

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

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системное программирование (СП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовому проекту  
на тему

Игра «Змейка»

БГУИР КП 1-40 01 01 009 ПЗ

Студент: гр. 151003 Матошко И.В.

Руководитель: асс. Низовцов Д.В.

Минск 2023

Учреждение образования

«Белорусский государственный университет информатики и  
радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ  
Заведующий кафедрой ПОИТ

\_\_\_\_\_  
(подпись)  
Лапицкая Н.В. 2023г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Матошко Ивану Викторовичу

1. Тема работы Игра «Змейка»

2. Срок сдачи студентом законченной работы 19.12.2023г.

3. Исходные данные к работе Среда программирования Visual Studio. Наличие графической реализации интерфейса игры. Возможность двумерного управления одним персонажем. Возможность взаимодействия карты со змейкой. В игре реализованы следующие возможности: рестарт игры, сбор игровых очков, завершение игры при столкновении с картой, змейкой или сборе самого большого количества игровых очков.

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 Низовцов Д.В.

7. Дата выдачи задания 20.10.2023г.

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

Раздел 1. Введение к 30.10.2023г. – 10 % готовности работы;

Раздел 2 к 15.10.2023г. – 30% готовности работы;

Раздел 3 к 15.11.2023г. – 60% готовности работы;

Разделы 4, 5, Заключение к 09.12.2023 – 90 % готовности работы;

Оформление пояснительной записки и графического материала к 10.12.2023 – 100 % готовности работы.

Защита курсового проекта с 18.12.2023г. по 19.12.2023г.

РУКОВОДИТЕЛЬ \_\_\_\_\_ Низовцов Д.В.  
(подпись)

Задание принял к исполнению \_\_\_\_\_ Матошко И.В. 20.10.2023г.  
(дата и подпись студента)

## СОДЕРЖАНИЕ

Введение.....	5
1 анализ литературных источников.....	6
1.1 Анализ существующих аналогов.....	6
1.2 Анализ методов и способов разработки .....	9
2 Постановка задачи.....	13
2.1 Назначение разработки.....	13
2.2 Перечень функциональных требований .....	13
2.3 Структура программы.....	14
2.4 Входные и выходные параметры.....	14
2.5 Состав и параметры технических и программных средств .....	15
3 Разработка программного средства.....	16
3.1 Описание алгоритмов решения задачи .....	16
3.2 Структура данных .....	19
3.3 Схема алгоритмов решения задач по ГОСТ 19.701-90 .....	23
4 тестирование и проверка работоспособности программного средства.....	36
4.1 Запуск игры/перезапуск игры .....	36
4.2 Управление змейкой .....	37
4.3 Взаимодействие змейки и объектов карты.....	39
5 Руководство по использованию программного средства .....	41
5.1 Задача игры и прохождение уровней .....	41
5.2 Управление персонажем.....	41
5.3 Сбор игровых очков .....	41
5.4 Дополнительные опции .....	41
Заключение .....	42
Список использованной литературы.....	43
Приложение А .....	44
Приложение Б .....	51
Приложение В.....	52
Приложение Г .....	55
Приложение Д.....	57
Приложение Е.....	58

## ВВЕДЕНИЕ

Данная работа посвящена созданию игры «Змейка».

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.

Змейка, на самом деле, является целым жанром компьютерных игр. В них игрок управляет «головой», растущей линией змеи, и не должен позволить ей столкнуться с препятствиями, в том числе со своим «хвостом».

Примеры: Tron, Google Snake – Snake Game, Slither.io.

Целью работы является создание проекта игры-змейки, сопровождающуюся документацией в виде пояснительной записки.

# 1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

## 1.1 Анализ существующих аналогов

Игры «Змейка» представляют собой игры, в которых основу игрового процесса составляют управление длинным, тонким существом, напоминающим змею, которое ползает по плоскости (как правило, ограниченной стенками), собирая еду (или другие предметы), избегая столкновения с собственным хвостом и краями игрового поля (существуют варианты, где при прохождении через край змея выходит из противоположного края поля). Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру. В другом варианте двое играют двумя такими змеями так, чтобы вынудить соперника врезаться во что-то. Примером усложненной версии игры может быть известная браузерная игра Slither.io, где множество игроков соревнуются по сети Интернет. Сегодня существует много представителей «Змейки»: «Tron», «Google Snake – Snake Game», «Slither.io» и др.

### 1.1.1 Игра «Tron»

«Tron» – видеоигра, подхватила историю оригинального фильма много лет спустя. Примечательно, что главным персонажем игры считается сын главного героя фильма. Цель игры проста – победить противника, вынудив столкнуться его с препятствием, а именно краем карты или световым шлейфом. Игрок передвигается на мотоцикле будущего, который оставляет за собой цветной шлейф. При столкновении вражеского персонажа со шлейфом, он проигрывает. Интересно, что игра является именно модификацией «Змейки», поскольку игроку не требуется собирать игровые очки, а элемент усложнения игры состоит в высокой скорости передвижения персонажей.

Достоинства игры:

- казуальность;
- музыкальное сопровождение.

В свою очередь к недостаткам можно отнести:

- малая разновидность текстур с невысоким качеством.



Рисунок 1.1 – «Tron»

#### 1.1.2 Игра «Google Snake – Snake Game»

«Google Snake – Snake Game» – мобильная игра в жанре «Змейки». Google Snake Game – это забавная игра, в которой вы играете роль змеи, и ваша цель – есть фрукты, чтобы стать больше. Тип фруктов, которые ест змея, зависит от выбора игрока. В Google Snake есть много видов фруктов на выбор. Фрукт по умолчанию — яблоко. Некоторые дополнительные фрукты — персики, арбузы, виноград и многие другие. Google Snake также предлагает множество игровых режимов на выбор. Основным игровым режимом — оригинальный Google Snake, то есть оригинальная «Змейка». В этом режиме цель игрока — съесть много фруктов и не врезаться в стену или хвост. Некоторые из других игровых режимов, которые предлагает Google Snake, — это исчезающие блоки, бесконечная карта, телепортирующаяся змея и многое другое на выбор.

Достоинства игры:

- высокий уровень графики;
- уникальные механики;
- кастомизация;
- отличное музыкальное сопровождение;
- наличие браузерной версии для ПК.

Недостатки:

- низкая оптимизация для смартфонов;
- необходимость наличия Интернета и реклама.

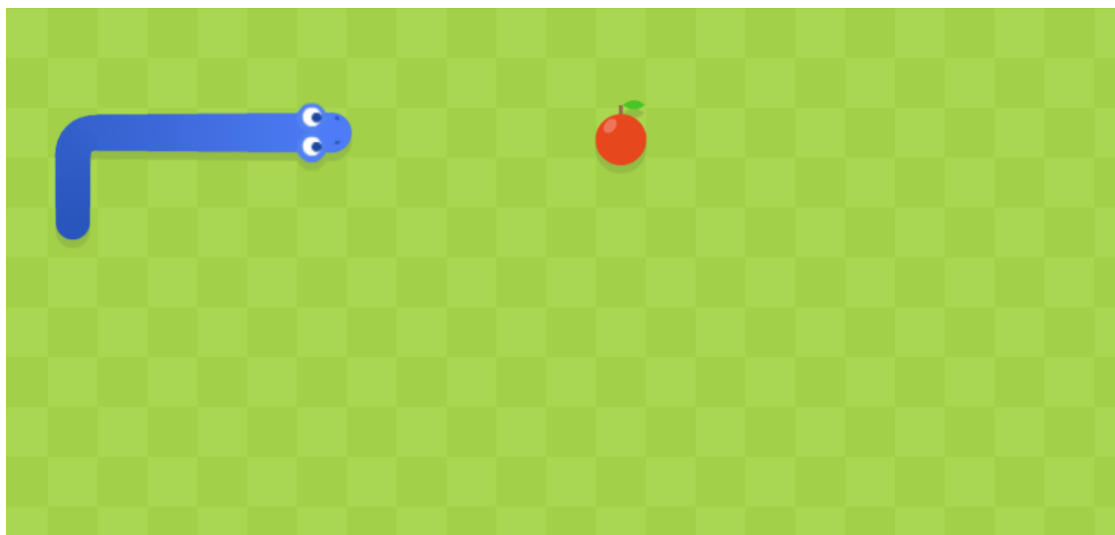


Рисунок 1.2 – «Google Snake – Snake Game»

### 1.1.3 Игра «Slither.io»

«Slither.io» – многопользовательская компьютерная игра, разработанная Стивом Хаузом и выпущенная для iOS, Android и веб-браузеров в марте 2016 года. По своей концепции Slither.io похожа на классическую «Змейку» и другую популярную браузерную игру Agar.io. В Slither.io игрок управляет «змеей» — длинным существом, которое передвигается по обширному игровому полю и поедает разноцветные гранулы; с каждой съеденной гранулой змея становится длиннее. Гранулы появляются на игровом поле случайным образом и также остаются от погибших змей. В начале игры игрок получает под управление короткую змею, которая сразу же начинает движение по полю; игрок направляет её движение, подбирая гранулы. Если голова змеи столкнётся со змеей другого игрока, игра заканчивается; тело погибшей змеи рассыпается на яркие и достаточно крупные гранулы, которые могут съесть змеи других игроков. Игровое поле в игре не бесконечно и имеет границы. Если змея выйдет за границу поля, она немедленно погибнет и не оставит после себя гранул. В правом верхнем углу отображаются рейтинг из десяти игроков-лидеров с самыми длинными змеями.

Достоинства игры:

- игра в режиме онлайн с другими игроками;
- наличие кастомизации;
- кроссплатформенность;
- плавная 3D графика;
- низкий порог вхождения.

В свою очередь к недостаткам можно отнести:

- чрезвычайно раздражающая реклама после проигрыша;
- нет полноценного локального кооператива.





Рисунок 1.3 – «Slither.io»

## 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 змейки. Змейка должна быть способна двигаться в четырех направлениях. В то же время необходимо отладить взаимодействие между частями змейки, игровыми очками и краями карты, голову змейки, центральные части тела и хвост, врагов, движущиеся платформы и др. Для этих целей будет использована сторонняя библиотека SFML, а также интерфейс Win 32 API.

В разработке применяется Win 32 API. Интерфейс предоставляет прямой способ взаимодействия приложений пользователя с операционной системой Windows[5]. В данной работе будет производиться интеграция графической библиотеки SFML с указанным интерфейсом. Главное окно игры будет создаваться с помощью Win 32 API, которое затем будет передаваться SFML, для рисования графики. Это позволит пользоваться как высококачественными элементами библиотеки SFML и отображать стандартные элементы Win 32 API в одном и том же окне.

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[2].

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.

В то же время ставится задача создания квадратной двумерной графики с высоким разрешением. Будет использован тайловый, или плиточный, метод создания карты и изображения различных объектов. Тайлы – небольшие изображения одинаковых размеров, служащие фрагментами большой картины. Количество тайлов на один «мир» может достигать нескольких сотен. Матрица клеток при этом хранит только номера тайлов, за счет чего достигается экономия памяти при построении огромных двумерных пространств. Для этих задач подходящим решением является часть сторонней библиотеки TinyXML, а также редактор тайл-карт Tiled.

Tiled – кроссплатформенный открытый редактор тайловых карт для игр[4]. Он позволяет создавать карты для 2-мерных игр (с видом сбоку, таких, как платформеров, или видом сверху, к примеру JRPG). Начиная с версии 0.11.0 поддерживаются гексагональные тайлы. Tiled был использован при разработке достаточно большого числа игр, как свободных, так и проприетарных.

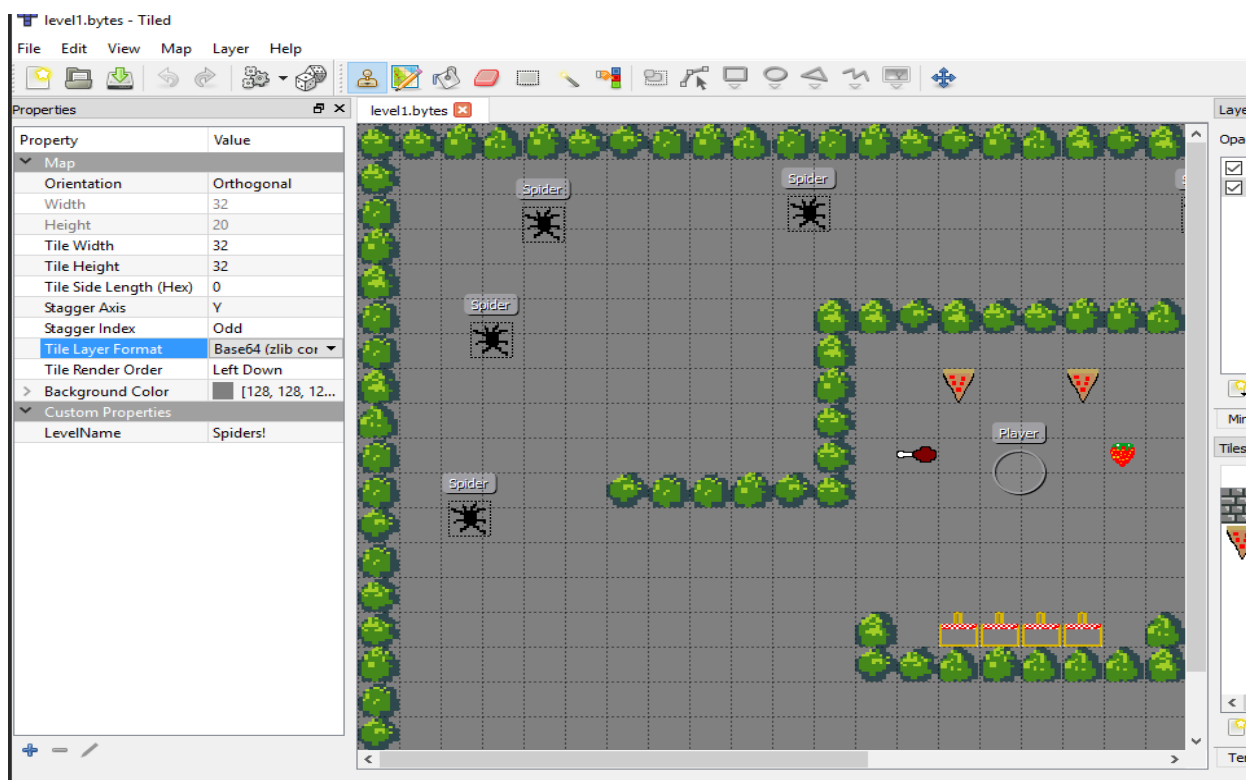


Рисунок 1.4 – Интерфейс редактора тайловых карт Tiled

Редактор сохраняет уровни в основном на XML формате, чтение которого обеспечивает ряд библиотек для различных языков программирования. Одной из таких библиотек является TinyXML.

TinyXML – простой, небольшой, бесплатный C++ анализатор XML[3]. TinyXML – одно из многих средств, используемых для анализа XML документов, обладает удобным и компактным внешним интерфейсом, не требует специальных знаний и длительного обучения для использования.

В качестве языка программирования для классического приложения Win 32 API будет использован C++. Основанием для использования является поддержка используемыми библиотеками данного языка программирования. В то же время для проектирования игр удобно использовать объектно-ориентированное программирование. Оно позволит не описывать каждый объекта заново (даже если они практически одинаковые), а создать родительский класс, от которого будут наследоваться все похожие между собой объекты.

### 1.2.2 Используемые структуры данных

В данном проекте будут использоваться две основные структуры данных:

- список (list);
- вектор (vector).

В данной работе используется структура list предназначена для хранения всех объектов, загруженных с карты формата XML. Это структура данных, которая построена на двусвязных списках. Это значит, что из любого элемента можно получить доступ к предыдущему и последующим элементам[1]. Структура двусвязного списка list представлена на рисунке 1.5.

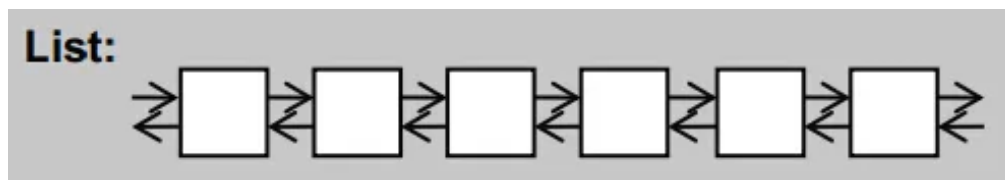


Рисунок 1.5 – Двусвязный список

В свою очередь структура vector предназначена для первоначальной загрузки объектов и их хранения, до того, как они попадут в общую структуру list. Кроме того, все элементы змейки, кроме головы и хвоста, хранятся в общем векторе для удобства взаимодействия. Вектор – это структура данных, которая является моделью динамического массива[1]. Отличительной чертой данной реализации массива является отсутствие необходимости пользоваться указателями и ручным созданием динамического массива. Структура vector представлена на рисунке 1.6.

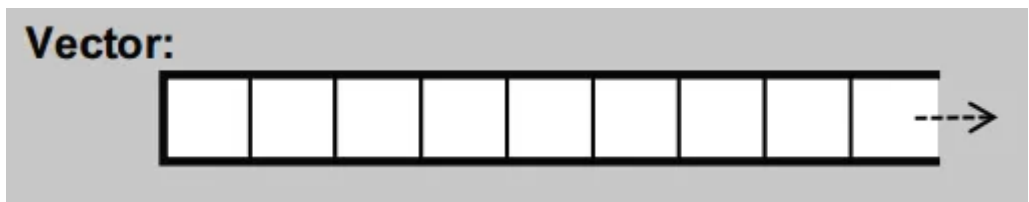


Рисунок 1.6 – Вектор

## **2 ПОСТАНОВКА ЗАДАЧИ**

### **2.1 Назначение разработки**

Назначением проектирования является разработка игры в жанре «Змейка». 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 частей змейки;
- разработка алгоритмов взаимодействия объектов между собой;
- разработка алгоритма загрузки карты уровня из внешнего файла;
- разработка алгоритма загрузки объектов карты из внешнего файла;
- разработка алгоритмов отрисовки анимации объектов;
- разработка алгоритмов отображения сопровождающей информации;
- тестирование приложения.

### **2.2 Перечень функциональных требований**

Целью разработки игры в жанре «Змейка» является объединение основных достоинств рассмотренных существующих аналогов, а также компенсация недостатков этих игр. В результате разработки необходимо предоставить реализацию следующих функций:

- загрузка карты из внешнего файла;
- управление змейкой в декартовой системе координат;
- взаимодействие между частями змейки;
- взаимодействие змейкой и картой;
- взаимодействие между змейкой и игровыми очками;
- отображение количества очков.

## 2.3 Структура программы

При разработке приложения будет использовано 9 модулей:

- main.cpp – главный модуль, содержащий отображаемое окно игры;
- Apple.cpp (Apple.h) – модуль, представляющий собой игровые очки;
- Snake.cpp (Snake.h) – модуль, представляющий собой все части змейки как единое целое;
- SnakeBody.cpp (SnakeBody.h) – модуль, представляющий собой тело змейки (центральные части);
- SnakeHead.cpp (SnakeHead.h) – модуль, представляющий собой голову змейки;
- SnakeTail.cpp (SnakeTail.h) – модуль, представляющий собой хвост змейки;
- SnakePart.cpp (SnakePart.h) – модуль, представляющий базовый класс для всех частей змейки;
- level.h – модуль для работы с картой формата XML;
- tinyxml.cpp (tinyxml.h) – модуль, является частью сторонней библиотеки для работы с файлами формата XML;
- tinystl.cpp (tinystl.h) – модуль, является частью сторонней библиотеки для работы с файлами формата XML;
- tinyxmlerror.cpp – модуль, является частью сторонней библиотеки для работы с файлами формата XML;
- tinyxmlparser.cpp – модуль, является частью сторонней библиотеки для работы с файлами формата XML;

## 2.4 Входные и выходные параметры

Входными данными для приложения являются следующие данные:

- нажатие клавиши управления змейкой;
- данные о координатах объектов карты (края карты, а также исходные части змейки).

Выходными данными для приложения будут выступать следующие данные:

- изменение координат всех частей змейки;
- начало новой игры или ее конец;
- удаление игровых очков с карты при их поглощении;
- отрисовка измененной карты и объектов на экране игрока.

## **2.5 Состав и параметры технических и программных средств**

Игра в жанре «Змейка» 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	main	Служит отправной точкой выполнения программы. Создает окно игры. Производит инициализацию карты. Использует алгоритмы: initGame(generator); checkCollisionWithMap(snake, obj); SnakePart::init(); SnakePart::getRect(); generateApple(snake, generator); Snake::control(); SnakeHead::update(); SnakeBody::update(); SnakeTail::update()		Функция. Возвращаемый параметр – число вызывающего его процесса (целочисленный тип)
2	initGame(generator)	Служит для инициализации глобальных переменных и перезапуска игры. Использует алгоритмы: generateApple(snake, generator)	generator – генератор псевдослучайных чисел .	Процедура
3	generateApple(snake, generator)	Служит для генерации нового объекта игровых очков.	snake – объект змейки; generator – генератор псевдослучайных чисел	Функция. Возвращаемый параметр – указатель на созданный



Продолжение таблицы 1 – Описание алгоритмов решения задачи

				объект игровых очков (тип Apple*)
4	checkCollisionWithMap (snake, obj)	Осуществляет проверку пересечения змейки с картой. Использует алгоритмы: SnakePart::getRect()	snake – объект змейки; obj – вектор объектов каты	Функция. Возвращаемый параметр – логическое значение
5	SnakePart::getRect()	Возвращает представление части змейки как прямоугольник		Функция. Возвращаемый параметр – объект прямоугольника (тип FloatRect)
6	SnakePart::init()	Инициализирует общие для всех частей змейки переменные		Процедура
7	Snake::control()	Осуществляет обновление и контроль движения змейки как единого целого		Процедура
8	SnakeHead::update()	Осуществляет обновление координат и позиции спрайта головы змейки		Процедура
9	SnakeBody::update()	Осуществляет обновление координат и позиции спрайта тела змейки. Обновляет направление движения, скорость и спрайт при повороте		Процедура
10	SnakeTail::update()	Осуществляет обновление координат и позиции спрайта тела змейки. Обновляет направление движения,		Процедура

Продолжение таблицы 1 – Описание алгоритмов решения задачи

		скорость и спрайт при повороте		
11	Snake::checkSnakeSelfCollision()	Осуществляет проверку пересечения головы змейки с телом. Использует алгоритмы: SnakePart::getRect()		Функция. Возвращаемый параметр – логическое значение

## 3.2 Структура данных

### 3.2.1 Структура типов программы

Таблица 2 – Структура типов программы

Элементы данных	Рекомендуемый тип	Назначение
SnakePart	<pre>class SnakePart { public:     float x, y, w, h, dx, dy, speed;     int dir;     static String File;     static Image image;     static Texture texture;     Sprite sprite;      static void init(); public:     virtual void update() = 0;     FloatRect getRect(); }</pre>	<p>Класс родитель для всех частей змейки.</p> <p>Поля данных:</p> <p>x, y – координаты объекта;</p> <p>w, h – ширина и высота объекта;</p> <p>dx, dy – векторы движения объекта;</p> <p>speed – скорость объекта;</p> <p>dir – направление движения объекта;</p> <p>File – файл загрузки тайлов змейки;</p> <p>Image – картинка тайлов змейки;</p> <p>Texture – текстура частей змейки;</p> <p>sprite – спрайт объекта</p> <p>Методы класса:</p> <p>update() – обновление части змейки;</p> <p>getRect() – предоставляет доступ к объекту как к прямоугольнику</p>
SnakeHead	<pre>class SnakeHead: public SnakePart { public:     void update(); }</pre>	<p>Класс головы змейки.</p> <p>Методы класса:</p> <p>update() – осуществляет обновление координат и позиции спрайта головы змейки</p>
SnakeBody	<pre>class SnakeBody: public SnakePart {</pre>	<p>Класс тела змейки.</p> <p>Поля данных:</p>

Продолжение таблицы 2 – Структура типов программы

	<pre> public:     vector&lt;turn&gt; turns; public:      void update();         </pre>	<p>turns – вектор необработанных поворотов части тела</p> <p>Методы класса: update() – осуществляет обновление координат и позиции спрайта тела змейки. Обновляет направление движения, скорость и спрайт при повороте</p>
SnakeTail	<pre> class SnakeTail: public SnakePart { public:     vector&lt;turn&gt; turns; public:      void update();         </pre>	<p>Класс хвоста змейки. Поля данных: turns – вектор необработанных поворотов части тела</p> <p>Методы класса: update() – осуществляет обновление координат и позиции спрайта хвоста змейки. Обновляет направление движения, скорость и спрайт при повороте</p>
Snake	<pre> class Snake { public:     SnakeHead snakeHead;     vector&lt;SnakeBody&gt; snakeBody;     SnakeTail snakeTail;     int score; public:     void control();     bool checkSnakeSelfCollision();         </pre>	<p>Класс целой змейки. Поля данных: snakeHead – голова змейки; snakeBody – вектор частей тела змейки; snakeTail – хвост змейки; score – количество собранных игровых очков</p> <p>Методы класса:</p>

Продолжение таблицы 2 – Структура типов программы

		control() – осуществляет обновление и контроль движения змейки как единого целого; checkSnakeSelfCollision() – осуществляет проверку пересечения головы змейки с телом
turn	<pre> struct turn {     float x, y, speed;     int fromDir, dir;     bool applied;     bool beRemoved; }         </pre>	<p>Структура данных о необработанном повороте.</p> <p>Поля данных:</p> <p>x, y – координаты точки поворота;</p> <p>speed – новая скорость после поворота;</p> <p>fromDir, dir – исходное и новое направление движение части змейки;</p> <p>applied – применен ли поворот к данной части змейки;</p> <p>beRemoved – необходимость удаление данного поворота</p>

### 3.2.2 Структура данных программы

Таблица 3 – Структура данных программы

Элементы данных	Рекомендуемый тип	Назначение	Комментарии
gameRestart	bool	Логическая переменная для перезапуска игры	Становится логической истиной, когда требуется перезапустить игру
lev	class Level	Переменная, которая хранит карту	Инициализируется при загрузке карты из файла
score	class Text	Переменная, которая хранит текст счета	Используется при выводе количества очков на экран
obj	vector<Object>	Переменная для хранения твердых объектов карты	Инициализируется загрузкой solid объектов из карты
snakeHead, snakeInitBody, snakeTail	class Object	Переменные для хранения начальных частей змейки	Инициализируются при загрузке из файла карты
head, body, tail	classes SnakeHead, SnakeBody, SnakeTail	Переменные, хранящие итоговые инициализирующие части змейки	Создаются и инициализируются данными из snakeHead, snakeInitBody, snakeTail
snake	class Snake	Переменная для хранения объекта целой змейки	Создается из head, body, tail
apple	Apple*	Переменная указатель на текущий бонус на игровом поле	Инициализируется функцией генерации игровых очков
appleClock, gameTimer	class Clock	Таймеры анимации яблока и самой игры	Необходимы для анимации спрайта игровых очков и движения змейки
first	bool	Переменная – индикатор первого цикла игры	Необходима для проверки коллизий, начиная со второго цикла

### **3.3 Схема алгоритмов решения задач по ГОСТ 19.701-90**

#### **3.3.1 Схема алгоритма initGame**

Схема алгоритма управления перезапуском игры.

В первую очередь осуществляется запуск игры, а именно вызов метода, который инициализирует основные глобальные переменные.

В дальнейшем этот метод может вызываться при принудительном перезапуске игры.

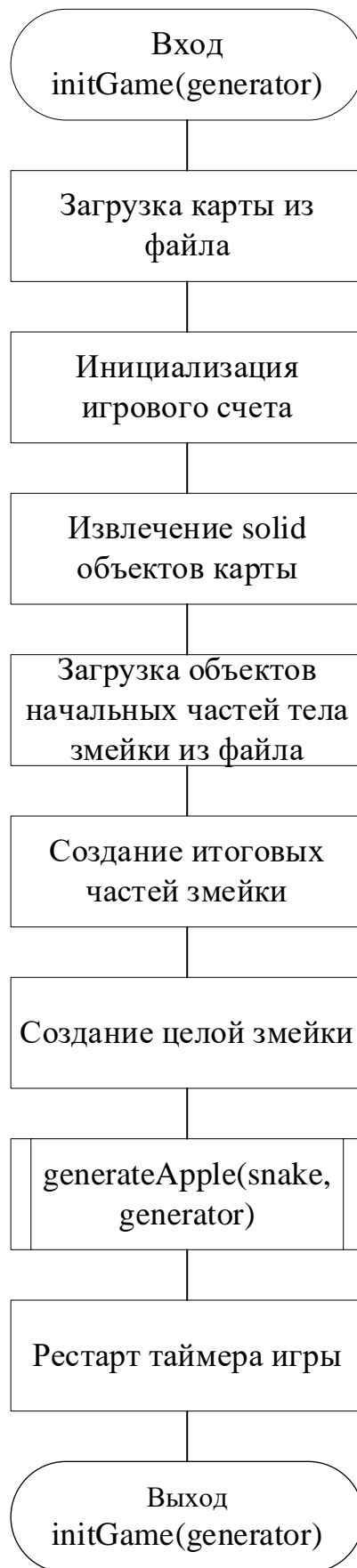


Рисунок 3.1 – Схема алгоритма initGame



### 3.3.2 Схема алгоритма generateApple

Схема алгоритма генерации игровых очков.

Прежде всего происходит инициализация всех глобальных полей при первом запуске игры.

В этот момент в первый раз вызывается функция генерации игровых очков, которая генерирует один объект и возвращает указатель на него. Игровые очки генерируются на случайном квадрате игрового поля, кроме блоков, на которых находится змейки или края карты.

Эта функция также будет вызываться при сборе очков, поскольку требуется генерация новых очков на новом месте.

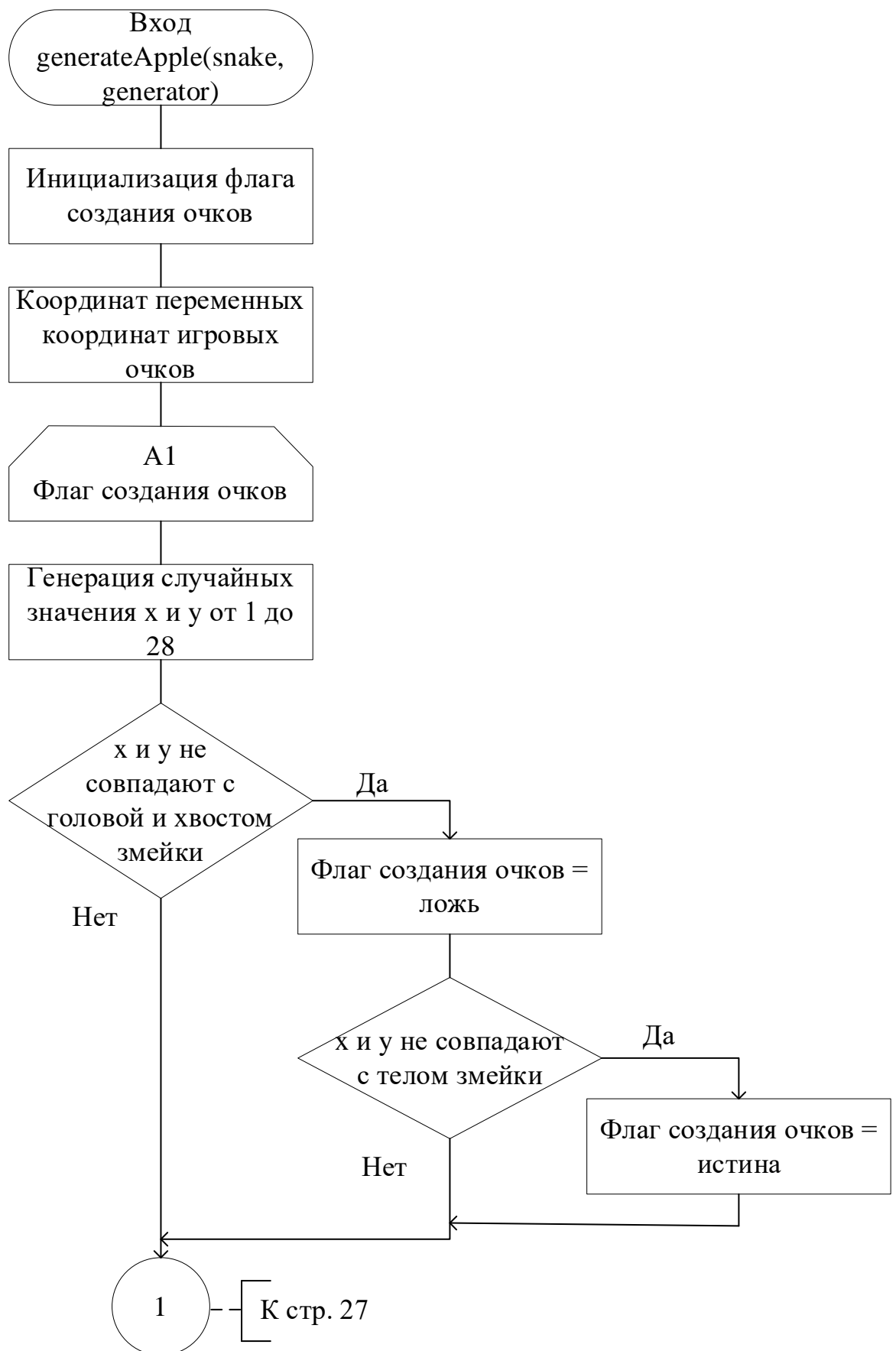


Рисунок 3.2 – Схема алгоритма generateApple (часть 1)

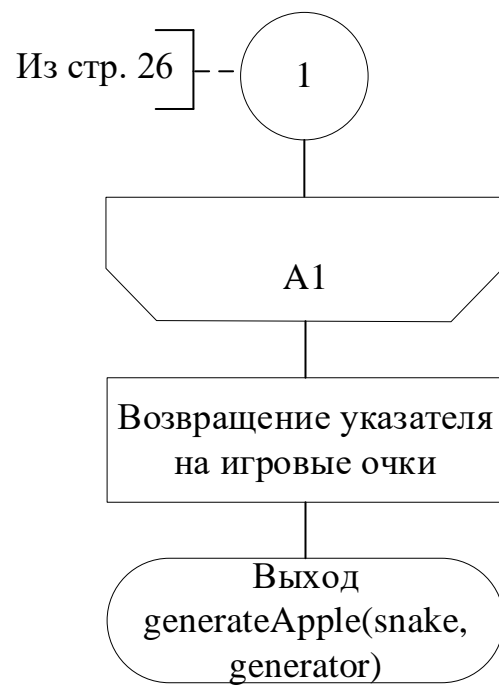


Рисунок 3.2 – Схема алгоритма generateApple (часть 2)

### 3.3.3 Схема алгоритма checkCollisionWithMap

Схема алгоритма проверки столкновения змейки с краями карты (объектами solid).

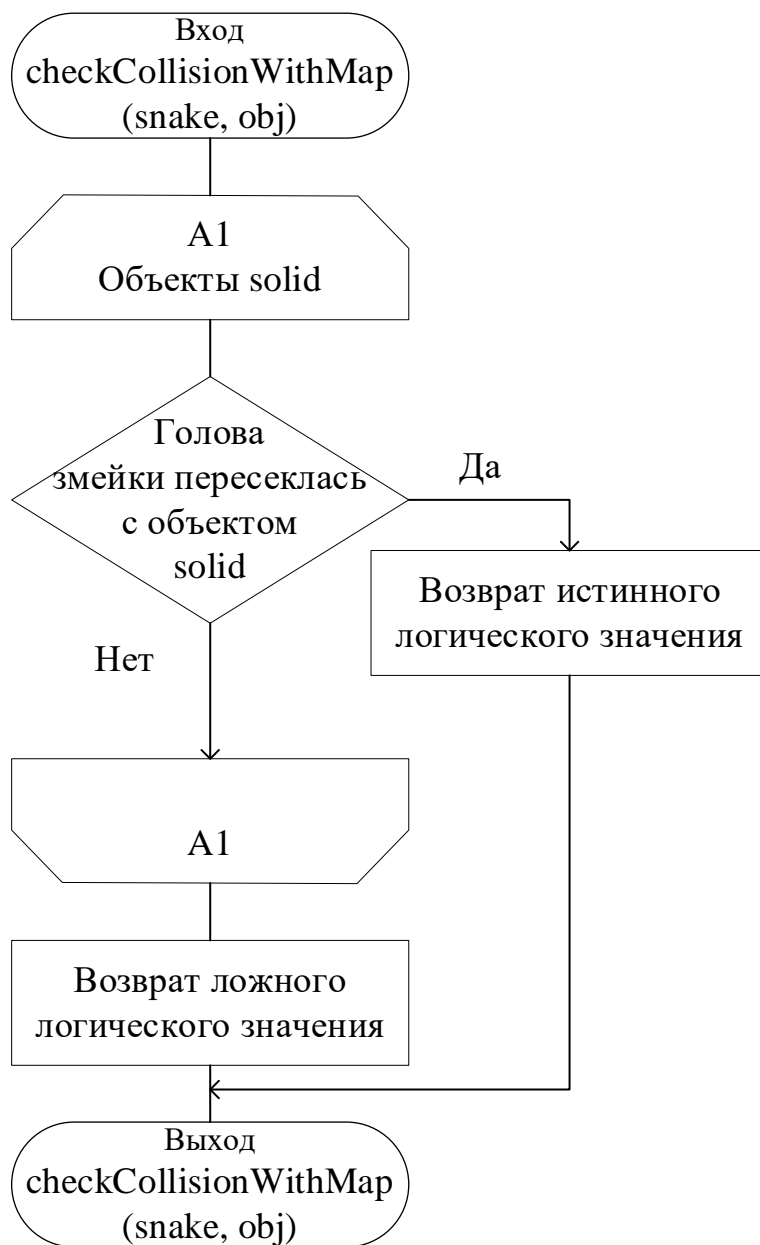


Рисунок 3.3 – Схема алгоритма checkCollisionWithMap

### 3.3.4 Схема алгоритма SnakePart::getRect

Схема алгоритма возврата представление объекта в виде прямоугольника, является основой для формирования взаимодействия между объектами. Возвращаемые параметры имеют особый тип FloatRect.

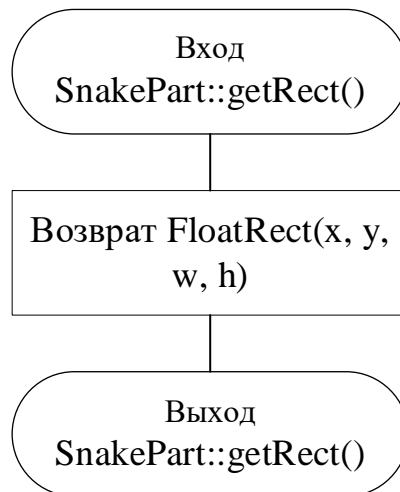


Рисунок 3.4 – Схема алгоритма SnakePart::getRect

### 3.3.5 Схема алгоритма SnakePart::init

Схема алгоритма загрузки и инициализации картинки и текстуры для всех спрайтов игры.



Рисунок 3.5 – Схема алгоритма SnakePart::init

### 3.3.6 Схема алгоритма Snake::control

Схема алгоритма контроля передвижения змейки. Изменяет направление движения головы змейки, а также мгновенно обновляет спрайт головы змейки. Для других частей змейки (частей тела и хвоста) создаются структуры, представляющие собой необработанные повороты змейки.

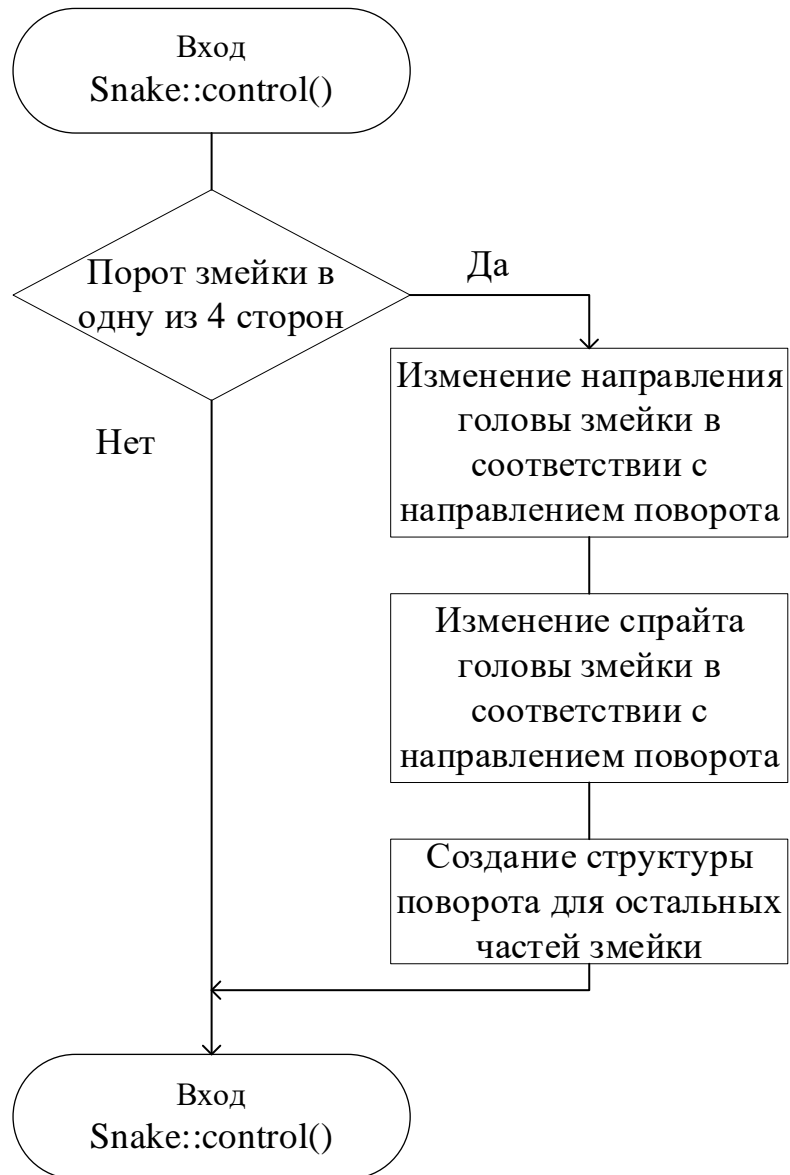


Рисунок 3.6 – Схема алгоритма Snake::control

### 3.3.7 Схема алгоритма SnakeHead::update

Схема алгоритма обновления головы змейки отражает процесс ее обновления на карте.

Обновление головы змейки сводится к обновлению координат в зависимости от вектора движения и обновлению позиции спрайта.

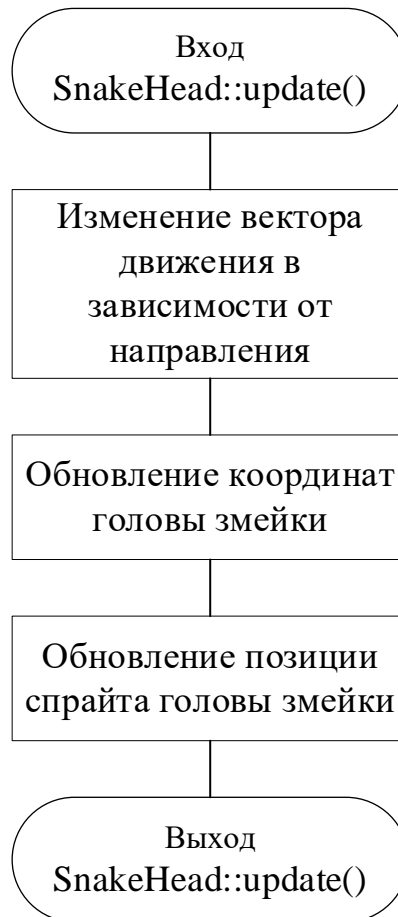


Рисунок 3.7 – Схема алгоритма SnakeHead::update



### 3.3.8 Схема алгоритма Character::update

Схема алгоритма обновления хвоста змейки отражает процесс ее обновления на карте.

Обновление хвоста змейки сводится к обновлению координат в зависимости от вектора движения и обновлению позиции спрайта.

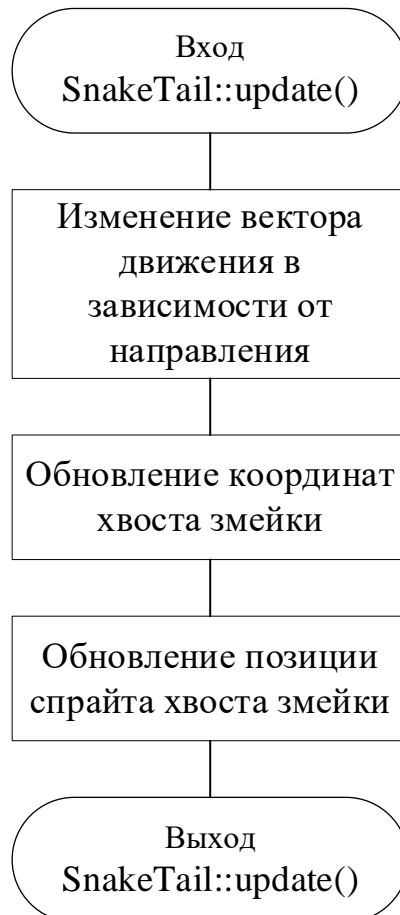


Рисунок 3.8 – Схема алгоритма SnakeTail::update

### 3.3.9 Схема алгоритма Snake::checkSnakeSelfCollision

Схема алгоритма проверки пересечения змейки с самой собой. По сути, это пересечение головы и любой частью тела или хвостом.

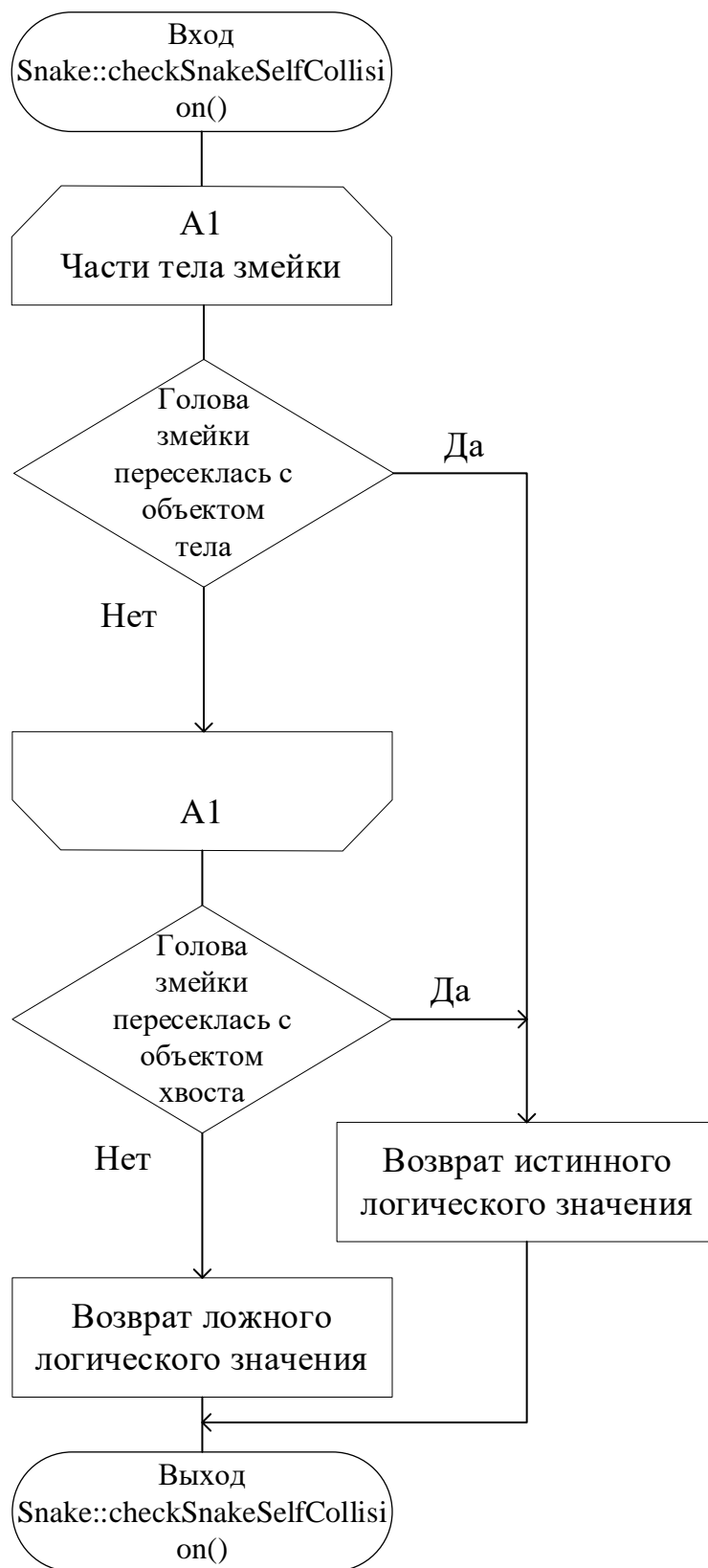


Рисунок 3.9 – Схема алгоритма Snake::checkSnakeSelfCollision

## 4 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

### 4.1 Запуск игры/перезапуск игры

#### 4.1.1 Тест 1

Таблица 4 – Тест 1

Тестовая ситуация:	Проверка корректности запуска игры при открытии программы
Исходный набор данных:	Запуск игры
Ожидаемый результат:	Корректная загрузка карты игры
Полученный результат:	

#### 4.1.2 Тест 2

Таблица 5 – Тест 2

Тестовая ситуация:	Проверка корректности перезапуска игры при нажатии на соответствующую кнопку
Исходный набор данных:	Перезапуск игры
Ожидаемый результат:	Корректная загрузка карты игры


Продолжение таблицы 5 – Тест 2

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




4.2 Управление змейкой

4.2.1 Тест 5

Таблица 6 – Тест 3

Тестовая ситуация:	Проверка поворота змейки в различные направления только при наличии исходных частей тела
Исходный набор данных:	Нажатие на кнопку клавиши W, A, S, D
Ожидаемый результат:	Передвижение модели змейки: W, A, S, D
Полученный результат:	<div>Поворот влево:</div>  <div>Поворот вправо:</div>

Продолжение таблицы 6 – Тест 3




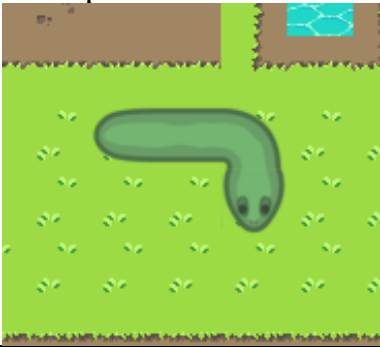
	 <p>Поворот направо:</p>  <p>Поворот вниз:</p> 
--	---

4.2.2 Тест 4

Таблица 7 – Тест 4

Тестовая ситуация:	Проверка поворота змейки в различные направления при наличии дополнительных частей тела
Исходный набор данных:	Нажатие на кнопку клавиши W, A, S, D
Ожидаемый результат:	Передвижение модели змейки: W, A, S, D
Полученный результат:	Поворот влево:

Продолжение таблицы 7 – Тест 4

	 <p>Поворот вправо:</p>  <p>Поворот вверх:</p>  <p>Поворот вниз:</p> 
--	--

4.3 Взаимодействие змейки и объектов карты

4.3.1 Тест 5

Таблица 8 – Тест 5

Тестовая ситуация:	Проверка взаимодействия края карты и змейки
Исходный набор данных:	Змейка близко подходит к краю карты

Продолжение таблицы 8 – Тест 5

Ожидаемый результат:	Проигрыш
Полученный результат:	Проигрыш и закрытие программы

4.3.2 Тест 6

Таблица 9 – Тест 6

Тестовая ситуация:	Проверка получение игровых очков, при столкновении головы змейки с яблоком
Исходный набор данных:	Голова змейки пересекается с яблоком
Ожидаемый результат:	Увеличение количества очков
Полученный результат:	



## **5 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА**

### **5.1 Задача игры и прохождение уровней**

Задачей игры, как и в классической «Змейке», является собрать как можно больше игровых очков, яблок. На пути змейки будут встречаться игровые очки, которые при подборе увеличат длину змейки, а также начислят очки к игровому счету. Змейка может передвигаться только в пределах игрового поля, а при столкновении с краем карты игра закончится. Целью игры является сбор максимального количества очков, когда змейка займет все игровое поле.

### **5.2 Управление персонажем**

Управление змейкой в игре происходит посредством взаимодействия игрока с клавиатурой. В игре предоставлены возможности передвижения в декартовой системе координат, в четырех плоскостях.

Клавиши управления и их комбинации:

- A, D – поворот змейки влево и вправо соответственно;
- W, S – поворот змейки вверх и вниз соответственно.

### **5.3 Сбор игровых очков**

Сбор игровых очков происходит автоматически при пересечении змейки с яблоком.

### **5.4 Дополнительные опции**

В игре доступно:

- Restart game – кнопка меню, которая перезапустит игру.

## ЗАКЛЮЧЕНИЕ

В конечном счете, разработанная игра в жанре «Змейка» 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, а также познакомиться с инструментом создания тайловых карт Tiled. В то же время потребовалось изучить возможности библиотек SFML и tinyXML.

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] Серебряная Л.В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л.В. Серебряная, И.М. Марина. – Минск: БГУИР, 2013. – 5 с.
- [2] Библиотека SFML[Электронный ресурс]. – Режим доступа: <https://www.sfm1-dev.org/> – Дата обращения: 04.10.2023.
- [3] Библиотека TinyXML[Электронный ресурс]. – Режим доступа: <https://sourceforge.net/projects/tinyxml/> – Дата обращения: 06.10.2023.
- [4] Обзор Tiled[Электронный ресурс]. – Режим доступа: [https://ps-group.github.io/cxx/sfm1\\_tiled](https://ps-group.github.io/cxx/sfm1_tiled) – Дата обращения: 06.10.2023.
- [5] MSDN Documentation[Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/> - Дата обращения 30.10.2023.

**ПРИЛОЖЕНИЕ А**  
**(обязательное)**  
**Исходный код программы**  
**Модуль main**

```
#include "main.h"
using namespace sf;

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wp,
LPARAM lp);
HWND initWindow(HINSTANCE instance);
bool checkCollisionWithMap(Snake& snake, const
vector<Object>& obj);

bool gameRestart = true;
Level lev;
Font font;
Text score;
vector<Object> obj;
Object snakeHead;
Object snakeInitBody;
Object snakeTail;
SnakeHead head;
SnakeBody body;
SnakeTail tail;
Snake snake;
Apple* apple;
Clock appleClock;
Clock gameTimer;
bool first;
MSG msg;

struct PRNG
{
    std::mt19937 engine;
};
void initGame(PRNG& generator);
Apple* generateApple(const Snake& snake, PRNG& generator);

void initGenerator(PRNG& generator)
{
    std::random_device device;
    generator.engine.seed(device());
}

unsigned random(PRNG& generator, unsigned minValue,
unsigned maxValue)
{
```

```

        assert(minValue < maxValue);
        std::uniform_int_distribution<unsigned>
distribution(minValue, maxValue);
        return distribution(generator.engine);
    }

void menu(HWND hwnd) {
    HMENU hMenu = CreateMenu();
    AppendMenu(hMenu, MF_POPUP, 1, L"Restart game");
    SetMenu(hwnd, hMenu);
}

int main()
{
    SnakePart::init();
    HINSTANCE instance = GetModuleHandle(NULL);

    HWND hwnd = initWindow(instance);
    RenderWindow window(hwnd);
    window.setPosition(sf::Vector2i(100, 0));
    PRNG generator;
    initGenerator(generator);
    PlaySound(L"sounds/game.wav", NULL, SND_FILENAME |
SND_LOOP | SND_ASYNC);
    while (msg.message != WM_QUIT)
    {
        if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        else
        {
            if (gameRestart) {
                initGame(generator);
                gameRestart = false;
            }
            if (gameTimer.getElapsedTime().asSeconds() >=
0.15f) {
                gameTimer.restart();

                float time =
appleClock.getElapsedTime().asMicroseconds();
                appleClock.restart();
                time = time / 1000;

```

```

        if ((first &&
checkCollisionWithMap(snake, obj)) ||
snake.checkSnakeSelfCollision() || snake.score == 781) {
            return msg.message = WM_QUIT;
        }

        if
(snake.snakeHead.getRect().intersects((*apple).getRect()))
{
            snake.score++;
            score.setString("Score: " +
to_string(snake.score));
            delete apple;
            apple = generateApple(snake,
generator);

            // creation of new body
            SnakeBody newBody =
SnakeBody(snake.snakeTail.x, snake.snakeTail.y,
snake.snakeTail.w, snake.snakeTail.h);
            (newBody).dir = snake.snakeTail.dir;
            (newBody).turns =
vector<turn>(snake.snakeTail.turns);
            if (newBody.dir <= 1) {

                newBody.sprite.setTextureRect(IntRect(1 * newBody.w, 0
* newBody.h, newBody.w, newBody.h));
            }
            else
            {

                newBody.sprite.setTextureRect(IntRect(2 * newBody.w, 1
* newBody.h, newBody.w, newBody.h));
            }
            snake.snakeBody.push_back(newBody);
            switch (snake.snakeTail.dir)
            {
            case 0: {
                snake.snakeTail.x -= 32;
                break;
            }
            case 1: {
                snake.snakeTail.x += 32;
                break;
            }
            case 2: {
                snake.snakeTail.y -= 32;
                break;
            }
            }

```

```

        case 3: {
            snake.snakeTail.y += 32;
            break;
        }
        default:
            break;
    }
}

snake.control();
snake.snakeHead.update();
(*apple).update(time);
for (auto it = snake.snakeBody.begin();
it != snake.snakeBody.end(); it++) {
    it->update();
}
snake.snakeTail.update();

window.clear();
lev.Draw(window);
score.setPosition(40,40);
window.draw(score);
window.draw((*apple).sprite);
for (auto it = snake.snakeBody.begin();
it != snake.snakeBody.end(); it++) {
    window.draw(it->sprite);
}
window.draw(snake.snakeTail.sprite);
window.draw(snake.snakeHead.sprite);
window.display();

if (!first) {
    first = true;
}
}
}
return EXIT_SUCCESS;
}

```

```

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wp,
LPARAM lp)
{
    switch (msg)
    {
    case WM_CREATE:
    {
        menu(hwnd);
    }
    }
}

```

```

        break;
    }
    case WM_COMMAND:
    {
        switch (LOWORD(wp))
        {
            case 1:
            {
                gameRestart = true;
                break;
            }
            default:
                break;
        }
        break;
    }

    // Quit when we close the main window
    case WM_CLOSE:
    {
        PostQuitMessage(0);
        return 0;
    }
}

return DefWindowProc(hwnd, msg, wp, lp);
}

HWND initWindow(HINSTANCE instance)
{
    WNDCLASSEX wc;
    memset(&wc, 0, sizeof(wc));
    wc.cbSize = sizeof(wc);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wc.hInstance = instance;
    wc.lpfnWndProc = WndProc;
    wc.lpszClassName = L"Snake";
    wc.style = CS_HREDRAW | CS_VREDRAW;
    RegisterClassEx(&wc);
    return CreateWindowEx(WS_EX_WINDOWEDGE, L"Snake",
L"Snake", WS_VISIBLE | WS_SYSMENU, CW_USEDEFAULT,
CW_USEDEFAULT, 978, 1030, NULL, NULL, instance, NULL);
}

bool checkCollisionWithMap(Snake& snake, const
vector<Object>& obj)
{
    for (int i = 0; i < obj.size(); i++) {

```



```

        if
(snake.snakeHead.getRect().intersects(obj[i].rect))
        {
            if (obj[i].name == "solid")
            {
                return true;
            }
        }
    }
    return false;
}

Apple* generateApple(const Snake& snake, PRNG& generator) {
    bool flag = true;
    int x = 0;
    int y = 0;
    while (flag) {
        x = random(generator, 1, 28);
        y = random(generator, 1, 28);
        if ((snake.snakeHead.x != x * snake.snakeHead.w)
&& (snake.snakeHead.y != y * snake.snakeHead.h)
&& (snake.snakeTail.x != x *
snake.snakeTail.w) && (snake.snakeTail.y != y *
snake.snakeTail.h)) {
            flag = false;
            for (auto it = snake.snakeBody.begin(); it !=
snake.snakeBody.end(); it++) {
                if ((it->x == (x * it->w)) && (it->y ==
(y * it->h))) {
                    flag = true;
                }
            }
        }
    }
    return (new Apple(x * 32, y * 32, 32, 32));
}

void initGame(PRNG& generator) {
    lev.LoadFromFile("map.tmx");
    font.loadFromFile("fonts/Inkulinati-Regular.otf");
    score = Text("Score: 0", font, 20);
    score.setFill(Color::Red);
    score.setStyle(Text::Bold);
    obj = lev.GetObjects("solid");
    snakeHead = lev.GetObject("snakeHead");
    snakeInitBody = lev.GetObject("snakeInitBody");
    snakeTail = lev.GetObject("snakeTail");
}

```

```
        head = SnakeHead(snakeHead.rect.left,
snakeHead.rect.top, snakeHead.rect.width,
snakeHead.rect.height);
        body = SnakeBody(snakeInitBody.rect.left,
snakeInitBody.rect.top, snakeInitBody.rect.width,
snakeInitBody.rect.height);
        tail = SnakeTail(snakeTail.rect.left,
snakeTail.rect.top, snakeTail.rect.width,
snakeTail.rect.height);
        snake = Snake(head, body, tail);
        apple = generateApple(snake, generator);

        gameTimer.restart();

        first = false;
        msg.message = WM_NULL;
    }
```

**ПРИЛОЖЕНИЕ Б**  
**(обязательное)**  
**Исходный код программы**  
**Модуль Apple**

```
#include "Apple.h"

void Apple::update(float time) {
    currentFrame += 0.01 * time;
    if (currentFrame > 3) currentFrame -= 3;
    sprite.setTextureRect(IntRect(w * int(currentFrame), 3
* h, w, h));
}

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

**ПРИЛОЖЕНИЕ В**  
**(обязательное)**  
**Исходный код программы**  
**Модуль Snake**

```
#include "Snake.h"

void Snake::control() {
    if (GetAsyncKeyState(0x41) && (snakeHead.dir != 1) &&
        (snakeHead.dir != 0) && !GetAsyncKeyState(0x44) &&
        !GetAsyncKeyState(0x57) && !GetAsyncKeyState(0x53)) {
        int prevDir = snakeHead.dir;
        snakeHead.dir = 1;
        snakeHead.sprite.setTextureRect(IntRect(3 *
snakeHead.w, 1 * snakeHead.h, snakeHead.w, snakeHead.h));

        snakeTail.turns.push_back(turn{ snakeHead.x,
snakeHead.y, snakeHead.speed, prevDir, snakeHead.dir,
false, false });

        for (auto it = snakeBody.begin(); it !=
snakeBody.end(); it++) {
            it->turns.push_back(turn{ snakeHead.x,
snakeHead.y, snakeHead.speed, prevDir, snakeHead.dir,
false, false });
        }
        if (GetAsyncKeyState(0x44) && (snakeHead.dir != 0) &&
            (snakeHead.dir != 1) && !GetAsyncKeyState(0x41) &&
            !GetAsyncKeyState(0x57) && !GetAsyncKeyState(0x53)) {
            int prevDir = snakeHead.dir;
            snakeHead.dir = 0;
            snakeHead.sprite.setTextureRect(IntRect(4 *
snakeHead.w, 0 * snakeHead.h, snakeHead.w, snakeHead.h));

            snakeTail.turns.push_back(turn{ snakeHead.x,
snakeHead.y, snakeHead.speed, prevDir, snakeHead.dir,
false, false });

            for (auto it = snakeBody.begin(); it !=
snakeBody.end(); it++) {
                it->turns.push_back(turn{ snakeHead.x,
snakeHead.y, snakeHead.speed, prevDir, snakeHead.dir,
false, false });
            }
        }
    }
```

```

        if (GetAsyncKeyState(0x57) && (snakeHead.dir != 3) &&
(snakeHead.dir != 2) && !GetAsyncKeyState(0x41) &&
!GetAsyncKeyState(0x44) && !GetAsyncKeyState(0x53)) {
            int prevDir = snakeHead.dir;
            snakeHead.dir = 3;
            snakeHead.sprite.setTextureRect(IntRect(3 *
snakeHead.w, 0 * snakeHead.h, snakeHead.w, snakeHead.h));

            snakeTail.turns.push_back(turn{ snakeHead.x,
snakeHead.y, snakeHead.speed, prevDir, snakeHead.dir,
false, false });

            for (auto it = snakeBody.begin(); it !=
snakeBody.end(); it++) {
                it->turns.push_back(turn{ snakeHead.x,
snakeHead.y, snakeHead.speed, prevDir, snakeHead.dir,
false, false });
            }
        }
        if (GetAsyncKeyState(0x53) && (snakeHead.dir != 2) &&
(snakeHead.dir != 3) && !GetAsyncKeyState(0x41) &&
!GetAsyncKeyState(0x44) && !GetAsyncKeyState(0x57)) {
            int prevDir = snakeHead.dir;
            snakeHead.dir = 2;
            snakeHead.sprite.setTextureRect(IntRect(4 *
snakeHead.w, 1 * snakeHead.h, snakeHead.w, snakeHead.h));

            snakeTail.turns.push_back(turn{ snakeHead.x,
snakeHead.y, snakeHead.speed, prevDir, snakeHead.dir,
false, false });

            for (auto it = snakeBody.begin(); it !=
snakeBody.end(); it++) {
                it->turns.push_back(turn{ snakeHead.x,
snakeHead.y, snakeHead.speed, prevDir, snakeHead.dir,
false, false });
            }
        }
    }

bool Snake::checkSnakeSelfCollision() {
    for (auto it = snakeBody.begin(); it !=
snakeBody.end(); it++) {
        if (it->getRect().intersects(snakeHead.getRect()))
        {
            return true;
        }
    }
}

```

```
    }  
    if  
(snakeHead.getRect().intersects(snakeTail.getRect())) {  
        return true;  
    }  
    return false;  
}
```

**ПРИЛОЖЕНИЕ Г**  
**(обязательное)**  
**Исходный код программы**  
**Модуль SnakeBody**

```
#include "SnakeBody.h"

bool SnakeBody::removeTurnsPredicate(turn currentTurn) {
    return currentTurn.beRemoved == true;
}

void SnakeBody::update() {
    turns.erase(remove_if(turns.begin(), turns.end(),
&removeTurnsPredicate), turns.end());
    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;
    y += dy;

    sprite.setPosition(x, y);

    for (auto it = turns.begin(); it != turns.end(); it++)
    {
        if ((it->x == x) && (it->y == y) && (it->applied
== false)) {
            if ((it->fromDir == 1) && (it->dir == 3)) {
                sprite.setTextureRect(IntRect(0 * w, 1 *
h, w, h));
            }
            if ((it->fromDir == 0) && (it->dir == 3)) {
                sprite.setTextureRect(IntRect(2 * w, 2 *
h, w, h));
            }
            if ((it->fromDir == 1) && (it->dir == 2)) {
                sprite.setTextureRect(IntRect(0 * w, 0 *
h, w, h));
            }
            if ((it->fromDir == 0) && (it->dir == 2)) {
                sprite.setTextureRect(IntRect(2 * w, 0 *
h, w, h));
            }
        }
    }
}
```

```

        if ((it->fromDir == 3) && (it->dir == 0)) {
            sprite.setTextureRect(IntRect(0 * w, 0 *
h, w, h));
        }
        if ((it->fromDir == 3) && (it->dir == 1)) {
            sprite.setTextureRect(IntRect(2 * w, 0 *
h, w, h));
        }
        if ((it->fromDir == 2) && (it->dir == 0)) {
            sprite.setTextureRect(IntRect(0 * w, 1 *
h, w, h));
        }
        if ((it->fromDir == 2) && (it->dir == 1)) {
            sprite.setTextureRect(IntRect(2 * w, 2 *
h, w, h));
        }
        dir = it->dir;
        speed = it->speed;
        it->applied = true;
    }
    else {
        if (it->applied == true) {
            if ((dir == 0) || (dir == 1)) {
                sprite.setTextureRect(IntRect(1 * w,
0 * h, w, h));
            }
            else
            {
                sprite.setTextureRect(IntRect(2 * w,
1 * h, w, h));
            }
            it->beRemoved = true;
        }
    }
}
}
}

```



**ПРИЛОЖЕНИЕ Д**  
**(обязательное)**  
**Исходный код программы**  
**Модуль SnakeHead**

```
#include "SnakeHead.h"

void SnakeHead::update() {
    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;
    y += dy;

    sprite.setPosition(x, y);
}
```

**ПРИЛОЖЕНИЕ Е**  
**(обязательное)**  
**Исходный код программы**  
**Модуль SnakeTail**

```
#include "SnakeTail.h"

bool SnakeTail::removeTurnsPredicate(turn currentTurn) {
    return currentTurn.beRemoved == true;
}

void SnakeTail::update() {
    turns.erase(remove_if(turns.begin(), turns.end(),
&removeTurnsPredicate), turns.end());
    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;
    y += dy;

    sprite.setPosition(x, y);

    for (auto it = turns.begin(); it != turns.end(); it++)
    {
        if ((it->x == x) && (it->y == y) && (it->applied
== false)) {
            if (it->dir == 0) {
                sprite.setTextureRect(IntRect(4 * w, 2 *
h, w, h));
            }
            if (it->dir == 1) {
                sprite.setTextureRect(IntRect(3 * w, 3 *
h, w, h));
            }
            if (it->dir == 2) {
                sprite.setTextureRect(IntRect(4 * w, 3 *
h, w, h));
            }
            if (it->dir == 3) {
                sprite.setTextureRect(IntRect(3 * w, 2 *
h, w, h));
            }
            dir = it->dir;
            speed = it->speed;
        }
    }
}
```

```
        it->applied = true;
        it->beRemoved = true;
    }
}
```