

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

Игровое приложение «2D Танки»

БГУИР КП 1-40 01 01 009 ПЗ

Студент: гр. 151003 Матошко И.В.

Руководитель: Красковский П. Н.

Минск 2023

Учреждение образования

«Белорусский государственный университет информатики и
радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ПОИТ

(подпись)
Лапицкая Н.В. 2022г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Матошко Ивану Викторовичу

1. Тема работы Игровое приложение «2D Танки»

2. Срок сдачи студентом законченной работы 22.05.2023г.

3. Исходные данные к работе Среда программирования Visual Studio. Наличие графической реализации интерфейса игры. Наличие приложения-сервера и приложения-клиента. Возможность двухмерного управления одним танком. Возможность взаимодействия объектов карты с танком и танками других игроков. В реализованы следующие возможности: подключение игроков к серверу, возможности взаимодействия игроков в реальном времени путём столкновений и выстрелов. Реализована синхронизация анимации движения танков, а также их выстрелов на различных клиентах. Имеется возможность уничтожения врагов с помощью выстрелов.

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение

1 Анализ литературных источников

2 Постановка задачи

3 Разработка программного средства

4 Тестирование и проверка работоспособности программного средства

5 Руководство по использованию программного средства

Заключение

Список использованной литературы

Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

Схема алгоритма в формате A1

6. Консультант по курсовой работе Красковский П. Н. _____

7. Дата выдачи задания 27.02.2023г.

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объема работы):

Раздел 1. Введение к 15.03.2023г. – 10 % готовности работы;

Раздел 2 к 30.03.2023г. – 30% готовности работы;

Раздел 3 к 10.04.2023г. – 60% готовности работы;

Разделы 4, 5, Заключение к 01.05.2023 – 90 % готовности работы;

Оформление пояснительной записки и графического материала к 21.05.2023 – 100 % готовности работы.

Защита курсового проекта с 21.05.2023г. по 22.05.2023г.

РУКОВОДИТЕЛЬ _____ Красковский П. Н.
(подпись)

Задание принял к исполнению _____ Матошко И.В. 16.09.2022г.
(дата и подпись студента)

СОДЕРЖАНИЕ

Введение.....	6
1 анализ литературных источников.....	7
1.1 Анализ существующих аналогов.....	7
1.1.1 Танки «Tank Battalion»	7
1.1.2 Танки «Battle City»	9
1.1.3 Танки «Hard Tank»	10
1.2 Анализ методов и способов разработки	12
1.2.1 Используемые библиотеки и технологии	12
1.2.2 Используемые структуры данных	12
2 Постановка задачи.....	14
2.1 Назначение разработки.....	14
2.2 Перечень функциональных требований	15
2.3 Структура программы.....	16
2.4 Состав и параметры технических и программных средств	17
3 Разработка программного средства.....	18
3.1 Описание алгоритмов решения задачи	18
3.2 Структура данных	20
3.2.1 Структура типов программы.....	20
3.3 Схема алгоритмов решения задач по ГОСТ 19.701-90	25
3.3.1 Схема алгоритма Player::update	25
3.3.2 Схема алгоритма startGame	27
3.3.3 Схема алгоритма Bullet::update.....	29
3.3.4 Схема алгоритма LifeBar::draw	31
3.3.5 Схема алгоритма getPlayercoordinateforview	33
4 тестирование и проверка работоспособности программного средства.....	36
4.1 Запуск приложений клиента и сервера	36
4.1.1 Тест 1	36
4.1.2 Тест 2	36
4.2 Подключение игрока/игроков к серверу.....	37
4.2.1 Тест 3	37
4.2.2 Тест 4	38
4.3 Взаимодействие танков между собой и со стенками карты	38
4.3.1 Тест 5	38
4.3.2 Тест 6	39
4.3.3 Тест 7	39
4.3.4 Тест 8	40
4.3.1 Тест 9	40
5 Руководство по использованию программного средства	41
5.1 Управление персонажем.....	41
5.2 Подключение	41
Заключение	42

Список использованной литературы.....	43
Приложение А	44
Приложение Б.....	60

ВВЕДЕНИЕ

Данная работа посвящена созданию игры 2D Танки.

Игра – неперенный спутник развития человечества. На стадии археокультуры игры выполняли чрезвычайно важные функции. Они использовались для социализации подрастающего поколения.

Развитие и совершенствование игр тесно связано с развитием компьютерного обеспечения и технологий. Сейчас многие составные части компьютеров разрабатываются специально для игр. Например, дорогие видеокарты, стоимость которых доходит до половины стоимости удовлетворительного компьютера для работы. Все игры разрабатываются с учётом последних новинок компьютерной техники, реагируя на все достижения и всё ближе подходя к реальности изображения и звука. На сегодняшний день существуют поражающие своей правдоподобностью игры с хорошим графическим и звуковым оформлением, почти полностью имитирующим жизнь. Люди воспринимают игры по-разному: для одних это способ развлечься и отдохнуть, для других игры являются неотъемлемой частью жизни, когда речь заходит о киберспорте. Но факт остается неизменным, сегодня игры представляют собой огромную и развитую культуру, объединяющую людей со всего мира.

Четкая классификация игр затруднена из-за того, что трудно отнести игру к какому-нибудь конкретному жанру. Игра может представлять собой как смешение существующих жанров, так и не относиться ни к одному из них. В свою очередь, данное игровое приложение относится к жанру аркады.

Понятие аркады пришло с аркадных автоматов. Именно был наиболее популярен. Первые аркадные игры использовали бесцветную векторную графику. К особенностям относится игра на одном экране для одного пользователя и отсутствие сюжета и истории. Потенциально, аркадная игра может продолжаться неограниченное количество времени. Поэтому и классическая аркада либо сводится к скорейшей победе над искусственным или реальным противником, либо и к вовсе самой сути бесконечной игры.

Примеры аркад: Battle City, Tank Battalion, Hard Tank, Pacman и другие.

Целью работы является создание проекта игрового приложения в жанре аркады «2D Танки», сопровождающегося документацией в виде пояснительной записки.

1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

1.1 Анализ существующих аналогов

Современные аркады представляют собой большие игры, в которых основу игрового процесса может даже составлять и сюжетная линия. Изначально аркадами были названы игры на игровых автоматах. Этим фактом и обусловлена их специфика. Во-первых, автоматы не подходили для сложных сюжетных игр. Во-вторых, их ставили в местах большого скопления людей, поэтому они должны были быть привлекательными для широкой целевой аудитории. Сегодня до нас дошло много аркадных игр, и они не перестают выходить. В их числе находятся и ветераны игровой индустрии танков: «Tank Battalion», «Battle City», «Hard Tank» и др.

1.1.1 Танки «Tank Battalion»

«Tank Battalion» – это аркадный шутер, разработанный Namco и выпущенный для аркадных автоматов в 1980 году. Игрок, управляющий танком, должен уничтожить двадцать вражеских танков в каждом уровне, которые появляются на игровом поле в верхней части экрана. Вражеские танки пытаются уничтожить базу игрока (обозначенную на карте в виде орла), а также сам танк игрока. Уровень завершён, когда игрок уничтожает все двадцать вражеских танков, но игра заканчивается только тогда, когда игрок потеряет все жизни, либо не сможет защитить базу. Достоинства игры:

- качественная 2D графика для своего времени;
- музыкальное и звуковое сопровождение.

В свою очередь к недостаткам можно отнести:

- возможность игра только с компьютером.



Рисунок 1.1 – «Tank Battalion»



Рисунок 1.2 – «Tank Battalion»

1.1.2 Танки «Battle City»

«Battle City» – В России и странах СНГ выпускалась на пиратских картриджах как в оригинальном виде, так и в модификации Tank 1990, и известна под неофициальным названием «Танчики». Её предшественником была аркадная игра Tank Battalion, выпущенная фирмой Namco в 1980 году.

Полигон действий виден сверху. Игрок должен, управляя своим танком, уничтожить все вражеские танки на уровне, которые постепенно появляются вверху игрового поля. Враги пытаются уничтожить штаб игрока (внизу игрового поля в виде орла) и его танк. На каждом уровне нужно уничтожить двадцать единиц бронетехники противника разных видов. Если противник (или игрок) сможет разрушить штаб или лишит игрока всех жизней — игра окончена. Через 7 лет после выхода, появились модификации для игры вдвоем.

Достоинства игры:

- высокий уровень графики по сравнению с аналогами того времени;
- уникальные механики и элементы ландшафта;
- забавный сюжет;
- отличное музыкальное сопровождение.

Недостатков выявлено не было.



Рисунок 1.3 – «Battle City»



Рисунок 1.4 – «Battle City»

1.1.3 Танки «Hard Tank»

«Hard Tank» – это игра в жанре экшны, разработанная Wolderado. Она была выпущена в 2018. Hard Tank доступна на PC и Web. Игра продаётся в itch.io. Игра разработана как проект для #WeeklyGameJam26 всего за 7 дней, поэтому исходная версия игры не имела всего функционала. В игре 15 уровней с возрастающим темпом усложнения. После завершения игры, игрок получает бонусные очки времени. Отличительной особенностью является возможность соревноваться на онлайн-табло, созданном с помощью dreamlo.

Достоинства игры:

- высокий уровень графики и улучшенные спецэффекты;
- точное управление.

В свою очередь к недостаткам можно отнести:

- малое количество игрового контента.

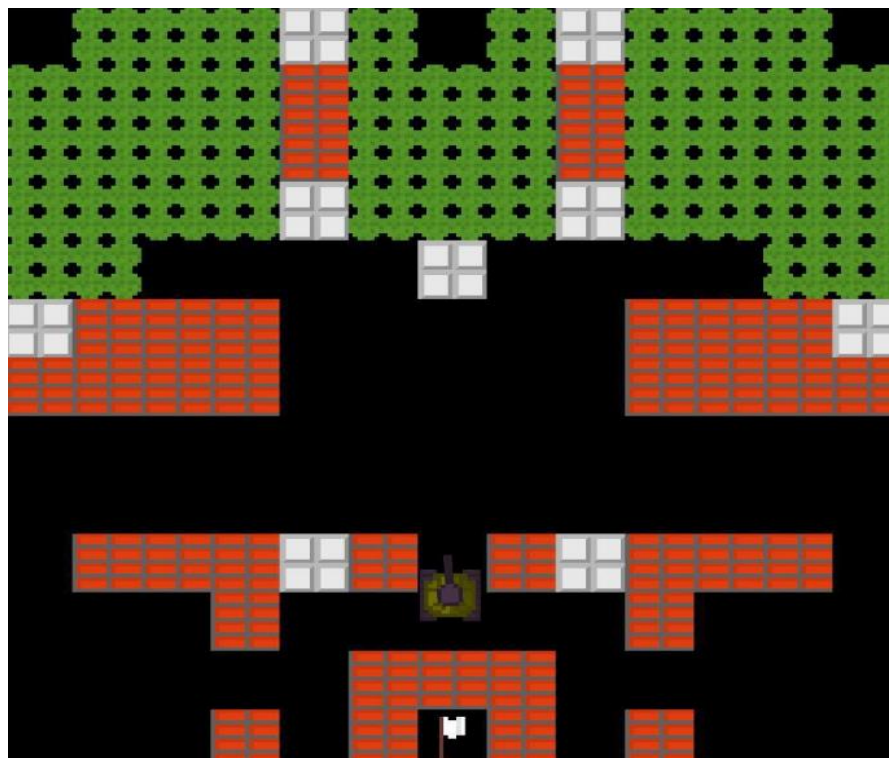


Рисунок 1.5 – «Hard Tank»



Рисунок 1.6 – «Hard Tank»

1.2 Анализ методов и способов разработки

1.2.1 Используемые библиотеки и технологии

Предполагается, что разрабатываемая игра будет обладать двумерной графикой с возможностью базовых передвижений танка. Главный герой должен быть способен двигаться в четырех направлениях и стрелять. В то же время необходимо отладить взаимодействие между стенками карты, включая врагов и пули. Для этих целей будет использована сторонняя библиотека SFML.

SFML (англ. Simple and Fast Multimedia Library) – свободная кроссплатформенная мультимедийная библиотека. Написана на C++, но доступна также для C, C#, .Net, D, Java, Python, Ruby, OCaml, Go и Rust. Представляет собой объектно-ориентированный аналог SDL.

SFML содержит ряд модулей для простого программирования игр и мультимедиа приложений. Предполагается использование следующих модулей:

- Window – управление окнами и взаимодействием с пользователем;
- Graphics – делает простым отображение графических примитивов и изображений;
- Audio – предоставляет интерфейс для управления звуком.

В то же время ставится задача реализовать игру нескольких игроков. Для этого будут использоваться сокеты и возможности одного из модулей библиотеки SFML:

- Network – предоставляет интерфейс для управления сокетами и удобной реализации игровой сети.

Для построения архитектуры взаимодействия клиента и сервера будут использованы сокеты. Сокет - это интерфейс между вашим приложением и внешним миром: через сокет вы можете отправлять и получать данные. Поэтому любой сетевой программе, скорее всего, придется иметь дело с сокетами, они являются центральным элементом сетевого взаимодействия.

Существует несколько видов сокетов, каждый из которых предоставляет определенные функции. В SFML реализованы наиболее распространенные из них: сокеты TCP и UDP.

1.2.2 Используемые структуры данных

В данном проекте в качестве основной структуры данных будет использован вектор (vector).

Структура vector предназначена для первоначальной загрузки для хранения всех подключенных клиентов на сервере, а также копий объектов других игроков на данном клиенте. Вектор – это структура данных, которая является моделью динамического массива. Отличительной чертой данной

реализации массива является отсутствие необходимости пользоваться указателями и ручным созданием динамического массива. Структура vector представлена на рисунке 1.11.

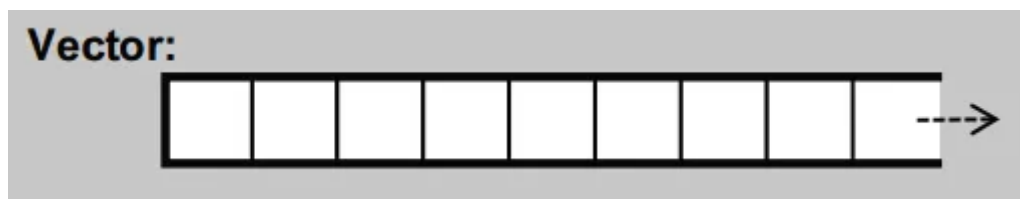


Рисунок 1.7 – Вектор

2 ПОСТАНОВКА ЗАДАЧИ

2.1 Назначение разработки

Назначением проектирования является разработка игры 2D Танки. На основании произведенного обзора существующих аналогов, выявленных преимуществ и недостатков данных игр, сделан вывод, что для решения поставленной цели необходимо выполнить следующие задачи:

- проектирование архитектуры игры;
- проектирование графического сопровождения;
- разработка алгоритмов передвижения танков;
- разработка алгоритмов взаимодействия объектов между собой;
- разработка алгоритма инициализации сервера;
- разработка алгоритма инициализации клиента;
- разработка алгоритмов обмена данными между пользователями;
- разработка алгоритмов синхронизации анимации между клиентами;
- разработка алгоритмов отрисовки анимации объектов на экране;
- разработка алгоритмов работы с камерой;
- разработка алгоритмов отображения сопровождающей информации;
- тестирование приложения.

2.2 Перечень функциональных требований

Целью разработки игры 2D Танки является объединение основных достоинств рассмотренных существующих аналогов, а также компенсация недостатков этих игр. В результате разработки необходимо предоставить реализацию следующих функций:

- подключение клиента к серверу;
- инициализация сервера и подключенного клиента;
- взаимодействие между танками всех клиентов;
- взаимодействие между танками всех клиентов и пулями;
- взаимодействие между пулями и картой;
- отображение здоровья;
- завершение игры для убитого врага.

2.3 Структура программы

При разработке приложения будет использовано 7 модулей:

- MultiplayerProject_Client.cpp – главный модуль, содержащий приложение игрока;
- NetworkClient.cpp (NetworkClient.h) – модуль, обеспечивающий сетевое взаимодействие игрока;
- LifeBar.h – модуль, обеспечивающий отображение здоровья игрока;
- view.h – модуль, предназначенный для работы с камерой вида;
- map.h – модуль с картой;
- MultiplayerProject_Server.cpp – главный модуль, содержащий приложение сервера;
- NetworkServer.cpp – модуль сервера, обеспечивающий сетевое взаимодействие с клиентом;

2.4 Состав и параметры технических и программных средств

Игра 2D Танки должна функционировать на персональных компьютерах со следующими характеристиками:

- Операционная система Windows 10;
- RAM: 2 GB;
- Пространство на диске: 1 GB;
- Процессор: минимальное требование - Pentium 2 266 МГц;
- Монитор;
- Мышь;
- Клавиатура.

В данном разделе указаны минимальные технические требования для запуска игры. Для эксплуатации в реальных условиях могут потребоваться более мощные технические средства. Разработанная игра должна корректно функционировать на более мощном оборудовании.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Описание алгоритмов решения задачи

Таблица 1 – Описание алгоритмов решения задачи

№ п.п.	Наименование алгоритма	Назначение алгоритма	Формальные параметры	Предлагаемый тип реализации
1	MultiplayerProject_Client::main()	Служит отправной точкой выполнения программы клиента. Создает окно игры. Производит инициализацию вектора других игроков. Использует алгоритмы: init(), registerOnServer(), receiveConnectedClientsNames()		Функция. Возвращаемый параметр имеет – число вызывающего его процесса (целочисленный тип)
2	MultiplayerProject_Server::main	Служит отправной точкой выполнения программы сервера. Производит принятие и отправку данных другим пользователям. Использует алгоритмы: registerNewClients(), sendConnectedClientsRecords(), sendDataToAll(), receiveData()		Функция. Возвращаемый параметр имеет – число вызывающего его процесса (целочисленный тип)
3	Player::update(time)	Служит для обновления координат игрока и его спрайта. Использует алгоритмы: interactionWithMap()	time – время между событиями запуска игры и текущим моментом	Метод

Продолжение таблицы 1 – Описание алгоритмов решения задачи

4	Player::Interaction-WithMap()	Осуществляет проверку столкновения игрока с объектами карты		Процедура
5	Player::draw-Vec(window)	Рисует танки других игроков на карте	window – игровое поле	Процедура
6	Bullet::update(time)	Служит для обновления координат пули и проверки пересечения пули с картой	time – время между событиями запуска игры и текущим моментом	Процедура
7	addPlayer(t_player, font, clientName, spawnX, spawnY)	Осуществляет добавление игрока в вектор всех игроков клиента	t_player – текстура игрока; font – шрифт имени игрока; clientName – имя клиента; spawnX – точка x появления нового игрока; spawnY – точка y появления нового игрока;	Процедура
8	getplayercoordinateforview(x, y)	Производит управление камерой вида	x, y – координаты игрока	Функция. Возвращаемый параметр – камера вида (тип View)

3.2 Структура данных

3.2.1 Структура типов программы

Таблица 2 – Структура типов программы

Элементы данных	Рекомендуемый тип	Назначение
Client	<pre>struct Client { string name; IpAddress Ip; UdpSocket* dataSocket; unsigned short port; Packet rDataPacket; Packet sDataPacket; bool done = true; Vector2f pos{ 5,5 }; bool turned; int direction; float curFrame; int bulSize; bool changeBulSize; }</pre>	Структура для хранения данных о клиенте на сервере. Поля данных: name – имя игрока; Ip – ip адрес игрока; dataSocket – udp сокет игрока; port – порт udp сокета; rDataPacket, sDataPacket – пакеты данных игрока; done – вспомогательный флаг; pos – дефолтная позиция игрока; turned – логическая переменная, отображает поворот игрока; direction – отображает направление игрока; curFrame – текущий кадр анимации игрока; bulSize – размер вектора пуль игрока; changeBulSize – логическая переменная, отображает изменение размера вектора пуль игрока
NetworkServer	<pre>class NetworkServer { short regStep = 0; TcpListener listener; TcpSocket regSocket;</pre>	regStep – шаг регистрации игрока на сервере; listener – слушатель tcp порта; regSocket – tcp порт регистрации игрока;

Продолжение таблицы 2 – Структура типов программы

	<pre> Packet packet; Clock sendingsRate- Timer; Int32 sendingsRate = 4; vector<Client> clientsVec; Status init(); Status registerNew- Clients(); Status sendConnected- ClientsRecords(); Status receive- Data(unsigned int receivedClient Index); Status sendDataToAll (Packet dataPacket); Status acceptIncoming- Connection(); Status receiveClient- RegData(); Status sendNewClient- DataToAll(); Status sendDedicated- DataPort(); } </pre>	<p>packet – пакет данных. sendingsRateTimer – частота отправки; sendingsRate – частота отправки; clientsVec – вектор клиентов на сервере.</p> <p>Методы класса: init() – инициализирует сервер, выводит порт сервера; registerNewClients() – регистрирует игрока на сервере, возвращает статус сервера; sendConnected- ClientsRecords() – отправляет имена зарегистрированных клиентов новому клиенту, возвращает статус сервера; receiveData(unsigned int) – получает данные от игрока, возвращает статус сервера; sendDataToAll(Packet dataPacket) – посылает данные игрока всем другим клиентам, возвращает статус сервера; acceptIncomingConnection() – принимает входящее подключение игрока, возвращает статус сервера; receiveClientRegData() – получает регистрационные данные игрока и создает запись на сервере, возвращает статус сервера; sendNewClientDataToAll() – отправка имени нового</p>
--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Продолжение таблицы 2 – Структура типов программы

		игрока другим игрокам, возвращает статус сервера; sendDedicatedDataPort() – отправка игроку udp порта, возвращает статус сервера
Player	<pre> class Player { float x, y, w, h, dx, dy, speed; int dir; int health; Sprite sprite, netGhost; Text t; bool possessed = false; string name; bool turned = false; bool isShoot; float currentFrame; bool FirstShoot; Clock timeShootC; int sendBulletSize; bool changeBulSize; std::list<Bullet*> bullets; std::list<Bullet*>:: iterator it; void update(float time); void interaction- WithMap(); void load(Texture texture, Font font); bool isPossesed(); void draw (RenderWindow window); FloatRect getRect(); void drawVec(Render- Window window); } </pre>	<p>Класс игрока.</p> <p>Поля данных:</p> <p>x, y, w, h, dx, dy, speed – координаты, ширина, высота, векторы движения и скорость игрока;</p> <p>dir – текущее направление движения игрока;</p> <p>health – здоровье игрока;</p> <p>sprite, netGhost – спрайты игрока;</p> <p>t – имя игрока на экране;</p> <p>possessed – логическая переменная, отвечает за регистрацию игрока;</p> <p>name – имя игрока;</p> <p>turned – логическая переменная, отвечает за поворот игрока;</p> <p>isShoot – логическая переменная, отвечает за стрельбу;</p> <p>currentFrame – текущий кадр анимации игрока;</p> <p>FirstShoot – логическая переменная, отвечает за стрельбу игрока;</p> <p>timeShootC – время, отвечает за задержку стрельбы;</p>

Продолжение таблицы 2 – Структура типов программы

		<p>sendBulletSize – отправляемый размер вектора пуль; changeBilSize – логическая переменная, отвечает за создание пуль у других игроков; bullets – вектор пуль игрока; it – итератор пуль игрока.</p> <p>Методы класса: update(float time) – отвечает за обновление координат и спрайта игрока; interactionWithMap() – отвечает за коллизию игрока с картой; load(Texture texture, Font font) – загружает текстуру и шрифт для игрока; isPossesed() – проверяет, зарегистрирован ли игрок, возвращает логическое значение; draw(RenderWindow window) – отрисовка игрока; getRect() – возвращает прямоугольник игрока; drawVec(RenderWindow window) – отрисовка вектора других игроков на экране</p>
Bullet	<pre>class Bullet { int direction; float x, y, w, h, dx, dy, speed;</pre>	<p>Класс пули. Поля данных: direction – направление движения пули;</p>

Продолжение таблицы 2 – Структура типов программы

	<pre> Sprite sprite; Image BulletImage; Texture bulletTexture; bool life; FloatRect getRect(); void update(float time); } </pre>	<p>х, у, w, h, dx, dy, speed – координаты, ширина, высота, векторы движения и скорость пули;</p> <p>sprite – спрайт пули;</p> <p>bulletTexture – текстура пули;</p> <p>life – логическая переменная, отражает наличие пули.</p> <p>Методы класса:</p> <p>getRect() – возвращает прямоугольник пули;</p> <p>update(float time) – отвечает за обновление координат и спрайта пули, за коллизию пули с картой</p>
--	-----------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3.3 Схема алгоритмов решения задач по ГОСТ 19.701-90

3.3.1 Схема алгоритма Player::update

Схема алгоритма обновления движения танк.

В первую очередь осуществляется изменение векторов движения в зависимости от текущего направления. После этого происходит обновление самих координат танка и обнуление скорости.

Заключительным этапом является обновление позиций спрайта танка и его подписи. Вызывается функция проверки коллизии с объектами карты.



Рисунок 3.1 – Схема алгоритма Player::update

3.3.2 Схема алгоритма startGame

Схема алгоритма проверки пересечения танка с объектами карты позволяет проверить лишь те объекты, которые действительно мог пересечь танк, в зависимости от текущих координат. Таким образом обеспечивается максимальное быстродействие.

Алгоритм проверяет пересечение танка со стенками и объектом телепорта.

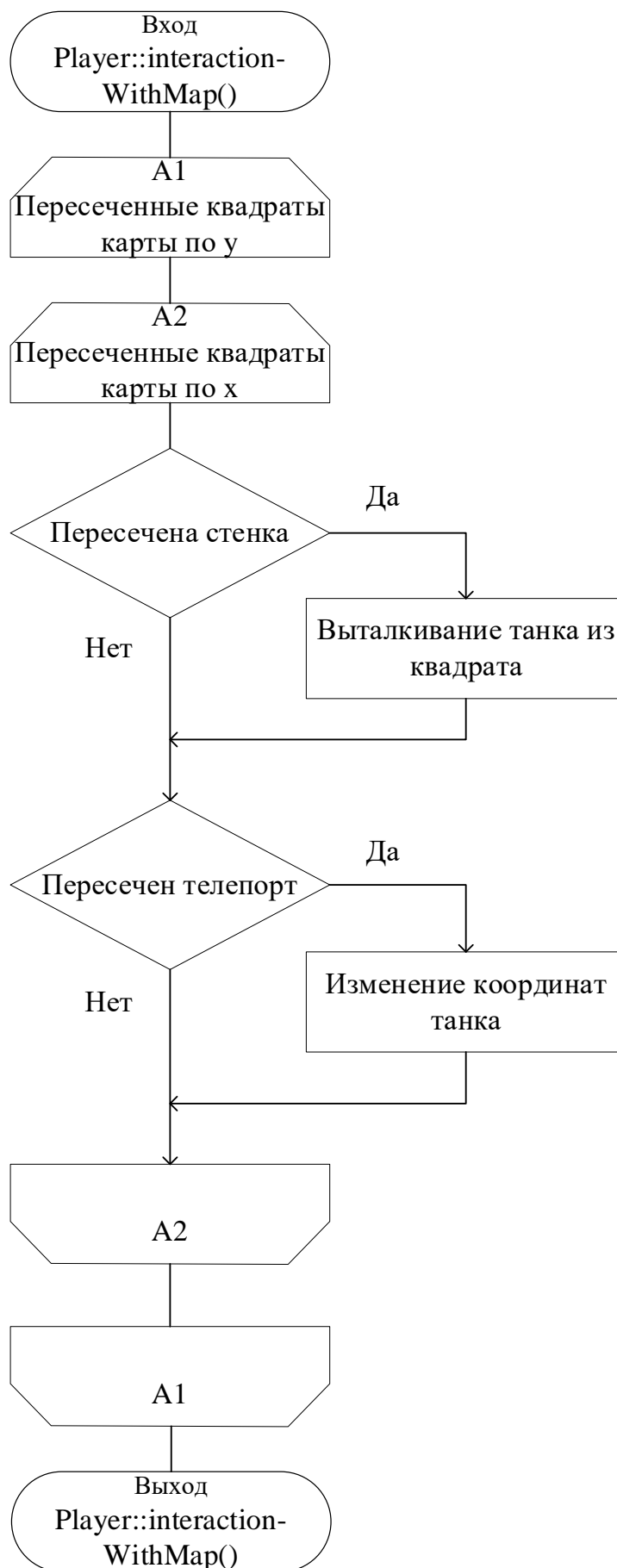


Рисунок 3.2 – Схема алгоритма Player::interactionWithMap

3.3.3 Схема алгоритма Bullet::update

Схема алгоритма обновления пули. Аналогично танку производится изменение векторов движения и координат. Помимо этого, проверяется пересечение пули со стенками карты, производится удаление пули при соударении со стеной.



Рисунок 3.3 – Схема алгоритма Bullet::update

3.3.4 Схема алгоритма LifeBar::draw

Схема алгоритма отрисовки и обновления полосы здоровья танка текущего игрока. Обновлению подлежат координаты полосы здоровья, координаты закрашивающего треугольника и его размер.

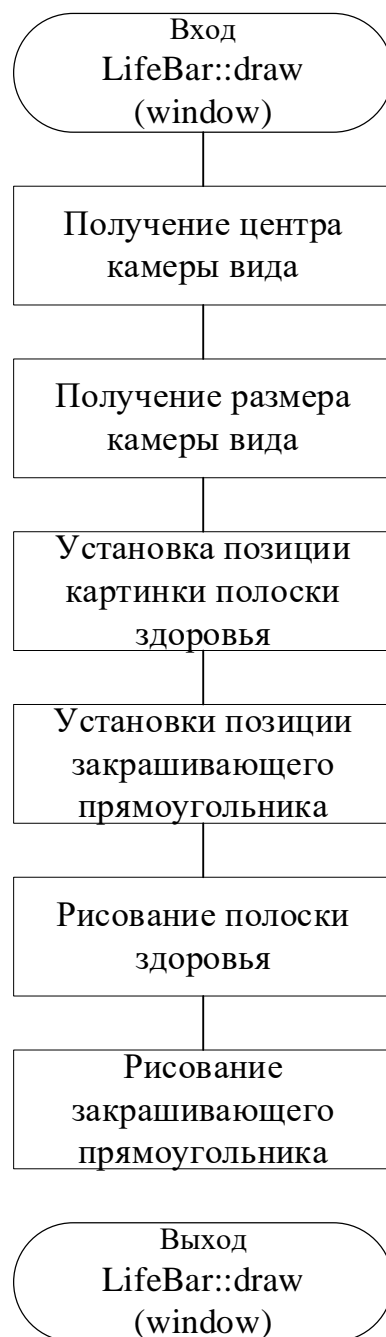


Рисунок 3.4 – Схема алгоритма LifeBar::draw

3.3.5 Схема алгоритма getplayercoordinateforview

Схема алгоритма получения камеры вида в соответствии с текущим положением танка игрока.

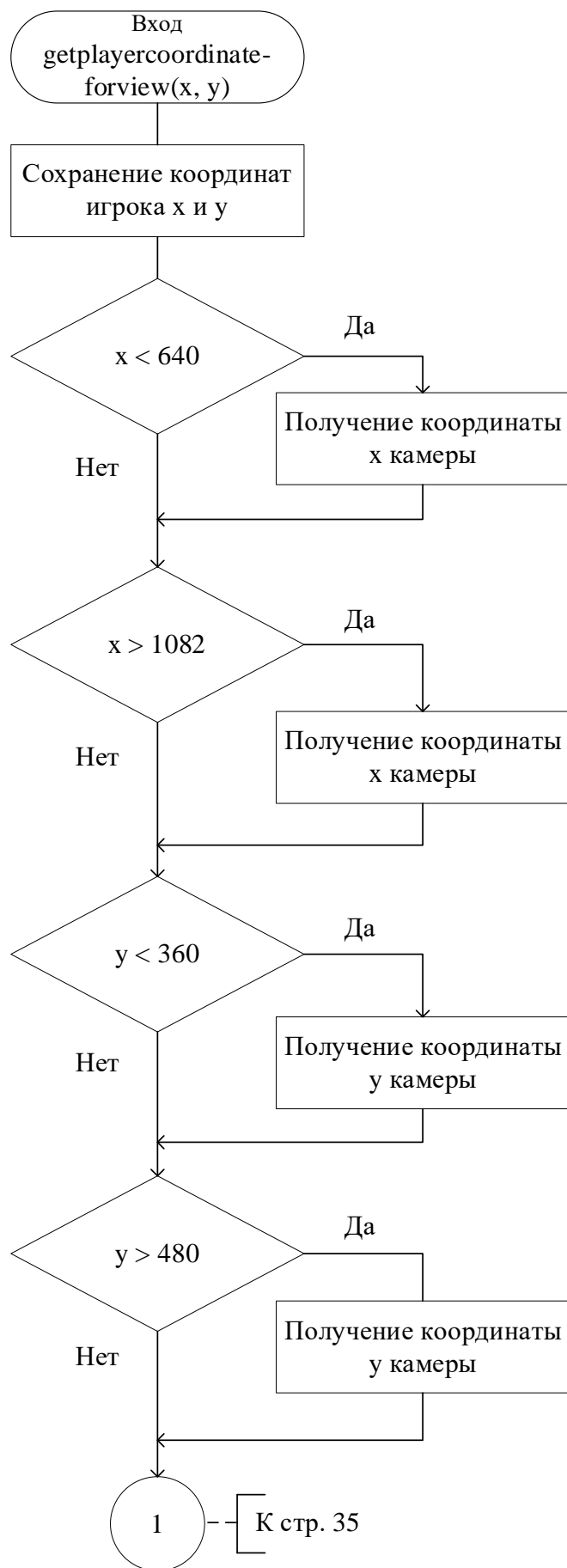


Рисунок 3.5 – Схема алгоритма `getplayercoordinateforview` (часть 1)

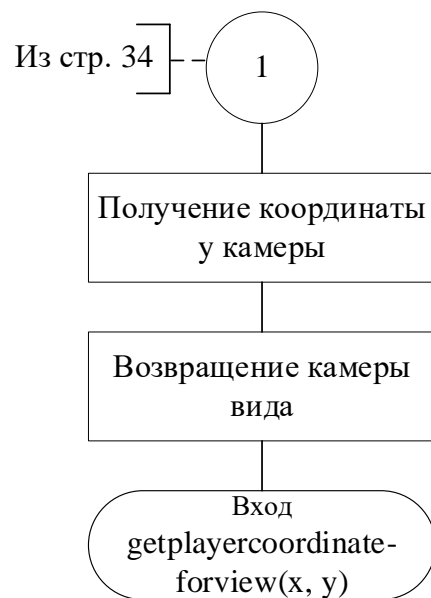


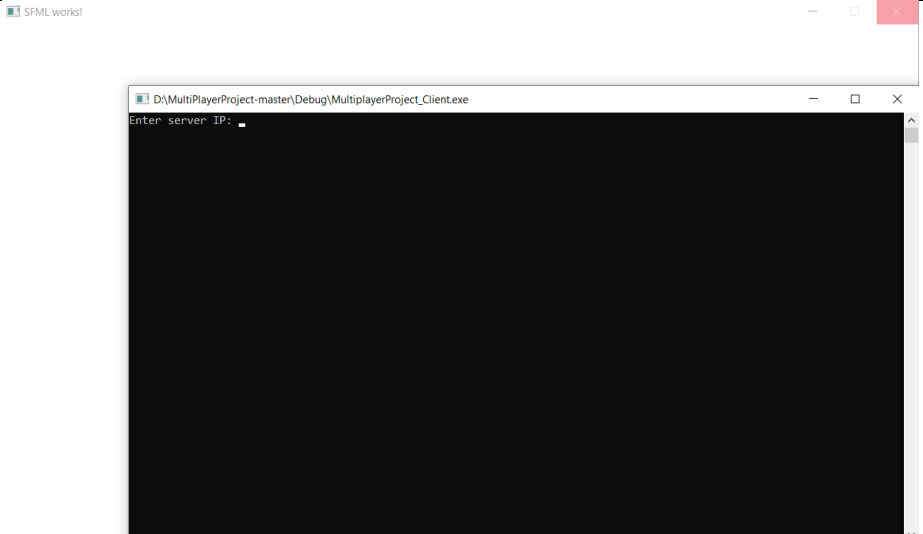
Рисунок 3.5 – Схема алгоритма getplayercoordinateforview (часть 2)

4 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

4.1 Запуск приложений клиента и сервера

4.1.1 Тест 1

Таблица 3 – Тест 1

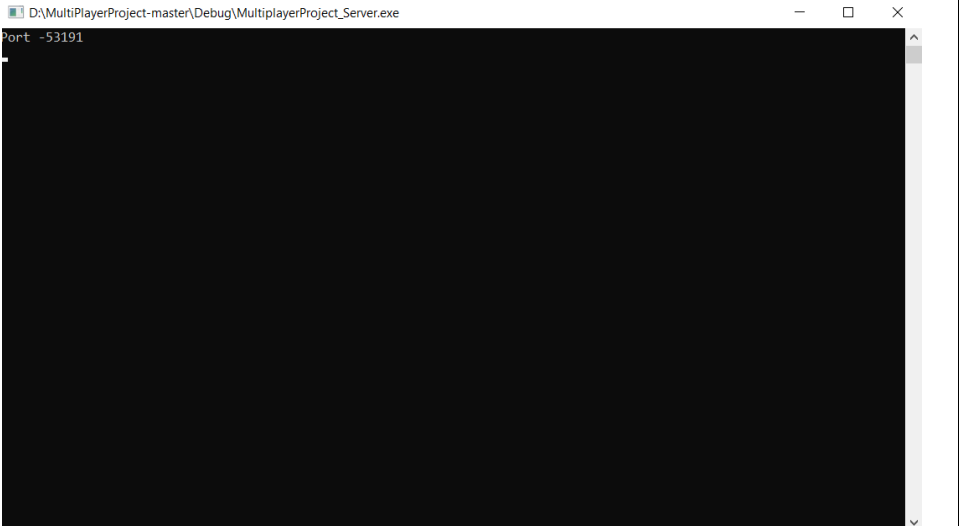
Тестовая ситуация:	Проверка корректности запуска приложения клиента
Исходный набор данных:	Запуск приложения игрока
Ожидаемый результат:	Корректная загрузка приложения и ожидание ввода ip
Полученный результат:	

4.1.2 Тест 2

Таблица 4 – Тест 2

Тестовая ситуация:	Проверка корректности запуска приложения сервера
Исходный набор данных:	Запуск приложения сервера
Ожидаемый результат:	Корректная загрузка приложения сервера и отображение порта подключения

Продолжение таблицы 5 – Тест 2

Полученный результат:	
-----------------------	------------------------------------------------------------------------------------

4.2 Подключение игрока/игроков к серверу

4.2.1 Тест 3

Таблица 6 – Тест 3

Тестовая ситуация:	Проверка подключения одного игрока к серверу
Исходный набор данных:	Работающий сервер и запущенное клиентское приложение
Ожидаемый результат:	Подключение к серверу(отображение лога на стороне сервера) и отображение карты на стороне игрока
Полученный результат:	

4.2.2 Тест 4

Таблица 7 – Тест 4

Тестовая ситуация:	Проверка подключения двух игроков к серверу
Исходный набор данных:	Работающий сервера и два запущенных клиентских приложения
Ожидаемый результат:	Отображение 2 игроков в приложении каждого игрока
Полученный результат:	

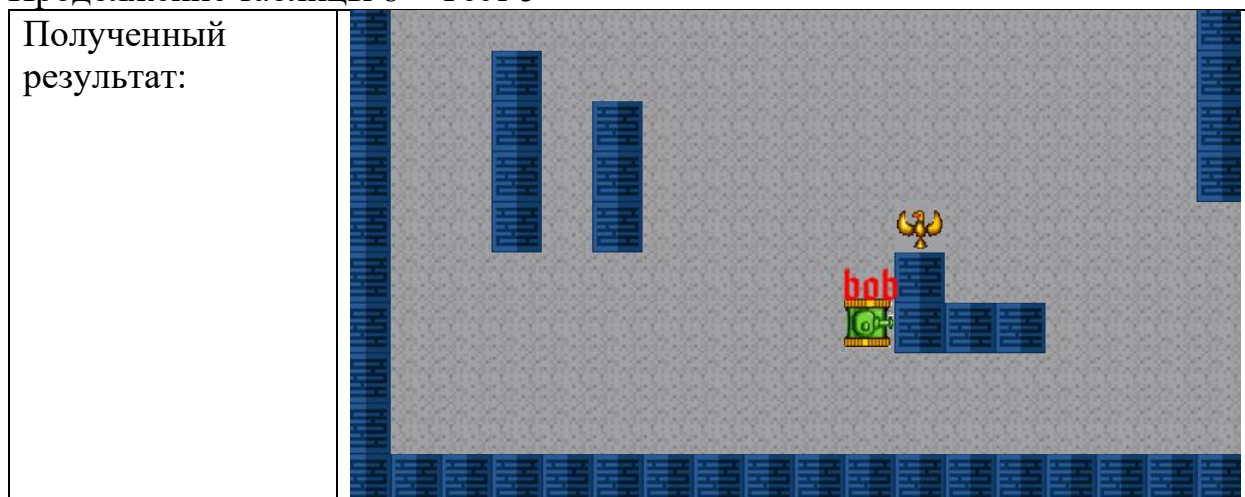
4.3 Взаимодействие танков между собой и со стенками карты

4.3.1 Тест 5

Таблица 8 – Тест 5

Тестовая ситуация:	Проверка взаимодействия танка с картой
Исходный набор данных:	Танк врезается в стенку карты
Ожидаемый результат:	Выталкивание танка из текстуры стены

Продолжение таблицы 8 – Тест 5



4.3.2 Тест 6

Таблица 9 – Тест 6

Тестовая ситуация:	Проверка пересечения пули и стенки карты
Исходный набор данных:	Выстрел танка
Ожидаемый результат:	Удаление пули при попадании в стену
Полученный результат:	Удаление пули при попадании в стену

4.3.3 Тест 7

Таблица 10 – Тест 7

Тестовая ситуация:	Проверка перемещения танка при пересечении иконки телепорта
Исходный набор данных:	Танк заехал на телепорт
Ожидаемый результат:	Мгновенное перемещение танка
Полученный результат:	Мгновенное перемещение танка

4.3.4 Тест 8

Таблица 11 – Тест 8

Тестовая ситуация:	Проверка синхронизации анимации танков у нескольких игроков
Исходный набор данных:	Два игрока на сервере
Ожидаемый результат:	Синхронизированные анимации движения танков, пуль и выстрелов
Полученный результат:	Синхронизированные анимации движения танков, пуль и выстрелов

4.3.1 Тест 9

Таблица 12 – Тест 9

Тестовая ситуация:	Проверка уменьшения здоровья игрока при попадании по нем пули
Исходный набор данных:	Два игрока на сервере
Ожидаемый результат:	Уменьшение здоровья одного из игроков
Полученный результат:	

5 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

5.1 Управление персонажем

Управление танка в игре происходит посредством взаимодействия игрока с клавиатурой. В игре предоставлены передвижения вверх, вниз, влево и вправо. Имеется возможность стрельбы

Клавиши управления и их комбинации:

- W, A, S, D – передвижение танка;
- Space – стрельба танка.

5.2 Подключение

Подключение игрока осуществляется путем ввода в консоль ip адреса сервера, порта сервера и имени игрока. Порт сервера выводится в консоль при старте приложения сервера.

ЗАКЛЮЧЕНИЕ

В конечном счете, разработанная игра в жанре аркады «2D Танки» представляет собой программное средство, которое предоставляет основные механики и возможности игр в этом жанре. Все поставленные задачи в рамках курсового проекта были выполнены.

Данная игра является простой в освоении и интуитивно понятной, так как все функции реализованы с использованием максимально понятных механик.

Для успешного выполнения всех поставленных целей потребовалось ознакомиться со средой разработки Visual Studio. Для создания графического интерфейса требовалось изучить различные аналоги игры в данном жанре. В то же время потребовалось изучить возможности библиотек SFML. Для работы с сетью потребовалось ознакомиться с SFML Network

Приложение прошло все этапы тестирования, в результате которых были устранены все неполадки. Приложение имеет относительно высокую скорость работы. Возможна дальнейшая доработка при выявлении ошибок.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- [1] Глухова Л.А. Основы алгоритмизации и программирования: лабораторный практикум / Л.А. Глухова, Е.П. Фадеева, Е.Е. Фадеева. – Минск: БГУИР, 2004. – 1 ч.
- [2] Глухова Л.А. Основы алгоритмизации и программирования: лабораторный практикум / Л.А. Глухова, Е.П. Фадеева, Е.Е. Фадеева. – Минск: БГУИР, 2005. – 2 ч.
- [3] Глухова Л.А. Основы алгоритмизации и программирования: лабораторный практикум / Л.А. Глухова, Е.П. Фадеева, Е.Е. Фадеева. – Минск: БГУИР, 2007. – 3 ч.
- [4] Глухова Л.А. Основы алгоритмизации и программирования: лабораторный практикум / Л.А. Глухова, Е.П. Фадеева, Е.Е. Фадеева. – Минск: БГУИР, 2013. – 4 ч.
- [5] Серебряная Л.В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л.В. Серебряная, И.М. Марина. – Минск: БГУИР, 2013. – 5 с.
- [6] Вирт Н. Алгоритмы и структуры данных / Н. Вирт. – Москва: Мир 1989. – 90 с.
- [7] Глухова Л.А. Основы алгоритмизации и программирования: учебное пособие / Л.А. Глухова. – Минск: БГУИР, 2006. – 1 ч.
- [8] Библиотека SFML[Электронный ресурс]. – Режим доступа: <https://www.sfml-dev.org/> – Дата обращения: 04.10.2022.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код программы
Модуль MultiplayerProject_Client

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Network.hpp>
#include <vector>
#include "map.h"
#include "view.h"
#include <list>
#include "LifeBar.h"
#include "NetworkClient.h"
using namespace sf;
using namespace std;
class Bullet;
class Player {
public:
    float x, y, w, h, dx, dy, speed;
    int dir;
    int health;
    Sprite sprite, netGhost;
    Text t;
    bool possessed = false;
    string name;
    bool turned = false;
    bool isShoot;
    float currentFrame;
    bool FirstShoot;
    Clock timeShootC;
    int sendBulletSize;
    bool changeBulSize;
    std::list<Bullet*> bullets;
    std::list<Bullet*>::iterator it;
public:
    Player(float X, float Y, float W, float H, bool
        possessed = false) :possesed(possesed) {
        dx = 0; dy = 0; speed = 0.08; dir = 3;
        w = W; h = H;
        x = X; y = Y;
        sprite.setTextureRect(IntRect(0, 0, w, h));
        currentFrame = 0;
        isShoot = false;
        FirstShoot = false;
        sendBulletSize = 0;
        changeBulSize = false;
        health = 100;
```

```

};

void update(float time)
{
    switch (dir)
    {
        case 0: dx = speed; dy = 0; break;
        case 1: dx = -speed; dy = 0; break;
        case 2: dx = 0; dy = speed; break;
        case 3: dx = 0; dy = -speed; break;
    }

    x += dx * time;
    y += dy * time;

    speed = 0;
    sprite.setPosition(x, y);
    t.setPosition(x + w / 2 -
        t.getGlobalBounds().width / 2, y -
        t.getGlobalBounds().height);
    interactionWithMap();
};

void interactionWithMap()
{
    for (int i = y / 42; i < (y + h) / 42; i++)
        for (int j = x / 42; j < (x + w) / 42; j++)
        {
            if (TileMap[i][j] == '0')
            {
                if (dy > 0)
                {
                    y = i * 42 - h;
                }
                if (dy < 0)
                {
                    y = i * 42 + 42;
                }
                if (dx > 0)
                {
                    x = j * 42 - w;
                }
                if (dx < 0)
                {
                    x = j * 42 + 42;
                }
            }
        }
}

```

```

        }

        if (TileMap[i][j] == 's') {
            x = 1300; y = 336;
        }
        if (TileMap[i][j] == 'c') {
            x = 378; y = 420;
        }
    }
}

void load(Texture& texture, Font& font)
{
    sprite.setTexture(texture);
    sprite.setTextureRect(IntRect(0, 0, w, h));
    if (!possesed) sprite.setColor(Color::Red);
    netGhost.setTexture(texture);

    t.setFont(font);
    t.setString(name);
    t.setFillColor(sf::Color::Red);
    t.setPosition(w / 2 -
        t.getGlobalBounds().width / 2, y -
        t.getGlobalBounds().height);
};

bool isPossesed() { return possesed; };

void draw(RenderWindow& window)
{
    window.draw(sprite);
    window.draw(t);
};

FloatRect getRect() {
    return FloatRect(x, y, w, h);
}

void drawVec(RenderWindow& window)
{
    switch (dir)
    {
    {
    case 0: {sprite.setTextureRect(IntRect(42 *
        int(currentFrame),
        42 * 3, 42, 42)); break; }
    case 1: {sprite.setTextureRect(IntRect(42 *
        int(currentFrame) + 42,
        42 * 3, -42, 42)); break; }
    }
    }
}

```

```

        case 2: { sprite.setTextureRect(IntRect(42 *
            int(currentFrame),
            42, 42, -42)); break; }
        case 3: { sprite.setTextureRect(IntRect(42 *
            int(currentFrame),
            0, 42, 42)); break; }
    }
    window.draw(sprite);
    window.draw(t);
};
};

```

```

class Bullet {
public:
    int direction;
    float x, y, w, h, dx, dy, speed;
    Sprite sprite;
    Image BulletImage;
    Texture bulletTexture;
    bool life;
    Bullet(float X, float Y, int W, int H, int dir) {
        x = X;
        y = Y;
        direction = dir;
        speed = 0.2;
        w = h = W;
        life = true;
        BulletImage.loadFromFile("images/tank.png");
        BulletImage.createMaskFromColor
            (sf::Color::White);
        bulletTexture.loadFromImage(BulletImage);
        sprite.setTexture(bulletTexture);
        sprite.setTextureRect(IntRect
            (42 * 7, 84, 42, 42));
        sprite.setPosition(x, y);
    }
    FloatRect getRect() {
        return FloatRect(x + 16, y + 16, 9, 9);
    }

    void update(float time)
    {
        switch (direction)
        {
            case 1: dx = -speed; dy = 0;    break;
            case 0: dx = speed; dy = 0;    break;

```

```

        case 2: dx = 0; dy = speed;    break;
        case 3: dx = 0; dy = -speed;   break;
    }
    x += dx * time;
    y += dy * time;
    if (x <= 0) x = 1;
    if (y <= 0) y = 1;

    for (int i = 0; i < HEIGHT_MAP; i++) {
        for (int j = 0; j < WIDTH_MAP; j++) {

            if (TileMap[i][j] == '0' &&
                getRect().intersects(FloatRect
                    (j * 42, i * 42, 42, 42)))
            {
                life = false;
            }
        }
        sprite.setPosition(x, y);
    }
};

vector<Player> playersVec;
IpAddress S_Ip;
unsigned short S_port;
string clientName;
NetworkClient netC;

void getUserInputData(string& playerName);
void addPlayer(Texture& t_player, Font& font, string
    clientName, int spawnX, int spawnY);

int main()
{
    RenderWindow window(sf::VideoMode(1280, 720),
        "SFML works!");

    Image playerImage;
    playerImage.loadFromFile("images/tank.png");
    playerImage.createMaskFromColor(sf::Color::White);
    playerImage.createMaskFromColor(Color(254, 254, 254)
);
    playerImage.createMaskFromColor(Color(254, 254,
        254));
    Texture t_player;
    t_player.loadFromImage(playerImage);
    Player player(250, 250, 42, 42, true);

```



```

view.reset(FloatRect(player.x, player.y, 1280,
    720));
LifeBar lifeBarPlayer;

Font font;
font.loadFromFile("fonts/Inkulinati-Regular.otf");
getUserInputData(player.name);
player.load(t_player, font);

netC.init();
netC.registerOnServer(S_Ip, S_port, player.name);

vector<string> namesVec;
netC.receiveConnectedClientsNames(namesVec);
for (int i = 0; i < namesVec.size(); i++)
{
    int tempCoordY = 0;
    int tempCoordX = 0;
    switch (i)
    {
        case 0: {
            int tempCoordX = 250;
            int tempCoordY = 250;
            break;
        }
        case 1: {
            int tempCoordX = 250;
            int tempCoordY = 714;
            break;
        }
        case 2: {
            int tempCoordX = 1340;
            int tempCoordY = 250;
            break;
        }
        default:
            int tempCoordX = 250;
            int tempCoordY = 250;
            break;
    }
    addPlayer(t_player, font, namesVec[i],
        tempCoordX, tempCoordY);
}

switch (namesVec.size())
{
    case 0: {
        player.x = 250;

```

```

        player.y = 250;
        view.reset(FloatRect(player.x, player.y, 1280,
            720));
        break;
    }
    case 1: {
        player.x = 250;
        player.y = 714;
        view.reset(FloatRect(player.x, player.y, 1280,
            720));
        break;
    }
    case 2: {
        player.x = 1340;
        player.y = 250;
        view.reset(FloatRect(player.x, player.y, 1280,
            720));
        break;
    }
    default: {
        player.x = 1340;
        player.y = 714;
        view.reset(FloatRect(player.x, player.y, 1280,
            720));
        break;
    }
}

```

```

Packet receivedDataPacket;
Packet sendDataPacket;
Clock clock;
Image map_image;
map_image.loadFromFile("images/tank.png");
Texture map;
map.loadFromImage(map_image);
Sprite s_map;
s_map.setTexture(map);

```

```

while (window.isOpen())
{
    float time =
        clock.getElapsedTime().asMicroseconds();
    clock.restart();
    time = time / 800;

    if (netC.receiveData(receivedDataPacket, S_Ip,
        S_port) == Socket::Status::Done)
    {

```

```

if (receivedDataPacket.getDataSize() > 0)
{
    string s;
    if (receivedDataPacket >> s)
    {
        if (s == "NEW")
        {
            if (receivedDataPacket >> s)
            {
                if (s != clientName)
                {
                    int tempCoordX = 0;
                    int tempCoordY = 0;
                    switch
                    (playersVec.size())
                    {
                        case 0: {
                            tempCoordX =
                                250;
                            tempCoordY =
                                714;
                            break;
                        }
                        case 1: {
                            tempCoordX =
                                1340;
                            tempCoordY =
                                250;
                            break;
                        }
                        case 2: {
                            tempCoordX =
                                1340;
                            tempCoordY =
                                714;
                            break;
                        }
                        default:
                            tempCoordX =
                                1340;
                            tempCoordY =
                                714;
                            break;
                    }
                    addPlayer(t_player,
font, s, tempCoordX,
tempCoordY);
                    cout << "New player

```

```

        connected: " <<
            playersVec.back().name
        << endl;
    }
}

if (s == "DATA")
{
    while(!receivedDataPacket.
        endOfPacket())
    {
        float x, y;
        receivedDataPacket >>
            s;
        receivedDataPacket >>
            x;
        receivedDataPacket >>
            y;
        bool turn;
        receivedDataPacket >>
            turn;
        int direction;
        receivedDataPacket >>
            direction;
        float curFrame;
        receivedDataPacket >>
            curFrame;
        int size;
        receivedDataPacket >>
            size;
        bool changeSize;
        receivedDataPacket >>
            changeSize;
        for (int i = 0; i <
            playersVec.size(); i++)
        {
            if (s ==
                playersVec[i].
                name) {

                playersVec[i].turned = turn;
                playersVec[i].x = x;
                playersVec[i].y = y;
                playersVec[i].
                    sprite.setPosition(x, y);
                playersVec[i].

t.setPosition(x + playersVec[i].w / 2 -

```



```

        playersVec[i].bullets.push_back(new
Bullet(playersVec[i].x, playersVec[i].y, 42, 42,
playersVec[i].dir));
    }

    if (window.hasFocus())
    {
        if ((Keyboard::isKeyPressed
            (Keyboard::Left) ||
            (Keyboard::isKeyPressed
            (Keyboard::A))) {
            player.dir = 1; player.speed = 0.08;
            player.currentFrame += 0.005 * time;
            if (player.currentFrame > 8)
                player.currentFrame -= 8;

            player.sprite.setTextureRect(IntRect(42 *
                int(player.currentFrame) + 42, 42 * 3, -42, 42));
        }

        if ((Keyboard::isKeyPressed
            (Keyboard::Right) ||
            (Keyboard::isKeyPressed
            (Keyboard::D))) {
            player.dir = 0; player.speed = 0.08;
            player.currentFrame += 0.005 * time;
            if (player.currentFrame > 8)
                player.currentFrame -= 8;

            player.sprite.setTextureRect(IntRect(42 *
                int(player.currentFrame), 42 * 3, 42, 42));
        }

        if ((Keyboard::isKeyPressed
            (Keyboard::Up) ||
            (Keyboard::isKeyPressed
            (Keyboard::W))) {
            player.dir = 3; player.speed = 0.08;
            player.currentFrame += 0.005 * time;
            if (player.currentFrame > 8)
                player.currentFrame -= 8;

            player.sprite.setTextureRect(IntRect(42 *
                int(player.currentFrame), 0, 42, 42));
        }

        if ((Keyboard::isKeyPressed

```

```

        (Keyboard::Down) || (Keyboard::isKeyPressed
        (Keyboard::S)))) {
            player.dir = 2; player.speed = 0.08;
            player.currentFrame += 0.005 * time;
            if (player.currentFrame > 8)
                player.currentFrame -= 8;

        player.sprite.setTextureRect(IntRect(42 *
        int(player.currentFrame), 42, 42, -42));
        }
        if
        (Keyboard::isKeyPressed(Keyboard::Space) &&
        (player.FirstShoot == false ||
        player.timeShootC.getElapsedTime().asSeconds() > 0.5)) {
            player.isShoot = true;

        }
    }

    Getplayercoordinateforview
    (player.x, player.y);
    for (player.it = player.bullets.begin();
    player.it != player.bullets.end();)
    {
        Bullet* b = *player.it;
        b->update(time);
        if (b->life == false) { player.it =
            player.bullets.erase(player.it);
            delete b; }
        else player.it++;
    }

    for (int i = 0; i < playersVec.size(); i++)
    {
        if (playersVec[i].health > 0) {
            std::list<Bullet*>::iterator
            iterator;
            for (iterator =
            playersVec[i].bullets.begin();
            iterator != playersVec[i].
            bullets.end();)
            {
                Bullet* b = *iterator;
                b->update(time);
                if (b->life == false) { iterator
                    = playersVec[i].bullets.
                    erase(iterator); delete b; }
                else iterator++;
            }
        }
    }

```

```

        }
    }
}

player.update(time);
lifeBarPlayer.update(player.health);

for (int i = 0; i < playersVec.size(); i++)
{
    if (player.getRect().intersects
        (playersVec[i].getRect()) &&
        playersVec[i].health > 0) {
        if (player.dy > 0)
        {
            player.y = playersVec[i].y -
                player.h;
        }
        if (player.dy < 0)
        {
            player.y = playersVec[i].y +
                player.h;
        }
        if (player.dx > 0)
        {
            player.x = playersVec[i].x -
                player.w;
        }
        if (player.dx < 0)
        {
            player.x = playersVec[i].x +
                player.w;
        }
    }
}

window.setView(view);
window.clear();

for (int i = 0; i < HEIGHT_MAP; i++)
{
    for (int j = 0; j < WIDTH_MAP - 1; j++)
    {
        if (TileMap[i][j] == ' ')
            s_map.setTextureRect(IntRect
                (0, 42 * 2, 42, 42));
        if ((TileMap[i][j] == 's') ||
            (TileMap[i][j] == 'c'))
            s_map.setTextureRect(IntRect

```



```

        (42 * 3, 42 * 2, 42, 42));
    if ((TileMap[i][j] == '0'))
        s_map.setTextureRect(IntRect
            (42 * 2, 42 * 2, 42, 42));
    s_map.setPosition(j * 42, i * 42);
    window.draw(s_map);
}
}

for (int i = 0; i < playersVec.size(); i++)
{
    if (playersVec[i].health > 0) {
        std::list<Bullet*>::iterator
            iterator;
        for (iterator =
            playersVec[i].bullets.begin();
            iterator != playersVec[i].
                bullets.end(); iterator++)
        {
            if ((*iterator)-
                >getRect().intersects
                    (player.getRect())) {
                (*iterator)->life = false;
                player.health -= 50;
            }
            for (int j = 0; j < i; j++)
            {
                if ((*iterator)-
                    >getRect().intersects
                        (playersVec[j].
                            getRect()) &&
                            playersVec[j].
                                health > 0) {
                    (*iterator)->life =
                        false;
                    playersVec[j].health -=
                        50;
                }
            }
            for (int j = i + 1; j <
                playersVec.size(); j++)
            {
                if ((*iterator)->getRect().
                    intersects(
                        playersVec[j].getRect())
                    && playersVec[j].
                        health > 0) {
                    (*iterator)->life =

```

```

        false;
        playersVec[j].health -=
            50;
    }
}
}
}

for (player.it = player.bullets.begin();
     player.it != player.bullets.end();
     player.it++)
{
    for (int i = 0; i <
          playersVec.size(); i++)
    {
        if (playersVec[i].health > 0) {
            if ((*player.it)-
                >getRect().intersects
                (playersVec[i].getRect())){
                (*player.it)->life = false;
                playersVec[i].health -= 50;
            }
        }
    }
}

for (int i = 0; i < playersVec.size(); i++)
{
    if (playersVec[i].health > 0) {
        std::list<Bullet*>::iterator
            iterator;
        for (iterator =
              playersVec[i].bullets.begin();
              iterator != playersVec[i].
              bullets.end();
              iterator++)
        {
            window.draw((*iterator)-
                        >sprite);
        }
        playersVec[i].drawVec(window);
    }
}

for (player.it = player.bullets.begin();
     player.it != player.bullets.end();

```

```

        player.it++)
    {
        window.draw((*player.it)->sprite);
    }

    player.draw(window);
    if (player.health <= 0) {
        break;
    }
    lifeBarPlayer.draw(window);
    window.display();
}
return 0;
}

void getUserInputData(string& playerName)
{
    cout << "Enter server IP: ";
    cin >> S_Ip;
    //S_Ip = "localhost";
    cout << endl;
    cout << "Enter server registration port: ";
    cin >> S_port;
    cout << endl;
    cout << "Enter name: ";
    cin >> playerName;
};

void addPlayer(Texture& t_player, Font& font, string
    clientName, int spawnX, int spawnY)
{
    Player p(spawnX, spawnY, 42, 42, true);
    playersVec.push_back(p);
    playersVec.back().name = clientName;
    playersVec.back().load(t_player, font);
};

```

ПРИЛОЖЕНИЕ Б
(обязательное)
Исходный код программы
Модуль MultiplayerProject_Server

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Network.hpp>
#include <vector>
#include "NetworkServer.h"
using namespace sf;
using namespace std;

NetworkServer netS;
int main()
{
    netS.init();
    Packet packet;
    packet << "DATA";
    for (int i = 0; i < netS.clientsVec.size(); i++)
    {
        packet << netS.clientsVec[i].name <<
        netS.clientsVec[i].pos.x <<
        netS.clientsVec[i].pos.y <<
        netS.clientsVec[i].turned <<
        netS.clientsVec[i].direction <<
        netS.clientsVec[i].curFrame <<
        netS.clientsVec[i].bulSize <<
        netS.clientsVec[i].changeBulSize;
    }
    while (true)
    {
        netS.registerNewClients();
        netS.sendConnectedClientsRecords();
        if (netS.sendDataToAll(packet) ==
            Socket::Status::Done)
        {
            packet.clear();
            packet << "DATA";
            for (int i = 0; i <
                netS.clientsVec.size(); i++)
            {
                packet << netS.clientsVec[i].name <<
                netS.clientsVec[i].pos.x <<
                netS.clientsVec[i].pos.y <<
                netS.clientsVec[i].turned <<
                netS.clientsVec[i].direction <<
                netS.clientsVec[i].curFrame <<
```

```

        netS.clientsVec[i].bulSize <<
        netS.clientsVec[i].changeBulSize;
    }
}
unsigned int receivedClientIndex;
if (netS.receiveData(receivedClientIndex) ==
    Socket::Status::Done)
{
    if (netS.clientsVec[receivedClientIndex].
        rDataPacket.getDataSize() > 0)
    {
        string s;
        if
        (netS.clientsVec[receivedClientIndex].
            rDataPacket >> s)
        {
            if (s == "DATA")
            {
                float x, y;
                if (netS.clientsVec
                    [receivedClientIndex].
                        rDataPacket >> x)
                {
                    netS.clientsVec
                    [receivedClientIndex].
                        pos.x = x;
                }
                if (netS.clientsVec
                    [receivedClientIndex].
                        rDataPacket >> y)
                {
                    netS.clientsVec
                    [receivedClientIndex].
                        pos.y = y;
                }
                bool turn;
                if (netS.clientsVec
                    [receivedClientIndex].
                        rDataPacket >> turn)
                {
                    netS.clientsVec
                    [receivedClientIndex].
                        turned = turn;
                }
                int dn;
                if (netS.clientsVec
                    [receivedClientIndex].
                        rDataPacket >> dn)

```

```

        {
            netS.clientsVec
                [receivedClientIndex].
                direction = dn;
        }
        float frame;
        if (netS.clientsVec
            [receivedClientIndex].
            rDataPacket >> frame)
        {
            netS.clientsVec
                [receivedClientIndex].
                curFrame = frame;
        }
        int bulNum;
        if (netS.clientsVec
            [receivedClientIndex].
            rDataPacket >> bulNum)
        {
            netS.clientsVec
                [receivedClientIndex].
                bulSize = bulNum;
        }
        bool changeSize;
        netS.clientsVec
            [receivedClientIndex].
            changeBulSize = false;
        if (netS.clientsVec
            [receivedClientIndex].
            rDataPacket >>
            changeSize)
        {
            netS.clientsVec
                [receivedClientIndex].
                changeBulSize =
                    changeSize;
        }
        netS.clientsVec
            [receivedClientIndex].
            rDataPacket.clear();
    }
}

}

}

}

getchar();
return 0;
}

```