Белорусский Государственный Университет Информатики и Радиоэлектроники

Факультет Компьютерных Систем и Сетей

Кафедра Электронных Вычислительных Машин

Кроссплатформерное программирование

Отчет

По лабораторным работам

На тему : «Игра Tetris с использованием JavaFX»

Выполнил:

ст. гр. 350504

Пискун А.А.

Проверил:

Кухарчук И.В.

Содержание:

Лабораторная работа №1 – Создание пользовательского интерфейса будущей игры:

Для того, чтобы разработать пользовательский интерфейс в JavaFx используется Stage, как контейнер главного уровня, Group, для создания новой группы сцен (Scene) с помощью которых и происходит отрисовка интерфейса.

При создании интерфейса было реализовано 3 Группы :

1) Главная группа, меню – menu

2) Группа для обработки событий игры – game

Для каждой группы была реализована отдельная сцена, для отрисовки элементов, находящихся в ней.

С помощью кнопок происходит навигация между группами :

– newGameBtn – переход на группу событий игры game

– backBtn – возврат в главную группу из game

Рис. 1 – Главное меню (menu)

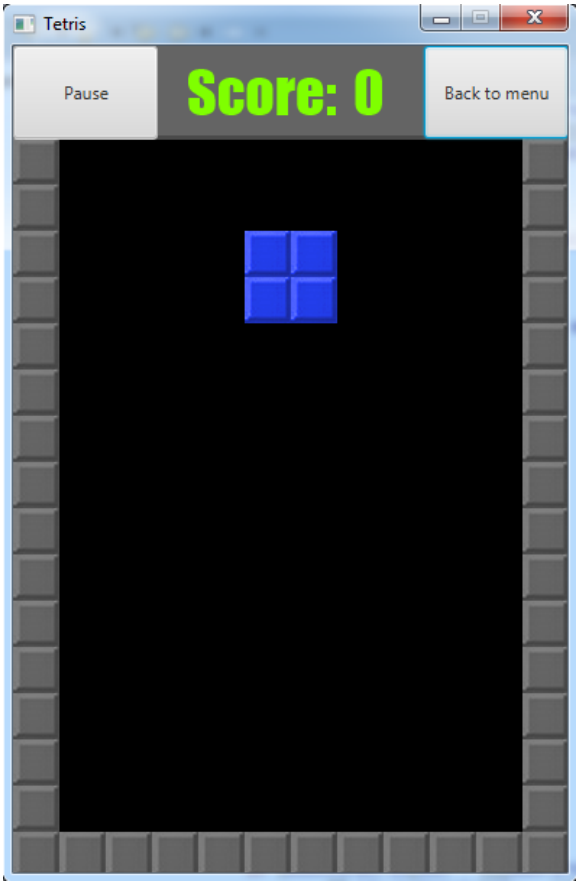
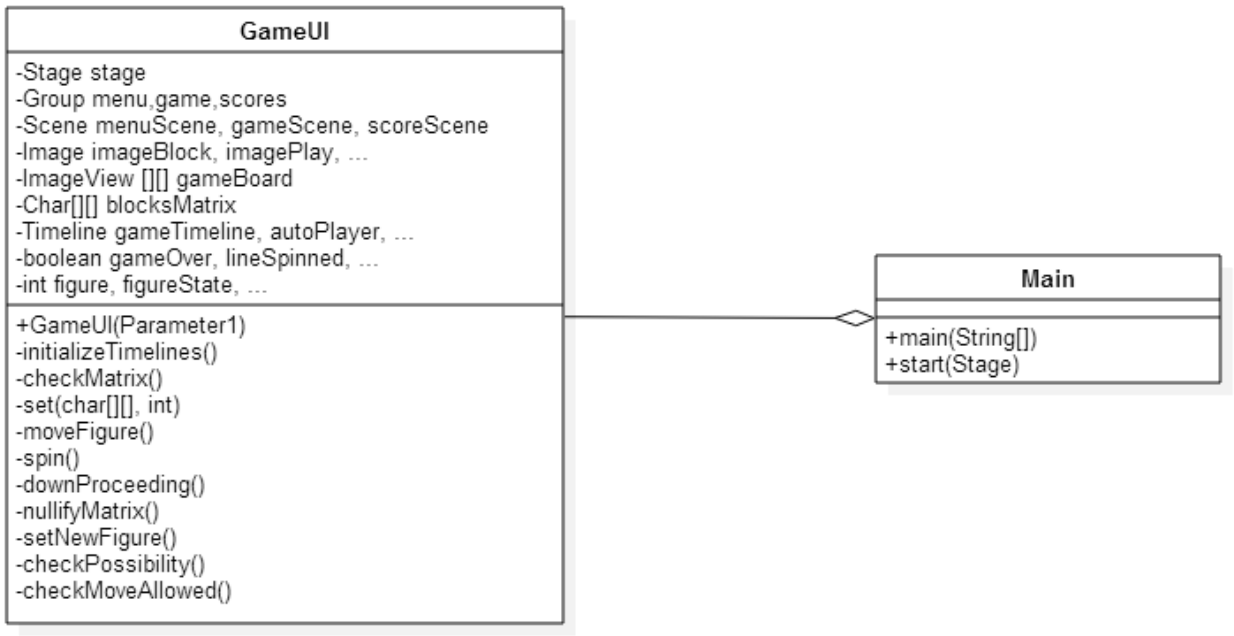


Рис. 2 – Группа для событий игры (game)

Лабораторная работа №2 – Разработка приложения:

При проектировании приложения, были созданы и развиты 2 класса. Один из них выполняет функцию создания окна и передачи параметров в другой класс. Второй - класс, реализующий игровой процесс: движение всех элементов игры, их отрисовка, управление счетом и его отображение, а также вывод сообщения об исходе игры.

Основные методы и данные классов, а также их связь можно увидеть на диаграмме классов:



Описание классов:

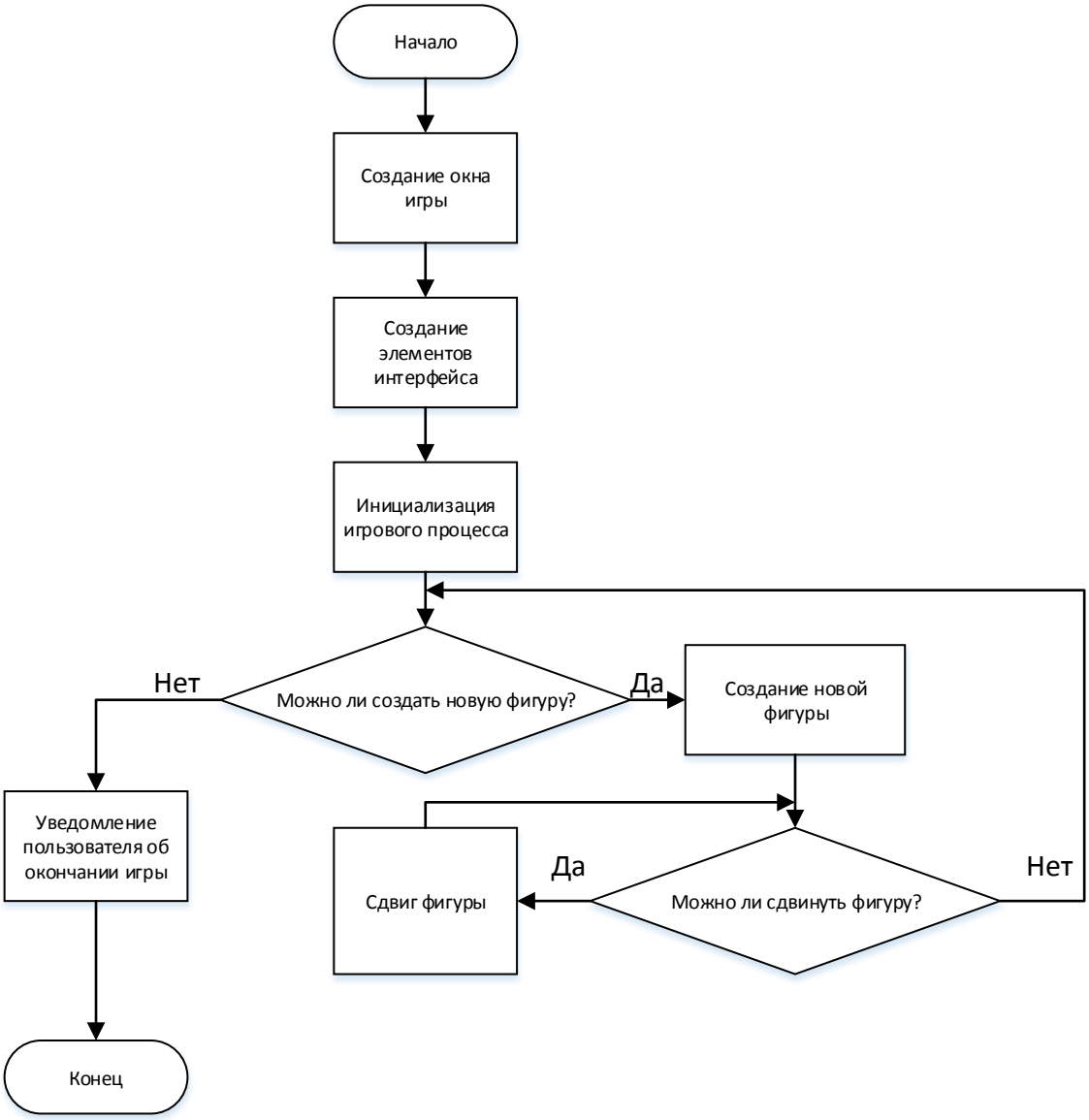
В данном проекте было решено использовать 2 класса, для реализации поставленной задачи. 2 класса – оптимально количество для полноценной реализации игры. Так же это позволило избежать избытка кода в некоторых местах приложения, и облегчило работу с классами.

Основной класс служит для создания главного окна игры (stage), а так же создания элемента класса игровой логики (GameUI).

Класс игровой логики выполняет функцию добавления на экран и отрисовки различных элементов игры, а так же работу с ними, и многочисленные проверки. Элементами игры являются: кнопки, поля для текста, бекграунд, а так же матрица элементов фигур.

При написании обоих классов были учтены основные принципы ООП, а так же был выбран единый стиль написания кода.

Блок-схема алгоритма игры :



Код является самодокументирующимся.

Лабораторная работа №3 – Реализация механизма воспроизведения игры:

В главном классе игры GameUI реализован механизм обработки и создания сохранений игры, а так же воспроизведения уже законченного игрового сеанса. Для этого используются классы PrintWriter и BufferedReader.

Механизм сохранения:

Сложность игры (цифра)	Набор сдвигов влево \ вправо	Набор сдвигов вниз	Новая фигура (F+ номер)	Набор сдвигов …		Набор сдвигов вниз	Конец файла

Загрузка происходит аналогично записи сохранения, но со считыванием каждого символа в отдельности, и выполнении того или иного участка кода в зависимости от считанного символа.

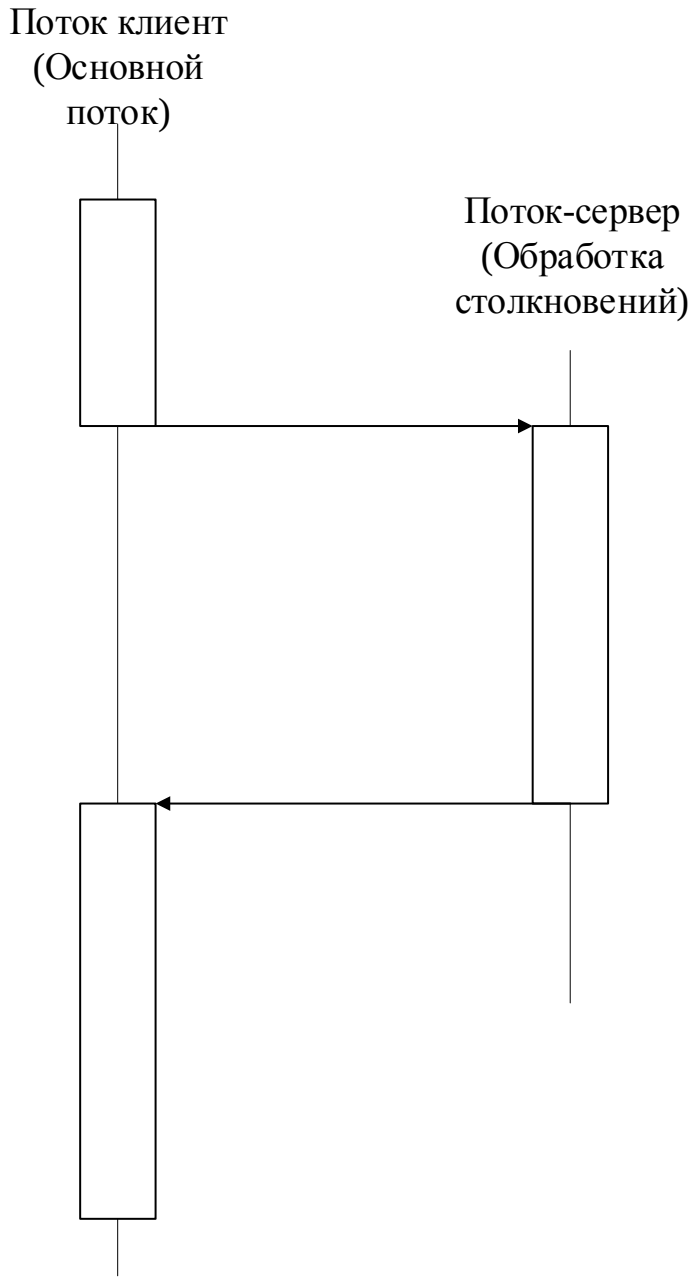
По окончании воспроизведения игра переходит в ждущий режим, ожидая сигнала на продолжение. Так же реализован механизм дозаписи в файл, чтобы не терялось текущее сохранение при продолжении игры.

Лабораторная работа №4 – Разбивание игры на 2 потока. Создание потока-сервера и потока-клиента:

Для реализации поставленной задачи было принято решение о добавлении еще одного класса в проект, а так же перенесении части игровой логики в новый класс. Этой частью была выбрана проверка на столкновения фигуры с другими фигурами или границами игрового поля (при сдвиге вниз).

Для того, чтобы наш новый класс-поток работал как обработчик столкновений требовалось унаследовать наш класс от класс Thread, и добавить логику взаимодействия с потоком-клиентом. Для использования данной проверки класс-клиент использует экземпляр класса-сервера. Для того, чтобы потоки работали синхронно был использован блок synchronized. После выполнения проверки в классе-сервере устанавливается Boolean флаг в нужное нам состояние, а в классе-клиенте считывается это состояние. Исходя из полученных результатов, мы выполняем те или иные дальнейшие действия в классе – клиенте. Для правильной работы потока, он был указан как daemon(требуется для фоновой работы потока).

```
engine = new CheckingEngine(semaphore);

engine.setDaemon(true);

engine.start();

try {

        engine.setMatrix(blocksMatrix);

        synchronized (semaphore) {

                engine.startChecking();

                semaphore.notifyAll();

                if (engine.getChecking())

                        semaphore.wait();

        }

        cannotMove = engine.getMoveState();

} catch (Throwable error) {

        error.printStackTrace();}
```



После выполнения данного действия (выполнения кода в доп.потоке) основной поток считывает состояние флага из экземпляра класса потока.

Лабораторная работа №5 – Сортировка сохраненных игр на языке Scala:

Для выполнения лабораторной работы требовалось отсортировать сохраненные игры нашего приложения, для нахождения и вывода на экран (в консоль) номера наилучшей игровой сессии (кратчайшей).

Файлы сохранения игры могут создаваться как игроком, так и компьютером (автоигрок). Все сохранения игры находятся в папке saves корневой директории проекта.

Алгоритм поиска кратчайшей сессии был выбран исходя из стиля заполнения файлов сохранений. Наикратчайшей игровой сессией является та, в файле сохранения которой находится наименьшее количество строк (сдвигов вниз). Данный алгоритм вычисляет длину всех файлов сохранений построчно, и заносит значения в массив. Далее данный массив сортируется 300тыс. раз как на языке Scala, так и на языке Java, чтобы отразить разницу в скорости сортировок.

Алгоритм сортировки на языке Scala построен на принципе хвостовой рекурсии, а так же реализует сортировку Хоара.

Для нахождения времени сортировки используется функция System.currentTimeMillis().

Исходя из полученных результатов, можно заметить, что сортировка на языке Scala выполняется значительно дольше, чем на языке Java.

Лабораторная работа №6 – Построение карты игры:

Для выполнения лабораторной работы требовалось с помощью информации, полученной при выполнении 5 лабораторной работы построить карту игры (информацию по играм)

Для выполнения поставленной задачи были выбраны направления: карта первых 10 игр по длинам, наиболее популярное движение среди всех игр.

В переменную типа List<Integer> topTen = new ArrayList<>() передается список лучших игр (по длине), полученный из функции, написанной на Scala.

На Scala же этот список формируется из значений, основанных на сортировке из предыдущей лабораторной работы.

val topGames = sortedGames(0) :: sortedGames(1) :: … :: NiL

Вся требуемая информация выводится в отдельном окне (movesStage) в понятном для пользователя виде.

Лабораторная работа №7 – Подмена игровой нотации псевдокодом:

А данной лабораторной работы требовалось выполнить подмену разработанной для сохранений нотации (в данном случае только лучшей игры) псевдокодом для повышения ее читаемости. Поставленная задача была выполнена посредством метода сопоставления с образцом.

Оригинальная нотация была набором символов (F, U, R, L, N) и цифр, исходя из этого было принято решение получать требуемый псевдокод построчно с помощью case-объектов.

```
def changeNotation(originalStr: String): String= originalStr match {
    case “L” => “Left move of figure”
    case “R” => “Right move of figure”
    case “U” => “Spinning of a figure clockwise”
    case “F” => “New figure”
    case “N” => “Figure is down proceeded”
    case “0” => “New Game Difficulty :” + originalStr
    case “2” => “New Game Difficulty :” + originalStr
    case “4” => “New Game Difficulty :” + originalStr
    case “9” => “New Game Difficulty :” + originalStr
    case _ => println(“Error in changing notation”);
}
```

Далее полученные строки соединяются посредством StringBuilder и записываются в новый файл (bestGameNotation.txt), так же на Scala.

```
def createFile(fileName :String, infoString: String){
    val writer = new PrintWriter(new File(fileName))
    writer.write(infoString)
    writer.close()
}
```