Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники

Кафедра «Программное обеспечение автоматизированных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе**

по дисциплине «Объектно-ориентированный анализ и программирование»

на тему: «Проектирование ~~и реализация~~ программы с использованием объектно-ориентированного подхода»

(индивидуальное задание – вариант № 1, подвариант № 3)

Студент: Дудкин Д. М.

Группа: ПрИн-367

Работа зачтена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ «       »                            20\_\_ г.

Руководитель проекта, нормоконтроллер \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Литовкин Д. В.

Волгоград 2020 г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники

Направление 09.03.04 «Программная инженерия»   
Кафедра «Программное обеспечение автоматизированных систем»

Дисциплина «Объектно-ориентированный анализ и программирование»

Утверждаю

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_ Орлова Ю.А.

**Задание**

**на курсовую работу**

Студент: Дудкин Д. М.

Группа: ПрИн-367

1. Тема: «Проектирование ~~и реализация~~ программы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант № 1, подвариант № 3)

Утверждена приказом от «24» января 2020г. № 101-ст

2. Срок представления работы к защите « 04 »   июня  2020 г.

3. Содержание пояснительной записки:

формулировка задания, требования к программе, структура программы, типовые процессы в программе, человеко-машинное взаимодействие, код программы и модульных тестов

4. Перечень графического материала:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Дата выдачи задания «13» февраля 2020 г.

Руководитель проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Литовкин Д.В.

Задание принял к исполнению: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дудкин Д. М.

«      »                         2020 г.

**Содержание**

1 Формулировка задания 4

2 Нефункциональные требования 4

3 Первая итерация разработки 5

3.1 Формулировка упрощенного варианта задания 5

3.2 Функциональные требования (сценарии) 5

3.3 Словарь предметной области 7

3.4 Структура программы на уровне классов 7

3.5 Типовые процессы в программе 10

3.6 Человеко-машинное взаимодействие 16

3.7 Реализация ключевых классов 19

3.8 Реализация ключевых тестовых случаев 30

4 Вторая итерация разработки 42

4.1 Функциональные требования (сценарии) 42

4.2 Словарь предметной области 47

4.3 Структура программы на уровне классов 48

4.4 Типовые процессы в программе 51

4.5 Человеко-машинное взаимодействие 55

4.6 Реализация ключевых классов 61

4.7 Реализация ключевых тестовых случаев 75

5 Список использованной литературы и других источников 90

# 1 Формулировка задания

Игра "Коза и капуста".

Правила игры:

* имеется загон NxM клеток, в котором находится коза, капуста, ящики и стены (располагаются внутри клеток);
* ящики и стены, возможно, образуют непроходимый лабиринт;
* цель козы — добраться до капусты, двигая ящики;
* коза может двигать только один ящик от себя или на себя;
* количество шагов козы ограничено.

Подвариант 3: необходимо предусмотреть в программе точки расширения, используя которые можно реализовать вариативную часть программы (в дополнение к базовой функциональности).

Вариативность: возможны ящики с различным видом взаимодействия (зацепления) между собой. Если несколько ящиков сцепляются, то коза толкает и отталкивает сразу несколько ящиков (если это возможно). Ящики разного вида должны быть визуально различимы.

НЕ изменяя ранее созданные классы, а используя точки расширения, реализовать:

* магнитные ящики, которые притягиваются или отталкиваются в зависимости от полюса (стороны ящика). Область действия магнитного поля - соседняя клетка. Отталкивающиеся ящики не могут находится в соседних клетках.
* металлический ящик, который притягивается к магнитному ящику.

# 2 Нефункциональные требования

1. Программа должна быть реализована на языке Java SE 14 с использованием стандартных библиотек, в том числе, библиотеки Swing.
2. Форматирование исходного кода программы должно соответствовать Java Code Conventions, September 12, 1997.

# 3 Первая итерация разработки

### 3.1 Формулировка упрощенного варианта задания

Игра "Коза и капуста".

Правила игры:

- имеется загон NxM клеток, в котором находится коза, капуста и стены (располагаются внутри клеток);

- стены, возможно, образуют непроходимый лабиринт;

- цель козы — добраться до капусты;

- количество шагов козы ограничено.

### 3.2 Функциональные требования (сценарии)

1. Главный сценарий - коза добралась до капусты

1. Пользователь инициирует начало игры.

2. Игра создаёт поле, состоящее из ячеек, и размещает в них козу, капусту, ящики и стены.

3. **Делать**:

1. По указанию пользователя коза перемещается на соседнюю ячейку с уменьшением счётчика шагов в заданном пользователем направлении.

**Пока** коза не добралась до ячейки с капустой и счётчик шагов != 0.

4. Игра определяет исход - игра успешна.

2. Сценарий - создание поля

1. Игра создаёт поле N x M клеток.

2. Игра создаёт и расставляет на поле стены.

3. Игра создаёт козу и наделяет её счётчиком шагов, ставит её на поле.

4. Игра создаёт капусту и ставит её на поле.

3. Сценарий - коза перемещается на соседнюю ячейку с уменьшением счётчика шагов

1. Пользователь задаёт направление перемещения козы.
2. Коза спрашивает у ячейки соседнюю ячейку в данном направлении.
3. Текущая ячейка сообщает соседнюю ячейку козе.
4. Коза спрашивает у соседней ячейки свободна ли она.
5. Соседняя ячейка сообщает, что она свободна.
6. Коза просит счётчик шагов уменьшиться
7. Счётчик шагов уменьшается на единицу, так как у него достаточно шагов.
8. Коза просит текущую ячейку изъять себя из неё.
9. Текущая ячейка извлекает козу из себя.
10. Коза просит соседнюю ячейку поместить себя в неё.
11. Соседняя ячейка помещает козу в себя, так как она свободна.

3.1. Альтернативный сценарий - соседней ячейки в заданном направлении нет

1. Сценарий начинается после п. 2 сценария 3
2. Текущая ячейка сообщает, что у неё нет соседней ячейки.
3. Сценарий переходит к п. 3. главного сценария

3.2. Альтернативный сценарий - соседняя ячейка занята

1. Сценарий начинается после п. 4 сценария 3
2. Соседняя ячейка сообщает, что она занята
3. Сценарий переходит к п. 3. главного сценария

3.3 Альтернативный сценарий - у счётчика шагов недостаточно шагов

1. Сценарий начинается после п. 6 сценария 3
2. Счётчик сообщает, что у него кончились шаги
3. Сценарий переходит к п. 4 главного сценария

4. Сценарий - игра определяет исход - игра успешна

1. Игра спрашивает у поля ячейку с капустой
2. Поле сообщает ячейку с капустой
3. Игра спрашивает у ячейки с капустой находится ли в ней коза
4. Ячейка сообщает, что в ней находится коза
5. Игра определяет исход - “успешна”

4.1. Альтернативный сценарий - игра определяет исход - шаги кончились, а коза не в ячейке с капустой

1. Сценарий начинается после п. 3 сценария 4
2. Ячейка сообщает, что в ней нет козы
3. Игра спрашивает у козы счётчик шагов
4. Игра спрашивает у счётчика шагов сколько осталось шагов
5. Счётчик сообщает, что шаги кончились
6. Игра определяет исход - “неуспешна”

### 3.3 Словарь предметной области

**Игра.** Знает о поле. Управляет игровым циклом: определяет окончание игры и её исход. Определяет начальную расстановку элементов игрового поля (коза, капуста, стены…).

**Поле** –прямоугольная область, состоящая из ячеек. Знает о ячейке с капустой и козе.

**Ячейка** – квадратная область поля. Знает о четырёх соседних ячейках. Может содержать в себе либо стену, либо козу.

**Ячейка с капустой** – разновидность ячейки. На поле может находиться одна ячейка такого типа. Может содержать только козу.

**Коза** перемещается между соседними ячейками. Для перемещения использует счётчик шагов. Занимает одну ячейку поля.

**Стена** содержится в ячейке. Является препятствием.

**Счётчик шагов** управляет количеством шагов козы**.**

### 3.4 Структура программы на уровне классов

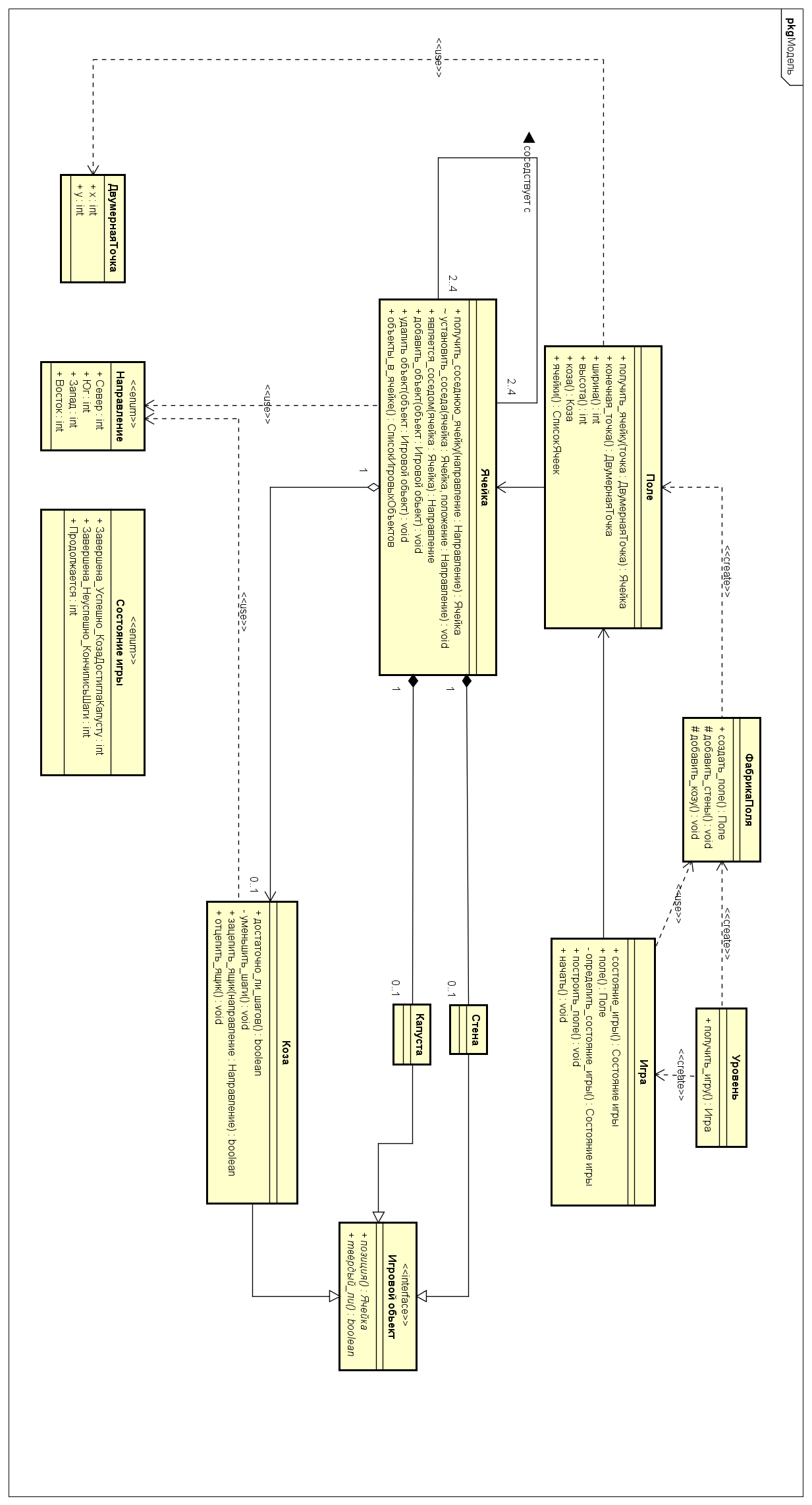


Рис. 1. Диаграмма классов модели упрощённой версии

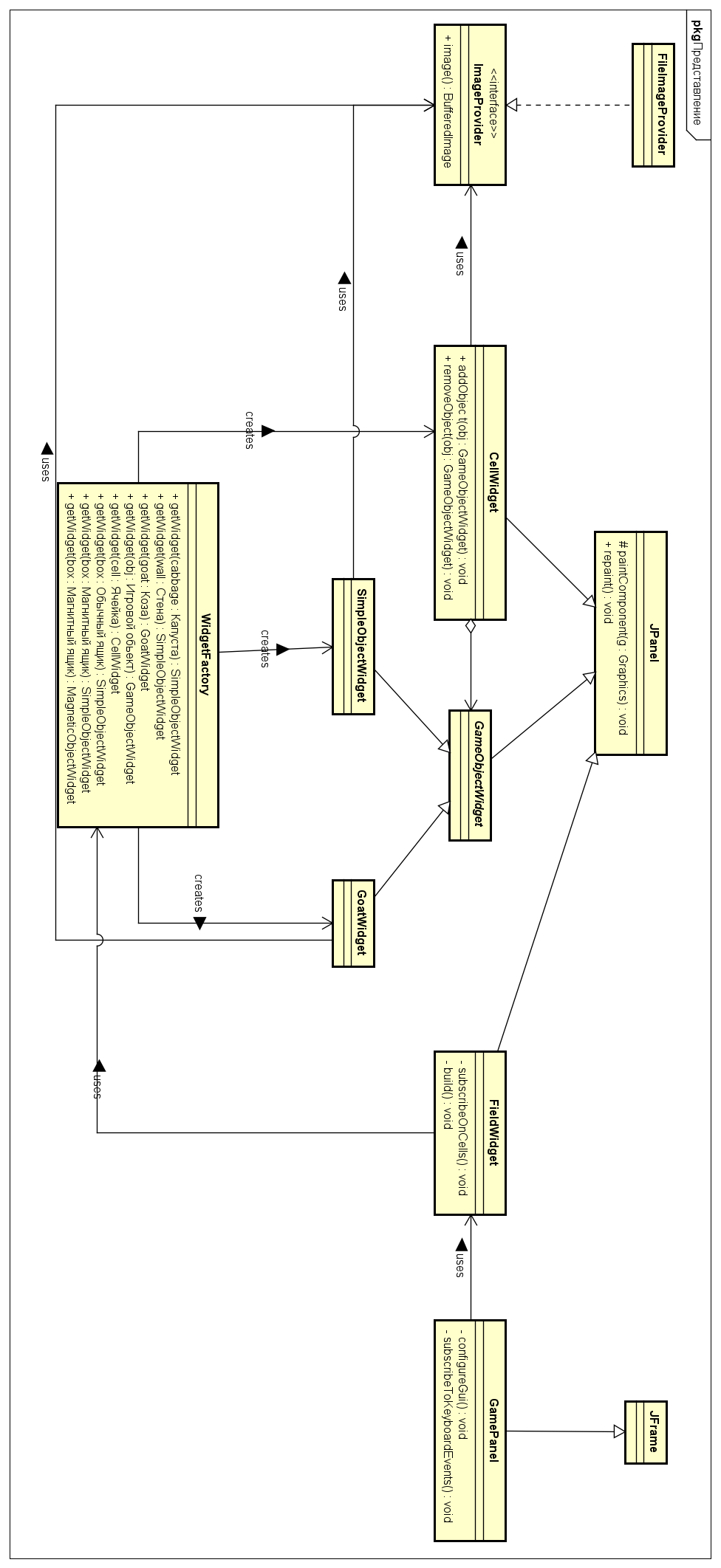


Рис. 2. Диаграмма классов представления упрощённой версии

### 3.5 Типовые процессы в программе

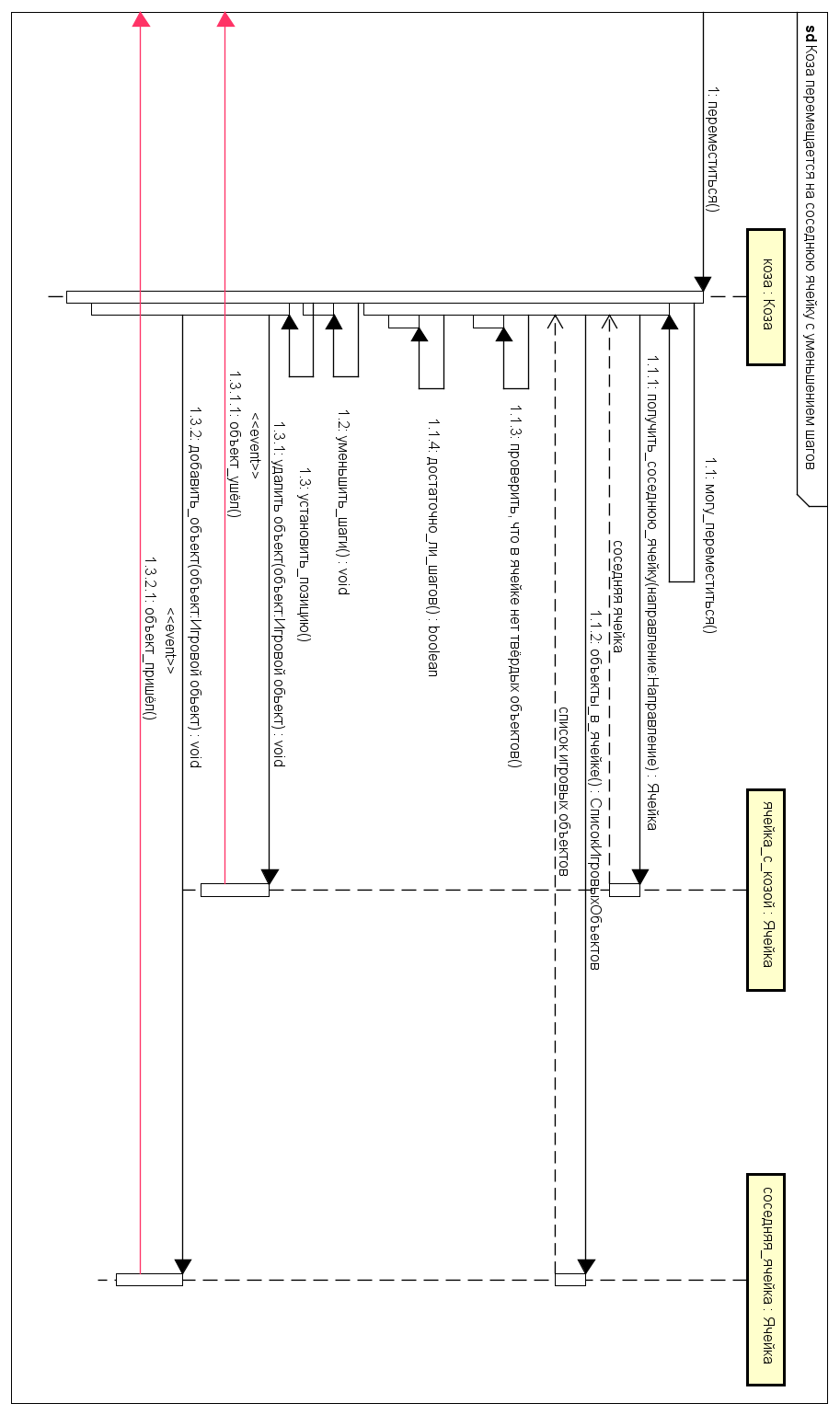


Рис. 3. Диаграмма последовательности модели упрощённой версии «Коза перемещается на соседнюю ячейку с уменьшением шагов»

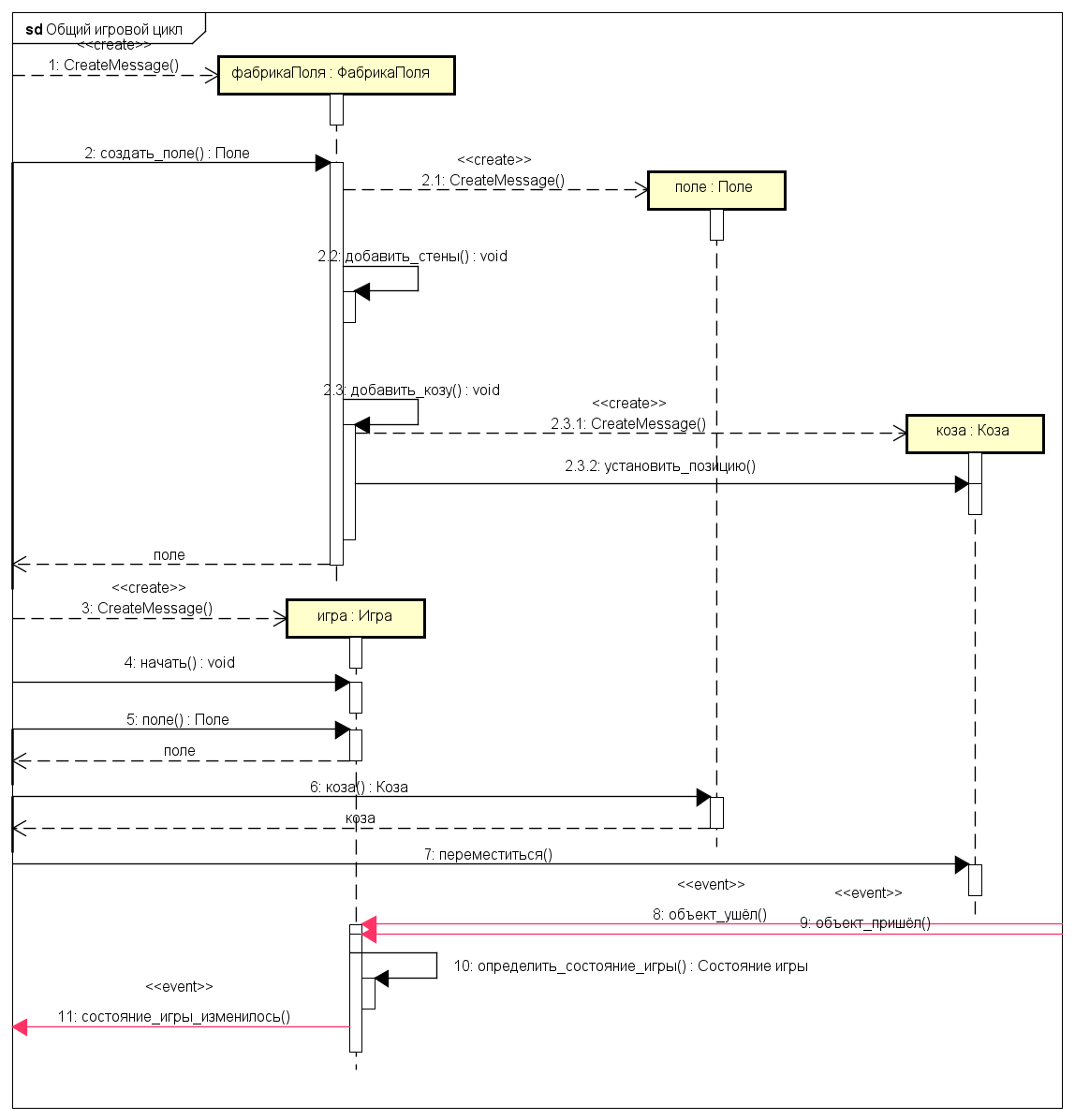


Рис. 4. Диаграмма последовательности модели упрощённой версии «Общий игровой цикл»

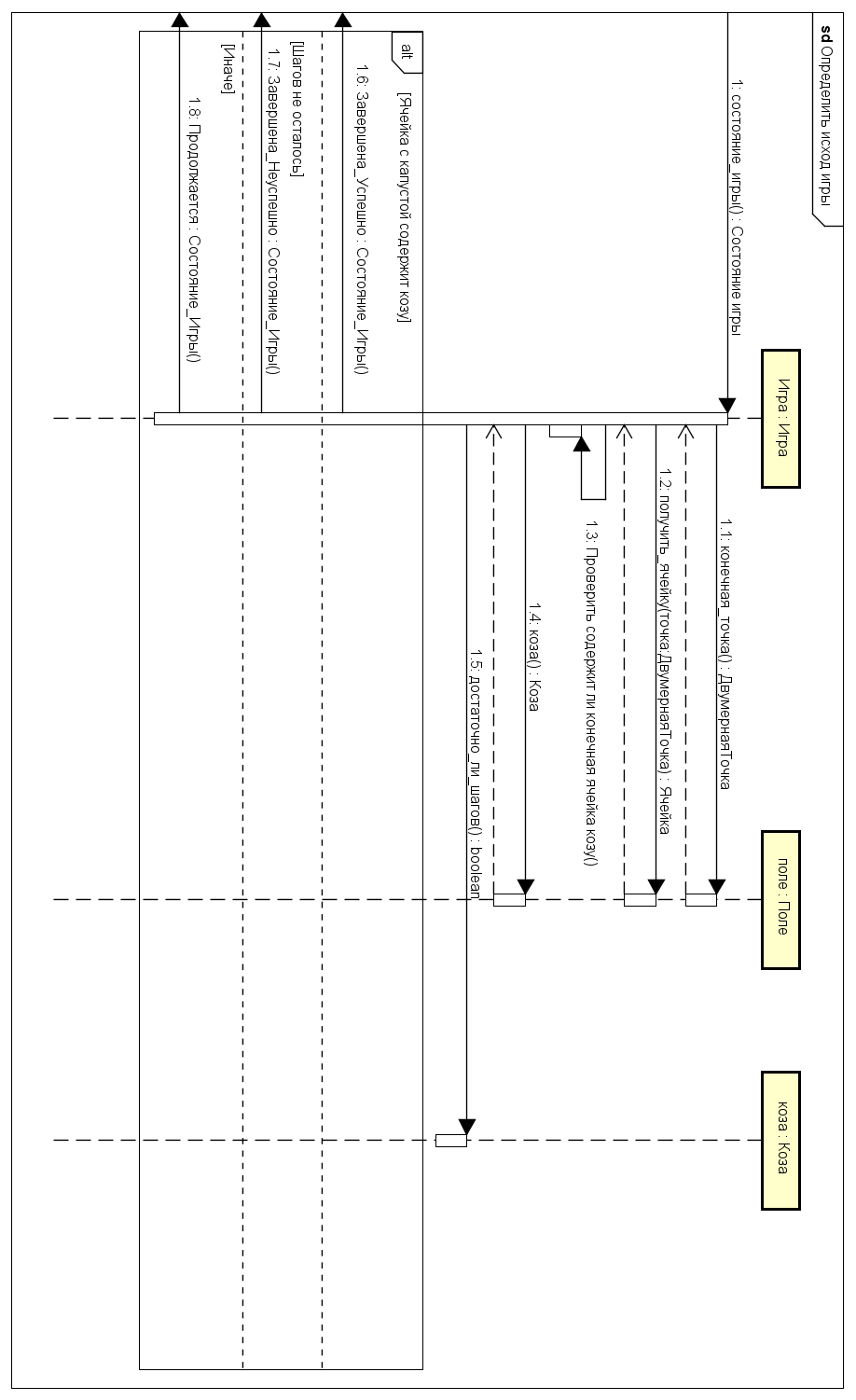


Рис. 5. Диаграмма последовательности модели упрощённой версии «Определить исход игры»

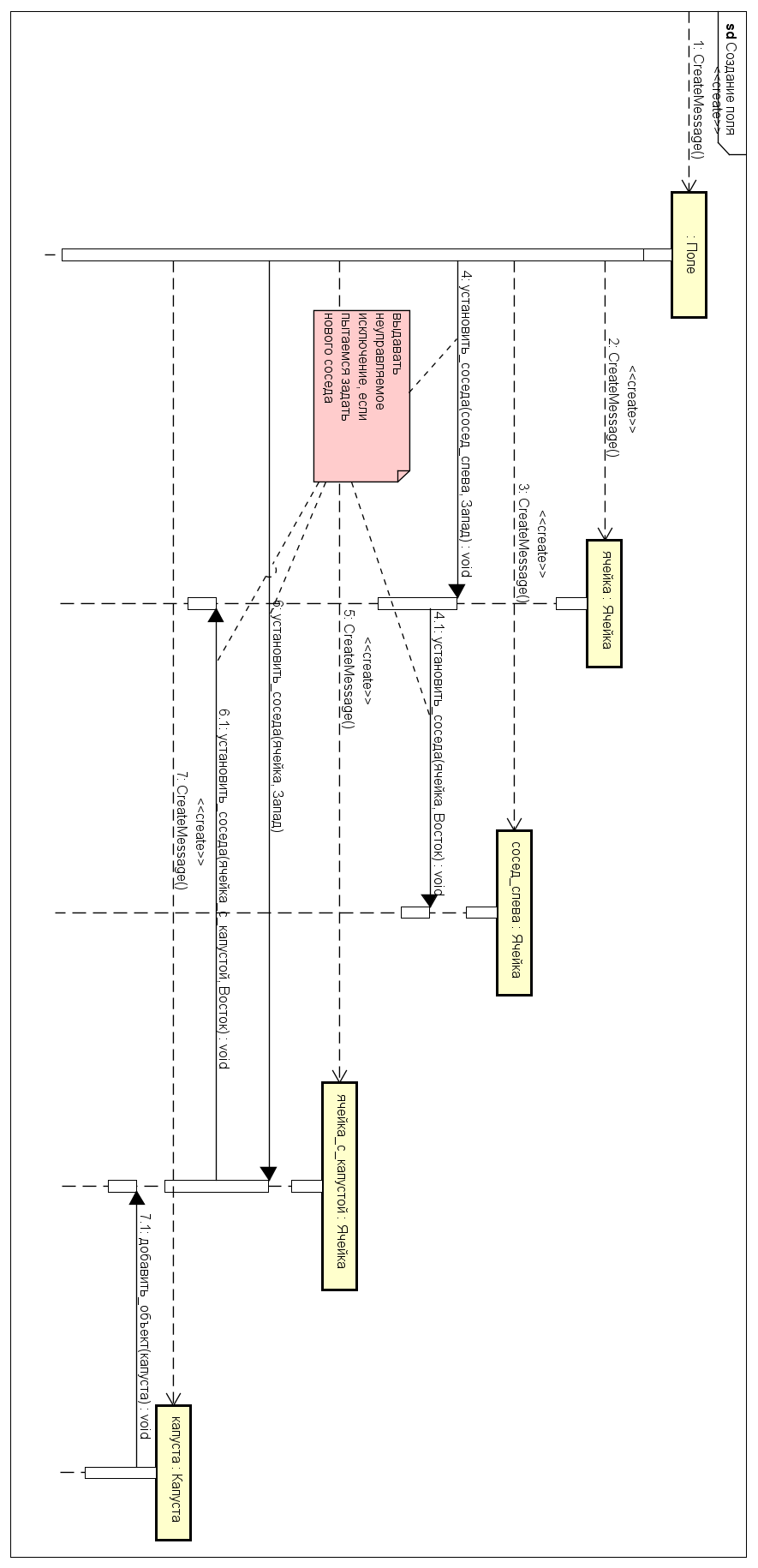


Рис. 6. Диаграмма последовательности модели упрощённой версии «Создание поля»

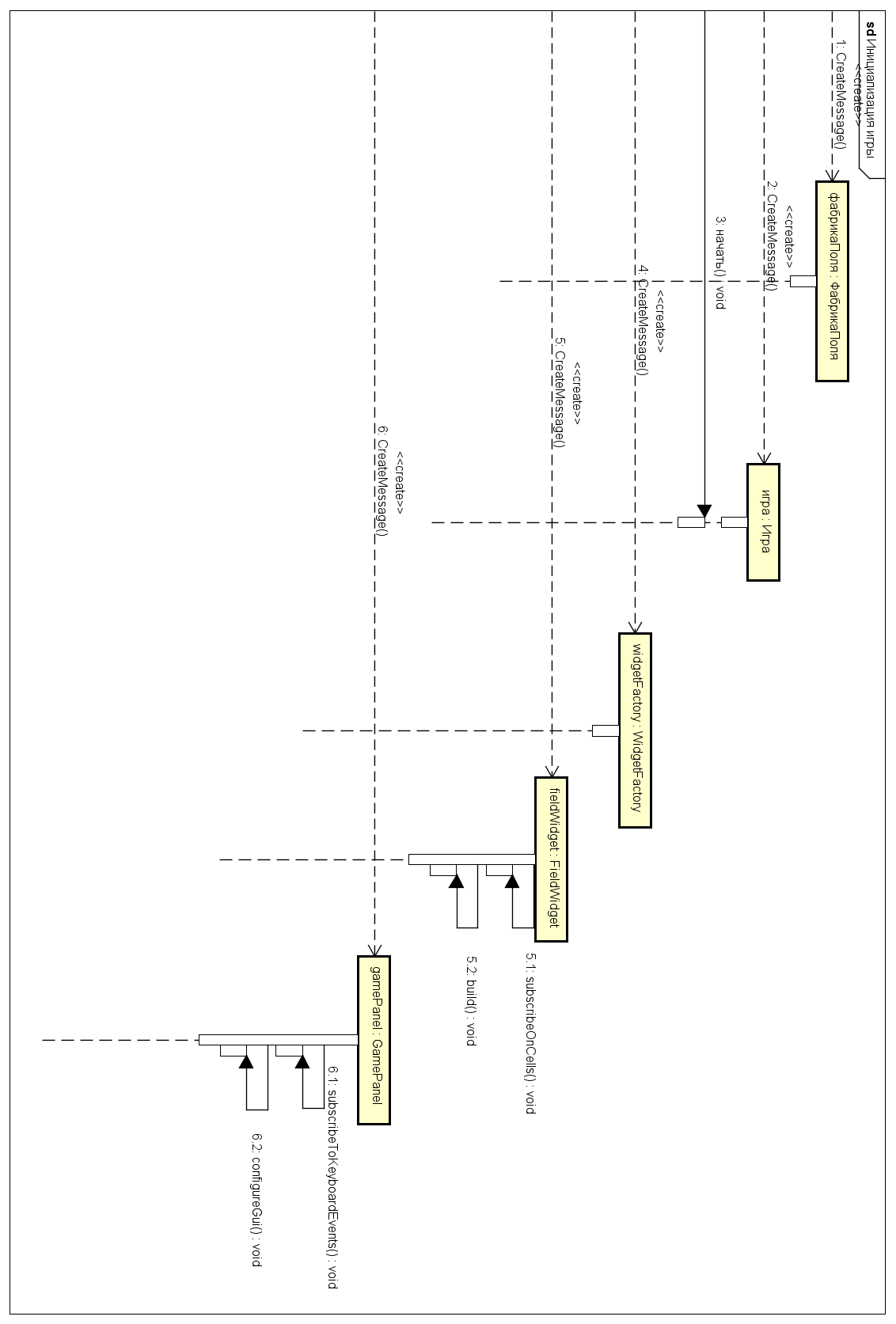


Рис. 7. Диаграмма последовательности представления упрощённой версии «Инициализация игры»

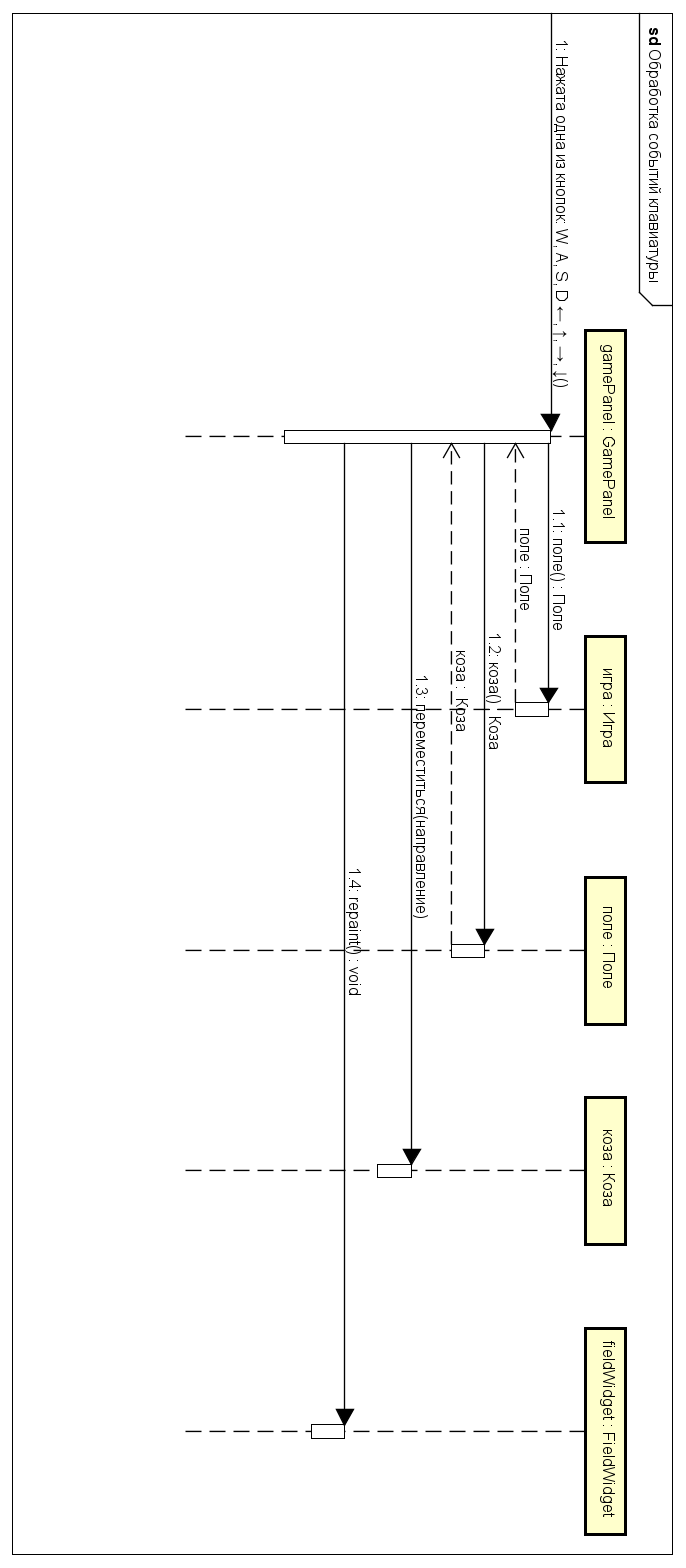


Рис. 8. Диаграмма последовательности представления упрощённой версии «Обработка событий клавиатуры»

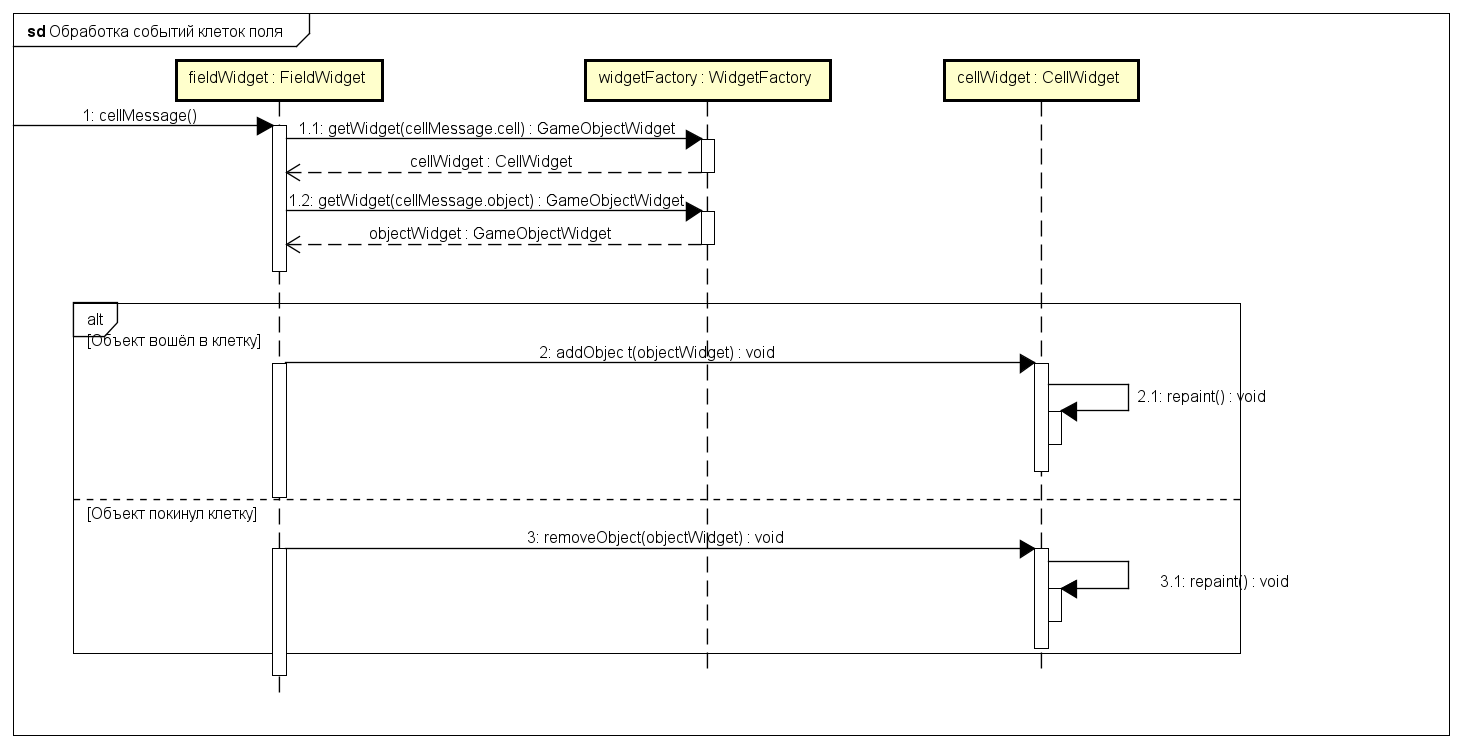


Рис. 9. Диаграмма последовательности представления упрощённой версии «Обработка событий клеток поля»

### 3.6 Человеко-машинное взаимодействие

1. Начальное состояние игры

При запуске игры появляется начальное состояние уровня. Количество шагов козы указывается слева вверху.

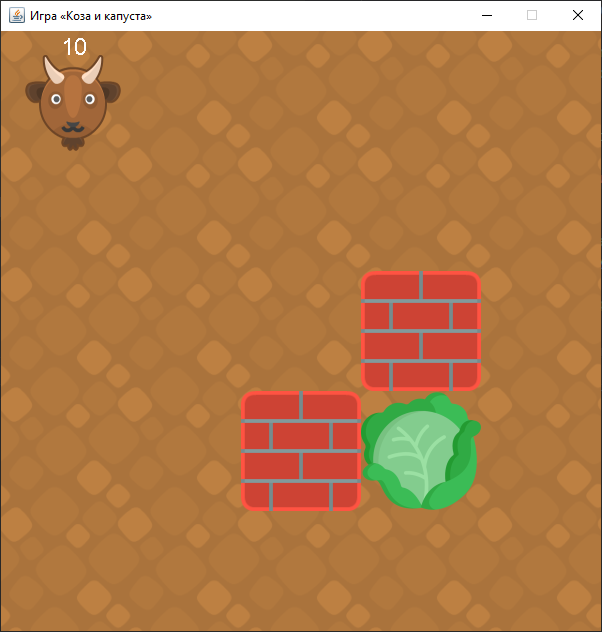


Рис. 10. Начальное состояние игры (упрощённая версия)

1. Состояние игры после хода

Движение козы осуществляется клавишами:

* W, ↑ - вверх;
* A, ← - влево;
* S, ↓ - вниз;
* D, → - вправо.

После движения козы её количество шагов уменьшается.

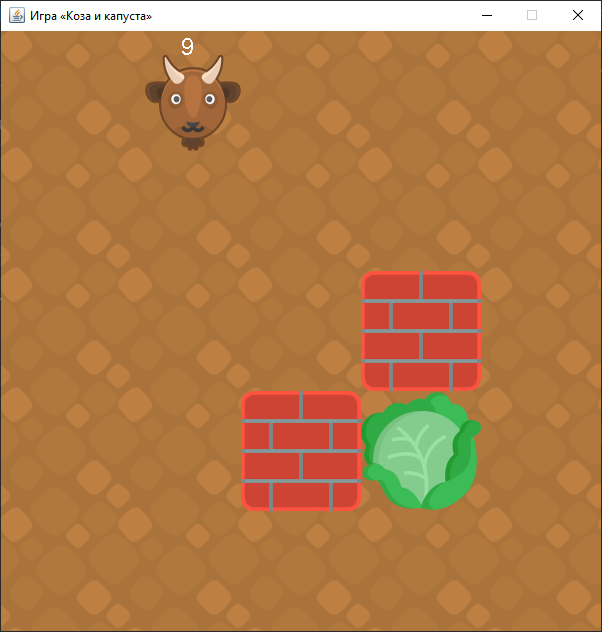


Рис. 11. Состояние игры после хода (упрощённая версия)

1. Игра завершена успешно - коза достигла капусты

При достижении козой капусты высвечивается MessageBox с сообщением об успешном завершении игры и окно игры закрывается.

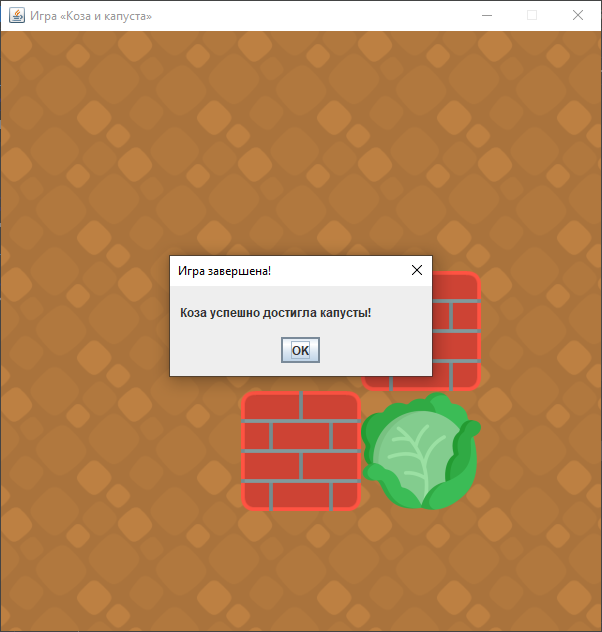


Рис. 12. Игра завершена успешно - коза достигла капусты (упрощённая версия)

1. Игра завершена неуспешно - у козы кончились шаги

Когда у козы закончились шаги, и она не достигла капусты, высвечивается MessageBox с сообщением о неуспешном завершении игры и окно игры закрывается.

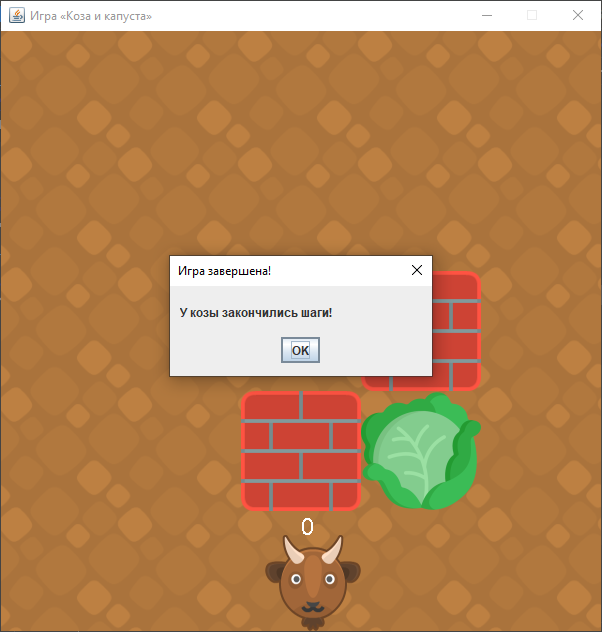


Рис. 13. Игра завершена неуспешно - у козы кончились шаги (упрощённая версия)

### 3.7 Реализация ключевых классов

Листинг 1. Реализация класса игрового поля

package model;

import events.MessageSender;

import model.exceptions.IllegalDimensionException;

import model.exceptions.PointIsNotInFieldRangeException;

import org.jetbrains.annotations.NotNull;

import utils.Direction;

import utils.Point;

import utils.collections.ReadOnlyList;

import java.util.HashMap;

import java.util.Map;

import java.util.stream.Collectors;

public class GameField {

private final @NotNull Map<Point, Cell> cells = new HashMap<>();

private final int width;

private final int height;

private final @NotNull Point exitPoint;

private final @NotNull MessageSender messageSender;

private final @NotNull Cabbage cabbage;

private Goat goat;

public GameField(int width, int height, @NotNull Point exitPoint, @NotNull MessageSender messageSender) {

assertDimensionIsCorrect(width, "width");

assertDimensionIsCorrect(height, "height");

this.width = width;

this.height = height;

assertPointInRange(exitPoint);

this.exitPoint = exitPoint;

this.messageSender = messageSender;

setup();

cabbage = new Cabbage(cell(exitPoint));

}

private void setup() {

for (int y = 0; y < height; ++y) {

for (int x = 0; x < width; ++x) {

var point = new Point(x, y);

var cell = new Cell(messageSender);

if (x > 0)

cell(new Point(point.x - 1, point.y)).setNeighbor(cell, Direction.EAST);

if (y > 0)

cell(new Point(point.x, point.y - 1)).setNeighbor(cell, Direction.SOUTH);

cells.put(point, cell);

}

}

}

public Cell cell(@NotNull Point point) {

if (!isPointInRange(point)) {

return null;

}

return cells.get(point);

}

private boolean isPointInRange(@NotNull Point point) {

return point.x >= 0

&& point.x < width()

&& point.y >= 0

&& point.y < height();

}

public int width() {

return width;

}

public int height() {

return height;

}

public Goat goat() {

if (goat != null)

return goat;

var possibleGoat = cells.values().stream().flatMap(cell -> cell.objects().stream())

.filter(gameObject -> gameObject instanceof Goat).findFirst();

possibleGoat.ifPresent(gameObject -> goat = (Goat)gameObject);

return goat;

}

public ReadOnlyList<CellWithPosition> cells() {

return ReadOnlyList.fromList(cells.entrySet().stream().map(entry -> new CellWithPosition(entry.getValue(), entry.getKey())).collect(Collectors.toList()));

}

public Point exitPoint() {

return exitPoint;

}

private void assertPointInRange(@NotNull Point point) {

if (!isPointInRange(point))

throw new PointIsNotInFieldRangeException(point, width, height);

}

private void assertDimensionIsCorrect(int dimension, @NotNull String name) {

if (dimension <= 0)

throw new IllegalDimensionException(dimension, name);

}

}

Листинг 2. Реализация класса клетки игрового поля

package model;

import events.MessageSender;

import events.MessageSource;

import model.events.CellMessage;

import model.exceptions.CellAlreadyHasNeighborForDirectionException;

import org.jetbrains.annotations.NotNull;

import utils.Direction;

import utils.collections.ReadOnlyList;

import java.util.\*;

public class Cell implements MessageSource {

private final Map<Direction, Cell> neighbors = new EnumMap<>(Direction.class);

private final List<GameObject> objects = new ArrayList<>();

private final MessageSender messageSender;

public Cell(@NotNull MessageSender messageSender) {

this.messageSender = messageSender;

}

public Cell neighborCell(@NotNull Direction direction) {

return neighbors.get(direction);

}

void setNeighbor(@NotNull Cell cell, @NotNull Direction direction) {

if (neighbors.containsKey(direction) && neighbors.containsValue(cell))

return;

if (neighbors.containsKey(direction))

throw new CellAlreadyHasNeighborForDirectionException(this, direction);

neighbors.put(direction, cell);

if (cell.neighborCell(direction.opposite()) == null) {

cell.setNeighbor(this, direction.opposite());

}

}

public Direction neighborDirection(@NotNull Cell cell) {

for (var i : neighbors.entrySet()) {

if (i.getValue().equals(cell))

return i.getKey();

}

return null;

}

public void addObject(@NotNull GameObject object) {

if (objects.contains(object))

return;

objects.add(object);

messageSender.emitMessage(this, new CellMessage(CellMessage.Type.OBJECT\_ENTERED, this, object));

}

public void removeObject(@NotNull GameObject object) {

objects.remove(object);

messageSender.emitMessage(this, new CellMessage(CellMessage.Type.OBJECT\_LEAVED, this, object));

}

public @NotNull ReadOnlyList<GameObject> objects() {

return ReadOnlyList.fromList(new ArrayList<>(objects));

}

@Override

public boolean equals(Object o) {

return this == o;

}

@Override

public int hashCode() {

return Objects.hash(neighbors.size());

}

}

Листинг 3. Реализация класса «Коза»

package model;

import model.exceptions.NoEnoughStepsException;

import org.jetbrains.annotations.NotNull;

import utils.Direction;

import java.util.Objects;

public class Goat implements GameObject {

static final int STEP\_COST = 1;

private Cell position;

private int steps;

public Goat(int initialSteps) {

steps = initialSteps;

}

public void move(@NotNull Direction direction) {

if (!canMoveTo(direction))

return;

setPosition(position.neighborCell(direction));

decreaseSteps();

}

void setPosition(Cell cell) {

if (position != null)

position.removeObject(this);

if (cell != null)

cell.addObject(this);

position = cell;

}

boolean canMoveTo(@NotNull Direction direction) {

var neighbor = position.neighborCell(direction);

return neighbor != null

&& neighbor.objects().stream().noneMatch(GameObject::isSolid)

&& hasEnoughSteps();

}

public boolean hasEnoughSteps() {

return steps() - STEP\_COST >= 0;

}

public Cell cell() {

return position;

}

private void assertHasSteps() {

if (!hasEnoughSteps())

throw new NoEnoughStepsException();

}

void decreaseSteps() {

assertHasSteps();

steps -= STEP\_COST;

}

public int steps() {

return steps;

}

@Override

public boolean isSolid() {

return true;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

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

Goat goat = (Goat) o;

return steps == goat.steps &&

position.equals(goat.position);

}

@Override

public int hashCode() {

return Objects.hash("Goat");

}

}

Листинг 4. Реализация класса «Игра»

package model;

import events.\*;

import model.events.GameMessage;

import org.jetbrains.annotations.NotNull;

public class Game implements MessageListener, MessageSource {

private final @NotNull FieldFactory fieldFactory;

private final @NotNull SubscriptionHandler subscriptionHandler;

private final @NotNull MessageSender messageSender;

private GameField gameField;

private GameState gameState;

public Game(@NotNull FieldFactory fieldFactory, @NotNull SubscriptionHandler subscriptionHandler, @NotNull MessageSender messageSender) {

this.fieldFactory = fieldFactory;

this.subscriptionHandler = subscriptionHandler;

this.messageSender = messageSender;

}

public void start() {

gameField = fieldFactory.create();

for (var cell : gameField.cells()) {

subscriptionHandler.subscribeTo(cell.cell, this);

}

gameState = GameState.CONTINUING;

}

private GameState determineOutcomeGame() {

if (gameField.goat() == null)

return gameState;

var goat = gameField.goat();

var isGoatOnCellWithCabbage = gameField.cell(gameField.exitPoint()).objects().stream().anyMatch(gameObject -> gameObject == goat);

var hasNoSteps = !goat.hasEnoughSteps();

if (isGoatOnCellWithCabbage)

return GameState.ENDED\_SUCCESS\_GOAT\_REACHED\_CABBAGE;

if (hasNoSteps)

return GameState.ENDED\_FAILURE\_STEPS\_EXPIRED;

return GameState.CONTINUING;

}

public GameState gameState() {

return determineOutcomeGame();

}

public @NotNull GameField gameField() {

return gameField;

}

@Override

public void handleMessage(MessageSource source, MessageData data) {

gameState = determineOutcomeGame();

messageSender.emitMessage(this, new GameMessage(gameState));

}

}

Листинг 5. Реализация класса виджета козы

package view.widgets;

import model.Goat;

import org.jetbrains.annotations.NotNull;

import view.providers.ImageProvider;

import view.utils.ImageUtils;

import java.awt.\*;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import static view.utils.DirectionUtils.\*;

public class GoatWidget extends GameObjectWidget {

private final @NotNull Goat goat;

private final @NotNull ImageProvider imageProvider;

public GoatWidget(@NotNull Goat goat, @NotNull ImageProvider imageProvider) {

this.goat = goat;

this.imageProvider = imageProvider;

// addKeyListener(new KeyController());

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

var delta = CellWidget.CELL\_DIMENSION.width / 5;

var middle = CellWidget.CELL\_DIMENSION.width / 2;

var resizeDimension = new Dimension(CellWidget.CELL\_DIMENSION.width - delta, CellWidget.CELL\_DIMENSION.height - delta);

var resizedImage = ImageUtils.resizeImage(imageProvider.image(), resizeDimension);

g.setFont(new Font("Arial", Font.PLAIN, delta));

g.setColor(Color.WHITE);

g.drawString(String.valueOf(goat.steps()), middle, delta);

g.drawImage(resizedImage, delta, delta, null);

}

// private class KeyController implements KeyListener {

//

// @Override

// public void keyTyped(KeyEvent e) {

// }

//

// @Override

// public void keyPressed(KeyEvent e) {

// System.out.println("I'm here!");

// if (isDirectionKeyCode(e.getKeyCode())) {

// goat.move(keyCodeToDirection(e.getKeyCode()));

// System.out.println("Goat moves to " + keyCodeToDirection(e.getKeyCode()));

// repaint();

// }

// }

//

// @Override

// public void keyReleased(KeyEvent e) {

// }

// }

}

Листинг 6. Реализация виджета игровой клетки

package view.widgets;

import org.jetbrains.annotations.NotNull;

import view.providers.ImageProvider;

import view.utils.ImageUtils;

import javax.swing.\*;

import java.awt.\*;

import java.util.ArrayList;

import java.util.List;

public class CellWidget extends JPanel {

static final Dimension CELL\_DIMENSION = new Dimension(120, 120);

private final List<GameObjectWidget> objects = new ArrayList<>(2);

private final @NotNull ImageProvider backgroundProvider;

public CellWidget(@NotNull ImageProvider backgroundProvider) {

super();

this.backgroundProvider = backgroundProvider;

setPreferredSize(CELL\_DIMENSION);

}

public void addObject(@NotNull GameObjectWidget gameObjectWidget) {

objects.add(gameObjectWidget);

repaint();

}

public void removeObject(@NotNull GameObjectWidget gameObjectWidget) {

objects.remove(gameObjectWidget);

repaint();

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

g.drawImage(ImageUtils.resizeImage(backgroundProvider.image(), CELL\_DIMENSION), 0, 0, null);

for (var object : objects) {

object.paintComponent(g);

}

}

}

Листинг 6. Реализация класса виджета поля

package view.widgets;

import events.MessageData;

import events.MessageListener;

import events.MessageSource;

import events.SubscriptionHandler;

import model.GameField;

import model.events.CellMessage;

import org.jetbrains.annotations.NotNull;

import view.WidgetFactory;

import javax.swing.\*;

import java.awt.\*;

import java.util.Comparator;

import java.util.stream.Collectors;

public class FieldWidget extends JPanel {

private final @NotNull GameField gameField;

private final @NotNull WidgetFactory widgetFactory;

public FieldWidget(@NotNull GameField gameField, @NotNull WidgetFactory widgetFactory, @NotNull SubscriptionHandler subscriptionHandler) {

this.gameField = gameField;

this.widgetFactory = widgetFactory;

setLayout(new GridLayout(gameField.height(), gameField.width()));

subscribeOnCells(subscriptionHandler);

build();

}

private void subscribeOnCells(@NotNull SubscriptionHandler subscriptionHandler) {

var cellController = new CellController();

for (var cellWithPosition : gameField.cells()) {

subscriptionHandler.subscribeTo(cellWithPosition.cell, cellController);

}

}

private void build() {

var sortedCells = gameField.cells().stream()

.sorted(Comparator.comparingInt(cellWithPosition -> cellWithPosition.position.x))

.sorted(Comparator.comparingInt(cellWithPosition -> cellWithPosition.position.y))

.map(cellWithPosition -> cellWithPosition.cell)

.collect(Collectors.toList());

for (var cell : sortedCells) {

var cellWidget = widgetFactory.getWidget(cell);

add(cellWidget);

}

}

private class CellController implements MessageListener {

@Override

public void handleMessage(MessageSource source, MessageData data) {

if (data instanceof CellMessage) {

var cellMessage = (CellMessage) data;

switch (cellMessage.type) {

case OBJECT\_ENTERED -> onObjectEntered(cellMessage);

case OBJECT\_LEAVED -> onObjectLeaved(cellMessage);

}

}

}

private void onObjectLeaved(CellMessage cellMessage) {

var cellWidget = widgetFactory.getWidget(cellMessage.cell);

var objectWidget = widgetFactory.getWidget(cellMessage.object);

if (objectWidget == null)

return;

cellWidget.removeObject(objectWidget);

}

private void onObjectEntered(CellMessage cellMessage) {

var cellWidget = widgetFactory.getWidget(cellMessage.cell);

var objectWidget = widgetFactory.getWidget(cellMessage.object);

if (objectWidget == null)

return;

cellWidget.addObject(objectWidget);

}

}

}

Листинг 7. Реализация класса фабрики виджетов

package view;

import model.\*;

import org.jetbrains.annotations.NotNull;

import view.providers.ImageProvider;

import view.widgets.CellWidget;

import view.widgets.GameObjectWidget;

import view.widgets.GoatWidget;

import view.widgets.StaticObjectWidget;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

public class WidgetFactory {

private final @NotNull Map<GameObject, GameObjectWidget> gameObjectWidgets = new HashMap<>();

private final @NotNull Map<Cell, CellWidget> cellWidgets = new HashMap<>();

private final @NotNull Map<Class, ImageProvider> imageProviders;

public WidgetFactory(@NotNull Map<Class, ImageProvider> imageProviders) {

this.imageProviders = imageProviders;

}

public GameObjectWidget getWidget(@NotNull Cabbage cabbage) {

return getGameObjectWidget(cabbage, gameObject -> new StaticObjectWidget(imageProviders.get(cabbage.getClass())));

}

public GameObjectWidget getWidget(@NotNull Wall wall) {

return getGameObjectWidget(wall, gameObject -> new StaticObjectWidget(imageProviders.get(wall.getClass())));

}

public GameObjectWidget getWidget(@NotNull Goat goat) {

return getGameObjectWidget(goat, gameObject -> new GoatWidget(goat, imageProviders.get(goat.getClass())));

}

public GameObjectWidget getWidget(@NotNull GameObject gameObject) {

return gameObjectWidgets.get(gameObject);

}

public CellWidget getWidget(@NotNull Cell cell) {

cellWidgets.computeIfAbsent(cell, cell1 -> {

var cellWidget = new CellWidget(imageProviders.get(cell.getClass()));

for (var object : cell.objects()) {

if (object instanceof Cabbage) {

cellWidget.addObject(getWidget((Cabbage) object));

} else if (object instanceof Goat) {

cellWidget.addObject(getWidget((Goat) object));

} else if (object instanceof Wall) {

cellWidget.addObject(getWidget((Wall) object));

} else {

throw new IllegalArgumentException("Object creation is not supported" + object);

}

}

return cellWidget;

});

return cellWidgets.get(cell);

}

private GameObjectWidget getGameObjectWidget(@NotNull GameObject gameObject, Function<GameObject, GameObjectWidget> mappingFunction) {

gameObjectWidgets.computeIfAbsent(gameObject, mappingFunction);

return gameObjectWidgets.get(gameObject);

}

}

Листинг 8. Реализация класса игрового окна (панели)

package view;

import events.\*;

import model.Game;

import model.GameState;

import model.events.GameMessage;

import org.jetbrains.annotations.NotNull;

import view.widgets.FieldWidget;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.event.WindowEvent;

import static javax.swing.JOptionPane.\*;

import static view.utils.DirectionUtils.isDirectionKeyCode;

import static view.utils.DirectionUtils.keyCodeToDirection;

public class GamePanel extends JFrame implements MessageListener {

private final @NotNull Game game;

private final @NotNull FieldWidget fieldWidget;

public GamePanel(@NotNull Game game, @NotNull FieldWidget fieldWidget, @NotNull SubscriptionHandler subscriptionHandler) throws HeadlessException {

this.game = game;

this.fieldWidget = fieldWidget;

subscriptionHandler.subscribeTo(game, this);

configureGui();

subscribeToKeyboardEvents();

}

private void configureGui() {

setVisible(true);

JPanel content = (JPanel) this.getContentPane();

content.add(this.fieldWidget);

pack();

setResizable(false);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setFocusable(true);

setTitle("Игра «Коза и капуста»");

}

private void subscribeToKeyboardEvents() {

addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

}

@Override

public void keyPressed(KeyEvent e) {

if (isDirectionKeyCode(e.getKeyCode())) {

game.gameField().goat().move(keyCodeToDirection(e.getKeyCode()));

fieldWidget.repaint();

}

handleState(game.gameState());

}

@Override

public void keyReleased(KeyEvent e) {

}

});

}

private void handleState(GameState gameState) {

final String MESSAGE\_TITLE = "Игра завершена!";

if (gameState == GameState.ENDED\_SUCCESS\_GOAT\_REACHED\_CABBAGE) {

showMessageDialog(this, "Коза успешно достигла капусты!", MESSAGE\_TITLE, PLAIN\_MESSAGE);

close();

} else if (gameState == GameState.ENDED\_FAILURE\_STEPS\_EXPIRED) {

showMessageDialog(this, "У козы закончились шаги!", MESSAGE\_TITLE, PLAIN\_MESSAGE);

close();

}

}

@Override

public void handleMessage(MessageSource source, MessageData data) {

var gameMessage = (GameMessage) data;

handleState(gameMessage.gameState);

}

private void close() {

dispatchEvent(new WindowEvent(this, WindowEvent.WINDOW\_CLOSING));

}

}

### 3.8 Реализация ключевых тестовых случаев

Листинг 9. Реализация тестов игрового поля

package model;

import events.MessageSender;

import model.exceptions.IllegalDimensionException;

import model.exceptions.PointIsNotInFieldRangeException;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import utils.Direction;

import utils.Pair;

import utils.Point;

import java.util.Arrays;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.mock;

class GameFieldShould {

private int width;

private int height;

private Point exitPoint;

private MessageSender messageSender;

@BeforeEach

void beforeEach() {

width = 2;

height = 2;

exitPoint = new Point(1, 1);

messageSender = mock(MessageSender.class);

}

@Test

void connectCellsCorrectly() {

// Arrange & Act

var field = new GameField(width, height, exitPoint, messageSender);

var cell\_00 = field.cell(new Point(0, 0));

var cell\_01 = field.cell(new Point(1, 0));

var cell\_10 = field.cell(new Point(0, 1));

var cell\_11 = field.cell(new Point(1, 1));

// Assert

assertEquals(Direction.SOUTH, cell\_00.neighborDirection(cell\_10));

assertEquals(Direction.SOUTH, cell\_01.neighborDirection(cell\_11));

assertEquals(Direction.NORTH, cell\_11.neighborDirection(cell\_01));

assertEquals(Direction.NORTH, cell\_10.neighborDirection(cell\_00));

assertEquals(Direction.EAST, cell\_00.neighborDirection(cell\_01));

assertEquals(Direction.EAST, cell\_10.neighborDirection(cell\_11));

assertEquals(Direction.WEST, cell\_01.neighborDirection(cell\_00));

assertEquals(Direction.WEST, cell\_11.neighborDirection(cell\_10));

}

@Test

void insertCabbageToExitPointCell() {

// Arrange & Act

var field = new GameField(width, height, exitPoint, messageSender);

var exitPointCell = field.cell(exitPoint);

// Assert

assertTrue(exitPointCell.objects().stream().anyMatch(object -> object instanceof Cabbage));

}

@Test

void throw\_whenCreatedWithIncorrectDimensions() {

// Arrange

var dimensions = Arrays.asList(

new Pair<>(0, 0),

new Pair<>(-1, 0),

new Pair<>(0, -1),

new Pair<>(1, 0),

new Pair<>(0, 1),

new Pair<>(-1, 1),

new Pair<>(1, -1)

);

// Act & Assert

for (var dimension : dimensions) {

assertThrows(IllegalDimensionException.class, () -> new GameField(dimension.first, dimension.second, exitPoint, messageSender));

}

}

@Test

void throw\_whenExitPointIsOutOfField() {

// Arrange

var exitPoints = Arrays.asList(

new Point(width, height - 1),

new Point(width - 1, height),

new Point(width, height),

new Point(-1, 0),

new Point(0, -1),

new Point(-1, -1)

);

// Act & Assert

for (var point : exitPoints)

assertThrows(PointIsNotInFieldRangeException.class, () -> new GameField(width, height, point, messageSender));

}

@Test

void haveGivenWidth() {

// Arrange

var field = new GameField(width, height, exitPoint, messageSender);

// Act & Assert

assertEquals(width, field.width());

}

@Test

void haveGivenHeight() {

// Arrange

var field = new GameField(width, height, exitPoint, messageSender);

// Act & Assert

assertEquals(height, field.height());

}

@Test

void haveActualGoat() {

// Arrange

var goat = new Goat(Goat.STEP\_COST);

var field = new GameField(4, 5, exitPoint, messageSender);

assertNull(field.goat());

for (int y = 0; y < field.height(); ++y) {

for (int x = 0; x < field.width(); ++x) {

goat.setPosition(field.cell(new Point(x, y)));

// Act & Assert

assertSame(goat, field.goat());

}

}

}

@Test

void returnNull\_whenRequestedCellDoesntExist() {

// Arrange

var points = Arrays.asList(

new Point(width, height - 1),

new Point(width - 1, height),

new Point(width, height),

new Point(-1, 0),

new Point(0, -1),

new Point(-1, -1)

);

var field = new GameField(width, height, exitPoint, messageSender);

// Act & Assert

for (var point : points)

assertNull(field.cell(point));

}

}

Листинг 10. Реализация тестов клетки игрового поля

package model;

import events.MessageSender;

import model.events.CellMessage;

import model.exceptions.CellAlreadyHasNeighborForDirectionException;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import utils.Direction;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.verify;

class CellShould {

private MessageSender messageSender;

private Cell neighborCell;

private Direction direction;

private GameObject gameObject;

@BeforeEach

void beforeEach() {

messageSender = mock(MessageSender.class);

neighborCell = mock(Cell.class);

direction = Direction.NORTH;

gameObject = mock(GameObject.class);

}

@Test

void addNeighbor\_whenNoNeighborInDirection() {

// Arrange

var cell = new Cell(messageSender);

// Act

cell.setNeighbor(neighborCell, direction);

// Assert

assertEquals(direction, cell.neighborDirection(neighborCell));

assertSame(neighborCell, cell.neighborCell(direction));

}

@Test

void throw\_whenAddNewNeighborInUsedDirection() {

// Arrange

var cell = new Cell(messageSender);

cell.setNeighbor(neighborCell, direction);

var newCell = mock(Cell.class);

// Act & Assert

assertThrows(CellAlreadyHasNeighborForDirectionException.class, () -> cell.setNeighbor(newCell, direction));

}

@Test

void notThrow\_whenAddSameNeighborInUsedDirection() {

// Arrange

var cell = new Cell(messageSender);

cell.setNeighbor(neighborCell, direction);

// Act & Assert

assertDoesNotThrow(() -> cell.setNeighbor(neighborCell, direction));

}

@Test

void setItselfAsNeighbor\_whenNewNeighborAdded() {

// Arrange

var cell = new Cell(messageSender);

// Act

cell.setNeighbor(neighborCell, direction);

// Assert

verify(neighborCell).setNeighbor(cell, direction.opposite());

}

@Test

void addObject() {

// Arrange

var cell = new Cell(messageSender);

// Act

cell.addObject(gameObject);

// Assert

assertTrue(cell.objects().contains(gameObject));

}

@Test

void removeObject() {

// Arrange

var cell = new Cell(messageSender);

cell.addObject(gameObject);

// Act

cell.removeObject(gameObject);

// Assert

assertFalse(cell.objects().contains(gameObject));

}

@Test

void sendMessage\_whenObjectAdded() {

// Arrange

var cell = new Cell(messageSender);

var messageToBeSent = new CellMessage(CellMessage.Type.OBJECT\_ENTERED, cell, gameObject);

// Act

cell.addObject(gameObject);

// Assert

verify(messageSender).emitMessage(cell, messageToBeSent);

}

@Test

void sendMessage\_whenObjectRemoved() {

// Arrange

var cell = new Cell(messageSender);

var messageToBeSent = new CellMessage(CellMessage.Type.OBJECT\_LEAVED, cell, gameObject);

cell.addObject(gameObject);

// Act

cell.removeObject(gameObject);

// Assert

verify(messageSender).emitMessage(cell, messageToBeSent);

}

@Test

void beEmpty\_afterCreation() {

// Arrange

var cell = new Cell(messageSender);

// Act & Assert

assertTrue(cell.objects().isEmpty());

}

@Test

void returnNull\_whenNoNeighbor() {

// Arrange

var cell = new Cell(messageSender);

// Act & Assert

assertNull(cell.neighborCell(Direction.EAST));

assertNull(cell.neighborDirection(mock(Cell.class)));

}

}

Листинг 11. Реализация тестов класса «Коза»

package model;

import events.MessageSender;

import model.exceptions.NoEnoughStepsException;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import utils.Direction;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.\*;

class GoatShould {

private Cell cell1;

private Cell cell2;

private Direction direction;

private GameObject solidGameObject;

private GameObject notSolidGameObject;

@BeforeEach

void beforeEach() {

// Создание клеток поля для имитации движения козы

var messageSender = mock(MessageSender.class);

cell1 = new Cell(messageSender);

cell2 = new Cell(messageSender);

direction = Direction.EAST;

cell1.setNeighbor(cell2, direction);

// Настройка поведения мок объекта, который ведёт себя как твёрдый объект

solidGameObject = mock(GameObject.class);

when(solidGameObject.isSolid()).thenReturn(true);

// Настройка поведения мок объекта, который ведёт себя как нетвёрдый объект

notSolidGameObject = mock(GameObject.class);

when(notSolidGameObject.isSolid()).thenReturn(false);

}

@Test

void haveEnoughSteps\_whenStepsHasLargerOrEqualCount() {

// Arrange

var goat1 = new Goat(Goat.STEP\_COST \* 4);

var goat2 = new Goat(Goat.STEP\_COST + 1);

var goat3 = new Goat(Goat.STEP\_COST);

// Act & Assert

assertTrue(goat1.hasEnoughSteps());

assertTrue(goat2.hasEnoughSteps());

assertTrue(goat3.hasEnoughSteps());

}

@Test

void notHaveEnoughSteps\_whenStepCounterHasSmallerThanStepCost() {

// Arrange

var goat = new Goat(Goat.STEP\_COST - 1);

// Act & Assert

assertFalse(goat.hasEnoughSteps());

}

@Test

void move\_toEmptyCell() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4);

goat.setPosition(cell1);

// Act

goat.move(direction);

// Assert

assertSame(cell2, goat.cell());

}

@Test

void move\_toCellWithNotSolidObject() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4);

goat.setPosition(cell1);

cell2.addObject(notSolidGameObject);

// Act

goat.move(direction);

// Assert

assertSame(cell2, goat.cell());

}

@Test

void notMove\_toCellWithSolidObject() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4);

goat.setPosition(cell1);

cell2.addObject(solidGameObject);

// Act

goat.move(direction);

// Assert

assertSame(cell1, goat.cell());

}

@Test

void addItselfToNewCell\_whenMoveSuccessful() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4);

goat.setPosition(cell1);

var objectsBeforeMove = cell2.objects();

// Act

goat.move(direction);

// Assert

assertFalse(objectsBeforeMove.contains(goat));

assertTrue(cell2.objects().contains(goat));

}

@Test

void removeItselfFromLastCell\_whenMoveSuccessful() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4);

goat.setPosition(cell1);

var objectsBeforeMove = cell1.objects();

// Act

goat.move(direction);

// Assert

assertTrue(objectsBeforeMove.contains(goat));

assertFalse(cell1.objects().contains(goat));

}

@Test

void notAddItselfToNewCell\_whenMoveNotSuccessful() {

// Arrange

var goat = spy(new Goat(Goat.STEP\_COST \* 4));

goat.setPosition(cell1);

// Намеренно запрещаем козе куда-либо передвигаться

when(goat.canMoveTo(direction)).thenReturn(false);

// Act

goat.move(direction);

// Assert

assertFalse(cell2.objects().contains(goat));

}

@Test

void notRemoveItselfFromLastCell\_whenMoveNotSuccessful() {

// Arrange

var goat = spy(new Goat(Goat.STEP\_COST \* 4));

goat.setPosition(cell1);

// Намеренно запрещаем козе куда-либо передвигаться

when(goat.canMoveTo(direction)).thenReturn(false);

// Act

goat.move(direction);

// Assert

assertTrue(cell1.objects().contains(goat));

}

@Test

void beSolid() {

// Arrange & Act & Assert

assertTrue(new Goat(Goat.STEP\_COST \* 4).isSolid());

}

@Test

void initializeWithGivenSteps() {

// Arrange

var steps = 15;

// Act

var goat = new Goat(steps);

// Assert

assertEquals(steps, goat.steps());

}

@Test

void throw\_whenNotEnoughSteps() {

// Arrange

var initialSteps = 0;

var goat = new Goat(initialSteps);

// Act & Assert

assertThrows(NoEnoughStepsException.class, goat::decreaseSteps);

}

}

Листинг 12. Реализация тестов игрового класса

package model;

import events.MessageSender;

import model.exceptions.NoEnoughStepsException;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import utils.Direction;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.\*;

class GoatShould {

private Cell cell1;

private Cell cell2;

private Direction direction;

private GameObject solidGameObject;

private GameObject notSolidGameObject;

@BeforeEach

void beforeEach() {

// Создание клеток поля для имитации движения козы

var messageSender = mock(MessageSender.class);

cell1 = new Cell(messageSender);

cell2 = new Cell(messageSender);

direction = Direction.EAST;

cell1.setNeighbor(cell2, direction);

// Настройка поведения мок объекта, который ведёт себя как твёрдый объект

solidGameObject = mock(GameObject.class);

when(solidGameObject.isSolid()).thenReturn(true);

// Настройка поведения мок объекта, который ведёт себя как нетвёрдый объект

notSolidGameObject = mock(GameObject.class);

when(notSolidGameObject.isSolid()).thenReturn(false);

}

@Test

void haveEnoughSteps\_whenStepsHasLargerOrEqualCount() {

// Arrange

var goat1 = new Goat(Goat.STEP\_COST \* 4);

var goat2 = new Goat(Goat.STEP\_COST + 1);

var goat3 = new Goat(Goat.STEP\_COST);

// Act & Assert

assertTrue(goat1.hasEnoughSteps());

assertTrue(goat2.hasEnoughSteps());

assertTrue(goat3.hasEnoughSteps());

}

@Test

void notHaveEnoughSteps\_whenStepCounterHasSmallerThanStepCost() {

// Arrange

var goat = new Goat(Goat.STEP\_COST - 1);

// Act & Assert

assertFalse(goat.hasEnoughSteps());

}

@Test

void move\_toEmptyCell() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4);

goat.setPosition(cell1);

// Act

goat.move(direction);

// Assert

assertSame(cell2, goat.cell());

}

@Test

void move\_toCellWithNotSolidObject() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4);

goat.setPosition(cell1);

cell2.addObject(notSolidGameObject);

// Act

goat.move(direction);

// Assert

assertSame(cell2, goat.cell());

}

@Test

void notMove\_toCellWithSolidObject() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4);

goat.setPosition(cell1);

cell2.addObject(solidGameObject);

// Act

goat.move(direction);

// Assert

assertSame(cell1, goat.cell());

}

@Test

void addItselfToNewCell\_whenMoveSuccessful() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4);

goat.setPosition(cell1);

var objectsBeforeMove = cell2.objects();

// Act

goat.move(direction);

// Assert

assertFalse(objectsBeforeMove.contains(goat));

assertTrue(cell2.objects().contains(goat));

}

@Test

void removeItselfFromLastCell\_whenMoveSuccessful() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4);

goat.setPosition(cell1);

var objectsBeforeMove = cell1.objects();

// Act

goat.move(direction);

// Assert

assertTrue(objectsBeforeMove.contains(goat));

assertFalse(cell1.objects().contains(goat));

}

@Test

void notAddItselfToNewCell\_whenMoveNotSuccessful() {

// Arrange

var goat = spy(new Goat(Goat.STEP\_COST \* 4));

goat.setPosition(cell1);

// Намеренно запрещаем козе куда-либо передвигаться

when(goat.canMoveTo(direction)).thenReturn(false);

// Act

goat.move(direction);

// Assert

assertFalse(cell2.objects().contains(goat));

}

@Test

void notRemoveItselfFromLastCell\_whenMoveNotSuccessful() {

// Arrange

var goat = spy(new Goat(Goat.STEP\_COST \* 4));

goat.setPosition(cell1);

// Намеренно запрещаем козе куда-либо передвигаться

when(goat.canMoveTo(direction)).thenReturn(false);

// Act

goat.move(direction);

// Assert

assertTrue(cell1.objects().contains(goat));

}

@Test

void beSolid() {

// Arrange & Act & Assert

assertTrue(new Goat(Goat.STEP\_COST \* 4).isSolid());

}

@Test

void initializeWithGivenSteps() {

// Arrange

var steps = 15;

// Act

var goat = new Goat(steps);

// Assert

assertEquals(steps, goat.steps());

}

@Test

void throw\_whenNotEnoughSteps() {

// Arrange

var initialSteps = 0;

var goat = new Goat(initialSteps);

// Act & Assert

assertThrows(NoEnoughStepsException.class, goat::decreaseSteps);

}

}

# 4 Вторая итерация разработки

### 4.1 Функциональные требования (сценарии)

1. Главный сценарий - коза добралась до капусты

1. Пользователь инициирует начало игры.

2. Игра создаёт поле, состоящее из ячеек и размещает в них козу, капусту, ящики и стены.

3. **Делать**:

1. По желанию пользователя коза захватывает ящик в заданном пользователем направлении, если он там находится.

2. По желанию пользователя коза отпускает ящик, если ранее ей был захвачен ящик.

3. По указанию пользователя коза перемещается на соседнюю ячейку с уменьшением счётчика шагов или коза перемещается на соседнюю клетку с уменьшением счётчика шагов с ящиком, если он зацеплен, в заданном пользователем направлении.

**Пока** коза не добралась до ячейки с капустой и счётчик шагов != 0.

4. Игра определяет исход - игра успешна.

2. Сценарий - создание поля

1. Игра создаёт поле N x M клеток.

2. Игра создаёт и расставляет на поле стены.

3. Игра создаёт козу и наделяет её счётчиком шагов, ставит её на поле.

4. Игра создаёт капусту и ставит её на поле.

3. Сценарий - коза перемещается на соседнюю ячейку с уменьшением счётчика шагов

1. Пользователь задаёт направление перемещения козы.
2. Коза спрашивает у ячейки соседнюю ячейку в данном направлении.
3. Текущая ячейка сообщает соседнюю ячейку козе.
4. Коза спрашивает у соседней ячейки все объекты, которые находятся в ней.
5. Коза проверяет, что все объекты в соседней клетке не являются препятствиями.
6. Коза проверяет, что у неё достаточно шагов.
7. Коза просит текущую ячейку изъять себя из неё.
8. Текущая ячейка извлекает козу из себя.
9. Коза просит соседнюю ячейку поместить себя в неё.
10. Соседняя ячейка помещает себя в неё.
11. Коза уменьшает количество шагов.

3.1. Альтернативный сценарий - соседней ячейки в заданном направлении нет

1. Сценарий начинается после п. 2 сценария 3
2. Текущая ячейка сообщает, что у неё нет соседней ячейки.
3. Сценарий переходит к п. 3. главного сценария

3.2. Альтернативный сценарий - в соседней клетке есть препятствие

1. Сценарий начинается после п. 5 сценария 3
2. Коза проверила - в клетке есть препятствие
3. Сценарий переходит к п. 3. главного сценария

3.3 Альтернативный сценарий - у козы недостаточно шагов

1. Сценарий начинается после п. 6 сценария 3
2. Коза сообщает, что у неё недостаточно шагов для перемещения.
3. Сценарий переходит к п. 4 главного сценария

4. Сценарий - коза перемещается на соседнюю клетку с уменьшением счётчика шагов с ящиком, если он зацеплен

1. Пользователь задаёт направление перемещения козы.
2. Коза спрашивает у ячейки соседнюю ячейку в данном направлении.
3. Текущая ячейка сообщает соседнюю ячейку козе.
4. Коза спрашивает у соседней ячейки все объекты, которые находятся в ней.
5. Коза проверяет, что все объекты в соседней клетке не являются препятствиями, не учитывая зацепленный ящик.
6. Коза проверяет, что у неё достаточно шагов.
7. Коза спрашивает у зацепленного ящика может ли он переместиться в данном направлении.
8. Зацепленный ящик отвечает, что может.
9. Коза просит текущую ячейку изъять себя из неё.
10. Текущая ячейка извлекает козу из себя.
11. Коза просит соседнюю ячейку поместить себя в неё.
12. Соседняя ячейка помещает себя в неё.
13. Коза просит зацепленный ящик переместиться.
14. Коза уменьшает количество шагов.

4.1. Альтернативный сценарий - соседней ячейки в заданном направлении нет

1. Сценарий начинается после п. 2 сценария 4
2. Текущая ячейка сообщает, что у неё нет соседней ячейки.
3. Сценарий переходит к п. 3. главного сценария

4.2. Альтернативный сценарий - в соседней клетке есть препятствие

1. Сценарий начинается после п. 5 сценария 4
2. Коза проверила - в клетке есть препятствие
3. Сценарий переходит к п. 3. главного сценария

4.3 Альтернативный сценарий - у козы недостаточно шагов

1. Сценарий начинается после п. 6 сценария 4
2. Коза сообщает, что у неё недостаточно шагов для перемещения.
3. Сценарий переходит к п. 4 главного сценария

4.4 Альтернативный сценарий - зацепленный ящик не может переместиться

1. Сценарий начинается после п. 7 сценария 4.
2. Зацепленный ящик сообщает, что не может переместиться.
3. Сценарий переходит к п. 4 главного сценария.

5. Сценарий - металлический ящик проверяет возможность перемещения в направлении

1. Ящик спрашивает у текущей клетки соседнюю клетку в направлении
2. Текущая клетка сообщает соседнюю
3. Ящик проверяет, что в соседней ячейке нет препятствий
4. Ящик спрашивает остальные зацепленные ящики, могут ли они переместиться
5. Все зацепленные ящики могут переместиться
6. Ящик сообщает, что может переместиться

5.1. Альтернативный сценарий - соседней ячейки в заданном направлении нет

1. Сценарий начинается после п. 1 сценария 4
2. Текущая ячейка сообщает, что у неё нет соседней ячейки.
3. Ящик сообщает, что не может переместиться

5.2. Альтернативный сценарий - в соседней клетке есть препятствие

1. Сценарий начинается после п. 3 сценария 4
2. Ящик проверил - в клетке есть препятствия
3. Ящик сообщает, что не может переместиться

5.3. Альтернативный сценарий - зацепленный ящик не может переместиться

1. Сценарий начинается после п. 4 сценария 4
2. Один (или более) зацепленных ящиков сказали, что не могут переместиться
3. Ящик сообщает, что не может переместиться

6. Сценарий - магнитный ящик проверяет возможность перемещения в направлении

1. Ящик спрашивает у текущей клетки соседнюю клетку в направлении
2. Текущая клетка сообщает соседнюю
3. Ящик проверяет, что в соседней ячейке нет препятствий
4. Ящик спрашивает у соседней ячейки её соседей.
5. Ящик проверяет, что в соседней клетке нет других магнитных объектов, полюса которых совпадают с полюсами ящика.
6. Ящик спрашивает остальные зацепленные ящики, могут ли они переместиться
7. Все зацепленные ящики могут переместиться
8. Ящик сообщает, что может переместиться

6.1. Альтернативный сценарий - соседней ячейки в заданном направлении нет

1. Сценарий начинается после п. 1 сценария 4
2. Текущая ячейка сообщает, что у неё нет соседней ячейки.
3. Ящик сообщает, что не может переместиться

6.2. Альтернативный сценарий - в соседней клетке есть препятствие

1. Сценарий начинается после п. 3 сценария 4
2. Ящик проверил - в клетке есть препятствия
3. Ящик сообщает, что не может переместиться

6.3. Альтернативный сценарий - есть магнитный объект, чей полюс совпадает с полюсом ящика

1. Сценарий начинается после п. 3 сценария 4
2. Ящик проверил - есть магнитный объект, чей полюс совпадает с полюсом ящика
3. Ящик сообщает, что не может переместиться

6.4. Альтернативный сценарий - зацепленный ящик не может переместиться

1. Сценарий начинается после п. 6 сценария 4
2. Один (или более) зацепленных ящиков сказали, что не могут переместиться
3. Ящик сообщает, что не может переместиться

7. Сценарий - игра определяет исход - игра успешна

1. Игра спрашивает у поля ячейку с капустой
2. Поле сообщает ячейку с капустой
3. Игра спрашивает у ячейки с капустой находится ли в ней коза
4. Ячейка сообщает, что в ней находится коза
5. Игра определяет исход - “успешна”

7.1. Альтернативный сценарий - игра определяет исход - шаги кончились, а коза не в ячейке с капустой

1. Сценарий начинается после п. 3 сценария 4
2. Ячейка сообщает, что в ней нет козы
3. Игра спрашивает у козы счётчик шагов
4. Игра спрашивает у счётчика шагов сколько осталось шагов
5. Счётчик сообщает, что шаги кончились

Игра определяет исход - “неуспешна”

### 4.2 Словарь предметной области

* **Игра**. Знает о поле. Управляет игровым циклом: определяет окончание игры и её исход. Определяет начальную расстановку элементов игрового поля (коза, капуста, стены…).
* **Поле** – прямоугольная область, состоящая из ячеек. Знает о ячейке с капустой и козе.
* **Ячейка** – квадратная область поля. Знает о четырёх соседних ячейках. Может содержать в себе либо стену, либо козу.
* **Коза** перемещается между соседними ячейками. Для перемещения использует счётчик шагов. Занимает одну ячейку поля.
* **Стена** содержится в ячейке. Является препятствием. Не может перемещаться.
* **Капуста** содержится в ячейку. Не является препятствием. Является целью козы. Не может перемещаться.
* **Ящик** содержится в ячейке. Является препятствием. Может перемещаться под действием козы.
* **Металлический ящик** - разновидность ящика. Автоматически цепляется к магнитному ящику.
* **Магнитный ящик** - разновидность ящика. На каждой грани расположен магнитный полюс - северный или южный. Может цепляться к другому магнитному ящику, если на полюс другого ящика противоположный.

### 4.3 Структура программы на уровне классов

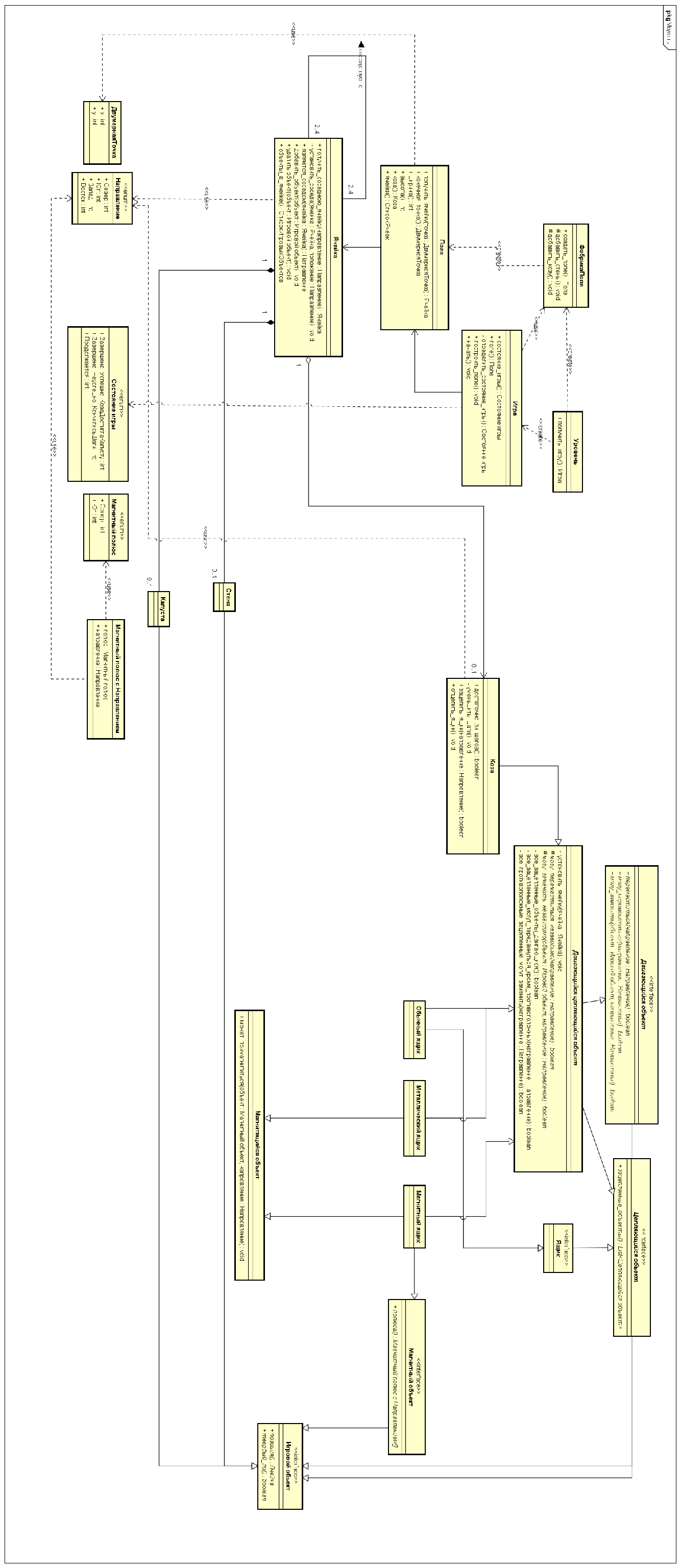


Рис. 14. Диаграмма классов модели (полная версия)

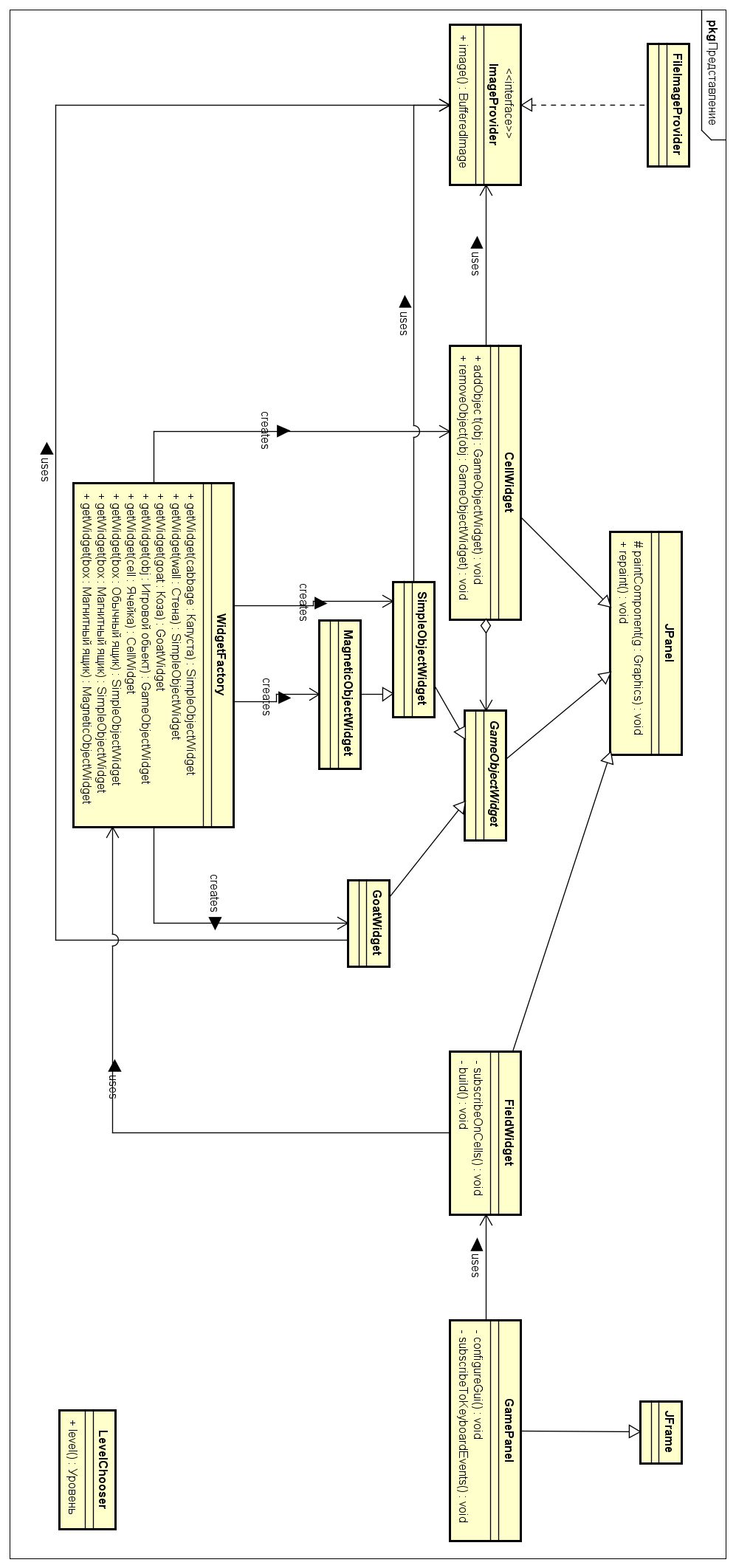


Рис. 15. Диаграмма классов представления (полная версия)

### 4.4 Типовые процессы в программе

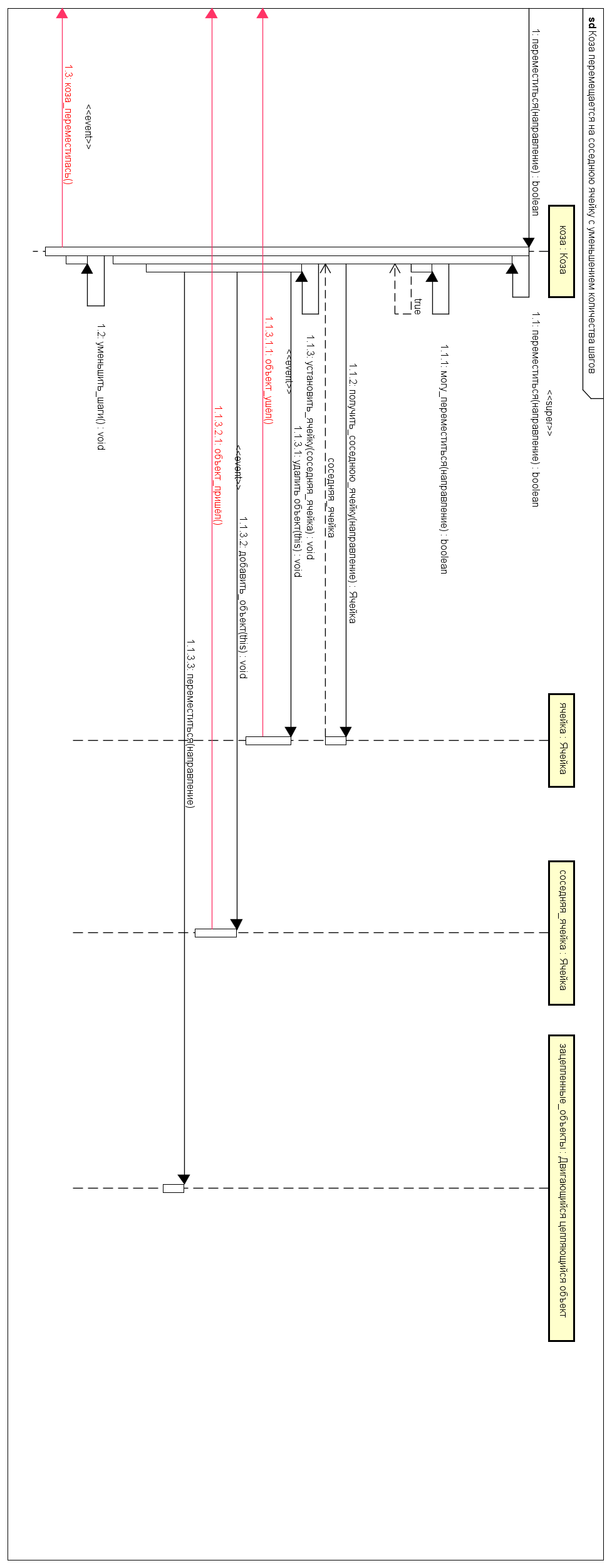


Рис. 16. Диаграмма последовательности модели «Коза перемещается на соседнюю ячейку с уменьшением количества шагов».

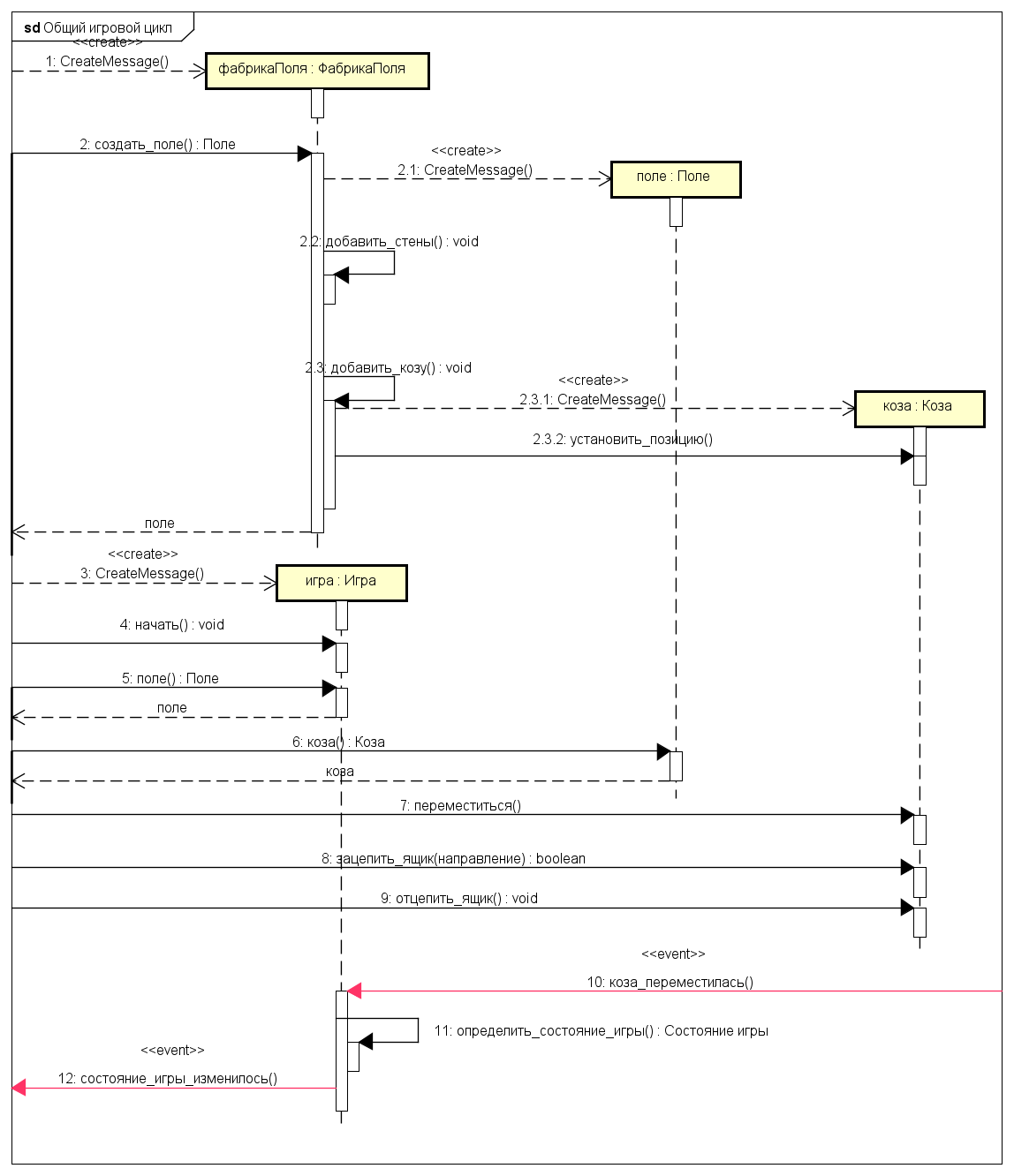


Рис. 17. Диаграмма последовательности модели «Общий игровой цикл»

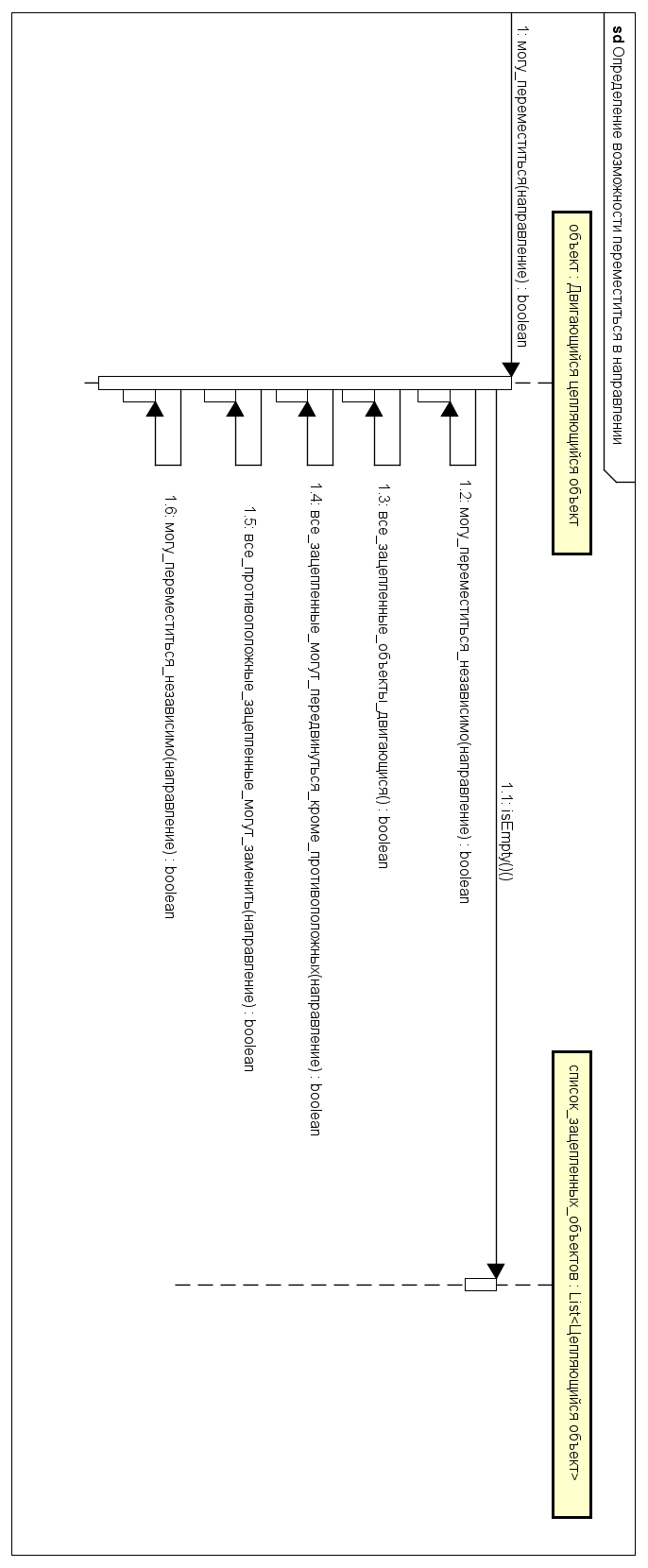


Рис. 18. Диаграмма последовательности модели «Определение возможности переместиться в направлении»

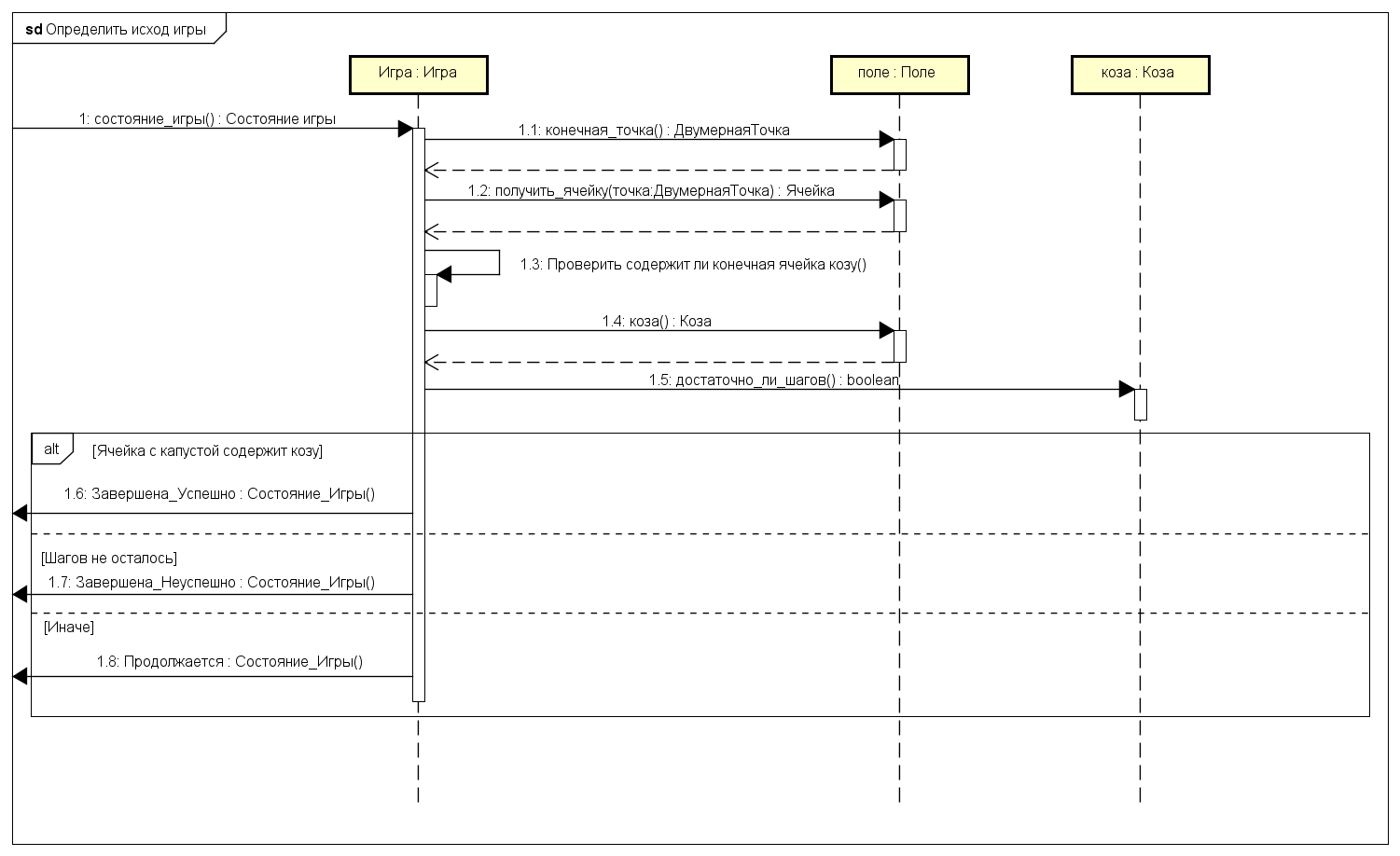


Рис. 19. Диаграмма последовательности модели «Определить исход игры»

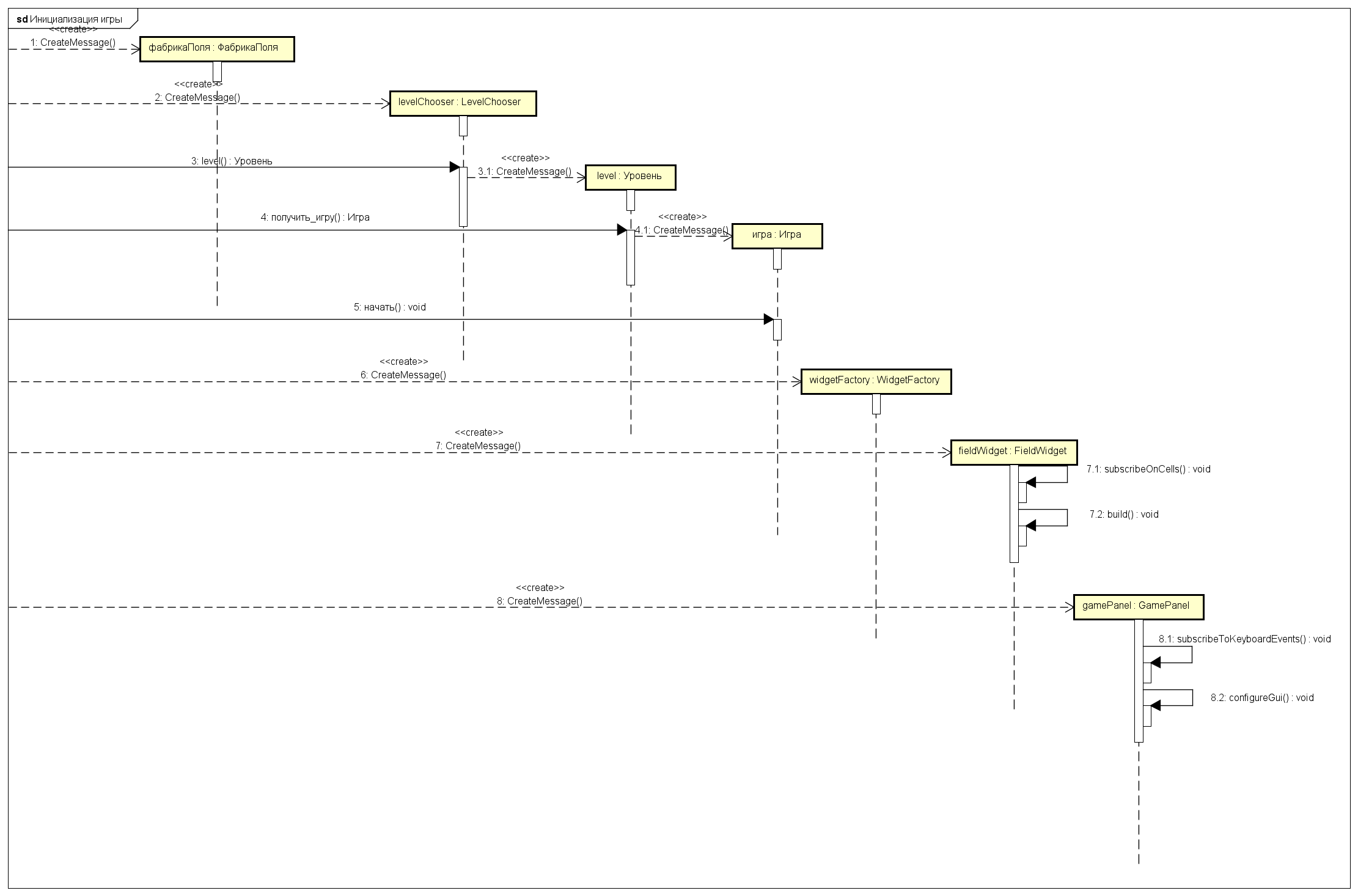


Рис. 20. Диаграмма последовательности представления «Инициализация игры»

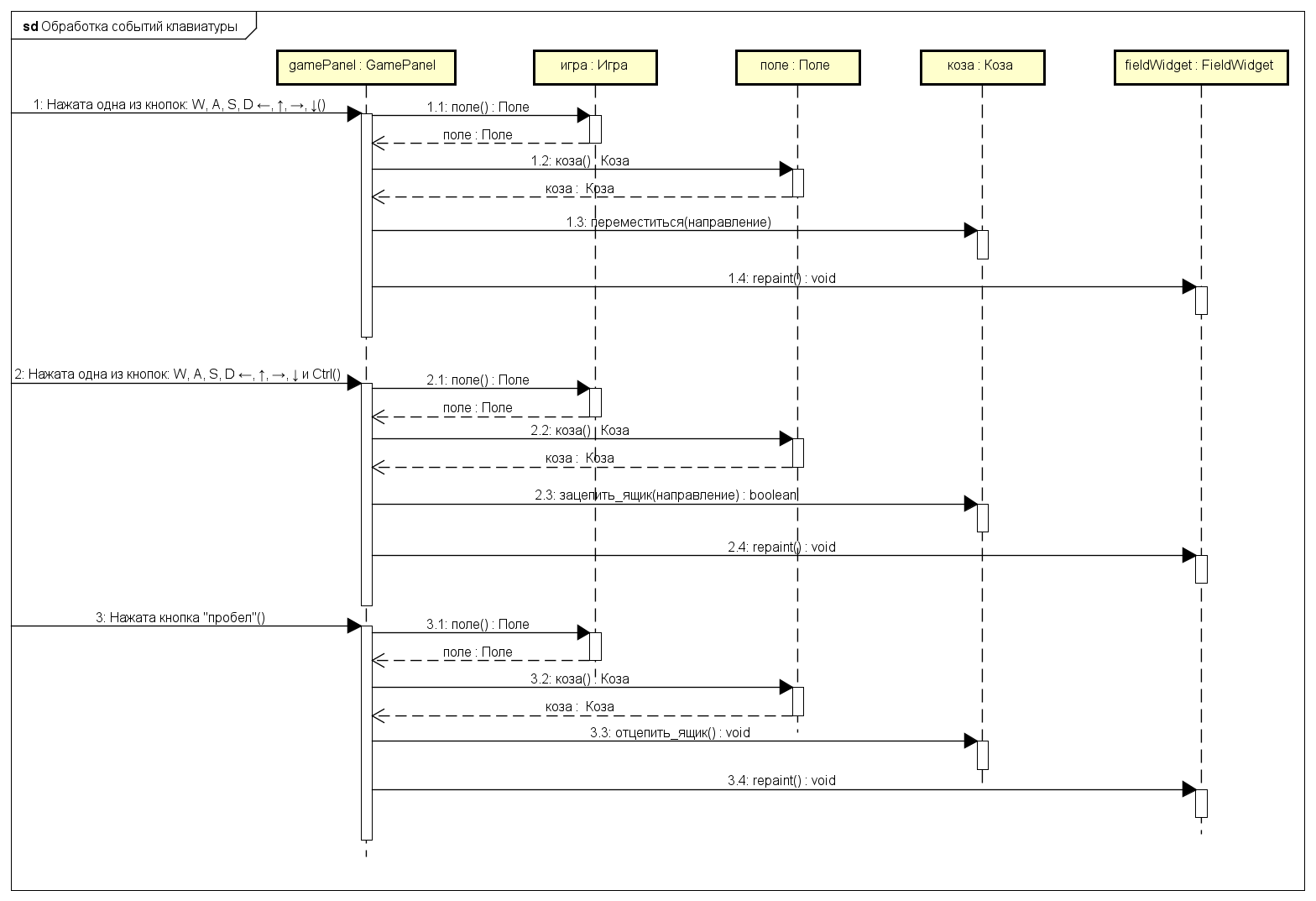


Рис. 21. Диаграмма последовательности представления «Обработка событий клавиатуры»

### 4.5 Человеко-машинное взаимодействие

1. Начальный экран - Выбор уровня

При запуске игры можно выбрать уровень для игры.

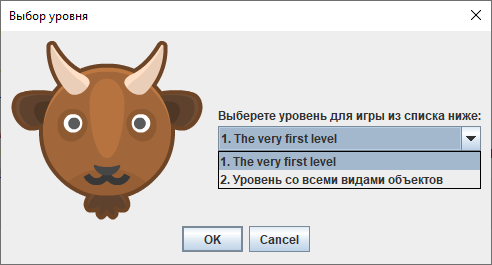


Рис. 22. Начальный экран - Выбор уровня

1. Начальное состояние игры

После выбора появляется начальное состояние уровня. Количество шагов козы указывается слева вверху.

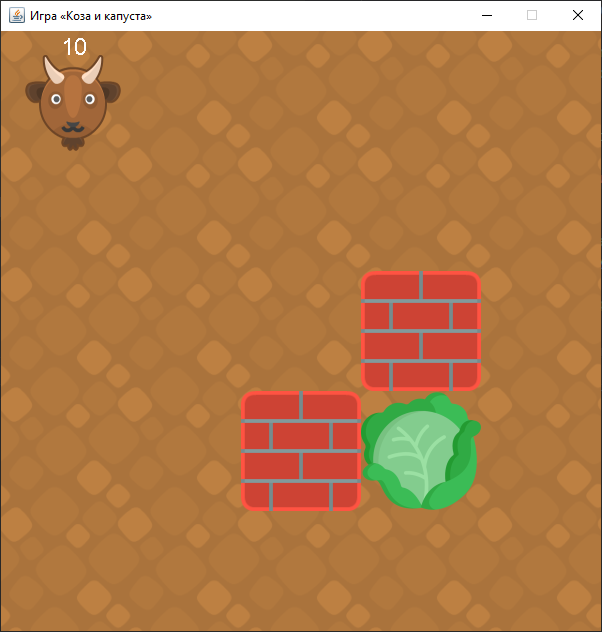


Рис. 23. Начальное состояние игры

1. Состояние игры после хода

Движение козы осуществляется клавишами:

W, ↑ - вверх;

A, ← - влево;

S, ↓ - вниз;

D, → - вправо.

После движения козы её количество шагов уменьшается.

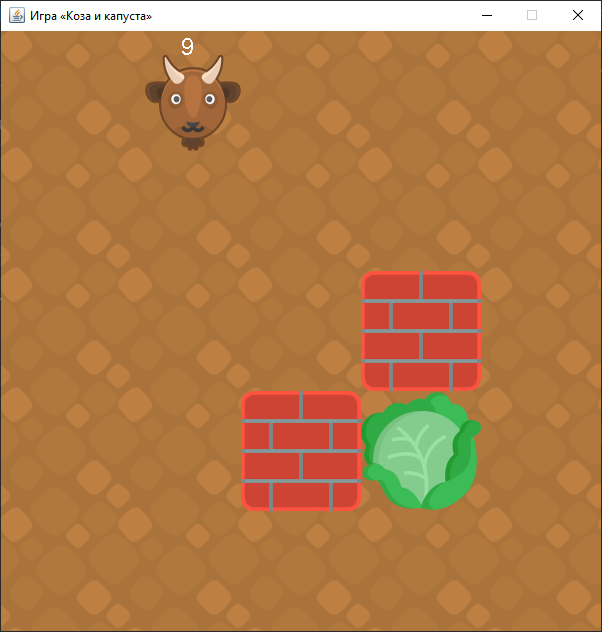


Рис. 24. Состояние игры после хода.

1. Состояние игры после перемещения козы с ящиком

Коза может осуществлять захват ящиков. Для этого надо зажать клавишу Ctrl и использовать кнопки из предыдущего пункта для указания направления, в котором захватить ящик.

Чтобы отпустить ящик надо нажать клавишу "пробел".

Перемещение козы с одним ящиком:

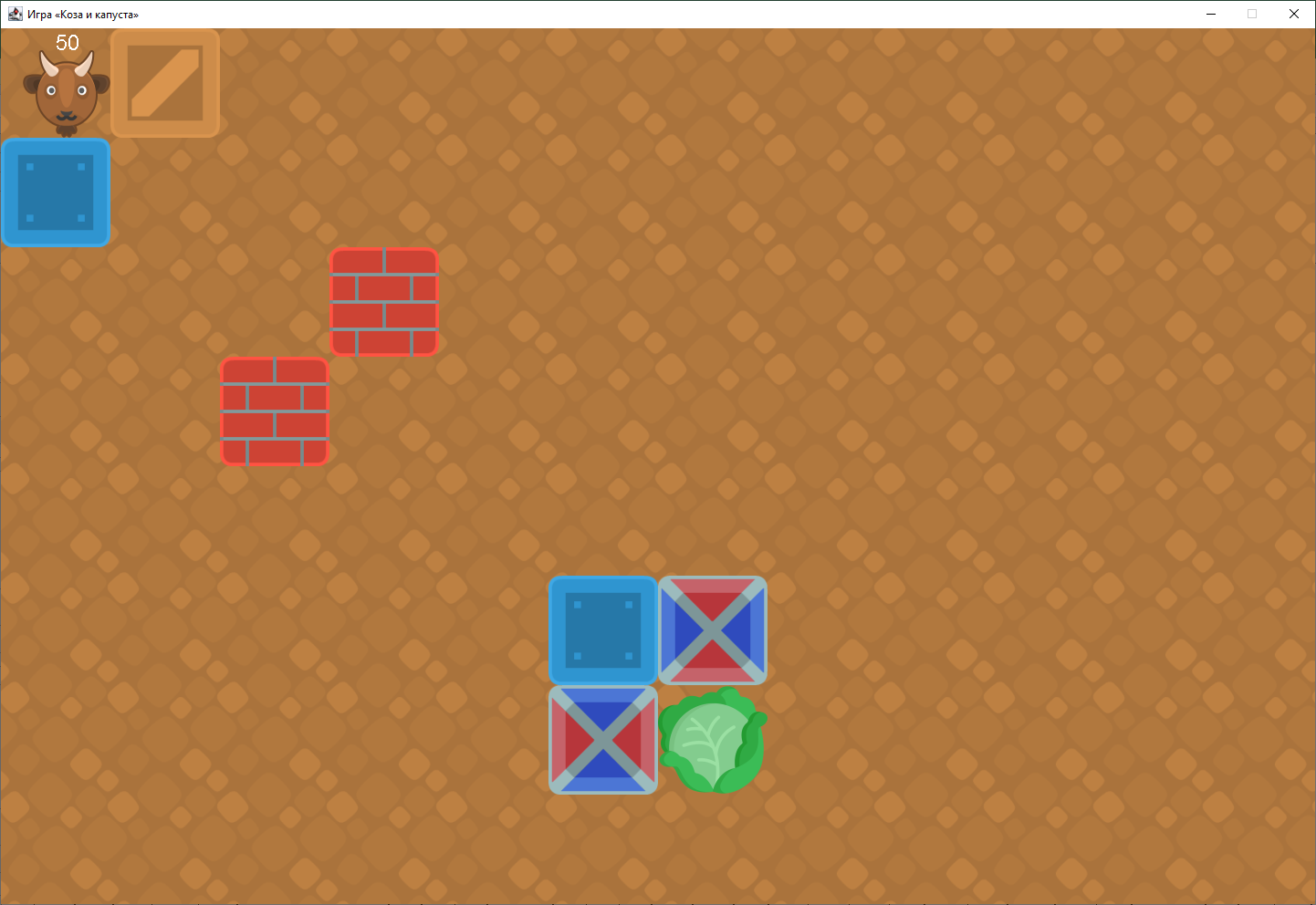


Рис. 25. Игровое поле до перемещения козы с захваченным ящиком

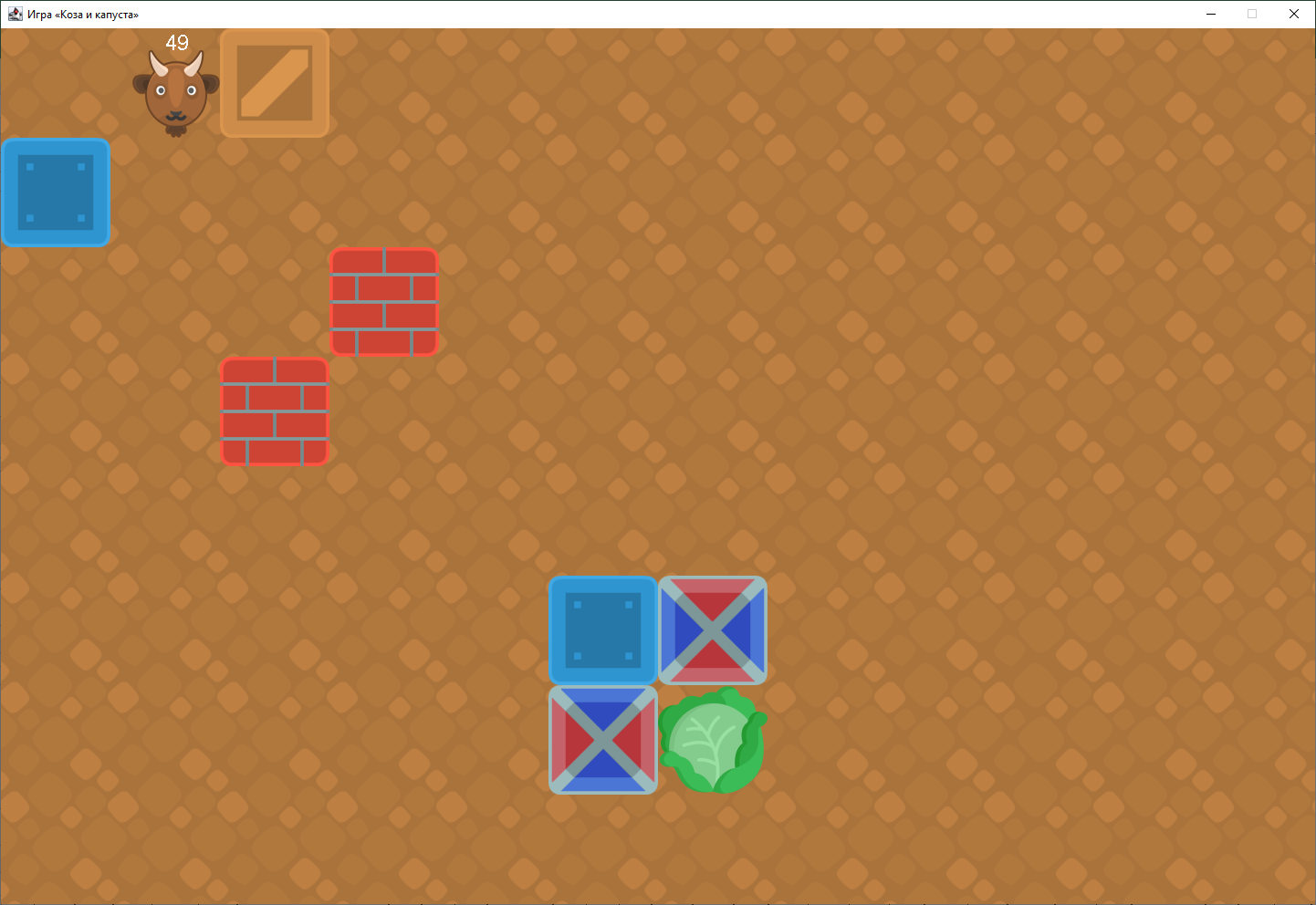


Рис. 26. Игровое поле после перемещения козы с захваченным ящиком

Перемещение козы с несколькими ящиками:



Рис. 27. Игровое поле до перемещения козы с захваченными ящиками

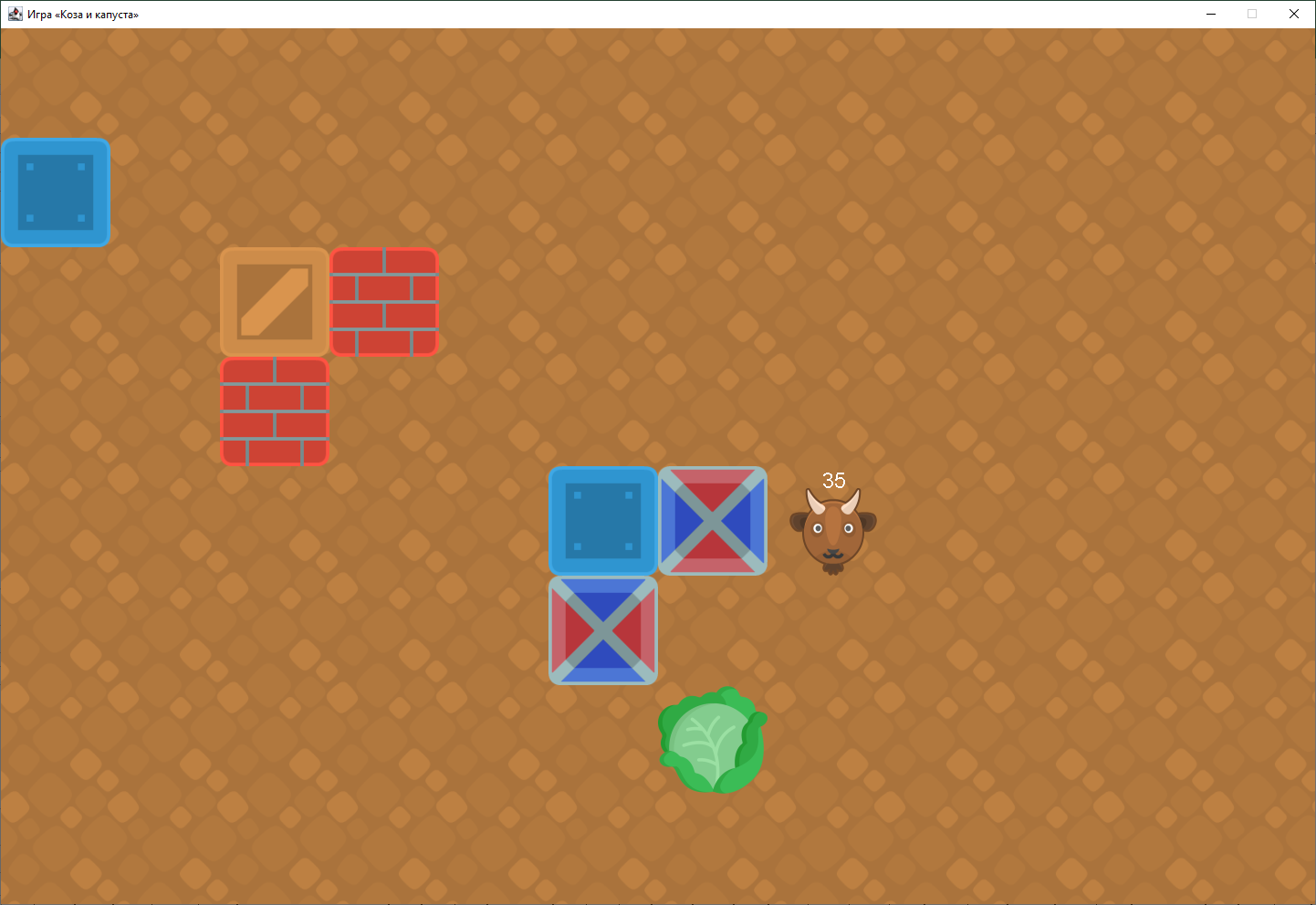


Рис. 28. Игровое поле после перемещения козы с захваченными ящиками

1. Игра завершена успешно - коза достигла капусты

При достижении козой капусты высвечивается MessageBox с сообщением об успешном завершении игры и окно игры закрывается.

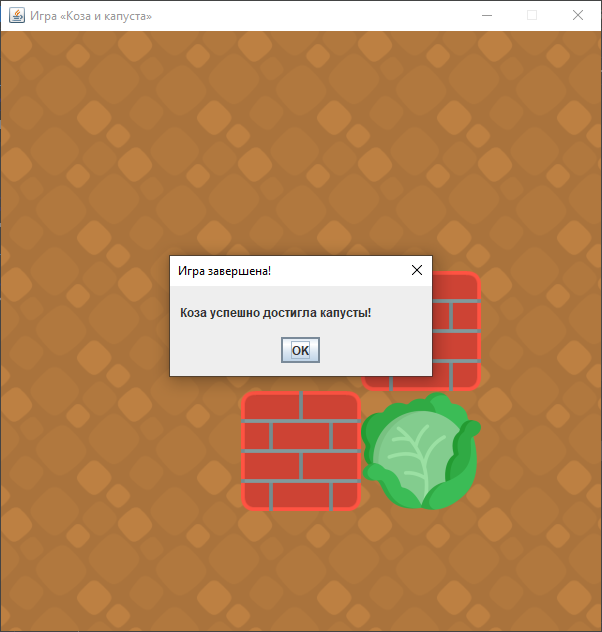


Рис. 29. Игра завершена успешно - коза достигла капусты

1. Игра завершена неуспешно - у козы кончились шаги

Когда у козы закончились шаги, и она не достигла капусты, высвечивается MessageBox с сообщением о неуспешном завершении игры и окно игры закрывается.

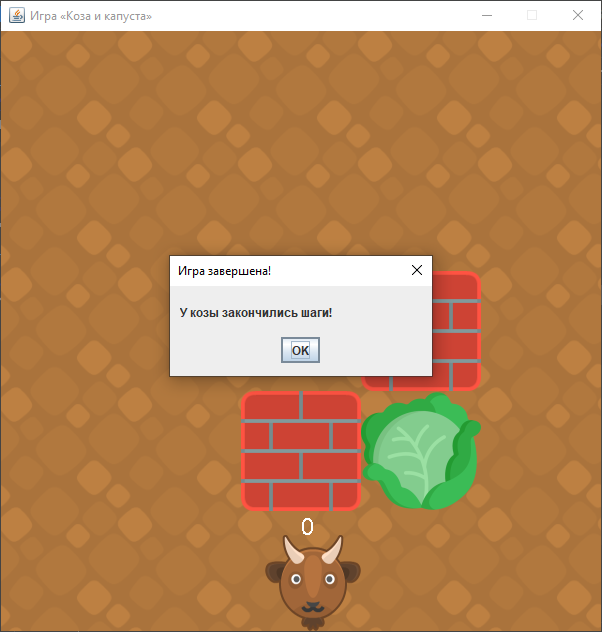


Рис. 30. Игра завершена неуспешно - у козы кончились шаги

### 4.6 Реализация ключевых классов

Листинг 13. Реализация абстрактного класса двигающегося цепляющегося объекта

package xyz.unterumarmung.model.objects;

import org.jetbrains.annotations.Contract;

import org.jetbrains.annotations.NotNull;

import xyz.unterumarmung.model.Cell;

import xyz.unterumarmung.utils.Direction;

import xyz.unterumarmung.utils.Pair;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.stream.Collectors;

import java.util.stream.Stream;

import static java.util.stream.Collectors.\*;

public abstract class MovableHookable implements MovableObject, HookableObject {

private Cell cell;

public MovableHookable(Cell cell) {

setCell(cell);

}

@Override

public boolean move(@NotNull Direction direction) {

if (canMoveTo(direction)) {

var allHooked = collectAllHooked();

for (var hooked : allHooked) {

var newCell = hooked.cell().neighborCell(direction);

hooked.setCell(newCell);

}

return true;

}

return false;

}

private Set<MovableHookable> collectAllHooked() {

var allHooked = new HashSet<MovableHookable>();

allHooked.add(this);

collectAllHooked(allHooked);

return allHooked;

}

private void collectAllHooked(Set<MovableHookable> hooked) {

var objs = castHookedToMovableHookable();

for (var obj : objs.stream().map(pair -> pair.first).filter(obj -> !hooked.contains(obj)).collect(toList())) {

hooked.add(obj);

obj.collectAllHooked(hooked);

}

}

private List<Pair<MovableHookable, Direction>> castHookedToMovableHookable() {

return hookedObjects().stream().map(Pair::<MovableHookable>castFirst).collect(toList());

}

void setCell(Cell cell) {

if (this.cell != null)

this.cell.removeObject(this);

if (cell != null)

cell.addObject(this);

this.cell = cell;

}

@Override

public boolean canMoveTo(@NotNull Direction direction) {

return canMoveTo(direction, new HashSet<>());

}

private <T extends HookableObject> List<Pair<T, Direction>> filterObjects(Stream<Pair<T, Direction>> objs, Set<MovableHookable> except) {

return objs.filter(pair -> !except.contains(pair.first)).collect(toList());

}

private boolean canMoveTo(@NotNull Direction direction, Set<MovableHookable> exceptObjects) {

var hookedObjects = filterObjects(hookedObjects().stream(), exceptObjects);

if (hookedObjects.isEmpty() && canMoveToIndependent(direction))

return true;

if (!allHookedAreMovable(hookedObjects))

return false;

var movableHookedObjects = castHookedToMovable(hookedObjects);

exceptObjects.add(this);

var canMove = allHookedCanMoveExceptOppositeObjects(movableHookedObjects, direction, exceptObjects)

&& allOppositeObjectsCanReplaceThis(movableHookedObjects, direction, exceptObjects);

if (!isAnyOfHookedInDirection(hookedObjects, direction))

canMove &= canMoveToIndependent(direction);

return canMove;

}

@Override

public boolean canReplace(@NotNull GameObject gameObject, @NotNull Direction direction) {

return canReplace(gameObject, direction, new HashSet<>());

}

private boolean canReplace(@NotNull GameObject gameObject, @NotNull Direction direction, Set<MovableHookable> exceptObjects) {

var hookedObjects = filterObjects(hookedObjects().stream(), exceptObjects);

if (hookedObjects.isEmpty() && canReplaceIndependent(gameObject, direction))

return true;

if (!allHookedAreMovable(hookedObjects))

return false;

var movableHookedObjects = castHookedToMovable(hookedObjects);

exceptObjects.add(this);

return allOppositeObjectsCanReplaceThis(movableHookedObjects, direction, exceptObjects) && canReplaceIndependent(gameObject, direction);

}

protected abstract boolean canReplaceIndependent(@NotNull GameObject gameObject, @NotNull Direction direction);

private boolean isAnyOfHookedInDirection(List<Pair<HookableObject, Direction>> hookedObjects, @NotNull Direction direction) {

return hookedObjects.stream().anyMatch(hookedObject -> hookedObject.second == direction);

}

private List<Pair<MovableHookable, Direction>> castHookedToMovable(@NotNull List<Pair<HookableObject, Direction>> hookedObjects) {

return hookedObjects.stream().map(Pair::<MovableHookable>castFirst).collect(toList());

}

private boolean allHookedAreMovable(List<Pair<HookableObject, Direction>> hookedObjects) {

return hookedObjects.stream().allMatch(hookedObject -> hookedObject.first instanceof MovableHookable);

}

private boolean allHookedCanMoveExceptOppositeObjects(@NotNull List<Pair<MovableHookable, Direction>> movableHookedObjects, @NotNull Direction direction, Set<MovableHookable> except) {

return movableHookedObjects.stream()

.filter(hookedObject -> hookedObject.second != direction.opposite())

.allMatch(hookedObject -> hookedObject.first.canMoveTo(direction, except));

}

private boolean allOppositeObjectsCanReplaceThis(@NotNull List<Pair<MovableHookable, Direction>> movableHookedObjects, @NotNull Direction direction, Set<MovableHookable> except) {

return movableHookedObjects.stream()

.filter(hookedObject -> hookedObject.second == direction.opposite())

.allMatch(hookedObject -> hookedObject.first.canReplace(this, direction, except));

}

protected abstract boolean canMoveToIndependent(@NotNull Direction direction);

protected boolean noneSolidInCell(Cell cellToMove) {

return cellToMove != null && cellToMove.objects().stream().noneMatch(GameObject::isSolid);

}

protected boolean noneSolidInCellExcept(Cell cell, GameObject gameObject) {

return cell != null

&& cell.objects().stream().filter(o -> o != gameObject).noneMatch(GameObject::isSolid);

}

@Contract(pure = true)

@Override

public Cell cell() {

return cell;

}

@Override

public int hashCode() {

return hash();

}

}

Листинг 14. Реализация класса козы

package xyz.unterumarmung.model.objects;

import org.jetbrains.annotations.NotNull;

import xyz.unterumarmung.events.MessageSender;

import xyz.unterumarmung.events.MessageSource;

import xyz.unterumarmung.model.Cell;

import xyz.unterumarmung.model.events.GoatMessage;

import xyz.unterumarmung.model.exceptions.NoEnoughStepsException;

import xyz.unterumarmung.utils.Direction;

import xyz.unterumarmung.utils.Pair;

import xyz.unterumarmung.utils.collections.ReadOnlyList;

import static xyz.unterumarmung.utils.collections.ReadOnlyList.empty;

import static xyz.unterumarmung.utils.collections.ReadOnlyList.of;

public class Goat extends MovableHookable implements SolidObject, MovableObject, MessageSource {

static final int STEP\_COST = 1;

private final @NotNull MessageSender messageSender;

private int steps;

private Pair<Box, Direction> hookedBox;

public Goat(int initialSteps, Cell initialCell, @NotNull MessageSender messageSender) {

super(initialCell);

steps = initialSteps;

this.messageSender = messageSender;

}

public boolean hookBox(@NotNull Direction direction) {

var possibleBox = cell().neighborCell(direction).objects().stream().filter(gameObject -> gameObject instanceof Box).findFirst();

if (possibleBox.isPresent()) {

hookedBox = new Pair<>((Box) possibleBox.get(), direction);

return true;

}

return false;

}

public void unhookBox() {

hookedBox = null;

}

@Override

public boolean move(@NotNull Direction direction) {

var cellFrom = cell();

var moved = super.move(direction);

if (moved) {

decreaseSteps();

messageSender.emitMessage(this, new GoatMessage(cellFrom, cell()));

}

return moved;

}

@Override

protected boolean canMoveToIndependent(@NotNull Direction direction) {

var neighbor = cell().neighborCell(direction);

return noneSolidInCell(neighbor) && hasEnoughSteps();

}

@Override

protected boolean canReplaceIndependent(@NotNull GameObject gameObject, @NotNull Direction direction) {

return noneSolidInCellExcept(cell().neighborCell(direction), gameObject);

}

public boolean hasEnoughSteps() {

return steps() - STEP\_COST >= 0;

}

private void assertHasSteps() {

if (!hasEnoughSteps())

throw new NoEnoughStepsException();

}

void decreaseSteps() {

assertHasSteps();

steps -= STEP\_COST;

}

public int steps() {

return steps;

}

@Override

public boolean equals(Object o) {

return this == o;

}

@Override

public @NotNull ReadOnlyList<Pair<HookableObject, Direction>> hookedObjects() {

if (hookedBox == null)

return empty();

else

return of(hookedBox.castFirst());

}

}

Листинг 15. Реализация класса простого ящика

package xyz.unterumarmung.model.objects;

import org.jetbrains.annotations.NotNull;

import xyz.unterumarmung.model.Cell;

import xyz.unterumarmung.utils.Direction;

import xyz.unterumarmung.utils.Pair;

import xyz.unterumarmung.utils.collections.ReadOnlyList;

import static xyz.unterumarmung.utils.collections.ReadOnlyList.empty;

public class SimpleBox extends MovableHookable implements Box {

public SimpleBox(@NotNull Cell initialCell) {

super(initialCell);

}

@Override

/// Так как мы знаем, что простой ящик никогда не захватывает объекты сам, то смысла в коде из MovableHookable нет

/// Так что override'им, чтобы упростить вычисления

public boolean canMoveTo(@NotNull Direction direction) {

return canMoveToIndependent(direction);

}

@Override

protected boolean canMoveToIndependent(@NotNull Direction direction) {

return noneSolidInCell(cell().neighborCell(direction));

}

@Override

protected boolean canReplaceIndependent(@NotNull GameObject gameObject, @NotNull Direction direction) {

return noneSolidInCellExcept(cell().neighborCell(direction), gameObject);

}

@Override

public @NotNull ReadOnlyList<Pair<HookableObject, Direction>> hookedObjects() {

return empty();

}

}

Листинг 16. Реализация класса металлического ящика

package xyz.unterumarmung.model.objects;

import org.jetbrains.annotations.NotNull;

import xyz.unterumarmung.model.Cell;

import xyz.unterumarmung.utils.Direction;

import xyz.unterumarmung.utils.Pair;

import xyz.unterumarmung.utils.collections.ReadOnlyList;

import java.util.List;

import java.util.stream.Collectors;

import static xyz.unterumarmung.utils.collections.ReadOnlyList.empty;

import static xyz.unterumarmung.utils.collections.ReadOnlyList.fromList;

public class MetalBox extends MovableHookable implements Box, SolidObject, MagnitableObject {

public MetalBox(Cell initialCell) {

super(initialCell);

}

@Override

public @NotNull ReadOnlyList<Pair<HookableObject, Direction>> hookedObjects() {

if (cell() == null)

return empty();

var hooked = getAllPossibleHooked();

return fromList(hooked);

}

@NotNull

private List<Pair<HookableObject, Direction>> getAllPossibleHooked() {

return cell().neighbours().stream()

.flatMap(cellWithDirection -> cellWithDirection.cell.objects().stream()

.filter(o -> o instanceof MagneticObject)

.map(o -> new Pair<>((HookableObject) o, cellWithDirection.direction)))

.collect(Collectors.toList());

}

@Override

protected boolean canReplaceIndependent(@NotNull GameObject gameObject, @NotNull Direction direction) {

return noneSolidInCellExcept(cell().neighborCell(direction), gameObject);

}

@Override

protected boolean canMoveToIndependent(@NotNull Direction direction) {

return noneSolidInCell(cell().neighborCell(direction));

}

@Override

public boolean isMagnitableTo(@NotNull MagneticObject magneticObject, @NotNull Direction direction) {

return true;

}

}

Листинг 17. Реализация класса магнитного ящика

package xyz.unterumarmung.model.objects;

import org.jetbrains.annotations.NotNull;

import xyz.unterumarmung.model.Cell;

import xyz.unterumarmung.utils.Direction;

import xyz.unterumarmung.utils.Pair;

import xyz.unterumarmung.utils.collections.ReadOnlyList;

import java.util.EnumMap;

import java.util.List;

import java.util.Map;

import java.util.stream.Collectors;

import static java.util.Collections.unmodifiableMap;

import static xyz.unterumarmung.model.objects.MagneticBox.Alignment.VERTICAL\_NORTH\_HORIZONTAL\_SOUTH;

import static xyz.unterumarmung.utils.collections.ReadOnlyList.empty;

import static xyz.unterumarmung.utils.collections.ReadOnlyList.fromList;

public class MagneticBox extends MovableHookable implements MagneticObject, MagnitableObject, Box {

private final @NotNull Map<Direction, MagneticPole> magneticPoles;

private final @NotNull Alignment alignment;

public MagneticBox(Cell cell) {

this(cell, VERTICAL\_NORTH\_HORIZONTAL\_SOUTH);

}

public MagneticBox(Cell cell, @NotNull Alignment alignment) {

super(cell);

magneticPoles = alignmentToPoles(alignment);

this.alignment = alignment;

}

@NotNull

private static Map<Direction, MagneticPole> alignmentToPoles(@NotNull Alignment alignment) {

var map = new EnumMap<Direction, MagneticPole>(Direction.class);

switch (alignment) {

case VERTICAL\_NORTH\_HORIZONTAL\_SOUTH -> {

map.put(Direction.NORTH, MagneticPole.NORTH);

map.put(Direction.EAST, MagneticPole.SOUTH);

map.put(Direction.SOUTH, MagneticPole.NORTH);

map.put(Direction.WEST, MagneticPole.SOUTH);

}

case VERTICAL\_SOUTH\_HORIZONTAL\_NORTH -> {

map.put(Direction.NORTH, MagneticPole.SOUTH);

map.put(Direction.EAST, MagneticPole.NORTH);

map.put(Direction.SOUTH, MagneticPole.SOUTH);

map.put(Direction.WEST, MagneticPole.NORTH);

}

}

return unmodifiableMap(map);

}

public Alignment alignment() {

return alignment;

}

@Override

public Map<Direction, MagneticPole> magneticPoles() {

return magneticPoles;

}

@Override

public boolean isMagnitableTo(@NotNull MagneticObject magneticObject, @NotNull Direction direction) {

var objectPoleInDirection = magneticObject.magneticPoles().get(direction.opposite());

if (objectPoleInDirection == null)

return false;

var minePoleInOppositeDirection = magneticPoles().get(direction);

if (minePoleInOppositeDirection == null)

return false;

return objectPoleInDirection == minePoleInOppositeDirection.opposite();

}

@Override

protected boolean canMoveToIndependent(@NotNull Direction direction) {

var cellToMove = cell().neighborCell(direction);

if (cellToMove == null)

return false;

var noneSolidInDirection = noneSolidInCell(cellToMove);

var neighbourObjectsWithDirection =

getNeighbourObjectsWithDirectionExceptOpposite(cellToMove, direction);

return noneSolidInDirection && noneOfNeighborCellHasWrongPole(neighbourObjectsWithDirection);

}

private boolean noneOfNeighborCellHasWrongPole(List<Pair<GameObject, Direction>> neighbourObjectsWithDirection) {

return neighbourObjectsWithDirection.stream()

.filter(pair -> pair.first instanceof MagneticObject).map(Pair::<MagneticObject>castFirst)

.allMatch(magneticObject -> isMagnitableTo(magneticObject.first, magneticObject.second));

}

@NotNull

private List<Pair<GameObject, Direction>> getNeighbourObjectsWithDirectionExceptOpposite(Cell cellToMove, @NotNull Direction direction) {

return cellToMove.neighbours().stream()

.filter(cellWithDirection -> cellWithDirection.direction != direction.opposite())

.flatMap(cellWithDirection -> cellWithDirection.cell.objects().stream()

.map(gameObject -> new Pair<>(gameObject, cellWithDirection.direction))

).collect(Collectors.toList());

}

@Override

public @NotNull ReadOnlyList<Pair<HookableObject, Direction>> hookedObjects() {

if (cell() == null)

return empty();

var hooked = getAllPossibleHooked();

return fromList(hooked);

}

@NotNull

private List<Pair<HookableObject, Direction>> getAllPossibleHooked() {

return cell().neighbours().stream()

.flatMap(cellWithDirection -> cellWithDirection.cell.objects().stream()

.filter(o -> o instanceof MagnitableObject)

.filter(o -> ((MagnitableObject) o).isMagnitableTo(this, cellWithDirection.direction.opposite()))

.map(o -> new Pair<>((HookableObject) o, cellWithDirection.direction)))

.collect(Collectors.toList());

}

@Override

protected boolean canReplaceIndependent(@NotNull GameObject gameObject, @NotNull Direction direction) {

var cellToMove = cell().neighborCell(direction);

var noneSolidInDirection = noneSolidInCellExcept(cellToMove, gameObject);;

var neighbourObjectsWithDirection =

getNeighborObjectsWithDirectionOnLateralSides(direction, cellToMove);

return noneSolidInDirection && noneOfNeighborCellHasWrongPole(neighbourObjectsWithDirection);

}

@NotNull

private List<Pair<GameObject, Direction>> getNeighborObjectsWithDirectionOnLateralSides(@NotNull Direction direction, Cell cellToMove) {

return cellToMove.neighbours().stream()

.filter(cellWithDirection -> cellWithDirection.direction != direction.opposite())

.filter(cellWithDirection -> cellWithDirection.direction != direction)

.flatMap(cellWithDirection -> cellWithDirection.cell.objects().stream()

.map(o -> new Pair<>(o, cellWithDirection.direction))

).collect(Collectors.toList());

}

public enum Alignment {

VERTICAL\_NORTH\_HORIZONTAL\_SOUTH,

VERTICAL\_SOUTH\_HORIZONTAL\_NORTH

}

}

Листинг 18. Реализация класса виджета простого объекта

package xyz.unterumarmung.view.widgets;

import org.jetbrains.annotations.NotNull;

import xyz.unterumarmung.view.providers.ImageProvider;

import java.awt.\*;

import java.awt.image.BufferedImage;

import static xyz.unterumarmung.view.utils.ImageUtils.resizeImage;

public class SimpleObjectWidget extends GameObjectWidget {

private static final int CELL\_DIMENSION\_DELTA = 0;

protected final @NotNull ImageProvider imageProvider;

public SimpleObjectWidget(@NotNull ImageProvider imageProvider) {

super();

this.imageProvider = imageProvider;

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

var resizeDimension = new Dimension(CellWidget.CELL\_DIMENSION.width - CELL\_DIMENSION\_DELTA, CellWidget.CELL\_DIMENSION.height - CELL\_DIMENSION\_DELTA);

var resizedImage = resizeImage(image(), resizeDimension);

g.drawImage(resizedImage, CELL\_DIMENSION\_DELTA / 2, CELL\_DIMENSION\_DELTA / 2, null);

}

protected @NotNull BufferedImage image() {

return imageProvider.image();

}

}

Листинг 19. Реализация виджета магнитного ящика

package xyz.unterumarmung.view.widgets;

import org.jetbrains.annotations.NotNull;

import xyz.unterumarmung.model.objects.MagneticBox.Alignment;

import xyz.unterumarmung.view.providers.ImageProvider;

import java.awt.image.BufferedImage;

import static xyz.unterumarmung.model.objects.MagneticBox.Alignment.VERTICAL\_NORTH\_HORIZONTAL\_SOUTH;

import static xyz.unterumarmung.view.utils.ImageUtils.rotateImageClockwise90;

public class MagneticBoxWidget extends SimpleObjectWidget {

private static final Alignment DEFAULT\_ALIGNMENT = VERTICAL\_NORTH\_HORIZONTAL\_SOUTH;

private final @NotNull BufferedImage image;

public MagneticBoxWidget(@NotNull ImageProvider imageProvider, @NotNull Alignment alignment) {

super(imageProvider);

if (alignment != DEFAULT\_ALIGNMENT) {

image = rotateImageClockwise90(imageProvider.image());

} else {

image = imageProvider.image();

}

}

@Override

protected @NotNull BufferedImage image() {

return image;

}

}

Листинг 20. Реализация класса окна выбор уровня

package xyz.unterumarmung.view;

import org.jetbrains.annotations.NotNull;

import xyz.unterumarmung.Level;

import xyz.unterumarmung.serialization.LevelLoader;

import xyz.unterumarmung.utils.collections.ReadOnlyList;

import xyz.unterumarmung.view.providers.ImageProvider;

import javax.swing.\*;

import static java.lang.Integer.parseInt;

import static javax.swing.JOptionPane.PLAIN\_MESSAGE;

import static javax.swing.JOptionPane.showInputDialog;

public class LevelChooser {

private final @NotNull LevelLoader levelLoader;

private final @NotNull ImageProvider iconProvider;

public LevelChooser(@NotNull final LevelLoader levelLoader, @NotNull final ImageProvider iconProvider) {

this.levelLoader = levelLoader;

this.iconProvider = iconProvider;

}

public Level levelByUser() {

final var levels = levelLoader.levels();

final var levelStrings = levelsToStrings(levels);

final var userReply = showInputDialog(

null,

"Выберете уровень для игры из списка ниже:",

"Выбор уровня",

PLAIN\_MESSAGE,

new ImageIcon(iconProvider.image()),

levelStrings,

null);

if (userReply == null)

return null;

final var levelId = extractIdFromString((String) userReply);

return findLevelById(levels, levelId);

}

private Object[] levelsToStrings(@NotNull ReadOnlyList<Level> levels) {

return levels.stream().map(level -> level.id + ". " + level.description).toArray();

}

private int extractIdFromString(@NotNull String levelString) {

final var dotIndex = levelString.indexOf('.');

final var idString = levelString.substring(0, dotIndex);

return parseInt(idString);

}

private Level findLevelById(@NotNull ReadOnlyList<Level> levels, int id) {

for (final var level : levels) {

if (level.id == id)

return level;

}

return null;

}

}

Листинг 21. Реализация класса игровой панели

package xyz.unterumarmung.view;

import org.jetbrains.annotations.NotNull;

import xyz.unterumarmung.events.MessageData;

import xyz.unterumarmung.events.MessageListener;

import xyz.unterumarmung.events.MessageSource;

import xyz.unterumarmung.events.SubscriptionHandler;

import xyz.unterumarmung.model.Game;

import xyz.unterumarmung.model.GameState;

import xyz.unterumarmung.model.events.GameMessage;

import xyz.unterumarmung.view.widgets.FieldWidget;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.event.WindowEvent;

import static javax.swing.JOptionPane.PLAIN\_MESSAGE;

import static javax.swing.JOptionPane.showMessageDialog;

import static xyz.unterumarmung.view.utils.DirectionUtils.isDirectionKeyCode;

import static xyz.unterumarmung.view.utils.DirectionUtils.keyCodeToDirection;

public class GamePanel extends JFrame implements MessageListener {

private final @NotNull Game game;

private final @NotNull FieldWidget fieldWidget;

public GamePanel(@NotNull final Game game, @NotNull final FieldWidget fieldWidget, @NotNull final SubscriptionHandler subscriptionHandler) throws HeadlessException {

this.game = game;

this.fieldWidget = fieldWidget;

subscriptionHandler.subscribeTo(game, this);

configureGui();

subscribeToKeyboardEvents();

}

private void configureGui() {

setVisible(true);

final var content = (JPanel) this.getContentPane();

content.add(this.fieldWidget);

pack();

setResizable(false);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setFocusable(true);

setTitle("Игра «Коза и капуста»");

}

private void subscribeToKeyboardEvents() {

addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

}

@Override

public void keyPressed(KeyEvent e) {

if (isDirectionKeyCode(e.getKeyCode()) && e.isControlDown()) {

game.gameField().goat().hookBox(keyCodeToDirection(e.getKeyCode()));

fieldWidget.repaint();

} else if (isDirectionKeyCode(e.getKeyCode())) {

game.gameField().goat().move(keyCodeToDirection(e.getKeyCode()));

fieldWidget.repaint();

} else if (e.getKeyCode() == KeyEvent.VK\_SPACE) {

game.gameField().goat().unhookBox();

fieldWidget.repaint();

}

handleState(game.gameState());

}

@Override

public void keyReleased(KeyEvent e) {

}

});

}

private void handleState(GameState gameState) {

final var MESSAGE\_TITLE = "Игра завершена!";

if (gameState == GameState.ENDED\_SUCCESS\_GOAT\_REACHED\_CABBAGE) {

showMessageDialog(this, "Коза успешно достигла капусты!", MESSAGE\_TITLE, PLAIN\_MESSAGE);

close();

} else if (gameState == GameState.ENDED\_FAILURE\_STEPS\_EXPIRED) {

showMessageDialog(this, "У козы закончились шаги!", MESSAGE\_TITLE, PLAIN\_MESSAGE);

close();

}

}

@Override

public void handleMessage(MessageSource source, MessageData data) {

final var gameMessage = (GameMessage) data;

handleState(gameMessage.gameState);

}

private void close() {

dispatchEvent(new WindowEvent(this, WindowEvent.WINDOW\_CLOSING));

}

}

### 4.7 Реализация ключевых тестовых случаев

Листинг 22. Реализация тестов класса двигающегося цепляющегося объекта

package xyz.unterumarmung.model.objects;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import xyz.unterumarmung.events.MessageSender;

import xyz.unterumarmung.model.GameField;

import xyz.unterumarmung.utils.Direction;

import xyz.unterumarmung.utils.Pair;

import xyz.unterumarmung.utils.Point;

import xyz.unterumarmung.utils.collections.ReadOnlyList;

import static org.junit.jupiter.api.Assertions.assertFalse;

import static org.mockito.Mockito.\*;

class MovableHookableShould {

private MovableHookable obj1;

private MovableHookable obj2;

private MovableHookable obj3;

private GameField gameField;

@BeforeEach

void beforeEach() {

gameField = new GameField(10, 10, new Point(0, 0), mock(MessageSender.class));

var cell1 = gameField.cell(new Point(2, 2));

var cell2 = gameField.cell(new Point(3, 2));

var cell3 = gameField.cell(new Point(4, 2));

obj1 = mock(MovableHookable.class, withSettings().useConstructor(cell1).defaultAnswer(CALLS\_REAL\_METHODS));

obj2 = mock(MovableHookable.class, withSettings().useConstructor(cell2).defaultAnswer(CALLS\_REAL\_METHODS));

obj3 = mock(MovableHookable.class, withSettings().useConstructor(cell3).defaultAnswer(CALLS\_REAL\_METHODS));

when(obj1.hookedObjects()).thenReturn(ReadOnlyList.of(new Pair<>(obj2, Direction.EAST)));

when(obj2.hookedObjects()).thenReturn(ReadOnlyList.of(new Pair<>(obj1, Direction.WEST), new Pair<>(obj3, Direction.EAST)));

when(obj3.hookedObjects()).thenReturn(ReadOnlyList.of(new Pair<>(obj2, Direction.WEST)));

for (var direction : Direction.all()) {

when(obj1.canMoveToIndependent(direction)).thenReturn(true);

when(obj2.canMoveToIndependent(direction)).thenReturn(true);

when(obj3.canMoveToIndependent(direction)).thenReturn(true);

}

when(obj1.canReplace(obj2, Direction.EAST)).thenReturn(true);

when(obj2.canReplace(obj1, Direction.WEST)).thenReturn(true);

when(obj2.canReplace(obj3, Direction.EAST)).thenReturn(true);

when(obj3.canReplace(obj2, Direction.WEST)).thenReturn(true);

}

@Test

void moveOnlyOnce() {

obj1.move(Direction.WEST);

verify(obj1, atMostOnce()).setCell(any());

verify(obj2, atMostOnce()).setCell(any());

verify(obj3, atMostOnce()).setCell(any());

}

@Test

void notMove\_whenOppositeCannotReplace() {

when(obj2.canReplace(obj1, Direction.WEST)).thenReturn(false);

var result = obj1.move(Direction.WEST);

assertFalse(result);

}

@Test

void notMove\_whenLeadingCant() {

when(obj1.canMoveToIndependent(Direction.WEST)).thenReturn(false);

var result = obj2.move(Direction.WEST);

assertFalse(result);

}

@Test

void notMove\_whenHasNoMovable() {

var obj4 = mock(HookableObject.class);

when(obj3.hookedObjects()).thenReturn(ReadOnlyList.of(new Pair<>(obj2, Direction.WEST), new Pair<>(obj4, Direction.SOUTH)));

var result = obj1.move(Direction.WEST);

assertFalse(result);

}

}

Листинг 23. Реализация тестов класса металлического ящика

package xyz.unterumarmung.model.objects;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import xyz.unterumarmung.events.MessageSender;

import xyz.unterumarmung.model.Cell;

import xyz.unterumarmung.model.GameField;

import xyz.unterumarmung.utils.Direction;

import xyz.unterumarmung.utils.Pair;

import xyz.unterumarmung.utils.Point;

import java.util.stream.Collectors;

import java.util.stream.Stream;

import static com.codepoetics.protonpack.StreamUtils.zip;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.when;

class MetalBoxShould {

private GameField gameField;

private SolidObject solidObject;

private GameObject notSolidGameObject;

private Cell initialCell;

@BeforeEach

void beforeEach() {

var messageSender = mock(MessageSender.class);

gameField = new GameField(5, 5, new Point(0, 0), messageSender);

solidObject = mock(SolidObject.class);

when(solidObject.isSolid()).thenReturn(true);

initialCell = gameField.cell(new Point(0, 0));

// Настройка поведения мок объекта, который ведёт себя как нетвёрдый объект

notSolidGameObject = mock(GameObject.class);

when(notSolidGameObject.isSolid()).thenReturn(false);

}

@Test

void beSolid() {

// Arrange

var box = new MetalBox(initialCell);

// Act & Assert

assertTrue(box.isSolid());

}

@Test

void beAbleToMove\_toEmptyCell() {

// Arrange

var box = new MetalBox(initialCell);

// Act && Assert

assertTrue(box.canMoveTo(Direction.EAST));

}

@Test

void beAbleToMove\_toCellWithNotSolidObject() {

// Arrange

var box = new MetalBox(initialCell);

gameField.cell(new Point(1, 0)).addObject(notSolidGameObject);

// Act & Assert

assertTrue(box.canMoveTo(Direction.EAST));

}

@Test

void notBeAbleToMove\_toCellWithSolidObject() {

// Arrange

var box = new MetalBox(initialCell);

gameField.cell(new Point(1, 0)).addObject(solidObject);

// Act & Assert

assertFalse(box.canMoveTo(Direction.EAST));

}

@Test

void addItselfToCellObjects() {

// Arrange

var box = new MetalBox(initialCell);

// Act & Assert

assertTrue(initialCell.objects().contains(box));

}

@Test

void beAbleToReplaceSolidObject() {

// Arrange

var box = new MetalBox(initialCell);

gameField.cell(new Point(1, 0)).addObject(solidObject);

// Act & Assert

assertTrue(box.canReplace(solidObject, Direction.EAST));

}

@Test

void beMagnitableAlways() {

// Arrange

var box = new MetalBox(initialCell);

// Act & Assert

assertTrue(box.isMagnitableTo(mock(MagneticObject.class), Direction.EAST));

assertTrue(box.isMagnitableTo(mock(MagneticObject.class), Direction.WEST));

assertTrue(box.isMagnitableTo(mock(MagneticObject.class), Direction.NORTH));

assertTrue(box.isMagnitableTo(mock(MagneticObject.class), Direction.SOUTH));

}

@Test

void hookToMagneticObjectsOnly() {

// Arrange

var cell = gameField.cell(new Point(2, 2));

var box = new MetalBox(cell);

var objects = Stream.of(mock(MagneticObject.class), mock(HookableObject.class),

mock(SolidObject.class), mock(MagneticObject.class));

var neighbourCells = cell.neighbours().stream().map(cellWithDirection -> cellWithDirection.cell);

var cellToObject = zip(neighbourCells, objects, Pair::new).collect(Collectors.toList());

for (var pair : cellToObject) {

pair.first.addObject(pair.second);

}

// Act

var hookedObjects = box.hookedObjects();

// Assert

assertTrue(hookedObjects.stream().map(pair -> pair.first).allMatch(object -> object instanceof MagneticObject));

assertEquals(2, hookedObjects.size());

}

@Test

void haveNoHooked\_whenNoCell() {

// Arrange

var box = new MetalBox(null);

// Act

var hookedObjects = box.hookedObjects();

// Assert

assertTrue(hookedObjects.isEmpty());

}

}

Листинг 24. Реализация тестов магнитного ящика

package xyz.unterumarmung.model.objects;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import xyz.unterumarmung.events.MessageSender;

import xyz.unterumarmung.model.Cell;

import xyz.unterumarmung.model.GameField;

import xyz.unterumarmung.utils.Direction;

import xyz.unterumarmung.utils.Point;

import java.util.EnumMap;

import java.util.HashMap;

import java.util.Map;

import java.util.stream.Collectors;

import static java.util.Collections.unmodifiableMap;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.\*;

import static xyz.unterumarmung.model.objects.MagneticBox.Alignment;

class MagneticBoxShould {

private GameField gameField;

private SolidObject solidObject;

private GameObject notSolidGameObject;

private Cell initialCell;

@BeforeEach

void beforeEach() {

var messageSender = mock(MessageSender.class);

gameField = new GameField(5, 5, new Point(0, 0), messageSender);

solidObject = mock(SolidObject.class);

when(solidObject.isSolid()).thenReturn(true);

initialCell = gameField.cell(new Point(0, 0));

// Настройка поведения мок объекта, который ведёт себя как нетвёрдый объект

notSolidGameObject = mock(GameObject.class);

when(notSolidGameObject.isSolid()).thenReturn(false);

}

@Test

void beSolid() {

// Arrange

var box = new MagneticBox(initialCell);

// Act & Assert

assertTrue(box.isSolid());

}

@Test

void beAbleToMove\_toEmptyCell() {

// Arrange

var box = new MagneticBox(initialCell);

// Act && Assert

assertTrue(box.canMoveTo(Direction.EAST));

}

@Test

void beAbleToMove\_toCellWithNotSolidObject() {

// Arrange

var box = new MagneticBox(initialCell);

gameField.cell(new Point(1, 0)).addObject(notSolidGameObject);

// Act & Assert

assertTrue(box.canMoveTo(Direction.EAST));

}

@Test

void notBeAbleToMove\_toCellWithSolidObject() {

// Arrange

var box = new MagneticBox(initialCell);

gameField.cell(new Point(1, 0)).addObject(solidObject);

// Act & Assert

assertFalse(box.canMoveTo(Direction.EAST));

}

@Test

void notBeAbleToMove\_whenHaveSamePoleInNeighbor() {

// Arrange

var cell = gameField.cell(new Point(2, 3));

var box = new MagneticBox(cell);

var moveToDirection = Direction.NORTH;

var moveToCell = cell.neighborCell(moveToDirection);

var moveToCellNeighbours = moveToCell.neighbours().stream()

.filter(cellWithDirection -> cellWithDirection.direction != moveToDirection.opposite())

.collect(Collectors.toList());

// Act & Assert

for (var neighbourCell : moveToCellNeighbours) {

var magnetic = mock(MagneticObject.class);

var poleDirection = neighbourCell.direction.opposite();

when(magnetic.magneticPoles()).thenReturn(Map.of(poleDirection, box.magneticPoles().get(poleDirection)));

neighbourCell.cell.addObject(magnetic);

assertFalse(box.canMoveTo(moveToDirection));

}

}

@Test

void notBeAbleToReplace\_whenHaveSamePoleInNeighbor() {

// Arrange

var cell = gameField.cell(new Point(2, 3));

var box = new MagneticBox(cell);

var moveToDirection = Direction.NORTH;

var moveToCell = cell.neighborCell(moveToDirection);

moveToCell.addObject(solidObject);

var moveToCellNeighbours = moveToCell.neighbours().stream()

.filter(cellWithDirection -> cellWithDirection.direction != moveToDirection.opposite())

.filter(cellWithDirection -> cellWithDirection.direction != moveToDirection)

.collect(Collectors.toList());

// Act & Assert

for (var neighbourCell : moveToCellNeighbours) {

var magnetic = mock(MagneticObject.class);

var poleDirection = neighbourCell.direction.opposite();

when(magnetic.magneticPoles()).thenReturn(Map.of(poleDirection, box.magneticPoles().get(poleDirection)));

neighbourCell.cell.addObject(magnetic);

assertFalse(box.canReplace(solidObject, moveToDirection));

}

}

@Test

void beMagnitable\_whenPolesMatch() {

var box = new MagneticBox(initialCell);

for (var direction : Direction.all()) {

var magneticObject = mock(MagneticObject.class);

var poleDirection = direction.opposite();

var pole = box.magneticPoles().get(poleDirection).opposite();

when(magneticObject.magneticPoles()).thenReturn(Map.of(poleDirection, pole));

assertTrue(box.isMagnitableTo(magneticObject, direction));

}

}

@Test

void notBeMagnitable\_whenPolesNotMatch() {

var box = new MagneticBox(initialCell);

for (var direction : Direction.all()) {

var magneticObject = mock(MagneticObject.class);

var poleDirection = direction.opposite();

var pole = box.magneticPoles().get(poleDirection);

when(magneticObject.magneticPoles()).thenReturn(Map.of(poleDirection, pole));

assertFalse(box.isMagnitableTo(magneticObject, direction));

}

for (var direction : Direction.all()) {

var magneticObject = mock(MagneticObject.class);

assertFalse(box.isMagnitableTo(magneticObject, direction));

}

for (var direction : Direction.all()) {

var magneticObject = mock(MagneticObject.class);

var poleDirection = direction.opposite();

var pole = MagneticPole.NORTH;

when(magneticObject.magneticPoles()).thenReturn(Map.of(poleDirection, pole));

var spyBox = spy(box);

when(spyBox.magneticPoles()).thenReturn(new HashMap<>());

assertFalse(spyBox.isMagnitableTo(magneticObject, direction));

}

}

@Test

void addItselfToCellObjects() {

// Arrange

var box = new MagneticBox(initialCell);

// Act & Assert

assertTrue(initialCell.objects().contains(box));

}

@Test

void beAbleToReplaceSolidObject() {

// Arrange

var box = new MagneticBox(initialCell);

gameField.cell(new Point(1, 0)).addObject(solidObject);

// Act & Assert

assertTrue(box.canReplace(solidObject, Direction.EAST));

}

@Test

void haveNoHooked\_whenNoCell() {

// Arrange

var box = new MagneticBox(null);

// Act

var hookedObjects = box.hookedObjects();

// Assert

assertTrue(hookedObjects.isEmpty());

}

@Test

void haveCorrectPoles\_forAlignment() {

var alignment1 = Alignment.VERTICAL\_NORTH\_HORIZONTAL\_SOUTH;

var alignment2 = Alignment.VERTICAL\_SOUTH\_HORIZONTAL\_NORTH;

var magneticBox1 = new MagneticBox(initialCell, alignment1);

var magneticBox2 = new MagneticBox(gameField.cell(new Point(3, 3)), alignment2);

assertEquals(mapForAlignment(alignment1), magneticBox1.magneticPoles());

assertEquals(mapForAlignment(alignment2), magneticBox2.magneticPoles());

}

private Map<Direction, MagneticPole> mapForAlignment(Alignment alignment) {

var map = new EnumMap<Direction, MagneticPole>(Direction.class);

switch (alignment) {

case VERTICAL\_NORTH\_HORIZONTAL\_SOUTH -> {

map.put(Direction.NORTH, MagneticPole.NORTH);

map.put(Direction.EAST, MagneticPole.SOUTH);

map.put(Direction.SOUTH, MagneticPole.NORTH);

map.put(Direction.WEST, MagneticPole.SOUTH);

}

case VERTICAL\_SOUTH\_HORIZONTAL\_NORTH -> {

map.put(Direction.NORTH, MagneticPole.SOUTH);

map.put(Direction.EAST, MagneticPole.NORTH);

map.put(Direction.SOUTH, MagneticPole.SOUTH);

map.put(Direction.WEST, MagneticPole.NORTH);

}

}

return unmodifiableMap(map);

}

}

Листинг 25. Реализация тестов простого ящика

package xyz.unterumarmung.model.objects;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import xyz.unterumarmung.events.MessageSender;

import xyz.unterumarmung.model.Cell;

import xyz.unterumarmung.model.GameField;

import xyz.unterumarmung.utils.Direction;

import xyz.unterumarmung.utils.Point;

import static org.junit.jupiter.api.Assertions.assertFalse;

import static org.junit.jupiter.api.Assertions.assertTrue;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.when;

class SimpleBoxShould {

private GameField gameField;

private SolidObject solidObject;

private GameObject notSolidGameObject;

private Cell initialCell;

@BeforeEach

void beforeEach() {

var messageSender = mock(MessageSender.class);

gameField = new GameField(5, 5, new Point(0, 0), messageSender);

solidObject = mock(SolidObject.class);

when(solidObject.isSolid()).thenReturn(true);

initialCell = gameField.cell(new Point(0, 0));

// Настройка поведения мок объекта, который ведёт себя как нетвёрдый объект

notSolidGameObject = mock(GameObject.class);

when(notSolidGameObject.isSolid()).thenReturn(false);

}

@Test

void haveNoHookedObjects() {

// Arrange

var simpleBox = new SimpleBox(initialCell);

// Act & Assert

assertTrue(simpleBox.hookedObjects().isEmpty());

}

@Test

void beSolid() {

// Arrange

var simpleBox = new SimpleBox(initialCell);

// Act & Assert

assertTrue(simpleBox.isSolid());

}

@Test

void beAbleToMove\_toEmptyCell() {

// Arrange

var box = new SimpleBox(initialCell);

// Act && Assert

assertTrue(box.canMoveTo(Direction.EAST));

}

@Test

void beAbleToMove\_toCellWithNotSolidObject() {

// Arrange

var box = new SimpleBox(initialCell);

gameField.cell(new Point(1, 0)).addObject(notSolidGameObject);

// Act & Assert

assertTrue(box.canMoveTo(Direction.EAST));

}

@Test

void notBeAbleToMove\_toCellWithSolidObject() {

// Arrange

var box = new SimpleBox(initialCell);

gameField.cell(new Point(1, 0)).addObject(solidObject);

// Act & Assert

assertFalse(box.canMoveTo(Direction.EAST));

}

@Test

void addItselfToCellObjects() {

// Arrange

var box = new SimpleBox(initialCell);

// Act & Assert

assertTrue(initialCell.objects().contains(box));

}

@Test

void beAbleToReplaceSolidObject() {

// Arrange

var box = new SimpleBox(initialCell);

gameField.cell(new Point(1, 0)).addObject(solidObject);

// Act & Assert

assertTrue(box.canReplace(solidObject, Direction.EAST));

}

}

Листинг 26. Реализация тестов козы

package xyz.unterumarmung.model.objects;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import xyz.unterumarmung.events.MessageSender;

import xyz.unterumarmung.model.Cell;

import xyz.unterumarmung.model.GameField;

import xyz.unterumarmung.model.events.GoatMessage;

import xyz.unterumarmung.model.exceptions.NoEnoughStepsException;

import xyz.unterumarmung.utils.Direction;

import xyz.unterumarmung.utils.Pair;

import xyz.unterumarmung.utils.Point;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.\*;

class GoatShould {

private Cell cell1;

private Cell cell2;

private Direction direction;

private GameObject solidGameObject;

private GameObject notSolidGameObject;

private Box box;

private GameField gameField5x5;

private MessageSender messageSender;

@BeforeEach

void beforeEach() {

// Создание клеток поля для имитации движения козы

messageSender = mock(MessageSender.class);

var gameField = new GameField(2, 1, new Point(1, 0), messageSender);

cell1 = gameField.cell(new Point(0, 0));

cell2 = gameField.cell(new Point(1, 0));

direction = Direction.EAST;

// Настройка поведения мок объекта, который ведёт себя как твёрдый объект

solidGameObject = mock(GameObject.class);

when(solidGameObject.isSolid()).thenReturn(true);

// Настройка поведения мок объекта, который ведёт себя как нетвёрдый объект

notSolidGameObject = mock(GameObject.class);

when(notSolidGameObject.isSolid()).thenReturn(false);

box = mock(Box.class);

gameField5x5 = new GameField(5, 5, new Point(4, 4), messageSender);

}

@Test

void haveEnoughSteps\_whenStepsHasLargerOrEqualCount() {

// Arrange

var goat1 = new Goat(Goat.STEP\_COST \* 4, cell1, messageSender);

var goat2 = new Goat(Goat.STEP\_COST + 1, cell1, messageSender);

var goat3 = new Goat(Goat.STEP\_COST, cell1, messageSender);

// Act & Assert

assertTrue(goat1.hasEnoughSteps());

assertTrue(goat2.hasEnoughSteps());

assertTrue(goat3.hasEnoughSteps());

}

@Test

void notHaveEnoughSteps\_whenStepCounterHasSmallerThanStepCost() {

// Arrange

var goat = new Goat(Goat.STEP\_COST - 1, cell1, messageSender);

// Act & Assert

assertFalse(goat.hasEnoughSteps());

}

@Test

void move\_toEmptyCell() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4, cell1, messageSender);

// Act

goat.move(direction);

// Assert

assertSame(cell2, goat.cell());

}

@Test

void move\_toCellWithNotSolidObject() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4, cell1, messageSender);

cell2.addObject(notSolidGameObject);

// Act

goat.move(direction);

// Assert

assertSame(cell2, goat.cell());

}

@Test

void notMove\_toCellWithSolidObject() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4, cell1, messageSender);

cell2.addObject(solidGameObject);

// Act

goat.move(direction);

// Assert

assertSame(cell1, goat.cell());

}

@Test

void addItselfToNewCell\_whenMoveSuccessful() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4, cell1, messageSender);

var objectsBeforeMove = cell2.objects();

// Act

goat.move(direction);

// Assert

assertFalse(objectsBeforeMove.contains(goat));

assertTrue(cell2.objects().contains(goat));

}

@Test

void removeItselfFromLastCell\_whenMoveSuccessful() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4, cell1, messageSender);

var objectsBeforeMove = cell1.objects();

// Act

goat.move(direction);

// Assert

assertTrue(objectsBeforeMove.contains(goat));

assertFalse(cell1.objects().contains(goat));

}

@Test

void notAddItselfToNewCell\_whenMoveNotSuccessful() {

// Arrange

var goat = spy(new Goat(Goat.STEP\_COST \* 4, cell1, messageSender));

// Намеренно запрещаем козе куда-либо передвигаться

when(goat.canMoveTo(direction)).thenReturn(false);

// Act

goat.move(direction);

// Assert

assertFalse(cell2.objects().contains(goat));

}

@Test

void notRemoveItselfFromLastCell\_whenMoveNotSuccessful() {

// Arrange

var goat = new Goat(Goat.STEP\_COST \* 4, cell1, messageSender);

var spyGoat = spy(goat);

// Намеренно запрещаем козе куда-либо передвигаться

when(spyGoat.canMoveTo(direction)).thenReturn(false);

// Act

spyGoat.move(direction);

// Assert

var objects = cell1.objects();

assertTrue(objects.contains(goat));

}

@Test

void beSolid() {

// Arrange & Act & Assert

assertTrue(new Goat(Goat.STEP\_COST \* 4, cell1, messageSender).isSolid());

}

@Test

void initializeWithGivenSteps() {

// Arrange

var steps = 15;

// Act

var goat = new Goat(steps, cell1, messageSender);

// Assert

assertEquals(steps, goat.steps());

}

@Test

void throw\_whenNotEnoughSteps() {

// Arrange

var initialSteps = 0;

var goat = new Goat(initialSteps, cell1, messageSender);

// Act & Assert

assertThrows(NoEnoughStepsException.class, goat::decreaseSteps);

}

@Test

void hookBox\_whenInDirection() {

// Arrange

var goat = new Goat(10, gameField5x5.cell(new Point(0, 0)), messageSender);

gameField5x5.cell(new Point(1, 0)).addObject(box);

// Act

var result = goat.hookBox(Direction.EAST);

// Assert

assertTrue(result);

assertTrue(goat.hookedObjects().stream().anyMatch(pair -> pair.equals(new Pair<>(box, Direction.EAST))));

}

@Test

void notHookBox\_whenNoInDirection() {

// Arrange

var goat = new Goat(10, gameField5x5.cell(new Point(0, 0)), messageSender);

// Act

var result = goat.hookBox(Direction.EAST);

// Assert

assertFalse(result);

assertTrue(goat.hookedObjects().isEmpty());

}

@Test

void sendMessage\_whenMoved() {

// Arrange

var goat = new Goat(10, gameField5x5.cell(new Point(0, 0)), messageSender);

// Act

goat.move(Direction.EAST);

var expectedMessage = new GoatMessage(gameField5x5.cell(new Point(0, 0)), gameField5x5.cell(new Point(1, 0)));

// Assert

verify(messageSender).emitMessage(eq(goat), eq(expectedMessage));

}

@Test

void notSendMessage\_whenNotMoved() {

// Arrange

var goat = new Goat(10, gameField5x5.cell(new Point(0, 0)), messageSender);

gameField5x5.cell(new Point(1, 0)).addObject(solidGameObject);

// Act

goat.move(Direction.EAST);

// Assert

verify(messageSender, never()).emitMessage(eq(goat), any());

}

@Test

void beAbleToReplaceSolidObject() {

// Arrange

var box = new Goat(10, gameField5x5.cell(new Point(0, 0)), messageSender);

gameField5x5.cell(new Point(1, 0)).addObject(solidGameObject);

// Act & Assert

assertTrue(box.canReplace(solidGameObject, Direction.EAST));

}

@Test

void unhookBox() {

// Arrange

var goat = new Goat(10, gameField5x5.cell(new Point(0, 0)), messageSender);

gameField5x5.cell(new Point(1, 0)).addObject(box);

// Act

goat.hookBox(Direction.EAST);

goat.unhookBox();

// Assert

assertTrue(goat.hookedObjects().isEmpty());

}

}

# 5 Список использованной литературы и других источников

1. Логинова, Ф.С. Объектно-ориентированные методы программирования. [Электронный ресурс] : учеб. пособие — Электрон. дан. — СПб. : ИЭО СПбУТУиЭ, 2012. — 208 с. — Режим доступа: http://e.lanbook.com/book/64040
2. Васильев, А.Н. Самоучитель Java с примерами и программами. [Электронный ресурс] : самоучитель — Электрон. дан. — СПб. : Наука и Техника, 2016. — 368 с. — Режим доступа: http://e.lanbook.com/book/90231
3. Программирование на языке Java. Конспект лекций. [Электронный ресурс] : учеб. пособие / А.В. Гаврилов [и др.]. — Электрон. дан. — СПб.: НИУ ИТМО, 2015. — 126 с. — Режим доступа: http://e.lanbook.com/book/91488

**Перечень замечаний к работе**