой модуль для обмена данными по базовому протоколу TCP и протоколу UDP.

Для модульного тестирования C++ кода используется фреймворк CppUnitTestFramework. Он предоставляет базовые инструменты тестирования, такие как: утверждения, инициализация перед тестами и «очистка» после тестов, а также различные атрибуты для работы с тестами.

Весь основной код можно найти в приложении А.

## **2.1 UML диаграмма**

Ниже представлена UML диаграмма основных классов проекта.

Изображение выглядит как снимок экрана, монитор

Автоматически созданное описание

## **2.2 Игровой цикл**

Класс Game является стартовой точкой игры. Он содержит защищенное поле с окном игры, защищенные виртуальные методы: processEvent для обработки событий пользовательского ввода, update для обновления состояния игры и render для вывода графического представления игры в окно. Также класс содержит публичный метод run, который запускает цикл игры.

От класса Game наследуется класс TDGame, который отвечает за игровую логику проекта. Он содержит все игровые объекты: массив с защитными башнями, расположенными на карте, массив с врагами, присутствующими в игре и массив с пулями, находящимися в движении. Также класс переопределяет унаследованные от класса Game методы: processEvent, update, render. Метод processEvent используется для оповещения игровых объектов о каком-либо пользовательском вводе. В методе update находится игровая логика, а именно: генерация новых врагов, движение врагов, стрельба танков, столкновение пуль с врагами. Метод render выводит графическое изображение игры в окно. Также класс контролирует состояние проигрыша или выигрыша игрока и различные игровые показатели.

## **2.3 Защитные башни**

Основным объектом в играх жанра Tower Defense является защитная башня. Обычно, она может стрелять чем-либо во врагов и ее можно строить или улучшать за игровую валюту. В данном проекте башню описывает класс Tank и, соответственно, башня условно называется танком.

Класс Tank является базовым для всех остальных танков. Он содержит графическое представление танка, область видимости врагов, задержку стрельбы, урон от пуль и скорость пуль. Также класс содержит виртуальный метод rotateTo, который вращает танк в направлении какой-либо точки и виртуальный метод shoot, который содержит логику стрельбы танка. Для уменьшения повторного использования кода в классе Tank определен защищенный виртуальный метод createBullets, который наследующиеся классы могут переопределить, для сохранения базовой логики стрельбы, но с переопределением генерации пуль. Таким образом, применяется шаблон проектирования – шаблонный метод. Класс наследуется от абстрактного класса Drawable и переопределяет метод draw, для отображения графического представления танка и его области видимости, если установлен соответствующий флаг показа этой области. Реализацию класса можно найти в приложении А №2

От базового класса наследуются классы Twin и FlankGuard, которые переопределяют метод createBullets для генерации двух пуль, вместо одной и изменения направления движения пуль. Таким образом, можно создать огромное количество различных танков с наименьшим количеством изменений в логике базового танка.

Класс RectangularMenu представляет собой квадратное меню, которое состоит из максимум 4 ячеек. Класс является шаблонным и хранит массив с графическими представлениями ячейки и значения обобщенного типа. Для добавления, удаления и очищения ячеек реализованы методы addItem, removeItem и clear соответственно. Метод choice возвращает значение обобщенного типа для выбранной ячейки, если меню видимо и координаты нажатия пользователя попадают на одну из ячеек. Класс наследуется от абстрактного класса Drawable и переопределяет метод draw для вывода графического представления меню, если установлен флаг показа меню. В игре данный класс используется для предоставления меню улучшений танка пользователю.

Класс Platform используется для обозначения места для постройки танка и улучшений танка. Основным методом, который реализует класс является метод processClick, который обрабатывает нажатие на платформу и показывает меню улучшений и область видимости танка или улучшает танк, если координаты нажатия пользователя попадают на один из вариантов меню улучшений и игрок имеет достаточное количество денежных единиц для улучшения. Также класс наследуется от абстрактного класса Drawable для вывода графического представления танка или платформы (если танк отсутствует) и меню.

Класс Bullet описывает пулю танка. Он содержит геометрическую форму пули, направление движения, скорость и урон. Основным методом класса является метод move, который передвигает пулю в заданном направлении движения и скоростью. Также класс реализует две перегрузки метода hit, который определяется столкновения пули с прямоугольником или кругом. Данный класс наследуется от абстрактного класса Drawable и переопределяет метод draw, для вывода графического представления пули.

## **2.4 Враги**

В играх жанра Tower Defense враги идут по определенному маршруту, который, в конечном счете, приводит их к месту назначения и игрок теряет жизнь, если не успевает их уничтожить. Враги имеют определенный запас здоровья и скорость передвижения по маршруту.

В программе врага описывает класс Enemy. Он содержит путь, состоящий из точек, по которым должен двигаться враг, скорость передвижения, количество жизней и геометрическую форму врага. Также класс содержит полоску жизней, которая представлена в виде прямоугольника, который меняет свой размер в соответствии с соотношением максимального и текущего значения здоровья врага. Основным методом в классе является move. Данный метод на основе полученной скорости двигает врага в сторону текущей точки пути и автоматически меняет точку пути на следующую, если враг достигает текущую точку. Для вывода графического представления врага класс Enemy наследуется от абстрактного класса Drawable, который содержит метод draw. Данный метод переопределяется в классе и выводит соответственно врага и полоску здоровья. Различные типы врагов достигаются путем композиции, а не наследования. Так, в игре представлено 4 типа врагов: треугольник, квадрат, пятиугольник и альфа-пятиугольник. Они имеют разное графическое представление и параметры, но при этом являются экземплярами одного класса. Реализацию класса можно найти в приложении А №1.

Класс Spawner предназначен для генерации волн врагов. Он содержит массив с волнами врагов, который в свою очередь содержит массив с типами врагов и их количеством. Также класс содержит время задержки генерации новой волны и время задержки генерации нового врага. Основным методом класса является метод spawn. Данный метод возвращает нового врага, если с последнего момента времени генерации прошло больше времени задержки генерации (волны или врага).

## **2.5 Вспомогательные классы, структуры и перечисления**

Класс Textures содержит статический метод get для ленивой загрузки текстур из файловой системы. Также применяется паттерн одиночка для того, чтобы не загружать уже загруженные текстуры.

Перечисление EnemyType содержит возможные виды врагов: треугольник, квадрат, пятиугольник, альфа-пятиугольник. Данное перечисление используется в структуре EnemyGroup, которая содержит тип врага и их количество и используется для задания волн врагов и их последующей генерации в классе Spawner.

Перечисление TankType содержит возможные виды танков: обычный танк, боковой защитник, близнец. Данное перечисление используется в структуре Upgrade, которая содержит тип танка и его стоимость и используется в классе UpgradePlatform для улучшения танков.

Структура RectangularMenuItem используется в классе RectangularMenu и описывает отдельную ячейку меню. В структуре содержится графическое представление ячейки и значение, которое возвратит меню при выборе одной из ячеек.

## **2.6 Модуль Math**

В модуле Math содержатся дополнительные математические функции для проекта. Большинство из них содержат операции с векторами. Так, в модуле содержатся функции для получения длины вектора, нормализованного вектора, угла поворота вектора в пространстве по часовой стрелке в радианах, а также функция для получения скалярного произведения векторов.

В данном модуле присутствует большое количество функция для определения пересечения геометрических фигур, которые используется для определения столкновений игровых объектов. Так, здесь содержатся функции для определения расстояния от точки до отрезка, для определения пересечения точки с кругом, отрезка с кругом, двух кругов, прямоугольника и круга.

Также есть вспомогательная функция для перевода радиан в градусы, округления координат вектора и получения координат центра прямоугольника.

Реализацию модуля можно найти в приложении А №10.

## **2.7 Модульное тестирование**

Были протестированы основные классы: Tank, Enemy, Bullet, а также модуль Math, так как он используется во многих местах программы и класс RectangularMenu из-за большого количества возможных нестандартных случаев его использования. Все классы были протестированы на работу без ошибок в случае пустого конструктора, т. е. работы без параметров.

Класс Tank был протестирован на правильность работы метода shoot, который должен возвращать пули с учетом задержки стрельбы. В классе Enemy был проверен метод move, который должен двигать врага по точкам пути, при этом враг должен двигаться с постоянной скоростью и при этом не двинуться дальше этой точки, даже если скорость будет стремиться к бесконечности. В классе Bullet был протестирован метод move, который должен двигать пулю в определенном направлении, тестировать метод hit нет необходимости, т. к. внутри метода используются функции модуля Math, которые протестированы отдельно. В классе RectangularMenu были протестированы методы добавления, удаления и очищения меню, при этом количество элементов никогда не должно превышать четырех, также был протестирован метод choice, который должен возвращать значение в случае нажатия на одну из ячеек меню, причем только в случае, если установлен флаг видимости меню. Также были протестированы все функции модуля Math для различных параметров, чтобы исключить возможность скрытой ошибки, которую будет сложно отследить в классах, использующих этот модуль.

Реализация модульного тестирования находится в приложении Б.

# **Заключение**

Таким образом, реализованная игра в жанре Tower Defense решает различные проблемы, которые иногда косвенно связаны с самой игрой, но актуальны во многих других областях. Так, класс RectangularMenu можно использовать в различных пользовательских интерфейсах, модуль Math для последующего использования в других проектах и т. д.

Для реализации проекта была изучена различная научная и техническая литература, реализованы математические модели и алгоритмы для решения отдельных проблем, использованы различные паттерны проектирования и технические приемы, помогающие уменьшить связность кода, повысить производительность программы, увеличить количество пере используемого кода.

Что можно улучшить дальше? Алгоритм поиска столкновений зоны видимости танка с врагом могут работать быстрее, если применить алгоритм быстрой сортировки, сравнивая расстояние танка до врага, однако такой алгоритм приведет к тому, что танк будет стрелять в самого ближнего врага, хотя в играх данного жанра танк обычно стреляет в последнего врага. Для дальнейшего развития игры необходимо сделать редактор уровней, больше улучшений танков, меню, систему сохранения и разнообразить игровой процесс.

Инструкция для пользователя представление в приложении В.

# **Список литературы**

1. Страуструп Б. Программирование: принципы и практика использования C++, испр. изд.: Пер. с англ. — М.: ООО “И. Д. Вильямс”, 2011. — 1248 с.
2. Кент Б. Экстремальное программирование. Разработка через тестирование. Пер. с англ.: ООО Издательство «Питер», 2017. – 260с.
3. Фримен Э., Робсон Э., Сьерра К., Бейтс Б. Head First. Паттерны проектирования. Обновленное юбилейное издание. – СПб.: Питер, 2018. – 656 с.
4. Tower Defense [Электронный ресурс] : Материал из Википедии — свободной энциклопедии : Версия 104308942, сохранённая в 12:48 UTC 2 января 2020 / Авторы Википедии // Википедия, свободная энциклопедия. — Электрон. дан. — Сан-Франциско: Фонд Викимедиа, 2020. — Режим доступа: https://ru.wikipedia.org/?oldid=104308942
5. Расстояние от точки до прямой на плоскости [Электронный ресурс] : Материал из Википедии — свободной энциклопедии : Версия 102973494, сохранённая в 03:12 UTC 27 октября 2019 / Авторы Википедии // Википедия, свободная энциклопедия. — Электрон. дан. — Сан-Франциско: Фонд Викимедиа, 2019. — Режим доступа: https://ru.wikipedia.org/?oldid=102973494

**Приложение А №1**

**Основной код программы (класс Enemy)**

Заголовочный файл:

class Enemy : public Drawable

{

private:

CircleShape shape;

float speed;

float health;

float maxHealth;

vector<Vector2f> path;

int currPath;

RectangleShape healthBar;

Vector2f originalHealthBarSize;

public:

bool healthBarVisible = false;

float getSpeed() const;

const Vector2f& getPosition() const;

float getRadius() const;

float getHealth() const;

float getMaxHealth() const;

const Shape& getShape();

void setSpeed(float speed);

void setHealth(float health);

Enemy();

Enemy(const CircleShape& shape, const vector<Vector2f>& path);

virtual void draw(RenderTarget& target, RenderStates states) const;

virtual void move(float speedModifier = 1.f);

};

Исходный файл:

float Enemy::getSpeed() const { return speed; }

const Vector2f& Enemy::getPosition() const { return shape.getPosition(); }

float Enemy::getRadius() const { return shape.getRadius(); }

float Enemy::getHealth() const { return health; }

float Enemy::getMaxHealth() const { return maxHealth; }

void Enemy::setSpeed(float s) { speed = s; }

void Enemy::setHealth(float h)

{

if (h > maxHealth) maxHealth = h;

health = h;

float percent = health / maxHealth;

healthBar.setSize(originalHealthBarSize \* percent);

}

Enemy::Enemy() : Enemy(CircleShape(), {}) {}

Enemy::Enemy(const CircleShape& c, const vector<Vector2f>& p) : shape(c), path(p), currPath(0), speed(0), health(0), maxHealth(0),

healthBar(Vector2f(c.getRadius() \* 2, shape.getRadius() / 5)), originalHealthBarSize(healthBar.getSize())

{

shape.setOrigin(center(shape.getLocalBounds()));

healthBar.setOrigin(center(healthBar.getLocalBounds()));

healthBar.setFillColor(Color::Red);

if (path.empty()) return;

shape.setPosition(path[0]);

healthBar.setPosition(shape.getPosition());

healthBar.move(0, -healthBar.getLocalBounds().height / 2 - shape.getLocalBounds().height / 2 - healthBarOffsetFromEnemy);

currPath++;

}

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);

}

void Enemy::draw(sf::RenderTarget& target, sf::RenderStates states) const

{

target.draw(shape, states);

if (healthBarVisible) target.draw(healthBar, states);

}

**Приложение А №2**

**Основной код программы (класс Tank)**

Заголовочный файл:

class Tank : public Drawable {

protected:

Sprite sprite;

CircleShape area; // Область видимости танка.

Time firingDelay; // Задержка стрельбы.

Time lastShot; // Время последного выстрела.

CircleShape bulletShape;

float bulletDamage;

float bulletSpeed;

Vector2f getBulletSpawnPos(const FloatRect& bulletRect, const Vector2f& offsetFromGun);

virtual vector<Bullet> createBullets(const Vector2f& destination);

public:

bool areaVisible = false;

const Vector2f& getPosition() const;

float getAreaRadius() const;

const Time& getFiringDelay() const;

float getBulletDamage() const;

float getBulletSpeed() const;

void setFiringDelay(const Time& firingDelay);

void setBulletDamage(float damage);

void setBulletSpeed(float speed);

Tank();

Tank(const Sprite& sprite, const CircleShape& area, const CircleShape& bulletShape, const Vector2f& pos);

virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;

virtual void rotateTo(const Vector2f& point);

virtual vector<Bullet> shoot(const Vector2f& pos, const Time& currentTime);

};

Исходный файл:

const Vector2f& Tank::getPosition() const { return area.getPosition(); }

float Tank::getAreaRadius() const { return area.getRadius(); }

const Time& Tank::getFiringDelay() const { return firingDelay; }

float Tank::getBulletDamage() const { return bulletDamage; }

float Tank::getBulletSpeed() const { return bulletSpeed; }

void Tank::setFiringDelay(const Time& fd) { firingDelay = fd; }

void Tank::setBulletDamage(float damage) { bulletDamage = damage; }

void Tank::setBulletSpeed(float speed) { bulletSpeed = speed; }

Tank::Tank() : Tank(Sprite(), CircleShape(), CircleShape(), Vector2f(0, 0)) {}

Tank::Tank(const Sprite& s, const CircleShape& a, const CircleShape& bs, const Vector2f& p) : sprite(s), area(a),

bulletShape(bs), firingDelay(), lastShot(), bulletDamage(), bulletSpeed(), areaVisible()

{

sprite.setOrigin(center(sprite.getLocalBounds()));

sprite.setPosition(p);

area.setOrigin(center(area.getLocalBounds()));

area.setPosition(p);

}

Vector2f Tank::getBulletSpawnPos(const FloatRect& rect, const Vector2f& offset)

{

Transform t;

t.rotate(sprite.getRotation(), sprite.getPosition());

Vector2f point = sprite.getPosition() + Vector2f(0, sprite.getLocalBounds().height / 2) + Vector2f(0, rect.height / 2) + offset;

return t.transformPoint(point);

}

void Tank::draw(sf::RenderTarget& target, sf::RenderStates states) const {

target.draw(sprite, states);

if (areaVisible) target.draw(area, states);

}

void Tank::rotateTo(const Vector2f& p)

{

Vector2f distance = p - sprite.getPosition();

float angle = degrees(rotation(distance)) - 90;

sprite.setRotation(angle);

}

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;

}

vector<Bullet> Tank::createBullets(const Vector2f& destination)

{

vector<Bullet> bullets;

Vector2f offsetFromGun = Vector2f(0, 0);

bulletShape.setPosition(getBulletSpawnPos(bulletShape.getGlobalBounds(), offsetFromGun));

Bullet b(bulletShape, sprite.getPosition(), destination);

bullets.push\_back(b);

return bullets;

}

**Приложение А №3**

**Основной код программы (классы Twin и FlankGuard)**

Заголовочные файлы:

class Twin : public Tank

{

protected:

virtual vector<Bullet> createBullets(const Vector2f& destination);

public:

Twin();

Twin(const Sprite& sprite, const CircleShape& area, const CircleShape& bulletShape, const Vector2f& pos);

};

class FlankGuard : public Tank

{

protected:

virtual vector<Bullet> createBullets(const Vector2f& destination);

public:

FlankGuard();

FlankGuard(const Sprite& sprite, const CircleShape& area, const CircleShape& bulletShape, const Vector2f& pos);

};

Исходные файлы:

Twin::Twin() : Tank() {}

Twin::Twin(const Sprite& sprite, const CircleShape& area, const CircleShape& bulletShape, const Vector2f& pos) :

Tank(sprite, area, bulletShape, pos) {}

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;

}

FlankGuard::FlankGuard() : Tank() {}

FlankGuard::FlankGuard(const Sprite& sprite, const CircleShape& area, const CircleShape& bulletShape, const Vector2f& pos) :

Tank(sprite, area, bulletShape, pos) {}

vector<Bullet> FlankGuard::createBullets(const Vector2f& destination)

{

vector<Bullet> bullets;

Vector2f offsetFromGun;

offsetFromGun = Vector2f(0, 0);

bulletShape.setPosition(getBulletSpawnPos(bulletShape.getLocalBounds(), offsetFromGun));

Bullet b1(bulletShape, bulletShape.getPosition(), destination);

offsetFromGun = Vector2f(0, -sprite.getLocalBounds().height - bulletShape.getRadius());

bulletShape.setPosition(getBulletSpawnPos(bulletShape.getLocalBounds(), offsetFromGun));

Bullet b2(bulletShape, -normalize(destination - bulletShape.getPosition()));

bullets.push\_back(b1);

bullets.push\_back(b2);

return bullets;

}

**Приложение А №4**

**Основной код программы (класс Bullet)**

Заголовочный файл:

class Bullet : public Drawable

{

private:

CircleShape shape;

Vector2f direction;

float speed;

float damage;

public:

const Vector2f& getPosition() const;

float getRadius() const;

float getSpeed() const;

float getDamage() const;

void setSpeed(float speed);

void setDamage(float damage);

Bullet();

Bullet(const CircleShape& shape, const Vector2f& direction);

Bullet(const CircleShape& shape, const Vector2f& start, const Vector2f& destination);

virtual void draw(RenderTarget& target, RenderStates states) const;

virtual void move(float speedModifier = 1.f);

bool hit(const FloatRect& rect);

bool hit(const Vector2f& circlePos, float radius);

};

Исходный файл:

const Vector2f& Bullet::getPosition() const { return shape.getPosition(); }

float Bullet::getRadius() const { return shape.getRadius(); }

float Bullet::getSpeed() const { return speed; }

float Bullet::getDamage() const { return damage; }

void Bullet::setSpeed(float s) { speed = s; }

void Bullet::setDamage(float d) { damage = d; }

Bullet::Bullet() : Bullet(CircleShape(), Vector2f()) {}

Bullet::Bullet(const CircleShape& c, const Vector2f& direction) :

shape(c), direction(direction), speed(0), damage(0){}

Bullet::Bullet(const CircleShape& c, const Vector2f& start, const Vector2f& dest) :

Bullet(c, normalize(dest - start)){ }

void Bullet::draw(RenderTarget& target, RenderStates states) const

{

target.draw(shape);

}

void Bullet::move(float modifier)

{

shape.move(direction \* speed \* modifier);

}

bool Bullet::hit(const FloatRect& rect)

{

return intersects(shape.getPosition(), shape.getRadius(), rect);

}

bool Bullet::hit(const Vector2f& circlePos, float radius)

{

return intersects(circlePos, radius, shape.getPosition(), shape.getRadius());

}

**Приложение А №5**

**Основной код программы (класс Spawner)**

Заголовочный файл:

class Spawner

{

private:

Time lastSpawn;

bool waveEnded = false;

Enemy createEnemy(const EnemyType& type);

public:

queue<queue<EnemyGroup>> waves;

vector<Vector2f> enemyPath;

Time spawnDelay;

Time waveDelay;

Spawner();

optional<Enemy> spawn(const Time& gameTime);

};

Исходный файл:

Spawner::Spawner() : lastSpawn(), spawnDelay(), waveDelay() {}

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);

}

Enemy Spawner::createEnemy(const EnemyType& type)

{

CircleShape shape;

Color color;

Enemy e;

float speed(0.f);

float health(0.f);

switch (type)

{

case EnemyType::rectangle:

shape = CircleShape(rectRadius, rectVertexCount);

color = Color::Yellow;

speed = rectSpeed;

health = rectHealth;

break;

case EnemyType::triangle:

shape = CircleShape(triangleRadius, trinagleVertexCount);

color = Color::Red;

speed = triangleSpeed;

health = triangleHealth;

break;

case EnemyType::pentagon:

shape = CircleShape(pentagonRadius, pentagonVertexCount);

color = Color::Blue;

speed = pentagonSpeed;

health = pentagonHealth;

break;

case EnemyType::alphaPentagon:

shape = CircleShape(alphaPentagonRadius, pentagonVertexCount);

color = Color::Blue;

speed = alphaPentagonSpeed;

health = alphaPentagonHealth;

break;

}

shape.setFillColor(color);

shape.setOutlineColor(enemiesOutlineColor);

shape.setOutlineThickness(enemiesOutlineThickness);

e = Enemy(shape, enemyPath);

e.setSpeed(speed);

e.setHealth(health);

e.healthBarVisible = true;

return e;

}

**Приложение А №6**

**Основной код программы (класс RectangularMenu)**

template<typename T>

class RectangularMenu : public Drawable

{

private:

vector<RectangularMenuItem<T>> items;

FloatRect rect;

float offset;

public:

bool visible;

const vector<RectangularMenuItem<T>>& getItems() const { return items; }

RectangularMenu() : RectangularMenu(FloatRect(), 0.f) {}

RectangularMenu(const FloatRect& r, float offset) : rect(r), offset(offset), visible(false) {}

virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const

{

if (!visible) return;

for (int i = 0; i < items.size(); i++) target.draw(items[i].icon, states);

}

void addItem(const RectangularMenuItem<T>& item)

{

if (items.size() > 3) return;

RectangularMenuItem<T> i = item;

i.icon.setOrigin(center(i.icon.getGlobalBounds()));

i.icon.setPosition(center(rect));

Vector2f verticalOffset = Vector2f(0, rect.height / 2 + i.icon.getLocalBounds().height / 2 + offset);

Vector2f horizontalOffset = Vector2f(rect.width / 2 + i.icon.getLocalBounds().width / 2 + offset, 0);

switch (items.size())

{

case 0:

i.icon.move(-verticalOffset);

break;

case 1:

i.icon.move(horizontalOffset);

break;

case 2:

i.icon.move(verticalOffset);

break;

case 3:

i.icon.move(-horizontalOffset);

break;

default:

break;

}

items.push\_back(i);

}

void removeItem(int index) { items.erase(items.begin() + index); }

void clear() { items.clear(); }

optional<T> choice(const Vector2f& mouseDownPos)

{

if (!visible) return nullopt;

optional<T> c = nullopt;

for (int i = 0; i < items.size(); i++)

if (items[i].icon.getGlobalBounds().contains(mouseDownPos)) c = items[i].choice;

return c;

}

};

**Приложение А №7**

**Основной код программы (класс Textures)**

Заголовочный файл:

class Textures {

public:

static const Texture& get(const String& name);

static const Texture& get(const TankType& tankType);

static const Texture& getIcon(const TankType& tankType);

};

Исходный файл:

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];

}

const Texture& Textures::get(const TankType& type)

{

switch (type)

{

case TankType::tank:

return Textures::get("tank");

case TankType::twin:

return Textures::get("twin");

case TankType::flankGuard:

return Textures::get("flankGuard");

default:

return Texture();

}

}

const Texture& Textures::getIcon(const TankType& type)

{

switch (type)

{

case TankType::tank:

return Textures::get("tankicon");

case TankType::twin:

return Textures::get("twinIcon");

case TankType::flankGuard:

return Textures::get("flankGuardIcon");

default:

return Texture();

}

}

**Приложение А №8**

**Основной код программы (класс Game)**

Заголовочный файл:

class Game : public NonCopyable

{

protected:

RenderWindow window;

virtual void processEvent(const Event& e);

virtual void update(const Time& deltaTime);

virtual void render();

public:

Game(int width, int height, const string& name);

void run();

};

Исходный файл:

Game::Game(int width, int height, const string& name)

{

ContextSettings settings;

settings.antialiasingLevel = antialiasingLevel;

window.create(VideoMode(width, height), name, Style::Default, settings);

window.setFramerateLimit(FPS);

}

void Game::run()

{

Clock clock;

while (window.isOpen())

{

Event e;

while (window.pollEvent(e)) processEvent(e);

update(clock.restart());

render();

}

}

void Game::processEvent(const Event& e)

{

if (e.type == Event::Closed) window.close();

}

void Game::update(const Time& deltaTime)

{

}

void Game::render()

{

}

**Приложение А №9**

**Основной код программы (класс TDGame)**

Заголовочный файл:

class TDGame : public Game

{

private:

const FloatRect bounds;

vector<Platform> platforms;

vector<Enemy> enemies;

vector<Bullet> bullets;

Spawner spawner;

VertexArray upperPathBound;

VertexArray lowerPathBound;

Time gameTime;

int money, lives;

bool defeat, victory;

Text moneyText, livesText, endGameText;

void processEvent(const Event& e);

void update(const Time& deltaTime);

void render();

public:

TDGame();

};

Исходный файл:

#include "TDGame.h"

TDGame::TDGame() : Game(gameWidth, gameHeight, gameName), bounds(0, 0, gameWidth, gameHeight), money(initialMoney), lives(initialLives),

defeat(false), victory(false), moneyText(moneyInfo, Fonts::times()), livesText(livesInfo, Fonts::times()), endGameText(L"", Fonts::times(), largeCharacterSize)

{

moneyText.setFillColor(Color::Black);

livesText.setFillColor(Color::Black);

livesText.move(0, moneyText.getCharacterSize());

endGameText.move(bounds.width / 2, bounds.height / 2);

spawner.enemyPath = enemyPath;

spawner.waveDelay = seconds(enemyWaveSpawnDelay);

spawner.spawnDelay = seconds(enemySpawnDelay);

spawner.waves = enemyWaves;

for (int i = 0; i < platformPos.size(); i++)

platforms.push\_back(Platform(platformPos[i], upgrades));

for (int i = 0; i < upperEnemyPathBound.size(); i++) upperPathBound.append(Vertex(upperEnemyPathBound[i], Color::Black));

for (int i = 0; i < lowerEnemyPathBound.size(); i++) lowerPathBound.append(Vertex(lowerEnemyPathBound[i], Color::Black));

upperPathBound.setPrimitiveType(LineStrip);

lowerPathBound.setPrimitiveType(LineStrip);

}

void TDGame::update(const Time& deltaTime)

{

if (defeat || victory) return;

if (lives <= 0)

{

defeat = true;

return;

}

else if (enemies.size() <= 0 && spawner.waves.size() <= 0)

{

victory = true;

return;

}

gameTime += deltaTime;

optional<Enemy> e = spawner.spawn(gameTime);

if (e.has\_value()) enemies.push\_back(e.value());

for (int i = 0; i < enemies.size(); i++)

{

enemies[i].move(deltaTime.asSeconds());

if (enemies[i].getPosition() == enemyPath[enemyPath.size() - 1])

{

lives--;

enemies.erase(enemies.begin() + i);

}

}

for (int i = 0; i < platforms.size(); i++)

{

if (!platforms[i].getTank()) continue;

for (int j = 0; j < enemies.size(); j++)

{

if (intersects(platforms[i].getTank()->getPosition(), platforms[i].getTank()->getAreaRadius(), enemies[j].getPosition(), enemies[j].getRadius()))

{

platforms[i].getTank()->rotateTo(enemies[j].getPosition());

vector<Bullet> b = platforms[i].getTank()->shoot(enemies[j].getPosition(), gameTime);

bullets.insert(end(bullets), begin(b), end(b));

break;

}

}

}

for (int i = 0; i < bullets.size(); i++)

{

bullets[i].move(deltaTime.asSeconds());

if (!bounds.contains(bullets[i].getPosition()))

{

bullets.erase(bullets.begin() + i);

break;

}

for (int j = 0; j < enemies.size(); j++)

{

if (!bullets[i].hit(enemies[j].getPosition(), enemies[j].getRadius())) continue;

enemies[j].setHealth(enemies[j].getHealth() - bullets[i].getDamage());

bullets.erase(bullets.begin() + i);

if (enemies[j].getHealth() <= 0) {

money += enemies[j].getMaxHealth() / 2;

enemies.erase(enemies.begin() + j);

}

break;

}

}

}

void TDGame::processEvent(const Event& e)

{

Game::processEvent(e);

if (e.type == Event::MouseButtonPressed)

{

Vector2f mousePos = Vector2f(e.mouseButton.x, e.mouseButton.y);

for (int i = 0; i < platforms.size(); i++) platforms[i].processClick(mousePos, money);

}

}

void TDGame::render()

{

window.clear(Color::White);

if (defeat || victory)

{

endGameText.setString(defeat ? defeatInfo : victoryInfo);

endGameText.setFillColor(defeat ? Color::Red : Color::Green);

window.draw(endGameText);

}

else

{

moneyText.setString(moneyInfo + to\_wstring(money));

livesText.setString(livesInfo + to\_wstring(lives));

window.draw(moneyText);

window.draw(livesText);

window.draw(upperPathBound);

window.draw(lowerPathBound);

for (int i = 0; i < platforms.size(); i++) window.draw(platforms[i]);

for (int i = 0; i < enemies.size(); i++) window.draw(enemies[i]);

for (int i = 0; i < bullets.size(); i++) window.draw(bullets[i]);

}

window.display();

}

**Приложение А №10**

**Основной код программы (модуль Math)**

namespace

{

const double pi = 3.141592653589793238463f;

// Возвращает длину вектора.

float len(const Vector2f& v)

{

return sqrt(pow(v.x, 2) + pow(v.y, 2));

}

// Возвращает нормализованный вектор (сонаправленный данному с единичной длиной).

Vector2f normalize(const Vector2f& v)

{

return v / len(v);

}

// Возвращает поворот вектора в пространстве по часовой стрелке в радианах.

float rotation(const Vector2f& v)

{

return atan2f(v.y, v.x);

}

// Конвертирует радианы в градусы.

float degrees(float rad)

{

return rad \* (180.f / pi);

}

// Округляет координаты вектора до ближайших целых чисел.

Vector2i round(const Vector2f& v)

{

return Vector2i(roundf(v.x), roundf(v.y));

}

// Возвращает координаты центр квадрата.

Vector2f center(const FloatRect& r)

{

return Vector2f(r.left + r.width / 2, r.top + r.height / 2);

}

// Возвращает скалярное произведение вектров.

float dot(const Vector2f& v, const Vector2f& w)

{

return v.x \* w.x + v.y \* w.y;

}

// 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);

}

// Возвращает истину, если круг и точка пересекаются.

bool intersects(const Vector2f& circlePos, float radius, const Vector2f& point)

{

return sqrt(pow(circlePos.x - point.x, 2) + pow(circlePos.y - point.y, 2)) <= radius;

}

// 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 два круга пересекаются.

bool intersects(const Vector2f& circlePos1, float radius1, const Vector2f& circlePos2, float radius2)

{

return len(circlePos2 - circlePos1) <= radius1 + radius2;

}

// 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 });

}

}

**Приложение Б №1**

**Модульное тестирование (класс Bullet)**

namespace TestTD

{

Bullet b;

TEST\_CLASS(TestBullet)

{

public:

TEST\_METHOD\_INITIALIZE(initClass)

{

CircleShape bulletShape(8);

b = Bullet(bulletShape, Vector2f(0, 0), Vector2f(100, 0));

b.setSpeed(1);

}

TEST\_METHOD(testInit)

{

b = Bullet();

b.move();

Assert::IsTrue(b.getPosition() == Vector2f(0, 0));

}

TEST\_METHOD(testMove)

{

b.move(20);

Assert::IsTrue(b.getPosition() == Vector2f(20, 0));

}

};

}

**Приложение Б №2**

**Модульное тестирование (класс Enemy)**

namespace TestTD

{

Enemy e;

TEST\_CLASS(TestEnemy)

{

public:

TEST\_METHOD\_INITIALIZE(initClass)

{

CircleShape shape(16, 4);

vector<Vector2f> path =

{

Vector2f(0, 240),

Vector2f(854, 240),

Vector2f(854, 480),

Vector2f(0, 480)

};

e = Enemy(shape, path);

}

TEST\_METHOD(testInit)

{

e = Enemy();

e.move();

}

TEST\_METHOD(testMove)

{

e.setSpeed(100);

for (int i = 0; i < 9; i++) e.move();

Assert::IsTrue(e.getPosition() == Vector2f(854, 240));

for (int i = 0; i < 3; i++) e.move();

Assert::IsTrue(e.getPosition() == Vector2f(854, 480));

for (int i = 0; i < 10; i++) e.move();

Assert::IsTrue(e.getPosition() == Vector2f(0, 480));

}

};

}

**Приложение Б №3**

**Модульное тестирование (класс Tank)**

namespace TestTD

{

Tank t;

TEST\_CLASS(TestTank)

{

public:

TEST\_METHOD\_INITIALIZE(initClass)

{

Sprite s = Sprite();

CircleShape area(16);

CircleShape bulletShape(8);

t = Tank(s, area, bulletShape, Vector2f(500, 500));

}

TEST\_METHOD(testInit)

{

t = Tank();

vector<Bullet> b = t.shoot(Vector2f(0, 0), Time::Zero);

t.rotateTo(Vector2f(0, 0));

}

TEST\_METHOD(testShoot)

{

t.setBulletDamage(5);

t.setBulletSpeed(10);

t.setFiringDelay(seconds(1));

vector<Bullet> b = t.shoot(Vector2f(), Time::Zero);

Assert::IsTrue(b.empty());

b = t.shoot(Vector2f(), seconds(1));

Assert::AreEqual(b[0].getDamage(), 5.f);

Assert::AreEqual(b[0].getSpeed(), 10.f);

}

};

}

**Приложение Б №4**

**Модульное тестирование (класс RectangularMenu)**

namespace TestTD

{

RectangularMenu<int> r;

TEST\_CLASS(TestRectangularMenu)

{

public:

TEST\_METHOD\_INITIALIZE(initClass)

{

FloatRect rect(50, 50, 10, 10);

r = RectangularMenu<int>(rect, 10.f);

Texture texture = Texture();

texture.create(10, 10);

Sprite sprite = Sprite(texture);

sprite.setPosition(Vector2f(0, 0));

r.addItem(RectangularMenuItem<int>(sprite, 0));

r.addItem(RectangularMenuItem<int>(sprite, 1));

r.addItem(RectangularMenuItem<int>(sprite, 2));

r.addItem(RectangularMenuItem<int>(sprite, 3));

}

TEST\_METHOD(testInit)

{

r = RectangularMenu<int>();

r.addItem(RectangularMenuItem<int>());

r.visible = true;

optional<int> opt = r.choice(Vector2f(0, 0));

Assert::IsFalse(opt.has\_value());

}

TEST\_METHOD(testAddItem)

{

vector<RectangularMenuItem<int>> items = r.getItems();

Assert::IsTrue(items[0].icon.getPosition() == Vector2f(55, 35));

Assert::IsTrue(items[1].icon.getPosition() == Vector2f(75, 55));

Assert::IsTrue(items[2].icon.getPosition() == Vector2f(55, 75));

Assert::IsTrue(items[3].icon.getPosition() == Vector2f(35, 55));

}

TEST\_METHOD(testRemoveItem)

{

r.removeItem(0);

Assert::IsTrue(r.getItems().size() == 3);

}

TEST\_METHOD(testClear)

{

r.clear();

Assert::IsTrue(r.getItems().size() == 0);

}

TEST\_METHOD(testChoice)

{

Assert::IsFalse(r.choice(Vector2f(55, 35)).has\_value());

Vector2f pos[4] =

{

Vector2f(55, 35),

Vector2f(75, 55),

Vector2f(55, 75),

Vector2f(35, 55)

};

r.visible = true;

for (int i = 0; i < 4; i++) Assert::AreEqual(i, r.choice(pos[i]).value());

Assert::IsFalse(r.choice(Vector2f(55, 55)).has\_value());

}

};

}

**Приложение B №5**

**Модульное тестирование (модуль Math)**

namespace TestTD

{

TEST\_CLASS(TestMath)

{

public:

TEST\_METHOD(testLen)

{

sf::CircleShape s(1);

Vector2f v(1, 1);

float expected = sqrt(2);

Assert::AreEqual(expected, len(v));

}

TEST\_METHOD(testNormalize)

{

Vector2f v(1, 1);

Vector2f expected = Vector2f(1 / sqrt(2), 1/ sqrt(2));

Assert::IsTrue(expected == normalize(v));

}

TEST\_METHOD(testRotation)

{

Vector2f v(1, 1);

float expected = atan2f(1, 1);

Assert::AreEqual(expected, rotation(v));

}

TEST\_METHOD(testDegrees)

{

Vector2f v(1, 1);

float expected = 45.f;

Assert::AreEqual(expected, degrees(rotation(v)));

}

TEST\_METHOD(testRound)

{

Vector2f v(1.1, 1.9);

Vector2i expected(1, 2);

Assert::IsTrue(expected == round(v));

}

TEST\_METHOD(testCenter)

{

FloatRect r = FloatRect(1, 1, 2, 2);

Vector2f expected(2, 2);

Assert::IsTrue(expected == center(r));

}

TEST\_METHOD(testDot)

{

Vector2f v(1, 2);

Vector2f w(1, 2);

float expected = 5.f;

Assert::AreEqual(expected, dot(v, w));

}

TEST\_METHOD(testVerticalLineDistance)

{

Vector2f point(0, 0);

Vector2f line[2] = { Vector2f(1,1), Vector2f(1,2) };

float expected = sqrt(2);

float result = distance(point, line);

Assert::AreEqual(expected, distance(point, line));

}

TEST\_METHOD(testHorizontalLineDistance)

{

Vector2f point(0, 0);

Vector2f line[2] = { Vector2f(1, 1), Vector2f(2,1) };

float expected = sqrt(2);

Assert::AreEqual(expected, distance(point, line));

}

TEST\_METHOD(testDistance)

{

Vector2f point(0, 0);

Vector2f line[2] = { Vector2f(1,1), Vector2f(2,2) };

float expected = sqrt(2);

float dist = distance(point, line);

Assert::AreEqual(expected, distance(point, line));

}

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));

}

TEST\_METHOD(testCircleRectIntersection)

{

Vector2f circlePos(50, 50);

float radius = 10.f;

FloatRect rect1(45, 30, 10, 10);

FloatRect rect2(45, 40, 10, 10);

FloatRect rect3(0, 0, 100, 100);

FloatRect rect4(51, 51, 1, 1);

FloatRect rect5(51, 51, 1, 1);

FloatRect rect6(0, 0, 10, 10);

Assert::IsTrue(intersects(circlePos, radius, rect1));

Assert::IsTrue(intersects(circlePos, radius, rect2));

Assert::IsTrue(intersects(circlePos, radius, rect3));

Assert::IsTrue(intersects(circlePos, radius, rect4));

Assert::IsTrue(intersects(circlePos, radius, rect5));

Assert::IsFalse(intersects(circlePos, radius, rect6));

}

};

}

**Приложение В**

**Инструкция для пользователя**

Игра начинается непосредственно после запуска программы. Для того, чтобы построить защитную башню (танк), необходимо нажать на область внутри серой окружности. Затем откроется меню выбора танка (если существует возможность дальнейшего улучшения) с одной или несколькими ячейками, соответствующими определенному танку. Для того, чтобы построить танк, изображенный на ячейке, необходимо нажать на него и иметь достаточное количество денежной валюты.

Для того, чтобы увидеть область видимости врагов танка, необходимо нажать на него. Зеленая окружность обозначит эту область.

Суть игры состоит в том, чтобы не дать врагам пройти весь лабиринт путем строительства танков и их улучшений.

В игре существует 4 вида врагов:

1. Желтый квадрат. Он имеет малое количество жизней и медленную скорость. За каждое его убийство начисляется 5 единиц денежной валюты.
2. Красный треугольник. Он имеет среднее количество жизней и высокую скорость. За каждое его убийство начисляется 15 единиц денежной валюты.
3. Синий пятиугольник. Он имеет большое количество жизней и медленную скорость. За каждое его убийство начисляется 60 единиц денежной валюты.
4. Альфа-пятиугольник. Он имеет очень большое количество жизней и медленную скорость. За его убийство начисляется 500 единиц денежной валюты.

Также в игре есть три вида танка:

1. Обычный танк. Он обладает маленьким уроном и стреляет одной пулей в одно направление.
2. Близнец. Он обладает повышенным уроном и стреляет двумя пулями в одно направление.
3. Боковой защитник. Он обладает повышенным уроном и стреляет двумя пулями в противоположные направления.