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 №01\_03)

Студент: Бондаренко Б. П.

Группа: ПрИн-367

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

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

Волгоград 2021 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: Бондаренко Б. П.

Группа: ПрИн-367

1. Тема: «Проектирование и реализация программы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №01\_03)

Утверждена приказом от «24» января 2021г. № 101-ст

2. Срок представления работы к защите « 04 »   июня  2021 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 «12» февраля 2021 г.

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

Задание принял к исполнению: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Бондаренко Б. П.

«12» февраля 2021 г.

**Содержание**

[1 Формулировка задания 4](#_Toc73354616)

[2 Нефункциональные требования 5](#_Toc73354617)

[3 Первая итерация разработки 6](#_Toc73354618)

[3.1 Формулировка упрощённого варианта задания 6](#_Toc73354619)

[3.2 Функциональные требования (сценарии) 7](#_Toc73354620)

[3.3 Словарь предметной области 10](#_Toc73354621)

[3.4 Структура программы на уровне классов 11](#_Toc73354622)

[3.5 Типовые процессы в программе 12](#_Toc73354623)

[3.6 Человеко-машинное взаимодействие 15](#_Toc73354624)

[3.7 Реализация ключевых классов 16](#_Toc73354625)

[4 Вторя итерация разработки 21](#_Toc73354626)

[4.1 Функциональные требования (сценарии) 21](#_Toc73354627)

[4.2 Словарь предметной области 24](#_Toc73354628)

[4.3 Структура программы на уровне классов 26](#_Toc73354629)

[4.4 Типовые процессы в программе 27](#_Toc73354630)

[4.5 Человеко-машинное взаимодействие 31](#_Toc73354631)

[4.6 Реализация ключевых классов 32](#_Toc73354632)

[5 Список использованной литературы и других источников 36](#_Toc73354633)

# 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.

Дополнительные требования:

* 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.

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 8 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 упрощённого варианта задания

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;
* количество шагов козы ограничено.

### 3.2 Функциональные требования (сценарии)

**1) Сценарий** «Играть»

1. **По указанию пользователя,** Игра стартует.
2. **По указанию** Игры, Поле создаёт Ячейки и формирует из них себя.
3. **По указанию** Контроллера, Лабиринт создаёт и размещает на Поле Стены, Ящики, Козу и Капусту.
4. **В ответ на запрос** Игры, Поле **сообщает** о Козе, которая находится на нём.
5. **Делать**
   1. **По указанию** **пользователя,** Коза перемещается на соседнюю Ячейку с заёмом своей энергии.
   2. **Если** Коза перемещается на Капусту, то Коза съедает Капусту, что приводит к исчезновению Капусты с Поля.
   3. **В ответ на запрос** Контроллера, Игра **сообщает** о сытости Козы.

**Пока** Коза имеет энергию и не съела капусту.

1. Игра считается выигранной, если Коза съела Капусту.
2. Игра считается проигранной, если у Козы была израсходована вся энергия, при этом Капуста не была ею съедена.
3. **Сценарий завершается**.

**2) Дочерний сценарий «**Лабиринт создаёт и размещает на Поле Стены, Ящики, Козу и Капусту**»**

1. Лабиринт создаёт и расставляет последовательности Стен внутри Поля.
2. Лабиринт создаёт и расставляет последовательности Ящиков внутри Поля.
3. Лабиринт создаёт Козу и наделяет её энергией.
4. Лабиринт помещает на Поле Козу.
5. Лабиринт создаёт Капусту и помещает её на Поле.
6. **Сценарий завершается**.

**3) Дочерний сценарий** «Коза перемещается на соседнюю Ячейку с заемом своей энергии»

1. **В ответ на запрос** Козы, текущая Ячейка сообщает о соседней по направлению движения Козы Ячейке.
2. **В ответ на запрос** Козы, соседняя Ячейка сообщает о возможном Ящике на ней.
3. **В ответ на запрос** Козы, Ящик по возможности двигается в направлении желаемого движения Козы.
4. **В ответ на запрос** Козы, соседняя Ячейка сообщает, что является свободной.
5. Коза решает, что она может переместиться в соседнюю Ячейку.
6. **По указанию** Козы, её собственная энергия расходуется на величину, необходимую для перемещения, т.к. у неё энергии достаточно.
7. **По указанию** Козы, Ячейка извлекает Козу из себя.
8. **По указанию** Козы, соседняя Ячейка помещает её в себя, т.к. она свободна.
9. **Сценарий завершается**.

**3.1) Альтернативный сценарий** «Соседняя по направлению движения Козы Ячейка не является свободной». (продолжение с п. 3)

1. **В ответ на запрос** Козы, соседняя Ячейка **сообщает**, что не является свободной.
2. Коза решает, что она не может переместиться в соседнюю Ячейку.
3. **Сценарий завершается**.

**4) Дочерний сценарий** «Коза съедает Капусту, что приводит к исчезновению Капусты с Поля».

1. **В ответ на запрос** Козы, Ячейка **удаляет** Капусту из себя.
2. Коза становится сытой.
3. **Сценарий завершается**.

**5) Дочерний сценарий** «Игра считается выигранной, если Коза достигла ячейку с Капустой»

1. Игра **запрашивает** у Козы, сыта ли она.
2. Коза **сообщает** Игре, что она сыта.
3. Игра считается выигранной.
4. **Сценарий завершается.**

**6) Дочерний сценарий** «Игра считается проигранной, если у Козы была израсходована вся энергия, при этом Капуста не была ею достигнута».

1. Игра **запрашивает** у Козы, сыта ли она.
2. Коза **сообщает** Игре, что она не сыта.
3. Игра считается проигранной.
4. **Сценарий завершается.**

**7) Альтернативный сценарий** «Досрочное завершение игры». Сценарий **выполняется в любой точке** главного сценария

1. **По указанию пользователя,** программа завершается без определения исхода игры.
2. **Сценарий завершается.**

### 3.3 Словарь предметной области

**Игра** - управляет игровым циклом: инициирует создание всех сущностей, определяет окончание и исход игры.

1. **Поле** - прямоугольная область, состоящая из Ячеек.
2. **Ячейка** - квадратная область Поля. Знает о четырёх соседних Ячейках. На ней может располагаться одна Стена, или один Ящик, или одна Коза, или одна Капуста.
3. **Стена** - непроходимое препятствие для Козы и Ящиков, располагающееся на Ячейке.
4. **Создатель лабиринта** - умеет создавать Стены, Ящики, Козу и Капусту и размещать их на Поле в Ячейках.
5. **Коза** - умеет однократно перемещаться в течение своего хода. Коза перемещается в соседнюю Ячейку, затрачивая собственную энергию, но не может пройти через Стену или неподвижный Ящик. Но Коза может толкать один Ящик. В течение своего хода Коза может съесть Капусту из Ячейки, в которой находится. Коза знает о том, съела ли она Капусту.
6. **Ящик** – может быть сдвинут Козой на одно из 4 направлений в случае, если соседняя Ячейка свободна.
7. **Капуста** - съедается Козой, находящейся на ней.
8. **Направление** - может быть Север, Юг, Запад или Восток.
9. **Позиция 2D** – содержит координаты X, Y, имеется у игрового объекта для определения его местоположения на поле.

### 3.4 Структура программы на уровне классов

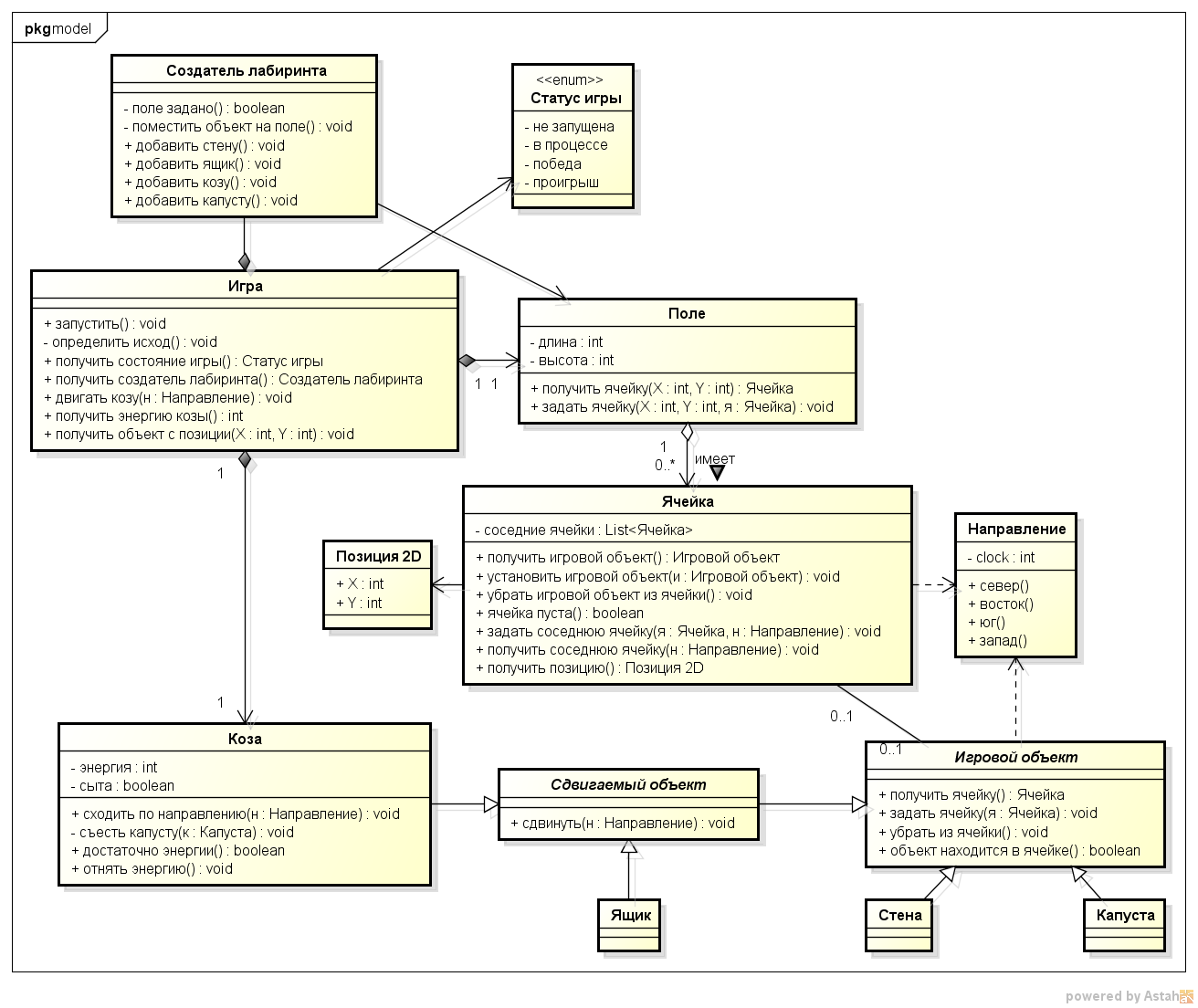


Рис. 1. Диаграмма классов вычислительной модели

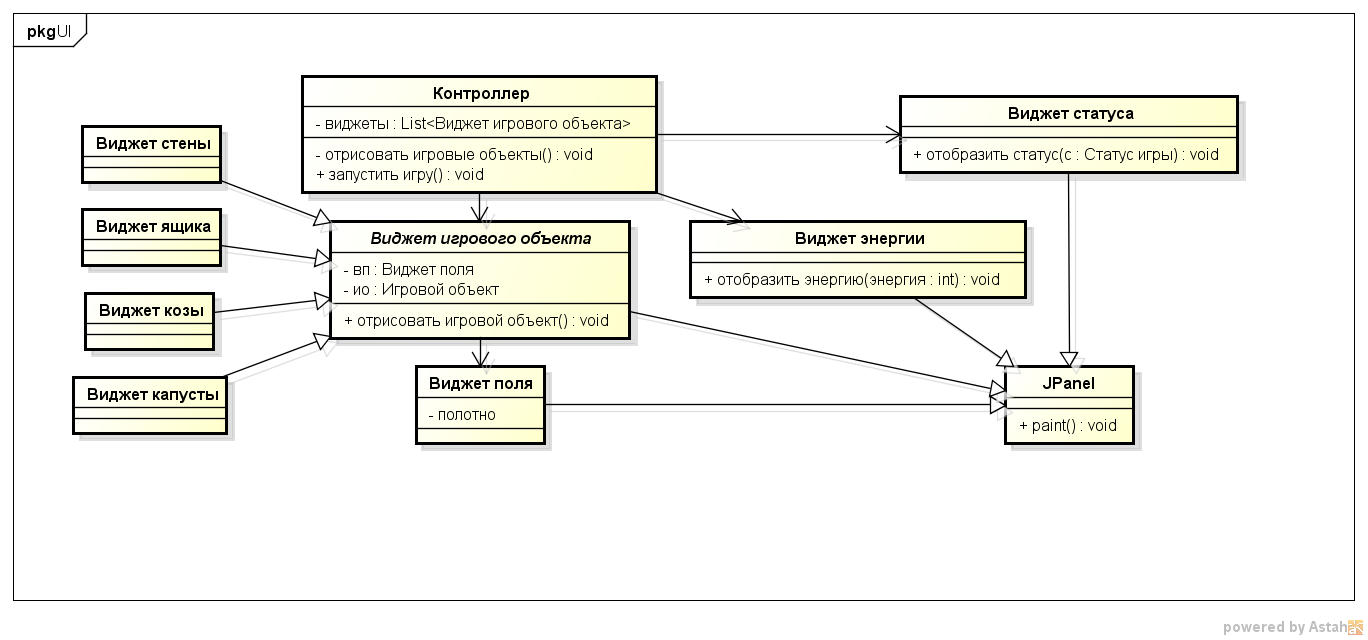


Рис. 2. Диаграмма классов представления

### 3.5 Типовые процессы в программе

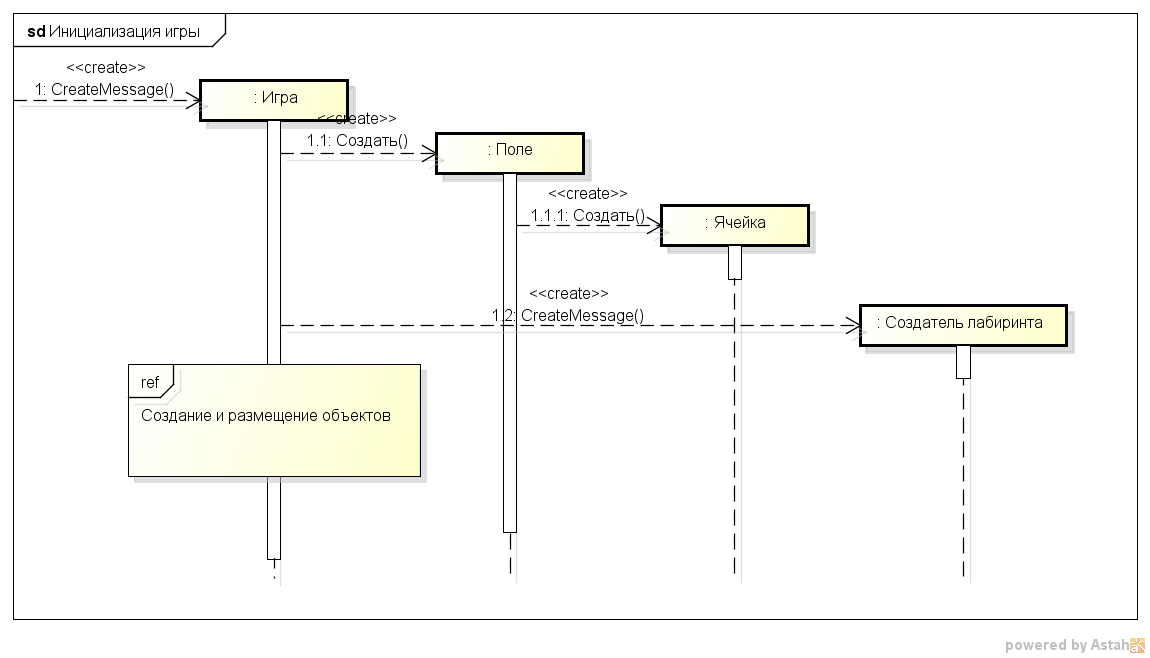


Рис. 3. Инициализация игры

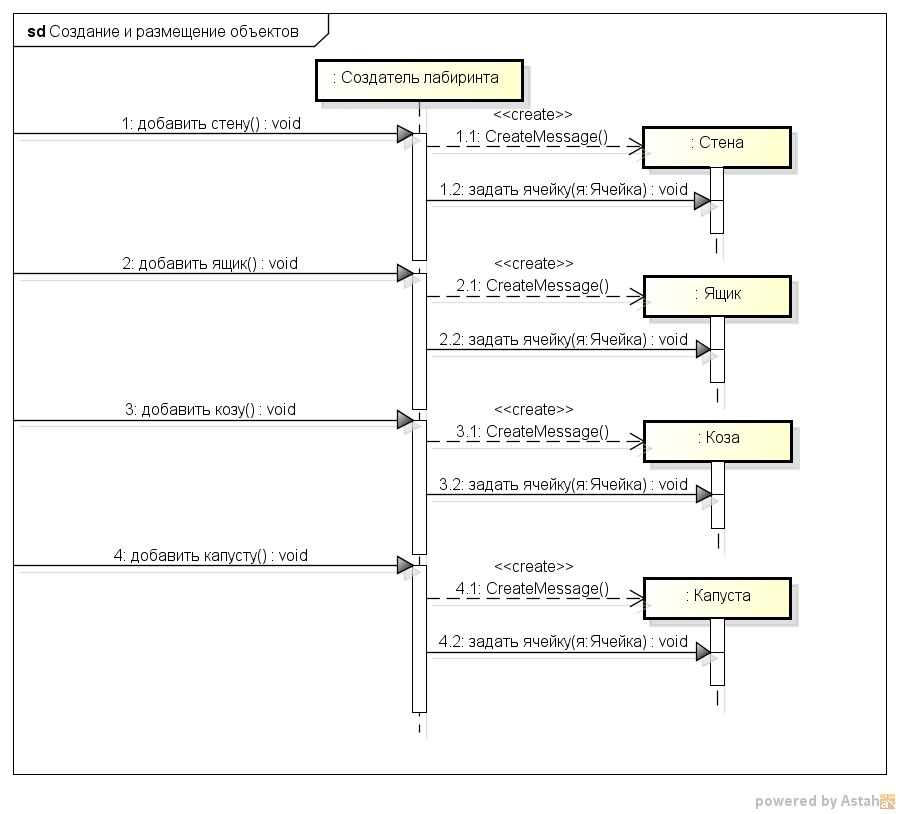


Рис. 4. Создание и размещение объектов

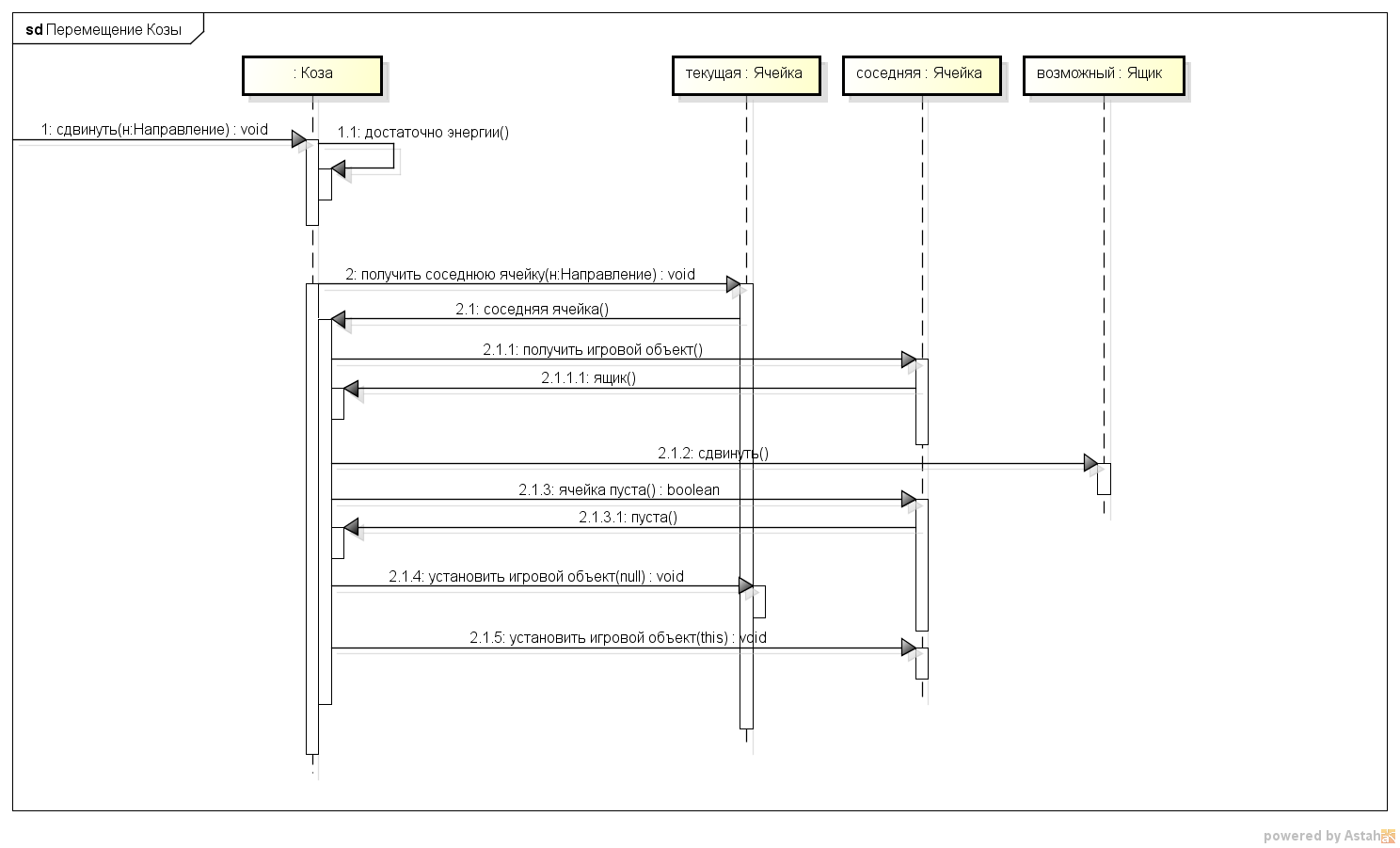


Рис. 5. Перемещение козы

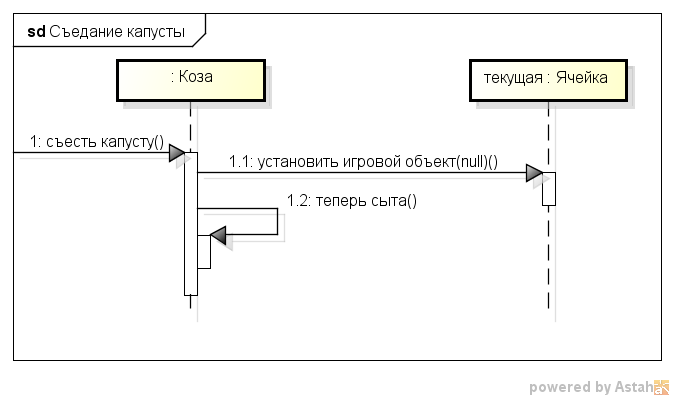


Рис. 6. Съедание капусты

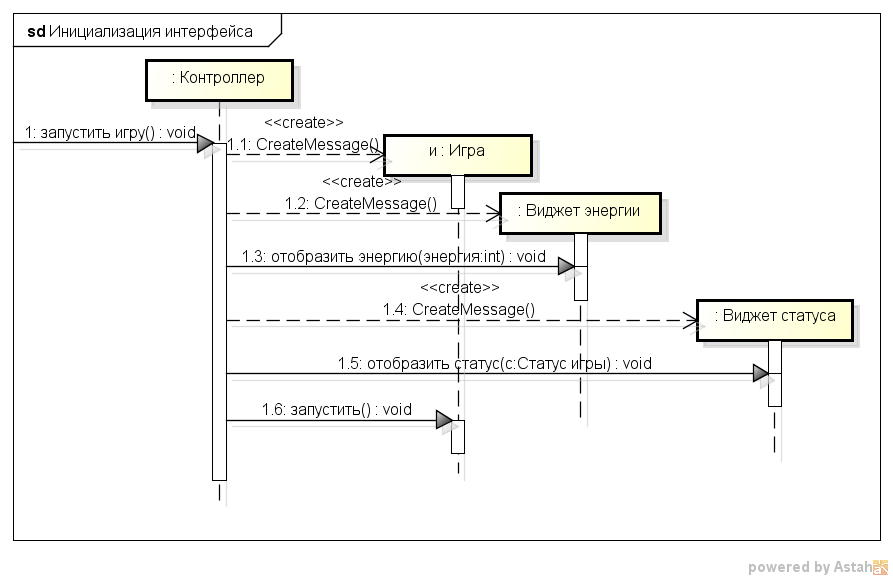


Рис. 7. Инициализация интерфейса

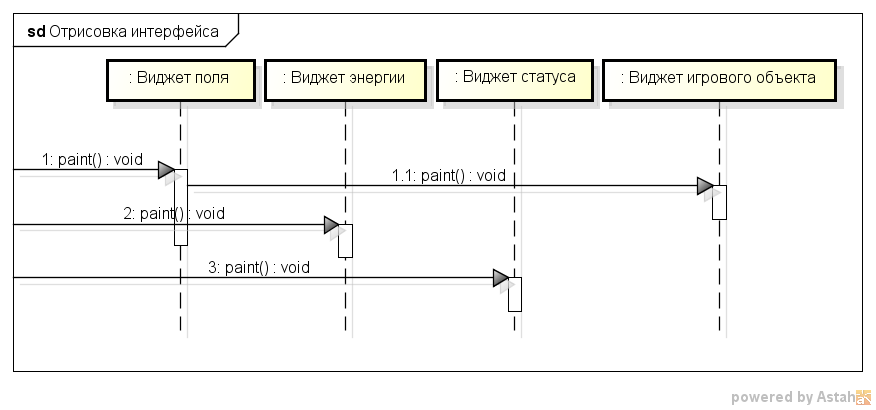


Рис. 8. Отрисовка интерфейса

### 3.6 Человеко-машинное взаимодействие

Общий вид главного экрана программы представлен ниже. На нём располагается игровое поле, на котором изображены следующие объекты: стена, ящик, коза и капуста. Энергия козы в начале игры равна 50.

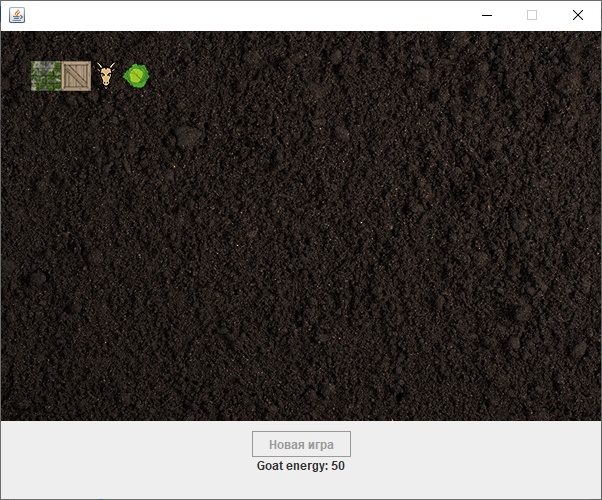


Рис. 9. Общий вид главного экрана программы

Управление козой пользователь осуществляет с помощью клавиатуры.

W – движение вверх.

S – движение вниз.

A – движение влево.

D – движение вправо.

Под виджетом поля располагается кнопка начала игры, затем виджет энергии козы, затем информация об исходе игры (выиграл игрок или проиграл).

### 3.7 Реализация ключевых классов

**Класс Game.java**

package Model;  
  
*// Игра*public class Game {  
  
 *// Инструмент создания лабиринта* public class MazeCreator {  
  
 private Game \_game; *//связанная игра* private Field \_field; *//поле для расставления объектов  
  
 //поле задано* private boolean fieldExists() {  
 return \_field != null;  
 }  
  
 *//поместить игровой объект на поле* private void addToField(GameObject go, int x, int y) {  
 if (fieldExists())  
 \_field.getCell(x, y).setGameObject(go);  
 else  
 throw new RuntimeException("Field is not specified!");  
 }  
  
 *//добавить стену на поле* public Wall addWall(int x, int y) {  
 Wall go = new Wall();  
 addToField(go, x, y);  
 return go;  
 }  
  
 *//добавить ящик на поле* public Box addBox(int x, int y) {  
 Box go = new Box();  
 addToField(go, x, y);  
 return go;  
 }  
  
 *//добавить козу на поле* public Goat addGoat(int x, int y, int energy) {  
 Goat go = new Goat(energy);  
 addToField(go, x, y);  
 \_game.\_goat = go;  
 return go;  
 }  
  
 *//добавить капусту на поле* public Cabbage addCabbage(int x, int y) {  
 Cabbage go = new Cabbage();  
 addToField(go, x, y);  
 return go;  
 }  
 }  
  
 *// Статус игры* public enum Status {  
 *DISABLED*, *//игра не запущена  
 PROCESS*, *//игра в процессе  
 WIN*, *//игра выиграна  
 LOSE //игра проиграна* }  
  
 private final MazeCreator \_mazeCreator; *//инструмент размещения объектов* private final Field \_field; *//игровое поле* private Goat \_goat; *//текущая коза* private Status \_status; *//статус игры* public Game() {  
 \_field = new Field(20, 13); *//создать поля* \_mazeCreator = new MazeCreator(); *//конструктор уровня* \_mazeCreator.\_game = this;  
 \_mazeCreator.\_field = \_field; *//привязка поля к конструктору* \_status = Status.*DISABLED*; *//игра пока не запущена* }  
  
 *//запустить игру* public void start() {  
 if (\_goat != null)  
 \_status = Status.*PROCESS*;  
 else  
 throw new RuntimeException("Goat doesn't exist");  
 }  
  
 *//определить исход игры* private void defineExodus() {  
 if (\_goat.gorged()) \_status = Game.Status.*WIN*;  
 else if (!\_goat.energyEnough()) \_status = Game.Status.*LOSE*;  
 }  
  
 *//получить состояние игры* public Game.Status getStatus() {  
 return \_status;  
 }  
  
 *//получить создатель лабиринта* public MazeCreator getMazeCreator() {  
 return \_mazeCreator;  
 }  
  
 *//двигать козу* public void moveGoat(Direction d) {  
 *//если игра в процессе* if (\_status == Status.*PROCESS*) {  
 \_goat.go(d); *//коза совершает ход* defineExodus(); *//определяется возможный исход* }  
 }  
  
 *//получить энергию козы* public int getGoatEnergy() {  
 return \_goat.energy();  
 }  
  
 *//получить объект с позиции* public GameObject getFromPosition(int x, int y) {  
 return \_field.getCell(x, y).getGameObject();  
 }  
}

**Класс Controller.java**

package UI;  
  
import Model.Game;  
import Model.Direction;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
import java.awt.event.KeyEvent;  
import java.awt.event.KeyListener;  
  
public class Controller extends JFrame {  
  
 private Game \_game;  
 private FieldWidget \_fw; *//виджет поля* private EnergyWidget \_enw; *//виджет энергии козы* private StatusWidget \_exw; *//виджет состояния игры* public Controller() {  
  
 JPanel content = (JPanel)getContentPane();  
 content.setLayout( new BoxLayout(content, BoxLayout.*Y\_AXIS*) );  
  
 createFieldWidget();  
  
 createGame();  
  
 content.add(Box.*createRigidArea*(new Dimension(0,10)));  
  
 createStartButton();  
 createEnergyWidget();  
 createStatusWidget();  
  
 content.add(Box.*createRigidArea*(new Dimension(0,10)));  
  
 pack(); *// подгоняем размеры окна под его содержимое* this.setResizable(false); *// в играх редко приходится изменять размер окна* \_fw.addKeyListener( new Controller.KeyController() );  
  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setVisible(true);  
 updateStatus();  
 }  
  
 *//создать игру* private void createGame() {  
  
 \_game = new Game();  
 *//создать игровые объекты и соответствующие виджеты* Game.MazeCreator creator = \_game.getMazeCreator();  
 \_fw.add(new WallWidget(creator.addWall(1, 1)));  
 \_fw.add(new BoxWidget(creator.addBox(2, 1)));  
 \_fw.add(new GoatWidget(creator.addGoat(3, 1, 50)));  
 \_fw.add(new CabbageWidget(creator.addCabbage(4, 1)));  
 }  
  
 *//создать виджет поля* private void createFieldWidget() {  
  
 \_fw = new FieldWidget();  
 add(\_fw);  
 }  
  
 *//создать кнопку запуска* private void createStartButton() {  
  
 JButton btnNewGame = new JButton();  
 btnNewGame.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 btnNewGame.setText("Новая игра");  
 btnNewGame.setFocusable(false);  
 add( btnNewGame );  
  
 btnNewGame.addActionListener(evt -> {  
 \_game.start();  
 updateStatus();  
 btnNewGame.setEnabled(false);  
 });  
 }  
  
 *//создать виджет энергии козы* private void createEnergyWidget() {  
  
 \_enw = new EnergyWidget();  
 \_enw.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 add(\_enw);  
 }  
  
 *//создать виджет состояния игры* private void createStatusWidget() {  
  
 \_exw = new StatusWidget();  
 \_exw.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 add(\_exw);  
 }  
  
 *//обновить статус игры* private void updateStatus() {  
 \_enw.setEnergy(\_game.getGoatEnergy());  
 \_exw.setStatus(\_game.getStatus());  
 }  
  
 */\*\*  
 \* Main method launching the application.  
 \*/* public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(Controller::new);  
 }  
  
 private class KeyController implements KeyListener {  
  
 @Override  
 public void keyTyped(KeyEvent e) {  
 }  
  
 @Override  
 public void keyPressed(KeyEvent e) {  
 int code = e.getKeyCode();  
  
 *// Управление козой* if(code == KeyEvent.VK\_UP) { *// перемещаемся вверх* \_game.moveGoat(Direction.north());  
 }  
 else if(code == KeyEvent.VK\_DOWN) { *// перемещаемся вниз* \_game.moveGoat(Direction.south());  
 }  
 else if(code == KeyEvent.VK\_LEFT) { *// перемещаемся влево* \_game.moveGoat(Direction.west());  
 }  
 else if(code == KeyEvent.VK\_RIGHT) { *// перемещаемся вправо* \_game.moveGoat(Direction.east());  
 }  
  
 updateStatus(); *// Обновляем статус* repaint(); *// Просим операционную систему перерисовать себя* }  
  
 @Override  
 public void keyReleased(KeyEvent e) {  
 }  
 }  
}

# 4 Вторя итерация разработки

### 4.1 Функциональные требования (сценарии)

**1) Сценарий** «Играть»

1. **По указанию пользователя,** Игра стартует.
2. **По указанию** Игры, Поле создаёт Ячейки и формирует из них себя.
3. **По указанию** Контроллера, Лабиринт создаёт и размещает на Поле Стены, Ящики, Козу и Капусту.
4. **В ответ на запрос** Игры, Поле **сообщает** о Козе, которая находится на нём.
5. **Делать**
   1. **По указанию** **пользователя,** Коза перемещается на соседнюю Ячейку с заёмом своей энергии.
   2. **Если** Коза перемещается на Капусту, то Коза съедает Капусту, что приводит к исчезновению Капусты с Поля.
   3. **В ответ на запрос** Контроллера, Игра **сообщает** о сытости Козы.

**Пока** Коза имеет энергию и не съела капусту.

1. Игра считается выигранной, если Коза съела Капусту.
2. Игра считается проигранной, если у Козы была израсходована вся энергия, при этом Капуста не была ею съедена.
3. **Сценарий завершается**.

**2) Дочерний сценарий «**Лабиринт создаёт и размещает на Поле Стены, Ящики, Козу и Капусту**»**

1. Лабиринт создаёт и расставляет последовательности Стен внутри Поля.
2. Лабиринт создаёт и расставляет последовательности Ящиков внутри Поля.
3. Лабиринт создаёт Козу и наделяет её энергией.
4. Лабиринт помещает на Поле Козу.
5. Лабиринт создаёт Капусту и помещает её на Поле.
6. **Сценарий завершается**.

**3) Дочерний сценарий** «Коза перемещается на соседнюю Ячейку с заемом своей энергии»

1. **В ответ на запрос** Козы, текущая Ячейка сообщает о соседней по направлению движения Козы Ячейке.
2. **В ответ на запрос** Козы, соседняя Ячейка сообщает о возможном Ящике на ней.
3. **В ответ на запрос** Козы, Ящик по возможности двигается в направлении желаемого движения Козы.
4. **В ответ на запрос** Козы, соседняя Ячейка сообщает, что является свободной.
5. Коза решает, что она может переместиться в соседнюю Ячейку.
6. **По указанию** Козы, соседний ящик притягивается к Козе и становится на её бывшую позицию.
7. **По указанию** Козы, её собственная энергия расходуется на величину, необходимую для перемещения, т.к. у неё энергии достаточно.
8. **По указанию** Козы, Ячейка извлекает Козу из себя.
9. **По указанию** Козы, соседняя Ячейка помещает её в себя, т.к. она свободна.
10. **Сценарий завершается**.

**3.1) Альтернативный сценарий** «Соседняя по направлению движения Козы Ячейка не является свободной». (продолжение с п. 3)

1. **В ответ на запрос** Козы, соседняя Ячейка **сообщает**, что не является свободной.
2. Коза решает, что она не может переместиться в соседнюю Ячейку.
3. **Сценарий завершается**.

**4) Дочерний сценарий** «Коза съедает Капусту, что приводит к исчезновению Капусты с Поля».

1. **В ответ на запрос** Козы, Ячейка **удаляет** Капусту из себя.
2. Коза становится сытой.
3. **Сценарий завершается**.

**5) Дочерний сценарий** «Игра считается выигранной, если Коза достигла ячейку с Капустой»

1. Игра **запрашивает** у Козы, сыта ли она.
2. Коза **сообщает** Игре, что она сыта.
3. Игра считается выигранной.
4. **Сценарий завершается.**

**6) Дочерний сценарий** «Игра считается проигранной, если у Козы была израсходована вся энергия, при этом Капуста не была ею достигнута».

1. Игра **запрашивает** у Козы, сыта ли она.
2. Коза **сообщает** Игре, что она не сыта.
3. Игра считается проигранной.
4. **Сценарий завершается.**

**7) Альтернативный сценарий** «Досрочное завершение игры». Сценарий **выполняется в любой точке** главного сценария

1. **По указанию пользователя,** программа завершается без определения исхода игры.
2. **Сценарий завершается.**

### 4.2 Словарь предметной области

**Игра** - управляет игровым циклом: инициирует создание всех сущностей, определяет окончание и исход игры.

1. **Поле** - прямоугольная область, состоящая из Ячеек.
2. **Ячейка** - квадратная область Поля. Знает о четырёх соседних Ячейках. На ней может располагаться одна Стена, или один Ящик любого вида, или одна Коза, или одна Капуста.
3. **Стена** - непроходимое препятствие для Козы и Ящиков, располагающееся на Ячейке.
4. **Создатель лабиринта** - умеет размещать игровые объекты на Поле в Ячейках.
5. **Коза** - умеет однократно перемещаться в течение своего хода. Коза перемещается в соседнюю Ячейку, затрачивая собственную энергию, но не может пройти через Стену или неподвижный Ящик. Но Коза может толкать либо тянуть один Ящик. В течение своего хода Коза может съесть Капусту из Ячейки, в которой находится. Коза знает о том, съела ли она Капусту.
6. **Ящик** – может быть сдвинут Козой на одно из 4 направлений в случае, если соседняя Ячейка свободна.

**Магнитный ящик** – имеет 4 полюса со всех сторон, осуществляет магнитные реакции с соседними Магнитными и Металлическими ящиками (отталкивает Ящик либо скрепляется с ним).

**Металлический ящик** – умеет осуществлять магнитные реакции с соседними Магнитными ящиками, всегда скрепляется с ними.

1. **Капуста** - съедается Козой, находящейся на ней.
2. **Направление** - может быть Север, Юг, Запад или Восток.
3. **Позиция 2D** – содержит координаты X, Y, имеется у игрового объекта для определения его местоположения на поле.

### 4.3 Структура программы на уровне классов

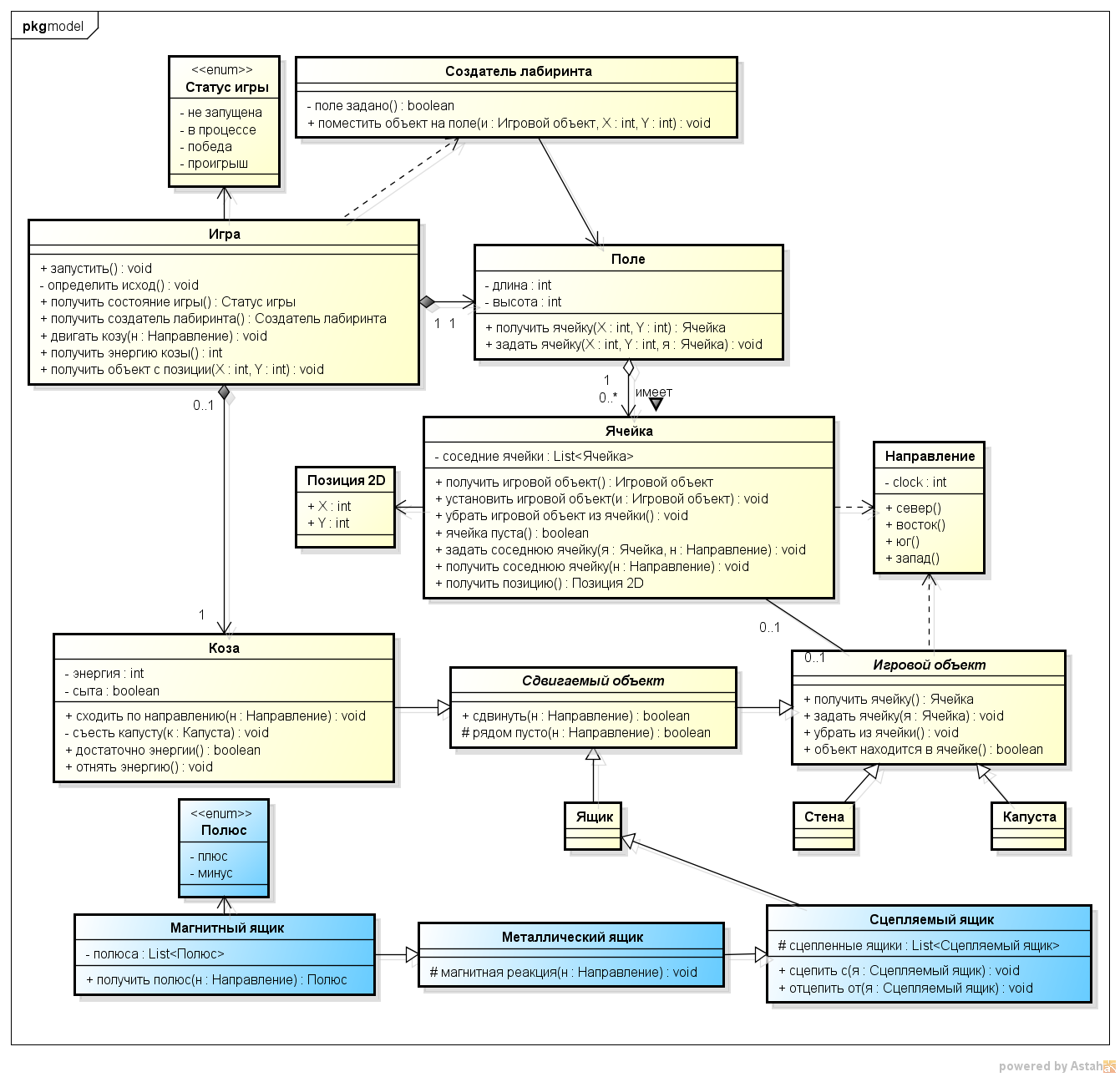


Рис. 10. Диаграмма классов вычислительной модели

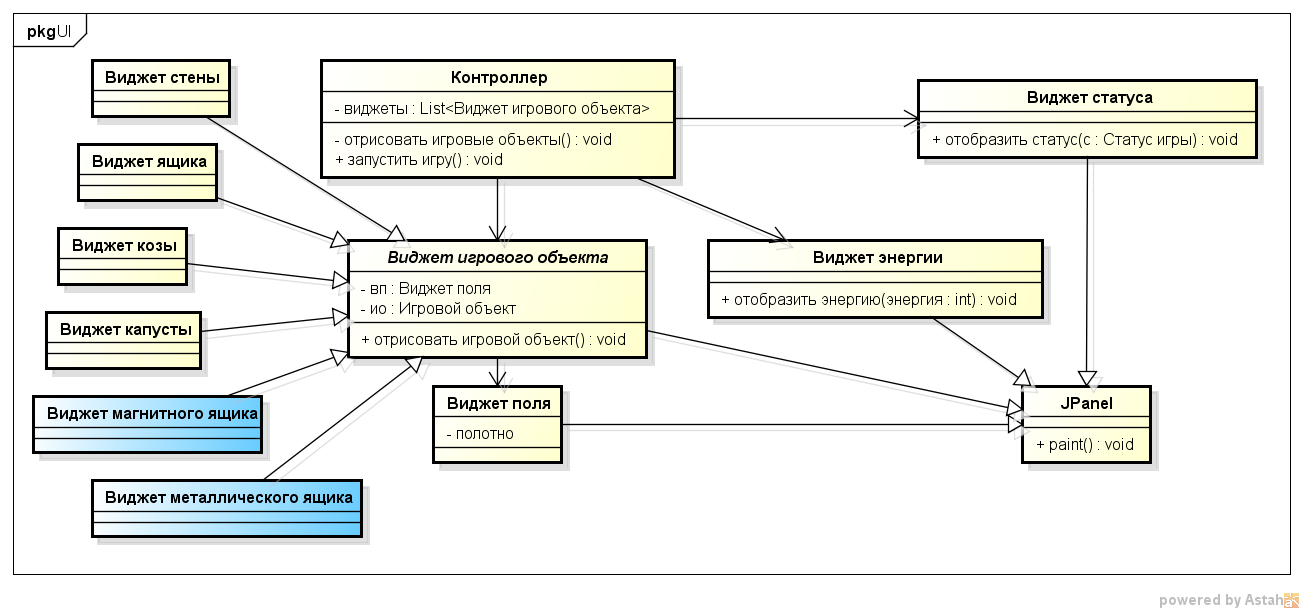


Рис. 11. Диаграмма классов представления

### 4.4 Типовые процессы в программе

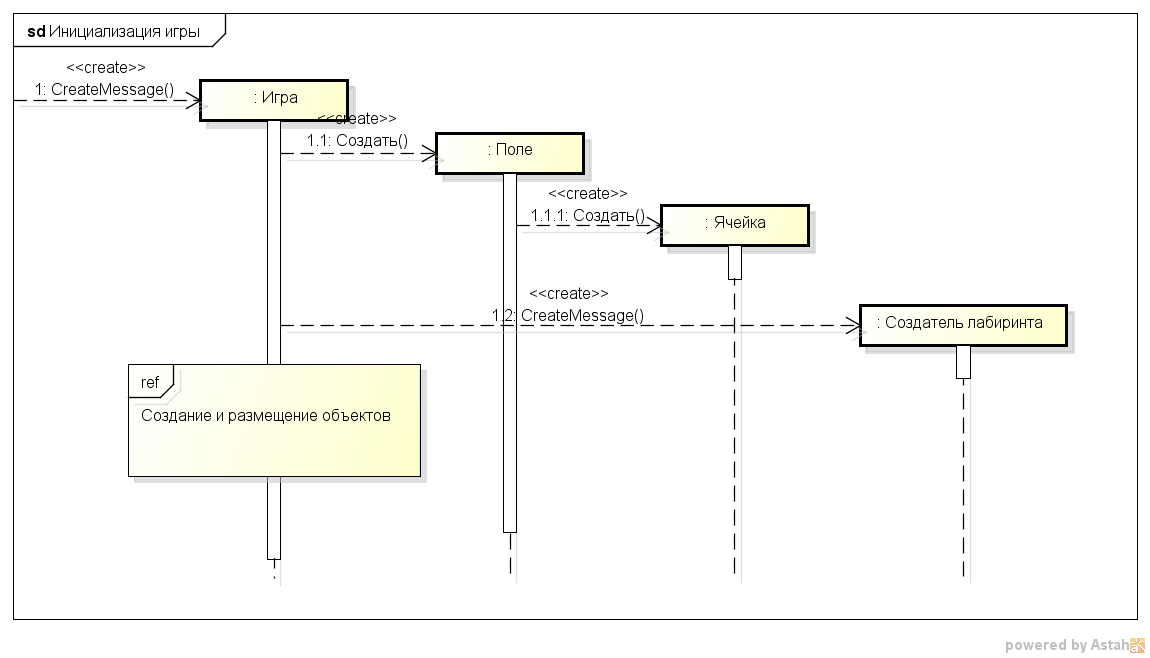


Рис. 12. Инициализация игры

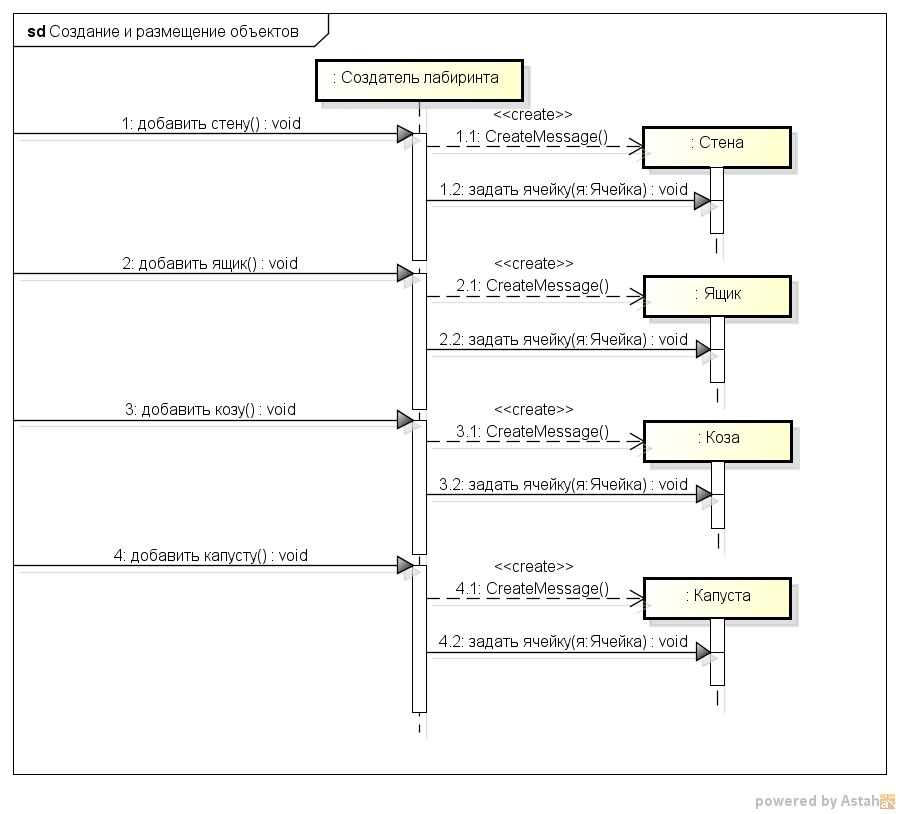


Рис. 13. Создание и размещение объектов

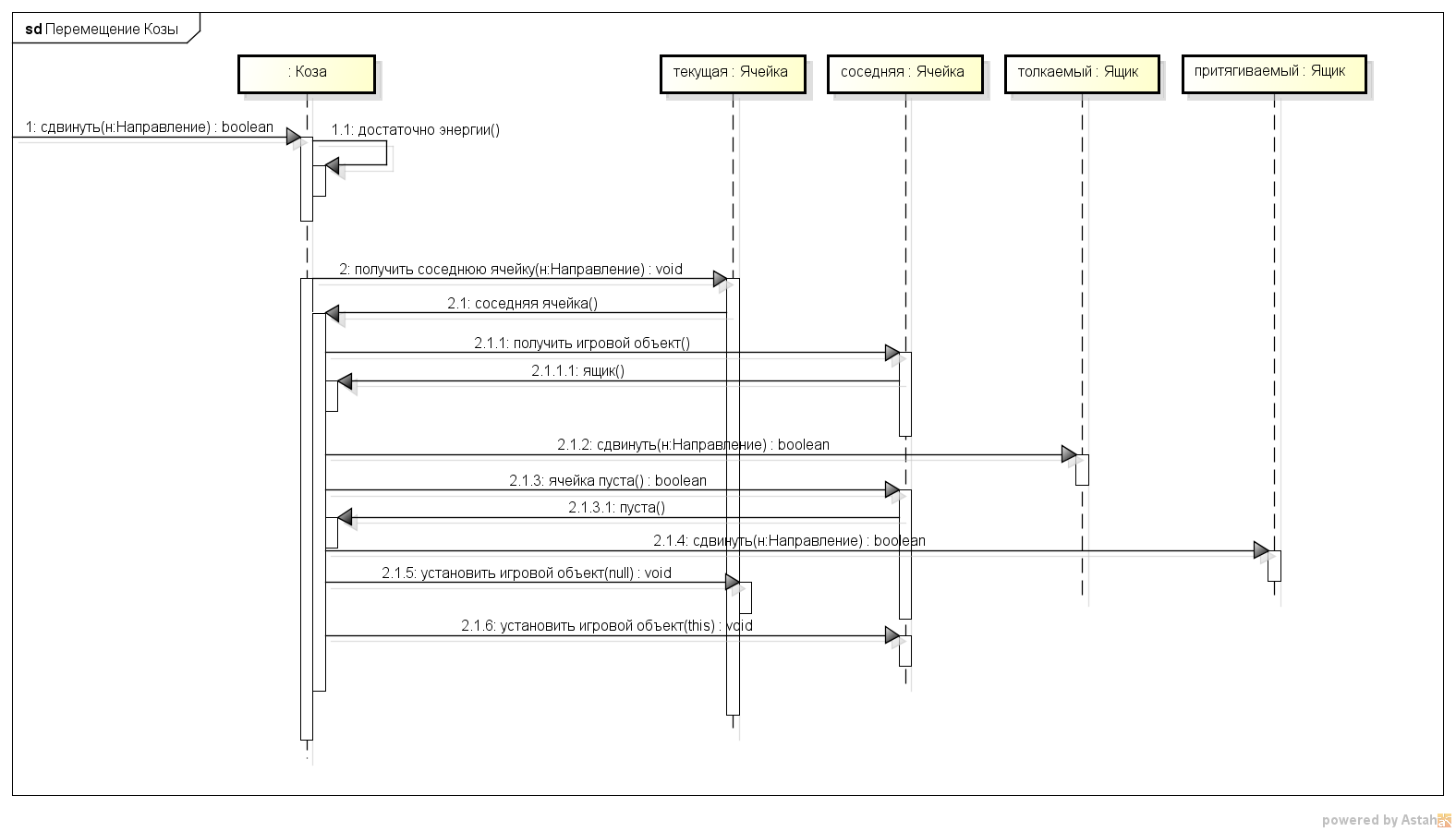


Рис. 14. Перемещение козы

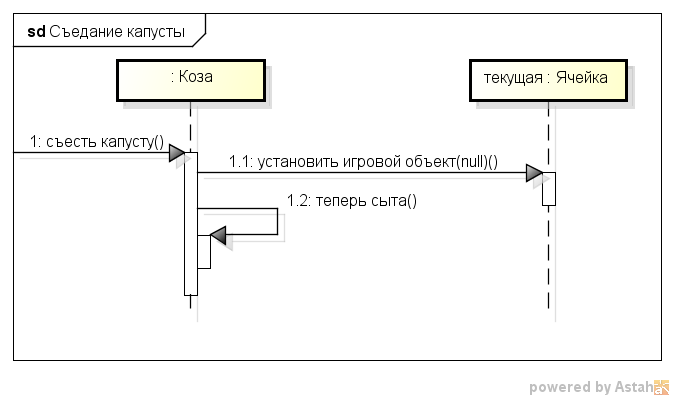


Рис. 15. Съедание капусты

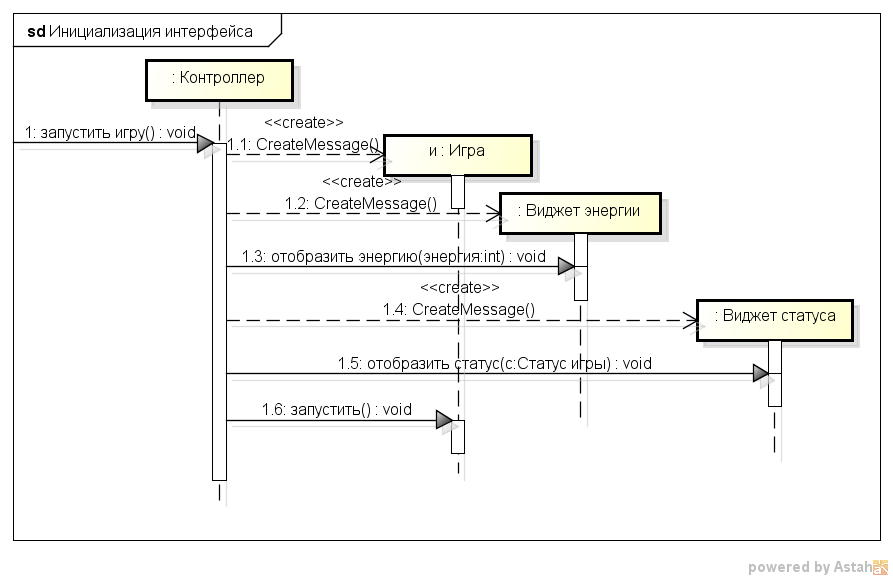


Рис. 16. Инициализация интерфейса

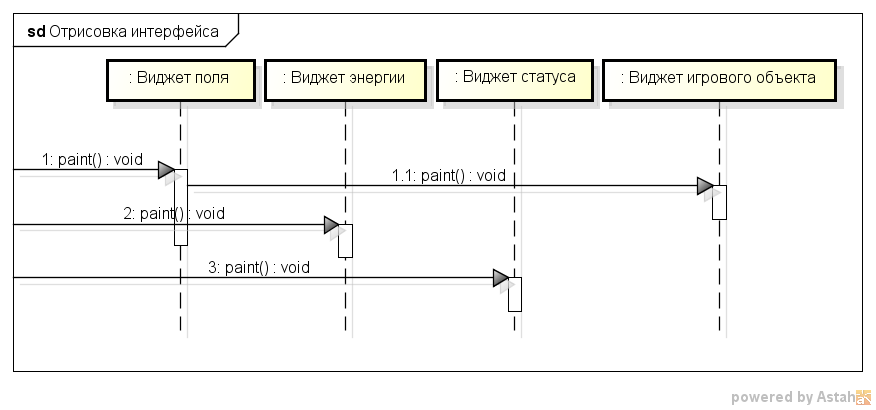


Рис. 17. Отрисовка интерфейса

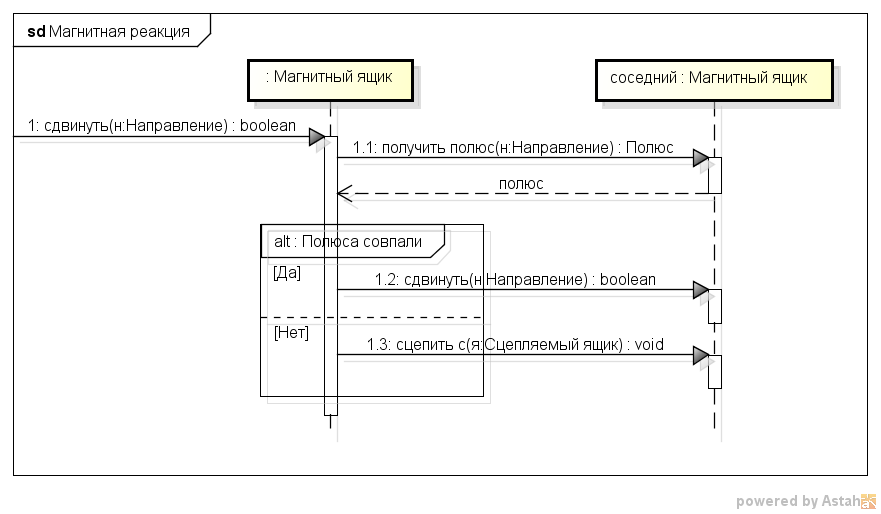


Рис. 18. Магнитная реакция

### 4.5 Человеко-машинное взаимодействие

Общий вид главного экрана программы представлен ниже. На нём располагается игровое поле, на котором изображены следующие объекты: стена, 2 деревянных ящика, коза, капуста, 4 магнитных ящика и один металлический ящик. Энергия козы в начале игры равна 150.

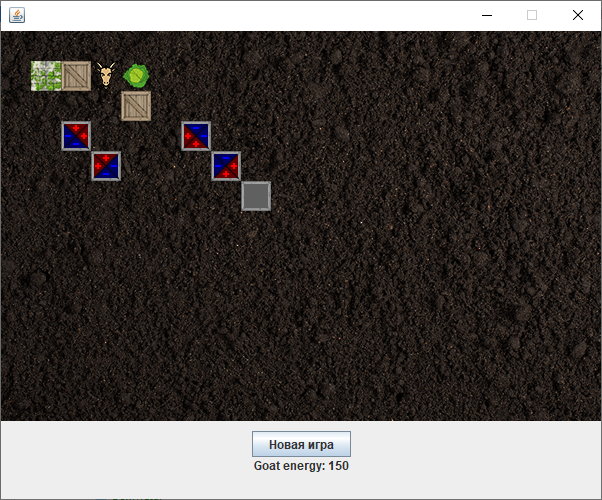


Рис. 19. Общий вид главного экрана программы

Управление козой пользователь осуществляет с помощью клавиатуры.

W – движение вверх.

S – движение вниз.

A – движение влево.

D – движение вправо.

Под виджетом поля располагается кнопка начала игры, затем виджет энергии козы, затем информация об исходе игры (выиграл игрок или проиграл).

У металлического ящика с каждой стороны подписан полюс, с помощью которых ящики контактируют между собой, если оказываются на соседних позициях.

### 4.6 Реализация ключевых классов

**Класс Game.java**

package Model;  
  
import java.util.Map;  
  
*// Игра*public class Game {  
  
 *// Инструмент создания лабиринта* public class MazeCreator {  
  
 private Game \_game; *//связанная игра* private Field \_field; *//поле для расставления объектов  
  
 //поле задано* private boolean fieldExists() {  
 return \_field != null;  
 }  
  
 *//поместить игровой объект на поле* public GameObject addToField(GameObject go, int x, int y) {  
  
 if (fieldExists()) { *//поле задано  
 //установить объект на поле* \_field.getCell(x, y).setGameObject(go);  
  
 *//захватить козу* if (go instanceof Goat)  
 \_game.\_goat = (Goat)go;  
  
 return go;  
 }  
 else *//поле не задано* throw new RuntimeException("Field is not specified!");  
 }  
 }  
  
 *// Статус игры* public enum Status {  
 *DISABLED*, *//игра не запущена  
 PROCESS*, *//игра в процессе  
 WIN*, *//игра выиграна  
 LOSE //игра проиграна* }  
  
 private final MazeCreator \_mazeCreator; *//инструмент размещения объектов* private final Field \_field; *//игровое поле* private Goat \_goat; *//текущая коза* private Status \_status; *//статус игры* public Game() {  
 \_field = new Field(20, 13); *//создать поля* \_mazeCreator = new MazeCreator(); *//конструктор уровня* \_mazeCreator.\_game = this;  
 \_mazeCreator.\_field = \_field; *//привязка поля к конструктору* \_status = Status.*DISABLED*; *//игра пока не запущена* }  
  
 *//запустить игру* public void start() {  
 if (\_goat != null)  
 \_status = Status.*PROCESS*;  
 else  
 throw new RuntimeException("Goat doesn't exist");  
 }  
  
 *//определить исход игры* private void defineExodus() {  
 if (\_goat.gorged()) \_status = Game.Status.*WIN*;  
 else if (!\_goat.energyEnough()) \_status = Game.Status.*LOSE*;  
 }  
  
 *//получить состояние игры* public Game.Status getStatus() {  
 return \_status;  
 }  
  
 *//получить создатель лабиринта* public MazeCreator getMazeCreator() {  
 return \_mazeCreator;  
 }  
  
 *//двигать козу* public void moveGoat(Direction d, boolean pull) {  
 *//если игра в процессе* if (\_status == Status.*PROCESS*) {  
 \_goat.go(d, pull); *//коза совершает ход* defineExodus(); *//определяется возможный исход* }  
 }  
  
 *//получить энергию козы* public int getGoatEnergy() {  
 return \_goat.energy();  
 }  
  
 *//получить объект с позиции* public GameObject getFromPosition(int x, int y) {  
 return \_field.getCell(x, y).getGameObject();  
 }  
}

**Класс Controller.java**

package UI;  
  
import Model.\*;  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
import java.awt.event.KeyEvent;  
import java.awt.event.KeyListener;  
import java.util.HashMap;  
  
public class Controller extends JFrame {  
  
 private Game \_game;  
 private FieldWidget \_fw; *//виджет поля* private EnergyWidget \_enw; *//виджет энергии козы* private StatusWidget \_exw; *//виджет состояния игры* public Controller() {  
  
 JPanel content = (JPanel)getContentPane();  
 content.setLayout( new BoxLayout(content, BoxLayout.*Y\_AXIS*) );  
  
 createFieldWidget();  
  
 createGame();  
  
 content.add(javax.swing.Box.*createRigidArea*(new Dimension(0,10)));  
  
 createStartButton();  
 createEnergyWidget();  
 createStatusWidget();  
  
 content.add(javax.swing.Box.*createRigidArea*(new Dimension(0,10)));  
  
 pack(); *// подгоняем размеры окна под его содержимое* this.setResizable(false); *// в играх редко приходится изменять размер окна* \_fw.addKeyListener( new Controller.KeyController() );  
  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setVisible(true);  
 updateStatus();  
 }  
  
 *//создать игру* private void createGame() {  
  
 \_game = new Game();  
 *//создать игровые объекты и соответствующие виджеты* Game.MazeCreator creator = \_game.getMazeCreator();  
  
 \_fw.add(new WallWidget((Wall)creator.addToField(new Wall(), 1, 1)));  
 \_fw.add(new BoxWidget((Model.Box)creator.addToField(new Model.Box(), 2, 1)));  
 \_fw.add(new GoatWidget((Goat)creator.addToField(new Goat(150), 3, 1)));  
 \_fw.add(new CabbageWidget((Cabbage)creator.addToField(new Cabbage(), 4, 1)));  
 \_fw.add(new BoxWidget((Model.Box)creator.addToField(new Model.Box(), 4, 2)));  
 \_fw.add(new BoxMagneticWidget((BoxMagnetic)creator.addToField(new BoxMagnetic(new HashMap<Direction, Pole>() {{  
 put(Direction.*north*(), Pole.*PLUS*);  
 put(Direction.*west*(), Pole.*MINUS*);  
 put(Direction.*south*(), Pole.*MINUS*);  
 put(Direction.*east*(), Pole.*PLUS*);  
 }}), 2, 3)));  
 \_fw.add(new BoxMagneticWidget((BoxMagnetic)creator.addToField(new BoxMagnetic(new HashMap<Direction, Pole>() {{  
 put(Direction.*north*(), Pole.*PLUS*);  
 put(Direction.*west*(), Pole.*PLUS*);  
 put(Direction.*south*(), Pole.*MINUS*);  
 put(Direction.*east*(), Pole.*MINUS*);  
 }}), 3, 4)));  
 \_fw.add(new BoxMagneticWidget((BoxMagnetic)creator.addToField(new BoxMagnetic(new HashMap<Direction, Pole>() {{  
 put(Direction.*north*(), Pole.*MINUS*);  
 put(Direction.*west*(), Pole.*PLUS*);  
 put(Direction.*south*(), Pole.*PLUS*);  
 put(Direction.*east*(), Pole.*MINUS*);  
 }}), 6, 3)));  
 \_fw.add(new BoxMagneticWidget((BoxMagnetic)creator.addToField(new BoxMagnetic(new HashMap<Direction, Pole>() {{  
 put(Direction.*north*(), Pole.*MINUS*);  
 put(Direction.*west*(), Pole.*MINUS*);  
 put(Direction.*south*(), Pole.*PLUS*);  
 put(Direction.*east*(), Pole.*PLUS*);  
 }}), 7, 4)));  
 \_fw.add(new BoxMetalWidget((BoxMetal)creator.addToField(new BoxMetal(), 8, 5)));  
 }  
  
 *//создать виджет поля* private void createFieldWidget() {  
  
 \_fw = new FieldWidget();  
 add(\_fw);  
 }  
  
 *//создать кнопку запуска* private void createStartButton() {  
  
 JButton btnNewGame = new JButton();  
 btnNewGame.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 btnNewGame.setText("Новая игра");  
 btnNewGame.setFocusable(false);  
 add( btnNewGame );  
  
 btnNewGame.addActionListener(evt -> {  
 \_game.start();  
 updateStatus();  
 btnNewGame.setEnabled(false);  
 });  
 }  
  
 *//создать виджет энергии козы* private void createEnergyWidget() {  
  
 \_enw = new EnergyWidget();  
 \_enw.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 add(\_enw);  
 }  
  
 *//создать виджет состояния игры* private void createStatusWidget() {  
  
 \_exw = new StatusWidget();  
 \_exw.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 add(\_exw);  
 }  
  
 *//обновить статус игры* private void updateStatus() {  
 \_enw.setEnergy(\_game.getGoatEnergy());  
 \_exw.setStatus(\_game.getStatus());  
 }  
  
 */\*\*  
 \* Main method launching the application.  
 \*/* public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(Controller::new);  
 }  
  
 private class KeyController implements KeyListener {  
  
 boolean shiftPressed = false; *//шифт зажат* @Override  
 public void keyTyped(KeyEvent e) {  
 }  
  
 @Override  
 public void keyPressed(KeyEvent e) {  
 int code = e.getKeyCode();  
  
 *// Управление козой* if(code == KeyEvent.*VK\_UP*) { *// перемещаемся вверх* \_game.moveGoat(Direction.*north*(), shiftPressed);  
 }  
 else if(code == KeyEvent.*VK\_DOWN*) { *// перемещаемся вниз* \_game.moveGoat(Direction.*south*(), shiftPressed);  
 }  
 else if(code == KeyEvent.*VK\_LEFT*) { *// перемещаемся влево* \_game.moveGoat(Direction.*west*(), shiftPressed);  
 }  
 else if(code == KeyEvent.*VK\_RIGHT*) { *// перемещаемся вправо* \_game.moveGoat(Direction.*east*(), shiftPressed);  
 }  
 else if (code == KeyEvent.*VK\_SHIFT*) { *//включить шифт* shiftPressed = true;  
 }  
  
 updateStatus(); *// Обновляем статус* repaint(); *// Просим операционную систему перерисовать себя* }  
  
 @Override  
 public void keyReleased(KeyEvent e) {  
 if (e.getKeyCode() == KeyEvent.*VK\_SHIFT*) { *//отключить шифт* shiftPressed = false;  
 }  
 }  
 }  
}

# 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**