**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**ФГБОУ ВО «БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**Кафедра «Информатика и программное обеспечение»**

**«Утверждаю»**

Зав. кафедрой «ИиПО», к.т.н., доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Копелиович Д.И.

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

**КУРСОВАЯ РАБОТА**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: «Разработка графического приложения**

**с использованием парадигмы ООП»**

**Вариант 13**

Студент гр. О-21-ИВТ-2-по-Б

\_\_\_\_\_\_\_\_\_\_\_Марченков М.И.

№ зач. книжки 21.0296

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_Радченко А.О.

Защищено с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

**Брянск 2023 г.**

Содержание

[Введение 3](#_Toc133680921)

[1. Аналитическая часть 4](#_Toc133680922)

[1.1. Анализ предметной области 4](#_Toc133680923)

[1.1.1. Frostpunk 4](#_Toc133680924)

[1.1.2. They Are Billions 5](#_Toc133680925)

[1.1.3. The Last Spell 6](#_Toc133680926)

[1.2. Выводы 8](#_Toc133680927)

[2. Конструкторская часть 9](#_Toc133680928)

[2.1. Архитектура проекта 9](#_Toc133680929)

[2.2. Абстрактные классы 9](#_Toc133680930)

[2.3. Основные методы 14](#_Toc133680931)

[3. Экспериментальная часть 18](#_Toc133680932)

[3.1. Руководство пользователя 18](#_Toc133680933)

[3.2. Тестирование приложения 18](#_Toc133680934)

[Заключение 21](#_Toc133680935)

[Список литературы 22](#_Toc133680936)

# Введение

Градостроительный симулятор ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) city-building game) — [компьютерная игра](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), в которой симулируется [строительство и управление](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80_%D1%81%D1%82%D1%80%D0%BE%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%B0_%D0%B8_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) городом. Как правило, относится к [экономическим стратегиям](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%81%D0%B8%D0%BC%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80). В играх такого рода обычно нет определённой цели, по достижении которой игра заканчивается, целью является сам процесс обустройства города.

Целью курсового проекта является закрепление навыков в объектно-ориентированном программировании на языке С++ с помощью программного средства Visual Studio, также обязательным пунктом является использование графической библиотеки SFML.

В качестве данной курсовой работы было принято решение разработать компьютерную игру, так как при разработке игр разработчику необходимо задействовать множество различных знаний, как в математике, так и в программировании.

# Аналитическая часть

## Анализ предметной области

Компьютерная игра в жанре градостроительный симулятор – представляет собой приложение, созданное для развлечения игроков. В основе этого жанра игр заключена механика строительства разнообразных зданий и скрупулёзный менеджмент всеми видами ресурсов.

Жанр зародился в [1968](https://ru.wikipedia.org/wiki/1968) с созданием [Дугом Дайментом](https://ru.wikipedia.org/wiki/%D0%94%D1%83%D0%B3_%D0%94%D0%B0%D0%B9%D0%BC%D0%B5%D0%BD%D1%82" \o "Дуг Даймент) игры [Hamurabi](https://ru.wikipedia.org/wiki/Hamurabi" \o "Hamurabi) на языке [Фокал](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D0%BA%D0%B0%D0%BB" \o "Фокал). Игра изначально была разработана для компьютеров [PDP-8](https://ru.wikipedia.org/wiki/PDP-8), но позже была переписана на языке [BASIC](https://ru.wikipedia.org/wiki/BASIC) и стала мультиплатформенной. Именно [Hamurabi](https://ru.wikipedia.org/wiki/Hamurabi" \o "Hamurabi) стала первой игрой по управлению городом, но при этом была проще последователей жанра (Simcity), например не позволяла строить новые здания и развивать инфраструктуру.

В ходе анализа предметной области были выбраны данные игры: Frostpunk, They Are Billions, The Last Spell.

### Frostpunk

Frostpunk  – [компьютерная игра](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0)  в жанре [градостроительного симулятора](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%BE%D1%81%D1%82%D1%80%D0%BE%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%81%D0%B8%D0%BC%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80) и [симулятор выживания](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80_%D0%B2%D1%8B%D0%B6%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), разработанная компанией [11 bit studios](https://ru.wikipedia.org/wiki/11_bit_studios) (рис. 1). Игра включает в себя элементы [стимпанка](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B8%D0%BC%D0%BF%D0%B0%D0%BD%D0%BA) и [альтернативной истории](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D1%8C%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F). Название игры является [неологизмом](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D0%B7%D0%BC), придуманным её создателями, и получено путём объединения слов frost ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) мороз) и steampunk (стимпанк).

Игра была выпущена 24 апреля 2018 года для платформы [Windows](https://ru.wikipedia.org/wiki/Windows), 11 октября 2019 года — для платформ [Xbox One](https://ru.wikipedia.org/wiki/Xbox_One), [PlayStation 4](https://ru.wikipedia.org/wiki/PlayStation_4). Версия для платформы [macOS](https://ru.wikipedia.org/wiki/MacOS) планировалась к выходу в феврале 2019 года, но из-за технических трудностей релиз был отложен и был выпущен 24 февраля 2021 года.

В игре присутствует несколько сюжетных компаний, которые в ходе своего прохождения потребуют от игрока решить множество проблем, связанных с обеспечением всего необходимого города, который находиться под управлением игрока. Карта города статична и никак не расширяется по ходу прохождения игры, потребности житель стоит учитывать, так как их удовлетворённость влияет на моральный дух и мотивацию от которых, зависит производительность работников и настроение в городе.



Рис. 1. Frostpunk

### They Are Billions

They Are Billions –  [компьютерная игра](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0) в жанре [стратегии в реальном времени](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F_%D0%B2_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8), разработанная испанской студией [Numantian Games](https://ru.wikipedia.org/w/index.php?title=Numantian_Games&action=edit&redlink=1). 18 июня 2019 года игра вышла для ПК, 5 июля состоялся релиз для [Xbox One](https://ru.wikipedia.org/wiki/Xbox_One), 9 июля — для [PlayStation 4](https://ru.wikipedia.org/wiki/PlayStation_4).

Действие игры происходит в [постапокалиптическом](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%82%D0%B0%D0%BF%D0%BE%D0%BA%D0%B0%D0%BB%D0%B8%D0%BF%D1%82%D0%B8%D0%BA%D0%B0) мире в стилистике [стимпанка](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B8%D0%BC%D0%BF%D0%B0%D0%BD%D0%BA), населённом зомби (рис. 2). Целью является постройка базы, изучение карты и сбор ресурсов, и одновременная борьба с зомби. Заражённые поселения, являющиеся идеальной целью для получения ресурсов и освоения близлежащих залежей [полезных ископаемых](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5%D0%B7%D0%BD%D1%8B%D0%B5_%D0%B8%D1%81%D0%BA%D0%BE%D0%BF%D0%B0%D0%B5%D0%BC%D1%8B%D0%B5).

Основа игры – это оборона от волн врагов о которых вы знаете заранее, для уничтожения пылищей зомби у вас представлен большой выбор построек и юнитов. Для добычи ресурсов используются только специальные здания, люди в этой игре не умирают, а пополняют ряды ходячих мертвецов. Ваше игровое пространство в ходе прохождения может увеличится если вы построите для этого специальное здание, захват территории в этой игре очень важен так как, источники ресурсов редко находятся в удобном для вас месте.



Рис. 2. Tre Are Billions

### The Last Spell

The Last Spell –  [тактическая ролевая игра 2021 года](https://en.m.wikipedia.org/wiki/Roguelike#Rogue-lites_and_procedural_death_labyrinths)[,](https://en.m.wikipedia.org/wiki/Tactical_role-playing_game)разработанная французской студией Ishtar Games и изданная The Arcade Crew. Игра вышла в [ранний доступ](https://en.m.wikipedia.org/wiki/Early_access) в [Steam](https://en.m.wikipedia.org/wiki/Steam_(service)" \o "Steam (сервис)) в июне 2021 года. Финальная версия игры, выпущенная 9 марта 2023 года, в тот же день, что и запуск консольных версий для [Nintendo Switch](https://en.m.wikipedia.org/wiki/Nintendo_Switch), [PlayStation 4](https://en.m.wikipedia.org/wiki/PlayStation_4) и [PlayStation 5](https://en.m.wikipedia.org/wiki/PlayStation_5).

В игре вам необходимо оборонять деревню от врагов ночью и развивать её днем, с помощью постройки новых зданий и оборонительных сооружений. В игре не так много ресурсов, но от того, как тщательно вы проработаете свою систему их добычи, очень сильно влияет на то победите вы или нет.

В отличии от двух вышеперечисленных игр в The Last Spell, жители играют не очень большую роль, в этой игре вам нужно будет всё своё внимание уделять вашим героям, их умениям и экипировке (рис. 3). Многие здания станут доступны вам только по ходу прохождения, а в начале игры у вас будет лишь ограниченный выбор из нескольких производственных и оборонительных построек.



Рис. 3. The Last Spell

В табл. 1 приведены результаты для сравнения:

Таблица 1

Сравнение программ

|  |  |  |  |
| --- | --- | --- | --- |
| Линии сравнения | Frostpunk | They Are Billions | The Last Spell |
| Система морального духа | Да | Нет | Да |
| Доступный интерфейс | Да | Да | Нет |
| Горожане умирают | Да | Да | Нет |
| Потребность горожан в еде | Да | Да | Нет |
| Размеры доступной карты | Не изменяются | Можно увеличить | Не изменяются |
| Сложность игры | Сложно | Сложно | Очень сложно |

## Выводы

Делая выводы исходя из проанализированной предметной области, можно выделить следующие пункты:

1. Создать простой и доступный интерфейс для пользователя;
2. Разработка системы морального духа и потребностей горожан в еде, жилье и лечении;
3. Создание игровой карты для размещения на ней разнообразных зданий с различными бонусами и предназначением.
4. Разработка системы сохранения игрового процесса для продолжения игровой сессии.

# Конструкторская часть

## Архитектура проекта

Данная игра написана на языке C++ с использованием графической библиотеки SFML в среде разработки Visual Studio 2019.

Архитектура данного приложения представлена на рис. 4.

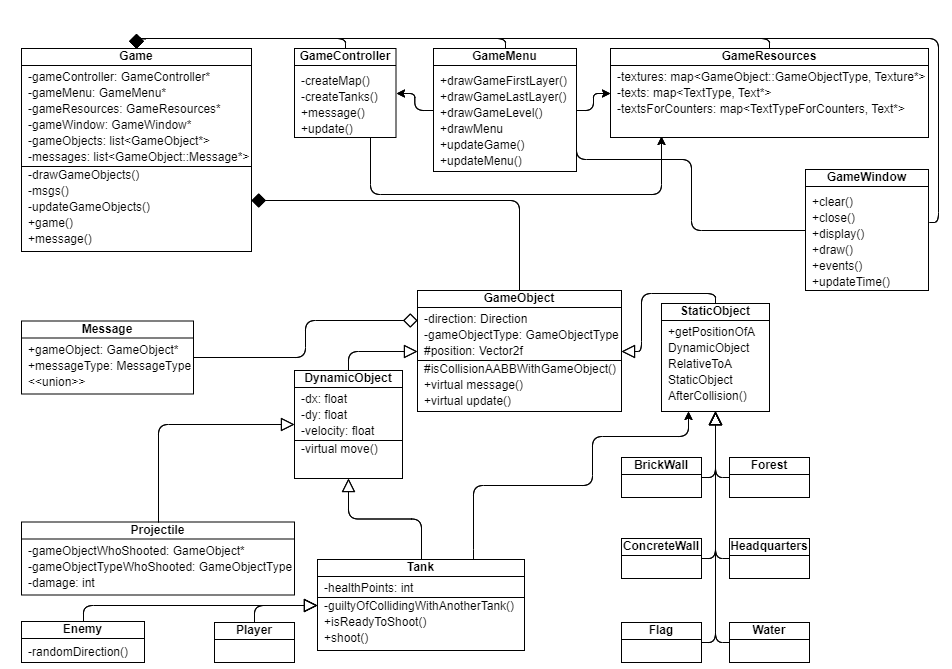


Рис. 4. Архитектура приложения

Файлами ресурсов являются изображения формата png, txt файл с содержимым игрового поля и шрифт формата ttf.

## Абстрактные классы

В данном проекте всего 4 абстрактных класса. Первый из них это GameObject от которого будут унаследованы все последующие классы. Он содержит в себе переменные, которые нужны каждому объекту: направление движения объекта, тип объекта, спрайт, ширина и высота спрайта, позиция объекта. В данном классе находятся все необходимые перечисления и структура сообщений. Из главных методов это нахождение коллизии между объектами, обновление объекта и обработка сообщения. Класс GameObject представлен в листинге 1.

Листинг 1

GameObject

class GameObject

{

public:

enum class Direction { UP,

DOWN,

LEFT,

RIGHT,

NONE };

enum class GameObjectType { BRICKWALL,

CONCRETEWALL,

ENEMY,

FLAG,

FOREST,

HEADQUARTERS,

PLAYER,

PROJECTILE,

WATER };

enum class MessageType { CREATE,

DEALDAMAGE,

DESTROY,

DESTROY\_ALL,

DESTROY\_THE\_WHOLE\_TYPE,

EMPTY };

struct Message

{

GameObject\* gameObject;

MessageType messageType;

union

{

struct

{

GameObjectType gameObjectType;

Vector2f position;

} create;

struct

{

GameObject\* gameObject;

int damage;

Продолжение листинга 1

} dealDamage;

struct

{

GameObjectType gameObjectType;

} destroyTheWholeType;

};

Message(GameObject\* \_gameObject,

GameObjectType \_gameObjectType,

MessageType \_messageType,

Vector2f \_position); // CREATE

Message(GameObject\* \_gameObjectWho,

GameObject\* \_gameObjectWhom,

int \_damage,

MessageType \_messageType); // DEALDAMAGE

Message(GameObject\* \_gameObject,

MessageType \_messageType); // DESTROY or EMPTY

Message(GameObject\* \_gameObject,

GameObjectType \_gameObjectType,

MessageType \_messageType); // DESTROY THE WHOLE TYPE

};

private:

bool destroyed;

Direction direction;

GameObjectType gameObjectType;

int w;

int h;

Sprite sprite;

protected:

Vector2f position;

bool isCollisionAABBWithGameObject(GameObject\* \_gameObject);

bool isDealDamage(Message\* \_message);

void createMessage(GameObjectType \_gameObjectType,

Vector2f \_position);

void destroyMessage();

void emptyMessage();

void setDirection(Direction \_direction);

void setPositionInSprite(Vector2f \_position);

public:

GameObject(Direction \_direction,

GameObjectType \_gameObjectType,

int \_w,

int \_h,

Texture\* \_texture,

Vector2f \_position);

Окончание листинга 1

Direction getDirection();

float getX();

float getY();

GameObjectType getGameObjectType();

int getW();

int getH();

Sprite getSprite();

Vector2f getPosition();

virtual void message(Message\* \_message) = 0;

virtual void update(float \_time) = 0;

};

Класс DynamicObject унаследован от GameObject. Он содержит в себе скорость каждого объекта и координаты на которые должен переместиться объект после каждого его обновления. Также содержит метод для перемещения объекта. Класс DynamicObject представлен в листинге 2.

Листинг 2

DynamicObject

class DynamicObject :

public GameObject

{

private:

virtual void move(float \_time) = 0;

protected:

float dx;

float dy;

float velocity;

public:

DynamicObject(Direction \_direction,

float \_velocity,

GameObjectType \_gameObjectType,

int \_w,

int \_h,

Texture\* \_texture,

Vector2f \_position);

};

StaticObject также унаследован от GameObject. Содержит в себе метод нахождения позиции динамического объекта после коллизии со статическим объектом. Класс StaticObject представлен в листинге 3.

Листинг 3

StaticObject

class StaticObject :

public GameObject

{

public:

StaticObject(GameObjectType \_gameObjectType,

int \_w,

int \_h,

Texture\* \_texture,

Vector2f \_position);

static Vector2f getPositionOfADynamicObjectRelativeToAStaticObjectAfterCollision(GameObject\* \_dynamicObject,

GameObject\* \_staticObject);

virtual void message(Message\* \_message) override;

void update(float \_time) override;

};

Класс Tank в свою очередь унаследован от DynamicObject. Хранит в себе здоровье танка, время, которое нужно дождаться до выстрела и текущее время для выстрела. Включает в себя метод нахождения виновника при коллизии и метод выстрела. Класс Tank представлен в листинге 4.

Листинг 4

Tank

class Tank :

public DynamicObject

{

private:

float cooldownMaxTimeForShooting;

float cooldownTimeForShooting;

int healthPoints;

bool guiltyOfCollidingWithAnotherTank(GameObject\* \_gameObject);

bool isDead();

public:

Tank(Direction \_direction,

float \_velocity,

GameObjectType \_gameObjectType,

Texture\* \_texture,

Vector2f \_position);

bool isReadyToShoot(float \_time);

void message(Message\* \_message) override;

void shoot();

};

## Основные методы

Основной метод обновления игровых объектов представлен в листинге 5. В цикле происходит обновление всех объектов в зависимости от времени между событиями.

Листинг 5

updateGameObjects

void Game::updateGameObjects()

{

for (auto gameObject : gameObjects)

gameObject->update(gameWindow->getTime());

}

Метод обработки сообщений, которые передали игровые объекты представлен в листинге 6. В цикле происходит обработка каждого сообщения, в зависимости от типа сообщения происходят разные события: создание или уничтожение объекта, сигнал всем объектам о том, что какой-то из других объектов получил урон или поменял свое местоположение.

Листинг 6

msgs

void Game::msgs()

{

for (auto message : messages)

{

switch (message->messageType)

{

case GameObject::MessageType::CREATE:

{

switch (message->create.gameObjectType)

{

case GameObject::GameObjectType::BRICKWALL:

gameObjects.push\_back(new BrickWall(message->create.position));

break;

case GameObject::GameObjectType::CONCRETEWALL:

gameObjects.push\_back(new ConcreteWall(message->create.position));

break;

case GameObject::GameObjectType::ENEMY:

gameObjects.push\_back(new Enemy(message->create.position));

break;

case GameObject::GameObjectType::FLAG:

Продолжение листинга 6

gameObjects.push\_back(new Flag(message->create.position));

break;

case GameObject::GameObjectType::FOREST:

gameObjects.push\_back(new Forest(message->create.position));

break;

case GameObject::GameObjectType::HEADQUARTERS:

gameObjects.push\_back(new Headquarters(message->create.position));

break;

case GameObject::GameObjectType::PLAYER:

gameObjects.push\_back(new Player(message->create.position));

break;

case GameObject::GameObjectType::PROJECTILE:

gameObjects.push\_back(new Projectile(message->gameObject->getDirection(),

message->gameObject,

message->gameObject->getGameObjectType(),

message->create.position));

break;

case GameObject::GameObjectType::WATER:

gameObjects.push\_back(new Water(message->create.position));

break;

}

break;

}

case GameObject::MessageType::DEALDAMAGE:

case GameObject::MessageType::EMPTY:

{

for (auto gameObject : gameObjects)

gameObject->message(message);

break;

}

case GameObject::MessageType::DESTROY:

{

gameController->message(message);

auto object = find(gameObjects.begin(),

gameObjects.end(),

message->gameObject);

delete\* object;

gameObjects.erase(object);

break;

}

case GameObject::MessageType::DESTROY\_ALL:

{

gameController->message(message);

for (auto gameObject : gameObjects)

delete gameObject;

Окончание листинга 6

gameObjects.clear();

break;

}

case GameObject::MessageType::DESTROY\_THE\_WHOLE\_TYPE:

{

list<GameObject\*> destroyedGameObjects;

for (auto gameObject : gameObjects)

if (gameObject->getGameObjectType() == message->destroyTheWholeType.gameObjectType)

{

delete gameObject;

destroyedGameObjects.push\_back(gameObject);

}

for (auto destroyedGameObject : destroyedGameObjects)

gameObjects.remove(destroyedGameObject);

destroyedGameObjects.clear();

break;

}

}

delete message;

}

messages.clear();

}

Метод обработки сообщения внутри объекта представлен в листинге 7.

Объект Projectile обрабатывает сообщения о передвижении всех объектов. Если произошла коллизия данного объекта с другим, то происходит дальнейшая обработка исходя из типа игрового объекта. Если снаряд выпущен врагом и попал во врага, то снаряд просто уничтожается, тоже самое происходит и при попадании снаряда в другой снаряд или при попадании в бетонную стену. При попадании игрока во врага и наоборот, происходит отправка сообщения о нанесении урона, тоже самое и с объектом кирпичной стены или штабом.

Листинг 7

message

void Projectile::message(Message\* \_message)

{

if (\_message->messageType == MessageType::EMPTY and

isCollisionAABBWithGameObject(\_message->gameObject))

{

if (\_message->gameObject->getGameObjectType() == GameObjectType::ENEMY and

gameObjectTypeWhoShooted == GameObjectType::ENEMY and

\_message->gameObject != gameObjectWhoShooted)

destroyMessage();

Окончание листинга 7

else if ((\_message->gameObject->getGameObjectType() == GameObjectType::ENEMY and

gameObjectTypeWhoShooted == GameObjectType::PLAYER) or

(\_message->gameObject->getGameObjectType() == GameObjectType::PLAYER and

gameObjectTypeWhoShooted == GameObjectType::ENEMY))

dealDamageMessage(\_message->gameObject,

damage);

else if (\_message->gameObject->getGameObjectType() == GameObjectType::PROJECTILE and

\_message->gameObject != this)

destroyMessage();

else if (\_message->gameObject->getGameObjectType() == GameObjectType::BRICKWALL or

\_message->gameObject->getGameObjectType() == GameObjectType::HEADQUARTERS)

dealDamageMessage(\_message->gameObject,

NULL);

else if (\_message->gameObject->getGameObjectType() == GameObjectType::CONCRETEWALL)

destroyMessage();

}

}

# Экспериментальная часть

## Руководство пользователя

После запуска приложения игрок может начать игру нажав на кнопку NEW GAME или продолжить уже существующую через кнопку CONTINUE, EXIT соответственно закрывает приложение (рис. 5).

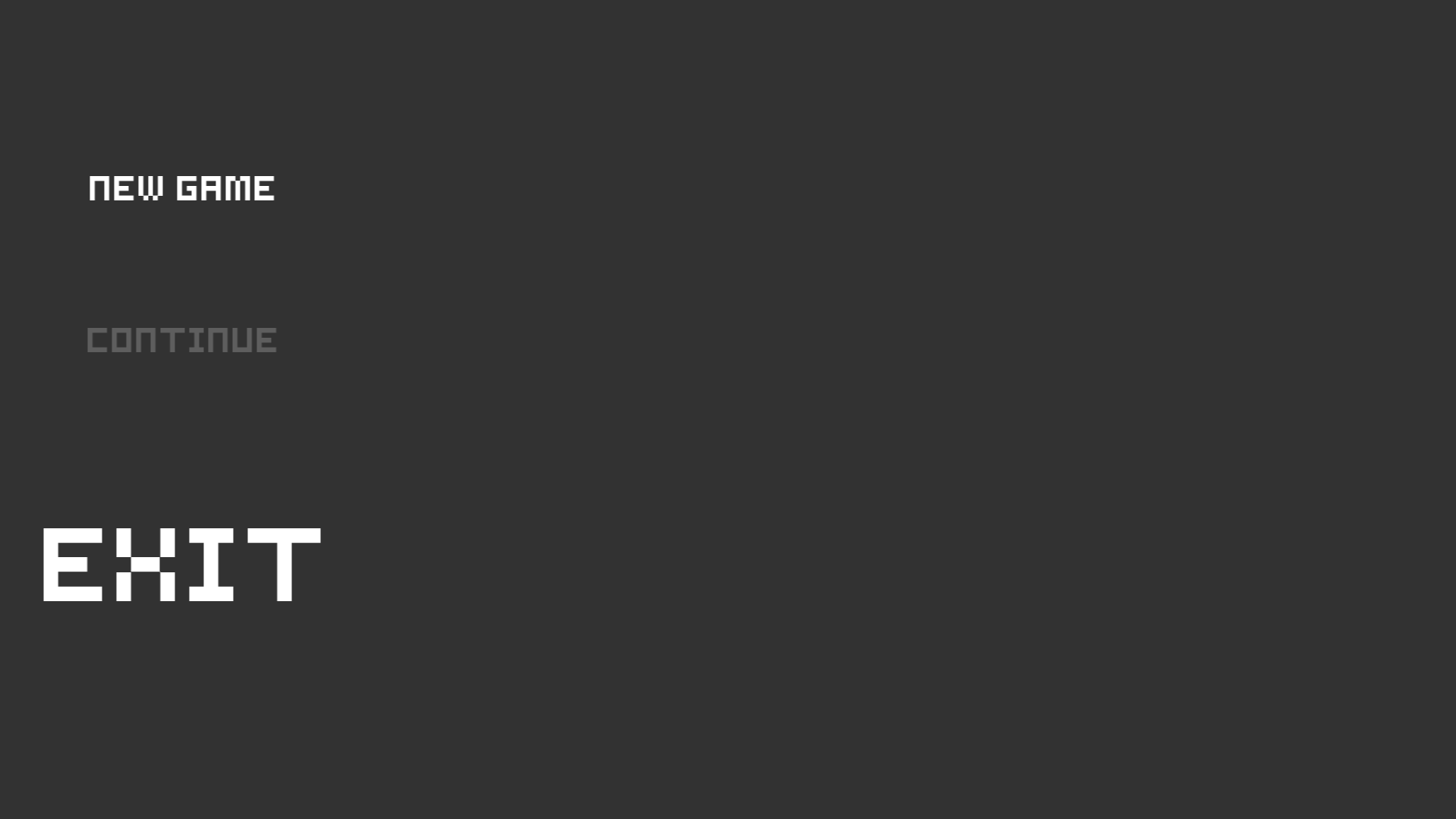


Рис. 5. Меню приложения

## Тестирование приложения

В первую очередь необходимо проверить корректность столкновения танков друг с другом. Для теста попробуем танком игрока столкнуться с вражеским танком и посмотрим на их реакцию. При столкновении танков лоб в лоб оба танка останавливаются, однако если первый танк столкнется со вторым, в то время как второй движется не по направлению к первому, то останавливается лишь первый танк, а второй может спокойно продолжить движение. Исходя из данных тестов, можно сделать вывод, что логика обработки коллизии танков работает корректно (рис. 6).

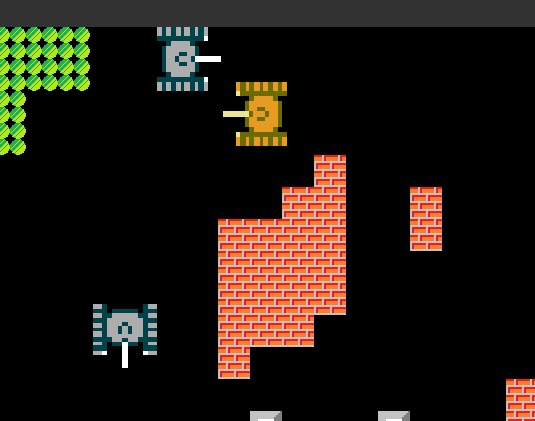


Рис. 6. Коллизия танков

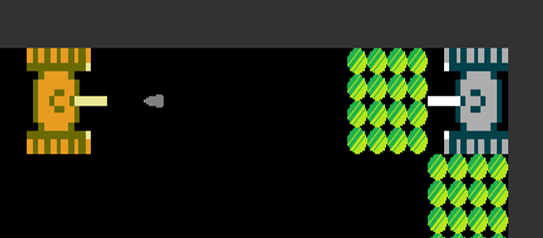
Во вторую очередь необходимо проверить корректность взаимодействия снарядов танков с другими объектами. Для теста будем пробовать уничтожать различные объекты. В результате попадания снаряда в кирпичную стену, танк, штаб или в чужой снаряд, объекты уничтожаются вместе со снарядом. В том случае если снаряд уходит за границу экрана или попадает в бетонную стену он уничтожается. В результате проделанных тестов, можно сказать, что логика взаимодействия снаряда с различными объектами – корректна (рис. 7).

Рис. 7. Взаимодействие снаряда с объектами

В последнюю очередь необходимо проверить корректность заезда танка в узкие пространства из препятствий. Для этого необходимо уничтожить несколько объектов бетонных стен, тем самым создав это узкое пространство. При попытке какого-либо танка заехать в узкое пространство, благодаря функции подгона координат для таких ситуаций, танк корректно заезжает в такие места. В результате можно сказать, что танк корректно заезжает в различные узкие пространства (рис. 8 и рис. 9).

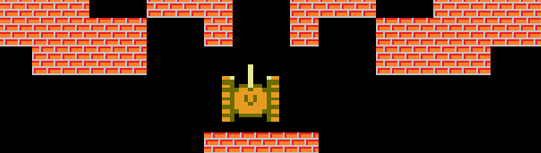


Рис. 8. Положение танка до заезда в узкое пространство

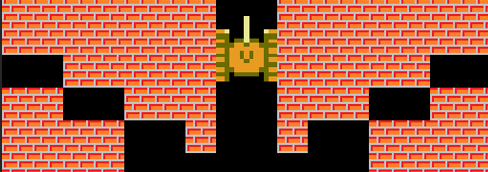


Рис. 9. Положение танка после заезда в узкое пространство

# Заключение

В качестве данной темы курсовой работы была выбрана разработка игры в жанре экшен с видом сверху на языке программирования C++ с использованием графической библиотеки SFML. Проведя анализ предметной области, были найдены главные особенности этого жанра и использованы в данном курсовом проекте.

Была разработана игра, соответствующая функциональным требованиям. В процессе написания программы были получены практические навыки разработки архитектуры игры в жанре экшен, различных механик для ее создания, а также графической библиотеки SFML и использования объектно-ориентированного программирования.

Данное приложение было протестировано по основным механикам, особенно важных ошибок в ходе данного процесса не выявлено. Проект можно считать успешно законченным.

В дальнейшем планируется добавить различные бонусы для игрока, новые виды танков и режим для двух игроков.

# Список литературы

1. Понамарев, В. Программирование на C++/C# в Visual Studio .NET 2003 / В. Понамарев. - М.: БХВ-Петербург, 2015. – 917 c.
2. Бьерн Страуструп Дизайн и эволюция языка С++ / Бьерн Страуструп. - М.: ДМК Пресс, 2014. - 446 c.
3. Пахомов, Борис C/C++ и MS Visual C++ 2012 для начинающих / Борис Пахомов. - Москва: СИНТЕГ, 2015. – 51 с.
4. Израилев, В.Я. Программирование на языке Си: Объектно- ориентированное программирование [Текст] + [Электронный ресурс]: практикум / В.Я. Израилев – Брянск: БГТУ, 2018. – 100 с.
5. Как настроить библиотеку SFML в Visual Studio [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/703500/ (Дата обращения 30.12.2022)
6. Уроки по SFML [Электронный ресурс]. Режим доступа: https://kychka-pc.ru/ (Дата обращения 30.12.2022)