МИНОБРНАУКИ РОССИИ

федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

**Институт компьютерных наук и технологий**

**Высшая инженерная школа**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**Создание игры «змейка» на языке программирования java**

по программе профессиональной переподготовки:

«Разработчик прикладного программного обеспечения (Язык Java)»

Выполнила:

Васильева Евгения Андреевна

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель:

Зайцев Игорь Владимирович

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург 2021

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc82098653)

[1. ВВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ 4](#_Toc82098654)

[2. ОПРЕДЕЛЕНИЕ ЦЕЛИ И ПОСТАНОВКА ЗАДАЧ 6](#_Toc82098655)

[3. ФАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ 7](#_Toc82098656)

[3.1 Разработка структуры классов приложения 7](#_Toc82098657)

[3.1.1 Разработка графического интерфейса 7](#_Toc82098658)

[3.1.2 Разработка механики игры 8](#_Toc82098659)

[3.2 Реализация графического интерфейса 11](#_Toc82098663)

[3.2.1 Создание класса клетки 11](#_Toc82098664)

[3.2.2 Создание класса игрового поля и общего графического интерфейса 12](#_Toc82098665)

[3.2.3 Создание класса для хранения констант 14](#_Toc82098666)

[3.2.4 Создание перечислимого типа для подгрузки текстур 15](#_Toc82098667)

[3.3 Реализация механики игры 16](#_Toc82098668)

[3.3.1 Создание класса клетки 16](#_Toc82098669)

[3.3.2 Создание главного управляющего класса 17](#_Toc82098670)

[4. ТЕСТИРОВНИЕ ПРИЛОЖЕНИЯ 22](#_Toc82098671)

[4.1 Разработка тестового плана и тестового сценария 22](#_Toc82098672)

[4.2 Запуск тестового сценария и составление отчета о тестировании 23](#_Toc82098673)

[ЗАКЛЮЧЕНИЕ 26](#_Toc82098674)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 28](#_Toc82098675)

# ВВЕДЕНИЕ

Язык программирования Java предоставляет своим пользователям огромное количество возможностей для реализации практически любых решений. В частности, на Java можно реализовать как «тяжелые» корпоративные приложения и интерфейсы, так и легковесные приложения небольшого масштаба, а также графический интерфейс [1]. Таким образом, язык программирования Java, несмотря на то, что был создан впервые в середине 90-x годов (официальная дата создания 23 мая 1995 года [2], не только не устарел, но и активно используется для создания современных «легковесных» приложений, а также за много лет его использования накопил большое количество различных библиотек для решения не только бизнес-задач, но и для создания развлекательного контента, в частности игр. В данной работе будет рассмотрен процесс создания игры «Змейка» с помощью Lightweight Java Game библиотеки в качестве демонстрации создания «легковесного» приложения посредством языка программирования Java.

# ВВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ

Java — строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (на данный момент принадлежит компании Oracle). Язык и основные реализующие его технологии распространяются по лицензии GPL.

Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины. [2]

Для создания графики приложения будет использована библиотека Lightweight Java Game (далее библиотека LWJG). Это открытая графическая библиотека, основной целью которой является предоставление простого и легковесного программного интерфейса для создателей компьютерных игр на языке Java.

Библиотека LWJG является высокопроизводительной кроссплатформенной библиотекой, широко используемой в разработке компьютерных игр и мультимедийных приложениях. Она предоставляет доступ к OpenGL, OpenAL, OpenCL и обеспечивает платформонезависимый доступ к различным манипуляторам, таким как геймпады, рули и джойстики.[…]

Библиотека LWJG доступна под BSD-лицензией. Будучи открытой и бесплатной, является основой многих игровых движков и библиотек. [3]

Для работы с библиотекой LWJG будет использоваться среда разработки IntelliJ IDEA, так как она является наиболее оптимальным инструментом для подключения пакетов библиотеки LWJG. IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Начиная с шестой версии продукта IntelliJ IDEA предоставляет интегрированный инструментарий для разработки графического пользовательского интерфейса. [4]

OpenGL (Open Graphics Library) — спецификация, определяющая платформонезависимый (независимый от [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) [программный интерфейс](https://ru.wikipedia.org/wiki/API) для написания приложений, использующих двумерную и трёхмерную [компьютерную графику](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0). [5]

Snake (Питон, Удав, Змейка и др.) — [компьютерная игра](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), в которой игрок управляет длинным, тонким существом, напоминающим [змею](https://ru.wikipedia.org/wiki/%D0%97%D0%BC%D0%B5%D1%8F), которое ползает по [плоскости](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%BE%D1%81%D0%BA%D0%BE%D1%81%D1%82%D1%8C_(%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F)) (как правило, ограниченной стенками), собирая еду (или другие предметы), избегая столкновения с собственным хвостом и краями игрового поля. Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру. [6]

# ОПРЕДЕЛЕНИЕ ЦЕЛИ И ПОСТАНОВКА ЗАДАЧ

Целью данной выпускной квалификационной работы является реализация игры «Змейка» на платформе языка Java с помощью Lightweight Java Game библиотеки.

Для реализации поставленной цели необходимо решить следующие задачи:

1. Разработать требования к приложению
2. Разработать структуру классов приложения
3. Реализовать графический интерфейс с помощью средств библиотеки Lightweight Java Game
4. Реализовать механику игры с помощью средств библиотеки Lightweight Java Game
5. Выполнить тестирование приложения:

* разработать план тестирования и тестовые сценарии;
* выполнить тестовые сценарии;
* составить отчет о тестировании.

# ФАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ

# Разработка требований

Основные требования к приложению:

1. Приложение должно представлять собой окно графического интерфейса с возможностями свёртки панели задач, увеличения на весь экран и полного закрытия окна с выходом из приложения

2. Приложение должно управляться пользователем с клавиатуры с помощью клавиш «вверх», «вниз», «вправо» и «влево»

3. Графический интерфейс приложения должен содержать поле, разделенное на ровные клетки. В клетках должны отображаться «фрукты» в произвольном порядке, а также «змейка», которая должна начинать свое движение вправо из левого нижнего угла.

4. Логика игры должна соответствовать блок-схеме, приведенной ниже на рисунке (рис 3.1):

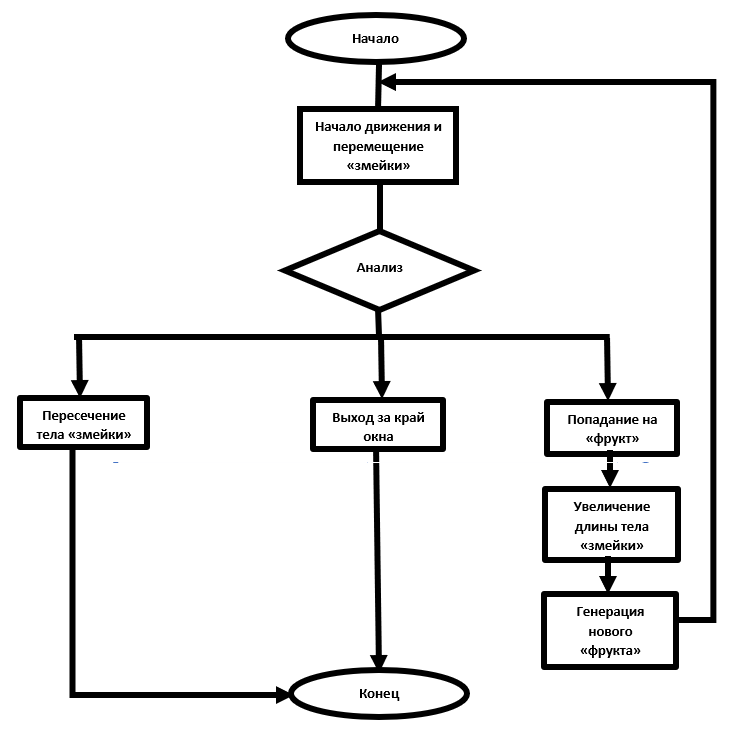


Рисунок 3.1 Блок-схема логики игры

# Разработка структуры классов приложения

Создание компьютерной игры в большинстве случаев включает в себя две основные задачи:

1. Создание графического интерфейса

2. Реализация механики игры

Соответственно при разработке и выборе структуры классов приложения прежде всего необходимо опираться на данные задачи.

### Разработка графического интерфейса

Игра «Змейка» включает в себя следующие основные графические элементы:

* Игровое поле (окно), размеченное по всему периметру одинаковыми по размеру клетками (ячейками)
* Фигура «змейки» (совокупность последовательно расположенных «кружков»), которая будет перемещаться по клеткам поля
* Фигура «фрукта», который будет располагаться в некоторых клетках

На уровне приложения мы реализуем это с помощью отдельного класса игрового поля и общего графического интерфейса. Заполнение поля клетками, передвижение «змейки» по клеткам и появление «фруктов» в клетках будет создаваться с использованием констант, соответственно для этого нам понадобится реализовать отдельный класс для хранения данных констант, чтобы была возможность обращаться к этим константам из класса клетки и класса игрового поля и общего графического интерфейса.

Также будет необходим отдельный класс перечислимого типа («enumeration») для хранения графического изображения («спрайт») тела «змейки» и «фрукта» и реализация отрисовки текстур, так как к графическим изображениям элементов игры и графическим текстурам также необходимо будет обращаться из класса клетки и класса игрового поля и общего графического интерфейса, поэтому будет эффективнее их вынести в отдельный класс.

### Разработка механики игры

Общий процесс игры «Змейка» следующий: «змейка» передвигается по клеткам поля, попадая на клетку поля, где находится «фрукт», «змейка» «съедает» «фрукт» и, таким образом, увеличивает длину своего тела на одну клетку. Если «змейка» «врезается» в край поля или в свое тело, игра заканчивается и осуществляется выход из приложения. Передвижение «змейки» управляется игроком с клавиатуры с помощью клавиш «вверх-вниз», «вправо-влево».

Реализовать логику перемещения «змейки» можно разными способами: например, можно перемещать каждый элемент «змейки» или перемещать только последние элементы «змейки».

Если перемещать каждый элемент, потребуется запоминать координаты точек изгиба «змейки», для того, чтобы каждый элемент проходил через эти точки. Второй метод гораздо проще и целесообразней, поэтому выберем его для реализации в игре. Для передвижения перемещаем последний элемент змейки и ставим его вперёд, по ходу движения. Таким образом, змейка движется по полю и каждый её элемент будет проходить через точку изгиба. Будет создан динамический массив из элементов змейки и помещён в цикл для перебора элементов.

Таким образом, каждый элемент тела «змейки» («кружок») должен изображаться на экране столько «тиков», какова длина «змейки». Значит, необходимо сообщить клетке, в которую попадает «змейка», что она должна «гореть» определённое количество секунд, а каждый «тик» уменьшать это число у каждой клетки с ненулевым таймером, и менять «спрайт», если «змейка» из клетки уже выползла (то есть таймер стал равен нулю). В случае же необходимости удлинить цепочку, достаточно просто не уменьшать время «горения» клеток на каком-то «тике». Для этого будет необходимо реализовать метод, который будет обновлять ячейку, в случае если «змейка» что-то «съела», и добавить его в оба класса, отвечающих за графическую реализацию.

Реализация алгоритма механики игры будет произведена в отдельном классе, который также будет являться входным и главным управляющим классом приложения, и будет использовать графический интерфейс, определенный в других классах приложения.

# Реализация графического интерфейса

### Создание класса игрового поля и общего графического интерфейса

Для создания игрового поля и общего графического интерфейса, как упоминалось ранее, будет создан отдельный класс GUI, также этот класс будет хранить и обновлять состояние всех графических элементов приложения. Класс будет наследовать классы библиотеки Lightweight Java Game (далее библиотека LWJG) для реализации графики приложения, а также классы самого приложения и классы библиотеки java.util.Random.

Класс GUI будет реализовывать следующие задачи:

1. Класс будет выполнять инициализацию OpenGL в отдельном методе.
2. Класс будет отрисовывать ячейки поля. Для данной задачи определим публичный метод draw, который будет рисовать все клетки поля и вызывать приватный метод drawElement, который будет рисовать элемент, переданный в аргументе.
3. В классе необходимо производить обновление ячеек поля, когда этого запросит главный цикл. Для этого определим публичный метод update, который будет получать на вход булевское значение и вызываться извне. А также приватный метод UpdateOpenGL, который будет использоваться только локально.

Структура класса игрового поля и общего графического интерфейса GUI (табл. 3.1):

Таблица 3.1

Класс GUI

|  |  |  |
| --- | --- | --- |
| Название поля/метода | Тип данных/Тип возвращаемого значения | Назначение |
| Cells | Cell[][] | Хранение текущих значений ячеек |
| initializeOpenGL() | void | Инициализация OpenGL, где задается размер и имя будущего окна и содается окно, а также задается фоновый цвет, прозрачность и текстура |
| draw() | void | Очищает экран от старого изображения и рисует все клетки |
| drawElement(Cell elem) | void | Рисует элемент, переданный в аргументе |
| getState(int x, int y) | cells[x][y].getState() | Вызов геттера для состояния клетки |
| setState(int x, int y, int state) | cells[x][y].setState(state) | Вызов сеттера для состояния клетки |
| updateOpenGL() | void | Производит обновление, когда этого запрашивает главный цикл (вызывается локально) |
| update(boolean have\_to\_decrease) | void | Производит обновление, когда этого запрашивает главный цикл (вызывается извне) |
| init() | void | Создает начальное поле |

### Создание класса для хранение констант

Для хранения констант, которые будут использоваться в классах приложения определим отдельный класс Constants с приватными статическим полями (табл. 3.2):

Таблица 3.2

Класс для хранения констант Constants

|  |  |  |
| --- | --- | --- |
| Название поля/метода | Тип данных/Тип возвращаемого значения | Назначение |
| CELL\_SIZE | int | Задает размер игровой ячейки |
| CELLS\_COUNT\_X | int | Задает размер игрового поля в ячейках |
| CELLS\_COUNT\_Y | int | Задает размер игрового поля в ячейках |
| INITIAL\_SPAWN\_CHANCE | float | Задает процент появления фруктов на старте приложения в процентах |
| FPS | int | Задает количество клеток за один фрейм |
| SCREEN\_WIDTH | int | Задает ширину окна |
| SCREEN\_HEIGHT | int | Задает высоту окна |
| SCREEN\_NAME | string | Задает название окна |

### Создание перечислимого типа для подгрузки текстур

За подгрузку текстур будет отвечать отдельный перечислимы тип Sprite (табл. 3.3):

Таблица 3.3

Перечислимый тип Sprite

|  |  |  |
| --- | --- | --- |
| Название поля/метода | Тип данных/Тип возвращаемого значения | Назначение |
| BODY | - | Текстура для змеи |
| APPLE | - | Текстура для фруктов |
| texture | Texture | Создание текстуры |
| Sprite(String texturename) | Sprite | Подгрузка текстуры |
| getTexture() | Texture | Геттер для текстуры |

# Реализация механики игры

### Создание класса клетки

В классе клетки Cell определим приватные поля для хранения координат ячейки, а также приватное поле для хранения состояния ячейки со следующей логикой:

* Значение 0 будет означать, что клетка пуста
* Значение > 0 будет означать, что в ячейке тело «змейки», которое будет там ещё N фреймов
* Значение < 0, например, -1 будет означать, что в ячейки «фрукт»

Далее определим конструктор класса, который будет выставлять начальные значения координат и состояния.

Также класс будет содержать геттеры координат и состояния ячейки, высоты и ширины окна, а также сеттер состояния ячейки, которые будут позже использоваться в классе GUI.

В классе реализуем публичный метод update для обновления клетки, который будет получать на вход булевское значение и уменьшать время «горения» клетки, если это необходимо.

Публичный метод getSprite будет отрисовывать изображение ячейки (змея, пустая ячейка или фрукт).

Далее нам будет необходимо добавить метод init, создающий начальное поле, в класс GUI: в методе будет инициализировать OpenGL, затем массив Cells [][] cell и будем заполнять его клетками со случайным полем state.

Структура класса клетки Cell (табл. 3.4):

Таблица 3.4

Класс клетки Cell

|  |  |  |
| --- | --- | --- |
| Название поля/метода | Тип данных/Тип возвращаемого значения | Назначение |
| 1 | 2 | 3 |
| x | int | Хранит координату клетки |
| y | int | Хранит координату клетки |
| state | int | Хранит состояние клетки |
| Cell(int x, int y, int state) | - | Конструктор для выставления начальных значений координат и состояния |
| getX() | int | Геттер для координаты Х |

Продолжение таблицы 3.4

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| getY() | int | Геттер для координаты Y |
| getHeight() | int | Геттер для высоты клетки |
| getWidth() | int | Геттер для ширины клетки |
| getState() | int | Геттер для состояния клетки |
| setState(int state) | int | Сеттер для состояния клетки |
| update(boolean have\_to\_increase) | void | Метод для обновления клетки |
| Sprite getSprite() | Sprite | Метод для изображения ячейки (змея, пустая ячейка или фрукт) |

### Создание главного управляющего класса

Для начала объявим в классе Main следующие поля:

* переменная, при обращении которой в true приложение закрывается
* переменные, которые будут предоставлять данные о «змейке»: «змейка» будет выползать будет из нижнего левого угла вправо (направления посчитаны по часовой стрелке от севера, т.е. 0 - вверх, 1 - вправо, 2 - вниз, 3 - влево)
* флаг, который будет обращаться в false, если на данном тике «змейка» что-то съела

В классе будет объявлен входной метод, в котором инициализируется графический интерфейс, создается фрукт в случайном месте, далее в цикле, пока не получен сигнал на закрытие приложения, проверяется ввод данных, двигается змейка и обновляются и рисуются графические элементы.

Также необходимо реализовать метод move, определяющий передвижение «змейки»:

1. Увеличивается длина «змейки», если на предыдущем шаге был съеден фрукт
2. Изменение координат в зависимости от направления с помощью цикла switch-case
3. Осуществляется проверка, не вышла ли «змейка» за границы окна с помощью цикла if-else
4. Осуществляется проверка состояния ячейки, в которую зашла «змейка» с помощью цикла if:

* если там «змейка», то выходим из приложения;
* если там «фрукт», то увеличиваем длину «змейки» на единицу, создаем новый «фрукт»

Метод generate\_new\_object будет создавать новый «фрукт» по следующему алгоритму генерации новой еды:

Высчитывается количество клеток, которые не заполнены «змейкой», по формуле (3.1):

*CELLS\_COUNT\_X\*CELLS\_COUNT\_Y-length* (3.1)

где *CELLS\_COUNT\_X* – количество клеток по горизонтали;

*CELLS\_COUNT\_Y* – количество клеток по вертикали;

*length* – длина «змейки».

Далее выбирается случайная клетка и её номер сохраняется в переменную point. Затем проходимся по всем клеткам, и, если в клетке не «змейка», счетчик уменьшается. Как только счётчик равен нулю, в этой клетке создается «фрукт» и осуществляется выход из цикла.

Для управления игрой определяем метод input, который будет обрабатывать события с клавиатуры с помощью цикла switch-case, а также обрабатывать событие на закрытие и выход из приложения.

Структура главного управляющего класса Main (табл. 3.5):

Таблица 3.5

Главный управляющий класс Main

|  |  |  |
| --- | --- | --- |
| Название поля/метода | Тип данных/Тип возвращаемого значения | Назначение |
| 1 | 2 | 3 |
| isExitRequired | boolean | Переменная, при обращении которой в true приложение закрывается |
| x | int | Переменные, которые определяют изначальное положение змейки и направление ее движения |
| y | int |
| direction | int |
| length | int |
| have\_to\_decrease | boolean | Переменная, которая обращается в false, если на данном тике «змейка» «съела» «фрукт» |
| main(String[] args) | void | Входной метод, в котором инициализируется графический интерфейс, создается фрукт в случайном месте, проверяется ввод данных, двигается змея, обновляются и рисуются графические элементы |
| move() | void | Метод, определяющий передвижение змеи: изменение координат в зависимости от направления, проверка не вышла ли змея за границы окна, проверка состояния ячейки, в которую зашла змея (если там змея, то выходим из приложения; если там фрукт, то увеличиваем длину змейки на единицу, создаем новый фрукт) |

Продолжение таблицы 3.5

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| input() | void | Перебираем события на клавиатуре |
| generate\_new\_obj() | void | Метод, который создает новый фрукт |

Таким образом, общая структура классов приложения будет выглядеть следующим образом (рис. 1.2):

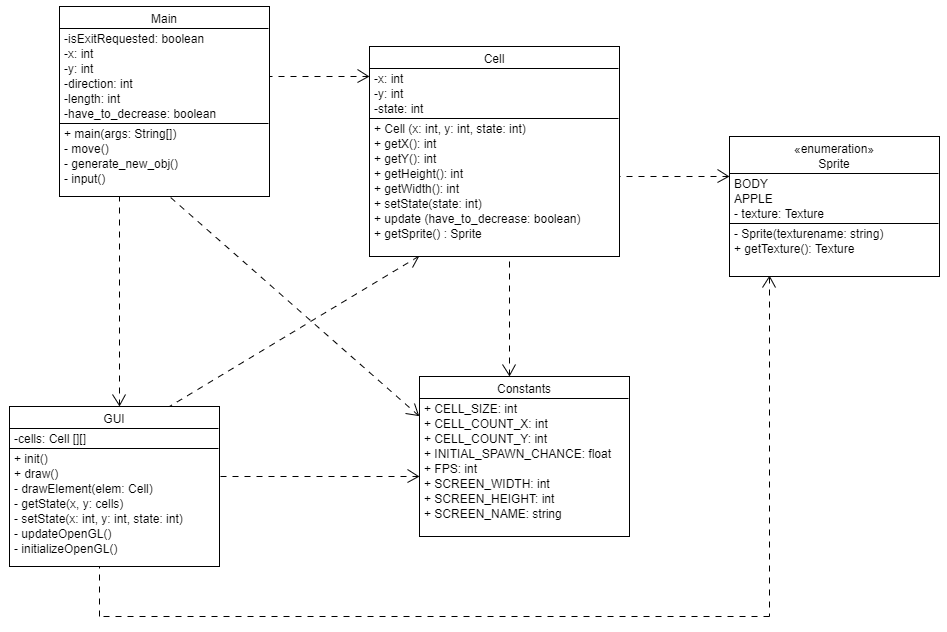


Рисунок 1.2 Общая схема классов приложения

Графический интерфейс приложения приведен на рисунке 1.3:

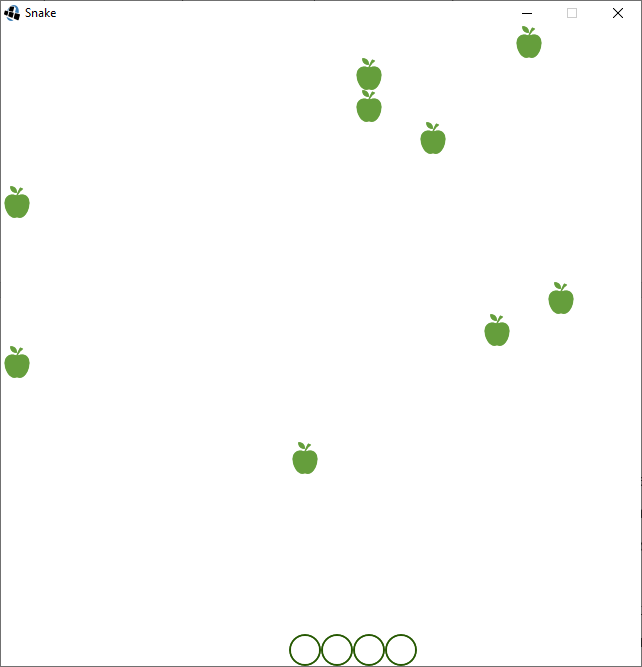


Рисунок 1.3 Графический интерфейс приложения

# ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

# Разработка тестового плана и тестового сценария

В рамках тестирования необходимо осуществить тестирование следующей основной функциональности приложения:

* Запуск приложения
* Завершение приложения
* Графический дизайн приложения
* Механика игры
* Управление с клавиатуры

Для осуществления полного функционального тестирования предлагается использовать следующий тестовый сценарий, который включает в себя всю основную функциональность приложения (табл. 3.6):

Таблица 3.6

Тестовый сценарий

|  |  |  |
| --- | --- | --- |
| № тест-кейса | Описание проверки | Ожидаемый результат |
| 1 | 2 | 3 |
| 1 | Запуск игры и графическая реализация | Приложение успешно запускается: запускается окно с полем, разбитым на равные клетки, в которых в произвольном порядке изображены "фрукты". "Змейка" начинает движение вправо из левого нижнего угла. |
| 2 | Графическая реализация | Окно приложения имеет наверху панель с названием игры («Snake»), справа наверху есть кнопки свертывания окна, увеличения окна и закрытия окна |
| 3 | Механика игры | Змейка передвигается поклеточно |
| 4 | Механика игры | При начале движения "змейка" двигается прямо, пока ее движение не будет изменено с помощью клавиш на клавиатуре |

Продолжение таблицы 3.6

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 5 | Механика игры | При нажатии клавиши "вверх" на клавиатуре "змейка" изменяет движение и начинает передвигаться наверх |
| 6 | Механика игры | При нажатии клавиши "вниз" на клавиатуре "змейка" изменяет движение и начинает передвигаться вниз |
| 7 | Механика игры | При нажатии клавиши "вправо" на клавиатуре "змейка" изменяет движение и начинает передвигаться направо |
| 8 | Механика игры | При нажатии клавиши "влево" на клавиатуре "змейка" изменяет движение и начинает передвигаться налево |
| 9 | Механика игры | При попадании на клетку, в которой изображен "фрукт", "змейка" "съедает" "фрукт" и увеличивается в длине на одну клетку |
| 10 | Механика игры | Если "змейка" доходит до окна с любой стороны, игра завершается и осуществляется выход из приложения посредством закрытия окна игры |
| 11 | Механика игры | Если "змейка" по падает на клетку, где находится ее тело, игра завершается и осуществляется выход из приложения посредством закрытия окна игры |
| 12 | Механика игры | Если "змейка" "съела" "фрукт", новый "фрукт" должен появиться в произвольной ячейке |
| 13 | Завершение игры | При нажатии Х в правом верхнем углу окна приложения, осуществляется выход из приложения посредством закрытия окна игры |

# Запуск тестового сценария и составление отчета о тестировании

В ходе тестирования было выполнено 13 тест-кейсов, которые проверяли работу приложения «Змейка». Суть тестирования заключалась в проверке работы всех обязательных функций приложения и его корректная графическая реализация.

Ниже приведены результаты тестирования (табл. 3.7):

Таблица 3.7

Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № тест-кейса | Описание проверки | Ожидаемый результат | Результат тестирования |
| 1 | 2 | 3 | 4 |
| 1 | Запуск игры и графическая реализация | Приложение успешно запускается: запускается окно с полем, разбитым на равные клетки, в которых в произвольном порядке изображены "фрукты". "Змейка" начинает движение вправо из левого нижнего угла. | Положительно |
| 2 | Графическая реализация | Окно приложения имеет наверху панель с названием игры («Snake»), справа наверху есть кнопки свертывания окна, увеличения окна и закрытия окна | Положительно |
| 3 | Механика игры | Змейка передвигается поклеточно | Положительно |
| 4 | Механика игры | При начале движения "змейка" двигается прямо, пока ее движение не будет изменено с помощью клавиш на клавиатуре | Положительно |
| 5 | Механика игры | При нажатии клавиши "вверх" на клавиатуре "змейка" изменяет движение и начинает передвигаться наверх | Положительно |
| 6 | Механика игры | При нажатии клавиши "вниз" на клавиатуре "змейка" изменяет движение и начинает передвигаться вниз | Положительно |

Продолжение таблицы 3.7

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 7 | Механика игры | При нажатии клавиши "вправо" на клавиатуре "змейка" изменяет движение и начинает передвигаться направо | Положительно |
| 8 | Механика игры | При нажатии клавиши "влево" на клавиатуре "змейка" изменяет движение и начинает передвигаться налево | Положительно |
| 9 | Механика игры | При попадании на клетку, в которой изображен "фрукт", "змейка" "съедает" "фрукт" и увеличивается в длине на одну клетку | Положительно |
| 10 | Механика игры | Если "змейка" доходит до окна с любой стороны, игра завершается и осуществляется выход из приложения посредством закрытия окна игры | Положительно |
| 11 | Механика игры | Если "змейка" по падает на клетку, где находится ее тело, игра завершается и осуществляется выход из приложения посредством закрытия окна игры | Положительно |
| 12 | Механика игры | Если "змейка" "съела" "фрукт", новый "фрукт" должен появиться в произвольной ячейке | Положительно |
| 13 | Завершение игры | При нажатии Х в правом верхнем углу окна приложения, осуществляется выход из приложения посредством закрытия окна игры | Положительно |

Таким образом, все тесты дали положительный результат и основная функциональность приложения работает без дефектов.

# ЗАКЛЮЧЕНИЕ

В работе представлены результаты разработки приложения-игры «Змейка». Программное обеспечение разработано в среде IntelliJ IDEA и с использованием библиотек Lightweight Java Game (LWJG). В ходе выполнения работы были решены следующие задачи:

* разработаны требования приложения
* разработана структура классов приложения
* разработаны классы приложения (5 классов)
* разработана логика приложения
* разработан пользовательский интерфейс приложения
* реализована механика игры в приложении
* разработан тестовый план для организации тестирования
* проведено тестирование

В ходе тестирования не было обнаружено дефектов, вся функциональность приложения работала в полном объеме. Продолжение работы над данной темой может включать в себя доработку функционала, оптимизацию и усложнения алгоритмов работы игры.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Хабр. Как выбрать язык программирования для создания Андроид — приложения: [Электронный документ]. (https://habr.com/ru/post/477578). Проверено 13.09.2021

2. Википедия. Java: [Электронный документ]. (https://ru.wikipedia.org/wiki/Java). Проверено 13.09.2021

3. Википедия. Lightweight Java Game Library: [Электронный документ]. (https://ru.wikipedia.org/wiki/Lightweight\_Java\_Game\_Library). Проверено 13.09.2021

4. Википедия. IntelliJ IDEA: [Электронный документ]. (https://ru.wikipedia.org/wiki/IntelliJ\_IDEA). Проверено 13.09.2021

5. Википедия. OpenGL: [Электронный документ]. (https://ru.wikipedia.org/wiki/OpenGL). Проверено 13.09.2021

6. Википедия. Snake (игра): [Электронный документ]. (https://ru.wikipedia.org/wiki/Snake\_(%D0%B8%D0%B3%D1%80%D0%B0)). Проверено 13.09.2021