|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**КУРСОВАЯ РАБОТА**

по дисциплине: «Объектно-ориентированное программирование»

по профилю: Информатизация организаций

направления профессиональной подготовки: Прикладная информатика, бакалавриат

Тема: «Разработка компьютерной игры “Змейка”»

Студент (ф.и.о. полностью): Автушко Даниил Александрович

Группа: ИНБО-04-18

Работа представлена к защите\_\_\_\_\_\_\_\_\_\_\_(дата)\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись и ф.и.о. студента)

Руководитель: Хлебникова Валерия Леонидовна старший преподаватель

Работа допущена к защите\_\_\_\_\_\_\_\_\_\_\_(дата)\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись ф.и.о. руководителя)

Оценка по итогам защиты: \_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. МИРЭА. 2019г.

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 1](#_Toc8742420)

[1 Введение 4](#_Toc8742421)

[2 Техническое задание 5](#_Toc8742422)

[2.1 Общие положения 5](#_Toc8742423)

[2.1.1 Полное наименование системы и ее условное обозначение 5](#_Toc8742424)

[2.1.2 Номер договора (контракта) 5](#_Toc8742425)

[2.1.3 Наименования организации-заказчика и организаций-участников работ 5](#_Toc8742426)

[2.1.4 Перечень документов, на основании которых создается система 5](#_Toc8742427)

[2.1.5 Плановые сроки начала и окончания работы по созданию системы 5](#_Toc8742428)

[2.1.6 Источники и порядок финансирования работ 5](#_Toc8742429)

[2.1.7 Порядок оформления и предъявления заказчику результатов работ по созданию системы 5](#_Toc8742430)

[2.1.8 Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ 6](#_Toc8742431)

[2.1.9 Определения, обозначения и сокращения 6](#_Toc8742432)

[2.2 НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ 6](#_Toc8742433)

[2.2.1 Назначение системы 6](#_Toc8742434)

[2.2.2 Цели создания системы 6](#_Toc8742435)

[2.3 ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ 6](#_Toc8742436)

[2.4 ТРЕБОВАНИЯ К СИСТЕМЕ 7](#_Toc8742437)

[2.4.1 Требования к системе в целом 7](#_Toc8742438)

[2.4.1.1 Требования к структуре и функционированию системы 7](#_Toc8742439)

[2.4.1.1.1 Перечень подсистем, их назначение и основные характеристики 7](#_Toc8742440)

[2.4.1.1.2 Требования к способам и средствам связи для информационного обмена между компонентами системы 7](#_Toc8742441)

[2.4.1.2 Требования к численности и квалификации персонала системы 7](#_Toc8742442)

[2.4.1.3 Показатели назначения 8](#_Toc8742443)

[2.4.1.4 Требования к надежности 8](#_Toc8742444)

[2.4.1.5 Требования к безопасности 8](#_Toc8742445)

[2.4.1.6 Требования к эргономике и технической эстетике 8](#_Toc8742446)

[2.4.1.7 Требования к транспортабельности для подвижных АС 8](#_Toc8742447)

[2.4.1.8 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы 8](#_Toc8742448)

[2.4.1.9 Требования к защите информации от несанкционированного доступа 8](#_Toc8742449)

[2.4.1.10 Требования по сохранности информации при авариях 9](#_Toc8742450)

[2.4.1.11 Требования к защите от влияния внешних воздействий 9](#_Toc8742451)

[2.4.1.12 Требования к патентной частоте 9](#_Toc8742452)

[2.4.1.13 Требования по стандартизации и унификации 9](#_Toc8742453)

[2.4.1.14 Дополнительные требования 9](#_Toc8742454)

[2.4.2 Требования к функциям (задачам), выполняемым системой 9](#_Toc8742455)

[2.4.3 Требования к видам обеспечения 10](#_Toc8742456)

[2.4.3.1 Требования к математическому обеспечению системы 10](#_Toc8742457)

[2.4.3.2 Требования информационному обеспечению системы 10](#_Toc8742458)

[2.4.3.3 Требования к лингвистическому обеспечению системы 10](#_Toc8742459)

[2.4.3.4 Требования к программному обеспечению системы 10](#_Toc8742460)

[2.4.3.5 Требования к техническому обеспечению 10](#_Toc8742461)

[2.4.3.6 Требования к метрологическому обеспечению 10](#_Toc8742462)

[2.4.3.7 Требования к организационному обеспечению 11](#_Toc8742463)

[2.4.3.8 Требования к методическому обеспечению 11](#_Toc8742464)

[2.5 СОСТАВ И СОДЕРЖАНИЕ РАБОТ ПО СОЗДАНИЮ (РАЗВИТИЮ) СИСТЕМЫ 11](#_Toc8742465)

[2.6 ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ СИСТЕМЫ 12](#_Toc8742466)

[2.6.1 Виды, состав, объем и методы испытаний системы 12](#_Toc8742467)

[2.6.2 Общие требования к приемке работ по стадиям 12](#_Toc8742468)

[2.6.3 Статус приемочной комиссии 12](#_Toc8742469)

[2.7 ТРЕБОВАНИЯ К СОСТАВУ И СОДЕРЖАНИЮ РАБОТ ПО ПОДГОТОВКЕ ОБЪЕКТА АВТОМАТИЗАЦИИ К ВВОДУ СИСТЕМЫ В ДЕЙСТВИЕ 12](#_Toc8742470)

[2.8 ТРЕБОВАНИЯ К ДОКУМЕНТИРОВАНИЮ 13](#_Toc8742471)

[3 Проектирование и разработка программы 14](#_Toc8742472)

[3.1 Проектирование программы 14](#_Toc8742473)

[3.1.1 Функциональные требования к программе 14](#_Toc8742474)

[3.1.2 Проектирование классов 14](#_Toc8742475)

[3.1.3 Разработка программы 18](#_Toc8742476)

[3.1.3.1 Описание проектного решения 18](#_Toc8742477)

[3.1.3.2 Тестирование программы 19](#_Toc8742478)

[4 Руководство пользователя 20](#_Toc8742479)

[5 Заключение 24](#_Toc8742480)

[6 Список литературы 25](#_Toc8742481)

[7 Приложение №1. Исходный код программы 26](#_Toc8742482)

[8 Приложение №2. UML-диаграмма 65](#_Toc8742483)

# Введение

Программным продуктом, реализованным в данной курсовой работе, является «Компьютерная игра “Змейка”» - программа, которая представляет собой всем известную игру “Змейка”, суть которой заключается в наборе максимального количества очков путем поедания змейкой еды.

Языком программирования, выбранным для реализации, стал объектно-ориентированный язык программирования C++. C++ один из самых широко распространенных языков. Преимуществами данного языка являются его мощность и гибкость, обусловленные такими составляющими, как указатели, шаблоны и множественное наследование.

Графический пользовательский интерфейс программного продукта реализован с помощью SFML. SFML (Simple and Fast Multimedia Library) — это библиотека для представления мультимедийных данных. SFML обеспечивает простой интерфейс для разработки игр и прочих мультимедийных приложений. Состоит она из пяти модулей: system, window, graphics, audio и network.

Унифицированный язык моделирования, использованный в данной курсовой работе, представляет собой графический язык, включающий в себя множество различных диаграмм, помогающий специалистам по системному анализу создавать алгоритмы, а программистам в принципах работы программы. UML является мощным инструментом, позволяющим сделать процесс программирования более легким и эффективным.

# Техническое задание



## Общие положения

### Полное наименование системы и ее условное обозначение

Полное наименование: Компьютерная игра “Змейка”.

### Номер договора (контракта)

Работа выполняется на основании Задания на выполнение курсовой работы.

### Наименования организации-заказчика и организаций-участников работ

Заказчик: РТУ МИРЭА

Исполнитель: Автушко Даниил Александрович студент группы ИНБО-04-18

### Перечень документов, на основании которых создается система

Документы, на основании которых создается система:

* Учебный план (№1382.9 09.03.03 ПИ Очн, пс, 4 года (8 сессий)

УМУ\_09.03.03\_ИО\_ИИТ\_2018).

### Плановые сроки начала и окончания работы по созданию системы

Плановый срок начала работы по созданию системы: 26 февраля 2019 года.

Плановый срок окончания работы по созданию системы: 20 мая 2019 года.

### Источники и порядок финансирования работ

Разработка Системы финансируется РТУ МИРЭА, а, в частности,

Федеральным бюджетом Российской Федерации.

### Порядок оформления и предъявления заказчику результатов работ по созданию системы

Система передаётся в виде готового программного обеспечения на базе средств вычислительной техники Заказчика в сроки, установленные в п. 2.5 данного ТЗ. Приёмка осуществляется в составе Исполнителя и уполномоченных представителей Заказчика.

Порядок предъявления системы, её испытаний и окончательной приёмки

определён в п.7 настоящего ТЗ. Совместно с предъявлением Системы Исполнителем производится сдача разработанного комплекта документации согласно п. 9 данного ТЗ.

### Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ

При разработке системы Исполнитель должен руководствоваться

требованиями следующих документов:

* ГОСТ 34.601-90 Комплекс стандартов Автоматизированные системы.

Стадии создания;

* Методические указания по выполнению курсовой работы для

бакалавров.

## НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ

### Назначение системы

Система предназначена для удовлетворения досуга пользователей.

### Цели создания системы

Цели создания Системы:

* демонстрация прикладного программного обеспечения, работающего с графической библиотекой;
* сдача курсовой работы;
* приобретение опыта разработки приложений средней сложности;
* изучение объектно-ориентированного программирования;
* приобретение навыков написания технической документации.

## ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ

Объектом автоматизации данной системы является компьютерная игра “Змейка”,

## ТРЕБОВАНИЯ К СИСТЕМЕ

### Требования к системе в целом

#### Требования к структуре и функционированию системы

##### Перечень подсистем, их назначение и основные характеристики

Разрабатываемая система, должна включать в себя несколько классов для моделирования змейки:

* Snake – класс для хранения основных характеристик змейки, таких как скорость, текстура, направление.
* SnakePart – класс для хранения данных о частях змейки.
* Food – класс, отвечающий за еду для змейки.
* Menu – класс, основной задачей которого является отображение меню перед началом игры и после.

##### Требования к способам и средствам связи для информационного обмена между компонентами системы

Связь между компонентами системы обеспечивается функциями, определёнными в компонентах системы.

#### Требования к численности и квалификации персонала системы

Для работы разрабатываемой системы необходимо достаточно одного человека. К работе с системой должны допускаться сотрудники, имеющие навыки работы на персональном компьютере, ознакомленные с правилами эксплуатации и прошедшие обучение работе с системой.

#### Показатели назначения

Система должна моделировать игру “Змейка”. Система должны моделировать движение змейки, появление еды и поглощение этой еды змейкой. Требования ко времени выполнения данных функций не предъявляются.

#### Требования к надежности

Отсутствуют.

#### Требования к безопасности

Отсутсвуют.

#### Требования к эргономике и технической эстетике

Графический интерфейс Системы должен разрабатыватьcя на основе офтальмологических исследований, с использованием сочетаний цветов, комфортных для глаз человека.

#### Требования к транспортабельности для подвижных АС

Разрабатываемая Система должна запускаться на любых устройствах под управлением операционной системы не старше чем Windows XP без процедуры установки или повторной компиляции данной Системы.

#### Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы

Для корректной работы разрабатываемая Система должна храниться в отдельном каталоге со всеми своими компонентами. Особых требований по обслуживанию данная Система не имеет.

#### Требования к защите информации от несанкционированного доступа

Требования к какой-либо защите информации в данной системе не предъявляются.

#### Требования по сохранности информации при авариях

Отсутствуют.

#### Требования к защите от влияния внешних воздействий

Защита от влияния внешних воздействий должна обеспечиваться средствами программно технического комплекса Заказчика.

#### Требования к патентной частоте

Данная Система должна являться интеллектуальной собственностью Исполнителя и быть патентно чистой по отношению ко всем странам мира.

#### Требования по стандартизации и унификации

Система должна быть реализована методами объектно-ориентированного программирования. Взаимодействие пользователей с прикладным программным обеспечением, входящим в состав системы должно осуществляться посредством визуального графического интерфейса (GUI). Интерфейс системы должен быть понятным и удобным, не должен быть перегружен графическими элементами и должен обеспечивать быстрое отображение экранных форм. Навигационные элементы должны быть выполнены в удобной для пользователя форме. Интерфейс должен соответствовать современным эргономическим требованиям и обеспечивать удобный доступ к основным функциям и операциям системы.

#### Дополнительные требования

Дополнительные требования не предъявляются.

### Требования к функциям (задачам), выполняемым системой

Система должна выполнять следующие функции:

* Отображение змейки на экране;
* Моделирование движения змейки;
* Появление еды на экране;
* Поглощение еды змейкой;
* Увеличение размеров змейки;

### Требования к видам обеспечения

#### Требования к математическому обеспечению системы

Отсутствуют.

#### Требования информационному обеспечению системы

Отсутствуют.

#### Требования к лингвистическому обеспечению системы

Для создания Системы должен быть использован низкоуровневый язык программирования C++. Для взаимодействия Системы с пользователем, руководства по использованию Системы и документации Системы должен использоваться русский язык.

#### Требования к программному обеспечению системы

Данная Система должна являться кроссплатформенным приложением и работать на всех операционных системах с поддержкой графического интерфейса.

#### Требования к техническому обеспечению

Используемое при разработке программное обеспечение и библиотеки программных кодов должны иметь широкое распространение, быть общедоступными и использоваться в промышленных масштабах. Базовой программной платформой должна являться операционная система MS Windows.

#### Требования к метрологическому обеспечению

Требования к метрологическому обеспечению не предъявляется.

#### Требования к организационному обеспечению

Организационное обеспечение системы должно быть достаточным для эффективного выполнения персоналом возложенных на него обязанностей при осуществлении автоматизированных и связанных с ними неавтоматизированных функций системы.

К работе с системой должны допускаться сотрудники, имеющие навыки работы на персональном компьютере, ознакомленные с правилами эксплуатации и прошедшие обучение работе с системой.

#### Требования к методическому обеспечению

Данная система должна поставляться с определённым пакетом документации, состоящем из:

* технического задания (ГОСТ);
* пояснительной записки (ГОСТ);
* задания на выполнение курсовой работы;
* руководства пользователя.

## СОСТАВ И СОДЕРЖАНИЕ РАБОТ ПО СОЗДАНИЮ (РАЗВИТИЮ) СИСТЕМЫ

Таблица 1 Содержание Этапов работ

|  |  |  |
| --- | --- | --- |
| № этапа | Содержание работ | Срок |
| 1 | Разработка структуры входных и выходных данных | 26.02.19-03.03.19 |
| 2 | Разработка алгоритма решения задачи | 04.03.19-17.03.19 |
| 3 | Разработка структуры программы | 18.03.19-31.03.19 |
| 4 | Разработка тестов | 01.04.19-07.04.19 |
| 5 | Написание текста программы | 08.04.19-28.04.19 |
| 6 | Отладка программы | 29.04.19-05.05.19 |
| 7 | Написание и оформление отчета о выполненной в виде работе пояснительной записки | 06.05.19-13.05.19 |

## ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ СИСТЕМЫ

### Виды, состав, объем и методы испытаний системы

Для разработанной системы будут сформированы тесты для проверки функций столкновения змейки со стенами и с собой. Тестовые входные записи будут иметь корректные выходные данные, сравнение выходных данных системы с которыми покажет правильность реализации алгоритмов.

### Общие требования к приемке работ по стадиям

Разработка данной Системы делится на шесть стадий:

* получение задания на выполнение курсовой работы;
* составление и согласование технического задания;
* создание и тестирование Системы Исполнителем;
* написание технической документации для Системы;
* демонстрация Системы Заказчику;
* защита курсовой работы.

### Статус приемочной комиссии

Статус приемочной комиссии определяется Заказчиком до проведения испытаний.

## ТРЕБОВАНИЯ К СОСТАВУ И СОДЕРЖАНИЮ РАБОТ ПО ПОДГОТОВКЕ ОБЪЕКТА АВТОМАТИЗАЦИИ К ВВОДУ СИСТЕМЫ В ДЕЙСТВИЕ

В ходе выполнения проекта на объекте автоматизации требуется выполнить работы по подготовке к вводу системы в действие. При подготовке к вводу в эксплуатацию системы должен обеспечить выполнение следующих работ:

* Определить подразделение и ответственных должностных лиц, ответственных за внедрение и проведение опытной эксплуатации системы;
* Обеспечить присутствие пользователей на обучении работе с системой, проводимом Исполнителем;
* Обеспечить соответствие помещений и рабочих мест пользователей системы в соответствии с требованиями, изложенными в настоящем ЧТЗ;
* Обеспечить выполнение требований, предъявляемых к программно-техническим средствам, на которых должно быть развернуто программное обеспечение;
* Совместно с Исполнителем подготовить план развертывания системы на технических средствах Заказчика;
* Привести поступающую в систему информацию (в соответствии с требованиями к информационному и лингвистическому обеспечению) к виду, пригодному для обработки с помощью ЭВМ;
* Провести опытную эксплуатацию системы.

Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие, включая перечень основных мероприятий и их исполнителей должны быть уточнены на стадии подготовки рабочей документации и по результатам опытной эксплуатации.

## ТРЕБОВАНИЯ К ДОКУМЕНТИРОВАНИЮ

Кроме создания работоспособной Системы Исполнитель должен составить пакет документации, состоящий из:

* технического задания;
* пояснительной записки.

# Проектирование и разработка программы



## Проектирование программы

### Функциональные требования к программе

Функциональные требования к программе описаны в пункте 2.4.2 Технического задания.

### Проектирование классов

Программа написана на языке программирования C++, с использованием библиотеки SFML. Для реализации функциональной части системы были реализованы следующие классы (Рис. 1):

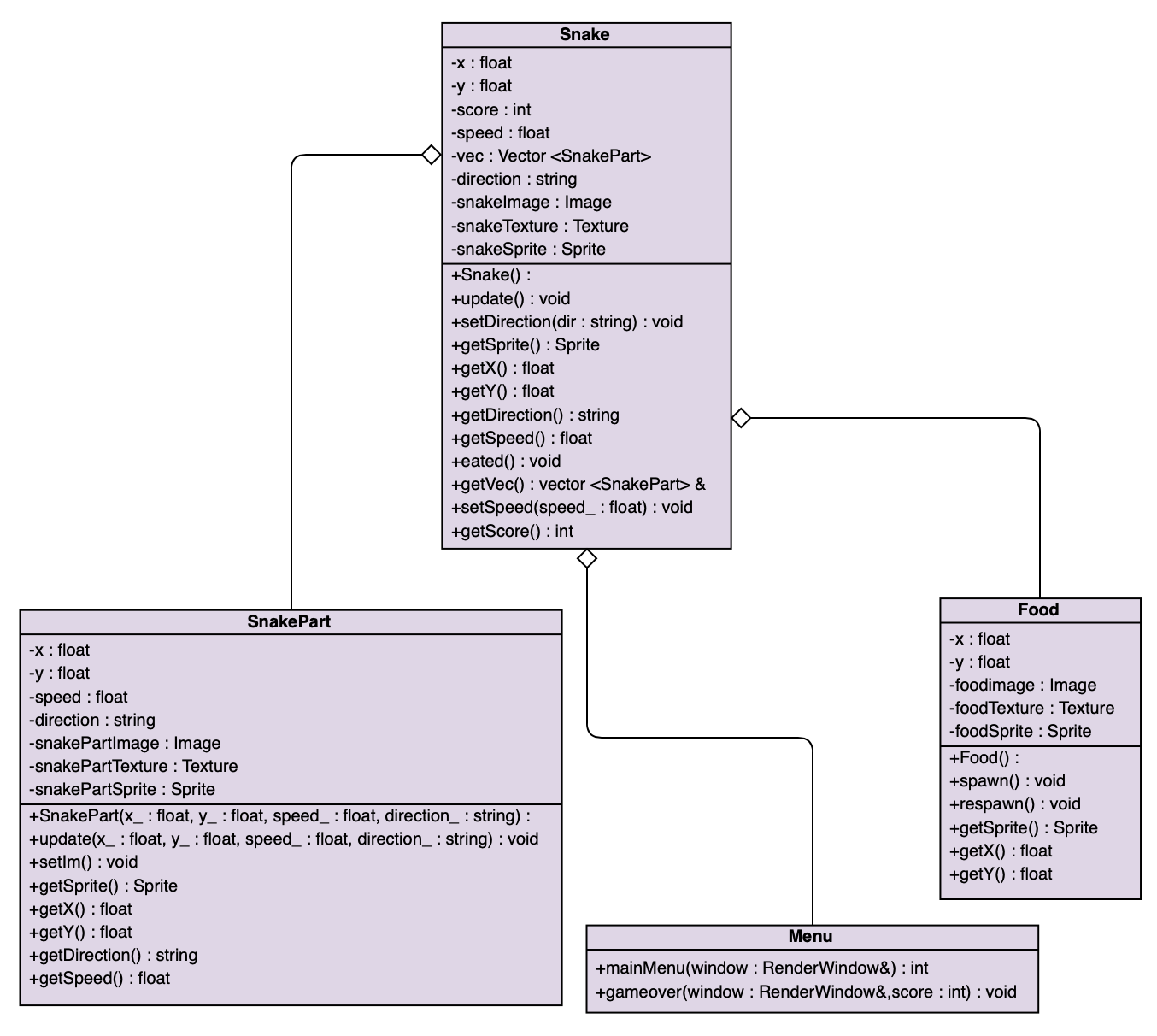


Рис. 1 Общая UML диаграмма

1. Класс Snake (Рис. 2)

Класс, описывающий змейку. Хранит поля, содержащие: координаты, скорость, направление, части тела змейки, текстуру, силуэт, счет, методы доступа к этим полям и методы обновляющие положение змейки и реагирующие на поедание.

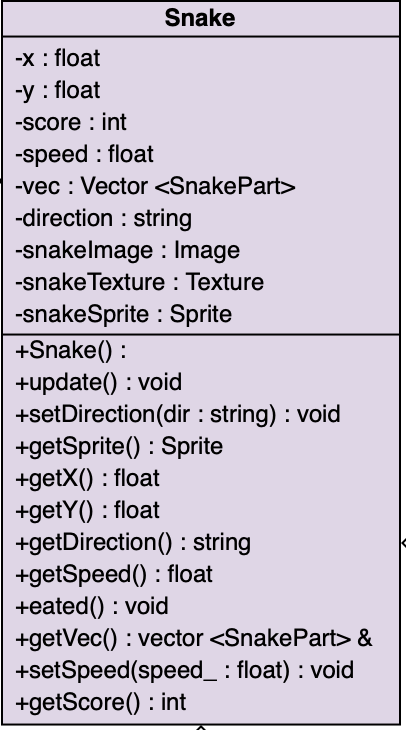


Рис. 2 Класс Snake

1. SnakePart (Рис. 3)

Класс, хранящий в себе координаты частей змейки, их скорость, направление, текстуры, а также методы для доступа к полям.

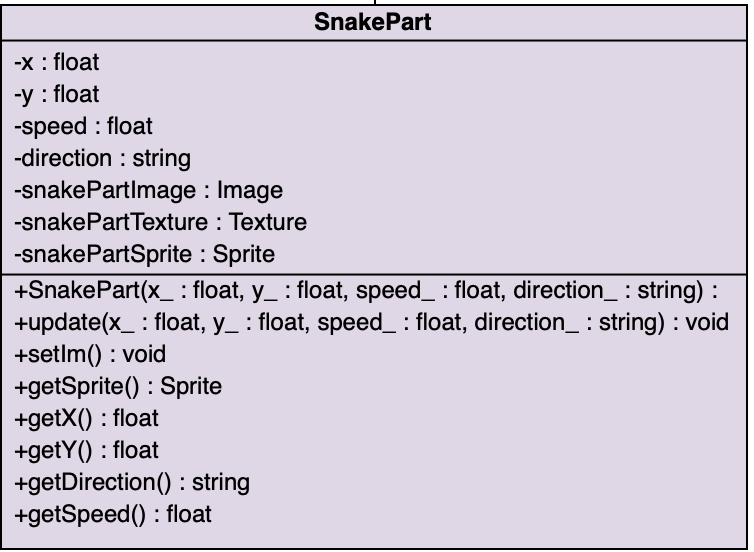


Рис. 3 Класс SnakePart

1. Food (Рис. 4)

Класс, хранящий координаты еды и их текстуры, и методы появление еды и доступа к полям класса.

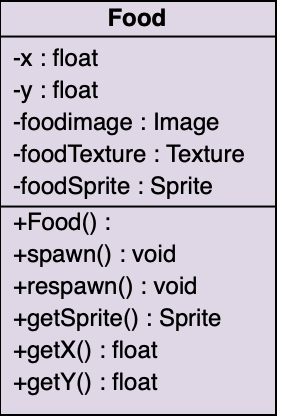


Рис. 4 Класс Food

1. Menu (Рис. 5)

Класс, основной задачей которого является создание главного меню перед началом игры и меню после игры.

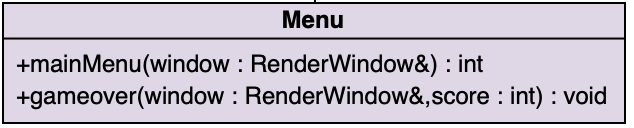


Рис. 5 Класс Menu

### Разработка программы

#### Описание проектного решения

После запуска программы открывается меню, на котором есть две кнопки “Start game” и “Exit” для старта игры и выхода соответственно.

При нажатии кнопки Start game открывается окно с картой и змейкой на ней. Управлять змейкой можно с помощью клавиш Right, Left, Up, Down.

В окне появляются красные точки, которые являются едой, при попадании змейки в координату еды, змейка съедает эту точку и становится больше.

При столкновении со стеной или частью змейки игра завершается и открывается меню.

В меню после игры написан ваш счёт и расположена кнопка “Exit” для выхода из программы.

При нажатии на кнопку Exit программа закрывается.

#### Тестирование программы

Программа была протестирована путем проверки всех ее функциональных возможностей. Все выходные данные соответствуют ожиданиям. Сбоев при работе программы не наблюдалось.

# Руководство пользователя

Программа запускается через exe-файл в папке со всеми необходимыми компонентами.

После запуска программы открывается меню, на котором есть две кнопки “Start game” и “Exit” для старта игры и выхода соответственно (Рис. 9).

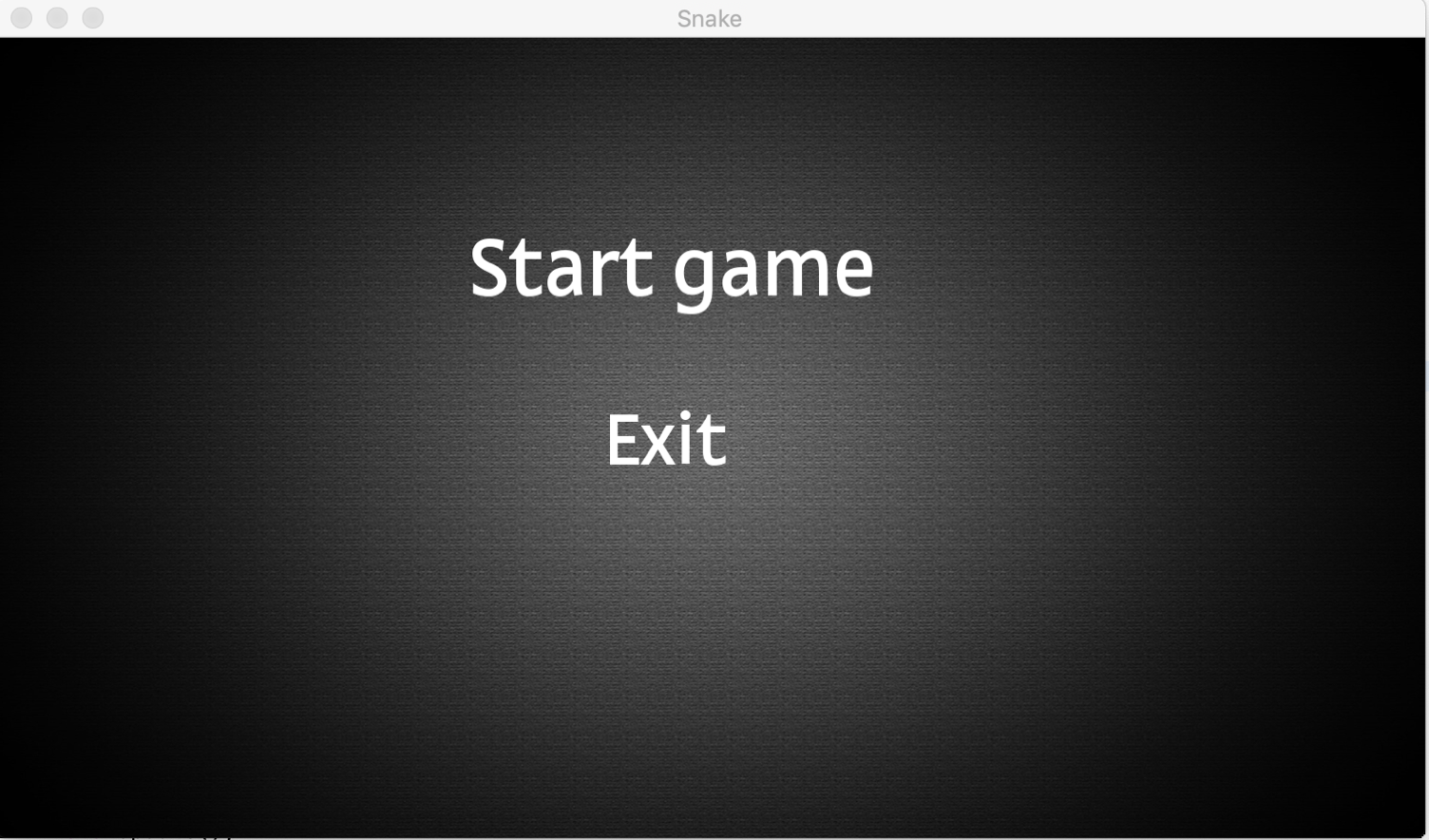


Рис. 9 Главное меню

При нажатии кнопки Start game открывается окно с картой и змейкой на ней. (Рис. 10).

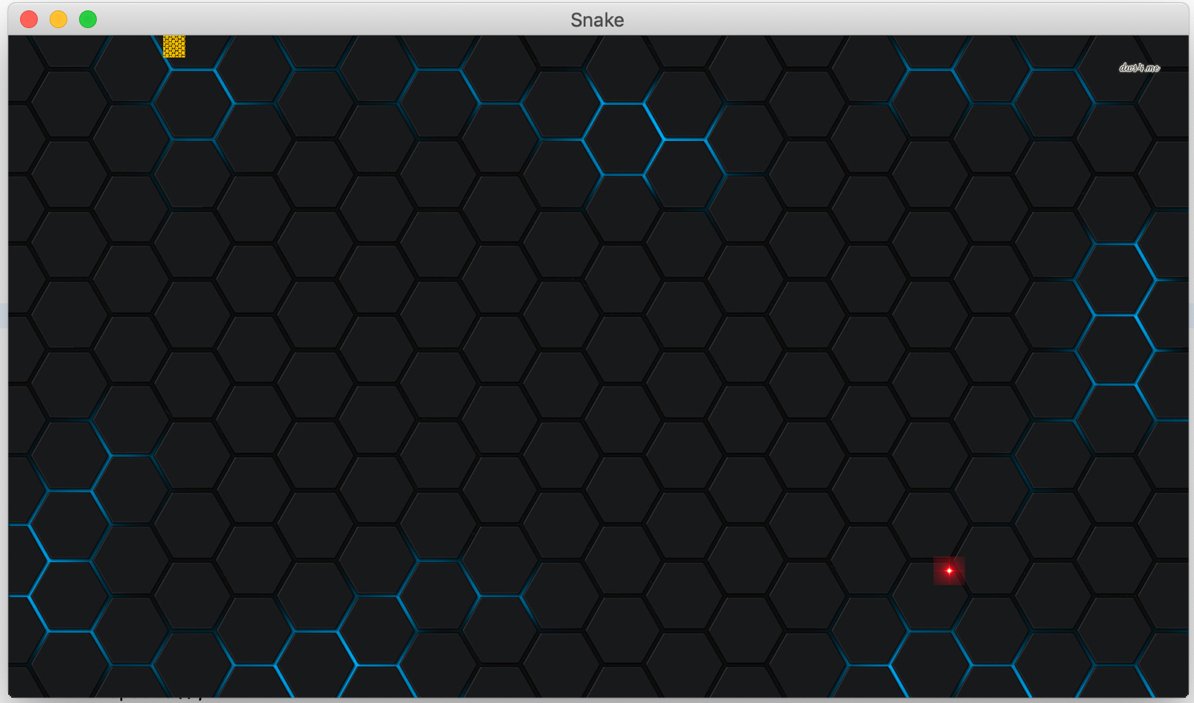


Рис. 10 Окно со змейкой

В окне появляются красные точки, которые являются едой, при попадании змейки в координату еды, змейка съедает эту точку и становится больше. (Рис. 11).



Рис. 11 Змейка в процессе игры

При столкновении со стеной или частью змейки игра завершается и открывается меню. В меню после игры написан ваш счёт и расположена кнопка “Exit” для выхода из программы. (Рис. 12).

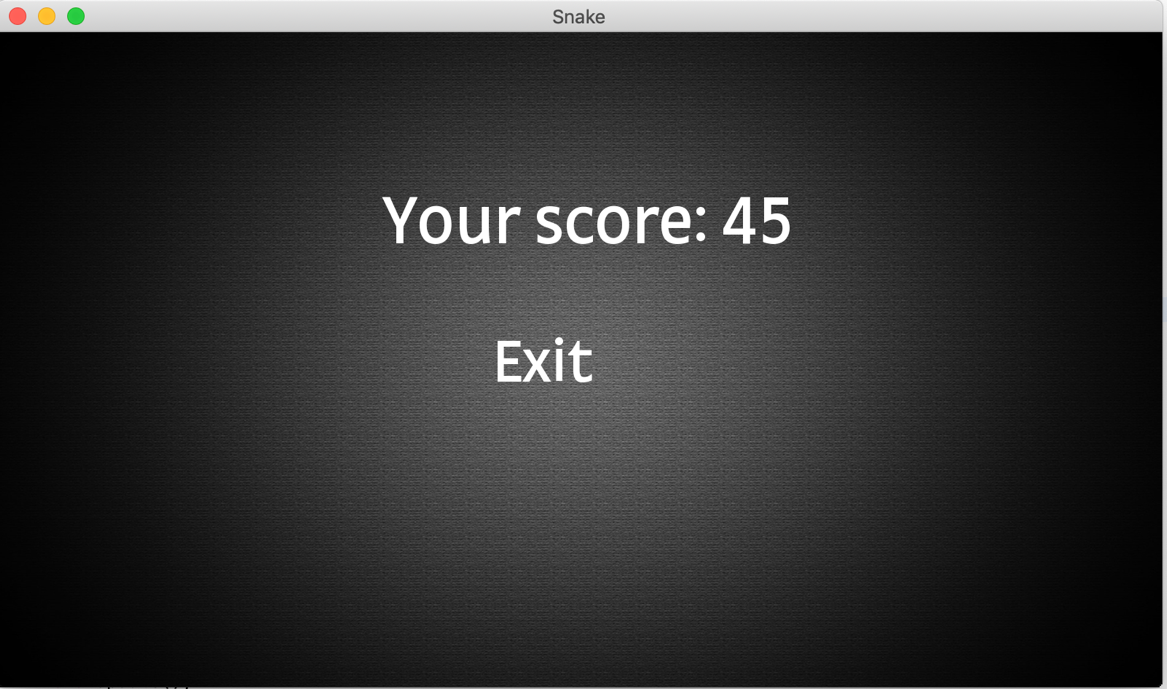


Рис. 12 Меню после игры

При нажатии на кнопку Exit программа закрывается.

# Заключение

В результате выполнения курсовой работы был разработан программный продукт, удовлетворяющий досуг пользователей. При разработке программного продукта были построены UML диаграммы, на основе, которых реализованы классы в самой программе. Для реализации методов классов были использованы структуры данных из STL. Для проверки корректности результатов, получаемых после исполнения алгоритмов, были сформированы выходные тестовые данные. Код проекта разбит на отдельные файлы на основе логики работы отдельных частей кода. Для удобства работы с функционалом картотеки клиентов был реализован пользовательский графический интерфейс. Все требования, предъявленные в Техническом задании, удовлетворены.

# Список литературы

В качестве источников разработки использовались данные ресурсы:

* Зорина, Н.В. Методические указания по выполнению курсовой работы для бакалавров, обучающихся по направлениям 09.03.02«Информационные системы и технологии», 09.03.04 «Программная инженерия» / Н.В. Зорина, Л.Б. Зорин, О.В. Соболев,- Москва, 2017 - 41 с.
* Лафоре Р. Объектно-ориентированное программирование C++. Классика Computer Science. 4-е изд. – СПБ.: Питер, 2018. – 928 с.: ил. – (Серия «Классика Computer Science»).
* https://www.sfml-dev.org/documentation/2.5.1/

# Приложение №1. Исходный код программы

1. **Main.cpp**

#include <SFML/Graphics.hpp>

#include <string>

#include "Snake.hpp"

#include "Food.hpp"

#include <iostream>

#include <cmath>

#include "Menu.hpp"

using namespace sf;

int main( )

{

bool fl = 0;

sf::RenderWindow window(sf::VideoMode( 1600, 900 ), "Snake" );

//SnakePart tst(10,10,1,"right");

sf::Image mapimage;

mapimage.loadFromFile("/Users/daniilavtusko/Desktop/coding/SnakeGame/SnakeGame/Img/mapTexture.jpg");

sf::Texture mapTexture;

mapTexture.loadFromImage(mapimage);

sf::Sprite mapSprite;

mapSprite.setTexture(mapTexture);

Clock clock;

Snake player;

Food f;

f.spawn();

sf::CircleShape shape(10);

shape.setFillColor(sf::Color::Red);

while (window.isOpen())

{

float time = clock.getElapsedTime().asMicroseconds();

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

}

if(abs(player.getX()-f.getX()) < 30 && abs(player.getY() - f.getY()) < 30){

f.respawn();

player.eated();

}

if (player.getX() < 0 || player.getX() > 1600 || player.getY() < 0 || player.getY() > 900){

gameover(window,player.getScore());

}

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

if (player.getVec()[i].getX() == player.getX() && player.getVec()[i].getY() == player.getY())

gameover(window,player.getScore());

//std::cout << player.getX() << ' ' << f.getX() << ' ' << player.getY() << ' ' << f.getY() << std::endl;

if (Keyboard::isKeyPressed(Keyboard::Left)) player.setDirection("left");

if (Keyboard::isKeyPressed(Keyboard::Right)) player.setDirection("right");

if (Keyboard::isKeyPressed(Keyboard::Up)) player.setDirection("up");

if (Keyboard::isKeyPressed(Keyboard::Down)) player.setDirection("down");

if (time > 80000\*player.getSpeed()){

//std::cout << time << std::endl;

if (!player.getVec().empty()){

for (int i = player.getVec().size()-1;i > 0;i--)

player.getVec()[i].update(player.getVec()[i-1].getX(), player.getVec()[i-1].getY(),player.getVec()[i-1].getSpeed(),player.getVec()[i-1].getDirection());

player.getVec()[0].update(player.getX(), player.getY(),player.getSpeed(),player.getDirection());

}

player.update();

clock.restart();

}

if (fl == 0){

int res = mainMenu(window);

fl = 1;

if (res == 1)

return EXIT\_SUCCESS;

}

window.clear();

window.draw(shape);

window.draw(mapSprite);

window.draw(f.getSprite());

window.draw(player.getSprite());

if (!player.getVec().empty()){

player.getVec()[0].setIm();

window.draw(player.getVec()[0].getSprite());

for (int i = 1;i < player.getVec().size();i++){

player.getVec()[i].setIm();

window.draw(player.getVec()[i].getSprite());

}

}

window.display();

}

return EXIT\_SUCCESS;

}

1. **Snake.hpp**

#ifndef Snake\_hpp

#define Snake\_hpp

#include <stdio.h>

#include <SFML/Graphics.hpp>

#include <string>

#include <vector>

#include "SnakePart.hpp"

using namespace sf;

class Snake{

float x,y,speed;

int score;

std::vector <SnakePart> vec;

std::string direction;

Image snakeimage;

Texture snakeTexture;

Sprite snakeSprite;

public:

Snake();

void update();

void setDirection(std::string dir);

Sprite getSprite();

float getX();

float getY();

std::string getDirection();

float getSpeed();

void eated();

std::vector <SnakePart> &getVec();

void setSpeed(float speed\_);

int getScore();

};

#endif /\* Snake\_hpp \*/

1. **Snake.cpp**

#include "Snake.hpp"

using namespace sf;

Snake::Snake(){

speed = 1;

score = 0;

direction = "right";

snakeimage.loadFromFile("/Users/daniilavtusko/Desktop/coding/SnakeGame/SnakeGame/Img/snakeTexture.jpg");

snakeTexture.loadFromImage(snakeimage);

snakeSprite.setTexture(snakeTexture);

}

void Snake::update(){

if (direction == "left"){ snakeSprite.setPosition(x-30, y); x += -30;}

else if (direction == "right"){ snakeSprite.setPosition(x+30, y); x += 30;}

else if (direction == "up"){ snakeSprite.setPosition(x, y-30); y += -30;}

else{ snakeSprite.setPosition(x, y+30); y += 30;}

}

void Snake::setDirection(std::string dir){

direction = dir;

}

Sprite Snake::getSprite(){

return snakeSprite;

}

float Snake::getX(){

return x;

}

float Snake::getY(){

return y;

}

void Snake::eated(){

score++;

if (speed > 0.3)

speed -= 0.01;

if (vec.empty()){

if (direction == "left"){

SnakePart p(x+30,y,speed,direction);

vec.push\_back(p);

}

else if (direction == "right"){

SnakePart p (x-30,y,speed,direction);

vec.push\_back(p);

}

else if (direction == "up"){

SnakePart p (x,y+30,speed,direction);

vec.push\_back(p);

}

else{

SnakePart p (x,y-30,speed,direction);

vec.push\_back(p);

}

}

else{

if (vec[vec.size()-1].getDirection() == "left"){

SnakePart p(vec[vec.size()-1].getX()+30,vec[vec.size()-1].getY(),vec[vec.size()-1].getSpeed(),vec[vec.size()-1].getDirection());

vec.push\_back(p);

}

else if (vec[vec.size()-1].getDirection() == "right"){

SnakePart p (vec[vec.size()-1].getX()-30,vec[vec.size()-1].getY(),vec[vec.size()-1].getSpeed(),vec[vec.size()-1].getDirection());

vec.push\_back(p);

}

else if (vec[vec.size()-1].getDirection() == "up"){

SnakePart p (vec[vec.size()-1].getX(),vec[vec.size()-1].getY()+30,vec[vec.size()-1].getSpeed(),vec[vec.size()-1].getDirection());

vec.push\_back(p);

}

else{

SnakePart p (vec[vec.size()-1].getX(),vec[vec.size()-1].getY()-30,vec[vec.size()-1].getSpeed(),vec[vec.size()-1].getDirection());

vec.push\_back(p);

}

}

}

std::vector<SnakePart> &Snake::getVec(){

return vec;

}

std::string Snake::getDirection(){

return direction;

}

float Snake::getSpeed(){

return speed;

}

int Snake::getScore(){

return score;

}

1. **Food.hpp**

#ifndef Food\_hpp

#define Food\_hpp

#include <stdio.h>

#include <SFML/Graphics.hpp>

using namespace sf;

class Food{

float x,y;

Image foodimage;

Texture foodTexture;

Sprite foodSprite;

public:

Food();

void spawn();

Sprite getSprite();

float getX();

float getY();

void respawn();

};

#endif /\* Food\_hpp \*/

1. **Food.cpp**

#include "Food.hpp"

Food::Food(){

srand(time(0));

foodimage.loadFromFile("/Users/daniilavtusko/Desktop/coding/SnakeGame/SnakeGame/Img/food.png");

foodTexture.loadFromImage(foodimage);

foodSprite.setTexture(foodTexture);

x = rand()%1550;

y = rand()%850;

}

void Food::spawn(){

foodSprite.setPosition(x, y);

}

Sprite Food::getSprite(){

return foodSprite;

}

float Food::getX(){

return x;

}

float Food::getY(){

return y;

}

void Food::respawn(){

x = rand()%1550;

y = rand()%850;

spawn();

}

1. **SnakePart.hpp**

#ifndef SnakePart\_hpp

#define SnakePart\_hpp

#include <stdio.h>

#include <SFML/Graphics.hpp>

#include <string>

using namespace sf;

class SnakePart{

std::string direction;

float x,y,speed;

Image snakePartImage;

Texture snakePartTexture;

Sprite snakePartSprite;

public:

SnakePart(float x\_,float y\_,float speed\_,std::string direction\_);

void update(float x\_,float y\_,float speed\_,std::string direction\_);

std::string getDirection();

float getSpeed();

Sprite getSprite();

float getX();

float getY();

void setIm();

};

#endif /\* SnakePart\_hpp \*/

1. **SnakePart.cpp**

#include "SnakePart.hpp"

SnakePart::SnakePart(float x\_,float y\_,float speed\_,std::string direction\_){

direction = direction\_;

speed = speed\_;

x = x\_;

y = y\_;

snakePartSprite.setPosition(x, y);

}

void SnakePart::update(float x\_,float y\_,float speed\_,std::string direction\_){

if (direction == "left"){ snakePartSprite.setPosition(x\_, y\_); x = x\_; y = y\_;}

else if (direction == "right"){ snakePartSprite.setPosition(x\_, y\_); x = x\_; y = y\_;}

else if (direction == "up"){ snakePartSprite.setPosition(x\_, y\_); y = y\_;x = x\_;}

else{ snakePartSprite.setPosition(x\_, y\_); y = y\_;x = x\_;}

direction = direction\_;

speed = speed\_;

}

std::string SnakePart::getDirection(){

return direction;

}

float SnakePart::getSpeed(){

return speed;

}

Sprite SnakePart::getSprite(){

return snakePartSprite;

}

float SnakePart::getX(){

return x;

}

float SnakePart::getY(){

return y;

}

void SnakePart::setIm(){

snakePartImage.loadFromFile("/Users/daniilavtusko/Desktop/coding/SnakeGame/SnakeGame/Img/snakePartTexture.jpg");

snakePartTexture.loadFromImage(snakePartImage);

snakePartSprite.setTexture(snakePartTexture);

}

1. **Menu.hpp**

#ifndef Menu\_hpp

#define Menu\_hpp

#include <stdio.h>

#include <SFML/Graphics.hpp>

#include <sstream>

using namespace sf;

void gameover(RenderWindow& window,int score);

int mainMenu(RenderWindow& window);

#endif /\* Menu\_hpp \*/

1. **Menu.cpp**

#include "Menu.hpp"

void gameover(RenderWindow& window,int score){

Font font;

std::ostringstream ScoreString;

ScoreString << score;

font.loadFromFile("/Users/daniilavtusko/Desktop/coding/SnakeGame/SnakeGame/fontGameOver.ttf");

Text sc("Your score:",font,90);

Text exit("Exit",font,80);

sc.setString("Your score: "+ScoreString.str());

sc.setPosition(530, 200);

exit.setPosition(680, 400);

Texture textureBg;

textureBg.loadFromFile("/Users/daniilavtusko/Desktop/coding/SnakeGame/SnakeGame/Img/textureBg.jpg");

Sprite spriteBg(textureBg);

bool isMenu = true;

int menuNum = 0;

while (isMenu) {

if (IntRect(680, 400, 100, 100).contains(Mouse::getPosition(window))) { exit.setColor(Color::Blue); menuNum = 1; }

if (!IntRect(680, 400, 100, 100).contains(Mouse::getPosition(window))) { exit.setColor(Color::White); }

if (Mouse::isButtonPressed(Mouse::Left))

{

if (menuNum == 1) isMenu = false;//если нажали первую кнопку, то выходим из меню

}

window.clear();

window.draw(spriteBg);

window.draw(exit);

window.draw(sc);

window.display();

}

window.close();

}

int mainMenu(RenderWindow& window){

Font font;

font.loadFromFile("/Users/daniilavtusko/Desktop/coding/SnakeGame/SnakeGame/fontGameOver.ttf");

Text start("Start game",font,90);

Text exit("Exit",font,80);

start.setPosition(530, 200);

exit.setPosition(680, 400);

Texture textureBg;

textureBg.loadFromFile("/Users/daniilavtusko/Desktop/coding/SnakeGame/SnakeGame/Img/textureBg.jpg");

Sprite spriteBg(textureBg);

bool isMenu = true;

int menuNum = 0;

while (isMenu) {

if (IntRect(530, 200, 500, 100).contains(Mouse::getPosition(window))) { start.setColor(Color::Blue); menuNum = 2; }

if (!IntRect(530, 200, 500, 100).contains(Mouse::getPosition(window))) { start.setColor(Color::White); }

if (IntRect(680, 400, 100, 100).contains(Mouse::getPosition(window))) { exit.setColor(Color::Blue); menuNum = 1; }

if (!IntRect(680, 400, 100, 100).contains(Mouse::getPosition(window))) { exit.setColor(Color::White); }

if (Mouse::isButtonPressed(Mouse::Left))

{

if (menuNum == 1) return 1;//если нажали первую кнопку, то выходим из меню

if (menuNum == 2) return 2;

}

window.clear();

window.draw(spriteBg);

window.draw(exit);

window.draw(start);

window.display();

}

return 0;

}

# Приложение №2. UML-диаграмма

