**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,**

**СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ СЕТЕЙ И СИСТЕМ (ИКСС)**

**КАФЕДРА ЗАЩИЩЕННЫХ СИСТЕМ СВЯЗИ (ЗСС)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Курсовая работа по предмету:

«Разработка защищенных сетевых приложений»

на тему:

«**Разработка простой пошаговой игры**»

Выполнил студент группы ИБС-23:

Панферов Кирилл Константинович

Проверил:

старший преподаватель ЗСС Цветков А.Ю.

**Содержание**

[**1.Введение.....................................................................................................2**](#Введение)

[**2. Создание экономической модели…………………………………….3**](#Эконом_модель)

[**2.1 Количество требуемых крестьян для захвата……………...3**](#колво_крестьян)

[**2.2 Возможные ходы игрока и их описание……………………..7**](#возмодные_ходы)

[**3. Создание искусственного интеллекта…………………………..….11**](#Создание_ИИ)

[**4. Создание функции сохранения…………………………..………….12**](#Созданиефункции_сохранения)

[**5. Создание графиков в конце игры…………………………………..14**](#графики_конец)

[**6.** **Блок схема программы………………………………………………15**](#Блок_схема)

[**7. Листинг программы…………………………………………….……16**](#листинг)

[**8. Ссылка на репозиторий GitHub……………………………….……43**](#гит)

[**9. Список использованных ресурсов………………………………….43**](#источники)

1. Введение

Целью курсовой работы является разработка простой пошаговой игры с экономической модель, искусственным интеллектом и минимальным графическим интерфейсом.

В связи с поставленной целью, необходимо решить следующие задачи:

1. Разработать сбалансированную и действующую экономическую модель
2. Разработать искуственный интеллект, принимающий правильные стратегические решения
3. Создать графический интерфейс для взаимодействия с игрой

2. Создание экономической модели

В связи с поставленной задачей в Введении, экономическую модель стоит разрабатывать опираясь на следующие аскиомы:

1. У каждого игрока (под двумя игроками имеется ввиду пользователь программы и искусственный интеллект) на начальной стадии есть поле с рисом и поле с водой
2. Чем дальше от игрока находится незахваченное поле, тем больше крестьян требуется для его захвата
3. Поля с рисом располагаются на карте случайным образом, однако их количество в правой нижней и левой верхней половинах должно быть одинаково (для баланса количества побед/поражений)

Такая экономическая модель обеспечивает соревновательную составляющую игры (благодаря случайной генерации полей с рисом), а так же исключает возможные события по типу отсутствие полей с рисом, следовательно, невозможность построить дом, следовательно невозможность захватить новую территорию.

2.1 Количество требуемых крестьян для захвата

Так как на каждом пустом поле игрок в праве построить дом, который будет давать по одному крестьянину в день, а крестьяне для захвата территории не расходуются, **количество крестьян со временем растет экспоненциально** (при условии, что игрок пытается получить максимальный доход и строит дома по мере возможности, а не бездействует). Из этого следует, что и **количество крестьян для захвата полей должно расти экспоненциально** по мере отдаления от начальных положений игроков.

Так как работа будет проходить с двумерным массивом, целесообразно будет оперировать переменными i и j, показывающими положение в этом двумерном массиве. Однако проходя от середины к краям эти переменные

* Убывают при движении влево вверх
* Возрастают при движении вправо вниз

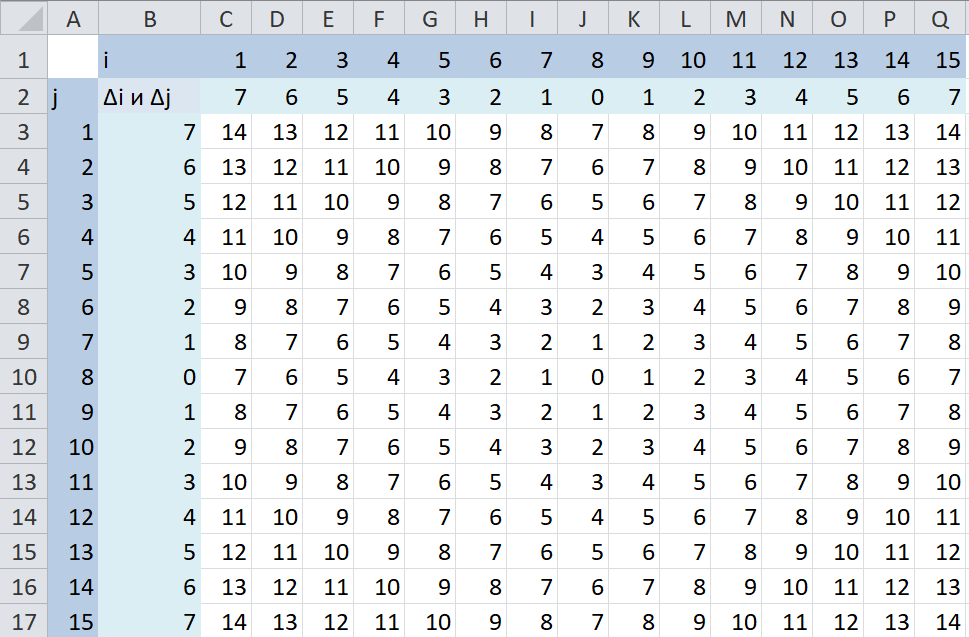
Следовательно, целесообразно ввести величину Δi и Δj:

Δi = **|iср - i|**; Δj = **|jср - j|**

А так же ввести новое правило:

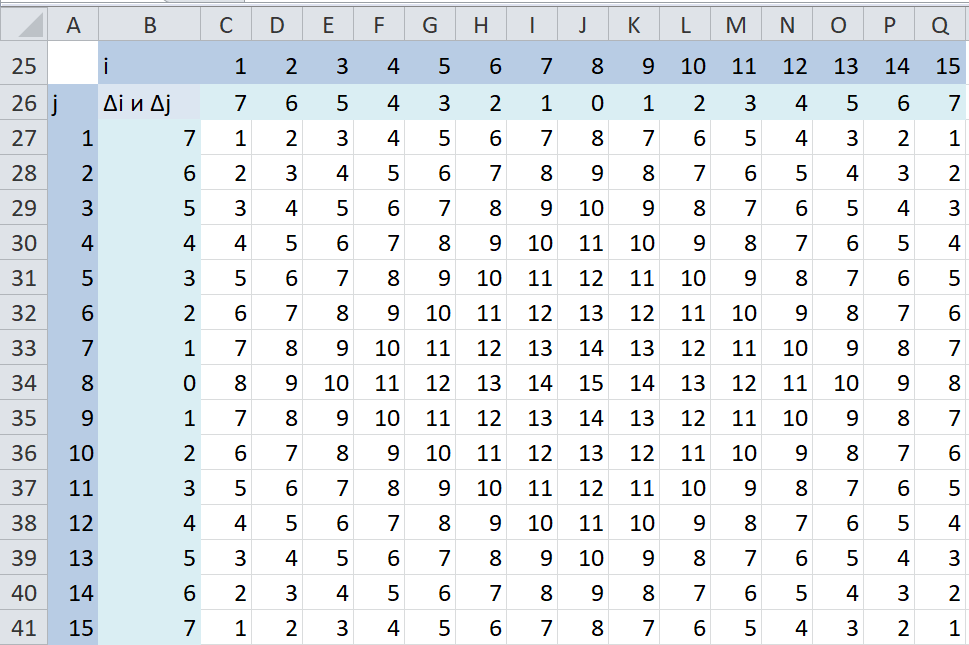
**«Количество рядов и количество столбцов должно быть нечетным»**

Таким образом, законы распределения Δi и Δjбудут обратны нужным нам:



поэтому изменим их на

Δi = |**iмакс -** |iср - i||; Δj = |**jмакс -** |jср - j||



Однако, в таком случае количество требуемых крестьян может убывать по мере отдаления от начальной точки, что не есть правильно. Так как задачей игры является захватить 50% территории, при такой экономической модели игроки будут стремиться захватить все поля с i = 0 и j = 0, после захвата которых их количество полей будет равно 15 \* 2/ (15 \* 15) = 2/15 = примерно 12% при минимальных затратах, после чего приоритетом станет захват полей с i = 1 и j = 1.

Так как игра должна вызывать интерес, действия игроков должны зависеть от выбранной стратегии и случайной составляющей. Для реализации этих требований игра не должна иметь наглядно простых способов победить, следовательно **каждая стратегия должна быть равносильна другой**. Если игрок начинает захватывать все крайние поля он сталкивается с проблемой: дальнейшее развитие возможно лишь в одну сторону – в строну отдаления от краев. Эта оговорка должна заставить игрока задуматься о пользе захвата крайних полей, однако при данной стоимости каждой клетки оговорка не столь важна – **крайние поля требуют слишком малых затрат.**

Исходя из этого, к уже имеющимся переменным Δi и Δj **добавляется значения отдаленности их координат от начальных координат игроков** **(0, 0) и (14, 14):**

При (**i + j < iср +jср**):

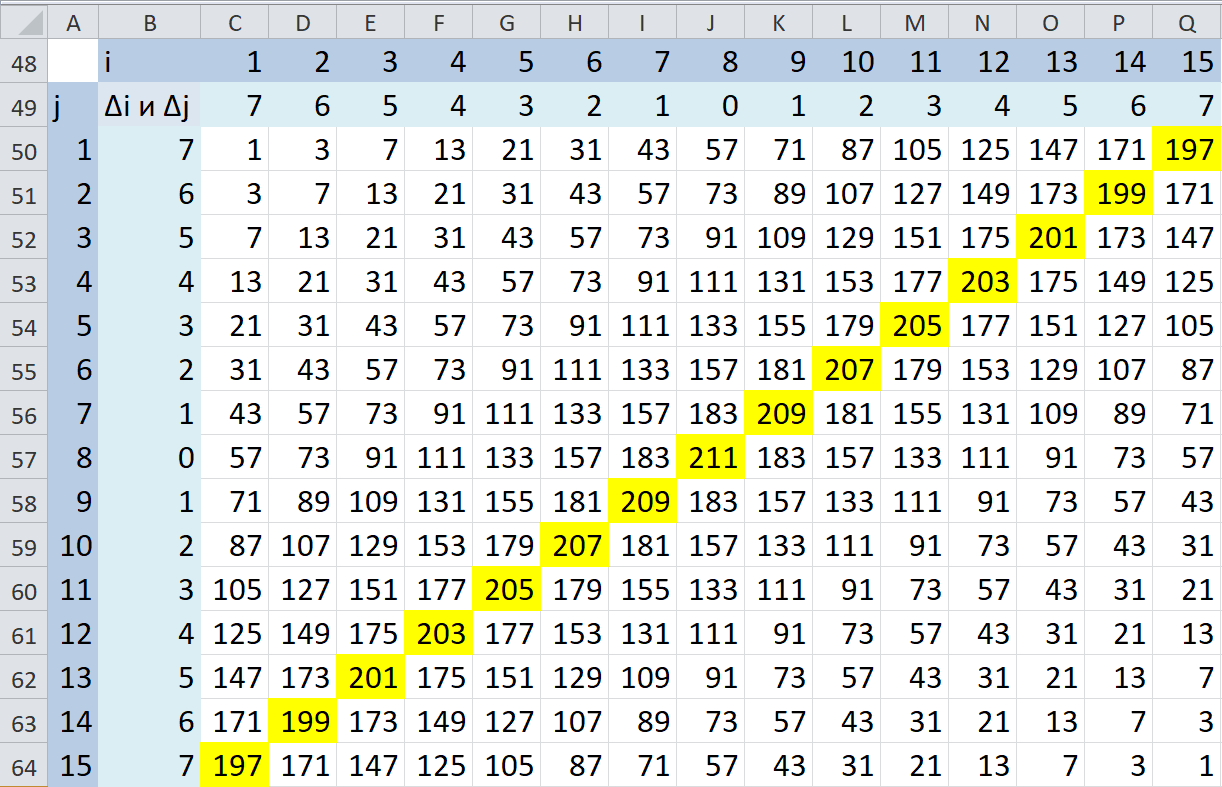
Δi = |iмакс - |iср - i|| **+( i + j) ^ 2**; Δj = |jмакс - |jср - j||**+( i + j) ^ 2**

При (**i + j < iср +jср**):

Δi = |iмакс - |iср - i|| **+(iмакс - i +jмакс – j) ^ 2;**

Δj = |jмакс - |jср - j||**+(iмакс - i +jмакс – j) ^ 2;**





Таким образом мы получили двумерный массив, показывающий, насколько «сложным» должен быть процесс захвата конкретного поля. Он позволяет нам сохранить главное правило логики экономической модели: **каждое последующее поле должно требовать больше крестьян для захвата.**

2.2 Возможные ходы игрока и их описание

Всего возможных ходов у игрока 4:

* Полить рис
* Набрать воды
* Построить дом
* Захватить новое поле

По условию каждое действие должно приводить к проверке на выигрыш (игрок захватил 50% территории) а затем к ходу ИИ.

Все действия относятся к конкретным полям в игровом мире, а некоторые – ко всем полям одного типа во владениях игрока (полив риса и воды). Так как по условию и задумке игры один ход – один день, нелогично тратить один ход на полив одного поля с рисом, следовательно при выборе игроком хода “Полить рис” **должны поливаться несколько полей с рисом** во владении игрока. Аналогично, при выборе хода “Набрать воды” **должна собираться вода с нескольких полей** во владении игрока. Далее будут более детально рассмотрены правила для каждого действия.

Описание действий

1. **Полить рис**

По условию при поливе риса расходуется вода, причем на следующий ход рис растет лучше (по условию одно поле риса каждый ход дает фиксированное количества риса).

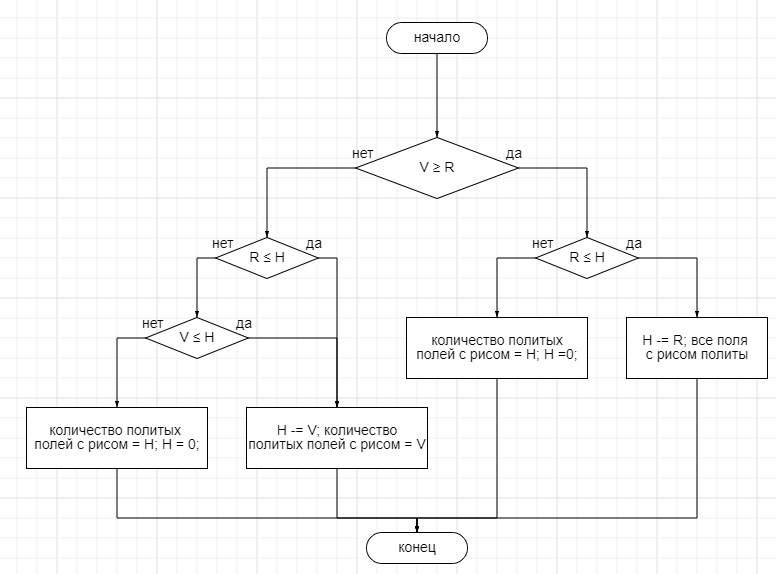
Для удобства, пусть минимальная единица риса будет “Количество риса, собранное с одного поля риса”, тогда **каждое поле риса будет по 1 минимальной единице риса (далее 1 единица риса) в день.**

При поливе риса рис должен давать больше единиц риса, причем это добавленное количество должно быть равноценным одному ходу. Учитывая, что за один ход и так дается 1 единица риса за каждое поле с рисом, логично что за один потраченный ход на сбор воды и один потраченный ход на полив риса будет даваться 2 единицы риса, то есть

Каждое поле с рисом дает 1 единицу риса в конце каждого хода

Каждое политое поле с рисом дает 2 единицы риса в конце каждого хода, причем политым поле с рисом остается не только на следующий ход после полива, но на 2 следующих хода.

Чтобы полить рис игрок должен нажать на поле с рисом в его владении. Так как темп игры задается условием “Дом крестьянина дает 1 крестьянина” и “Каждое поле с рисом в конце дня дает 1 единицу риса” логично избавиться от варианта “При поливании риса у игрока появляется лишь одно политое поле с рисом”, следовательно, **ход “Полить рис” должен приводить в состояние “Политый” несколько полей с рисом.** Однако, на полив одного поля с рисом требуется 1 единица воды, а так же один крестьянин, который, естественно не будет расходоваться при поливе.

Таким образом, при ходе “Полив риса”: 

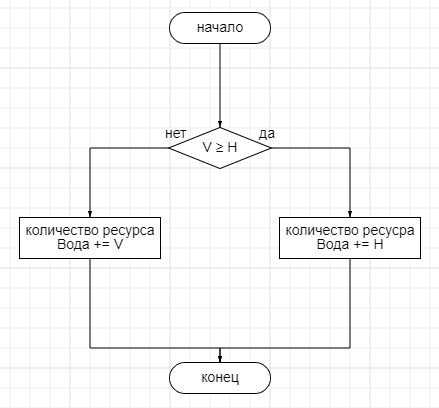
**Рис. 5 – блок схема хода «Полив риса»**

где **V** – количество жителей; **R** - количество полей с рисом; **H** – количество воды;

**2. Набрать воды**

По условию на набор воды тратится один ход, аналогично с ходом “Полив риса”, следуя из темпа игры стоит выбрать вариант “Собирается вода со всех полей с водой”, однако по правилам логики воду должен собирать крестьянин, следовательно игрок не может собрать больше воды чем у него имеется крестьян.

Таким образом, при ходе “Набрать воды”



**Рис. 6 – блок схема хода «Набрать воды»**

**3. Построить дом**

**По условию** построить дом можно на имеющемся пустом поле, причем расходуется 1 единица риса и 1 крестьянин. Однако, так как на следующий ход этот же дом даст нового крестьянина, **эта цена слишком мала**. Так же, из-за показательного роста количества крестьян и количества риса будет логично присвоить величине **h\_c** (house\_cost) показательное распределение, т.е. **после каждого построенного дома**:

- количество требуемого риса увеличивается на Δh\_c

- Δh\_c увеличивается линейно

-количество требуемых крестьян увеличивается линейно

**4. Захватить новое поле**

В пункте 1 была разобрана экономическая модель стоимости каждого нового поля, однако стоит добавить, что по данной экономической модели у каждого поля есть переменная **required\_villagers\_amount**, которая хранит в себе количество жителей, требуемое для захвата территории. По логике крестьянам нужна еда для того чтобы захватить территорию, а так же в процессе игры было выявлено, что если игрок может захватывать поля не тратя ресурсы – игра становится слишком быстрой и прямолинейной. По этой причине было решено **при каждом захвате поля уменьшать количество риса игрока на величину количества требуемых крестьян для захвата данного поля.**

В дальнейшем, после тестов было принято решение так же уменьшать количество крестьян после захвата поля. Для создания интереса соперничества, это уменьшение будет определяться для каждого захвата **случайным образом,** величина уменьшения количества крестьян «погибших при захвате» будет распределена случайно в интервале **[0;required\_villagers\_amount].**

3.Создание искусственного интеллекта

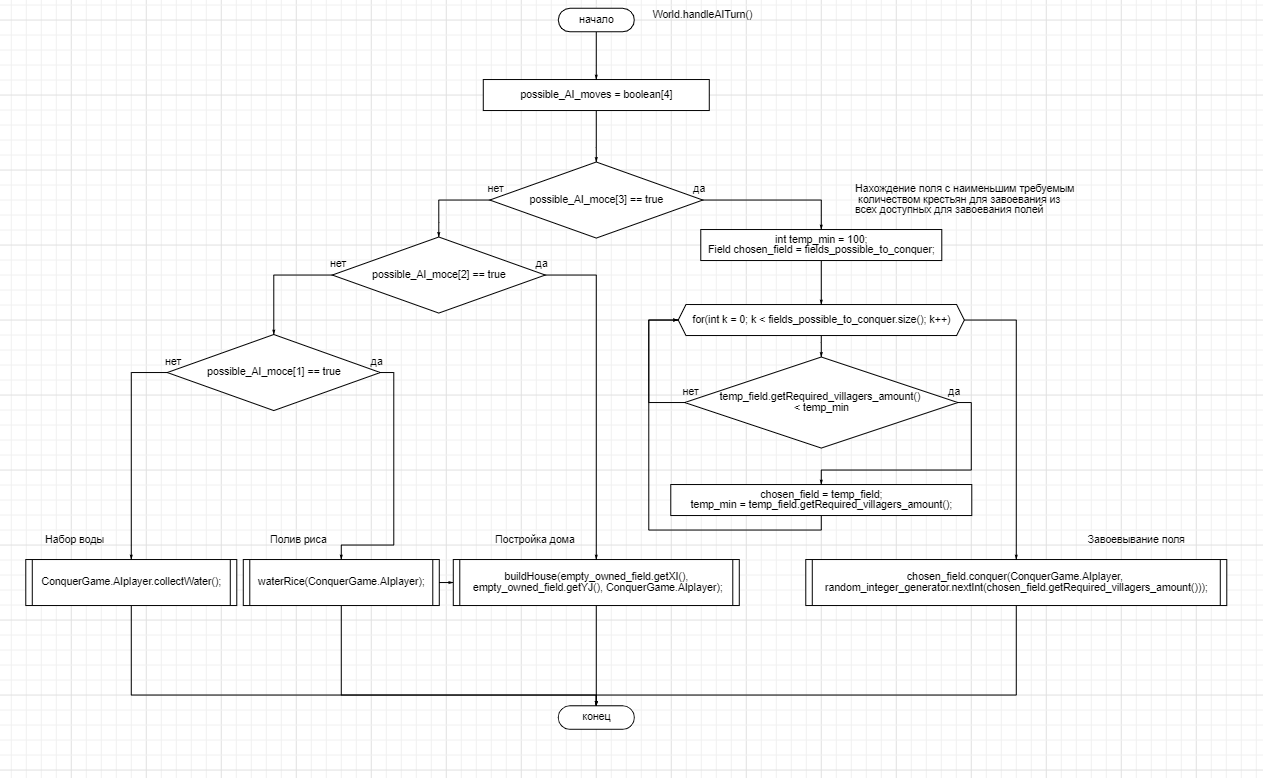
Искусственный интеллект основан на принципе выполнения самого приоритетного хода. По логике игры самый приоритетный ход – **захват поля**, так как он непосредственно приближает игрока к победе. Далее идет ход **построить дом,** так как для захвата полей нужны крестьяне и рис. Следующим по убыванию приоритета идет ход **полить рис,** увеличивающий ресурсы игрока, требуемые для победы. Последним в списке приоритета стоит ход **набрать воды**, в основном так как этот ход – возможен всегда, следовательно программе не требуется обрабатывать случай, когда у ИИ нет доступных ходов (пропуск хода).

Для реализации этого принципа используется массив 4 булевых значений **possible\_AI\_moves,** каждое значение отвечает за определенный тип хода:

{"Набрать воды", "Полить рис", "Построить дом", "Захватить поле"}

Принцип выбора самого приоритетного хода – **чем больше индекс тем больше приоритет хода.**

Таким образом, проходя по данному массиву с конца к началу и выполняя ход, если значение массива – **true** можно гарантировать выполнение самого приоритетного хода.



4. Создание функции сохранения

Для сохранения и продолжения игры требуется разработать структуру данных, с помощью которой будет хранится состояние игры на данный момент. Это можно реализовать через обычный текстовый документ с расширением .txt и условной структурой разделения и хранения данных.

Однако, хранить все данные игры, к примеру массив **grid[][]** состоящий из объектов класса **Field** будет не целесообразно. Поэтому **достаточно хранить элементарные данные** о состоянии полей и ресурсов игроков, а **при загрузке сохранения выстраивать игровой мир по новой** используя уже имеющиеся методы.

Для того, чтобы продолжить игровой процесс с конкретного моменты нужно знать:

1. Тип каждого поля (пустое, рис, вода или дом)
2. Принадлежность полей игрокам
3. Ресурсы игроков

1. Для того, чтобы сохранить тип каждого поля достаточно организовать структуру так: проходя по всем элементам массива **grid[][]** игрового мира в текстовый документ записывается **тип данного поля + пробел,** при переходе на следующую строчку (при выходе из вложенного цикла) записывается **переход на новую строку**.

Таким образом, при считывании файла можно будет прописать **.split()** и получить массив, состоящий из отдельных строк массива **grid[][],** которые, в свою очередь, так же делятся функцией **.split()** уже на отдельные элементы, при этом результат всех этих действий есть двумерный массив, хранящий в себе данные типе каждого поля первоначального массива **grid[][].**

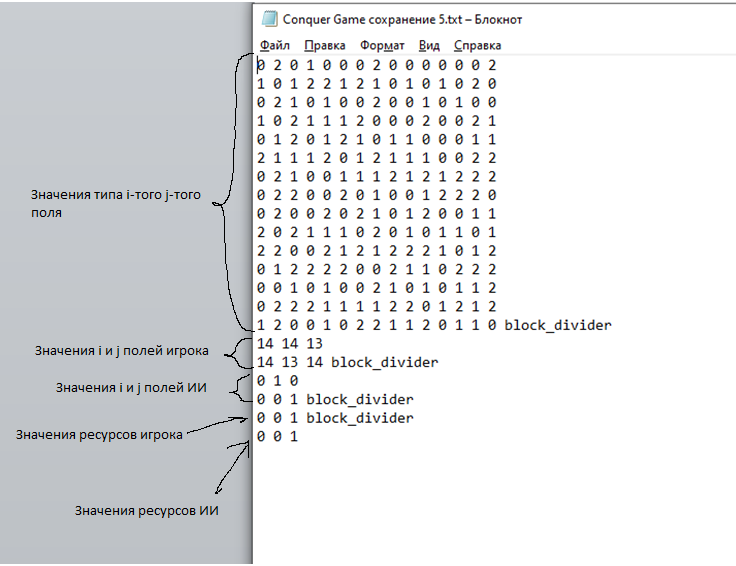
2. Чтобы сохранить принадлежность полей каждому игроку достаточно сохранить массивы **x\_fields** и **y\_fields** каждого игрока, а затем, при загрузке игры из файла, присваивать поля под этими индексами с помощью метода **conquer()** класса **Field.**

Данные индексов **i** полей во владении игроков сохраняются через пробел, после чего следует перенос строки и запись индексов **j** все так же через пробел.

3. Данные о ресурсах игроков сохраняются по правилу

**«Количество риса» + «*пробел*» + «Количество воды» + «*пробел*» + «Количество крестьян»**

Каждый «блок» данных разделяется служебным словом **«block\_divider»,** по которому изначально будет проводится **.split()** при загрузке сохранения.



Дополнение

Для выбора файла (при загрузке) и директории (при сохранении) используются библиотеки javafx.stage.FileChooser и javafx.stage.DirectoryChooser соответственно. Эти библиотеки позволяют запустить процесс открытия файла или папки с помощью проводника Windows. Сами методы этих классов позволяют лишь запустить окно выбора файла и получить путь до файла/папки, сами запись/чтение производятся с помощью java.io.FileWriter и java.io.File.

5. Создание графиков в конце игры

Для создания графиков на протяжении игры каждый ход собирается информация о:

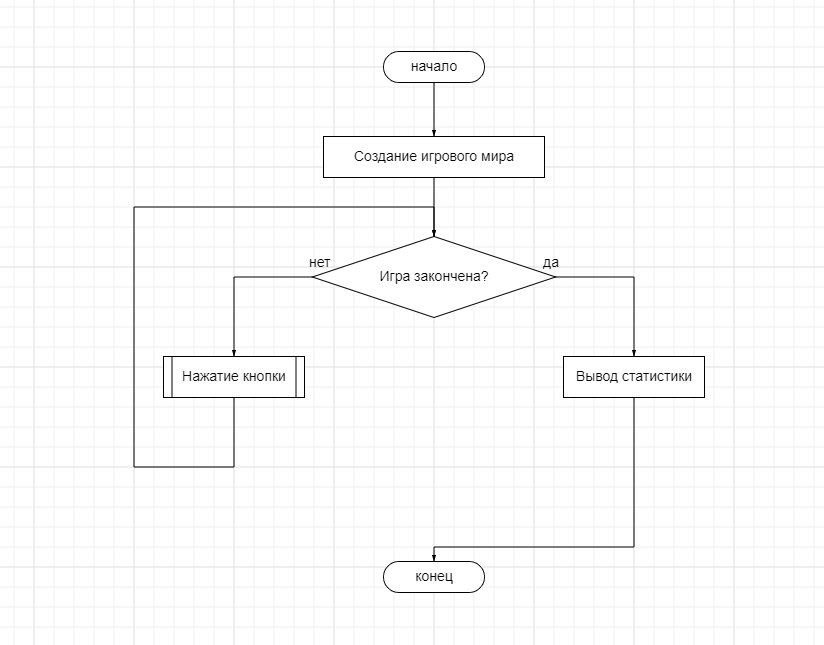
* Количестве риса игроков
* Количестве захваченных полей игроков
* Количестве крестьян

В конце игры, при победе или поражении игрока выводится на экран 3 графика: «Мониторинг количества риса». «Мониторинг количества крестьян» и «Мониторинг количества захваченных полей».

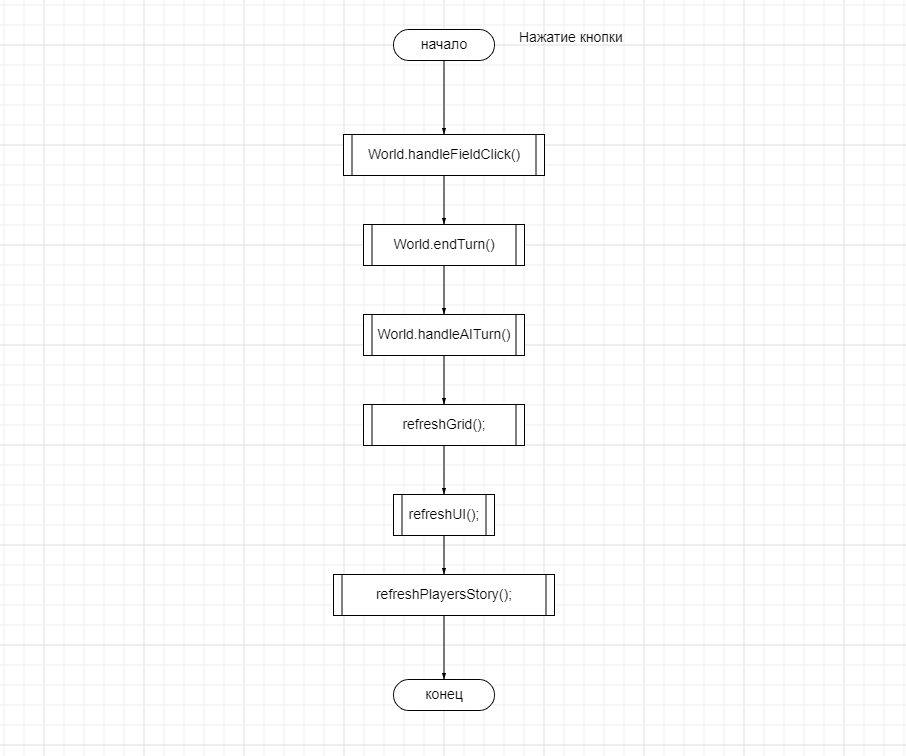
Эта функция реализована через метод World.showEndgameGraphs() с помощью класса LineChart библиотеки JavaFx. Данный класс позволяет составлять графики с любыми желаемыми данными, требует для работы библиотеки javafx.scene.chart.LineChart (непосредственно класс для создания графиков в JavaFx), javafx.scene.chart.NumberAxis (для объявления оси ординат и абсцисс каждого графика) и javafx.scene.chart.XYChart (для заполнения графиков).

6. Блок схема программы

Программа работает по принципу 1 нажатие на кнопку (поле) – 1 дневной цикл. При запуске программы открывается главное меню с тремя возможными кнопками: «Новая игра», «Продолжить игру» и «Выход». При нажатии «Новая игра» создается объект класса World, конструктор которого вызывает создание игрового поля.

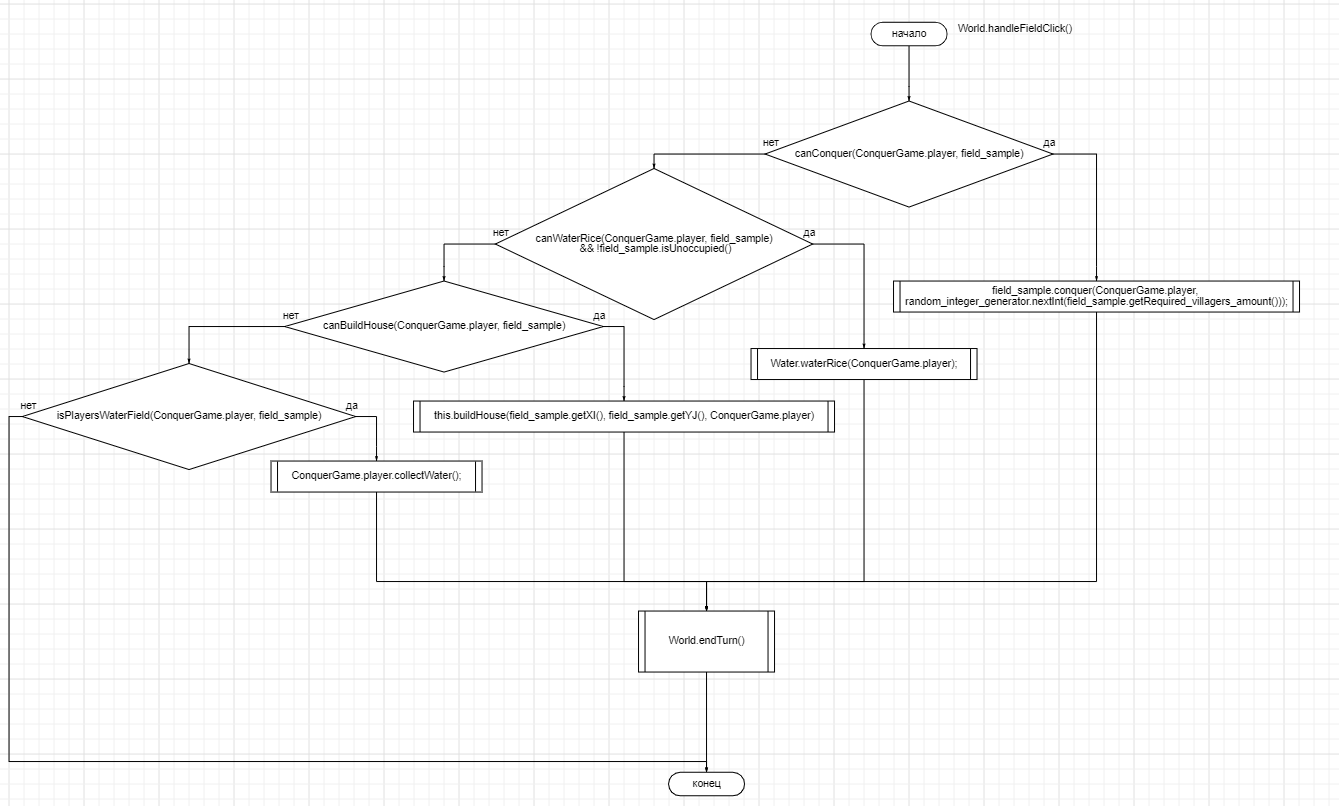


При создании всех полей к каждой кнопке указывается метод World.handleFieldClick(), запускающая отдельные методы, реализующие ходы.

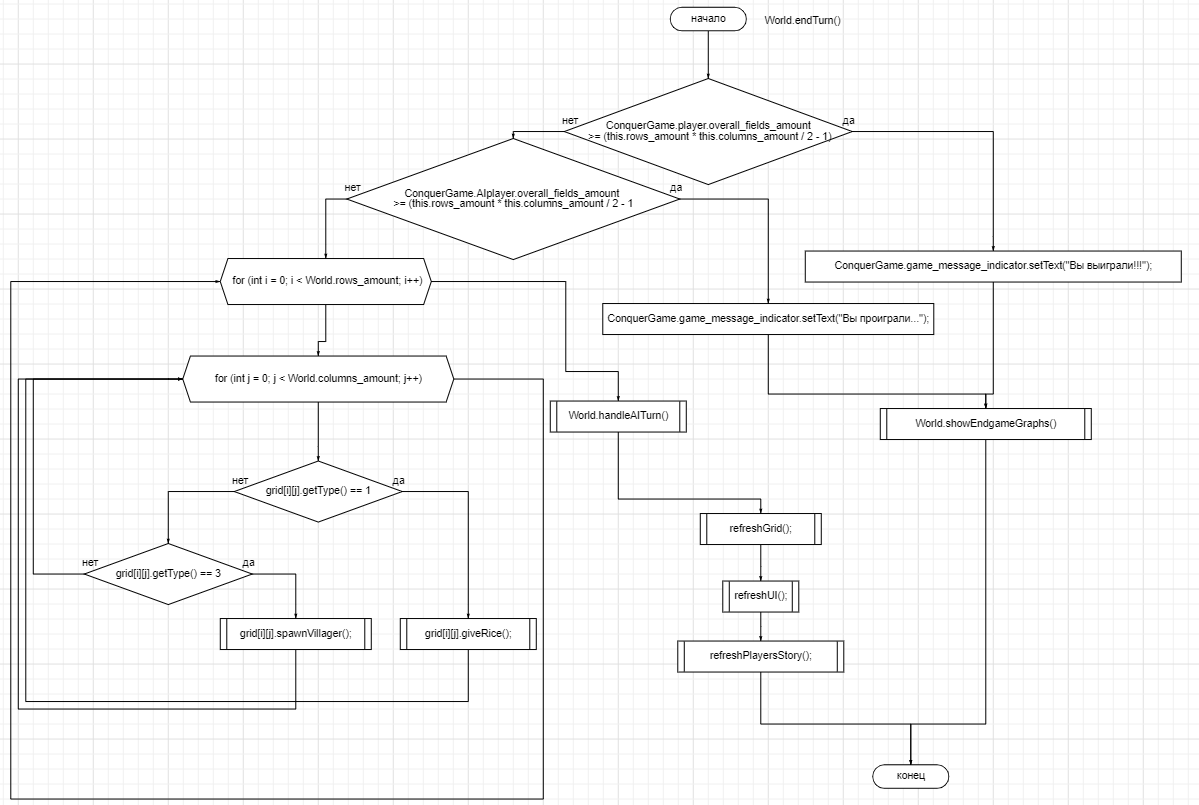


После выполнения всех команд (при условии, что World.endTurn() не привел к победе одного из игроков) игроку снова предоставляется выбор какую кнопку нажать. Эта функция, не смотря на то что не находится в цикле, постоянно вызывается, пока игрок не выиграет или проиграет.

World.handleFieldClick():



World.endTurn()



7. Листинг программы

Класс ConquerGame

package com.example.course\_paper;  
  
import javafx.application.Application;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
  
import java.io.File;  
  
public class ConquerGame extends Application {  
Переиспользованный код (здесь я скопировал структуру main() и в дальнейшем использолвал создание кнопок Button() по тому же принципу,как и в источнике.  
 // JavaFx запускается с помощью метода launch() наследуемого класса Application  
 public static void main(String[] args) {  
 *launch*();  
 }  
  
 // Объявление констант для работы программы  
 // cell\_height и cell\_width - размеры "полей"  
 // ПАВЕЛ ИВАНОВИЧ, ЕСЛИ БУДЕТЕ ТЕСТИТЬ ПОСТАВЬТЕ ИХ РАВНЫМИ 150  
 public static final int *cell\_height* = 50; // 150  
 public static final int *cell\_width* = 50; // 150  
  
 // Размеры окна программы  
 private final int stage\_height = 750;  
 private final int stage\_width = 750;  
 private final int user\_menu\_width = 150; // Ширина пользовательского меню  
  
 // Количество столбцов и рядов "полей"  
 private final int rows\_amount = stage\_height / *cell\_height*;  
 private final int columns\_amount = stage\_height / *cell\_width*;  
  
 // Объявление объектов класса Player  
 public static Player *player* = new Player(); // Это игрок  
 public static Player *AIplayer* = new Player(); // Это ИИ  
  
 // Объявление кнопок-индикаторов  
 public static Button *rice\_amount\_indicator* = new Button(); // Индикатор количества риса  
 public static Button *water\_amount\_indicator* = new Button(); // Индикатор количества воды  
 public static Button *villagers\_amount\_indicator* = new Button(); // Индикатор количества крестьян  
 public static Button *game\_message\_indicator* = new Button(); // Индикатор с сообщениями для игрока (неправильные действия, не хватает ресурсов)  
 public Button save\_game\_button = new Button(); // Кнопка сохранения игры  
 private World world; // Объявление объекта класса World через который проходят все вычисления и основная часть программы  
 static public Group *group* = new Group(); // "Группа" для хранения всех объектов интерфейса JavaFx  
 public static Stage *stage\_reference*; // Референс на переменную stage, нужен для доступа к сцене из методов класса World  
  
 // Создает кнопки-индикаторы (это по сути элемент графики, относящиеся к игровому процессу лишь количеством  
 // ресурсов игроков, а так как объекты Player - статические, можно объявить этот метод в главном классе)  
 public void createIndicators(){  
 // Создание кнопок-индикаторов количества ресурсов  
 *rice\_amount\_indicator*.setLayoutX(stage\_width);  
 *rice\_amount\_indicator*.setLayoutY(100);  
 *rice\_amount\_indicator*.setText("Рис: " + *player*.rice\_amount);  
  
 *water\_amount\_indicator*.setLayoutX(stage\_width);  
 *water\_amount\_indicator*.setLayoutY(200);  
 *water\_amount\_indicator*.setText("Вода: " + *player*.water\_amount);  
  
 *villagers\_amount\_indicator*.setLayoutX(stage\_width);  
 *villagers\_amount\_indicator*.setLayoutY(300);  
 *villagers\_amount\_indicator*.setText("Крестьяне: " + *player*.villagers\_amount);  
  
 *game\_message\_indicator*.setLayoutX(stage\_width);  
 *game\_message\_indicator*.setLayoutY(400);  
 *game\_message\_indicator*.setText("Ваш ход!");  
 *game\_message\_indicator*.setPrefSize(this.user\_menu\_width, 100);  
  
 save\_game\_button.setLayoutX(stage\_width + user\_menu\_width - user\_menu\_width \* 0.7);  
 save\_game\_button.setLayoutY(stage\_height - 75);  
 save\_game\_button.setText("Сохранить игру");  
 save\_game\_button.setPrefSize(user\_menu\_width \* 0.7, 50);  
 save\_game\_button.setOnAction(event -> this.world.saveGame());  
 }  
  
 // Главный метод start, запускающий приложение при запуске main()  
 @Override  
 public void start(Stage stage) {  
 // Создаем референс на переменную stage для доступа к экрану  
 *stage\_reference* = stage;  
  
 // Создаем объект класса Group для хранения всех кнопок  
 *group* = new Group();  
 // Создание главного меню  
 this.createStartMenu();  
  
 // Создаем окно интерфейса  
 Scene scene = new Scene(*group*, this.stage\_width + this.user\_menu\_width, this.stage\_height);  
  
 // Меняем надпись окна  
 stage.setTitle("Conquer Game");  
 stage.setScene(scene);  
 stage.show();  
 }  
  
 private void startNewGame(){  
 // Убираем кнопки главного меню  
 *group*.getChildren().clear();  
  
 this.world = new World(rows\_amount, columns\_amount, null);  
 this.createIndicators();  
  
 // Добавляем все кнопки полей в объект group вложенным циклом  
 for (int i = 0; i < this.rows\_amount; i++) {  
 for (int j = 0; j < this.columns\_amount; j++) {  
 *group*.getChildren().add(world.getGridMember(i, j));  
 }  
 }  
 // Добавляем кнопки - индикаторы  
 *group*.getChildren().add(*rice\_amount\_indicator*);  
 *group*.getChildren().add(*water\_amount\_indicator*);  
 *group*.getChildren().add(*villagers\_amount\_indicator*);  
 *group*.getChildren().add(*game\_message\_indicator*);  
 // Добавляем кнопку сохранения игры  
 *group*.getChildren().add(save\_game\_button);  
 }  
  
 private void createStartMenu(){  
 // Кнопка "Новая игра, создает новое поле и игра начинается с нуля  
 Button new\_game\_button = new Button();  
 new\_game\_button.setLayoutX((stage\_width + user\_menu\_width) / 2.0 - 75);  
 new\_game\_button.setLayoutY(100);  
 new\_game\_button.setPrefSize(150, 75);  
 new\_game\_button.setText("Новая игра");  
 new\_game\_button.setOnAction(event -> startNewGame());  
 *group*.getChildren().add(new\_game\_button);  
  
 // Кнопка "Продолжить игру", загружает выбранный файл с игрой  
 Button save\_game\_button = new Button();  
 save\_game\_button.setLayoutX((stage\_width + user\_menu\_width) / 2.0 - 75);  
 save\_game\_button.setLayoutY(300);  
 save\_game\_button.setPrefSize(150, 75);  
 save\_game\_button.setText("Продолжить игру");  
 save\_game\_button.setOnAction(event -> this.getSaveFile());  
 *group*.getChildren().add(save\_game\_button);  
  
 // Кнопка "Выход", завершает работу программы  
 Button exit\_game\_button = new Button();  
 exit\_game\_button.setLayoutX((stage\_width + user\_menu\_width) / 2.0 - 75);  
 exit\_game\_button.setLayoutY(500);  
 exit\_game\_button.setPrefSize(150, 75);  
 exit\_game\_button.setText("Выход");  
 try {  
 exit\_game\_button.setOnAction(event -> System.*exit*(0));  
 } catch (NullPointerException e){  
 *game\_message\_indicator*.setText("Не удалось загрузить");  
 *group*.getChildren().add(*game\_message\_indicator*);  
 }  
 *group*.getChildren().add(exit\_game\_button);  
 }

Переиспользованный код  
 private void getSaveFile() throws NullPointerException{  
 // Запускаем окно выбора файла  
 FileChooser file\_chooser = new FileChooser();  
 file\_chooser.setTitle("Выберите папку сохранения игры");  
 // Сохраняем выбранную директорию  
 File file = file\_chooser.showOpenDialog(*stage\_reference*);  
 *group*.getChildren().clear();  
 // Создаем новый мир с указанием пути к файлу сохранения  
 this.world = new World(this.rows\_amount, this.columns\_amount, file.getAbsolutePath());  
 // Если произошла ошибка во время считывания файла - программа завершается, интерфейс остается открытым  
 if (*game\_message\_indicator*.getText().startsWith("Невозможно открыть")){  
 return;  
 }  
 *group*.getChildren().clear();  
  
 // Добавляем все кнопки полей в объект group вложенным циклом  
 for (int i = 0; i < this.rows\_amount; i++) {  
 for (int j = 0; j < this.columns\_amount; j++) {  
 *group*.getChildren().add(world.getGridMember(i, j));  
 }  
 }  
 // Создаем кнопки - индикаторы  
 this.createIndicators();  
 // Добавляем кнопки - индикаторы  
 *group*.getChildren().add(*rice\_amount\_indicator*);  
 *group*.getChildren().add(*water\_amount\_indicator*);  
 *group*.getChildren().add(*villagers\_amount\_indicator*);  
 *group*.getChildren().add(*game\_message\_indicator*);  
 // Добавляем кнопку сохранения игры  
 *group*.getChildren().add(save\_game\_button);  
 }  
  
}

Класс World

package com.example.course\_paper;  
  
import javafx.scene.chart.LineChart;  
import javafx.scene.chart.NumberAxis;  
import javafx.scene.chart.XYChart;  
import javafx.stage.DirectoryChooser;  
  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.\*;  
  
public class World {  
 // Создание двумерного массива для хранения объектов класса Field  
 private final Field[][] grid;  
 final private int rows\_amount, columns\_amount;  
 // Значение порядкового номера серединного столбца и ряда  
 private final int average\_column\_amount;  
 private final int average\_rows\_amount;  
 public final int cell\_width = ConquerGame.*cell\_width*;  
  
 public final int cell\_height = ConquerGame.*cell\_height*;  
 // Создание объект класса Random для генерации псевдо-случайных чисел  
 Random random\_integer\_generator = new Random();  
 private final int[][] start\_fields\_i\_j;  
  
 // Массив из булевых значений, каждое определяет возможность ИИ совершить действие  
 // {"Набрать воды", "Полить рис", "Построить дом", "Захватить поле"}  
 private final boolean[] possible\_AI\_moves = {false, false, false, false};  
  
  
 public World(int rows\_amount, int columns\_amount, String load\_file\_path){  
 // Создание двумерного массива для хранения всех объектов класса Field  
 this.grid = new Field[rows\_amount][columns\_amount];  
 this.rows\_amount = rows\_amount;  
 this.columns\_amount = columns\_amount;  
 this.average\_column\_amount = columns\_amount / 2;  
 this.average\_rows\_amount = rows\_amount / 2;  
 start\_fields\_i\_j = new int[][] {{0, 0}, {1, 0}, {0, 1},  
 {rows\_amount - 1, columns\_amount - 1},  
 {rows\_amount - 1, columns\_amount - 2},  
 {rows\_amount - 2, columns\_amount - 1}};  
 if (load\_file\_path != null){  
 try {  
 this.loadGame(this.parseSaveFile(load\_file\_path));  
 } catch (FileNotFoundException e){  
 ConquerGame.*game\_message\_indicator*.setText("Невозможно открыть. Файл не найден.");  
 ConquerGame.*group*.getChildren().add(ConquerGame.*game\_message\_indicator*);  
 e.printStackTrace();  
 } catch (IndexOutOfBoundsException e){  
 ConquerGame.*game\_message\_indicator*.setText("Невозможно открыть. Закройте приложение и удалите этот файл сохранения.");  
 ConquerGame.*group*.getChildren().add(ConquerGame.*game\_message\_indicator*);  
 //e.printStackTrace();  
 }  
 } else {  
 this.generateGrid();  
 }  
 }  
  
  
 // Генерация случайного поля из возможных вариантов {рис; вода; пустое поле}  
 private void generateGrid(){  
 // Присваиваем начальные поля игрокам  
 this.createStartFields();  
 // Заполнение двумерного массива объектов класса Field  
 for (int i = 0; i < this.rows\_amount; i++) {  
 outer\_loop:  
 for (int j = 0; j < this.columns\_amount; j++) {  
 for (int k = 0; k < this.start\_fields\_i\_j.length; k ++) {  
 if (this.start\_fields\_i\_j[k][0] == i && this.start\_fields\_i\_j[k][1] == j){  
 continue outer\_loop;  
 }  
 }  
 // Заполнение случайным полем из списка {0 - пустое; 1 - рис; 2 - вода; 3 - дом} (все кроме 3)  
 this.addField(random\_integer\_generator.nextInt(3), i, j,  
 i \* this.cell\_width, j \* cell\_height, calculateRequired\_villagers\_amount(i, j));  
  
 // Присвоение размера полю grid[i][j]  
 grid[i][j].setPrefSize(this.cell\_width, this.cell\_height);  
  
 if (this.is\_next(i, j, ConquerGame.*player*)){  
 grid[i][j].setDisable(false);  
 } else {  
 grid[i][j].setDisable(true);  
 }  
  
 // Указываем на функцию, которая будет выполняться при нажатии кнопки.  
 Field temp\_field = grid[i][j]; // Создание переменной для передачи в lambda функцию  
 grid[i][j].setOnAction(event -> {  
 handleFieldClick(temp\_field);  
 });  
 }  
 }  
 }  
  
 private int calculateRequired\_villagers\_amount(int i, int j){  
 int temp\_integer;  
 int delta\_i, delta\_j;  
  
 // Присвоение количества требуемых жителей для захвата поля по математической модели  
 // (см отчет "Создание экономической модели")  
 delta\_j = Math.*abs*(this.average\_column\_amount - j);  
 delta\_i = Math.*abs*(this.average\_rows\_amount - i);  
 temp\_integer = Math.*abs*(this.columns\_amount - (delta\_i + delta\_j));  
  
 // Добавляем i и j по мере приближения к центру, чтобы количество требуемых крестьян неубывало  
 if (i + j < this.average\_rows\_amount + this.average\_column\_amount){  
 temp\_integer += (int) Math.*pow*((i + j), 2);  
 } else {  
 temp\_integer += (int) Math.*pow*((this.rows\_amount - i + this.columns\_amount - j - 2), 2);  
 }  
 return temp\_integer;  
 }  
  
 public void addField(int type, int x\_i, int y\_j, int x, int y, int required\_villages\_amount){  
 if (type == 0){  
 grid[x\_i][y\_j] = new EmptyField(type, x\_i, y\_j, x, y, required\_villages\_amount);  
 } else if (type == 1){  
 grid[x\_i][y\_j] = new RiceField(type, x\_i, y\_j, x, y, required\_villages\_amount);  
 } else if (type == 2){  
 grid[x\_i][y\_j] = new WaterField(type, x\_i, y\_j, x, y, required\_villages\_amount);  
 } else {  
 // Таким образом мы обрабатываем поля во время загрузки сохранения игры, заменяя поля с типом "дом" на  
 // пустые поля. Иначе сделать это невозможно, так как после создания подгружаемого массива grid[][] лоигка  
 // программы требует его полное заполнение.  
 grid[x\_i][y\_j] = new EmptyField(type, x\_i, y\_j, x, y, required\_villages\_amount);  
 }  
 }  
  
 public Field getGridMember(int i, int j){  
 return grid[i][j];  
 }  
  
 public boolean is\_next(int x, int y, Player player){  
 for (int i = 0; i < player.x\_fields.size(); i++){  
 if ((player.x\_fields.get(i) == x - 1) && (player.y\_fields.get(i) == y)){  
 return true;  
 } else if ((player.x\_fields.get(i) == x) && (player.y\_fields.get(i) == y - 1)){  
 return true;  
 } else if ((player.x\_fields.get(i) == x + 1) && (player.y\_fields.get(i) == y)){  
 return true;  
 } else if ((player.x\_fields.get(i) == x) && (player.y\_fields.get(i) == y + 1)){  
 return true;  
 } else if ((player.x\_fields.get(i) == x) && (player.y\_fields.get(i) == y)){  
 return true;  
 }  
 }  
 return false;  
 }  
  
 public void refreshGrid(){  
 for (int i = 0; i < this.rows\_amount; i++){  
 for (int j = 0; j < this.columns\_amount; j++){  
 if (this.is\_next(i, j, ConquerGame.*player*)){  
 this.grid[i][j].setDisable(false);  
 }  
 if (this.grid[i][j].owner == ConquerGame.*player* && this.grid[i][j].getType() == 3){  
 this.grid[i][j].setDisable(true);  
 }  
 }  
 }  
 }  
 public void buildHouse(int i, int j, Player player){  
 // Удаляем кнопку пустого поля из массива кнопок group  
 ConquerGame.*group*.getChildren().remove(grid[i][j]);  
 // Создаем новое поле дома  
 HouseField house = new HouseField(3, i, j, i \* this.cell\_width + 1, j \* this.cell\_height + 1, player);  
 // Перезаписываем соответствующее поле в двумерном массиве grid[][]  
 this.grid[i][j] = house;  
 // Присваиваем полю владельца  
 this.grid[i][j].conquer(player, 0);  
 // Добавляем кнопку поля дома в массив из кнопок group  
 ConquerGame.*group*.getChildren().add(this.getGridMember(i, j));  
 // Обновляем доступность полей  
 this.refreshGrid();  
 if (player == ConquerGame.*AIplayer*){  
 this.grid[i][j].setDisable(true);  
 }  
 }  
  
 public void handleFieldClick(Field field\_sample) {  
 // Обработка случая, когда игрок нажал на пустое незахваченное поле  
 if (canConquer(ConquerGame.*player*, field\_sample)){  
 field\_sample.conquer(ConquerGame.*player*,  
 random\_integer\_generator.nextInt(field\_sample.getRequired\_villagers\_amount()));  
 this.endTurn();  
 }  
 // Обработка случая, когда игрок нажал на свое поле риса (полив всего риса)  
 else if (canWaterRice(ConquerGame.*player*, field\_sample) && !field\_sample.isUnoccupied()){  
 // Полить весь рис  
 this.waterRice(ConquerGame.*player*);  
 this.endTurn();  
 }  
 // Обработка случая, когда игрок нажал на свое пустое поле (построить дом)  
 else if (canBuildHouse(ConquerGame.*player*, field\_sample)) {  
 this.buildHouse(field\_sample.getXI(), field\_sample.getYJ(), ConquerGame.*player*);  
 this.endTurn();  
 }  
 // Обработка случая, когда игрок нажал на свое поле с водой (набрать воды)  
 else if (isPlayersWaterField(ConquerGame.*player*, field\_sample)){  
 ConquerGame.*player*.collectWater();  
 this.endTurn();  
 }  
 }  
  
 // Возвращает true если данный игрок может захватить данное поле  
 private boolean canConquer(Player player, Field field\_sample){  
 if (field\_sample.isUnoccupied() && (player.villagers\_amount >= field\_sample.getRequired\_villagers\_amount()) &&  
 (player.rice\_amount >= field\_sample.getRequired\_villagers\_amount())){  
 return true;  
 } else if (player.villagers\_amount < field\_sample.getRequired\_villagers\_amount() && player == ConquerGame.*player*){  
 ConquerGame.*game\_message\_indicator*.setText("Недостаточно крестьян!");  
 return false;  
 } else if (player.rice\_amount < field\_sample.getRequired\_villagers\_amount() && player == ConquerGame.*player*){  
 ConquerGame.*game\_message\_indicator*.setText("Недостаточно риса!");  
 return false;  
 } else {  
 return false;  
 }  
 }  
 private boolean canWaterRice(Player player, Field field\_sample){  
 if ((field\_sample.owner == player && (field\_sample.getType() == 1)) && player.water\_amount > 0){  
 return true;  
 } else if (player.water\_amount == 0 && player == ConquerGame.*player*){  
 ConquerGame.*game\_message\_indicator*.setText("У вас нет воды!");  
 return false;  
 }  
 else {  
 return false;  
 }  
 }  
  
 private boolean isPlayersWaterField(Player player, Field field\_sample){  
 if ((field\_sample.owner == player && (field\_sample.getType() == 2))){  
 return true;  
 } else {  
 return false;  
 }  
 }  
  
 private boolean canBuildHouse(Player player, Field field\_sample){  
 // Если поле принадлежит игроку и это поле - пустое (EmptyField)  
 if (field\_sample.owner == player && field\_sample.getType() == 0){  
 // Если не хватает риса  
 if (player.rice\_amount < player.house\_rice\_cost){  
 // Если метод вызван игроком, то выводим сообщение о том, что для постройки дома не хватает риса  
 if (player == ConquerGame.*player*) {  
 ConquerGame.*game\_message\_indicator*.setText("Не хватает риса!");  
 }  
 return false;  
 }  
 // Если не хватает жителей  
 else if (player.villagers\_amount < player.house\_villager\_cost){  
 // Если метод вызван игроком, то выводим сообщение о том, что для постройки дома не хватает крестьян  
 if (player == ConquerGame.*player*) {  
 ConquerGame.*game\_message\_indicator*.setText("Не хватает крестьян!");  
 }  
 return false;  
 }  
 // Всего хватает  
 else {  
 return true;  
 }  
 } else {  
 return false;  
 }  
 }  
  
 public void createStartFields(){  
 for (int k = 0; k < this.start\_fields\_i\_j.length; k++){  
 // Временные переменные для читаемости кода  
 int i = this.start\_fields\_i\_j[k][0];  
 int j = this.start\_fields\_i\_j[k][1];  
  
 // Создаем нейтральное поле  
 this.addField(k % 3, i, j, i \* this.cell\_width, j \* this.cell\_height, 0);  
  
 // Присваиваем поле игроку или компьютеру  
 if (i < 3 ) {  
 grid[i][j].conquer(ConquerGame.*AIplayer*, 0);  
 } else {  
 grid[i][j].conquer(ConquerGame.*player*, 0);  
 }  
  
 // Выставляем размеры и текст кнопки (для JavaFX)  
 grid[i][j].setPrefSize(this.cell\_width, this.cell\_height);  
 grid[i][j].setText("");  
  
 // Создание переменной для передачи в lambda функцию  
 Field temp\_field = grid[i][j];  
 // Присваиваем функцию, которая будет запускаться при нажатии кнопки.  
 grid[i][j].setOnAction(event -> {  
 handleFieldClick(temp\_field);  
 });  
  
 // Включаем кнопки игрока, выключаем кнопки ИИ  
 if (i > 2) {  
 grid[i][j].setDisable(false);  
 } else {  
 grid[i][j].setDisable(true);  
 }  
 }  
 }  
 private void endTurn(){  
 if (ConquerGame.*player*.overall\_fields\_amount >= (this.rows\_amount \* this.columns\_amount / 2 - 1)) {  
 ConquerGame.*group*.getChildren().clear();  
 ConquerGame.*game\_message\_indicator*.setText("Вы выиграли!!!");  
 ConquerGame.*game\_message\_indicator*.autosize();  
 ConquerGame.*group*.getChildren().add(ConquerGame.*game\_message\_indicator*);  
 this.showEndgameGraphs(ConquerGame.*player*.story\_of\_rice, ConquerGame.*AIplayer*.story\_of\_rice,  
 "Номер хода", "Количество риса", "Мониторинг количества риса", 0);  
  
 this.showEndgameGraphs(ConquerGame.*player*.story\_of\_villagers, ConquerGame.*AIplayer*.story\_of\_villagers,  
 "Номер хода", "Количество крестьян", "Мониторинг количества крестьян",  
 this.columns\_amount \* cell\_height / 3);  
  
 this.showEndgameGraphs(ConquerGame.*player*.story\_of\_fields, ConquerGame.*AIplayer*.story\_of\_fields,  
 "Номер хода", "Количество полей", "Мониторинг количества полей",  
 this.columns\_amount \* cell\_height / 3 \* 2);  
 return;  
 } else if (ConquerGame.*AIplayer*.overall\_fields\_amount >= (this.rows\_amount \* this.columns\_amount / 2 - 1)){  
 ConquerGame.*group*.getChildren().clear();  
 ConquerGame.*game\_message\_indicator*.setText("Вы проиграли...");  
 ConquerGame.*game\_message\_indicator*.autosize();  
 ConquerGame.*group*.getChildren().add(ConquerGame.*game\_message\_indicator*);  
 this.showEndgameGraphs(ConquerGame.*player*.story\_of\_rice, ConquerGame.*AIplayer*.story\_of\_rice,  
 "Номер хода", "Количество риса", "Мониторинг количества риса", 0);  
  
 this.showEndgameGraphs(ConquerGame.*player*.story\_of\_villagers, ConquerGame.*AIplayer*.story\_of\_villagers,  
 "Номер хода", "Количество крестьян", "Мониторинг количества крестьян",  
 this.columns\_amount \* cell\_height / 3);  
  
 this.showEndgameGraphs(ConquerGame.*player*.story\_of\_fields, ConquerGame.*AIplayer*.story\_of\_fields,  
 "Номер хода", "Количество полей", "Мониторинг количества полей",  
 this.columns\_amount \* cell\_height / 3 \* 2);  
 return;  
 }  
 for (int i = 0; i < this.rows\_amount; i++){  
 for (int j = 0; j < this.columns\_amount; j++){  
 // Создание временной переменно в теле цикла чтобы не обращаться к ней каждый раз через двумерный массив  
 Field temp\_field = this.grid[i][j];  
 // Даем рис, если  
 if (temp\_field.getType() == 1){  
 temp\_field.giveRice();  
 } else if (temp\_field.getType() == 3) {  
 temp\_field.spawnVillager();  
 }  
 }  
 }  
 this.handleAITurn();  
  
 refreshGrid();  
 refreshUI();  
 refreshPlayersStory();  
  
 }  
 public void waterRice(Player player){ // Небольшое уточнение, слово water здесь идет как глагол, по всем правилам английского  
 int temp\_V = player.villagers\_amount;  
 int temp\_R = player.rice\_fields\_amount;  
 int x = 0;  
  
 while (x < player.x\_fields.size() && player.water\_amount > 0 && temp\_V > 0 && temp\_R > 0){  
 Field temp\_field = this.getGridMember(player.x\_fields.get(x), player.y\_fields.get(x));  
 if (temp\_field.getType() == 1 && temp\_field.owner == player){  
 temp\_field.waterRiceField();  
 temp\_V--;  
 temp\_R--;  
 player.water\_amount--;  
 }  
 x++;  
 }  
 }  
  
 private void handleAITurn(){  
 // Обнуляем массив с возможными ходами  
 Arrays.*fill*(this.possible\_AI\_moves, false);  
  
 // Проходим по всем полям  
 List<Field> fields\_possible\_to\_conquer = new ArrayList<>();  
 Field empty\_owned\_field = this.grid[0][0]; // Иначе java считает, что переменная может быть не инициализирована  
 for (int i = 0; i < this.rows\_amount; i++) {  
 for (int j = 0; j < this.columns\_amount; j++) {  
 // Объявляем временные локальные переменные  
 Field temp\_field = this.getGridMember(i, j);  
 if (temp\_field.isUnoccupied()){  
 // Проверка можно ли завоевать поле  
 if (this.is\_next(i, j, ConquerGame.*AIplayer*) && this.canConquer(ConquerGame.*AIplayer*, temp\_field)){  
 this.possible\_AI\_moves[3] = true;  
 // Добавляем в список всех полей, которые может завоевать ИИ в данный ход  
 fields\_possible\_to\_conquer.add(temp\_field);  
 }  
 } else {  
 if (temp\_field.owner == ConquerGame.*AIplayer*) {  
 // Проверка можно ли полить рис  
 if (temp\_field.getType() == 1 && this.canWaterRice(ConquerGame.*AIplayer*, temp\_field)){  
 this.possible\_AI\_moves[1] = true;  
 }  
 // Проверка можно ли набрать воды (вода есть всегда, набрать ее можно всегда, поэтому  
 // простая проверка на тип переменной  
 else if (temp\_field.getType() == 2){  
 this.possible\_AI\_moves[0] = true;  
 }  
 // Проверка можно ли построить дом  
 else if (temp\_field.getType() == 0 && this.canBuildHouse(ConquerGame.*AIplayer*, temp\_field)){  
 this.possible\_AI\_moves[2] = true;  
 empty\_owned\_field = temp\_field;  
 }  
 }  
 }  
 }  
 }  
  
 // Решение какой ход будет выполнен (см отчет "Блок схема выбора хода ИИ)  
 if (this.possible\_AI\_moves[3]){  
 Field chosen\_field = fields\_possible\_to\_conquer.get(0);  
 int temp\_min = 100;  
 for (int k = 0; k < fields\_possible\_to\_conquer.size(); k++){  
 Field temp\_field = fields\_possible\_to\_conquer.get(k);  
 if (temp\_field.getRequired\_villagers\_amount() < temp\_min){  
 chosen\_field = temp\_field;  
 temp\_min = temp\_field.getRequired\_villagers\_amount();  
 }  
 }  
 // ХОД 3: завоевать территорию  
 chosen\_field.conquer(ConquerGame.*AIplayer*,  
 random\_integer\_generator.nextInt(chosen\_field.getRequired\_villagers\_amount()));  
 } else if (this.possible\_AI\_moves[2]) {  
 // ХОД 2: построить дом  
 this.buildHouse(empty\_owned\_field.getXI(), empty\_owned\_field.getYJ(), ConquerGame.*AIplayer*);  
 } else if (this.possible\_AI\_moves[1]){  
 // ХОД 1: полить рис  
 this.waterRice(ConquerGame.*AIplayer*);  
 } else if (this.possible\_AI\_moves[0]){  
 // ХОД 0: набрать воды  
 ConquerGame.*AIplayer*.collectWater();  
 }  
 }  
 private void refreshUI(){  
 ConquerGame.*rice\_amount\_indicator*.setText("Рис: " + ConquerGame.*player*.rice\_amount);  
 ConquerGame.*villagers\_amount\_indicator*.setText("Крестьяне: " + ConquerGame.*player*.villagers\_amount);  
 ConquerGame.*water\_amount\_indicator*.setText("Вода: " + ConquerGame.*player*.water\_amount);  
 ConquerGame.*game\_message\_indicator*.setText("Ваш ход!");  
  
 }  
  
 private void refreshPlayersStory(){  
 ConquerGame.*player*.refreshStory();  
 ConquerGame.*AIplayer*.refreshStory();  
 }  
  
 public void saveGame(){  
 // Запускаем выбор директории с помощью javafx.stage.DirectoryChooser  
 DirectoryChooser directory\_chooser = new DirectoryChooser();  
 directory\_chooser.setTitle("Выберите папку для сохранения игры");  
 // Сохраняем выбранную директорию  
 File directory = directory\_chooser.showDialog(ConquerGame.*stage\_reference*);  
 // Считываем все файлы директории для нумерации нового сохранения  
 File[] files\_list = directory.listFiles();  
 assert files\_list != null;  
 int temp\_int = 1;  
 for (File file : files\_list) {  
 String file\_name = file.getName();  
 if (file\_name.startsWith("Conquer Game сохранение ")){  
 temp\_int++;  
 }  
 }  
  
 // Создаем путь к новому файлу  
 String new\_file\_path = directory.getAbsolutePath() + "\\Conquer Game сохранение " + Integer.*toString*(temp\_int) + ".txt";  
 System.*out*.println(new\_file\_path);  
 //File save\_file = new File(new\_file\_path);  
 // Создаем .txt файл и записываем в него все данные для сохранения  
 try {  
 FileWriter writer = new FileWriter(new\_file\_path);  
 writer.write(this.createSaveLog());  
 writer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 private String createSaveLog(){  
 // Создаем переменную типа StringBuilder для сбора данных об игровом поле в одну строку  
 StringBuilder save\_log = new StringBuilder();  
  
 // Сбор данных о типах полей  
 for (int i = 0; i < this.rows\_amount; i++){  
 for (int j = 0; j < this.columns\_amount; j++){  
 save\_log.append(this.grid[i][j].getType()).append(" ");  
 }  
 save\_log.append("\n");  
 }  
 save\_log.deleteCharAt(save\_log.length() - 1);  
 // Конец блока сбора данных о типах полей  
 save\_log.append("block\_divider\n");  
  
 // Сбор данных о полях во владениях игрока  
 for (int i = 0; i < ConquerGame.*player*.x\_fields.size(); i++){  
 save\_log.append(ConquerGame.*player*.x\_fields.get(i)).append(" ");  
 }  
 save\_log.append("\n");  
 for (int i = 0; i < ConquerGame.*player*.y\_fields.size(); i++){  
 save\_log.append(ConquerGame.*player*.y\_fields.get(i)).append(" ");  
 }  
  
 save\_log.append("block\_divider\n");  
  
 // Сбор данных о полях во владениях ИИ  
 for (int i = 0; i < ConquerGame.*AIplayer*.x\_fields.size(); i++){  
 save\_log.append(ConquerGame.*AIplayer*.x\_fields.get(i)).append(" ");  
 }  
 save\_log.append("\n");  
 for (int i = 0; i < ConquerGame.*AIplayer*.y\_fields.size(); i++){  
 save\_log.append(ConquerGame.*AIplayer*.y\_fields.get(i)).append(" ");  
 }  
  
 save\_log.append("block\_divider\n");  
  
 // Сбор данных о ресурсах игрока  
 save\_log.append(ConquerGame.*player*.rice\_amount).append(" ");  
 save\_log.append(ConquerGame.*player*.water\_amount).append(" ");  
 save\_log.append(ConquerGame.*player*.villagers\_amount).append(" ");  
 save\_log.append("block\_divider\n");  
 // Сбор данных о ресурсах игрока  
 save\_log.append(ConquerGame.*AIplayer*.rice\_amount).append(" ");  
 save\_log.append(ConquerGame.*AIplayer*.water\_amount).append(" ");  
 save\_log.append(ConquerGame.*AIplayer*.villagers\_amount).append(" ");  
 return save\_log.toString();  
 }  
  
 private String parseSaveFile(String load\_file\_path) throws FileNotFoundException {  
 // Считываем файл с помощью класса Scanner для которого нужна обработка ошибки "Файл не найден"  
 File save\_file = new File(load\_file\_path);  
 Scanner file\_scanner = new Scanner(save\_file);  
 StringBuilder save\_log\_string = new StringBuilder();  
 while (file\_scanner.hasNextLine()){  
 save\_log\_string.append(file\_scanner.nextLine()).append("\n");  
 }  
 file\_scanner.close();  
 return save\_log\_string.toString();  
 }  
  
 private void loadGame(String save\_log) throws IndexOutOfBoundsException{  
 // Разделение данных сохранения на блоки  
 String[] save\_log\_blocks = save\_log.split("block\_divider\n");  
  
 // Загрузка типов полей каждого поля  
 String[] field\_types\_row = save\_log\_blocks[0].split("\n");  
 String[][] fields\_types = new String[field\_types\_row.length][field\_types\_row.length];  
 for (int i = 0; i < field\_types\_row.length; i++){  
 fields\_types[i] = field\_types\_row[i].split(" ");  
 if (fields\_types[i].length != this.columns\_amount){  
 throw new IndexOutOfBoundsException();  
 }  
 }  
 if (fields\_types.length != this.rows\_amount){  
 throw new IndexOutOfBoundsException();  
 }  
  
 // Заполняем массив grid[][] объектами класса Field (аналогично методу generateGrid)  
 for (int i = 0; i < fields\_types.length; i++) {  
 for (int j = 0; j < fields\_types.length; j++) {  
 // Заполнение полем  
 this.addField(Integer.*parseInt*(fields\_types[i][j]), i, j,  
 i \* this.cell\_width, j \* cell\_height, calculateRequired\_villagers\_amount(i, j));  
  
 // Присвоение размера полю grid[i][j]  
 grid[i][j].setPrefSize(this.cell\_width, this.cell\_height);  
  
 if (this.is\_next(i, j, ConquerGame.*player*)){  
 grid[i][j].setDisable(false);  
 } else {  
 grid[i][j].setDisable(true);  
 }  
  
 // Указываем на функцию, которая будет выполняться при нажатии кнопки.  
 Field temp\_field = grid[i][j]; // Создание переменной для передачи в lambda функцию  
 grid[i][j].setOnAction(event -> {  
 handleFieldClick(temp\_field);  
 });  
 }  
 }  
  
 // Загрузка всех данных о полях во владении игрока  
 String[] x\_fields = save\_log\_blocks[1].split("\n")[0].split(" ");  
 String[] y\_fields = save\_log\_blocks[1].split("\n")[1].split(" ");  
 for (int i = 0; i < x\_fields.length; i++ ){  
 if (this.grid[Integer.*parseInt*(x\_fields[i])][Integer.*parseInt*(y\_fields[i])].getType() == 3){  
 this.buildHouse(Integer.*parseInt*(x\_fields[i]), Integer.*parseInt*(y\_fields[i]), ConquerGame.*player*);  
 } else {  
 this.grid[Integer.*parseInt*(x\_fields[i])][Integer.*parseInt*(y\_fields[i])].conquer(ConquerGame.*player*, 0);  
 }  
  
 }  
  
 x\_fields = save\_log\_blocks[2].split("\n")[0].split(" ");  
 y\_fields = save\_log\_blocks[2].split("\n")[1].split(" ");  
 for (int i = 0; i < x\_fields.length; i++ ){  
 if (this.grid[Integer.*parseInt*(x\_fields[i])][Integer.*parseInt*(y\_fields[i])].getType() == 3){  
 this.buildHouse(Integer.*parseInt*(x\_fields[i]), Integer.*parseInt*(y\_fields[i]), ConquerGame.*AIplayer*);  
 } else {  
 this.grid[Integer.*parseInt*(x\_fields[i])][Integer.*parseInt*(y\_fields[i])].conquer(ConquerGame.*AIplayer*, 0);  
 }  
 }  
  
  
 // Загрузка данных о ресурсах игрока  
 int rice\_amount = 0;  
 int water\_amount = 0;  
 int villagers\_amount = 0;  
 rice\_amount = Integer.*parseInt*(save\_log\_blocks[3].split(" ")[0]);  
 water\_amount = Integer.*parseInt*(save\_log\_blocks[3].split(" ")[1]);  
 villagers\_amount = Integer.*parseInt*(save\_log\_blocks[3].split(" ")[2]);  
 ConquerGame.*player*.rice\_amount = rice\_amount;  
 ConquerGame.*player*.water\_amount = water\_amount;  
 ConquerGame.*player*.villagers\_amount = villagers\_amount;  
  
 // Загрузка данных о ресурсах ИИ  
 rice\_amount = Integer.*parseInt*(save\_log\_blocks[4].split(" ")[0]);  
 water\_amount = Integer.*parseInt*(save\_log\_blocks[4].split(" ")[1]);  
 villagers\_amount = Integer.*parseInt*(save\_log\_blocks[4].split(" ")[2]);  
 ConquerGame.*AIplayer*.rice\_amount = rice\_amount;  
 ConquerGame.*AIplayer*.water\_amount = water\_amount;  
 ConquerGame.*AIplayer*.villagers\_amount = villagers\_amount;  
  
 this.refreshGrid();  
 }

**Переиспользованный код (источник указан в п. «Использованные ресурсы»**

private void showEndgameGraphs(List<Integer> player\_story\_of\_variable, List<Integer> AI\_story\_of\_variable,  
 String xAxisName, String yAxisName, String graph\_name, int y\_pos){  
 NumberAxis xAxis = new NumberAxis();  
 NumberAxis yAxis = new NumberAxis();  
  
 // Присваивание названия оси абсцисс  
 xAxis.setLabel(xAxisName);  
 // Присваивание названия оси ординат  
 yAxis.setLabel(yAxisName);  
  
 // Создание объекта графика (чарта)  
 LineChart<Number,Number> lineChart = new LineChart<Number,Number>(xAxis,yAxis);  
  
 // Присваивание названия оси ординат  
 lineChart.setTitle(graph\_name);  
 // Создание отдельного графика игрока  
 XYChart.Series series = new XYChart.Series();  
 // присвоение имени отдельному графику игрока  
 series.setName("Игрок");  
 // Заполнение графика данными  
 for (int i = 0; i < player\_story\_of\_variable.size(); i++){  
 series.getData().add(new XYChart.Data(i, player\_story\_of\_variable.get(i)));  
  
 }  
  
 // Создание отдельного графика ИИ  
 XYChart.Series series1 = new XYChart.Series();  
 // присвоение имени отдельному графику ИИ  
 series1.setName("ИИ");  
 // Заполнение графика данными  
 for (int i = 0; i < AI\_story\_of\_variable.size(); i++){  
 series1.getData().add(new XYChart.Data(i, AI\_story\_of\_variable.get(i)));  
  
 }  
 lineChart.getData().add(series);  
 lineChart.getData().add(series1);  
 lineChart.setPrefSize(this.rows\_amount \* cell\_width, this.columns\_amount \* cell\_height / 3.0);  
 lineChart.setLayoutY(10 + y\_pos);  
 ConquerGame.*group*.getChildren().add(lineChart);  
 }  
}

Класс Field

package com.example.course\_paper;  
  
import javafx.scene.control.Button;  
  
// Класс Поле -  
public class Field extends Button {  
 // Переменная type хранит значение Типа Поля - {0 - пустое; 1 - рис; 2 - вода; 3 - дом}  
 final private int type;  
 private int required\_villagers\_amount;  
 private int x\_pos, y\_pos;  
 private boolean is\_occupied = false;  
 public Player owner = null; //  
 private int x\_i, y\_j;  
  
 public Field(int type\_value, int i, int j, int x, int y){ // Присваивает Полю значение  
 super();  
 this.type = type\_value;  
 // Присвоение текстуры  
 this.setTexture();  
 // Присвоение порядковых номеров i и j нахождения в двумерном массиве  
 this.setRelativePos(i, j);  
 // Присвоение координат x и y полю grid[i][j] для правильного отображения  
 this.setGraphicsPos(x, y);  
  
  
 // Присвоение координат x и y кнопке grid[i][j] для правильного отображения (для JavaFX)  
 this.setLayoutX(x\_pos);  
 this.setLayoutY(y\_pos);  
 }  
  
 public int getType(){ // Возвращает значение типа поля  
 return this.type;  
 }  
  
 // Функция для присваивания количества требуемых крестьян для захвата  
 public void setRequired\_villagers\_amount(int required\_villagers\_amount\_value){  
 this.required\_villagers\_amount = required\_villagers\_amount\_value;  
 }  
  
 public int getRequired\_villagers\_amount(){ // Возвращает количество требуемых крестьян для захвата  
 return this.required\_villagers\_amount;  
 }  
  
 public void setGraphicsPos(int x, int y){ // Присваивает полю координаты x и y (координаты по пикселям)  
 this.x\_pos = x;  
 this.y\_pos = y;  
 }  
 public int getX(){ // Возвращает x координату поля  
 return this.x\_pos;  
 }  
 public int getY(){ // Возвращает y координату поля  
 return this.y\_pos;  
 }  
  
 public void conquer(Player player, int villagers\_amount\_decrease){  
 this.is\_occupied = true;  
 this.owner = player;  
 player.give\_field(this.x\_i, this.y\_j);  
 this.setText("");  
 // Если данное поле - поле воды или поле риса - увеличиваем количество соответствующих полей данного игрока  
 if (this.getType() == 1){  
 player.rice\_fields\_amount++;  
 } else if (this.getType() == 2){  
 player.water\_fields\_amount++;  
 }  
 if (villagers\_amount\_decrease != 0) {  
 // Увеличиваем общее количество полей во владении данного игрока  
 player.overall\_fields\_amount++;  
 // Уменьшаем количество риса игрока  
 player.rice\_amount -= this.required\_villagers\_amount;  
 // Уменьшаем количество жителей на случайную величину, сгенерированную в прошлой функции  
 player.villagers\_amount -= villagers\_amount\_decrease;  
 }  
 // Меняет цвет если поле принадлежит ИИ  
 if (player == ConquerGame.*AIplayer*) {  
 this.setStyle("-fx-background-color:#b786f7;");  
 } else if (player == ConquerGame.*player*) {  
 this.setStyle("-fx-background-color:#8bfcc9;");  
 }  
 }  
  
 public boolean isUnoccupied(){  
 return !this.is\_occupied;  
 }  
  
 public void setRelativePos(int i, int j){  
 this.x\_i = i;  
 this.y\_j = j;  
 }  
 public int getXI(){  
 return this.x\_i;  
 }  
 public int getYJ(){  
 return this.y\_j;  
 }  
  
 public void setTexture(){  
 }  
 public void giveRice(){  
 }  
 public void waterRiceField(){  
 }  
 public void spawnVillager() {  
 }  
}

Класс Player

package com.example.course\_paper;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Player {  
 public int rice\_amount;  
 public int water\_amount;  
 public int villagers\_amount;  
 public int houses\_amount = 0;  
 public int house\_rice\_cost = 1;  
 public int house\_villager\_cost = 1;  
 public int rice\_fields\_amount = 0;  
 public int water\_fields\_amount = 0;  
 public int overall\_fields\_amount = 3;  
 private int house\_rice\_cost\_increment = 1;  
  
 public List<Integer> x\_fields = new ArrayList<Integer>();  
 public List<Integer> y\_fields = new ArrayList<Integer>();  
 // Используется для хранения данных о том, как менялось количество риса на протяжении игры  
 public List<Integer> story\_of\_rice = new ArrayList<Integer>();  
 public List<Integer> story\_of\_villagers = new ArrayList<Integer>();  
 public List<Integer> story\_of\_fields = new ArrayList<Integer>();  
  
  
 public Player(){  
 rice\_amount = 0;  
 water\_amount = 0;  
 villagers\_amount = 1;  
 }  
 public void give\_field(int x, int y){  
 this.x\_fields.add(x);  
 this.y\_fields.add(y);  
 }  
 public void addHouse(){  
 this.houses\_amount++;  
 // Уменьшаем количество ресурсов на соответствующие стоимости постройки дома  
 this.rice\_amount -= this.house\_rice\_cost;  
 this.villagers\_amount -= this.house\_villager\_cost;  
  
 this.increaseHouseCost();  
 }  
 public void increaseHouseCost(){  
 // Рост цены постройки дома (требуемый рис) равноускоренно  
 this.house\_rice\_cost += this.house\_rice\_cost\_increment;  
 // Рост "ускорения роста" линейно  
 this.house\_rice\_cost\_increment++;  
 // Рост цены постройки дома (требуемые крестьяне) линейно  
 this.house\_villager\_cost++;  
 }  
  
 public void collectWater(){  
 if (this.water\_fields\_amount <= this.villagers\_amount){  
 this.water\_amount += this.water\_fields\_amount;  
 } else {  
 this.water\_amount += this.villagers\_amount;  
 }  
 }  
  
 public void refreshStory(){  
 this.story\_of\_rice.add(this.rice\_amount);  
 this.story\_of\_villagers.add(this.villagers\_amount);  
 this.story\_of\_fields.add(this.overall\_fields\_amount);  
 }  
}

Дочерний класс RiceField (extends Field)

package com.example.course\_paper;  
  
import javafx.scene.control.ContentDisplay;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
  
public class RiceField extends Field{  
 private boolean is\_just\_watered = false;  
 // Используется для подсчета оставшихся ходов, когда  
 private int wet\_level = 0;  
  
 public RiceField(int type\_value, int i, int j, int x, int y, int required\_villagers\_amount){  
 super(type\_value, i, j, x, y);  
 super.setRequired\_villagers\_amount(required\_villagers\_amount);  
  
 // Изменяем надпись на значение "сложности" захвата каждого поля  
 this.setText(Integer.*toString*(required\_villagers\_amount));  
 this.setContentDisplay(ContentDisplay.*TOP*);  
 }  
  
 @Override  
 public void setTexture(){  
 ImageView rice\_view = new ImageView(new Image("C:\\Users\\Кирилл\\Desktop\\course\_paper\\rice.png"));  
 rice\_view.setFitHeight(ConquerGame.*cell\_height* / 2.0);  
 rice\_view.setFitWidth(ConquerGame.*cell\_width* / 2.0);  
 rice\_view.setPreserveRatio(true);  
 rice\_view.setX(this.getX());  
 rice\_view.setY(this.getY());  
 rice\_view.autosize();  
 this.setGraphic(rice\_view);  
 }  
 @Override  
 public void giveRice(){  
 if (this.owner == null){  
 return;  
 }  
 // Это условие позволяет не давать игроку больше риса в конце хода за только что политый рис. Реализация  
 // достаточно странная, однако она позволяет  
 if (this.is\_just\_watered){  
 this.is\_just\_watered = false;  
 this.owner.rice\_amount++;  
 } else if (this.wet\_level > 0){  
 this.owner.rice\_amount += 2;  
 this.wet\_level--;  
 } else{  
 this.owner.rice\_amount++;  
 }  
 }  
 @Override  
 public void waterRiceField(){  
 if (this.wet\_level == 0) {  
 this.is\_just\_watered = true;  
 }  
 this.wet\_level = 2;  
 }  
}

Дочерний класс HouseField (extends Field)

package com.example.course\_paper;  
  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
  
public class HouseField extends Field{  
 public HouseField(int type\_value, int i, int j, int x, int y, Player owner) {  
 super(type\_value, i, j, x, y);  
 this.owner = owner;  
 owner.addHouse();  
 this.setPrefSize(ConquerGame.*cell\_width*, ConquerGame.*cell\_height*);  
 }  
  
 @Override  
 public void setTexture(){  
 ImageView house\_view = new ImageView(new Image("C:\\Users\\Кирилл\\Desktop\\course\_paper\\house.png"));  
 house\_view.setFitHeight(ConquerGame.*cell\_height* / 2.0);  
 house\_view.setFitWidth(ConquerGame.*cell\_width* / 2.0);  
 house\_view.setPreserveRatio(true);  
 house\_view.setX(this.getX());  
 house\_view.setY(this.getY());  
 house\_view.autosize();  
 this.setGraphic(house\_view);  
 }  
 @Override  
 public void spawnVillager(){  
 this.owner.villagers\_amount++;  
 }  
}

Дочерний класс EmptyField (extends Field)

package com.example.course\_paper;  
  
import javafx.scene.control.ContentDisplay;  
  
public class EmptyField extends Field{  
 public EmptyField(int type\_value, int i, int j, int x, int y, int required\_villagers\_amount){  
 super(type\_value, i, j, x, y);  
 super.setRequired\_villagers\_amount(required\_villagers\_amount);  
  
 // Изменяем надпись на значение "сложности" захвата каждого поля  
 this.setText(Integer.*toString*(required\_villagers\_amount));  
 this.setContentDisplay(ContentDisplay.*TOP*);  
 }  
}

Дочерний класс WaterField (extends Field)

package com.example.course\_paper;  
  
import javafx.scene.control.ContentDisplay;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
  
public class WaterField extends Field{  
 public WaterField(int type\_value, int i, int j, int x, int y, int required\_villagers\_amount) {  
 super(type\_value, i, j, x, y);  
 super.setRequired\_villagers\_amount(required\_villagers\_amount);  
  
 // Изменяем надпись на значение "сложности" захвата каждого поля  
 this.setText(Integer.*toString*(required\_villagers\_amount));  
 this.setContentDisplay(ContentDisplay.*TOP*);  
 }  
  
 @Override  
 public void setTexture(){  
 ImageView water\_view = new ImageView(new Image("C:\\Users\\Кирилл\\Desktop\\course\_paper\\water.png"));  
 water\_view.setFitHeight(ConquerGame.*cell\_height* / 2.0);  
 water\_view.setFitWidth(ConquerGame.*cell\_width* / 2.0);  
 water\_view.setPreserveRatio(true);  
 water\_view.setX(this.getX());  
 water\_view.setY(this.getY());  
 water\_view.autosize();  
 this.setGraphic(water\_view);  
 }  
}

9. Ссылка на репозиторий GitHub

<https://github.com/ROFLMAOL0L/course_paper.git>

10. Список использованных источников

1. Alla Redko; «JavaFX FileChooser»[электронный ресурс]; <https://docs.oracle.com/javafx/2/ui_controls/file-chooser.htm>; xx..09.2013; (дата обращения 16.12.2023); [Переиспользованный код](#Переиспользованный_код2)

2. Jakob Jenkov; «JavaFX DirectoryChooser»[электронный ресурс]; <https://jenkov.com/tutorials/javafx/directorychooser.html>; 24.01.2019; (дата обращения 16.12.2023)

3. Rob Gravelle; «JavaFx documentation» [электронный ресурс]; <https://www.techrepublic.com/article/java-directory-navigation/>; 14.07.2023; (последняя дата обращения 16.12.2023)

4. Jakob Jenkov; «JavaFX LineChart» [электронный ресурс]; <https://jenkov.com/tutorials/javafx/linechart.html>; 28.05.2016; (дата обращения 16.12.2023); [**Переиспользованный код**](#Переиспользованный_код)

5. Jakob Jenkov; «JavaFX Button» [электронный ресурс]; <https://jenkov.com/tutorials/javafx/button.html>; 09.12.2020; (дата обращения 13.12.2023); [Переиспользованный код](#Переиспользованный_код3)