Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

|  |
| --- |
| Институт информационных технологий и анализа данных |

наименование института

|  |
| --- |
| Допускаю к защите  Руководитель:  З.А.Бахвалова |
| И.О. Фамилия |

Разработка приложения с использованием   
объектно-ориентированного подхода

наименование темы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине

|  |
| --- |
| Объектно-ориентированное программирование |

|  |
| --- |
| 1.004.00.00 ПЗ |
| обозначение документа |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | ИСМб 19-1 |  |  |  | Е.Д.Бондаренко |
|  | шифр группы |  | подпись |  | Фамилия И.О |
| Нормоконтроль |  |  |  |  | З.А.Бахвалова |
|  |  |  | подпись |  | Фамилия И.О |

|  |  |
| --- | --- |
| Курсовой проект защищена с оценкой |  |

Иркутск 2021 г.

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**ЗАДАНИЕ**

**НА КУРСОВОЙ ПРОЕКТ**

|  |  |  |
| --- | --- | --- |
| По курсу | | Объектно-ориентированное программирование |
| Студенту | | Бондаренко Екатерине Денисовне |
|  | | (фамилия, инициалы) |
| Тема проекта: | | Разработка приложения с использованием объектно-ориентированного подхода | |
| Исходные данные: | Разработка игры "Змейка" с использованием технологии JavaFX | | |
| Рекомендуемая литература: | | | |
| 1. Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. Объектно-ориентированны анализ и проектирование с примерами приложений. Третье издание. М.: "Вильямс", 2010, -720 с.  2. Васильев А. Н. Java. Объектно-ориентированное программирование: для магистров и бакалавров. Базовый курс по объектно-ориентированному программированию / А. Н. Васильев . – СПб.: Питер, 2012. – 395 с. | | | |
| 3. Объектно-ориентированное программирование. Методические указания по курсовому проекту. Составил В.Л. Аршинский. – Иркутск: издво ИРНИТУ, 2018. – 13 c. | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Графическая часть на | 47 | | листах. | |
| Дата выдачи задания |  | 15 / 02 / 2021 г. | | |
| Задание получил студент | | | |  |  | Е.Д.Бондаренко |
|  | | | | подпись |  | Фамилия И.О. |

|  |  |
| --- | --- |
| Дата представления проекта руководителю | 27 / 05 / 2021 г. |
| Руководитель курсовой проекта |  |  | З.А.Бахвалова |
|  | подпись | Фамилия И.О. |

Содержание

[Введение 4](#_Toc74076106)

[1 Перечень условных обозначений и терминов 5](#_Toc74076107)

[2 Постановка задач 6](#_Toc74076108)

[3 Проектирование приложения 7](#_Toc74076109)

[3.1 Проектирование логической модели базы данных 7](#_Toc74076110)

[3.2 Проектирование графического интерфейса 8](#_Toc74076111)

[3.3 Реализация клиентской части приложения 11](#_Toc74076112)

[3.6 Класс Main 13](#_Toc74076113)

[3.7 Класс MenuController 13](#_Toc74076114)

[3.8 Класс RezultController 14](#_Toc74076115)

[3.9 Класс RecController 14](#_Toc74076116)

[3.10 Класс ToPlay 15](#_Toc74076117)

[3.11 Класс Snake 16](#_Toc74076118)

[3.12 Класс Food 17](#_Toc74076119)

[3.13 Класс BGround 17](#_Toc74076120)

[3.14 Класс BD 18](#_Toc74076121)

[3.15 Класс User 18](#_Toc74076122)

[4 Таблицы тестов 20](#_Toc74076123)

[5 Тестирование программы 22](#_Toc74076124)

[5.1 Результат тестирования интерфейса 22](#_Toc74076125)

[5.2 Результат тестирования ожидаемых ошибок 27](#_Toc74076126)

[Заключение 32](#_Toc74076127)

[Список литературы 33](#_Toc74076128)

[Приложение А. Исходный код 34](#_Toc74076129)

Введение

История игры «Змейка» началась за несколько лет до появления первых мобильных телефонов. В 1977 году компания Gremlin Industries выпустила игровой автомат Hustle, рассчитанный на одного или двух игроков, в которой нужно было управлять «змейками», направляя их на бессистемно появляющиеся цели. Для победы нужно было заполучить больше очков, чем у оппонента, преграждая по ходу игры ему путь к новым целям (в случае многопользовательской игры), или просто побить установленный на игровом автомате рекорд.

Оригинальная «Змейка» (Snake) от Nokia появилась в 1997 году благодаря стараниями разработчика Танели Орманто. В том же году компания выпустила первый телефон с этой игрой — Nokia 6110. Сама змейка состояла из чёрных квадратов и могла двигаться в четырёх направлениях. Игровая зона была ограничена размерами экрана телефона: при ударе головы змейки о край телефона игра завершалась. «Змейка» приобрела невероятную популярность, сравнимую разве что с популярностью современных хитов «Angry Birds» и «Cut the Rope».

Игрок управляет длинным, тонким существом, напоминающим змею (как правило изначально она состоит из 3-4 звеньев), и может двигаться в четырех направлениях. Игровая зона ограничена размерами экрана приложения: при ударе головы змейки о край экрана игра завершается. Также игра завершается при ударе головой змеи о собственное тело. Задачей змейки является съесть как можно больше фруктов, избегая столкновения. Каждый раз, когда змея съедает пищу, она становится длиннее, что постепенно усложняет игру.

Создание игрового приложения, может послужить источником опыта для оттачивания объектно-ориентированного подхода к программированию.

**Цель курсовой работы**: закрепить навыки объектно-ориентированного программирования, для этого создать игровое приложение "Змейка", для развлечения людей, развития реакции, скорости принятия решений, стратегического мышления у пользователей игры.

1 Перечень условных обозначений и терминов

Ниже представлен словарь терминов предметной области.

Таблица 1.1 – Словарь терминов

|  |  |  |
| --- | --- | --- |
| № п/п | Термин | Назначение |
|  | Пользователь | Человек, управляющий змейкой |
|  | Имя | Индивидуальное имя пользователя |
|  | Змейка | Участник игры |
|  | Еда / фрукты | Цель достижения змейки |
|  | Съесть | Достижение головы змейки фрукта (голова змейки и фрукт имеют одинаковые координаты) |
|  | Рисовать | Изображение предметов (змейки, еды, заднего фона) |
|  | Удар/столкновение | Голова змейки имеет такие же координаты что границы экрана или сегмент тела змейки |
|  | Ход | Смена направления движения змейки в результате нажатия на клавишу клавиатуры |

2 Постановка задач

Игрок управляет длинным, тонким существом, напоминающим змею (изначально она состоит из 3 звеньев), и может двигаться в четырех направлениях. Змейкой можно управлять при помощи клавиатуры, а именно: стрелка вверх - направление вверх, стрелка вниз - направление вниз, стрелка влево - направление влево, стрелка вправо - направление вправо.

Игровая зона ограничена размерами экрана приложения: при ударе головы змейки о край экрана игра завершается. Также игра завершается при ударе головой змеи о собственное тело. Задачей змейки является съесть как можно больше фруктов, избегая столкновения. Еда для змейки генерируется каждый раз после съедения в произвольной части экрана. Каждый раз, когда змея съедает пищу, она становится длиннее, что постепенно усложняет игру.

Количество очков за съедание одного фрукта зависит от количества сделанных ходов. Максимальное количество баллов получает игрок если сделает от 0 до 2 ходов (минимальное возможное число ходов для достижения фрукта из любого положения змейки). Если игрок до достижения фрукта сделает более двух ходов, то за каждый штрафной ход от максимального количества отнимается по одному баллу.

3 Проектирование приложения

3.1 Проектирование логической модели базы данных

База данных будет включать в себя одну таблицу:

* Пользователь (имя, счет, уровень)

Объект “Пользователь” хранит данные о зарегистрированных пользователях приложения.

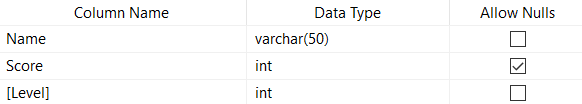


Рисунок 1 – Структура данных

3.2 Проектирование графического интерфейса

Приложение состоит из 4 окон. Каждое из них соединено минимум с двумя другими. Из окна “Меню” можно попасть в окна “Игра” и “Таблица рекордов”. Из окна “Таблица рекордов” можно попасть обратно в окно “Меню” или в окно “Завершение игры”, в зависимости от того из какого окна было вызвано окно “Таблица рекордов”. Из окна “Игра”, в результате совершения события, приводящего к завершению игры, можно попасть в окно “Завершение игры”. Из него можно попасть в окно “Таблица рекордов” или выйти из игрового приложения.

Меню

Надпись «Snake»

Кнопка «Уровень сложности»

RadioButton «Легкий»

RadioButton «Сложный»

Кнопка «Начать игру»

Кнопка «Рекорды»

Картинка

Нажатие на кнопку «Начать игру»

Нажатие на кнопку «Рекорды»

Игра

Старт игры

Клик манипулятором

Клик манипулятором

Клик манипулятором

Таблица рекордов

Надпись «Records»

Таблица TableRec

Колонка «Имя»

Колонка «Счет»

Колонка «Уровень»

Сортировать по Уровням

Сортировать по баллам

Кнопка «Сортировать»

Кнопка «Назад»

Нажатие на кнопку «Сортировать»

Нажатие на кнопку “назад”

Завершение игры

Надпись «Game over»

Поле для ввода имени

Кнопка «Сохранить результат»

Кнопка «Узнай рекорды других»

Кнопка «Выйти»»

Поле «Количество очков»

Поле «Место»

Надпись (Информационное сообщение)

Нажатие на кнопку «Сохранить результат»

Нажатие на кнопку «Узнай рекорды других»

Нажатие на кнопку “Выйти”

Клик манипулятором

Рисунок 2 – Диаграмма интерфейсных классов

Приложение стартует с окна “Меню”. В нем присутствует: надпись “Snake” (название игры); кнопки: “Уровень сложности” (делает видимыми RadioButton: “Легкий” и “Сложный”), “Начать игру” (запускает игру) и “Рекорды” (открывает окно с таблицей рекордов); два RadioButton: “Легкий” и “Сложный”, позволяющие выбрать уровень сложности игры; а также картинка.



Рисунок 3 – Макет интерфейса главного окна

Окно «Таблица рекордов» содержит: надпись с названием окна; таблицу со столбцами: “Имя”, “Счет”, “Уровень” (содержит сведения о игроках и результатах их игры); два RadioButton: “По уровням” и “По баллам", предназначенные для выбора вида сортировки; кнопки: “Сортировать” (упорядочивает данные в таблице рекордов по указанному виду сортировки) и “Назад” (выполняет возврат в главное меню или окно завершения игры, в зависимости от куда было вызвано окно таблицы рекордов); а также картинка.

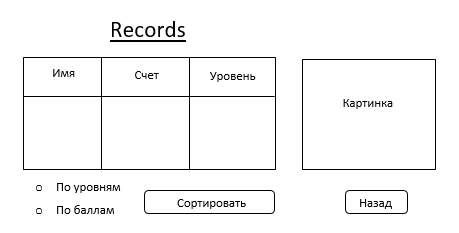


Рисунок 4 – Макет интерфейса окна рекордов

Окно “Результат” содержит: надпись с названием окна; поле для вывода количества очков; поле для вывода места, которое занял пользователь в результате игры; поле для ввода имени пользователя; скрытое поле для вывода информационного сообщения; три кнопки “Сохранить результат” (записывает данные пользователя в таблицу рекордов), “Узнай результаты других” (осуществляет переход в окно “Таблица рекордов”) и “Выход” (осуществляется выход из игрового приложения); а также картинка.



Рисунок 5 – Макет интерфейса окна завершения игры

3.3 Реализация клиентской части приложения

Клиентская часть приложения состоит из 3 окон (menu.fxml, TableRec.fxml, Rezult.fxml). В целях поддержания структуры и логики приложения классы расположены в соответствующих пакетах.

Menu – пакет, содержащий разметку и графический интерфейс для главного окна (окна меню).

Records – пакет, содержащий разметку и графический интерфейс для окна с таблицей рекордов.

rezult– пакет, содержащий разметку и графический интерфейс для окна завершение игры.

Play – пакет, содержащий графический интерфейс и игровую логику.

Most – пакет, содержащий класс для работы с базой данных BD, и класс User.

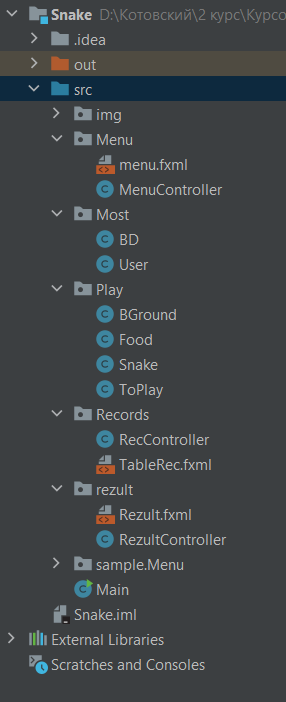


Рисунок 6 – Пакетная архитектура проекта

3.4 Реализация классов



Рисунок 7 – UML – диаграмма классов

3.5 Словесное описании классов

Игровое приложение состоит из 10 классов. 1 группа – классы сгруппированы по со своей графической разметкой в соответствующие пакеты. 2 группа – images хранит в себе все изображения еды. 3 группа – классы базы данных. 4 группа – логика игры.

Описание классов:

1. Main – точка входа в программу, запуск графического интерфейса
2. MenuController – контроллер окна “Меню”
3. RezultController – контроллер окна “Завершение игры”
4. RecController – контроллер окна “Таблица рекордов”
5. ToPlay – контроллер окна игры
6. Snake – логика игры, змейка
7. Food – логика игры, еда
8. BGround – логика игры, задний фон
9. BD – класс для работы с базой данных
10. User – класс для хранения информации о пользователях

3.6 Класс Main

Таблица 3.1 – Таблица методов класса Main

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| main | void | + | public | String[] args | Точка входа в программу, вызов метода launch(args) |
| start | void | - | public | Stage primaryStage | Вызов оконного интерфейса |

3.7 Класс MenuController

Поля класса:

1. private Button btnLevel – кнопка выбора уровня игры
2. private Button btnStart – кнопка для запуска игры
3. private Button btnRec – кнопка перехода к окну таблицы рекордов
4. private ToPlay game – вызов объекта класса
5. private ToggleGroup R1 – группа для RadioButton
6. private RadioButton easy – выбор легкого уровня игры
7. private RadioButton difficult – выбор сложного уровня игры

Таблица 3.2 – Таблица методов класса MenuController

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| initialize | void | - | public | URL url, ResourceBundle resourceBundle | Инициализация обработчиков ивентов нажатия кнопок в окне |
| Levels | void | - | public | - | метод для появления скрытых Radiobutton, отвечающих за выбор уровня сложности игры |

3.8 Класс RezultController

Поля класса:

1. private TextField NameText – поле ввода имени
2. private Button SaveRez – кнопка сохранения данных об игроке
3. private Button btnRec – кнопка перехода к таблице рекордов
4. private Button Exit – кнопка выхода из игрового приложения
5. private Label ScoreRez – поле для вывода счета игры
6. private Label lablerez – поле для вывода информационного сообщения
7. private Label Mesto – поле для вывода информационного сообщения

Таблица 3.3 – Таблица методов класса RezultController

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| initialize | void | - | public | URL url, ResourceBundle resourceBundle | Инициализация обработчиков ивентов нажатия кнопок в окне |
| mesto | int | - | private | int score, int level | Метод для подсчета места в зависимости от уровня сложности |

3.9 Класс RecController

Поля класса:

1. ObservableList personsData – поле с информацией о правильности введенных данных
2. private TableView<User> TableRec – таблица игроков
3. private TableColumn<User, String> ColNicName – колонка Имя – имя пользователя
4. private TableColumn<User, Integer> ColScore – колонка Счет – количество баллов
5. private TableColumn<User, Integer> ColLevel – колонка Уровень – уровень сложности
6. private Button back – кнопка назад
7. private Button Sort – кнопка сортировки
8. private ToggleGroup R – группа для RadioButton
9. private RadioButton levelSort – выбор вида сортировки (сортировка по уровню сложности)
10. private RadioButton ScoreSort – выбор вида сортировки (сортировка по количеству баллов)
11. private static String BackWindow – выбор окна для возвращения назад
12. private Label Linfo – надпись (информационное сообщение)

Таблица 3.4 – Таблица методов класса RecController

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| initialize | void | - | public | URL url, ResourceBundle resourceBundle | Инициализация обработчиков эвентов нажатия кнопок в окне |
| BackId | void | + | public | int idBack | Указывает из какого окна было вызвано окно “Таблица рекордов” |

3.10 Класс ToPlay

Поля класса:

1. int WIDTH – поле ширины игрового поля
2. int HEIGHT – поле высоты игрового поля
3. int ROWS – количество строк, на которое разбивается игровое поле
4. int COLOMNS – количество колонок, на которое разбивается игровое поле
5. int SQUARE\_SIZE – размер одной клетки игрового поля
6. int number\_moves – поле счета сделанных ходов
7. public static int score – поле счета баллов
8. boolean gameOver – поле значение проигрыша
9. private static final int RIGHT – поле направление вправо
10. private static final int LEFT – поле направление влево
11. private static final int UP – поле направление вверх
12. private static final int DOWN – поле направление вниз
13. private GraphicsContext gc – графика
14. Snake snake – объект класса змея
15. Food food – объект класса еда
16. Timeline timeline – объект класса таймер
17. public static int idLevel – поле номер уровня

Таблица 3.5 – Таблица методов класса ToPlay

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| start | void | - | pablic | int id | Метод запуска игрового окна |
| run | void | - | private | GraphicsContext gc | Метод старта игры |
| moveRight | void | - | private | - | Изменение направления Поворот вправо |
| moveLeft | void | - | private | - | Изменение направления  Поворот влево |
| moveUp | void | - | private | - | Изменение направления  Поворот вверх |
| moveDown | void | - | private | - | Изменение направления  Поворот вниз |
| gameOver | void | - | public | - | Метод для проверки врезалась ли змейка или нет |
| eatFood | void | - | private | - | Метод для подсчета очков |
| drawScore | void | - | private | - | Метод для вывода очков на экран |
| drawNumber\_moves | void | - | private | - | Метод для вывода количества сделанных ходов |

3.11 Класс Snake

Поля класса:

1. private int ROWS – количество строк, на которое разбивается игровое поле
2. private int SQUARE\_SIZE– размер одной клетки игрового поля
3. Point snakehead – поле голова змеи
4. public static List<Point> snakeBody – массив сегментов тела змеи

Конструктор класса:

1. public Snake(int SQUARE\_SIZE, int ROWS)

Таблица 3.6 – Таблица методов класса Snake

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| startBody | void | - | public | - | Появление змейки при старте программы и установка начального местоположения |
| drawSnake | void | - | public | GraphicsContext gc | Прорисовка элементов змейки |
| getSnakeBody | List<Point> | + | public | - | Возврат массива snakeBody |

3.12 Класс Food

Поля класса:

1. private String [] FOODS\_IMG – массив картинок еды
2. private Image foodImage – картинка еды
3. int foodX – координата X расположения еды
4. int foodY – координата Y расположения еды
5. private int ROWS – количество строк, на которое разбивается игровое поле
6. private int COLOMNS – количество колонок, на которое разбивается игровое поле
7. private int SQUARE\_SIZE – размер одной клетки игрового поля

Конструктор класса:

1. public Food(int SQUARE\_SIZE,int ROWS, int COLUMNS)

Таблица 3.7 – Таблица методов класса Food

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| drawFood | void | - | public | GraphicsContext gc | Прорисовка еды |
| generateFood | void | - | public | - | Обеспечивает появление еды в произвольной части игрового поля |

3.13 Класс BGround

Поля класса:

1. private int ROWS – количество строк, на которое разбивается игровое поле
2. private int COLUMNS – количество колонок, на которое разбивается игровое поле
3. private int SQUARE\_SIZE – размер одной клетки игрового поля

Конструктор класса:

1. public BGround(int SQUARE\_SIZE,int ROWS,int COLUMNS)

Таблица 3.8 – Таблица методов класса Universe

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| drawBackground | void | - | public | GraphicsContext gc | Прорисовка игрового пространства |

3.14 Класс BD

Поля класса:

1. private static Connection connection – подключение базы данных
2. private static final ObservableList<User> persons – список игроков

Таблица 3.9 – Таблица методов класса BD

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| Connector | void | + | public | - | Подключение базы данных |
| addItem | void | + | public | User person | Добавление данных об игроке и результатах его игры в базу данных |
| getPersons | ObservableList | + | public | - | Получение списка игроков |
| read | void | + | public | - | Метод считывания данных из базы данных |
| sort | void | + | public | String par | Метод сортировки данных, полученных из базы данных |

3.15 Класс User

Поля класса:

1. private String name – поле имени пользователя
2. private int score – поле счета баллов
3. private int level – поле уровня сложности

Конструкторы класса:

1. public Users(String name, int score, int level)

Таблица 3.10 – Таблица методов класса User

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| getName | String | - | public | - | Возврат значения поля name |
| getScore | int | - | public | - | Возврат значения поля score |
| getLevel | int | - | public | - | Возврат значения поля level |

4 Таблицы тестов

Таблица 4.1 – Тестирование интерфейса

|  |  |  |
| --- | --- | --- |
| № | Назначение теста | Ожидаемый результат |
| 1 | Запуск программы | Открытие главного окна |
| 2 | Нажатие на кнопку “Уровень сложности” | Появление скрытых RadioButton: “Легкий” и “Сложный” |
| 3 | Нажатие кнопки “Начать игру” с выбранным уровнем сложности | Запуск игры |
| 4 | Нажатие кнопки “Рекорды” | Открывается окно с таблицей участников, пожелавших сохранить свой результат |
| 5 | Нажатие кнопки “Назад” в окне “Таблица рекордов” (Вызов окна “Таблица рекордов” из окна “Меню”) | Возврат в окно “Меню” |
| 6 | Открытие окна “Завершение игры” после проигрыша | Открытие окна “Завершение игры” |
| 7 | Нажатие кнопки “Узнай рекорды других” | Открытие окна с таблицей рекордов |
| 8 | Нажатие кнопки “Назад” в окне “Таблица рекордов” (Вызов окна “Таблица рекордов” из окна “Завершение игры”) | Возврат в окно “Завершение игры” |
| 9 | Нажатие кнопки “Выход” | Выход из игрового приложения |

Таблица 4.2 – Таблица эквивалентности

|  |  |  |  |
| --- | --- | --- | --- |
| № | Описание класса эквивалентности | Классы корректных данных | Классы некорректных данных |
| 1 | Name | Любой символ, количество символов меньше 50 | Ничего, количество символов больше 50 |

Таблица 4.3 – Тестирование ожидаемых ошибок

|  |  |  |
| --- | --- | --- |
| № | Назначение теста | Ожидаемый результат |
| 1 | Конец игры | Появление окна “Завершение игры” с выводом количества набранных баллов и занятым местом |
| 2 | Нажатие кнопки “Сохранить результат”, введено имя игрока | Вывод сообщения на экран "Результаты добавлены" |
| 3 | Нажатие кнопки “Сохранить результат”, не введено имя игрока | Вывод сообщения “Укажите ваше имя” |

Продолжение таблицы 4.3

|  |  |  |
| --- | --- | --- |
| 4 | Нажатие кнопки “Сортировать”, когда не выбран критерий сортировки | Вывод сообщения “Выберете критерий сортировки” |
| 5 | Сортировать по уровням | Отсортированная таблица по уровням сложности |
| 6 | Сортировать по баллам | Отсортированная таблица по баллам |

5 Тестирование программы

5.1 Результат тестирования интерфейса

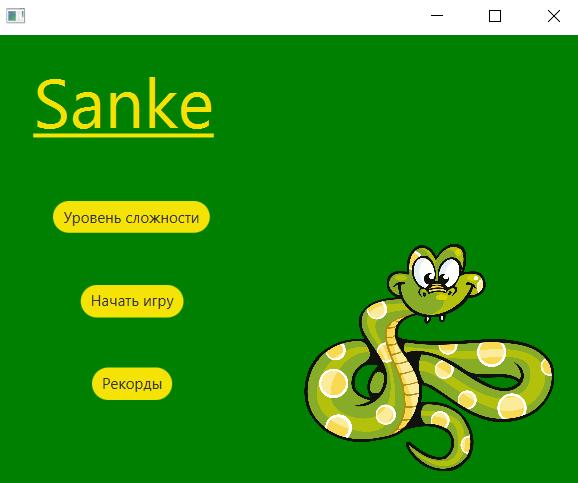


Рисунок 8 – Результат теста №1 (успешно)



Рисунок 9 – Результат теста №2 (успешно)

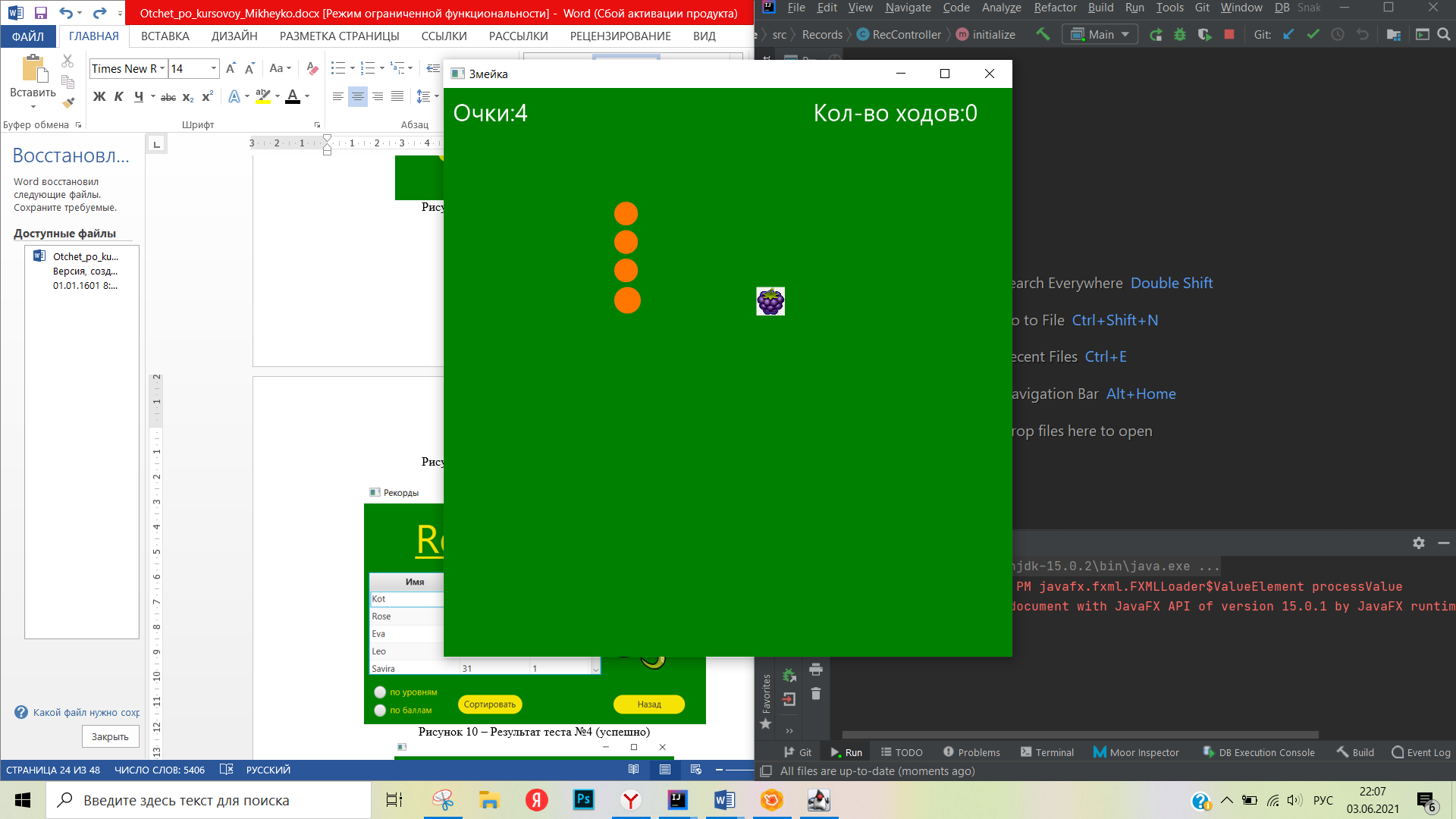


Рисунок 10 – Результат теста №3 (успешно)



Рисунок 11 – Результат теста №4 (успешно)

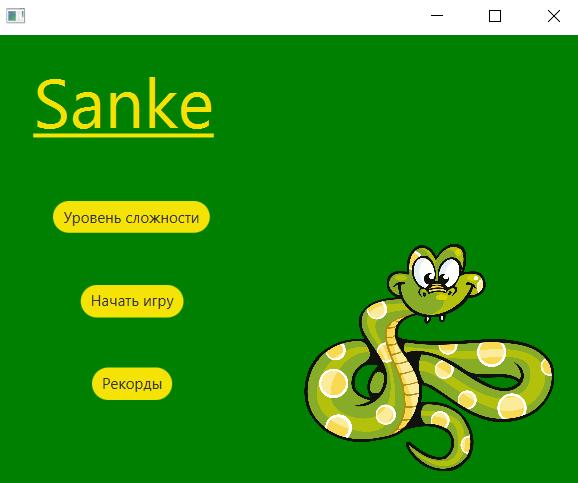


Рисунок 12 – Результат теста №5 (успешно)



Рисунок 13 – Результат теста №6 (успешно)



Рисунок 14 – Результат теста №7 (успешно)



Рисунок 15 – Результат теста №8 (успешно)

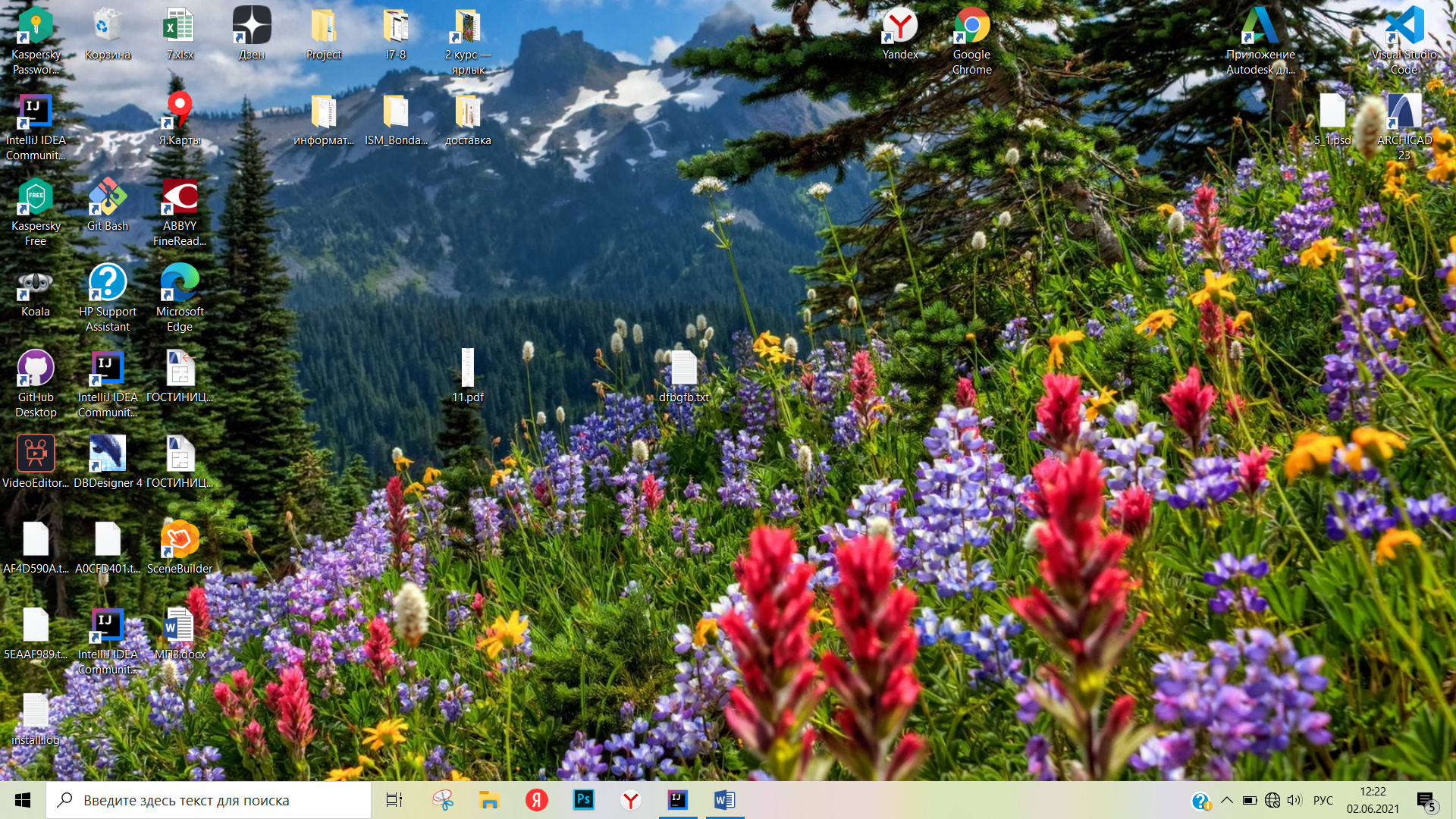


Рисунок 16 – Результат теста №9 (успешно)

5.2 Результат тестирования ожидаемых ошибок



Рисунок 17 – Результат теста №1 (успешно)



Рисунок 18 – Результат теста №2 (успешно)



Рисунок 19 – Результат теста №3 (успешно)



Рисунок 20 – Результат теста №4 (успешно)



Рисунок 21 – Результат теста №5 (успешно)



Рисунок 22 – Результат теста №6 (успешно)

Заключение

В ходе выполнения курсового проекта были получены практические навыки работы с объектно-ориентированным программированием на языке Java. Также были получен навыки работы с библиотеками JavaFX. Были получены навыки разработки игровых приложений. Получены практические навыки проектирования и разработки GUI. Были получены навыки работы с базами данных с использованием языка Java. Также были получены навыки построения UML-диаграмм, диаграмм интерфейсных классов. Усовершенствованы навыки построения тестов с помощью классов эквивалентности. Были получены навыки применения языка Java для решения поставленных задач.

Для подключения базы данных был использован Microsoft SQL Server Management Studio 18, была создана одна таблица, в нее входит информация о пользователях (имя, баллы и уровень сложности).

В ходе выполнения проекта было разработано 15 тестов: 9 тестов интерфейса и 6 тестов на ожидаемые ошибки, которые покрывают большинство непредвиденных ситуаций. Приложение успешно прошло все 15 тестов.

Плюсом данного проекта являются, что был применен подход создания приложения с разделением графической части и логической. Также плюсом является графический интерфейс, написанный с помощью языка разметки fxml и приложения SceneBuilder.

Цель курсовой работы была достигнута, поставленные задачи были выполнены.

Список литературы

1. Java. Экспресс-курс [электронный ресурс] // Сайт Александра Климова [сайт], URL: http://developer.alexanderklimov.ru/android/java/java.php (дата обращения: 11.01.2018)

2. API Specification for the Java 7 SE. [официальный сайт] URL: http://docs.oracle.com/javase/7/docs/api/ (дата обращения: 11.01.2018)

3. Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. Объектно-ориентированны анализ и проектирование с примерами приложений. Третье издание. М.: "Вильямс", 2010, -720 с.

4. Хабибуллин И.Ш. Java 7: для программистов / И. Ш. Хабибуллин. – Санкт-Петербург: БХВ–Петербург, 2014. – 768 с.

5. Васильев А. Н. Java. Объектно-ориентированное программирование: для магистров и бакалавров. Базовый курс по объектно-ориентированному программированию / А. Н. Васильев. – СПб.: Питер, 2012. – 395 с.

6. Отношения классов — от UML к коду [Электронный ресурс] // Хабрахабр: [сайт]. [2006]. URL: http://habrahabr.ru/post/150041/ (дата обращения: 11.01.2018)

7. Рамбо Дж. UML 2.0. Объектно-ориентированное моделирование и разработка / Дж. Рамбо, М. Блаха, 2007. - 540 с.

8. Эванс Б. Java. Новое поколение разработки: Техники Java 7 и многоязычное программирование: переводное издание / Б. Эванс, М. Вербург, 2014. - 556 с.

9. Эккель Б. Философия Java: монография / Б. Эккель, 2015. - 1165 с.

10. Хабибуллин И. Ш. Java 7: [наиболее полное руководство]: для программистов / И. Ш. Хабибуллин, 2014. - 768 с.

Приложение А. Исходный код

1 Класс Main

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

public class Main extends Application {

@Override

//Вызов оконного интерфейса (Вызов окна "Меню")

public void start(Stage primaryStage) throws Exception {

Parent root = FXMLLoader.load(getClass().getResource("Menu/menu.fxml"));

primaryStage.setScene(new Scene(root, 469, 400));

primaryStage.show();

}

//Точка входа в программу, вызов метода launch(args)

public static void main(String[] args) {

launch(args);

}

}

2 Класс MenuController

package Menu;

import Most.BD;

import Play.ToPlay;

import Records.RecController;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.stage.Modality;

import javafx.stage.Stage;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

//Контроллер для окна "Меню" реализует кнопки выбора уровня игры, начало игры, перехода к таблице рекордов

public class MenuController implements Initializable {

@FXML

private Button btnLevel;

@FXML

private Button btnStart;

@FXML

private Button btnRec;

@FXML

private ToggleGroup R1;

@FXML

private RadioButton easy;

@FXML

private RadioButton difficult;

ToPlay game;

//метод для появления скрытых Radiobutton, отвечающих за выбор уровня сложности игры, при нажатии на кнопку "Уровень сложности"

public void Levels(javafx.event.ActionEvent event) {

easy.setVisible(true);

difficult.setVisible(true);

}

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

//Кнопка "Начать игру". Запуск игры

btnStart.setOnAction(event -> {

game = new ToPlay();

if (this.R1.getSelectedToggle().equals(this.easy)) {

game.start(1);

}else game.start(2);

event.consume();

Stage stage = (Stage) btnStart.getScene().getWindow();

stage.close();

});

//Кнопка "рекорды". Переход в окно "Таблица рекордов"

btnRec.setOnAction((event -> {

BD.Connector();

Stage stage = (Stage) btnRec.getScene().getWindow();

stage.close();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("/Records/TableRec.fxml"));

RecController.BackId(1);

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

stage = new Stage();

stage.setScene(new Scene(root));

stage.setTitle("Рекорды");

stage.setResizable(false);

stage.show();

}));

}

}

3 Класс RezultController

package rezult;

import Most.BD;

import Most.User;

import Play.ToPlay;

import Records.RecController;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.stage.Modality;

import javafx.stage.Stage;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

public class RezultController implements Initializable {

@FXML

private Button btnRec;

@FXML

private Button Exit;

@FXML

private Button SaveRez;

@FXML

private Label ScoreRez;

@FXML

private Label lablerez;

@FXML

private Label Mesto;

@FXML

private TextField NameText;

ObservableList personsData = FXCollections.observableArrayList();

//btnRec

private int mesto(int score, int level){

personsData = BD.getPersons();

int s1=0;

int s2=0;

int s;

for (var person : personsData) {

if (((User) person).getLevel()==1){

s1=s1+1;

}else s2=s2+1;

}

if(level==1){

for (var person : personsData) {

if ((((User) person).getLevel()==1)&&(score>=((User) person).getScore())){

s1=s1-1;

}

}

s=s1;

}else {

for (var person : personsData) {

if ((((User) person).getLevel()==2)&&(score>=((User) person).getScore())){

s2=s2-1;

}

}

s=s2;

}

return (s+1);

}

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

ScoreRez.setText("Очков:"+String.valueOf(ToPlay.score));

Mesto.setText("Место:"+mesto(ToPlay.score,ToPlay.idLevel));

//Кнопка "Сохранить результат". Записывает данные о пользователе (Имя, количество баллов, уровень игры) базу данных

SaveRez.setOnAction(event -> {

String name = NameText.getText();

int score = ToPlay.score;

int level = ToPlay.idLevel;

if (name!=""){

BD.Connector();

BD.addItem(new User(name, score, level));

lablerez.setText("Результаты добавлены!");}

else{

lablerez.setText("Укажите ваше имя!");

}

});

// Кнопка "Узнай результаты других". Переход в окно "Таблица рекордов"

btnRec.setOnAction((event -> {

BD.Connector();

Stage stage = (Stage) btnRec.getScene().getWindow();

stage.close();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("/Records/TableRec.fxml"));

RecController.BackId(2);

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

stage = new Stage();

stage.setScene(new Scene(root));

stage.setTitle("Рекорды");

stage.setResizable(false);

stage.show();

}));

//Кнопка "Выход". Осуществляет выход из игрового приложения

Exit.setOnAction((event -> {

Runtime.getRuntime().exit(0);

}));

}

}

4 Класс RecController

package Records;

import Most.BD;

import Most.User;

import Play.ToPlay;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

// Контроллер окна "Таблица рекордов"

public class RecController implements Initializable {

private static String BackWindow;

ObservableList personsData = FXCollections.observableArrayList();

@FXML

private TableView<User> TableRec;

@FXML

private TableColumn<User, String> ColNicName;

@FXML

private TableColumn<User, Integer> ColScore;

@FXML

private TableColumn<User, Integer> ColLevel;

@FXML

private ToggleGroup R;

@FXML

private RadioButton levelSort;

@FXML

private RadioButton ScoreSort;

@FXML

private Button Sort;

@FXML

private Button back;

@FXML

private Label LInfo;

public static void BackId(int idBack){

if (idBack==1){

BackWindow = "/Menu/menu.fxml";

} else BackWindow = "/rezult/Rezult.fxml";

}

@Override

//Метод обработки нажатия кнопок и записи данных в таблицу

public void initialize(URL url, ResourceBundle resourceBundle) {

personsData = BD.getPersons();

//заполнение таблицы на форме

ColNicName.setCellValueFactory(new PropertyValueFactory<>("name"));

ColScore.setCellValueFactory(new PropertyValueFactory<>("score"));

ColLevel.setCellValueFactory(new PropertyValueFactory<>("level"));

TableRec.setItems(personsData);

//Кнапка "Сортировать". Осуществление сортировки данныхв зависимости от выбранного вида сортировки (по уровню сложности, по кол-ву баллов)

Sort.setOnAction(event2 -> {

if (this.R.getSelectedToggle()==null) {

LInfo.setVisible(true);

}else {

LInfo.setVisible(false);

if (this.R.getSelectedToggle().equals(this.levelSort)) {

BD.sort("Level");//по уровню сложности

}else

BD.sort("Score");//по кол-ву баллам}

}});

// Кнопка "Назад". Осуществляет возврат к окну из которого было вызвано окно "Таблица рекордов"

back.setOnAction((event -> {

Stage stage = (Stage) back.getScene().getWindow();

stage.close();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource(BackWindow));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

stage = new Stage();

stage.setScene(new Scene(root));

stage.setTitle("Рекорды");

stage.setResizable(false);

stage.show();

}));

}

}

5 Класс ToPlay

package Play;

import javafx.animation.Animation;

import javafx.animation.KeyFrame;

import javafx.animation.Timeline;

import javafx.event.EventHandler;

import javafx.fxml.FXMLLoader;

import javafx.scene.Group;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.canvas.Canvas;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.input.KeyCode;

import javafx.scene.input.KeyEvent;

import javafx.scene.layout.Background;

import javafx.scene.paint.Color;

import javafx.scene.text.Font;

import javafx.stage.Stage;

import javafx.stage.StageStyle;

import javafx.util.Duration;

import javax.swing.\*;

import java.awt.\*;

import java.io.IOException;

import static java.lang.String.valueOf;

public class ToPlay {

public static int score=0;

public static int idLevel=0;

private int WIDTH = 600;

private int HEIGHT = WIDTH;

private int ROWS = 20;

private int COLUMNS = ROWS;

private int SQUARE\_SIZE = WIDTH / ROWS;

private int number\_moves =0;

private static final int RIGHT = 0;

private static final int LEFT = 1;

private static final int UP = 2;

private static final int DOWN = 3;

private GraphicsContext gc;

private boolean gameOver;

private int oper;

Snake snake;

Food food;

Timeline timeline;

//Запуск игрового окна. Установка уровня сожности

public void start(int id) {

Stage primaryStage = new Stage();

Group root = new Group();

Canvas canvas = new Canvas(WIDTH, HEIGHT);

root.getChildren().add(canvas);

Scene scene = new Scene(root);

primaryStage.setScene(scene);

primaryStage.initStyle(StageStyle.UNDECORATED);

primaryStage.show();

gc = canvas.getGraphicsContext2D();

scene.setOnKeyPressed(new EventHandler<KeyEvent>() { //обработчик нажатия кнопок

@Override

public void handle(KeyEvent event) {

KeyCode code = event.getCode();

if (code == KeyCode.RIGHT) {

if (oper != LEFT) {

oper = RIGHT;

}

number\_moves=number\_moves+1;

} else if (code == KeyCode.LEFT) {

if (oper != RIGHT) {

oper = LEFT;

}

number\_moves=number\_moves+1;

} else if (code == KeyCode.UP) {

if (oper != DOWN) {

oper = UP;

}

number\_moves=number\_moves+1;

} else if (code == KeyCode.DOWN) {

if (oper != UP) {

oper = DOWN;

}

number\_moves=number\_moves+1;

}

}

});

snake = new Snake(SQUARE\_SIZE,ROWS);

food = new Food(SQUARE\_SIZE,ROWS,COLUMNS);

snake.startBody();

food.generateFood();

int speed = 0;

switch (id){

case 1: {

idLevel=1;

speed = 300;

break;

}

case 2:{

idLevel=2;

speed = 150;

break;

}

}

timeline = new Timeline((new KeyFrame(Duration.millis(speed), e -> run(gc))));

timeline.setCycleCount(Animation.INDEFINITE);

timeline.play();

}

//Старт игры

private void run(GraphicsContext gc) {

// при проигрыше открывается окно "Pfdthitybz buhs"

if (gameOver) {

timeline.stop();

Stage primaryStage = new Stage();

Parent root = null;

try {

root = FXMLLoader.load(getClass().getResource("/rezult/Rezult.fxml"));

} catch (IOException ioException) {

ioException.printStackTrace();

}

primaryStage.setScene(new Scene(root, 370, 370));

primaryStage.initStyle(StageStyle.UNDECORATED);

primaryStage.show();

}

BGround b = new BGround(SQUARE\_SIZE,ROWS,COLUMNS);

b.drawBackground(gc);

food.drawFood(gc);

snake.drawSnake(gc);

switch (oper) {

case RIGHT:{

moveRight();

break;}

case LEFT:{

moveLeft();

break;}

case UP:{

moveUp();

break;}

case DOWN:{

moveDown();

break;}

}

drawScore();

drawNumber\_moves();

eatFood();

gameOver();

}

//Поворот головы змеи вправо

private void moveRight() {

snake.snakeHead.x++;

number\_moves=number\_moves++;

}

//Поворот головы змеи влево

private void moveLeft() {

snake.snakeHead.x--;

number\_moves=number\_moves++;

}

//Поворот головы змеи вверх

private void moveUp() {

snake.snakeHead.y--;

number\_moves=number\_moves++;

}

//Поворот головы змеи вниз

private void moveDown() {

snake.snakeHead.y++;

number\_moves=number\_moves++;

}

//Метод для проверки врезалась ли змейка или нет (проверяется вышла ли голова змейки за пределы игрового поля; равны ли координаты головы координатам сегментов тела змейки)

private void gameOver() {

if (snake.snakeHead.x < 0 || snake.snakeHead.y < 0 || snake.snakeHead.x \* SQUARE\_SIZE >= WIDTH || snake.snakeHead.y \* SQUARE\_SIZE >= HEIGHT) {

gameOver = true;

}

for (int i = 1; i < snake.snakeBody.size(); i++) {

if (snake.snakeHead.x == snake.snakeBody.get(i).getX() && snake.snakeHead.getY() == snake.snakeBody.get(i).getY()) {

gameOver = true;

break;

}

}

}

//Подсчет очков. Максимальное количество баллов если совершено от 0 до 2 ходов (минимальное возможное число ходов для достижения фрукта из любого положения змейки). Если до достижения фрукта сделано более двух ходов, то за каждый штрафной ход от максимального количества отнимается по одному баллу.

private void eatFood() {

if (snake.snakeHead.getX() == food.foodX && snake.snakeHead.getY() == food.foodY) {

snake.snakeBody.add(new Point(-1, -1));

if ((number\_moves>=0)&&(number\_moves<=2)){

score += 5;

}else{

int n=0;

if ((number\_moves > 2) && (number\_moves < 7)) {

for(int i=3; i<=number\_moves;i++){

n++;

}

score+=5-n;

}else score+=0;

}

food.generateFood();

number\_moves=0;

}

}

//вывод очков на экран

private void drawScore() {

gc.setFill(Color.WHITE);

gc.setFont(new Font("Digital-7", 25));

gc.fillText("Очки:" + score, 10, 35);

}

//вывод количество сделанных ходов на экран

private void drawNumber\_moves() {

gc.setFill(Color.WHITE);

gc.setFont(new Font("Digital-7", 25));

gc.fillText("Кол-во ходов:" + number\_moves, 390, 35);

}

}

6 Класс Snake

package Play;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.paint.Color;

import java.awt.\*;

import java.util.ArrayList;

import java.util.List;

// Класс Змейка

public class Snake {

public static List<Point> snakeBody= new ArrayList();

private int ROWS;

private int SQUARE\_SIZE;

Point snakeHead;

//Конструктор с параметрами

public Snake(int SQUARE\_SIZE, int ROWS){

this.SQUARE\_SIZE = SQUARE\_SIZE;

this.ROWS = ROWS;

}

public static List<Point> getSnakeBody() {

return snakeBody;

}

//Появление змейки при старте программы и установка начального местоположения

public void startBody(){

for (int i = 0; i < 3; i++) {

snakeBody.add(new Point(5, ROWS / 2));

}

snakeHead = snakeBody.get(0);

}

//Прорисовка элементов змеи

public void drawSnake(GraphicsContext gc) {

gc.setFill(Color.web("#FF7700"));

gc.fillRoundRect(snakeHead.getX() \* SQUARE\_SIZE, snakeHead.getY() \* SQUARE\_SIZE, SQUARE\_SIZE-2 , SQUARE\_SIZE - 2, 60, 70);

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

gc.fillRoundRect(snakeBody.get(i).getX() \* SQUARE\_SIZE, snakeBody.get(i).getY() \* SQUARE\_SIZE, SQUARE\_SIZE - 5, SQUARE\_SIZE - 5, 40, 70);

}

for (int i = snakeBody.size() - 1; i >= 1; i--) {

snakeBody.get(i).x = snakeBody.get(i - 1).x;

snakeBody.get(i).y = snakeBody.get(i - 1).y;

}

}

}

7 Класс Food

package Play;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.image.Image;

import java.awt.\*;

// класс Еда

public class Food {

private String[] FOODS\_IMG = new String[]{

"/img/orange.png", "/img/apple.png", "/img/banana.png", "/img/e.png"};

private Image foodImage;

int foodX;

int foodY;

private int ROWS;

private int COLUMNS;

private int SQUARE\_SIZE;

// Конструктор с параметрами

public Food(int SQUARE\_SIZE,int ROWS, int COLUMNS){

this.ROWS = ROWS;

this.COLUMNS = COLUMNS;

this.SQUARE\_SIZE = SQUARE\_SIZE;

}

//Прорисовка еды

public void drawFood(GraphicsContext gc) {

gc.drawImage(foodImage, foodX \* SQUARE\_SIZE, foodY \* SQUARE\_SIZE, SQUARE\_SIZE, SQUARE\_SIZE);

}

//Метод для генерация еды в произвольной части экрана(обеспечивается с помощью мат. операции random)

public void generateFood() {

start:

while (true) {

foodX = (int) (Math.random() \* ROWS);

foodY = (int) (Math.random() \* COLUMNS);

for (Point snake : Snake.getSnakeBody())

if ((snake.getX() == foodX) && (snake.getY() == foodY)) {

continue start;

}

foodImage = new javafx.scene.image.Image(FOODS\_IMG[(int) (Math.random() \* FOODS\_IMG.length)]);

break;

}

}

}

8 Класс BGround

package Play;

import javafx.scene.canvas.GraphicsContext;

// Класс игрового поля

public class BGround {

private int ROWS;

private int COLUMNS;

private int SQUARE\_SIZE;

//Конструктор с параметрами

public BGround(int SQUARE\_SIZE,int ROWS,int COLUMNS){

this.SQUARE\_SIZE = SQUARE\_SIZE;

this.ROWS = ROWS;

this.COLUMNS = COLUMNS;

}

//Прорисовка игрового пространства

public void drawBackground(GraphicsContext gc) {

for (int i = 0; i < ROWS; i++) {

for (int j = 0; j < COLUMNS; j++) {

gc.setFill(javafx.scene.paint.Color.GREEN);

gc.fillRect(i \* SQUARE\_SIZE, j \* SQUARE\_SIZE, SQUARE\_SIZE, SQUARE\_SIZE);

}

}

}

}

9 Класс BD

package Most;

import Most.User;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import java.sql.\*;

public class BD {

private static Connection connection;

//Подключение базы данных Game (из СУБД - SQL Server)

public static void Connector(){

String url = "jdbc:sqlserver://LAPTOP-QU5BK6SA;databaseName=Game";

String username = "sa";

String password = "123";

try {

connection = DriverManager.getConnection(url,username,password);

System.out.println("Connect SQL");

} catch (SQLException throwable) {

throwable.printStackTrace();

}

}

private static final ObservableList<User> persons = FXCollections.observableArrayList();

//Получение списка игроков

public static ObservableList getPersons(){

BD.Connector();

read();

return persons;

}

//Метод считывания из базы данных

public static void read(){

String sql = "SELECT\*FROM TableRecords";

persons.clear();

try {

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(sql);

while (resultSet.next()){

String name = resultSet.getString("Name");

int score = resultSet.getInt("Score");

int level = resultSet.getInt("Level");

persons.add(new User(name, score, level));

}

} catch (Exception throwable) {

throwable.printStackTrace();

}

}

//Добавление данных об игроке и результатах его игры в базу данных

public static void addItem(User person){

String sql = "insert TableRecords values('"+ person.getName() + "', "+person.getScore()+", " + person.getLevel() + ")";

System.out.println(sql);

try {

Statement statement = connection.createStatement();

statement.executeUpdate(sql);

} catch (SQLException throwable) {

throwable.printStackTrace();

}catch (NullPointerException throwable){

throwable.printStackTrace();

}

}

//Метод для выполнения сортировки. В параметры метода передается вид сортировки (по уровню или по баллам)

public static void sort(String par){

persons.clear();

try {

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery("SELECT \* FROM TableRecords ORDER BY "+ par);

while (resultSet.next()){

String name = resultSet.getString("Name");

int score = resultSet.getInt("Score");

int level= resultSet.getInt("Level");

persons.add(new User(name, score, level));

}

} catch (Exception throwable) {

throwable.printStackTrace();

}

}

}

10 Класс User

package Most;

//Класс Игрок

public class User {

private String name;

private int score;

private int level;

//Конструктор с параметрами

public User(String name, int score, int level) {

this.name = name;

this.score = score;

this.level = level;

}

//Метод для получения имени

public String getName(){

return this.name;

}

//Метод для получения количества баллов

public int getScore(){

return this.score;

}

//Метод для получения уровня сложности

public int getLevel(){

return this.level;

}

}