МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Забайкальский государственный университет» (ФГБОУ ВО «ЗабГУ»)

Энергетический Факультет

Кафедра информатики вычислительной техники и прикладной математики

# КУРСОВАЯ РАБОТА

По дисциплине: Программирование На тему: Игра «Овца и волки»

Выполнил студент группы ИВТ-17(ивт) Бочкарев Никита Сергеевич

Руководитель работы: Доцент кафедры ИВТ и ПМ Соловьев Владимир Александрович

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Забайкальский государственный университет» (ФГБОУ ВО «ЗабГУ»)

Энергетический Факультет

Кафедра информатики вычислительной техники и прикладной математики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

По дисциплине: Программирование

На тему: Игра «Овца и волки»

Выполнил студент группы ИВТ-17(ивт) Бочкарев Никита Сергеевич

Руководитель работы: Доцент кафедры ИВТ и ПМ Соловьев Владимир Александрович

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Забайкальский государственный университет» (ФГБОУ ВО «ЗабГУ»)

Факультет Энергетический

Кафедра Информатики, вычислительной техники и прикладной математики

ЗАДАНИЕ

на курсовую работу

По дисциплине: Программирование

Студенту: Бочкареву Никите Сергеевичу

Специальности: Информатика и вычислительная техника

1. Тема курсовой работы: Игра «Овца и волки»

2. Срок подачи студентом законченной работы:

3. Перечень подлежащих разработке в курсовой работе вопросов: Процесс разработки, основные алгоритмы и руководство пользователя игры «Овца и волки»

Дата выдачи задания:

Руководитель курсовой работы /Соловьев В. А./

Задание принял к исполнению:

Подпись студента /Бочкарев Н. С./

# РЕФЕРАТ

Пояснительная записка – 30 с., 3 источника.

ИГРА, ЛОГИКА, ДОСКА, КЛЕТКИ, ГЕЙМПЛЕЙ, JAVA, КЛАСС, МОДУЛЬ, ПАКЕТ.

В данной работе рассматривается процесс разработки, основные алгоритмы и руководство пользователя игры «Овца и Волки».

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc52354185)

[1 Постановка задачи 7](#_Toc52354186)

[2 Диаграмма модулей 8](#_Toc52354187)

[3 Программная реализация 10](#_Toc52354188)

[3.1 Package GUI 10](#_Toc52354189)

[3.2 Package Logic 13](#_Toc52354190)

[3.3 Описание алгоритма ИИ 14](#_Toc52354191)

[3.4 Испытания программы 15](#_Toc52354192)

[4 Руководство пользователя 15](#_Toc52354193)

[ЗАКЛЮЧЕНИЕ 17](#_Toc52354194)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 18](#_Toc52354195)

[Приложение А 19](#_Toc52354196)

[Приложение Б 26](#_Toc52354197)

# ВВЕДЕНИЕ

В современном мире компьютерные игры занимают большое место в сфере развлечений. Они делятся на множество различных жанров, среди которых любой найдет себе игру по душе, будь то игра для спокойного отдыха, веселого времяпровождения или для серьезного соревнования. С каждым годом количество игроков и выпускаемых игр растет.

Однако многие и не задумываются о том, как сделаны игры, что у них «внутри» и как они работают. Разработка большой игры – серьезная задача и требует совместных усилий большого количества людей из разных сфер деятельности. Программирование же компьютерных игр также непростая задача, потому что зачастую требует не только прекрасное знание программных средств, но и глубокие знания теории алгоритмов и структур данных. В этой курсовой работе рассмотрена разработка логической игры «Овца и волки». При этом упор сделан на разработку искусственного интеллекта для данной игры.

Для написания игры использовался язык Java с комплектом разработчика Java Development Kit 10.0.2.

# 1 Постановка задачи

Выбрать наиболее подходящий алгоритм и написать Искусственный Интеллект (далее ИИ) на основе выбранного алгоритма для логической игры «Овца и волки». Реализовать графический интерфейс для игры и возможность играть в нее вдвоем.

Игра «Овца и волки» идёт на шахматной доске (только по темным клеткам), шашки могут ходить по диагонали на ближайшую незанятую клетку — волки только вперед, овца вперёд и назад. В начале игры овца и волки расставлены по краям доски друг против друга, овца ходит первой.

Цель овцы — дойти до противоположного края доски, в этом случае она выиграла партию. Волки выигрывают, если им удаётся окружить овцу или прижать её к краю, так что она не может больше ходить.

# Диаграмма модулей

Для удобства программирования, приложение разбито на два модуля – GUI и Logic. Модуль GUI отвечает только за визуальное отображение игры и «ничего не знает» про логику работы ИИ, проверку ходов на верность, условиях победы и т. д. За все вышеперечисленное отвечает модуль Logic.

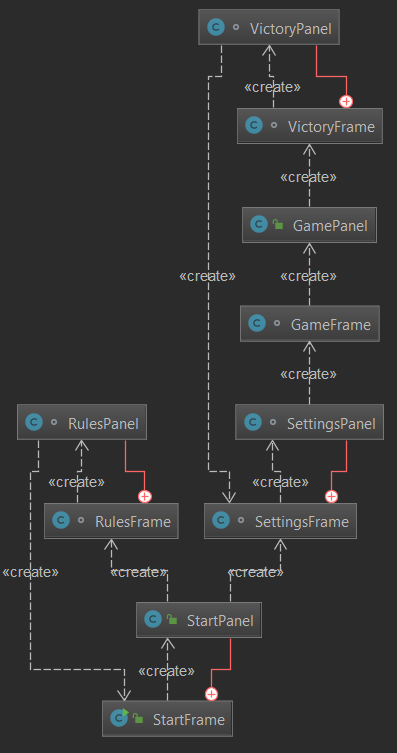
****

Рисунок 1 -Диаграмма классов из Package GUI

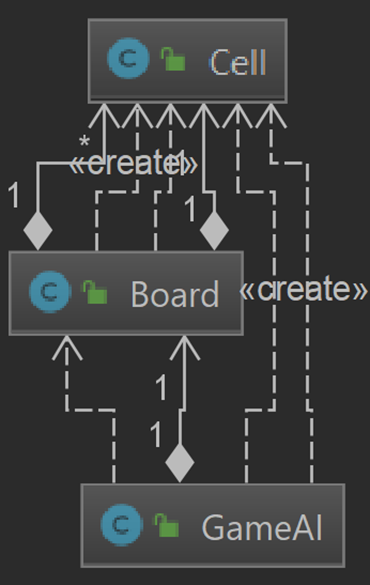


Рисунок 2 - Диаграмма классов из

Package Logic

# Программная реализация

# 3.1 Package GUI

Классы из package GUI отвечают за визуальное отображение игры и меню для задания параметров игры. Рассмотрим каждый класс по отдельности. Код представлен в Приложении А.

* **Класс StartFrame** отвечает за создание окна с включенной в него панелью класса StartPanel, который является внутренним классом для класса StartFrame. Панель содержит 3 кнопки, первая “Начать игру” предлагает пользователю начать игру, вторая “Правила” дает возможность пользователю ознакомиться с правилами игры и третья “Выход” позволяет пользователю выйти из игры. При нажатии кнопки “Начать игру” стартовое окно закрывается и вызывается окно класса SettingsFrame. При нажатии кнопки “Правила” стартовое окно закрывается и вызывается окно класса RulesFrame.
* **Класс RulesFrame** отвечает за создание окна с, включенной в него, панелью класса RulesPanel, который является внутренним классом для класса RulesFrame. Панель содержит текст с кратким описанием правил игры, а также кнопку “Назад”, при нажатии на которую окно с правилами закрывается и вызывается окно класса StartFrame.

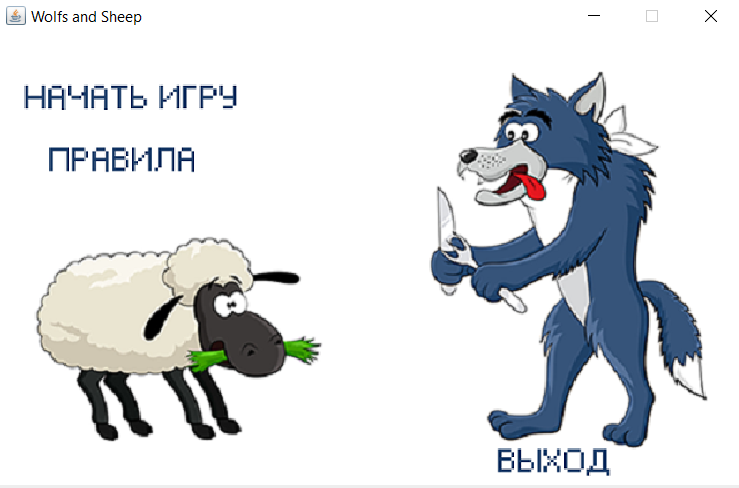


Рисунок 3 - Окно класса StartFrame

* **Класс SettingsFrame** отвечает за создание окна с, включенной в него, панелью класса SettingsPanel, который является внутренним классом для класса SettingsFrame. Панель содержит 3 радиокнопки, которые предлагают пользователю выбрать один из трех вариантов игры: “Играть за волков”, “Игрок против игрока”, “Играть за овцу”, также на панели имеется выпадающий список отвечающий за выбор уровня сложности и кнопка “Начать игру”, при нажатии на которую проверяется все ли опции выбрал пользователь, если оказалось, что пользователь не выбрал не одну из радиокнопок, то появляется сообщение предлагающее сделать выбор, в случае же когда пользователь выбрал все параметры, вызывается окно класса GameFrame в него передаются все данные, выбранные пользователем, после чего окно с настройками игры закрывается.
* **Класс GameFrame** отвечает за создание окна с включенной в него панелью класса GamePanel и передачу в нее данных об опциях игры.

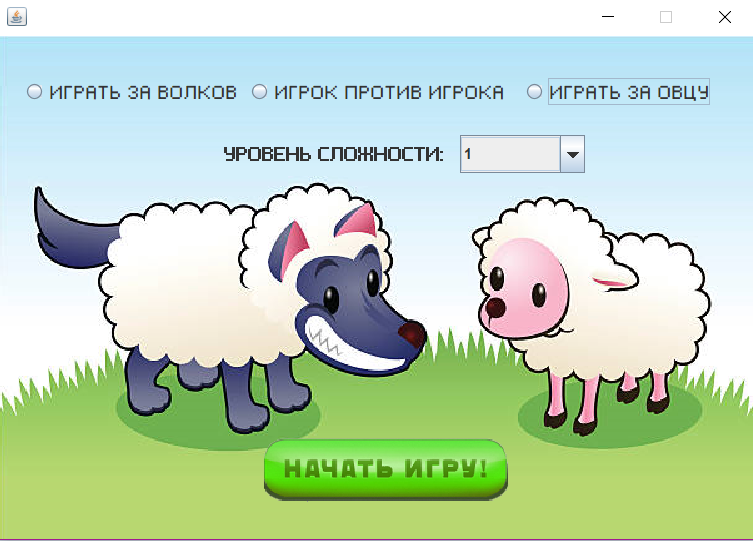


Рисунок 4 - Окно класса SettingsFrame

* **Класс GamePanel** отвечает за создание основной игровой панели вкоторой и происходит игра. Для обеспечения игрового процесса полями этого класса являются board класса Board и gameAI класса GameAI, также в других полях содержится информация о том, за какую сторону играет игрок, а за какую ИИ. Класс содержит метод onMouseClicked() который вызывается при клике пользователем на панель. В данном методе анализируется точка клика и в зависимости от координат клика происходит одно из следующих действий: если игрок кликнул на клетку, занимаемую той стороной, за которую он сейчас играет и, при этом, на данный момент нет активной клетки, то эта клетка становится активной; если игрок кликнул на пустую клетку, то, если уже есть активная клетка, анализируется корректность хода из активной в клетку, на которую кликнул игрок, если данный ход корректный, то он совершается, если нет, то ничего не происходит. После хода совершенного игроком, если выбран вариант игры против ИИ, совершается ход ИИ, если выбран вариант игры игрок против игрока, ход переходит другой стороне. При этом после каждого хода анализируется состояние доски и в случае победы одной из сторон вызывается окно класса VictoryFrame куда передаются данные, о том, кто победил овца или волки, после этого игровое окно закрывается.
* **Класс VictoryFrame** отвечает за создание окна с, включенной в него, панелью класса VictoryPanel, который является внутренним классом для класса VictoryFrame. Панель содержит сообщение оповещающее, какая из сторон победила, две кнопки “Начать заново” и “Выход”. При нажатии на “Начать заново” вызывается окно класса SettingsFrame, после чего окно VictoryFrame закрывается. При нажатии на “Выход” игра закрывается.

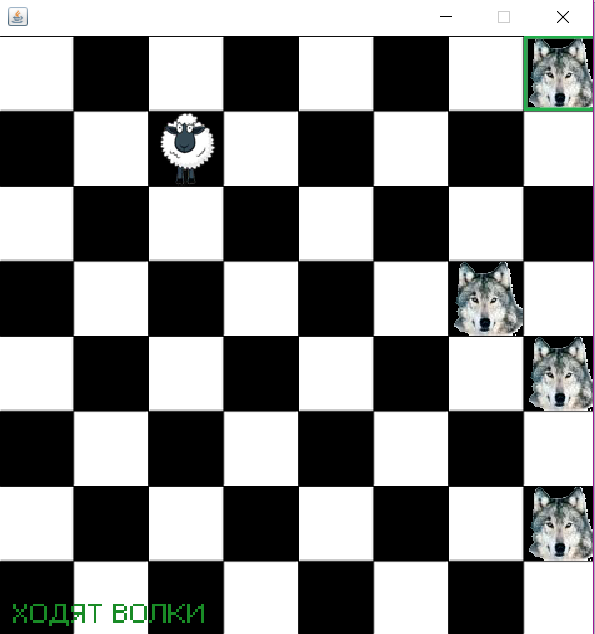


Рисунок 5 - Окно класса GameFrame

# 3.2 Package Logic

Классы из package Logic отвечают за ИИ и внутреннюю логику самой игры. Рассмотрим каждый класс по отдельности. Код представлен в Приложении Б.

* **Класс Cell** реализует клетку игрового поля. Хранит внутри себя координаты клетки, а также методы для работы с клетками которые позволяют: получить множество доступных ходов для данной клетки, проверить корректность некоторого хода, определить является ли данная клетка победной для овцы. Также реализованы методы toString(), hashcode() и equals(), которые используются соответственно для строкового представления клетки и для корректного сравнения клеток.
* **Класс Board** реализует игровое поле. Внутри себя хранит клетку, в которой располагается овца, и множество клеток занятых волками. В классе находятся методы для совершения хода, проверки некоторой клетки на пустоту или на нахождение в этой клетки овцы или волка, проверки победы одной из сторон, а также методы позволяющие получить множество доступных ходов для данной клетки, проверить корректность некоторого хода, определить является ли данная клетка победной для овцы, которые перенаправляются в аналогичные методы для клетки. Кроме того имеются методы для очистки доски и установления овцы и волков в определенные клетки, данные методы в самом игровом процессе не используются и необходимы для тестирования работы программы.
* **Класс GameAl** реализует ИИ для игры. Для того чтобы иметь доступ к состоянию доски внутри себя хранит игровое поле и уровень данного ИИ. Содержит методы для выбора лучшего хода, который работает на основе минимакса с альфа-бета отсечением. Подробнее алгоритм описан ниже.

# 3.3 Описание алгоритма ИИ

При выборе оптимального хода для ИИ, используется минимакс с альфа-бета отсечением. Данный алгоритм был выбран, потому что он хорошо подходит для игры двух лиц с нулевой суммой (т. е. игры, в которой участвуют два игрока, выигрыши которых противоположны) в случае последовательных ходов. Минимакс - это алгоритм, который перебирает все возможные ходы на определенное количество ходов вперед (в моем приложении за эту глубину перебора отвечает уровень ИИ) и при этом каждому ходу дает оценку, которая строиться с помощью эвристики. Причем чем больше эта оценка, тем лучше текущее положение для волков, чем меньше, тем лучше положение для овцы. Основывается эта оценка на том, сколько ходов нужно сделать овце, чтобы победить при неподвижных волках, если же овца в данной ситуации вообще не может достичь одной из победных клеток, то берется значение 100 - количество клеток, до которых овца в принципе может дойти. Далее стандартный алгоритм минимакса без модификаций после получения оценок для всех возможных ходов, выбирает наилучшую для данной стороны и совершает ход. Однако получение оценок для каждого хода, при достаточно большой глубине просчета и большом количестве возможных ходов, может занять значительное время, поэтому в моем приложении используется такая модификация минимакса, как альфа-бета отсечение. Альфа-бета отсечение вводит две дополнительные переменные alpha и beta, где alpha — текущее максимальное значение, меньше которого волки никогда не выберут, а beta — текущее минимальное значение, больше которого овца никогда не выберет. Как только условие alpha > beta станет верным, что означает конфликт ожиданий, мы прерываем анализ ходов следующих за текущим и возвращаем последнюю полученную оценку на текущем уровне глубины просчета ходов. Данный механизм позволяет не просчитывать полностью оценки для некоторых ходов, оценка которых будет заведомо хуже, чем полученная раннее, что позволяет существенно сократить время работы минимакса даже при значительной глубине просчета ходов.

# 3.4 Испытания программы

Для программы написаны тесты, проверяющие метод для совершения хода, метод для получения множества доступных ходов, метод проверки корректности некоторого хода. Также написаны тесты проверяющие корректность значения эвристики для некоторых состояний доски. Присутствуют тесты проверяющие поведение ИИ в разных ситуациях, для этого игровое поле устанавливается в определенное состояние, после чего вызывается метод отвечающий за ход ИИ и проверяется сделал ли ИИ наиболее логичный и выгодный для себя ход. Кроме того в ручном режиме был протестирован графический интерфейс приложения, всевозможный некорректный ввод, а также различные ситуации при совершении ходов.

**Покрытие классов из Package Logic тестами**

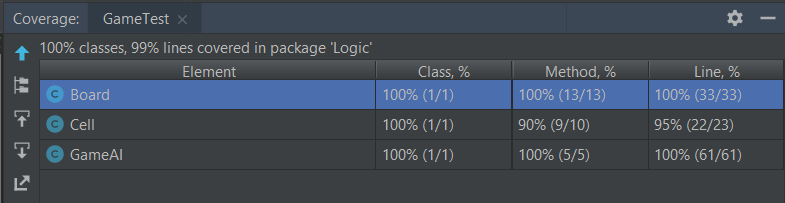


Рисунок6 - Покрытие классов из Package Logic тестами

Как видно из таблицы, все методы классов, входящих в Package Logic были задействованы при тестировании, кроме одного – это метод toString() класса Cell.

# Руководство пользователя

1. Навигация

В главном меню присутствуют кнопки для перехода в окно выбора уровня для игры, окно правил игры и выхода из игры.

В окне правил можно ознакомиться с правилами игры и вернуться в главное меню.

В окне выбора уровня можно выбрать вариант игры: игрок против игрока или игрок против ИИ. Во втором случае можно выбрать уровень ИИ.

После выбора опций запускается окно игры где и происходит основной геймплей до момента победы одной из сторон после чего появляется окно результата.

Окно результата содержит информацию о победившей стороне и кнопки предлагающие повторить игру или же выйти из игры.

1. Геймплей

Геймплей состоит в управлении овцой или же волками в зависимости от выбранной стороны. Для совершения хода необходимо нажать сначала на ту шашку которой хотите сходить, после чего нажать на клетку куда хотите сходить. При этом волки ходят по диагонали только вперед, а овца по диагонали в любую сторону.

# ЗАКЛЮЧЕНИЕ

Подводя итог, работа над данным приложением - это интересный и полезный опыт для меня. Хотя сама игра не является крайне сложной со стратегической точки зрения и рассчитана в основном на детей, было интересно и местами довольно трудно реализовать ИИ для нее. Процесс написания данной программы позволил мне в полной мере разобраться и до конца понять алгоритм минимакса с альфа-бета отсечением, что, несомненно, является полезным знанием, так как этот алгоритм, пусть и не в чистом виде, но может быть применим и к более сложным играм, наподобие шашек или шахмат, да и вообще почти к любой логической пошаговой игре. Из минусов можно отметить простоватое графическое оформление игры.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. КСПТ: Курс “ Алгоритмы и структуры данных” <http://kspt.icc.spbstu.ru/course/algorithms>
2. Интернет-энциклопедия Wikipedia <https://en.wikipedia.org/wiki/Minimax>
3. Минимакс на примере игры в зайца и волков <https://habr.com/post/146088/>

# Приложение А

**Класс StartFrame**

package GUI;

import javax.swing.\*;

import java.awt.\*;

public class StartFrame extends JFrame {

StartFrame(String s) {

super(s);

setSize(610, 402);

setResizable(false);

setLocationRelativeTo(null);

setContentPane(new StartPanel());

setVisible(true);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

}

public class StartPanel extends JPanel {

StartPanel() {

setLayout(null);

JButton newGame = initButton(new ImageIcon("files/buttons/newgame.png"), 20, 40);

newGame.addActionListener(e -> onNewGame());

JButton rules = initButton(new ImageIcon("files/buttons/rules.png"), 40, 90);

rules.addActionListener(e -> onRules());

JButton exit = initButton(new ImageIcon("files/buttons/exit.png"), 400, 330);

exit.addActionListener(e -> onExit());

setVisible(true);

}

private JButton initButton(ImageIcon icon, int x, int y) {

JButton button = new JButton(icon);

button.setBorderPainted(false);

button.setSize(button.getIcon().getIconWidth(), button.getIcon().getIconHeight());

button.setLocation(x, y);

add(button);

return button;

}

private void onRules() {

new RulesFrame();

JFrame frame = (JFrame) SwingUtilities.getWindowAncestor(this);

frame.dispose();

}

private void onNewGame() {

new SettingsFrame();

JFrame frame = (JFrame) SwingUtilities.getWindowAncestor(this);

frame.dispose();

}

private void onExit() {

System.exit(0);

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

Image back = new ImageIcon("files/backgrounds/startback.png").getImage();

g.drawImage(back, 0, 0, null);

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> new StartFrame("Wolfs and Sheep"));

}

}

**Класс RulesFrame**

package GUI;

import javax.swing.\*;

import java.awt.\*;

class RulesFrame extends JFrame {

RulesFrame() {

setSize(610, 420);

setResizable(false);

setLocationRelativeTo(null);

setContentPane(new RulesPanel());

setVisible(true);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

}

class RulesPanel extends JPanel {

RulesPanel() {

setLayout(null);

JButton back = new JButton(new ImageIcon("files/buttons/back.png"));

back.setBorderPainted(false);

back.setSize(back.getIcon().getIconWidth(), back.getIcon().getIconHeight());

back.setContentAreaFilled(false);

back.setLocation(20, 270);

back.addActionListener(e -> {

new StartFrame("Wolfs and Sheep");

JFrame frame = (JFrame) SwingUtilities.getWindowAncestor(this);

frame.dispose();

});

add(back);

setVisible(true);

}

@Override

public void paintComponent(Graphics g) {

Image background = new ImageIcon("files/backgrounds/rulesback.png").getImage();

g.drawImage(background, 0, 0, null);

}

}

}

**Класс SettingsFrame**

package GUI;

import javax.swing.\*;

import java.awt.\*;

class SettingsFrame extends JFrame {

SettingsFrame() {

setSize(620, 440);

setResizable(false);

setLocationRelativeTo(null);

setContentPane(new SettingsPanel());

setVisible(true);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

}

class SettingsPanel extends JPanel {

private JRadioButton but1, but2, but3;

private JComboBox<Integer> comboBox;

private ButtonGroup bg = new ButtonGroup();

SettingsPanel() {

setLayout(null);

Integer[] levels = {1, 2, 3, 4, 5, 6, 7, 8};

comboBox = new JComboBox<>(levels);

comboBox.setLocation(370, 80);

comboBox.setSize(100, 30);

add(comboBox);

but1 = new JRadioButton("Играть за овцу");

but2 = new JRadioButton("Играть за волков");

but3 = new JRadioButton("Игрок против игрока");

initRadioButton(but1, 200, 420);

initRadioButton(but2, 180, 20);

initRadioButton(but3, 220, 200);

JButton ok = new JButton(new ImageIcon("files/buttons/startgame.png"));

ok.setBorderPainted(false);

ok.setSize(ok.getIcon().getIconWidth(), ok.getIcon().getIconHeight());

ok.setContentAreaFilled(false);

ok.setLocation(210, 320);

ok.addActionListener(e -> onPressOK());

add(ok);

}

private void initRadioButton(JRadioButton button, int width, int x) {

button.setFont(new Font("PixelForce", Font.PLAIN, 14));

button.setContentAreaFilled(false);

button.setSize(width, 50);

button.setLocation(x, 20);

bg.add(button);

add(button);

}

private void onPressOK() {

if (!but1.isSelected() && !but2.isSelected() && !but3.isSelected()) {

JOptionPane.showMessageDialog(this, "Пожалуйста выберите вариант игры ");

} else {

boolean isPlayingForSheep = but1.isSelected() || but3.isSelected();

boolean isPlayingForWolfs = but2.isSelected() || but3.isSelected();

int level = comboBox.getItemAt(comboBox.getSelectedIndex());

new GameFrame(isPlayingForSheep, isPlayingForWolfs, level);

JFrame frame = (JFrame) SwingUtilities.getWindowAncestor(this);

frame.dispose();

}

}

@Override

public void paintComponent(Graphics g) {

Image back = new ImageIcon("files/backgrounds/settingsback.png").getImage();

g.drawImage(back, 0, 0, null);

setFont(new Font("PixelForce", Font.PLAIN, 14));

g.drawString("Уровень сложности: ", 180, 100);

}

}

}

**Класс GameFrame**

package GUI;

import javax.swing.\*;

class GameFrame extends JFrame {

GameFrame(boolean isPlayingForSheep, boolean isPlayingForWolfs, int level) {

setSize(490, 518);

setLocationRelativeTo(null);

setResizable(false);

setContentPane(new GamePanel(isPlayingForSheep, isPlayingForWolfs, level));

setVisible(true);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

}

}

**Класс GamePanel**

package GUI;

import Logic.Board;

import Logic.Cell;

import Logic.GameAI;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

public class GamePanel extends JPanel {

private Board board;

private GameAI gameAI;

private Cell currentCell;

private boolean isHumanPlayingForSheep;

private boolean isHumanPlayingForWolfs;

private boolean isStepOfSheep;

private final static int CELL\_WIDTH = 60;

GamePanel(boolean isHumanPlayingForSheep, boolean isHumanPlayingForWolfs, int level) {

this.isHumanPlayingForSheep = isHumanPlayingForSheep;

this.isHumanPlayingForWolfs = isHumanPlayingForWolfs;

isStepOfSheep = isHumanPlayingForSheep;

this.setLayout(null);

setSize(480, 480);

setVisible(true);

board = new Board();

gameAI = new GameAI(board, level);

if (!isHumanPlayingForSheep) gameAI.step(true);

this.addMouseListener(new MouseListener() {

public void mousePressed(MouseEvent e) { }

public void mouseClicked(MouseEvent e) {

onMouseClicked(e.getPoint());

}

public void mouseReleased(MouseEvent e) { }

public void mouseEntered(MouseEvent e) { }

public void mouseExited(MouseEvent e) { }

});

}

private void onMouseClicked(Point clickedPoint) {

Cell clickedCell = new Cell(clickedPoint.x / CELL\_WIDTH, clickedPoint.y / CELL\_WIDTH);

if (!board.isItEmptyCell(clickedCell) && (board.isItSheepCell(clickedCell) && isStepOfSheep && isHumanPlayingForSheep

|| board.isItWolfCell(clickedCell) && !isStepOfSheep && isHumanPlayingForWolfs)) {

currentCell = clickedCell;

}

if (currentCell != null && board.isItCorrectStep(currentCell, clickedCell)) {

board.step(currentCell, clickedCell);

if (isHumanPlayingForSheep && isHumanPlayingForWolfs) isStepOfSheep = !isStepOfSheep;

currentCell = null;

if (!isHumanPlayingForSheep) gameAI.step(true);

if (!isHumanPlayingForWolfs) gameAI.step(false);

if (board.isSheepWin()) onEndOfGame(true);

if (board.isWolfWin()) onEndOfGame(false);

}

GamePanel.super.repaint();

}

private void onEndOfGame(boolean isSheepWin) {

JFrame frame = (JFrame) SwingUtilities.getWindowAncestor(this);

frame.dispose();

new VictoryFrame(isSheepWin);

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

Image sheepImage = new ImageIcon("files/images/sheep.png").getImage();

Image wolfImage = new ImageIcon("files/images/wolf.png").getImage();

Image back = new ImageIcon("files/images/board.png").getImage();

Image currentImage = new ImageIcon("files/images/current.png").getImage();

g.drawImage(back, 0, 0, null);

g.drawImage(sheepImage, board.sheepCell.i \* CELL\_WIDTH, board.sheepCell.j \* CELL\_WIDTH, null);

for (Cell cell : board.wolfsCells) {

g.drawImage(wolfImage, cell.i \* CELL\_WIDTH, cell.j \* CELL\_WIDTH, null);

}

if (currentCell != null) {

g.drawImage(currentImage, currentCell.i \* CELL\_WIDTH, currentCell.j \* CELL\_WIDTH, null);

}

g.setColor(new Color(25, 139, 38));

g.setFont(new Font("PixelForce", Font.PLAIN, 20));

g.drawString(isStepOfSheep ? "Ходит овца" : "Ходят волки", 10, 470);

}

}

**Класс VictoryFrame**

package GUI;

import javax.swing.\*;

import java.awt.\*;

class VictoryFrame extends JFrame {

VictoryFrame(boolean isSheepWin) {

setSize(605, 407);

setResizable(false);

setLocationRelativeTo(null);

setContentPane(new VictoryPanel(isSheepWin));

setVisible(true);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

}

class VictoryPanel extends JPanel {

boolean isSheepWin;

VictoryPanel(boolean isSheepWin) {

this.isSheepWin = isSheepWin;

this.setLayout(null);

JButton restart = new JButton();

restart.setContentAreaFilled(false);

restart.setBorderPainted(false);

if (isSheepWin) {

restart.setSize(265, 30);

restart.setLocation(20, 330);

} else {

restart.setSize(295, 30);

restart.setLocation(10, 330);

}

restart.addActionListener(e -> {

new SettingsFrame();

dispose();

});

add(restart);

JButton exit = new JButton();

exit.setContentAreaFilled(false);

exit.setBorderPainted(false);

if (isSheepWin) {

exit.setSize(115, 30);

exit.setLocation(467, 330);

} else {

exit.setSize(130, 30);

exit.setLocation(410, 333);

}

exit.addActionListener(e -> System.exit(0));

add(exit);

}

@Override

public void paintComponent(Graphics g) {

Image back = new ImageIcon(isSheepWin ? "files/backgrounds/sheepwin.png"

: "files/backgrounds/wolfswin.png").getImage();

g.drawImage(back, 0, 0, null);

}

}

}

# Приложение Б

**Класс Cell**

package Logic;

import java.util.HashSet;

import java.util.Objects;

import java.util.Set;

public class Cell {

public int i;

public int j;

@Override

public String toString() {

return "Cell{" +

"i=" + i +

", j=" + j +

'}';

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Cell cell = (Cell) o;

return i == cell.i &&

j == cell.j;

}

@Override

public int hashCode() {

return Objects.hash(i, j);

}

public Cell(int i, int j) {

this.i = i;

this.j = j;

}

Set<Cell> getAvaliableSteps(boolean isItSheep) {

Set<Cell> result = new HashSet<>();

for (int di = -1; di <= 1; di += 2) {

for (int dj = -1; dj <= 1; dj += 2) {

Cell cell = new Cell(i + di, j + dj);

if (isItCorrectStep(cell, isItSheep)) result.add(cell);

}

}

return result;

}

boolean isItCorrectStep(Cell newCell, boolean isItSheep) {

if (isOutOfBoard(newCell)) return false;

if (isItSheep) return isItCorrectStepForSheep(newCell);

else return isItCorrectStepForWolf(newCell);

}

private boolean isItCorrectStepForSheep(Cell newCell) {

return ((Math.abs(newCell.i - i) == 1 && Math.abs(newCell.j - j) == 1));

}

private boolean isItCorrectStepForWolf(Cell newCell) {

return (newCell.i - i == -1 && Math.abs(newCell.j - j) == 1);

}

private static boolean isOutOfBoard(Cell cell) {

return (cell.i > 7 || cell.i < 0 || cell.j > 7 || cell.j < 0);

}

boolean isItSheepWinCell() {

return (i == 7);

}

}

**Класс Board**

package Logic;

import java.util.HashSet;

import java.util.Set;

public class Board {

public Set<Cell> wolfsCells = new HashSet<>();

public Cell sheepCell;

public Board() {

sheepCell = new Cell(0, 3);

wolfsCells.add(new Cell(7, 0));

wolfsCells.add(new Cell(7, 4));

wolfsCells.add(new Cell(7, 6));

wolfsCells.add(new Cell(7, 2));

}

public void setSheep(Cell cell) {

sheepCell = cell;

}

public void setWolf(Cell cell) {

wolfsCells.add(cell);

}

public void clear() {

wolfsCells.clear();

sheepCell = null;

}

public void step(Cell start, Cell end) {

if (start.equals(sheepCell)) sheepCell = end;

else {

wolfsCells.remove(start);

wolfsCells.add(end);

}

}

public boolean isItEmptyCell(Cell cell) {

return !(isItSheepCell(cell) || isItWolfCell(cell));

}

public boolean isItSheepCell(Cell cell) {

return sheepCell.equals(cell);

}

public boolean isItWolfCell(Cell cell) { return wolfsCells.contains(cell); }

public boolean isSheepWin() {

return sheepCell.isItSheepWinCell();

}

public boolean isWolfWin() {

for (Cell cell : sheepCell.getAvaliableSteps(true)) {

if (!wolfsCells.contains(cell)) return false;

}

return true;

}

public Set<Cell> getAvaliableSteps(Cell start, boolean isItSheep) {

Set<Cell> result = new HashSet<>();

start.getAvaliableSteps(isItSheep).forEach(cell -> {

if (!sheepCell.equals(cell) && !wolfsCells.contains(cell)) result.add(cell);

});

return result;

}

public boolean isItCorrectStep(Cell start, Cell end) {

return isItEmptyCell(end) && start.isItCorrectStep(end, isItSheepCell(start));

}

}

**Класс GameAI**

package Logic;

import java.util.HashSet;

import java.util.Set;

public class GameAI {

private int level;

private Board board;

public GameAI(Board board, int level) {

this.board = board;

this.level = level;

}

public void step(boolean isItStepOfSheepSheep) {

runMinMax(isItStepOfSheepSheep, 0, Integer.MIN\_VALUE, Integer.MAX\_VALUE);

}

/\*\* Возвращает оценку текущего состояния доски: от 0 до 100. При этом, чем ближе оценка к 0, тем лучше

\* текущее положение для овцы, чем ближе оценка к 100, тем лучше текущее положение для волков.

\* Для нахождения оценки проводиться поиск в глубину, чтобы найти наикратчайший путь овцы к одной из

\* победных клеток, при бездействии волков, в случае если такой путь существует функция возвращает длину

\* данного пути. Если же такого пути нет функция возвращает значение 100 - количество клеток, до которых

\* овца в принципе может дойти.

\*/

public int getHeuristic() {

Set<Cell> visitedCells = new HashSet<>();

Set<Cell> candidates = new HashSet<>();

visitedCells.add(board.sheepCell);

candidates.add(board.sheepCell);

int heuristic = 0;

if (board.sheepCell.isItSheepWinCell()) return heuristic;

while (!candidates.isEmpty()) {

heuristic++;

Set<Cell> newCandidates = new HashSet<>();

candidates.forEach(cell -> board.getAvaliableSteps(cell, true).forEach(avaliableStep -> {

if (!visitedCells.contains(avaliableStep)) newCandidates.add(avaliableStep);

}));

for (Cell cell : newCandidates) {

if (cell.isItSheepWinCell()) return heuristic;

}

visitedCells.addAll(newCandidates);

candidates = new HashSet<>(newCandidates);

}

return 100 - visitedCells.size();

}

public int runMinMax(boolean isItSheep, int recursionLevel, int alpha, int beta) {

int curHeuristic;

Cell bestStep = new Cell(0, 0);

Cell bestStepFor = new Cell(0, 0);

if (recursionLevel >= level) {

return getHeuristic();

}

int bestHeuristic = isItSheep ? 100 : 0;

if (isItSheep) {

for (Cell newCell : board.getAvaliableSteps(board.sheepCell, true)) {

Cell oldCell = board.sheepCell;

board.step(oldCell, newCell);

curHeuristic = runMinMax(!isItSheep, recursionLevel + 1, alpha, beta);

if (curHeuristic <= bestHeuristic) {

if (curHeuristic < bestHeuristic || newCell.i > bestStep.i) {

bestHeuristic = curHeuristic;

bestStep = newCell;

bestStepFor = oldCell;

}

}

board.step(newCell, oldCell);

beta = Math.min(curHeuristic, beta);

if (alpha > beta) break;

}

} else {

Set<Cell> wolfsPositionsCopy = new HashSet<>(board.wolfsCells);

for (Cell oldCell : wolfsPositionsCopy) {

for (Cell newCell : board.getAvaliableSteps(oldCell, false)) {

board.step(oldCell, newCell);

curHeuristic = runMinMax(!isItSheep, recursionLevel + 1, alpha, beta);

if (curHeuristic > bestHeuristic) {

bestHeuristic = curHeuristic;

bestStep = newCell;

bestStepFor = oldCell;

}

board.step(newCell, oldCell);

alpha = Math.max(curHeuristic, alpha);

if (alpha > beta) break;

}

}

}

if (recursionLevel == 0) {

board.step(bestStepFor, bestStep);

}

return bestHeuristic;

}

}