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 программы с использованием объектно-ориентированного подхода»

(индивидуальное задание – вариант №07\_02)

Студент: Воробьев А.С.

Группа: ПрИн-366

Работа зачтена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ « 11  »     июня   2022 г.

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

Волгоград 2022 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: Воробьев А.С.

Группа: ПрИн-366

1. Тема: «Проектированиепрограммы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №07)

Утверждена приказом от «24» января 2022г. № 101-ст

2. Срок представления работы к защите « 04 »   июня  2022 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 2022 г.

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

Задание принял к исполнению: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Воробьев А.С.

«13» февраля 2022 г.

**Содержание**

Оглавление

[Формулировка задания 4](#_Toc104409128)

[Нефункциональные требования 5](#_Toc104409129)

[Первая итерация разработки 6](#_Toc104409130)

[3.1 Формулировка упрощенного варианта задания 6](#_Toc104409131)

[3.2 Функциональные требования (сценарии) 6](#_Toc104409132)

[3.3 Словарь предметной области 10](#_Toc104409133)

[3.4 Структура программы на уровне классов 13](#_Toc104409134)

[3.5 Типовые процессы в программе 15](#_Toc104409135)

[3.6 Человеко-машинное взаимодействие 31](#_Toc104409136)

[3.7 Реализация ключевых классов 35](#_Toc104409137)

[3.8 Реализация ключевых тестовых случаев 67](#_Toc104409138)

[Вторая итерация разработки 69](#_Toc104409139)

[4.1 Функциональные требования (сценарии) 69](#_Toc104409140)

[4.2 Словарь предметной области 75](#_Toc104409141)

[4.3 Структура программы на уровне классов 78](#_Toc104409142)

[4.4 Типовые процессы в программе 80](#_Toc104409143)

[4.5 Человеко-машинное взаимодействие 86](#_Toc104409144)

[4.6 Реализация ключевых классов 88](#_Toc104409145)

[4.7 Реализация ключевых тестовых сценариев 109](#_Toc104409146)

[Список использованной литературы и других источников 119](#_Toc104409147)

# Формулировка задания

Правила игры «Глупый робот и умный робот»:

* имеется лабиринт, в котором действует маленький умный робот и большой глупый робот, их позиции задаются;
* маленьким роботом управляет человек, а большим - компьютер;
* игра ведется пошагово;
* в лабиринте имеются болота, а также вертикальные и горизонтальные препятствия (на стыке клеток);
* цель маленького робота – достичь выхода, цель большого робота – поймать маленького робота;
* большой робот рационально действует только, когда маленький робот попадает в поле его действия размером 2×2, в остальных случаях робот идет либо вправо, либо влево по направлению к маленькому роботу;
* маленькому роботу заходить в болото нельзя, а большой робот пропускает три хода.

Дополнительные требования:

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

Вариативность:

* предсоздание других ландшафтов, влияющих на перемещение обоих роботов (количество пройденных шагов и направление движения) и зависящих от сезона года. Ландшафты в разных состояниях должны визуально различаться.

Реализовать:

* болото, которое расширяется на новые клетки в период осени и весны (по окончанию дождливых периодов болото возвращается к исходным размерам), и замерзает в зимний период. По замерзшему болоту роботы передвигаются сразу на две клетки до препятствия, т.е. скользят.
* песок, который заставляет "буксовать" роботов, т.е. пропускать ход.

# Нефункциональные требования

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

# Первая итерация разработки

## 3.1 Формулировка упрощенного варианта задания

Правила игры «Глупый робот и умный робот»:

* имеется лабиринт, в котором действует маленький умный робот и большой глупый робот, их позиции задаются случайным образом;
* маленьким роботом управляет человек, а большим - компьютер;
* игра ведется пошагово;
* в лабиринте имеются болота, а также вертикальные и горизонтальные препятствия (на стыке клеток);
* цель маленького робота – достичь выхода, цель большого робота – поймать маленького робота;
* большой робот рационально действует только, когда маленький робот попадает в поле его действия размером 2×2, в остальных случаях робот идет либо вправо, либо влево по направлению к маленькому роботу;
* маленькому роботу заходить в болото нельзя, а большой робот пропускает три хода.

## 3.2 Функциональные требования (сценарии)

**1) Сценарий «Играть за маленького робота: выйти из лабиринта, не попавшись большому роботу»**

1. Пользователь инициирует начало игры.
2. Игра создаёт поле из ячеек и размещает на нем лабиринт из стен, болото(а), точку выхода и двух роботов: большого и маленького.
3. Игра делает активным маленького робота
4. Делать
   1. Робот перемещается на соседнюю ячейку.
   2. Игра делает активным следующего робота, который располагается на поле.

Пока маленький робот находится внутри поля.

1. Игра считает победителем игрока, если он “телепортирован” точкой выхода или компьютер, если маленький робот был уничтожен.

**2) Сценарий «игра создаёт поле из ячеек и размещает на нем лабиринт из стен, болото(а), точку выхода и двух роботов»**

1. Игра инициирует создание поля размером W на H ячеек.
2. Игра посредством лабиринта создает и расставляет последовательности стен по периметру поля.
3. Игра посредством лабиринта создает последовательности стен в середине поля.
4. Игра посредством лабиринта создает болото(а) случайным образом случайной формы.
5. Игра создаёт двух роботов.
6. Игра помещает на поле созданных роботов.
7. Игра создает и помещает на поле ячейку выхода.

**4) Сценарий «робот перемещается на соседнюю ячейку».**

1. Робот запрашивает у ячейки, в которой он находится, стену в направлении своего движения
2. Ячейка сообщает, что стены нет
3. Робот запрашивает у ячейки, в которой он находится, соседнюю ячейку в направлении своего движения
4. Ячейка сообщает о ячейке, с которой соседствует
5. Робот запрашивает у соседней ячейки робота (который возможно в ней находится)
6. Соседняя ячейка сообщает, что робот отсутствует в ней
7. Робот просит ячейку, в которой он находится, изъять его из нее
8. Ячейка извлекает робота из себя
9. Робот просит соседнюю ячейку поместить себя в нее
10. Ячейка помещает робота в себя, т.к. в ней нет другого робота
11. Робот определяет, находится ли в новой ячейке болото

**4.1) Альтернативный сценарий «ход большого робота»**

1. Сценарий начинается перед п. 1 Главного сценария 4
2. Большой робот вычисляет расстояние до маленького робота
3. Так как маленький робот находится в поле действия большого робота, большой робот составляет оптимальный маршрут до маленького робота
4. Большой робот выбирает направление движения по направлению к следующей клетке в построенном оптимальном пути
5. Сценарий переходит к п. 1 сценария 4

**4.1.1) Альтернативный сценарий «ход большого робота, пропуск хода»**

1. Сценарий начинается перед п. 1 альтернативного сценария «ход большого робота»
2. Так как робот все еще «оштрафован» за попадание в ячейку с болотом, он пропускает ход
3. Сценарий переходит к п. 1

**4.1.2) Альтернативный сценарий «ход большого робота, маленький робот вне поле действия большого робота»**

1. Сценарий начинается после п. 2 альтернативного сценария «ход большого робота»
2. Так как маленький робот находится вне поле действия большого робота, большой робот выбирает направление движения в сторону маленького робота из направлений “влево” и “вправо” или “пустое направление”
3. Сценарий переходит к п. 1 Главного сценария 4

**4.2) Альтернативный сценарий «в соседней ячейке находится робот»**

1. Сценарий начинается после п. 5 сценария 4
2. Соседняя ячейка сообщает, что в ней находится робот
3. Сценарий переходит к п 5.2. Главного сценария.

**4.2.1) Альтернативный сценарий «в соседней ячейке находится робот, ход большого робота»**

1. Сценарий начинается после п. 2 сценария 4.1
2. Большой робот инициирует уничтожение маленького робота
3. Сценарий переходит к п 5.1. Главного сценария

**4.3) Альтернативный сценарий «между соседней ячейкой и текущей находится стена»**

1. Сценарий начинается после п. 1 сценария 4
2. Ячейка сообщает, что стена имеется
3. Сценарий переходит к п. 5.2. Главного сценария

**4.4) Альтернативный сценарий «конец хода большого робота»**

1. Сценарий начинается после п. 4.1 Главного сценария
2. Большой робот анализирует расстояние до маленького робота
3. Сценарий переходит к п. 4.2 Главного сценария

**4.4.1) Альтернативный сценарий** **«конец хода большого робота, маленький робот в соседней клетке»**

1. Сценарий начинается после п. 2 альтернативного сценария «конец хода большого робота»
2. Так как маленький робот находится в соседней клетке, большой робот инициирует уничтожение маленького робота
3. Большой робот сообщает игре о том, что маленький робот уничтожен
4. Сценарий переходит к п. 5 Главного сценария

**4.5) Альтернативный сценарий** **«маленький робот попадает в болото»**

1. Сценарий начинается после п. 11 сценария «робот перемещается на соседнюю ячейку»
2. Так как маленький робот находится в ячейке, в которой есть болото, он инициирует свое уничтожение
3. Маленький робот сообщает игре о том, что он уничтожен
4. Сценарий переходит к п. 5 Главного сценария

**4.6) Альтернативный сценарий «большой робот попадает в болото»**

1. Сценарий начинается после п. 11 сценария «робот перемещается на соседнюю ячейку»
2. Так как большой робот находится в ячейке, в которой есть болото, робот пропускает три следующих хода
3. Сценарий переходит к п 4.2 Главного сценария

**5) Сценарий «победа - маленький робот телепортировался»**

1. Поле сообщает игре, что “телепортирован“ робот
2. Игра считает победителем “телепортированного” робота, то есть игрока.

**5.1) Альтернативный сценарий «проигрыш - большой робот, управляемый компьютером, “поймал” маленького»**

1. Сценарий начинается с п. 2 сценария 5
2. Большой робот сообщает игре, что он “поймал” маленького робота
3. Сценарий переходит к п. 5 Главного сценария

**6) Сценарий «досрочное завершение игры»**

1. Сценарий начинается в любой точке главного сценария, когда пользователь инициирует завершение игры.
2. Игра завершается без определения победителя.

## 3.3 Словарь предметной области

**Игра** - игра знает о поле. Игра управляет игровым циклом: определяет очередного игрока, определяет окончание, определяет победителя. Игра определяет начальную расстановку игровых элементов (роботы, стены, выход, батарейки) на поле. Игра “следит” за маленьким роботом, большим роботом, точкой выхода.

**Поле** - прямоугольная область, состоящая из ячеек. Между ячейками возможно расположение стен. По границе поля могут располагаться стены. Позволяет получить роботов, находящихся на поле, и роботов, “телепортированных” с поля.

**Ячейка** - квадратная область поля. Знает о четырёх соседних ячейках и граничащих с ней стен. Одновременно может содержать в себе только одного робота и только одну батарейку.

**Стена** - непроходимое препятствие для робота, располагающееся между ячейками.

**Лабиринт** - умеет создавать горизонтальные или вертикальные последовательности стен.

**Болото** - препятствие произвольного размера, располагающееся на ячейках поля. При попадании большого робота в болото, он пропускает три хода. Маленькому роботу заходить в болото нельзя.

**Точка выхода** - разновидность ячейки. На поле может быть только одна ячейка, являющаяся выходом. “Телепортирует” робота, находящегося на ней, с поля внутрь себя. Помнит всех “телепортированных роботов”.

**Робот** - умеет перемещаться в соседние ячейки. Не может переходить через стены.

**Маленький робот** - разновидность робота, управляемого игроком. Имеет все свойства робота. Может быть “телепортированным”, поместившись в “точку выхода”. Условие победы - достигнуть точки выхода.

**Большой робот** - разновидность робота, управляемого компьютером. Имеет все свойства робота. Действует рационально при приближенности к маленькому роботу не менее, чем на 2 клетки (идет оптимальным маршрутом), в противном случае передвигается только влево или вправо, в сторону маленького робота. Если находится вдалеке от маленького робота по оси Y, но на одной оси X, пропускает ход. Условие победы - уничтожить маленького робота.

**Активный робот** - робот, который может совершать действие в текущий ход.

**Игрок** - пользователь игры, управляющий маленьким роботом при помощи клавиатуры.

**Компьютер** - виртуальный оппонент игрока, управляющий большим роботом с заданным алгоритмом.

## 3.4 Структура программы на уровне классов

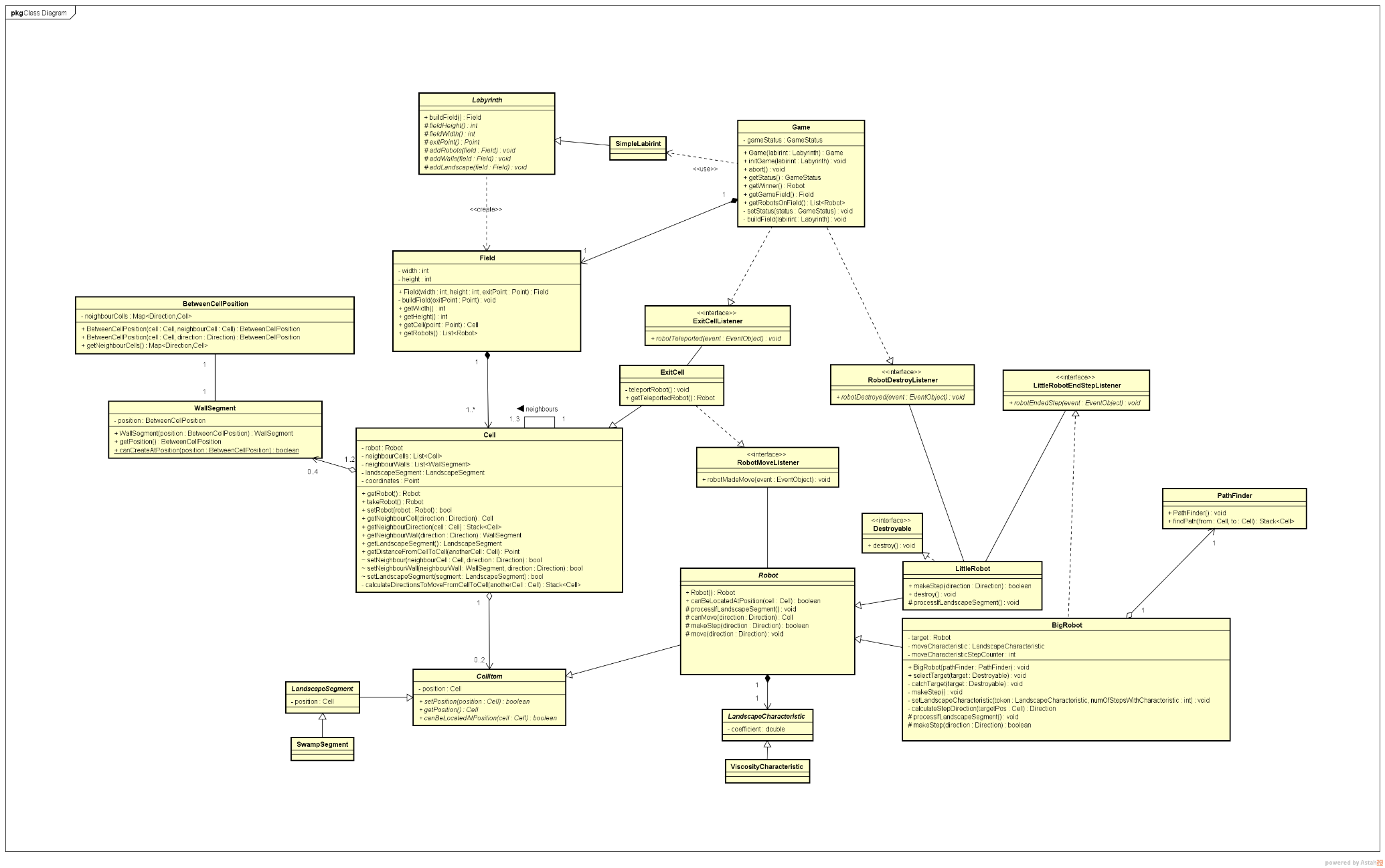


Диаграмма классов вычислительной модели

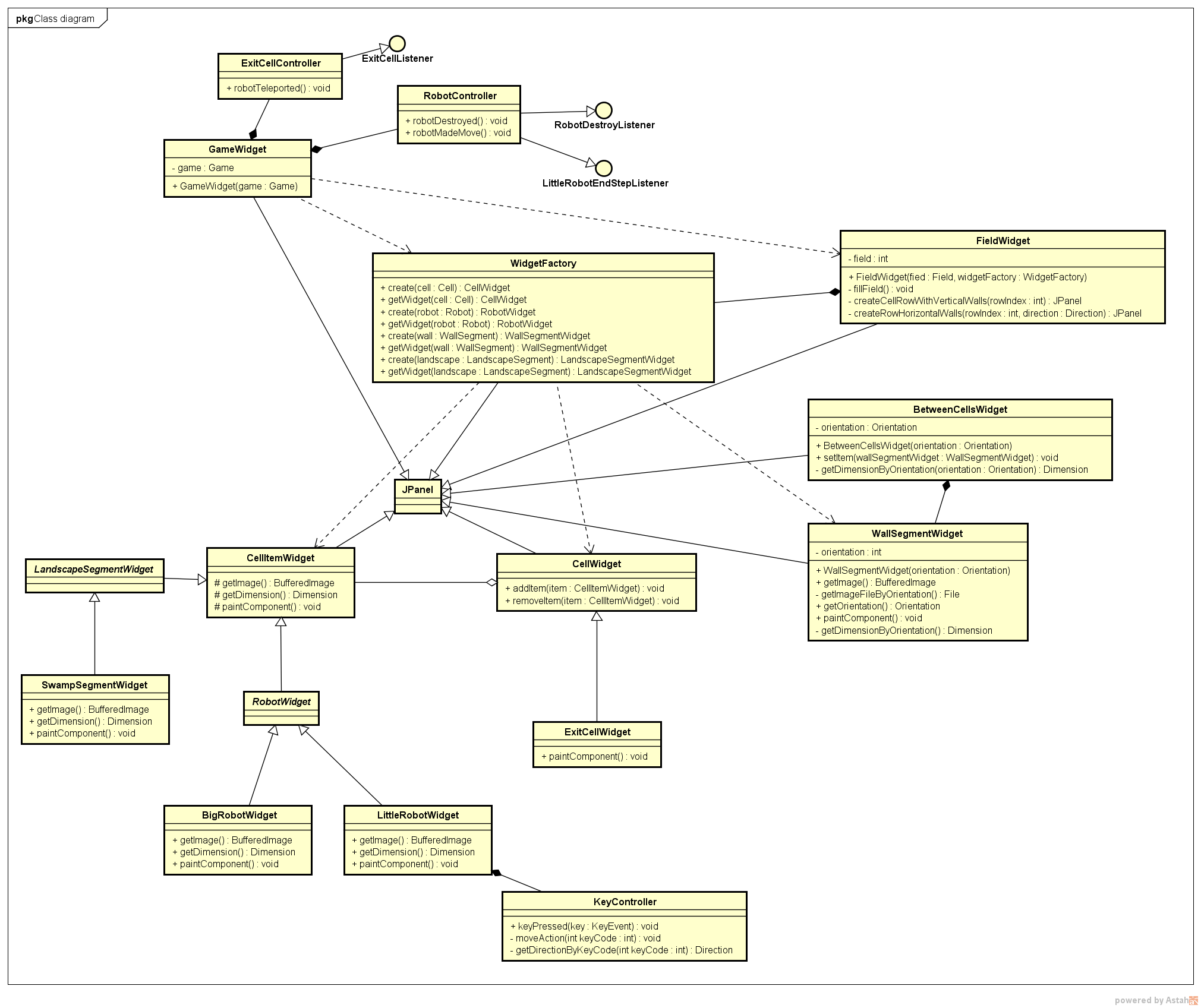
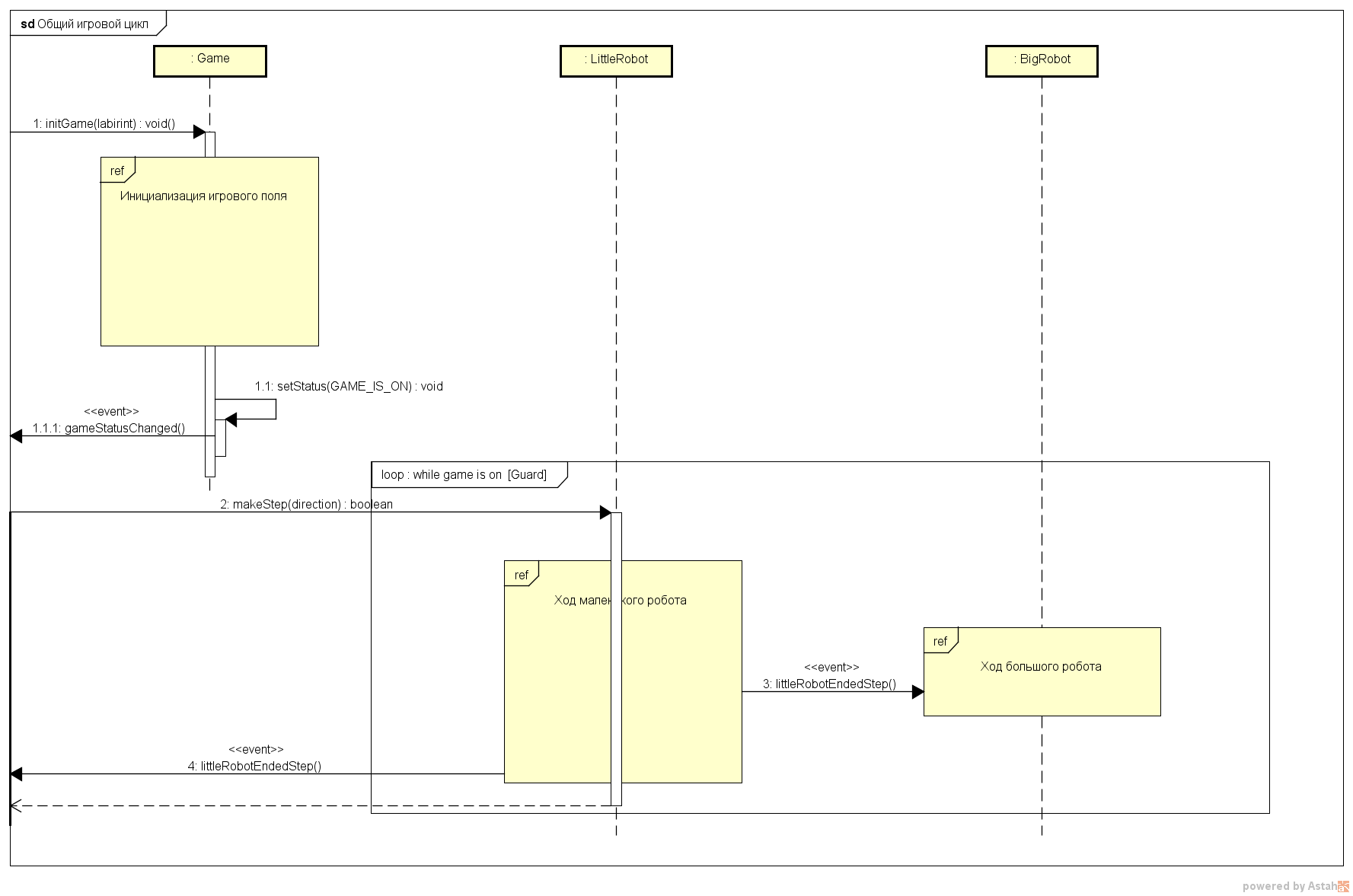
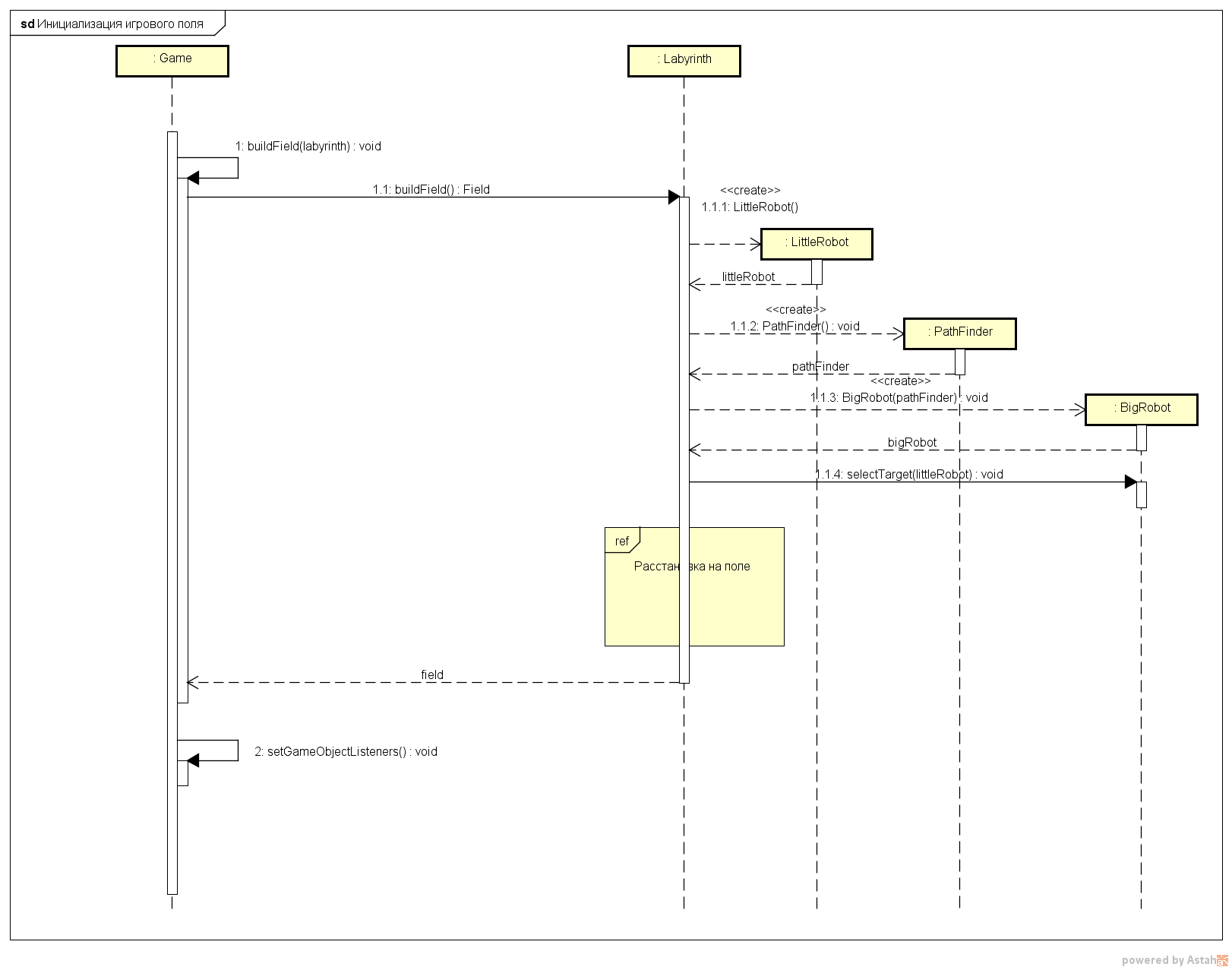


Диаграмма классов UI

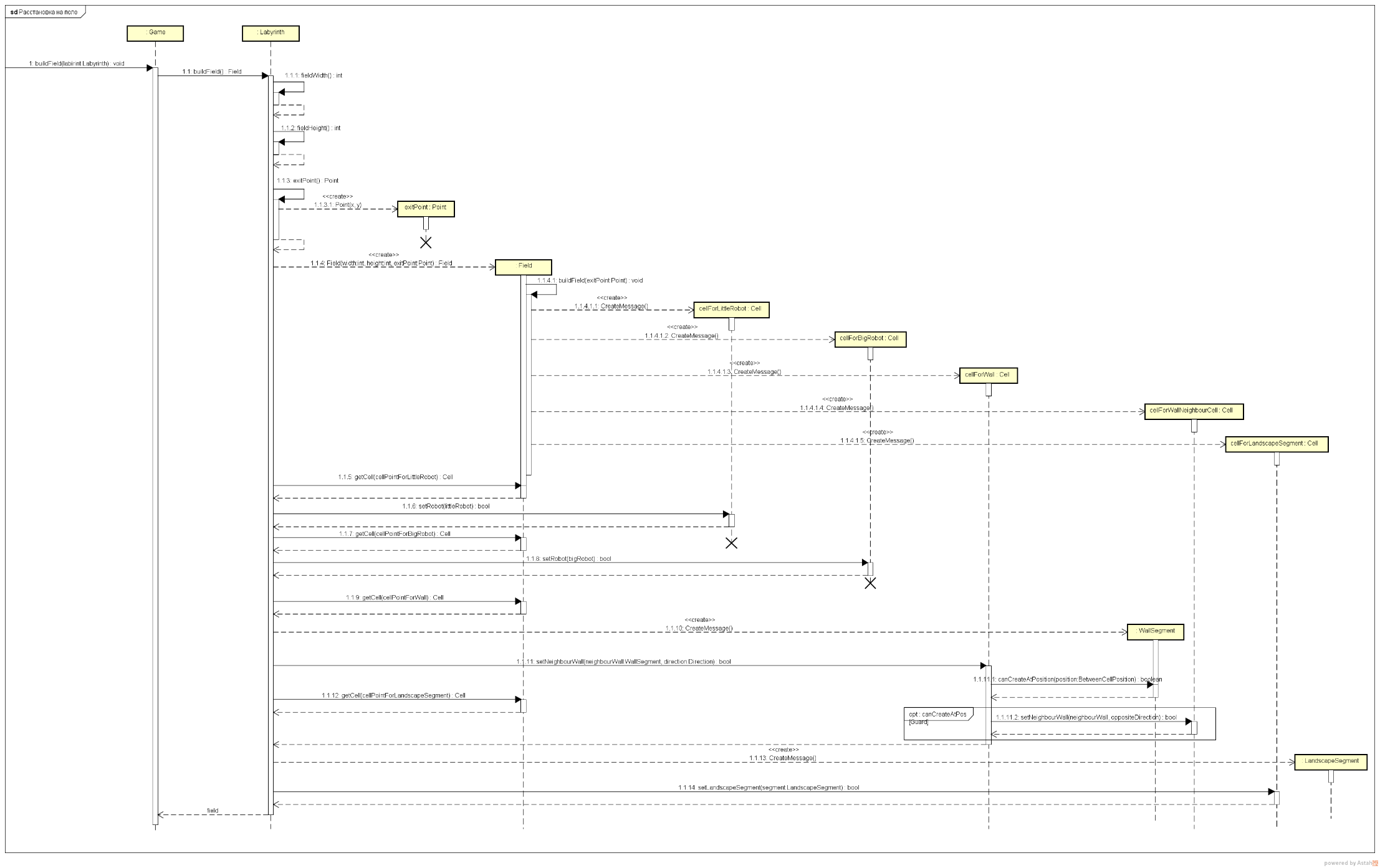
## 3.5 Типовые процессы в программе



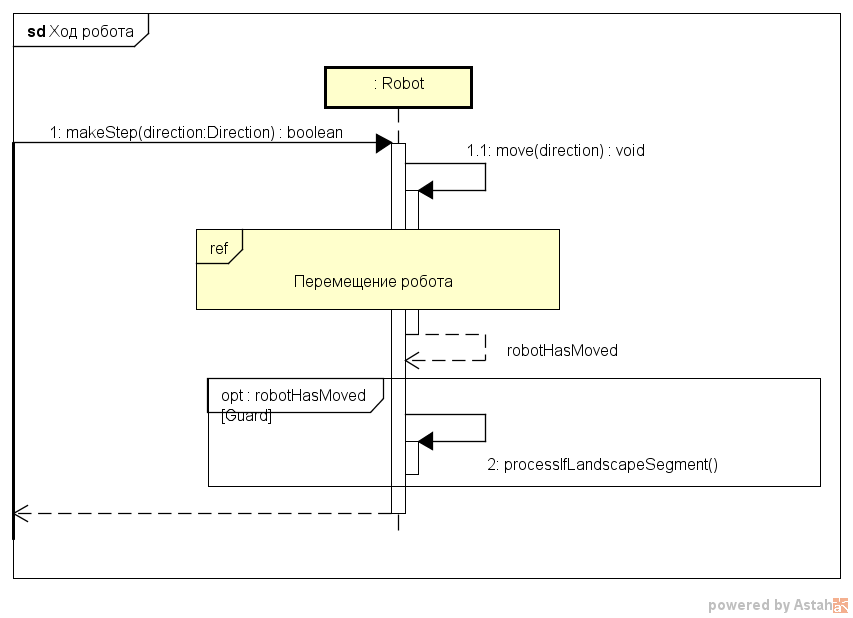
Общий игровой цикл



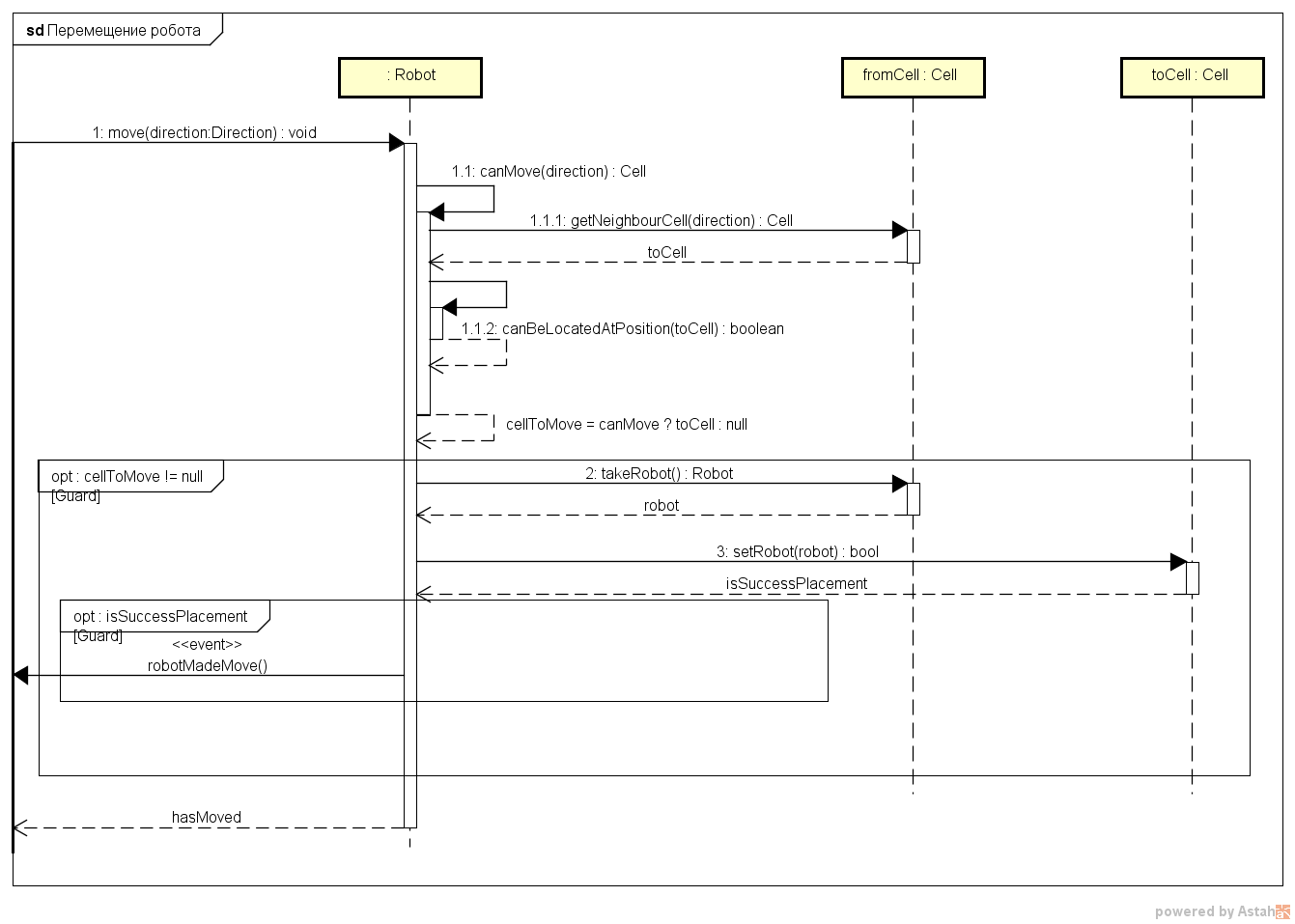
Инициализация игрового поля



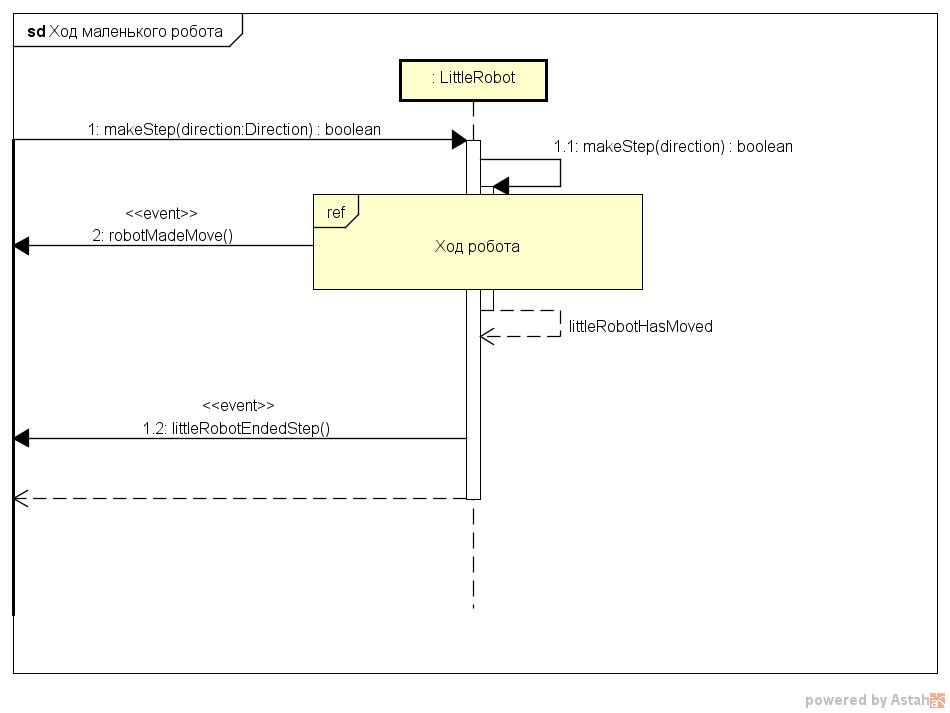
Расстановка на поле



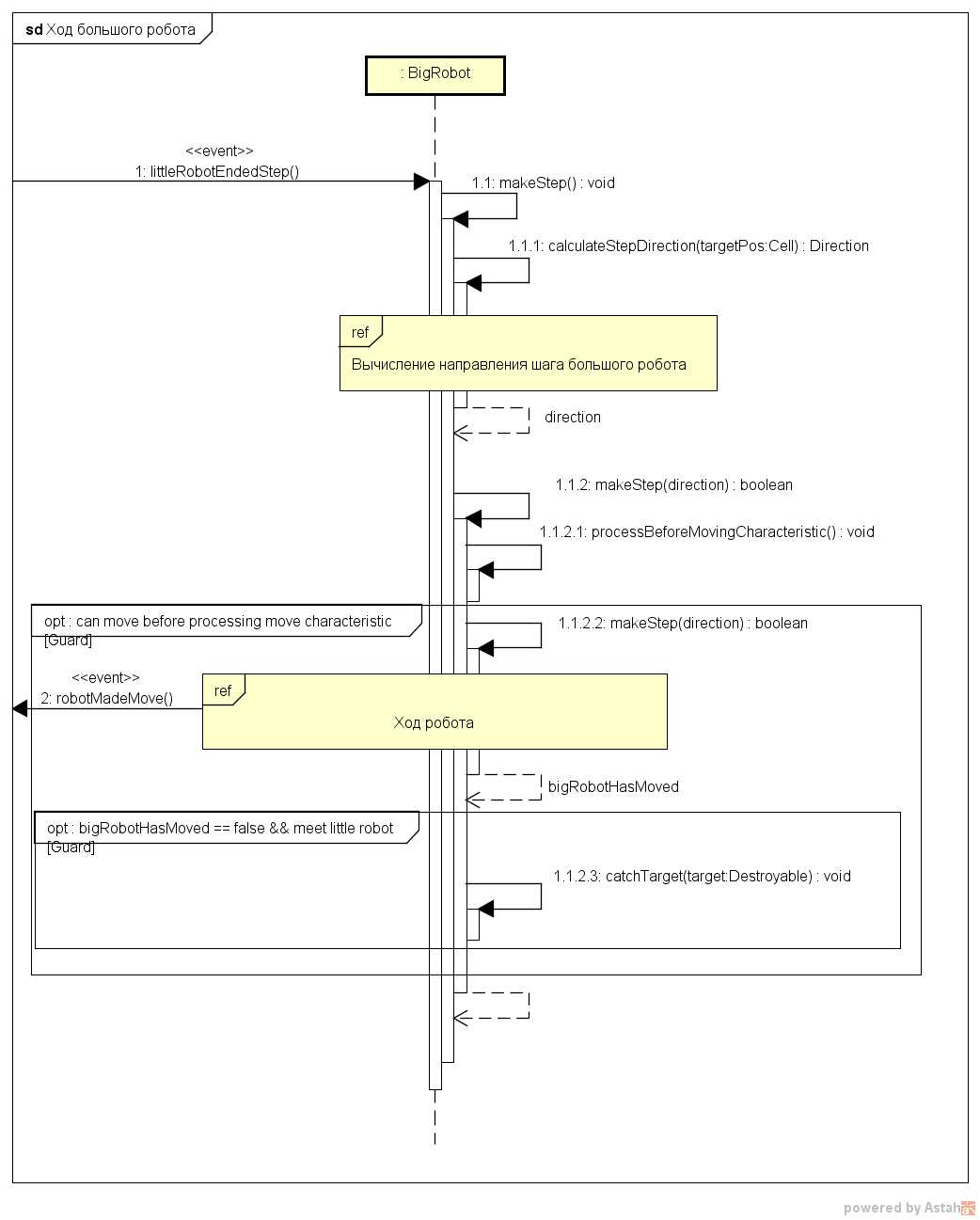
Ход робота



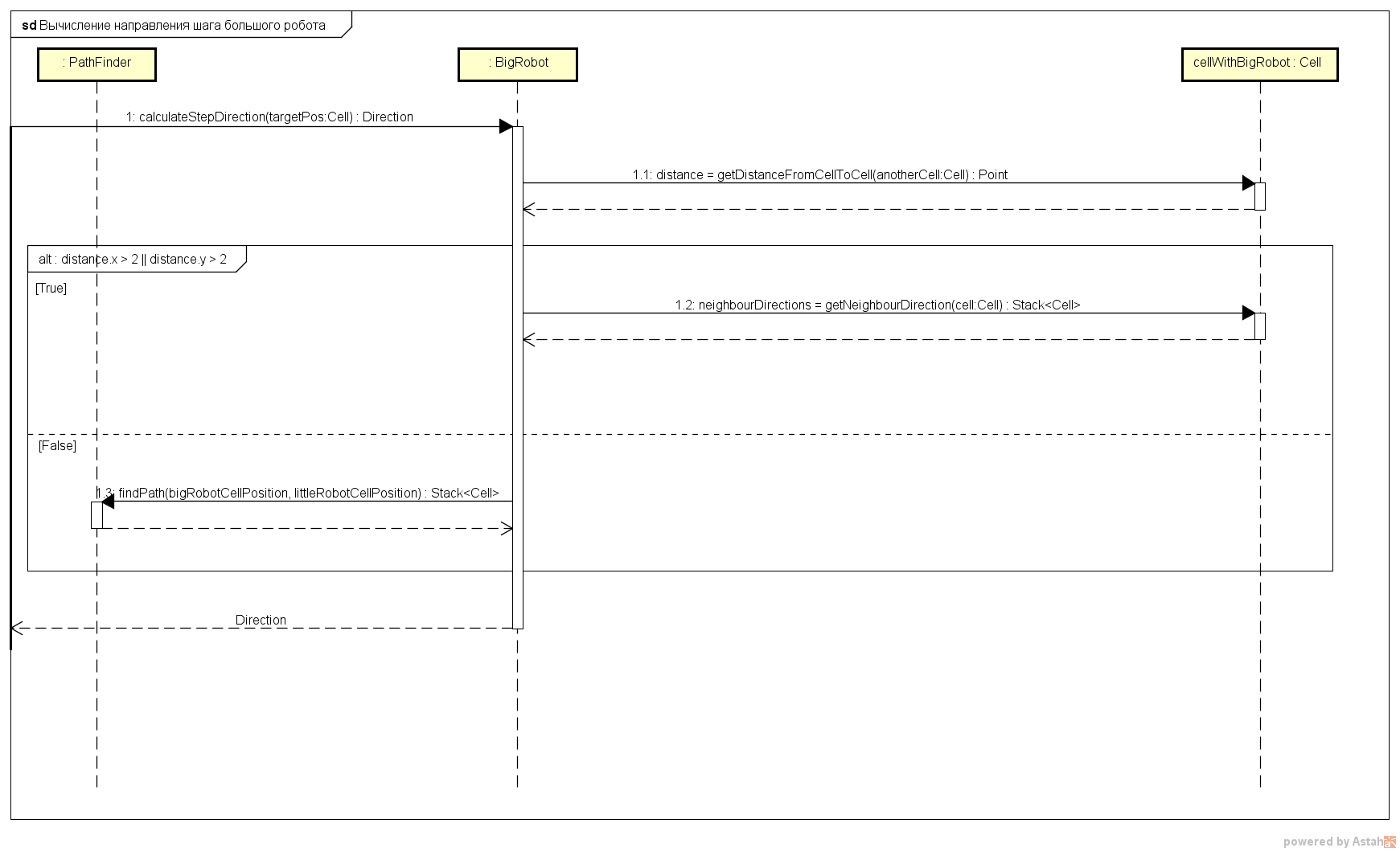
Перемещение робота



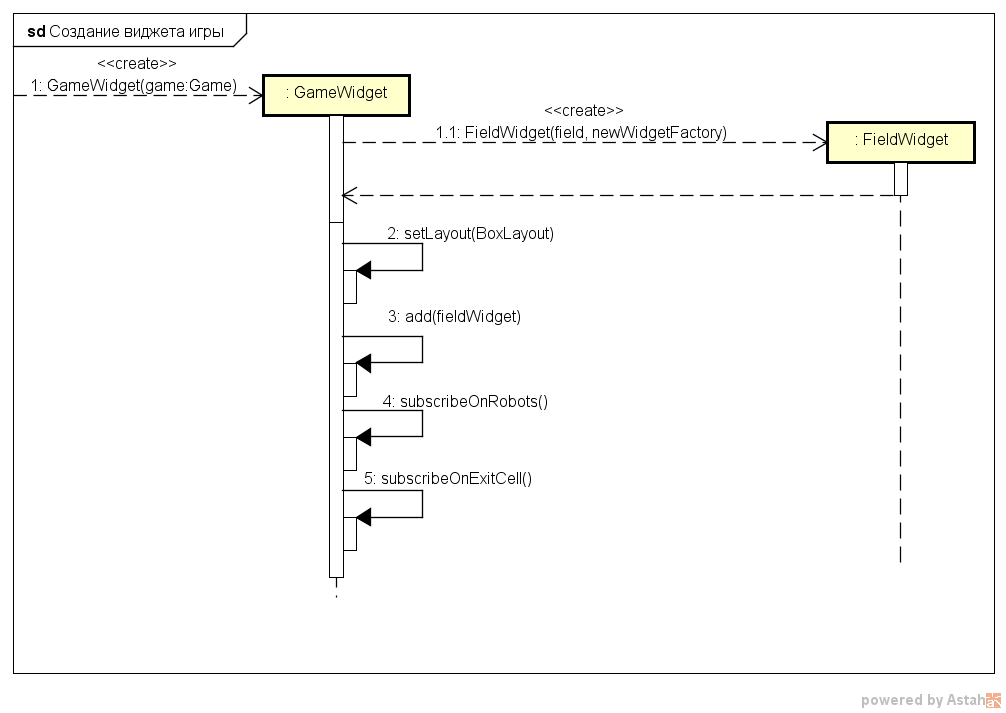
Ход маленького робота



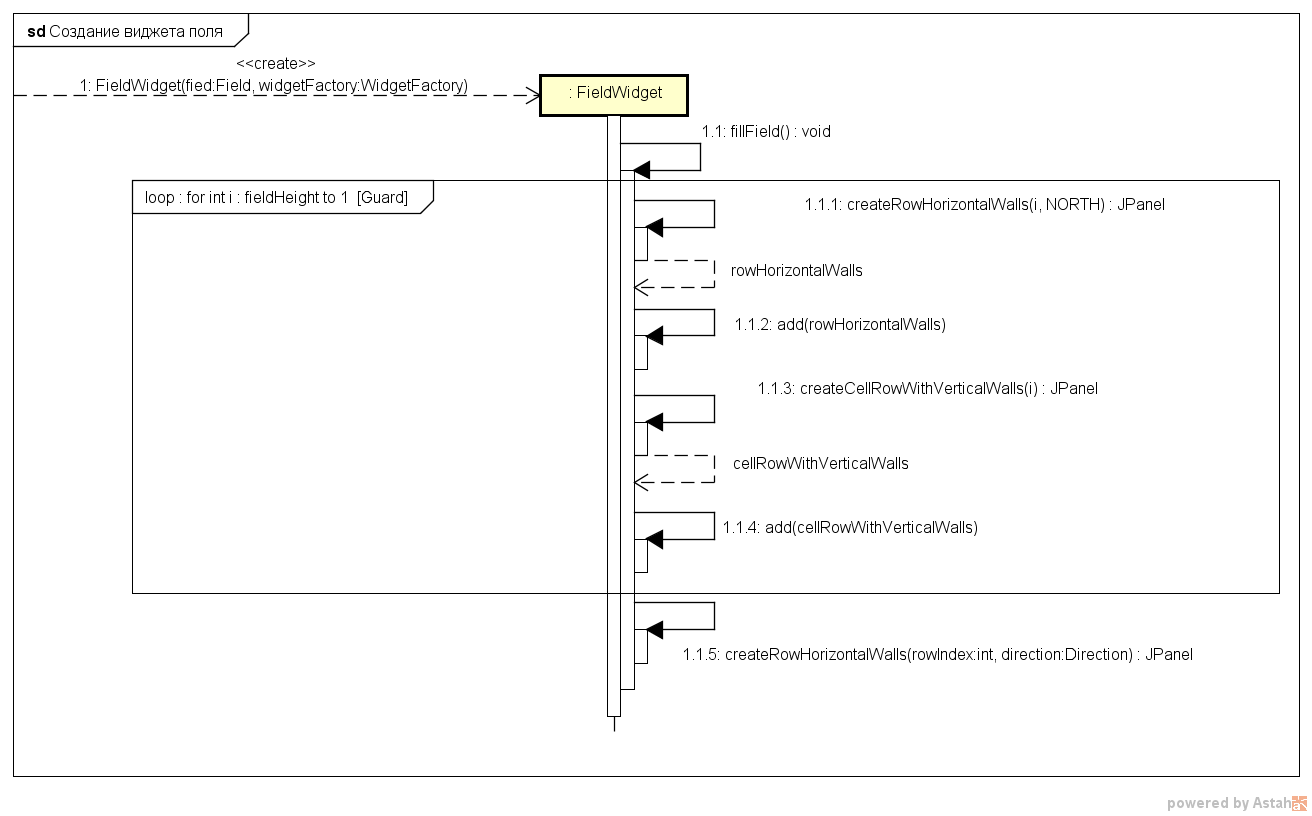
Ход большого робота



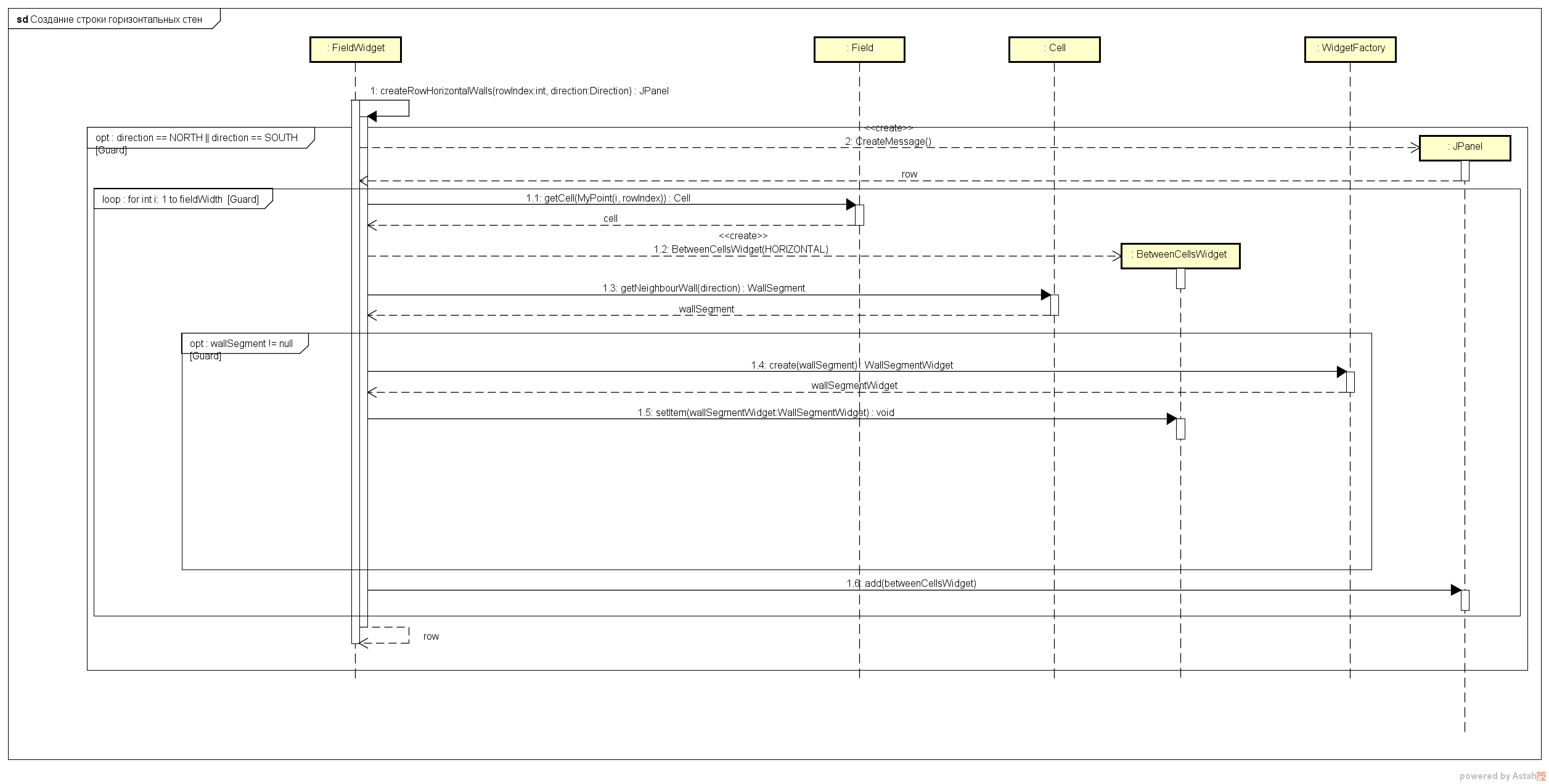
Вычисление направления шага большого робота



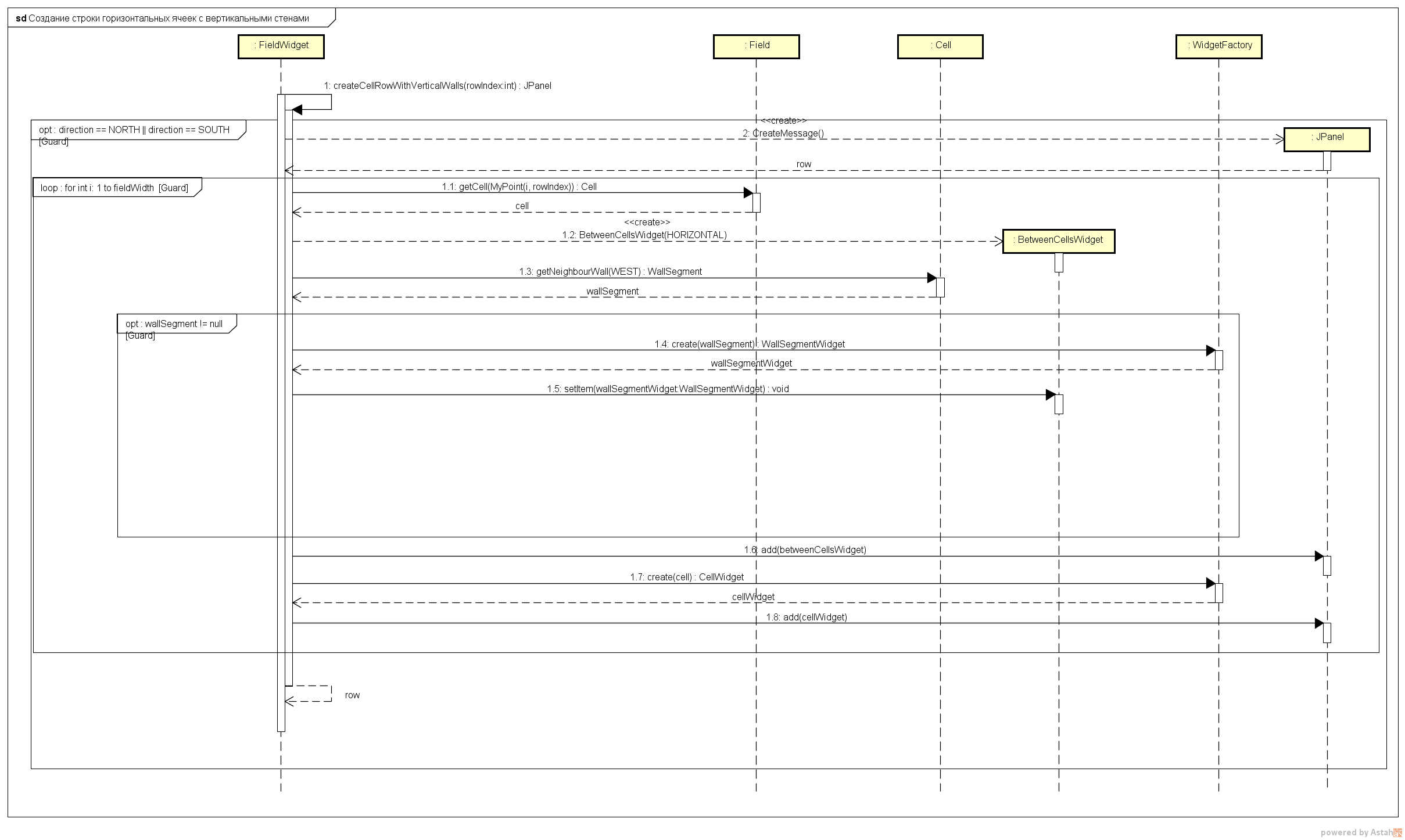
Создание виджета игры



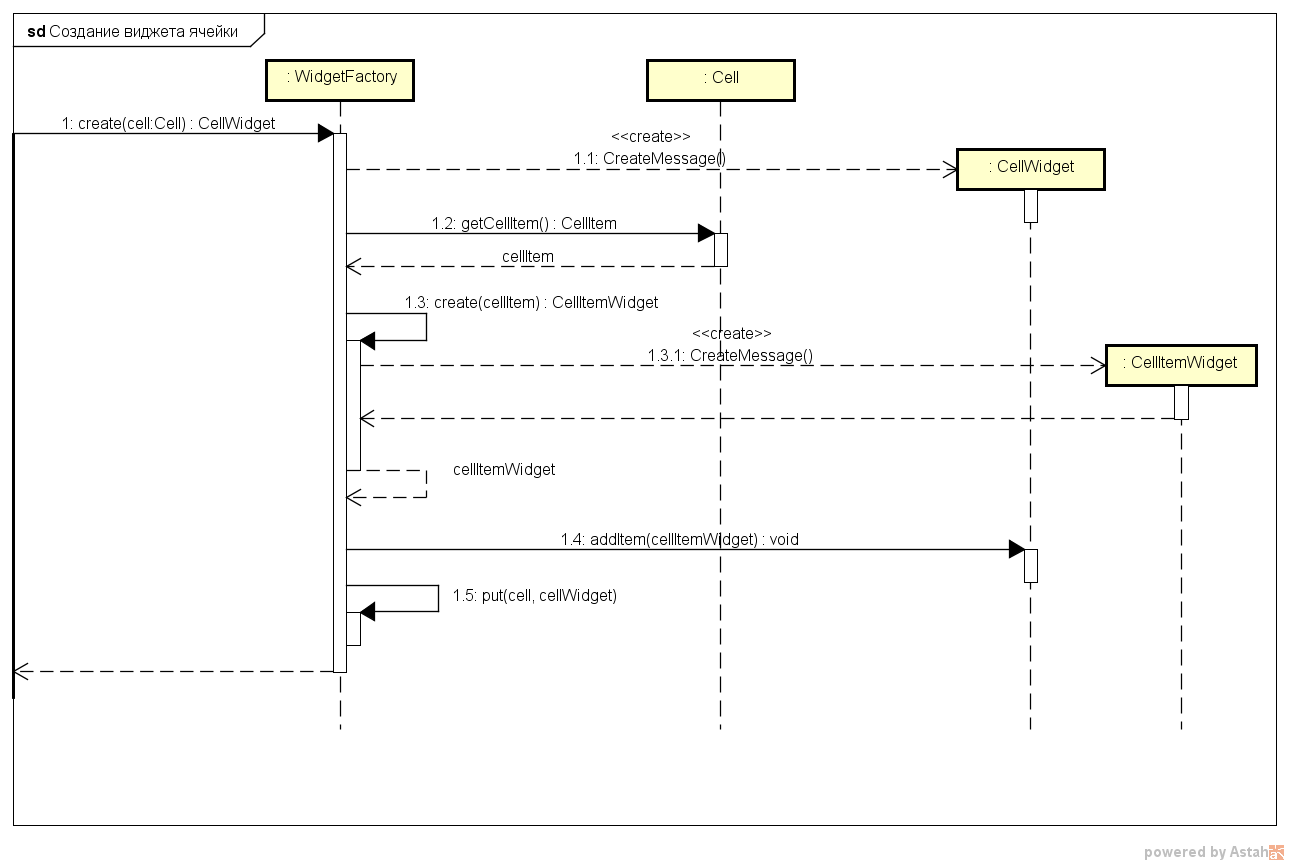
Создание виджета поля



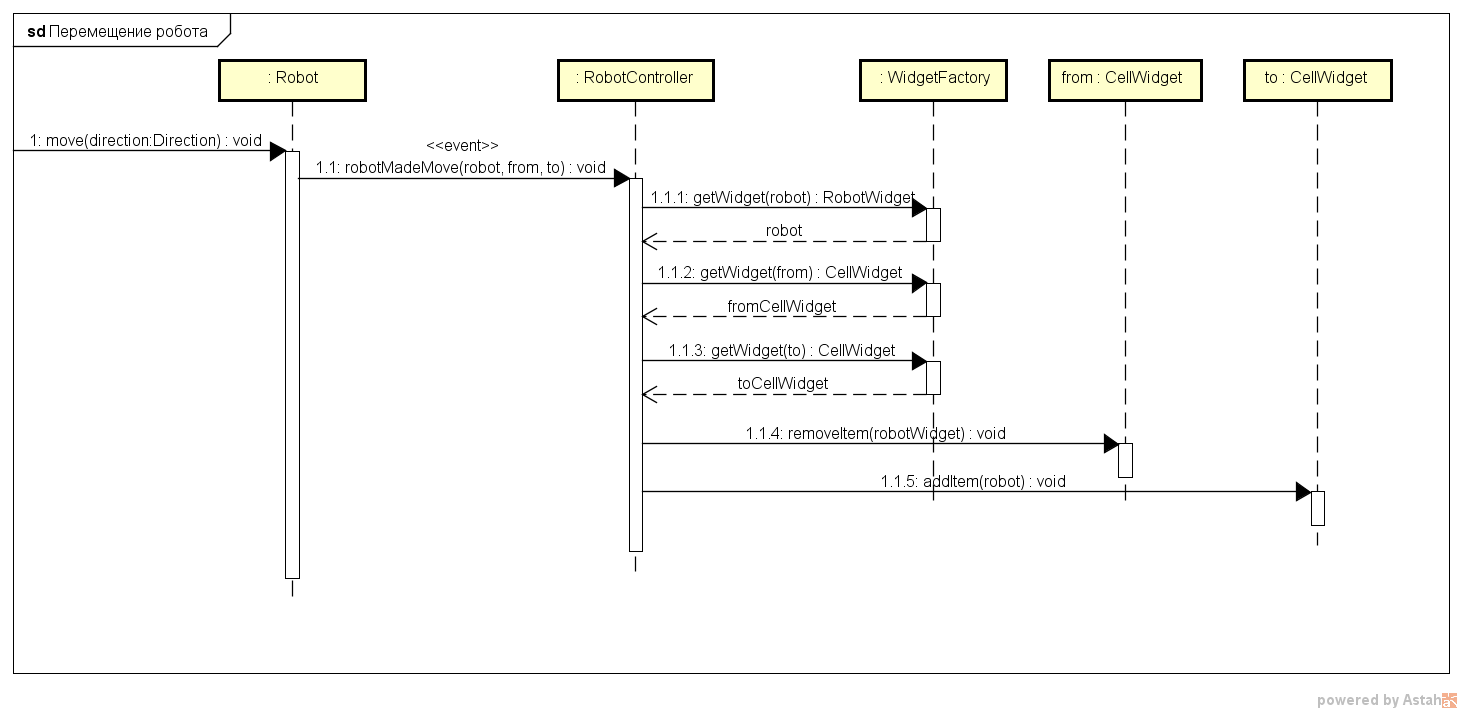
Создание строки горизонтальных стен



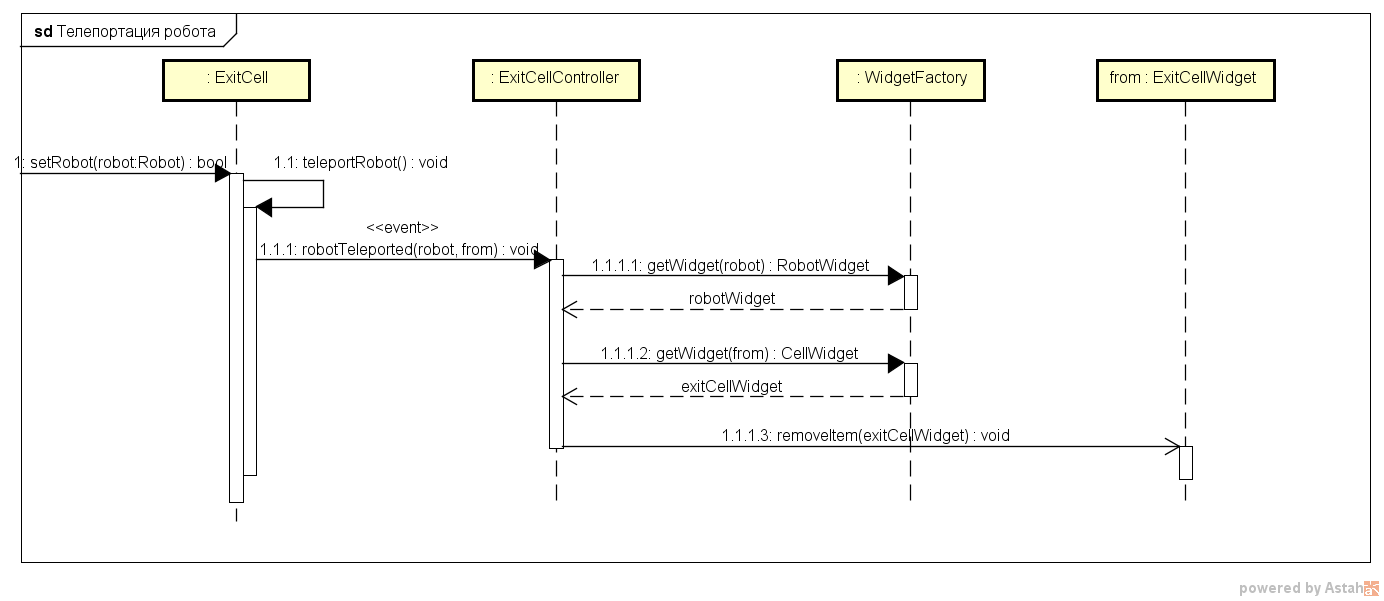
Создание горизонтальных ячеек с вертикальными стенами



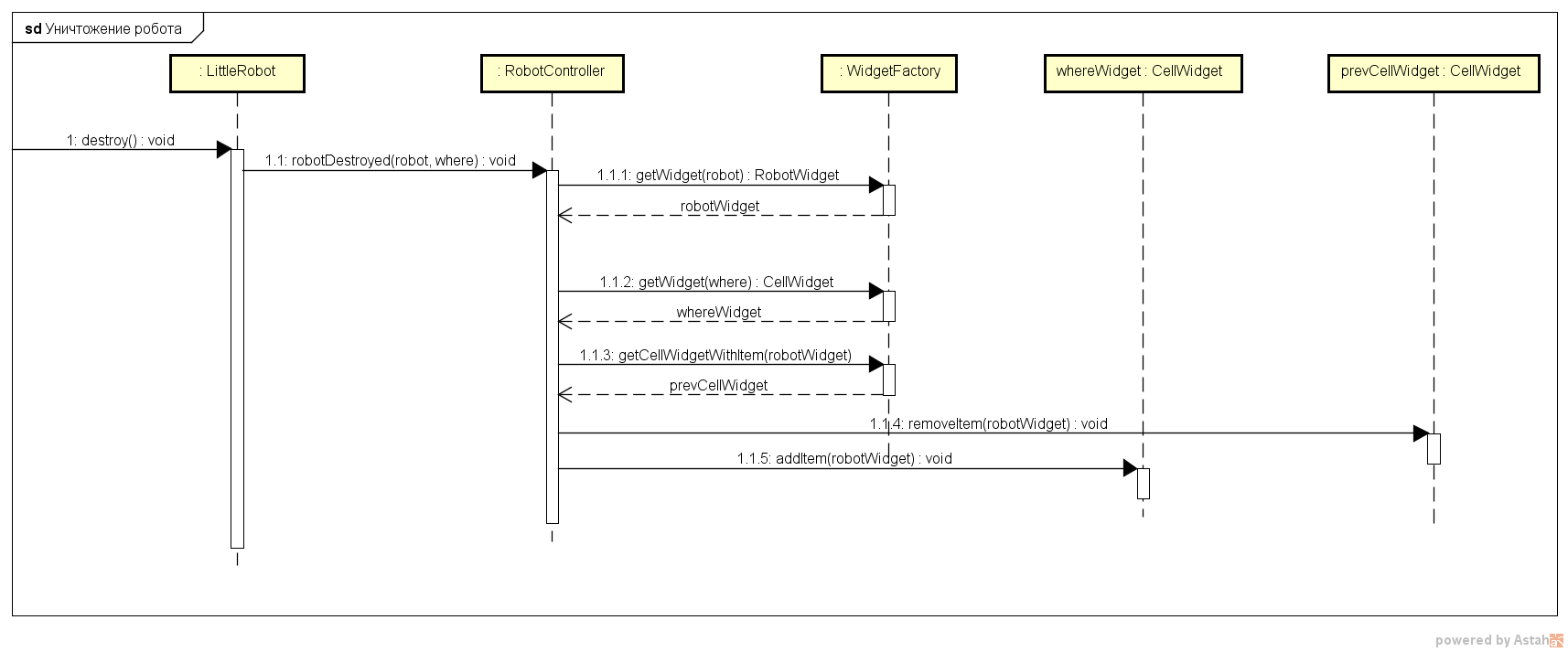
Создание виджета ячейки



Перемещение робота



Телепортация робота



Уничтожение робота

## 3.6 Человеко-машинное взаимодействие

Общий вид главного экрана представлен ниже на рис. 1. На нем располагается игровое поле, на котором изображены умный робот, глупый робот, телепорт (точка выхода), клетки с болотом и стены, стоящие между клетками.

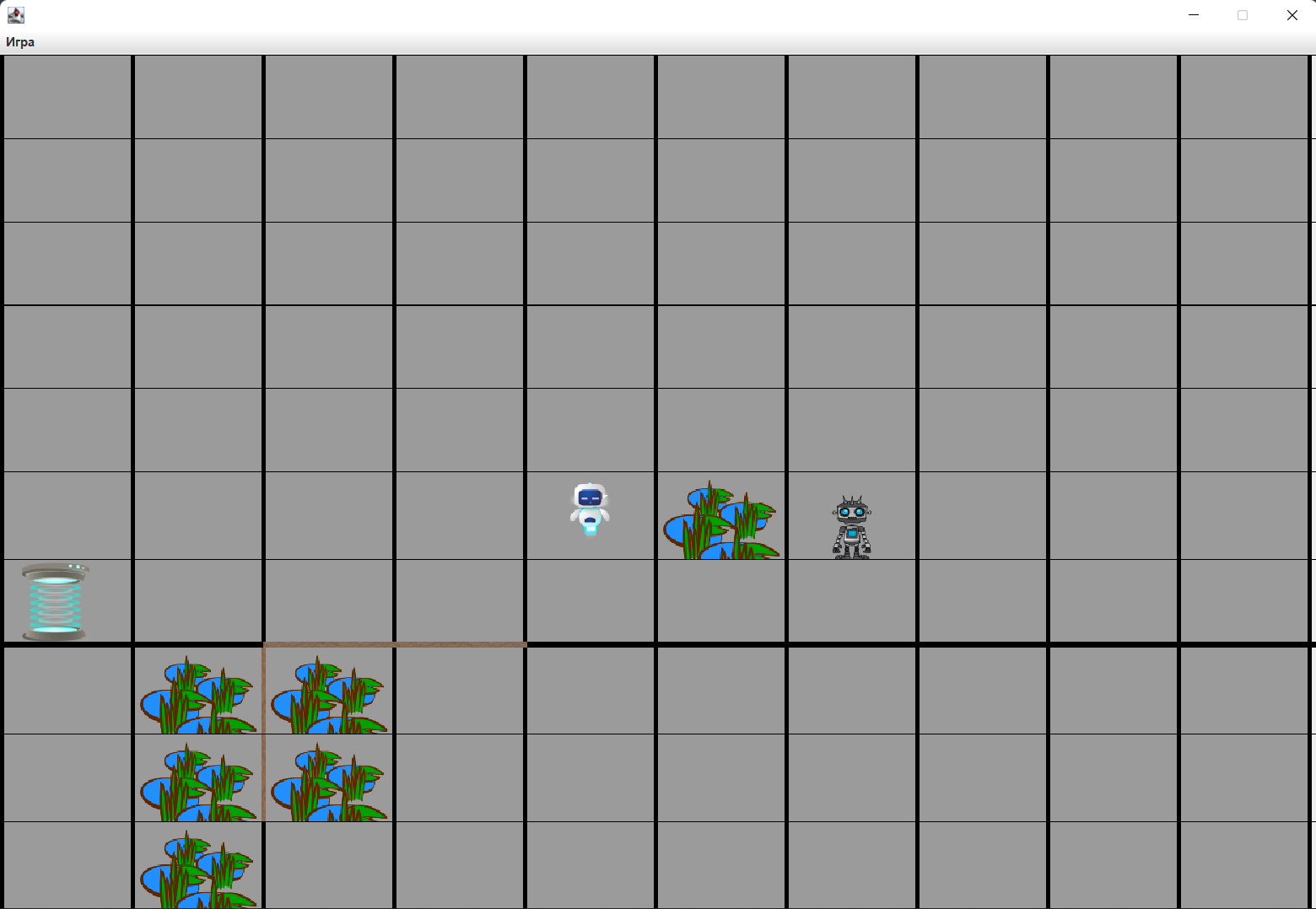


Рис 1. Пример пользовательского интерфейса игры

Управление умным роботом осуществляется посредством нажатия на клавиатуре следующих клавиш:

W – для передвижения вверх;  
A – для передвижения влево;  
S – для передвижения вниз;  
D – для передвижения вправо.

Изображение умного робота представлено на рисунке 2.

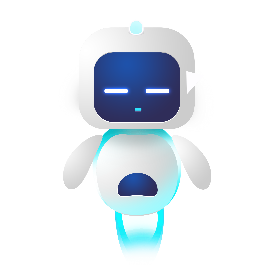


Рис. 2. Умный робот

Изображение уничтоженного умного робота представлено на рисунке 3.



Рис. 3. Уничтоженный умный робот

Изображение глупого робота представлено на рисунке 4.

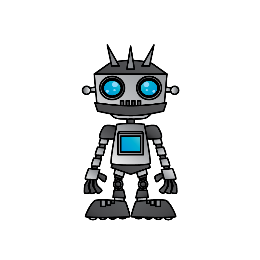


Рис. 4. Глупый робот

Изображение стены представлено на рисунке 5.



Рис. 5. Стена

Изображение болото представлено на рисунке 6.

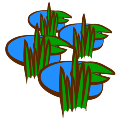


Рис. 6. Болото

Изображение точки выхода представлено на рисунке 7.

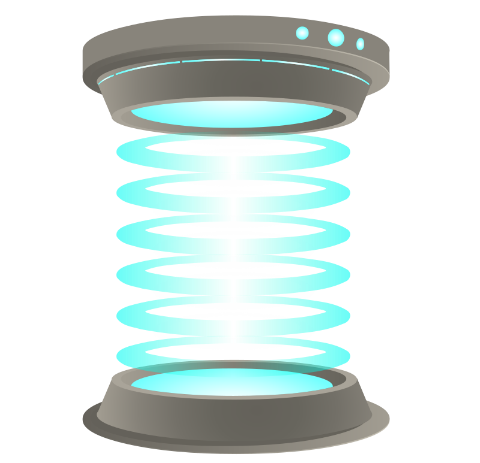


Рис 7. Телепорт (точка выхода)

## 3.7 Реализация ключевых классов

Класс игры

public class Game implements ExitCellListener, RobotDestroyListener {  
 // ----------------------------------------------- Статус игры ----------------------------------------------  
 private GameStatus \_gameStatus = null;  
  
 public GameStatus getGameStatus() {  
 return \_gameStatus;  
 }  
  
 private void setGameStatus(GameStatus gameStatus) {  
 \_gameStatus = gameStatus;  
 fireGameStatusChange();  
 }  
  
 // ----------------------------------------------- Инициализация игры ----------------------------------------------  
 public void initGame(Labyrinth labyrinth) {  
 buildField(labyrinth);  
 setGameObjectsListeners();  
 setGameStatus(GameStatus.*GAME\_IS\_ON*);  
 }  
  
 // ----------------------------------------------- Завершение игры ----------------------------------------------  
 public void abort() {  
 \_gameField = null;  
 setGameStatus(GameStatus.*GAME\_ABORTED*);  
 }  
  
 // ----------------------------------------------- Игровое поле ----------------------------------------------  
 private Field \_gameField = null;  
  
 private void buildField(Labyrinth labyrinth) {  
 \_gameField = labyrinth.buildField();  
 }  
  
 public Field getGameField() {  
 return \_gameField;  
 }  
  
 // ----------------------------------------------- Роботы ----------------------------------------------  
 public List<Robot> getRobotsOnField() {  
 return \_gameField.getRobots();  
 }  
  
 // ----------------------------------------------- Установить взаимную прослушку объектов --------------------------  
 private void setGameObjectsListeners() {  
 LittleRobot littleRobot = null;  
 BigRobot bigRobot = null;  
 for (var robot: \_gameField.getRobots()) {  
 if (robot instanceof LittleRobot) littleRobot = (LittleRobot)robot;  
 if (robot instanceof BigRobot) bigRobot = (BigRobot)robot;  
 }  
  
 littleRobot.addRobotMoveListener(\_gameField.getExitCell());  
 littleRobot.addRobotMoveListener(bigRobot);  
  
 littleRobot.addRobotDestroyListener(this);  
  
 \_gameField.getExitCell().addExitCellListener(this);  
 }  
  
 // ----------------------------------------------- Наблюдение за точкой выхода -------------------------------------  
 @Override  
 public void robotTeleported(RobotTeleportEvent e) {  
 setGameStatus(GameStatus.*WINNER\_IS\_LITTLE\_ROBOT*);  
 }  
  
 // ----------------------------------------------- Наблюдение за существованием маленького робота ------------------  
 @Override  
 public void robotDestroyed(RobotDestroyEvent e) {  
 setGameStatus(GameStatus.*WINNER\_IS\_BIG\_ROBOT*);  
 }  
  
 // ----------------------------------------------- Порождение события ----------------------------------------------  
 private final List<GameStatusChangedListener> \_listeners = new ArrayList<>();  
  
 public void addGameStatusChangedListener(GameStatusChangedListener l) {  
 \_listeners.add(l);  
 }  
  
 public void removeGameStatusChangedListener(GameStatusChangedListener l) {  
 \_listeners.remove(l);  
 }  
  
 protected void fireGameStatusChange() {  
 for (GameStatusChangedListener l : \_listeners) {  
 GameStatusChangeEvent e = new GameStatusChangeEvent(this);  
 e.setGameStatus(\_gameStatus);  
 l.gameStatusChanged(e);  
 }  
 }  
}

Виджет игры

public class GameWidget extends JPanel {  
 private final Game \_game;  
  
 private final HashMap<Field, FieldWidget> \_fields = new HashMap<>();  
  
 public GameWidget(@NotNull Game game) {  
 \_game = game;  
 \_fields.put(\_game.getGameField(), new FieldWidget(\_game.getGameField(), new WidgetFactory()));  
  
 setLayout(new BoxLayout(this, BoxLayout.*X\_AXIS*));  
 add(\_fields.get(\_game.getGameField()));  
  
 subscribeOnRobots();  
 subscribeOnExitCell();  
 }  
  
 private void subscribeOnRobots() {  
 List<Robot> robots = \_game.getRobotsOnField();  
 RobotController robotController = new RobotController();  
 for(Robot robot : robots) {  
 robot.addRobotMoveListener(0, robotController);  
 if (robot instanceof LittleRobot) {  
 ((LittleRobot) robot).addRobotDestroyListener(0, robotController);  
 }  
 }  
 }  
  
 private void subscribeOnExitCell() {  
 ExitCell exitCell = \_game.getGameField().getExitCell();  
 exitCell.addExitCellListener(0, new ExitCellController());  
 }  
  
 private class RobotController implements RobotMoveListener, RobotDestroyListener {  
  
 @Override  
 public void robotDestroyed(RobotDestroyEvent e) {  
 CellItemWidget robotWidget = \_fields.get(\_game.getGameField()).getWidgetFactory().getWidget(e.getDestroyedRobot());  
 CellWidget cellWhereDestroyedWidget = \_fields.get(\_game.getGameField()).getWidgetFactory().getWidget(e.getCellWhereDestroyed());  
  
 for (var neighbourCell : e.getCellWhereDestroyed().getNeighbourCells().values()) {  
 \_fields.get(\_game.getGameField()).getWidgetFactory().getWidget(neighbourCell).removeItem(robotWidget);  
 }  
  
 cellWhereDestroyedWidget.addItem(robotWidget);  
  
 robotWidget.repaint();  
 }  
  
 @Override  
 public void robotMadeMove(RobotMoveEvent e) {  
 CellItemWidget robotWidget = \_fields.get(\_game.getGameField()).getWidgetFactory().getWidget(e.getRobot());  
 CellWidget from = \_fields.get(\_game.getGameField()).getWidgetFactory().getWidget(e.getFromCell());  
 CellWidget to = \_fields.get(\_game.getGameField()).getWidgetFactory().getWidget(e.getToCell());  
 from.removeItem(robotWidget);  
 to.addItem(robotWidget);  
 robotWidget.repaint();  
  
 if (e.getRobot() instanceof LittleRobot && !(to instanceof ExitCellWidget)) {  
 robotWidget.requestFocusInWindow();  
 }  
 }  
 }  
  
 private class ExitCellController implements ExitCellListener {  
  
 @Override  
 public void robotTeleported(RobotTeleportEvent e) {  
 Robot robot = e.getTeleportedRobot();  
 Cell teleport = e.getTeleportCell();  
 CellWidget teleportWidget = \_fields.get(\_game.getGameField()).getWidgetFactory().getWidget(teleport);  
 CellItemWidget robotWidget = \_fields.get(\_game.getGameField()).getWidgetFactory().getWidget(robot);  
 teleportWidget.removeItem(robotWidget);  
 }  
 }  
}

Класс поля

public class Field implements ExitCellListener {  
 public Field(int width, int height, MyPoint exitPoint) {  
 \_width = width;  
 \_height = height;  
 buildField(exitPoint);  
 }  
  
 // ----------------------------------------------- Высота и ширина поле --------------------------------------------  
 private final int \_width, \_height;  
  
 public int getWidth() {  
 return \_width;  
 }  
  
 public int getHeight() {  
 return \_height;  
 }  
  
 // ----------------------------------------------- Ячейки поля -----------------------------------------------------  
 private final TreeMap<MyPoint, Cell> \_cells = new TreeMap<>();  
  
 public TreeMap<MyPoint, Cell> getCells() {  
 return \_cells;  
 }  
  
 public Cell getCell(MyPoint coordinates) {  
 return \_cells.get(coordinates);  
 }  
  
 public int getIndexOfCell(Cell cell) {  
 int i = 0;  
 for (Cell c : \_cells.values()) {  
 if (c == cell) return i;  
 ++i;  
 }  
 return -1;  
 }  
  
 public Cell getCellByIndex(int index) {  
 int i = 0;  
 for (Cell cell : \_cells.values()) {  
 if (i == index) return cell;  
 ++i;  
 }  
 return null;  
 }  
  
 public boolean contains(Cell cell) {  
 return \_cells.containsValue(cell);  
 }  
  
 // ----------------------------------------------- Роботы, находящиеся на поле -------------------------------------  
 private final ArrayList<Robot> \_robots = new ArrayList<>();  
  
 public ArrayList<Robot> getRobots() {  
 if (!\_robots.isEmpty()) return \_robots;  
 for (var cell : getCells().values()) {  
 if (cell.getRobot() != null) {  
 \_robots.add(cell.getRobot());  
 }  
 }  
 return \_robots;  
 }  
  
 // ----------------------------------------------- Точка выхода ----------------------------------------------------  
  
 private ExitCell \_exitCell;  
  
 public ExitCell getExitCell() {  
 return \_exitCell;  
 }  
  
 // ----------------------------------------------- Построение поля из ячеек с точкой выхода в exitPoint ------------  
  
 private void buildField(MyPoint exitPoint) {  
 for (int i = 1; i <= \_width; ++i) {  
 for (int j = 1; j <= \_height; ++j) {  
 MyPoint cellCoordinates = new MyPoint(i, j);  
 if (cellCoordinates.equals(exitPoint)) {  
 \_exitCell = new ExitCell(cellCoordinates);  
 \_cells.put(cellCoordinates, \_exitCell);  
 } else {  
 \_cells.put(cellCoordinates, new Cell(cellCoordinates));  
 }  
 }  
 }  
  
 for (var cell : \_cells.entrySet()) {  
 for (var direction : Direction.*values*()) {  
 Cell neighbourCell = \_cells.get(cell.getKey().getNeighbourPoint(direction));  
 if (neighbourCell != null) {  
 cell.getValue().setNeighbourCell(neighbourCell, direction);  
 }  
 }  
 }  
 }  
  
 // ----------------------------------------------- Наблюдение за точкой выхода -------------------------------------  
  
 @Override  
 public void robotTeleported(RobotTeleportEvent e) {  
 \_robots.remove((Robot)e.getTeleportedRobot());  
 }  
}

Виджет поля

public class FieldWidget extends JPanel {  
 public FieldWidget(@NotNull Field field, @NotNull WidgetFactory widgetFactory) {  
 \_widgetFactory = widgetFactory;  
 \_field = field;  
 setLayout(new BoxLayout(this, BoxLayout.*Y\_AXIS*));  
 fillField();  
 }  
 private final WidgetFactory \_widgetFactory;  
  
 public WidgetFactory getWidgetFactory() {  
 return \_widgetFactory;  
 }  
  
 private final Field \_field;  
  
 private void fillField() {  
 for (int i = \_field.getHeight(); i >= 1; --i) {  
 add(createRowHorizontalWalls(i, Direction.*NORTH*));  
 add(createCellRowWithVerticalWalls(i));  
 }  
 add(createRowHorizontalWalls(1, Direction.*SOUTH*));  
 }  
  
 private JPanel createCellRowWithVerticalWalls(int rowIndex) {  
 JPanel cellRow = new JPanel();  
 cellRow.setLayout(new BoxLayout(cellRow, BoxLayout.*X\_AXIS*));  
  
 for (int i = 1; i <= \_field.getWidth(); ++i) {  
 Cell cell = \_field.getCell(new MyPoint(i, rowIndex));  
  
 BetweenCellsWidget westBetweenCellsWidget = new BetweenCellsWidget(Orientation.*VERTICAL*);  
 WallSegment westWallSegment = cell.getWallSegment(Direction.*WEST*);  
 if (westWallSegment != null) {  
 WallSegmentWidget westWallWidget = \_widgetFactory.create(westWallSegment, Orientation.*VERTICAL*);  
 westBetweenCellsWidget.setItem(westWallWidget);  
 }  
  
 cellRow.add(westBetweenCellsWidget);  
  
 CellWidget cellWidget = \_widgetFactory.create(cell);  
  
 cellRow.add(cellWidget);  
 }  
  
 Cell cell = \_field.getCell(new MyPoint(rowIndex, \_field.getWidth()));  
 BetweenCellsWidget eastBetweenCellsWidget = new BetweenCellsWidget(Orientation.*VERTICAL*);  
 WallSegment eastWallSegment = cell.getWallSegment(Direction.*EAST*);  
 if (eastWallSegment != null) {  
 WallSegmentWidget eastWallWidget = \_widgetFactory.create(eastWallSegment, Orientation.*VERTICAL*);  
 eastBetweenCellsWidget.setItem(eastWallWidget);  
 }  
  
 cellRow.add(eastBetweenCellsWidget);  
  
 return cellRow;  
 }  
  
 private JPanel createRowHorizontalWalls(int rowIndex, Direction direction) {  
 if (direction != Direction.*NORTH* && direction != Direction.*SOUTH*) throw new IllegalArgumentException("Неверно задано направление расположения стен");  
 JPanel row = new JPanel();  
 row.setLayout(new BoxLayout(row, BoxLayout.*X\_AXIS*));  
  
 for (int i = 1; i <= \_field.getWidth(); ++i) {  
 Cell cell = \_field.getCell(new MyPoint(i, rowIndex));  
  
 BetweenCellsWidget betweenCellsWidget = new BetweenCellsWidget(Orientation.*HORIZONTAL*);  
 WallSegment wallSegment = cell.getWallSegment(direction);  
  
 if (wallSegment != null) {  
 WallSegmentWidget wallSegmentWidget = \_widgetFactory.create(wallSegment, Orientation.*HORIZONTAL*);  
 betweenCellsWidget.setItem(wallSegmentWidget);  
 }  
  
 row.add(betweenCellsWidget);  
 }  
  
 return row;  
 }  
}

Класс робота

public abstract class Robot extends CellItem {  
  
 // ----------------------------------------------- Позиция ---------------------------------------------------------  
 public boolean setPosition(Cell cell) {  
 if (cell == null) return false;  
 if (\_position == cell) return true;  
 if (!canBeLocatedAtPosition(cell) && cell.getRobot() != this) return false;  
 \_position = cell;  
 return cell.setRobot(this);  
 }  
  
 public void setNullPosition() {  
 \_position = null;  
 }  
  
 public Cell getPosition() {  
 return \_position;  
 }  
  
 public boolean canBeLocatedAtPosition(Cell cell) {  
 return cell != null && cell.getRobot() == null;  
 }  
  
 // ----------------------------------------------- Перемещение -----------------------------------------------------  
 protected Cell canMove(Direction direction) {  
 if (direction == null || \_position == null) return null;  
 Cell neighbourCell = \_position.getNeighbourCell(direction);  
 if (neighbourCell != null && canBeLocatedAtPosition(\_position.getNeighbourCell(direction))  
 && \_position.getWallSegment(direction) == null) {  
 return neighbourCell;  
 }  
 return null;  
 }  
  
 public boolean move(Direction direction) {  
 Cell cellToMove = canMove(direction);  
 if (cellToMove == null) return false;  
 Cell fromCell = getPosition();  
 getPosition().takeRobot();  
 if (setPosition(cellToMove)) {  
 processIfLandscapeSegment();  
 fireRobotMove(fromCell, cellToMove);  
 return true;  
 }  
 return false;  
 }  
  
 // ----------------------------------------------- Работа с ландшафтом ---------------------------------------------  
 protected abstract void processIfLandscapeSegment();  
  
 // ----------------------------------------------- Генерация события при перемещении -------------------------------  
 private final List<RobotMoveListener> \_moveListeners = new ArrayList<>();  
  
 public void addRobotMoveListener(RobotMoveListener l) {  
 \_moveListeners.add(l);  
 }  
  
 public void addRobotMoveListener(int index, RobotMoveListener l) {  
 \_moveListeners.add(index, l);  
 }  
  
 public void removeRobotMoveListener(RobotMoveListener l) {  
 \_moveListeners.remove(l);  
 }  
  
 protected void fireRobotMove(Cell fromCell, Cell toCell) {  
 for (RobotMoveListener l : \_moveListeners) {  
 RobotMoveEvent e = new RobotMoveEvent(this);  
 e.setRobot(this);  
 e.setFromCell(fromCell);  
 e.setToCell(toCell);  
 l.robotMadeMove(e);  
 }  
 }  
}

Класс маленького робота

public class LittleRobot extends Robot implements Destroyable {  
  
 // ----------------------------------------------- Перемещение -----------------------------------------------------  
 @Override  
 public boolean move(Direction direction) {  
 if (!super.move(direction)) {  
 if (\_position.getWallSegment(direction) == null && \_position.getNeighbourCell(direction) != null  
 &&\_position.getNeighbourCell(direction).getRobot() instanceof BigRobot) {  
 destroy();  
 return true;  
 }  
 return false;  
 }  
 return true;  
 }  
  
 // ----------------------------------------------- Работа с ландшафтом ---------------------------------------------  
 @Override  
 protected void processIfLandscapeSegment() {  
 if (\_position.getLandscapeSegment() != null && \_position.getLandscapeSegment() instanceof SwampSegment) {  
 destroy();  
 }  
 }  
  
 // ----------------------------------------------- Уничтожение -----------------------------------------------------  
 @Override  
 public void destroy() {  
 Cell whereDestroyedCell = \_position;  
 \_position.takeRobot();  
 fireRobotDestroy(whereDestroyedCell);  
 }  
  
 // ----------------------------------------------- Порождение события при уничтожении ------------------------------  
  
 private final List<RobotDestroyListener> \_destroyListeners = new ArrayList<>();  
  
 public void addRobotDestroyListener(RobotDestroyListener l) {  
 \_destroyListeners.add(l);  
 }  
  
 public void addRobotDestroyListener(int index, RobotDestroyListener l) {  
 \_destroyListeners.add(index, l);  
 }  
  
 public void removeRobotDestroyListener(RobotDestroyListener l) {  
 \_destroyListeners.remove(l);  
 }  
  
 protected void fireRobotDestroy(Cell whereDestroyedCell) {  
 for (RobotDestroyListener l : \_destroyListeners) {  
 RobotDestroyEvent e = new RobotDestroyEvent(this);  
 e.setWhereDestroyedCell(whereDestroyedCell);  
 e.setDestroyedRobot(this);  
 l.robotDestroyed(e);  
 }  
 }  
}

Виджет маленького робота

public class LittleRobotWidget extends RobotWidget {  
 public LittleRobotWidget(Robot robot) {  
 super(robot);  
 setFocusable(true);  
 requestFocusInWindow();  
 addKeyListener(new KeyController());  
 }  
  
 @Override  
 protected File getImageFile() {  
 return new File(ImageUtils.*IMAGE\_PATH* + "LittleRobot.png");  
 }  
  
 @Override  
 protected BufferedImage getImage() {  
 BufferedImage image = null;  
 try {  
 image = ImageIO.*read*(getImageFile());  
 image = ImageUtils.*resizeImage*(image, 70, 60);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return image;  
 }  
  
 @Override  
 protected Dimension getDimension() {  
 return new Dimension(70, 120);  
 }  
  
 private class KeyController implements KeyListener {  
  
 @Override  
 public void keyTyped(KeyEvent arg0) {  
 }  
  
 @Override  
 public void keyPressed(KeyEvent ke) {  
 int keyCode = ke.getKeyCode();  
  
 moveAction(keyCode);  
 }  
  
 @Override  
 public void keyReleased(KeyEvent arg0) {  
 }  
  
 private void moveAction(int keyCode){  
 Direction direction = directionByKeyCode(keyCode);  
 if(direction != null) {  
 \_robot.move(direction);  
 }  
 }  
  
 private Direction directionByKeyCode(int keyCode) {  
 Direction direction = switch (keyCode) {  
 case KeyEvent.*VK\_W* -> Direction.*NORTH*;  
 case KeyEvent.*VK\_S* -> Direction.*SOUTH*;  
 case KeyEvent.*VK\_A* -> Direction.*WEST*;  
 case KeyEvent.*VK\_D* -> Direction.*EAST*;  
 default -> null;  
 };  
 return direction;  
 }  
 }  
}

Класс большого робота

public class BigRobot extends Robot implements RobotMoveListener {  
 private PathFinder \_pathFinder;  
  
 public void setPathFinder(PathFinder pathFinder) {  
 if (pathFinder == null) return;  
 if (\_pathFinder == pathFinder) return;  
 dropPathFinder();  
 \_pathFinder = pathFinder;  
 \_pathFinder.setRobot(this);  
 }  
  
 public PathFinder getPathFinder() {  
 return \_pathFinder;  
 }  
  
 public void dropPathFinder() {  
 var pathFinder = \_pathFinder;  
 \_pathFinder = null;  
 if (pathFinder != null) pathFinder.forgetRobot();  
 }  
  
 // ----------------------------------------------- Цель ------------------------------------------------------------  
 private Destroyable \_target = null;  
  
 public void setTarget(Destroyable target) {  
 \_target = target;  
 }  
  
 private void catchTarget() {  
 if (\_target != null) {  
 \_target.destroy();  
 }  
 }  
  
 // ----------------------------------------------- Перемещение -----------------------------------------------------  
 private void makeStep() {  
 if (\_characteristic != null && \_characteristic instanceof ViscosityCharacteristic && \_numOfStepsWithCharacteristic > 0  
 || \_target == null) {  
 if (\_numOfStepsWithCharacteristic > 0) {  
 --\_numOfStepsWithCharacteristic;  
 }  
 if (\_numOfStepsWithCharacteristic == 0) {  
 \_characteristic = null;  
 }  
 return;  
 }  
 Direction directionToMove = calculateStepDirection(((LittleRobot) \_target).getPosition());  
 if (!move(directionToMove) && directionToMove != null  
 && \_position.getNeighbourCell(directionToMove) == ((LittleRobot) \_target).getPosition()) {  
 catchTarget();  
 }  
 }  
  
 private Direction calculateStepDirection(Cell targetPosition) {  
 if (targetPosition == null) return null;  
 Direction directionToMove = null;  
 MyPoint distanceToTarget = \_position.getDistanceFromCellToCell(targetPosition);  
 if (distanceToTarget.getX() <= 2 && distanceToTarget.getY() <= 2) {  
 var pathToMove = \_pathFinder.findPath(getPosition(), targetPosition);  
 if (!pathToMove.isEmpty()) {  
 pathToMove.pop();  
 directionToMove = \_position.getNeighbourDirection(pathToMove.pop());  
 }  
 } else {  
 Direction calculatedDirectionToMove = \_position.getDirectionsToMove(targetPosition).pop();  
 if (calculatedDirectionToMove == Direction.*EAST* || calculatedDirectionToMove == Direction.*WEST*) {  
 directionToMove = calculatedDirectionToMove;  
 }  
 }  
 return directionToMove;  
 }  
  
 // ----------------------------------------------- Работа с ландшафтом ---------------------------------------------  
 private LandscapeCharacteristic \_characteristic = null;  
 private int \_numOfStepsWithCharacteristic = 0;  
  
 @Override  
 protected void processIfLandscapeSegment() {  
 if (\_position.getLandscapeSegment() != null && \_position.getLandscapeSegment() instanceof SwampSegment) {  
 setLandscapeCharacteristic(\_position.getLandscapeSegment().getCharacteristic());  
 }  
 }  
  
 private void setLandscapeCharacteristic(LandscapeCharacteristic characteristic) {  
 \_characteristic = characteristic;  
 \_numOfStepsWithCharacteristic = (int) Math.*floor*(\_characteristic.getCoefficient());  
 }  
  
 // ----------------------------------------------- Наблюдение за перемещением маленького робота --------------------  
 @Override  
 public void robotMadeMove(RobotMoveEvent e) {  
 makeStep();  
 }  
}

Класс-помощник для поиска пути

public class PathFinder {  
 public PathFinder(Field field) {  
 \_field = field;  
 }  
  
 private final Field \_field;  
  
 private BigRobot \_robot;  
  
 public void setRobot(BigRobot robot) {  
 if (\_robot == robot) return;  
 forgetRobot();  
 \_robot = robot;  
 \_robot.setPathFinder(this);  
 }  
  
 public BigRobot getRobot() {  
 return \_robot;  
 }  
  
 public void forgetRobot() {  
 var robot = \_robot;  
 \_robot = null;  
 if (robot != null) robot.dropPathFinder();  
 }  
  
 public Stack<Cell> findPath(Cell fromCell, Cell toCell) {  
 if (\_field == null || !\_field.contains(fromCell) || !\_field.contains(toCell)) return null;  
 int indexFromCell = \_field.getIndexOfCell(fromCell);  
 int indexToCell = \_field.getIndexOfCell(toCell);  
  
 var markedCells = WavePropagation(indexFromCell, indexToCell);  
  
 return PathRecovery(indexFromCell, indexToCell, markedCells);  
 }  
  
 private int[] WavePropagation(int fromCellIndex, int toCellIndex) { // распространение волны  
 int[] markedCells = new int[\_field.getCells().size()]; // массив, где будут хранится "отметки" каждого узла  
 int markNumber = 1; // счетчик  
 markedCells[fromCellIndex] = markNumber; // инициализация стартового узла  
 while (markedCells[toCellIndex] == 0) { // пока не достигли финишного узла  
 var cpyMarkedCells = markedCells.clone();  
 for (int i = 0; i < markedCells.length; i++) {  
 if (markedCells[i] == markNumber) { // начинаем со стартового узла  
 var neighbours = getAvailableNeighbours(\_field.getCellByIndex(i));  
 for (Cell neighbour : neighbours) { // просматриваем все соседние узлы  
 int cellIndex = \_field.getIndexOfCell(neighbour);  
 if (markedCells[cellIndex] == 0) { // если он еще не получил "отметку"  
 markedCells[cellIndex] = (markNumber + 1);  
 }  
 }  
 }  
 }  
 markNumber++;  
 if (compareArrays(markedCells, cpyMarkedCells)) return new int[0];  
 }  
 return markedCells;  
 }  
  
 private boolean compareArrays(int[] first, int[] second) {  
 if (first.length != second.length) return false;  
  
 for (int i = 0; i < first.length; ++i) {  
 if (first[i] != second[i]) return false;  
 }  
  
 return true;  
 }  
  
 private Stack<Cell> PathRecovery(int fromCellIndex, int toCellIndex, int[] markedCells) { // восстановление пути  
 Stack<Cell> paramsPaveTheRoute = new Stack<>(); // массив, где хранится путь  
 if (markedCells.length == 0) return paramsPaveTheRoute;  
 if (markedCells[toCellIndex] != 0) { // еще раз проверяем дошел ли алгоритм до финишного узла  
 Cell currentNode = \_field.getCellByIndex(toCellIndex);  
 paramsPaveTheRoute.add(currentNode); // добавляем финишный узел к пути  
 int currentCellIndex = \_field.getIndexOfCell(currentNode);  
 while (currentCellIndex != fromCellIndex) { // пока не дошли до стартового узла  
 var neighbours = getAvailableNeighbours(currentNode);  
 for (Cell neighbour : neighbours) { // проверяем соседние узлы  
 if (markedCells[\_field.getIndexOfCell(neighbour)]  
 == markedCells[currentCellIndex] - 1) { // если значение пометки узла на 1 меньше, чем у предыдущего узла  
 currentNode = neighbour; //узел становится текущим  
 currentCellIndex = \_field.getIndexOfCell(currentNode);  
 paramsPaveTheRoute.push(currentNode); // заносится в массив  
 break;  
 }  
 }  
 }  
 }  
 return paramsPaveTheRoute;  
 }  
  
 private List<Cell> getAvailableNeighbours(Cell cell) {  
 List<Cell> neighbours = new ArrayList<>();  
  
 for (var cellEntry : cell.getNeighbourCells().entrySet()) {  
 if (moveIsPossible(cell, cellEntry.getKey())) {  
 neighbours.add(cellEntry.getValue());  
 }  
 }  
  
 return neighbours;  
 }  
  
 private boolean moveIsPossible(Cell position, Direction direction){  
 return position.getNeighbourCell(direction) != null && position.getWallSegment(direction) == null;  
 }  
}

Виджет большого робота

public class BigRobotWidget extends RobotWidget {  
 public BigRobotWidget(Robot robot) {  
 super(robot);  
 }  
  
 @Override  
 protected File getImageFile() {  
 return new File(ImageUtils.*IMAGE\_PATH* + "BigRobot.png");  
 }  
  
 @Override  
 protected BufferedImage getImage() {  
 BufferedImage image = null;  
 try {  
 image = ImageIO.*read*(getImageFile());  
 image = ImageUtils.*resizeImage*(image, 100, 96);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return image;  
 }  
  
 @Override  
 protected Dimension getDimension() {  
 return new Dimension(100, 120);  
 }  
}

Класс клетки

public class Cell {  
 public Cell(MyPoint coordinates) {  
 \_coordinates = coordinates;  
 }  
  
 // ----------------------------------------------- Координаты ------------------------------------------------------  
 private final MyPoint \_coordinates;  
  
 // ----------------------------------------------- Соседние клетки -------------------------------------------------  
 private final TreeMap<Direction, Cell> \_neighbourCells = new TreeMap<>();  
  
 public TreeMap<Direction, Cell> getNeighbourCells() {  
 return \_neighbourCells;  
 }  
  
 public boolean setNeighbourCell(Cell cell, final Direction direction) {  
 if (cell == null || direction == null) return false;  
 if (\_neighbourCells.get(direction) == null) {  
 \_neighbourCells.put(direction, cell);  
 return cell.setNeighbourCell(this, direction.getOppositeDirection());  
 }  
 return cell.\_neighbourCells.get(direction.getOppositeDirection()) == this;  
 }  
  
 public Cell getNeighbourCell(final Direction direction) {  
 if (direction == null) return null;  
 return \_neighbourCells.get(direction);  
 }  
  
 public Direction getNeighbourDirection(final Cell cell) {  
 if (cell == null) return null;  
 Direction direction = null;  
 for (var neighbourCell : \_neighbourCells.entrySet()) {  
 if (neighbourCell.getValue() == cell) {  
 direction = neighbourCell.getKey();  
 }  
 }  
 return direction;  
 }  
  
 // ----------------------------------------------- Работа с клетками -----------------------------------------------  
 public MyPoint getDistanceFromCellToCell(final Cell otherCell) {  
 if (otherCell == null) return null;  
 return \_coordinates.getDistance(otherCell.\_coordinates);  
 }  
  
 public Stack<Direction> getDirectionsToMove(final Cell otherCell) {  
 if (otherCell == null) return null;  
 Stack<Direction> directions = new Stack<>();  
 var difference = \_coordinates.getDifference(otherCell.\_coordinates);  
  
 if (difference.getY() > 0) directions.push(Direction.*NORTH*);  
 else if (difference.getY() < 0) directions.push(Direction.*SOUTH*);  
  
 if (difference.getX() > 0) directions.push(Direction.*EAST*);  
 else if (difference.getX() < 0) directions.push(Direction.*WEST*);  
  
 return directions;  
 }  
  
 // ----------------------------------------------- Робот -----------------------------------------------------------  
 private Robot \_robot = null;  
  
 public boolean setRobot(Robot robot) {  
 if (robot == null) return false;  
 if (\_robot == robot) return true;  
 if (!robot.canBeLocatedAtPosition(this)) return false;  
 \_robot = robot;  
 return \_robot.setPosition(this);  
 }  
  
 public Robot getRobot() {  
 return \_robot;  
 }  
  
 public Robot takeRobot() {  
 \_robot.setNullPosition();  
 Robot temp = \_robot;  
 \_robot = null;  
 return temp;  
 }  
  
 // ----------------------------------------------- Стены -----------------------------------------------------------  
 private final TreeMap<Direction, WallSegment> \_wallSegments = new TreeMap<>();  
  
 public WallSegment getWallSegment(final Direction direction) {  
 if (direction == null) return null;  
 return \_wallSegments.get(direction);  
 }  
  
 public boolean setWallSegment(WallSegment wallSegment, final Direction direction) {  
 if (wallSegment == null || direction == null) return false;  
 if (\_wallSegments.get(direction) != null) return \_wallSegments.get(direction) == wallSegment;  
 \_wallSegments.put(direction, wallSegment);  
 if (\_neighbourCells.get(direction) != null) {  
 \_neighbourCells.get(direction).setWallSegment(wallSegment, direction.getOppositeDirection());  
 return wallSegment.setPosition(new BetweenCellPosition(this, \_neighbourCells.get(direction)));  
 }  
 return wallSegment.setPosition(new BetweenCellPosition(this, direction));  
 }  
  
 // ----------------------------------------------- Ландшафт --------------------------------------------------------  
 private LandscapeSegment \_landscapeSegment = null;  
  
 public LandscapeSegment getLandscapeSegment() {  
 return \_landscapeSegment;  
 }  
  
 public boolean setLandscapeSegment(final LandscapeSegment landscapeSegment) {  
 if (landscapeSegment == null) return false;  
 if (landscapeSegment.setPosition(this)) {  
 \_landscapeSegment = landscapeSegment;  
 return true;  
 }  
  
 return false;  
 }  
}

Виджет клетки

public class CellWidget extends JPanel {  
  
 public CellWidget() {  
 setPreferredSize(new Dimension(*CELL\_SIZE*, *CELL\_SIZE*));  
 setBackground(ImageUtils.*BACKGROUND\_COLOR*);  
 }  
  
 private static final int *CELL\_SIZE* = 120;  
  
 private enum Layer {  
 *TOP*,  
 *BOTTOM* }  
  
 private final HashMap<Layer, CellItemWidget> \_widgetItems = new HashMap<>();  
  
 public void addItem(CellItemWidget widgetItem) {  
 if (\_widgetItems.size() > 2)  
 throw new IllegalArgumentException("Попытка установить в ячейку более двух виджетов");  
  
 int index = -1;  
 Layer layer = Layer.*BOTTOM*;  
  
 if (widgetItem instanceof RobotWidget) {  
 index = 0;  
 layer = Layer.*TOP*;  
 }  
  
 \_widgetItems.put(layer, widgetItem);  
 add(widgetItem, index);  
 }  
  
 public void removeItem(CellItemWidget widgetItem) {  
 for (var item : \_widgetItems.entrySet()) {  
 if (widgetItem == item.getValue()) {  
 remove(item.getKey() == Layer.*TOP* ? 0 : -1);  
 \_widgetItems.remove(item.getKey());  
 repaint();  
 }  
 }  
 }  
}

Класс точки выхода

public class ExitCell extends Cell implements RobotMoveListener {  
 public ExitCell(MyPoint coordinates) {  
 super(coordinates);  
 }  
  
 private void teleportRobot() {  
 getRobot().setNullPosition();  
 fireLittleRobotTeleport(takeRobot());  
 }  
  
 // ----------------------------------------------- Наблюдение за перемещением маленького робота --------------------  
 @Override  
 public void robotMadeMove(RobotMoveEvent e) {  
 if (getRobot() != null && getRobot() instanceof LittleRobot && getRobot() == e.getRobot()) {  
 teleportRobot();  
 }  
 }  
  
 // ----------------------------------------------- Порождение события ----------------------------------------------  
 private final List<ExitCellListener> \_listeners = new ArrayList<>();  
  
 public void addExitCellListener(ExitCellListener l) {  
 \_listeners.add(l);  
 }  
  
 public void addExitCellListener(int index, ExitCellListener l) {  
 \_listeners.add(index, l);  
 }  
  
 public void removeExitCellListener(ExitCellListener l) {  
 \_listeners.remove(l);  
 }  
  
 protected void fireLittleRobotTeleport(Robot robot) {  
 for (ExitCellListener l : \_listeners) {  
 RobotTeleportEvent e = new RobotTeleportEvent(this);  
 e.setTeleportedRobot(robot);  
 e.setTeleportCell(this);  
 l.robotTeleported(e);  
 }  
 }  
}

Виджет точки выхода

public class ExitCellWidget extends CellWidget {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
  
 try {  
 BufferedImage image = ImageIO.*read*(new File(ImageUtils.*IMAGE\_PATH* + "ExitCell.png"));  
 image = ImageUtils.*resizeImage*(image, 100, 80);  
 g.drawImage(image, 0, 0, null);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

Класс сегмента стены

public class WallSegment {  
 private BetweenCellPosition \_position = null;  
  
 public boolean setPosition(BetweenCellPosition betweenCellPosition) {  
 if (betweenCellPosition == null) return false;  
 if (\_position == betweenCellPosition) return true;  
 var entryCell = betweenCellPosition.getNeighbourCells().firstEntry();  
 if (!*canCreateAtPosition*(betweenCellPosition)  
 && entryCell.getValue().getWallSegment(entryCell.getKey().getOppositeDirection()) != this) return false;  
 \_position = betweenCellPosition;  
 for (var neighbourCell : \_position.getNeighbourCells().entrySet()) {  
 neighbourCell.getValue().setWallSegment(this, neighbourCell.getKey().getOppositeDirection());  
 }  
 return true;  
 }  
  
 public BetweenCellPosition getPosition() {  
 return \_position;  
 }  
  
 public static boolean canCreateAtPosition(BetweenCellPosition position) {  
 if (position == null) return false;  
 for (var cell : position.getNeighbourCells().entrySet()) {  
 if (cell.getValue().getWallSegment(cell.getKey().getOppositeDirection()) != null) return false;  
 }  
 return true;  
 }  
}

Виджет стены

public class WallSegmentWidget extends JPanel {  
 public WallSegmentWidget(Orientation orientation) {  
 \_orientation = orientation;  
 setPreferredSize(getDimensionByOrientation());  
 }  
  
 private static File *\_verticalWallFile* = null;  
 private static File *\_horizontalWallFile* = null;  
  
 private final Orientation \_orientation;  
  
 public Orientation getOrientation() {  
 return \_orientation;  
 }  
  
 private BufferedImage getImage() {  
 BufferedImage image = null;  
 try {  
 image = ImageIO.*read*(getImageFile());  
 Dimension dimension = getDimensionByOrientation();  
 image = ImageUtils.*resizeImage*(image, dimension.width, dimension.height);  
 } catch (IOException e) {  
 e.printStackTrace(); // !!! Для конечного пользователя это не лучшее решение (более дружественное сообщение для пользователя) *TODO* }  
 return image;  
 }  
  
 protected File getImageFile() {  
 switch(\_orientation) {  
 case *VERTICAL* -> {if (*\_verticalWallFile* == null) *\_verticalWallFile* = new File(ImageUtils.*IMAGE\_PATH* + "VerticalWall.png");}  
 case *HORIZONTAL* -> {if (*\_horizontalWallFile* == null) *\_horizontalWallFile* = new File(ImageUtils.*IMAGE\_PATH* + "HorizontalWall.png");}  
 }  
 return (\_orientation == Orientation.*VERTICAL*) ? *\_verticalWallFile* : *\_horizontalWallFile*;  
 }  
  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 g.drawImage(getImage(), 0, 0, null);  
 }  
  
 private Dimension getDimensionByOrientation() {  
 return (\_orientation == Orientation.*VERTICAL*) ? new Dimension(5, 120) : new Dimension(125, 5);  
 }  
}

События

public class GameStatusChangeEvent extends EventObject {  
 public GameStatusChangeEvent(Object source) {  
 super(source);  
 }  
  
 private GameStatus \_gameStatus = null;  
  
 public void setGameStatus(GameStatus gameStatus) {  
 \_gameStatus = gameStatus;  
 }  
  
 public GameStatus getGameStatus() {  
 return \_gameStatus;  
 }  
}

public class RobotMoveEvent extends EventObject {  
 public RobotMoveEvent(Object source) {  
 super(source);  
 }  
  
 // Робот, порождающий событие  
 private Robot \_robot;  
  
 public void setRobot(Robot robot) {  
 \_robot = robot;  
 }  
  
 public Robot getRobot() {  
 return \_robot;  
 }  
  
 // Ячейка, из которой робот перемещается  
 private Cell \_fromCell;  
  
 public void setFromCell(Cell fromCell) {  
 \_fromCell = fromCell;  
 }  
  
 public Cell getFromCell() {  
 return \_fromCell;  
 }  
  
 // Ячейка, в которую робот перемещается  
 private Cell \_toCell;  
  
 public void setToCell(Cell toCell) {  
 \_toCell = toCell;  
 }  
  
 public Cell getToCell() {  
 return \_toCell;  
 }  
}

public class RobotTeleportEvent extends EventObject {  
  
 public RobotTeleportEvent(Object source) {  
 super(source);  
 }  
  
 private Robot \_teleportedRobot;  
  
 public void setTeleportedRobot(Robot teleportedRobot) {  
 \_teleportedRobot = teleportedRobot;  
 }  
  
 public Robot getTeleportedRobot() {  
 return \_teleportedRobot;  
 }  
  
 private ExitCell \_teleportCell;  
  
 public void setTeleportCell(ExitCell teleportCell) {  
 \_teleportCell = teleportCell;  
 }  
  
 public ExitCell getTeleportCell() {  
 return \_teleportCell;  
 }  
}

public class RobotDestroyEvent extends EventObject {  
  
 public RobotDestroyEvent(Object source) {  
 super(source);  
 }  
  
 private Robot \_destroyedRobot;  
  
 public Robot getDestroyedRobot() {  
 return \_destroyedRobot;  
 }  
  
 public void setDestroyedRobot(Robot destroyedRobot) {  
 \_destroyedRobot = destroyedRobot;  
 }  
  
 private Cell \_whereDestroyedCell;  
  
 public Cell getCellWhereDestroyed() {  
 return \_whereDestroyedCell;  
 }  
  
 public void setWhereDestroyedCell(Cell whereDestroyedCell) {  
 \_whereDestroyedCell = whereDestroyedCell;  
 }  
}

## 3.8 Реализация ключевых тестовых случаев

public class GameTests {  
 @Test  
 public void littleRobotTeleportedGameTest() {  
 Game game = new Game();  
 game.initGame(new SimpleLabyrinth());  
  
 Assertions.*assertEquals*(GameStatus.*GAME\_IS\_ON*, game.getGameStatus());  
  
 LittleRobot littleRobot = null;  
 for (var robot : game.getRobotsOnField()) {  
 if (robot instanceof LittleRobot) {  
 littleRobot = (LittleRobot)robot;  
 }  
 }  
  
 Assertions.*assertTrue*(littleRobot.move(Direction.*NORTH*));  
 Assertions.*assertTrue*(littleRobot.move(Direction.*NORTH*));  
 Assertions.*assertTrue*(littleRobot.move(Direction.*NORTH*));  
  
 Assertions.*assertEquals*(GameStatus.*WINNER\_IS\_LITTLE\_ROBOT*, game.getGameStatus());  
 }  
  
 @Test  
 public void littleRobotDestroyedBySwampGameTest() {  
 Game game = new Game();  
 game.initGame(new SimpleLabyrinth());  
  
 Assertions.*assertEquals*(GameStatus.*GAME\_IS\_ON*, game.getGameStatus());  
  
 LittleRobot littleRobot = null;  
 for (var robot : game.getRobotsOnField()) {  
 if (robot instanceof LittleRobot) {  
 littleRobot = (LittleRobot)robot;  
 }  
 }  
  
 Assertions.*assertTrue*(littleRobot.move(Direction.*EAST*));  
  
 Assertions.*assertEquals*(GameStatus.*WINNER\_IS\_BIG\_ROBOT*, game.getGameStatus());  
 }

# Вторая итерация разработки

## 4.1 Функциональные требования (сценарии)

**1) Сценарий «Играть за маленького робота: выйти из лабиринта, не попавшись большому роботу»**

1. Пользователь инициирует начало игры.
2. Игра создаёт поле из ячеек и размещает на нем лабиринт из стен, болото(а), пустыни (песок), точку выхода и двух роботов: большого и маленького – а также устанавливает полю погоду .
3. Игра делает активным маленького робота
4. Делать
   1. Робот перемещается на соседнюю ячейку.
   2. Игра делает активным следующего робота, который располагается на поле.
   3. После хода маленького робота происходит инкрементация времени года и изменение погодных условий.

Пока маленький робот находится внутри поля.

1. Игра считает победителем игрока, если он “телепортирован” точкой выхода или компьютер, если маленький робот был уничтожен.

**2) Сценарий «игра создаёт поле из ячеек и размещает на нем лабиринт из стен, болото(а), точку выхода и двух роботов»**

1. Игра инициирует создание поля размером W на H ячеек.
2. Игра посредством лабиринта создает и расставляет последовательности стен по периметру поля.
3. Игра посредством лабиринта создает последовательности стен в середине поля.
4. Игра посредством лабиринта создает болото(а) и пустыни случайным образом случайной формы.
5. Игра создаёт двух роботов.
6. Игра помещает на поле созданных роботов.
7. Игра создает и помещает на поле ячейку выхода.

**4) Сценарий «робот перемещается на соседнюю ячейку».**

1. Робот запрашивает у ячейки, в которой он находится, стену в направлении своего движения
2. Ячейка сообщает, что стены нет
3. Робот запрашивает у ячейки, в которой он находится, соседнюю ячейку в направлении своего движения
4. Ячейка сообщает о ячейке, с которой соседствует
5. Робот запрашивает у соседней ячейки робота (который возможно в ней находится)
6. Соседняя ячейка сообщает, что робот отсутствует в ней
7. Робот просит ячейку, в которой он находится, изъять его из нее
8. Ячейка извлекает робота из себя
9. Робот просит соседнюю ячейку поместить себя в нее
10. Ячейка помещает робота в себя, т.к. в ней нет другого робота
11. Робот определяет, находится ли в новой ячейке болото или песок

**4.1) Альтернативный сценарий «ход маленького робота, пропуск хода»**

1. Сценарий начинается перед п. 1 альтернативного сценария «Робот перемещается на соседнюю ячейку»
2. Так как робот «оштрафован» за попадание в ячейку с песком или песком, он пропускает ход
3. Сценарий переходит к п. 4.2 Главного сценария

**4.2) Альтернативный сценарий «ход большого робота»**

1. Сценарий начинается перед п. 1 Главного сценария 4
2. Большой робот вычисляет расстояние до маленького робота
3. Так как маленький робот находится в поле действия большого робота, большой робот составляет оптимальный маршрут до маленького робота
4. Большой робот выбирает направление движения по направлению к следующей клетке в построенном оптимальном пути
5. Сценарий переходит к п. 1 сценария 4

**4.2.1) Альтернативный сценарий «ход большого робота, пропуск хода»**

1. Сценарий начинается перед п. 1 альтернативного сценария «ход большого робота»
2. Так как робот все еще «оштрафован» за попадание в ячейку с болотом или песком, он пропускает ход
3. Сценарий переходит к п. 1

**4.2.2) Альтернативный сценарий «ход большого робота, маленький робот вне поле действия большого робота»**

1. Сценарий начинается после п. 2 альтернативного сценария «ход большого робота»
2. Так как маленький робот находится вне поле действия большого робота, большой робот выбирает направление движения в сторону маленького робота из направлений “влево” и “вправо” или “пустое направление”
3. Сценарий переходит к п. 1 Главного сценария 4

**4.3) Альтернативный сценарий «в соседней ячейке находится робот»**

1. Сценарий начинается после п. 5 сценария 4
2. Соседняя ячейка сообщает, что в ней находится робот
3. Сценарий переходит к п 5.2. Главного сценария.

**4.3.1) Альтернативный сценарий «в соседней ячейке находится робот, ход большого робота»**

1. Сценарий начинается после п. 2 сценария 4.1
2. Большой робот инициирует уничтожение маленького робота
3. Сценарий переходит к п 5.1. Главного сценария

**4.4) Альтернативный сценарий «между соседней ячейкой и текущей находится стена»**

1. Сценарий начинается после п. 1 сценария 4
2. Ячейка сообщает, что стена имеется
3. Сценарий переходит к п. 5.2. Главного сценария

**4.5) Альтернативный сценарий «конец хода большого робота»**

1. Сценарий начинается после п. 4.1 Главного сценария
2. Большой робот анализирует расстояние до маленького робота
3. Сценарий переходит к п. 4.2 Главного сценария

**4.5.1) Альтернативный сценарий** **«конец хода большого робота, маленький робот в соседней клетке»**

1. Сценарий начинается после п. 2 альтернативного сценария «конец хода большого робота»
2. Так как маленький робот находится в соседней клетке, большой робот инициирует уничтожение маленького робота
3. Большой робот сообщает игре о том, что маленький робот уничтожен
4. Сценарий переходит к п. 5 Главного сценария

**4.6) Альтернативный сценарий** **«маленький робот попадает в болото»**

1. Сценарий начинается после п. 11 сценария «робот перемещается на соседнюю ячейку»
2. Так как маленький робот находится в ячейке, в которой есть болото, он инициирует свое уничтожение
3. Маленький робот сообщает игре о том, что он уничтожен
4. Сценарий переходит к п. 5 Главного сценария

**4.7) Альтернативный сценарий** **«робот попадает в песок»**

1. Сценарий начинается после п. 11 сценария «робот перемещается на соседнюю ячейку»
2. Так как робот находится в ячейке, в которой есть болото, он пропускает следующий ход
3. Сценарий переходит к п. 5 Главного сценария

**4.8) Альтернативный сценарий «робот попадает в замерзшее болото»**

1. Сценарий начинается после п. 11 сценария «робот перемещается на соседнюю ячейку»
2. Так как робот находится в ячейке, в которой есть замерзшее болото, робот передвигается дополнительно на одну ячейку по направлению, если нет препятствий
3. Сценарий переходит к п 4.2 Главного сценария

**4.8) Альтернативный сценарий «большой робот попадает в болото»**

1. Сценарий начинается после п. 11 сценария «робот перемещается на соседнюю ячейку»
2. Так как большой робот находится в ячейке, в которой есть болото, робот пропускает три следующих хода
3. Сценарий переходит к п 4.2 Главного сценария

**5) Сценарий «инкрементация текущего времени»**

1. Сценарий начинается с п. 4.3 Главного сценария
2. Погода инкрементирует текущее время
3. Сценарий переходит к п. 4.1 Главного сценария

**5.1) Альтернативный сценарий «смена времени года на зимнее»**

1. Сценарий начинается с п. 2 сценария «инкрементация текущего времени»
2. Т.к. закончился осенний сезон времени года, устанавливается новое время года – зимнее
3. Погода возвращает состояние размера болота в исходное
4. Погода замораживает болота
5. Сценарий переходит к п. 3 сценария «инкрементация текущего времени»

**5.2) Альтернативный сценарий «смена времени года на весеннее»**

1. Сценарий начинается с п. 2 сценария «инкрементация текущего времени»
2. Т.к. закончился зимний сезон, устанавливается весеннее время года
3. Погода меняет сегмент льда на сегмент болота
4. Погода меняет состояние размера болота на расширенное
5. Сценарий переходит к п. 3 сценария «инкрементация текущего времени»

**5.3) Альтернативный сценарий «смена времени года на летнее»**

1. Сценарий начинается с п. 2 сценария «инкрементация текущего времени»
2. Т.к. закончился весенний сезон, устанавливается летнее время года
3. Погода меняет состояние размера болота на исходное
4. Сценарий переходит к п. 3 сценария «инкрементация текущего времени»

**5.4) Альтернативный сценарий «смена времени года на осеннее»**

1. Сценарий начинается с п. 2 сценария «инкрементация текущего времени»
2. Т.к. закончился летний сезон, устанавливается осеннее время года
3. Погода меняет состояние размера болота на расширенное
4. Сценарий переходит к п. 3 сценария «инкрементация текущего времени»

**6) Сценарий «победа - маленький робот телепортировался»**

1. Поле сообщает игре, что “телепортирован“ робот
2. Игра считает победителем “телепортированного” робота, то есть игрока.

**6.1) Альтернативный сценарий «проигрыш - большой робот, управляемый компьютером, “поймал” маленького»**

1. Сценарий начинается с п. 2 сценария 5
2. Большой робот сообщает игре, что он “поймал” маленького робота
3. Сценарий переходит к п. 5 Главного сценария

**7) Сценарий «досрочное завершение игры»**

1. Сценарий начинается в любой точке главного сценария, когда пользователь инициирует завершение игры.
2. Игра завершается без определения победителя.

## 4.2 Словарь предметной области

**Игра** - игра знает о поле. Игра управляет игровым циклом: определяет очередного игрока, определяет окончание, определяет победителя. Игра определяет начальную расстановку игровых элементов (роботы, стены, выход, батарейки) на поле. Игра “следит” за маленьким роботом, большим роботом, точкой выхода.

**Поле** - прямоугольная область, состоящая из ячеек. Между ячейками возможно расположение стен. По границе поля могут располагаться стены. Позволяет получить роботов, находящихся на поле, и роботов, “телепортированных” с поля.

**Ячейка** - квадратная область поля. Знает о четырёх соседних ячейках и граничащих с ней стен. Одновременно может содержать в себе только одного робота и только одну батарейку.

**Стена** - непроходимое препятствие для робота, располагающееся между ячейками.

**Лабиринт** - умеет создавать горизонтальные или вертикальные последовательности стен.

**Погода** – абстрактный игровой объект, изменяющий состояние ландшафта (клеток с болотом) с течением времени. Зимой замораживает болото, весной и осенью – расширяет, летом – приводит к исходному размеру.

**Болото** - препятствие произвольного размера, располагающееся на ячейках поля. При попадании большого робота в болото, он пропускает три хода. Маленькому роботу заходить в болото нельзя. В летнее время года имеет исходный размер. В осеннее и весеннее – расширяется на одну клетку в стороны. В зимнее время года замерзает.

**Замерзшее болото** - препятствие произвольного размера, располагающееся на ячейках поля. При попадании робота в болото, он делает дополнительный шаг в заданном направлении, если ему не мешает препятствие (стена). Возникает в зимнее время года на месте болота.

**Песок** – препятствие произвольного размера, располагающееся на ячейках поля. При попадании робота в болото, он пропускает следующий ход.

**Точка выхода** - разновидность ячейки. На поле может быть только одна ячейка, являющаяся выходом. “Телепортирует” робота, находящегося на ней, с поля внутрь себя. Помнит всех “телепортированных роботов”.

**Робот** - умеет перемещаться в соседние ячейки. Не может переходить через стены.

**Маленький робот** - разновидность робота, управляемого игроком. Имеет все свойства робота. Может быть “телепортированным”, поместившись в “точку выхода”. Условие победы - достигнуть точки выхода.

**Большой робот** - разновидность робота, управляемого компьютером. Имеет все свойства робота. Действует рационально при приближенности к маленькому роботу не менее, чем на 2 клетки (идет оптимальным маршрутом), в противном случае передвигается только влево или вправо, в сторону маленького робота. Если находится вдалеке от маленького робота по оси Y, но на одной оси X, пропускает ход. Условие победы - уничтожить маленького робота.

**Активный робот** - робот, который может совершать действие в текущий ход.

**Игрок** - пользователь игры, управляющий маленьким роботом при помощи клавиатуры.

**Компьютер** - виртуальный оппонент игрока, управляющий большим роботом с заданным алгоритмом.

## 4.3 Структура программы на уровне классов

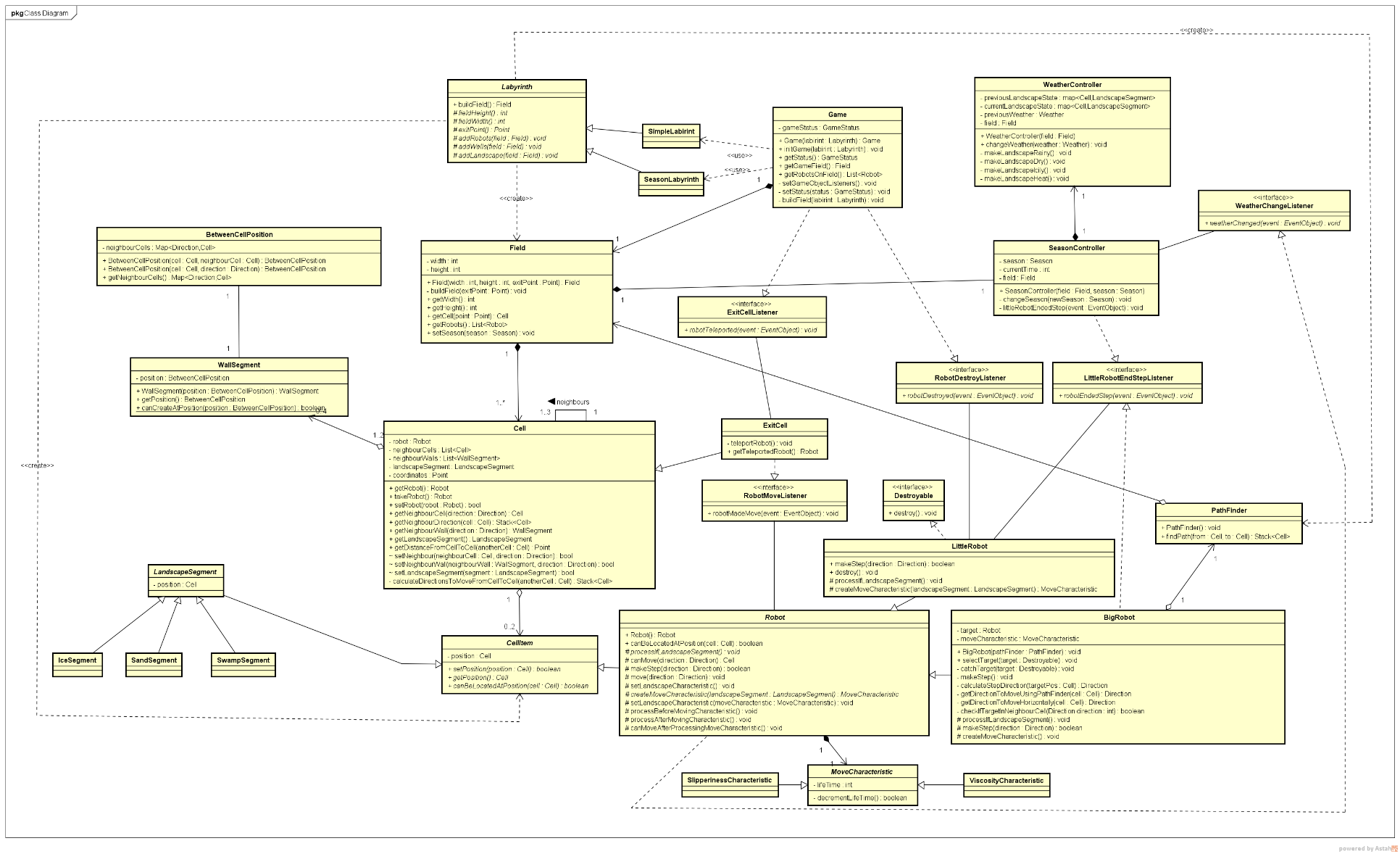


Диаграмма классов вычислительной модели

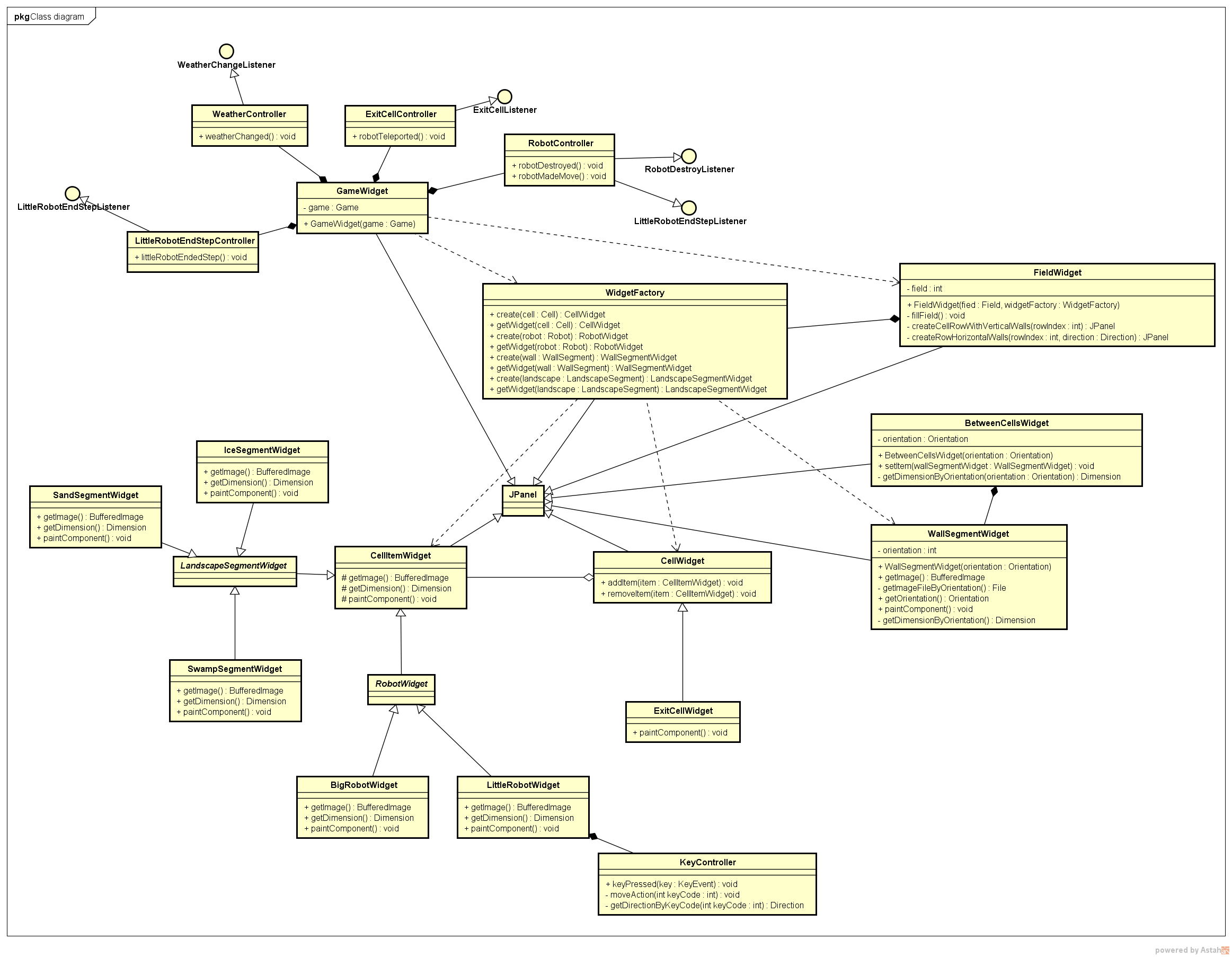
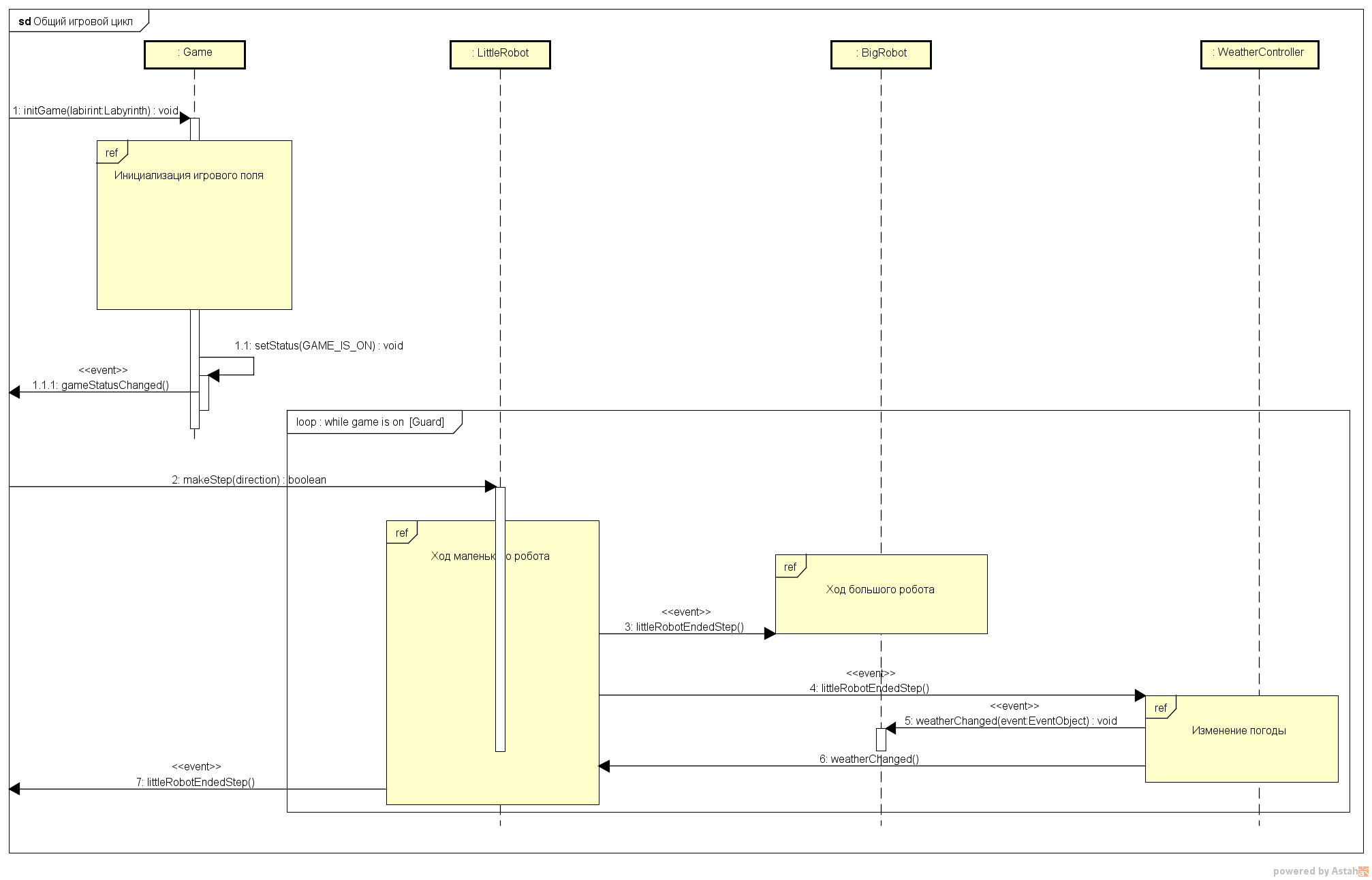
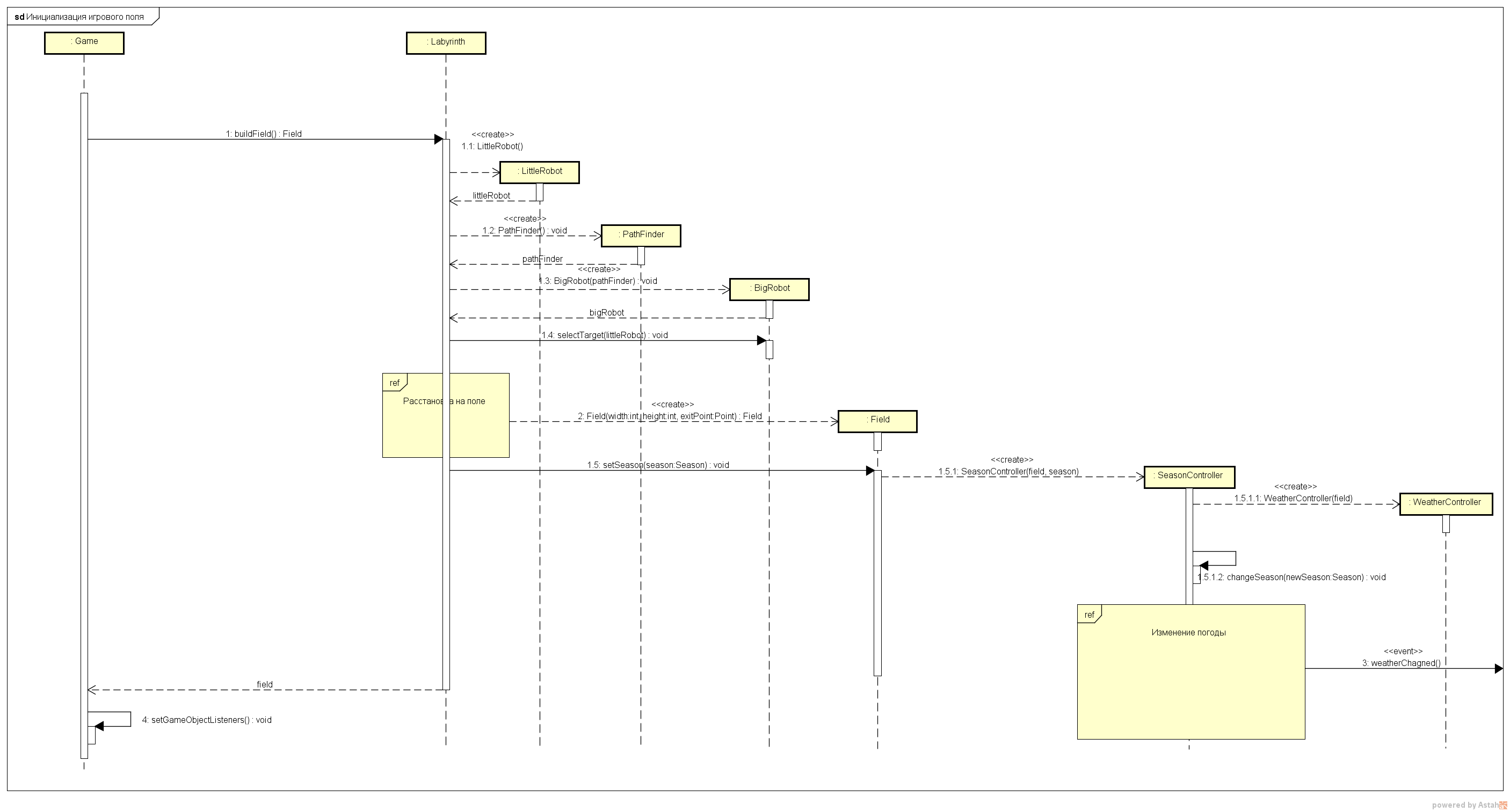


Диаграмма классов UI

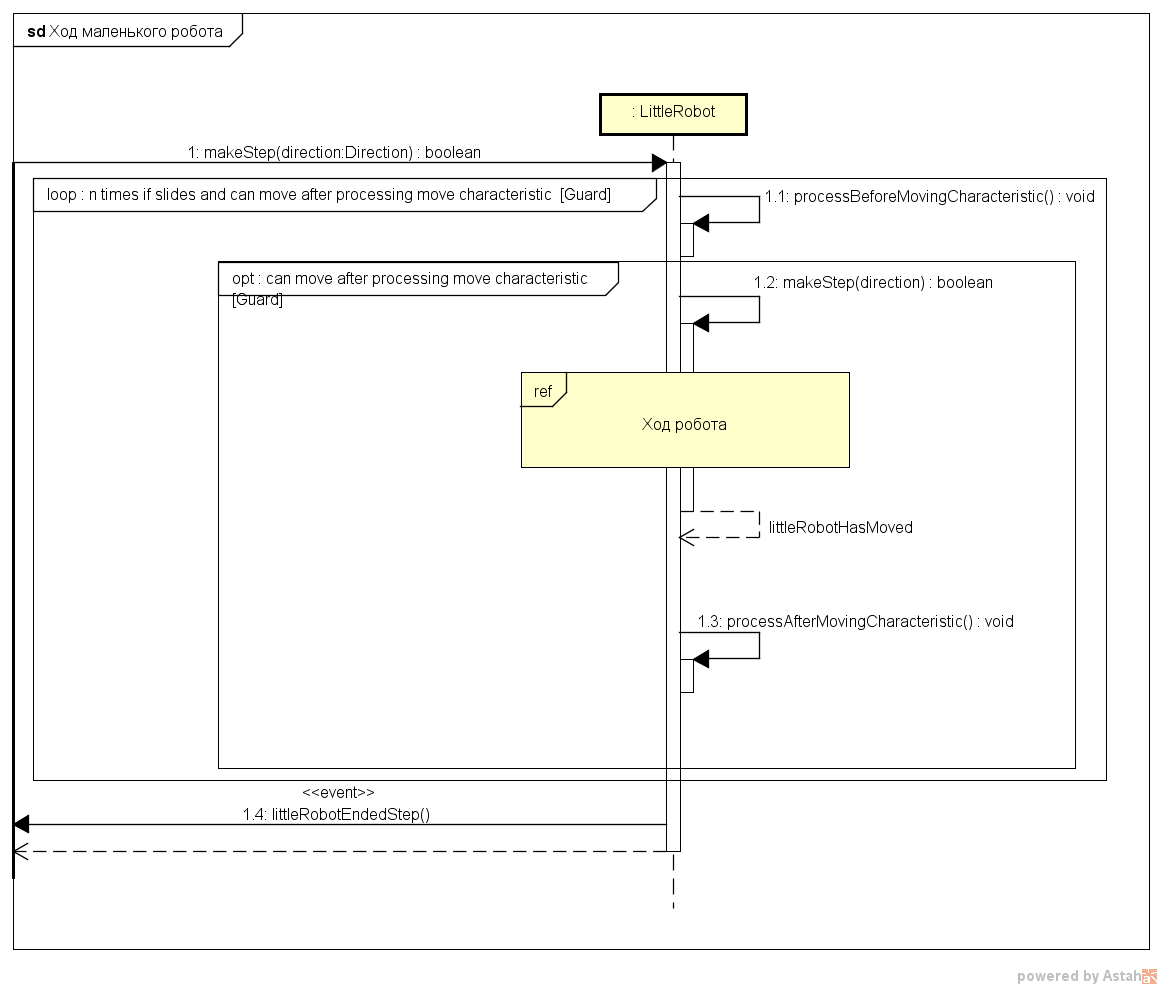
## 4.4 Типовые процессы в программе



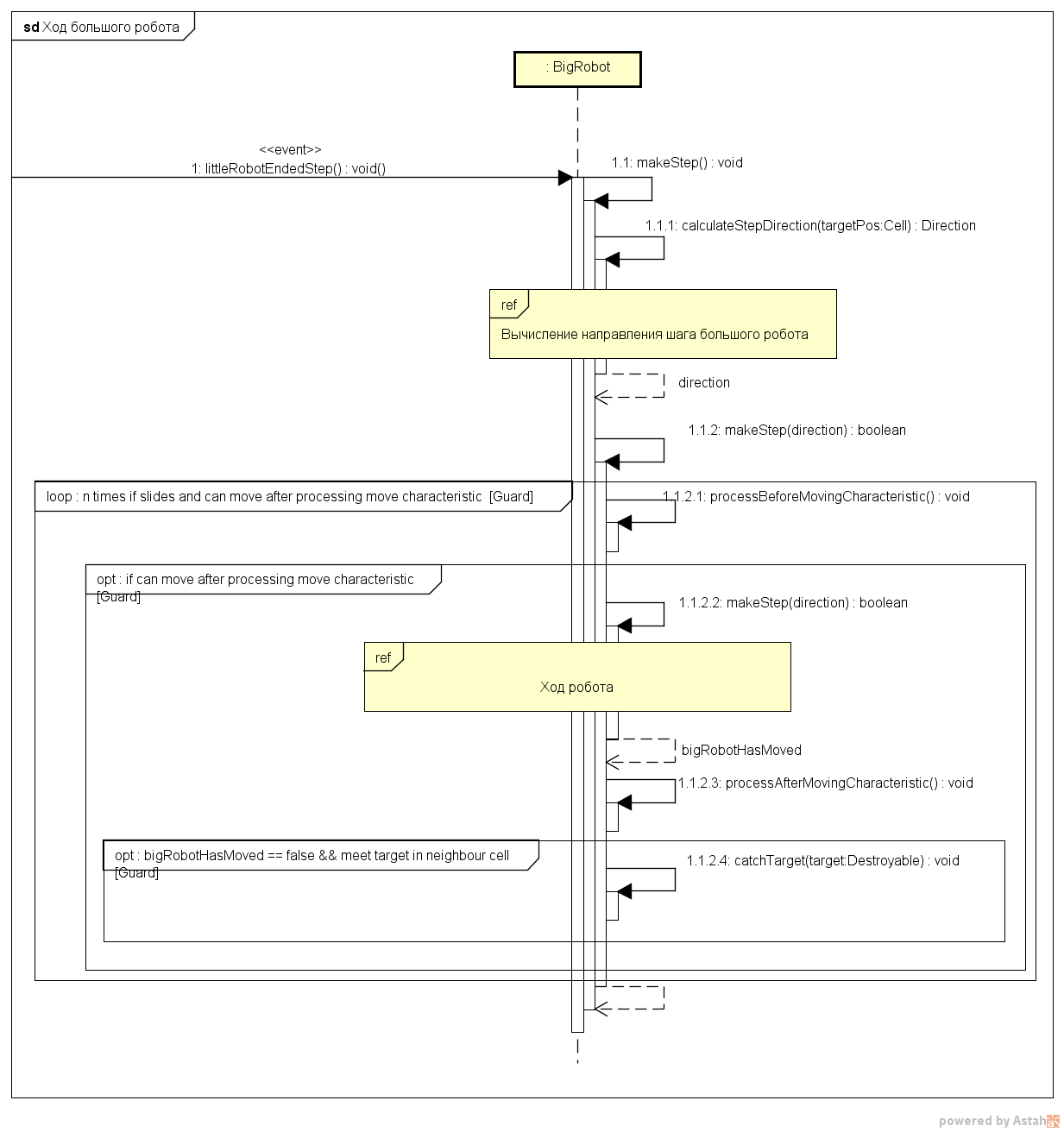
Общий игровой цикл



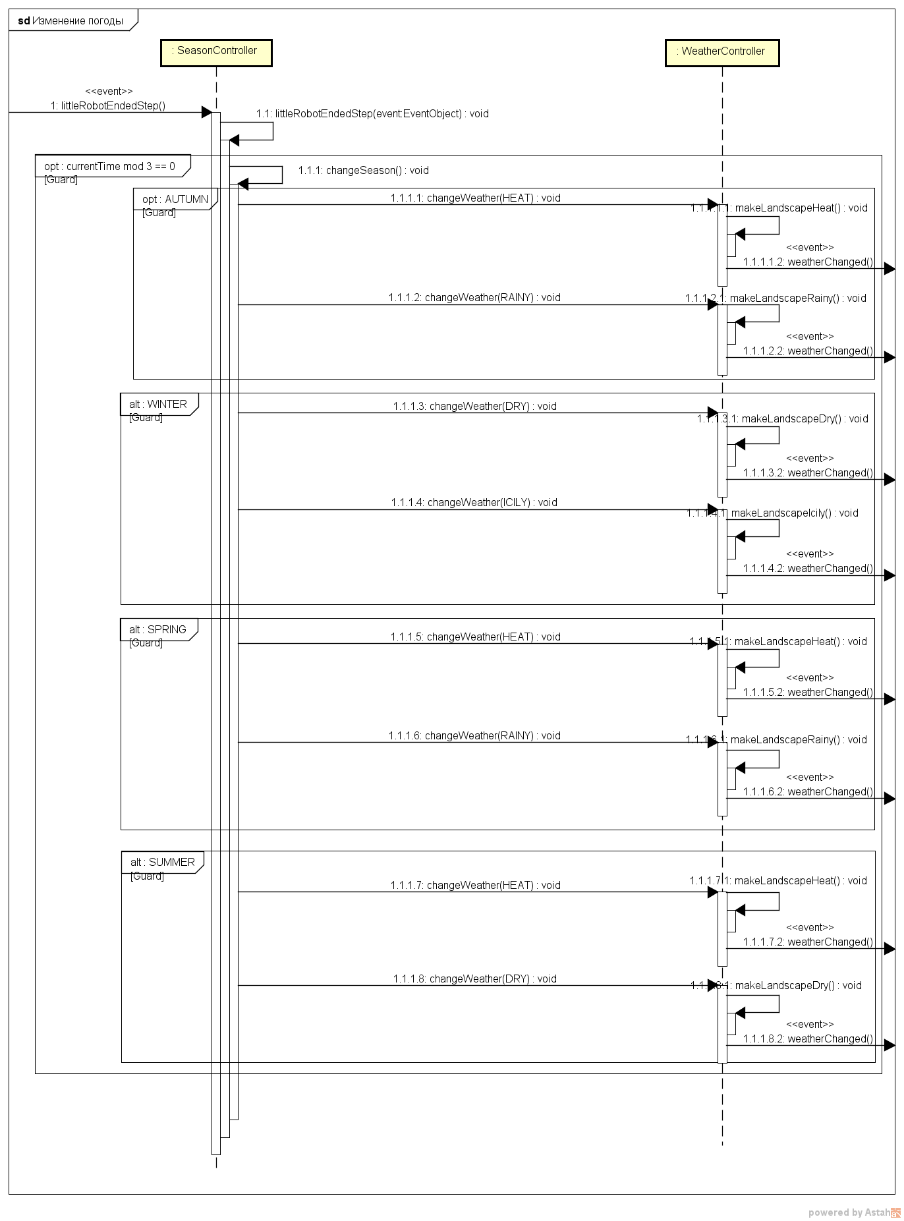
Инициализация игрового поля



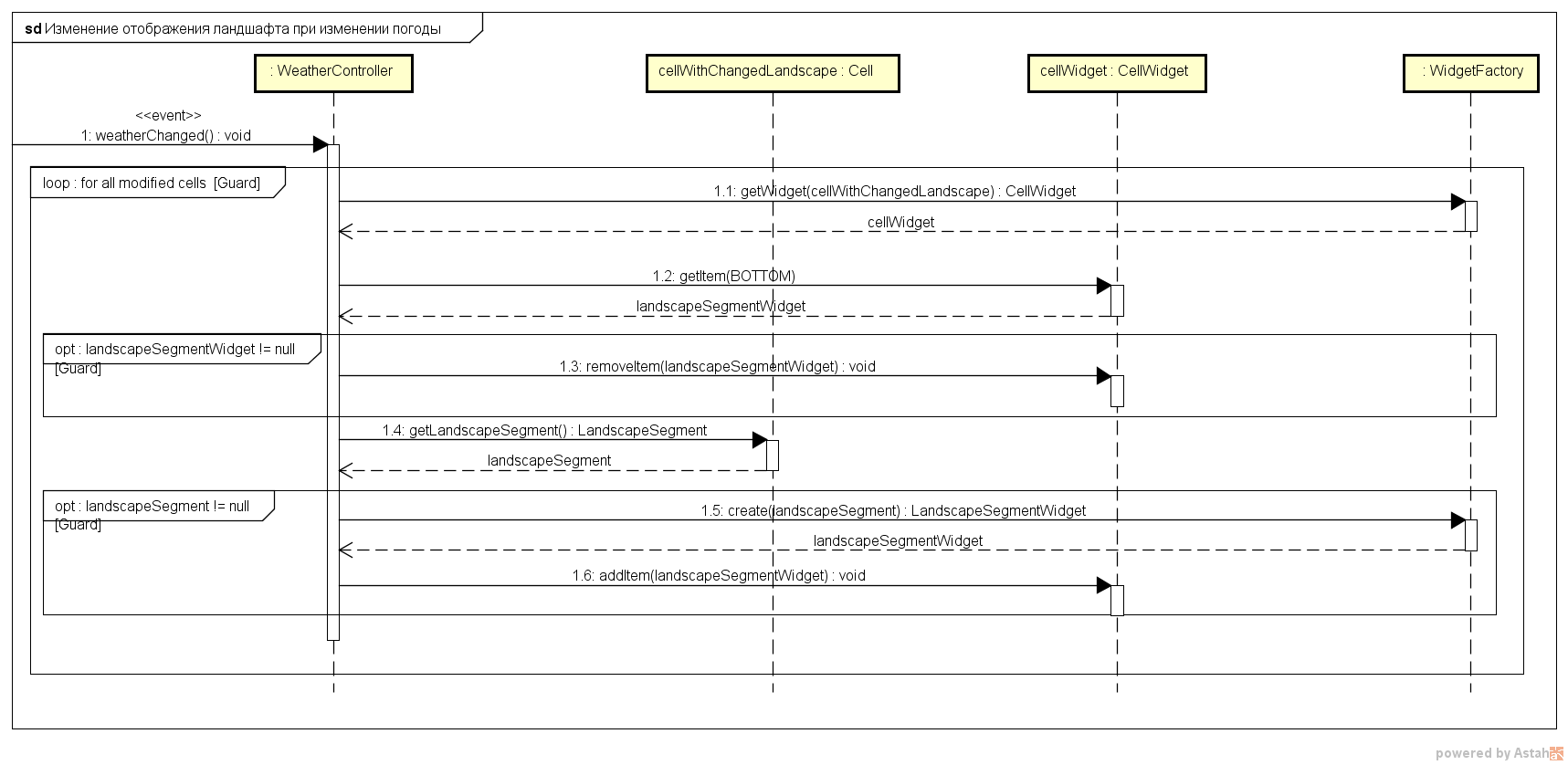
Ход маленького робота



Ход большого робота



Изменение погоды



Изменение отображения ландшафта при изменении погоды

## 4.5 Человеко-машинное взаимодействие

Произошедшие изменения: добавление ландшафта со льдом (рисунок 8) и песком (рисунок 9).



Рис. 8. Лед (замерзшее болото)

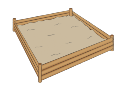


Рис. 9. Песок

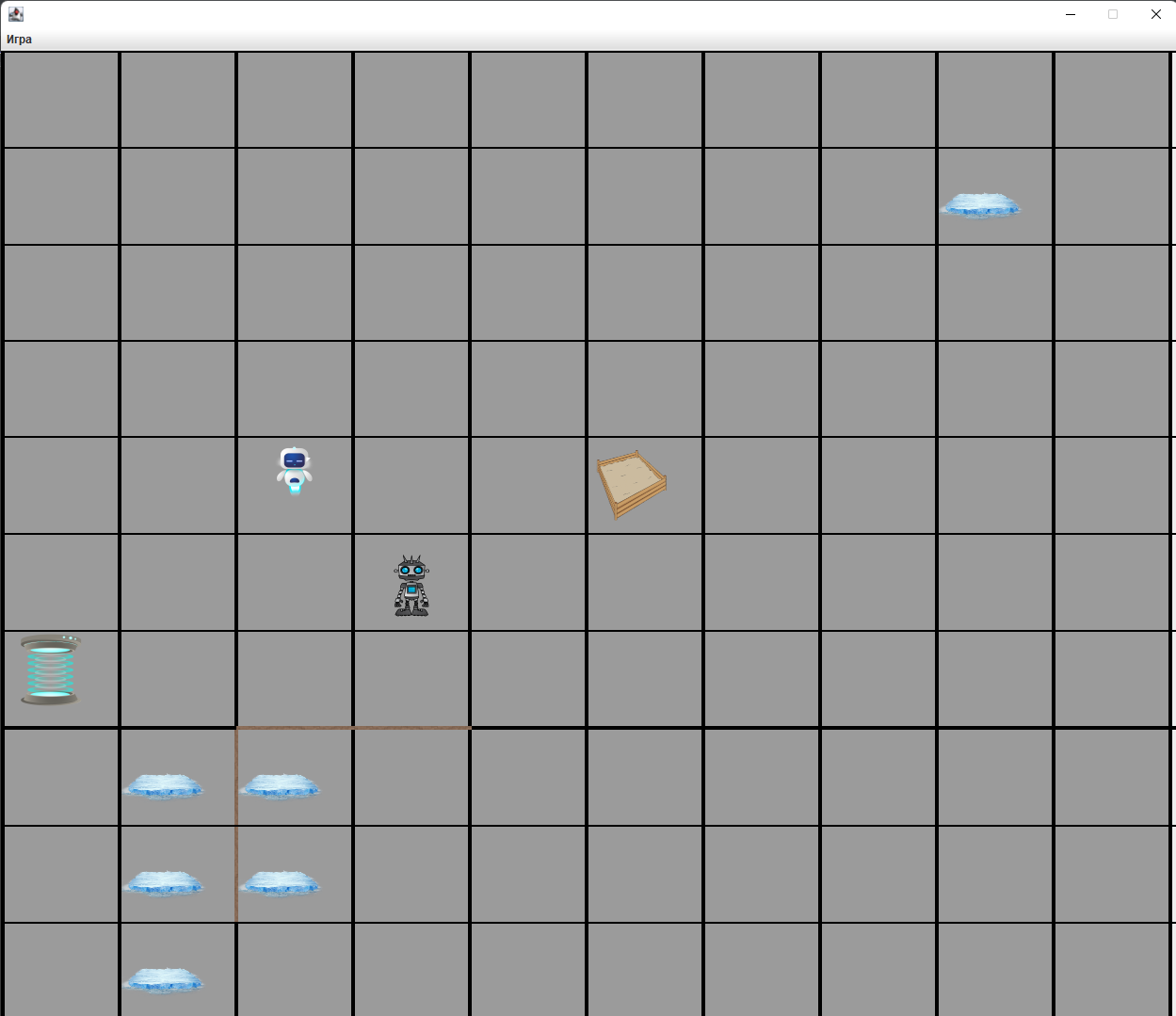


Рис. 10. Пример пользовательского интерфейса игры с установленными песком и замерзшим болтом

## 4.6 Реализация ключевых классов

Поле

public class Field implements ExitCellListener {  
 public Field(int width, int height, MyPoint exitPoint) {  
 \_width = width;  
 \_height = height;  
 buildField(exitPoint);  
 \_cellsArray = new ArrayList<>(\_cells.values());  
 }  
  
 // ----------------------------------------------- Погода ----------------------------------------------------------  
 private SeasonController \_seasonController = null;  
  
 public void setSeason(SeasonController.Season season) {  
 \_seasonController = new SeasonController(this, season);  
 }  
  
 public SeasonController getSeasonController() {  
 return \_seasonController;  
 }  
  
 // ----------------------------------------------- Высота и ширина поле --------------------------------------------  
 private final int \_width, \_height;  
  
 public int getWidth() {  
 return \_width;  
 }  
  
 public int getHeight() {  
 return \_height;  
 }  
  
 // ----------------------------------------------- Ячейки поля -----------------------------------------------------  
 private final TreeMap<MyPoint, Cell> \_cells = new TreeMap<>();  
  
 private final ArrayList<Cell> \_cellsArray;  
  
 public TreeMap<MyPoint, Cell> getCells() {  
 return \_cells;  
 }  
  
 public Cell getCell(MyPoint coordinates) {  
 return \_cells.get(coordinates);  
 }  
  
 public int getIndexOfCell(Cell cell) {  
 return \_cellsArray.indexOf(cell);  
 }  
  
 public Cell getCellByIndex(int index) {  
 Cell cell = null;  
 try {  
 cell = \_cellsArray.get(index);  
 } catch (Exception ignored) {}  
 return cell;  
 }  
  
 public boolean contains(Cell cell) {  
 return \_cells.containsValue(cell);  
 }  
  
 // ----------------------------------------------- Роботы, находящиеся на поле -------------------------------------  
 private final ArrayList<Robot> \_robots = new ArrayList<>();  
  
 public ArrayList<Robot> getRobots() {  
 if (!\_robots.isEmpty()) return \_robots;  
 for (var cell : getCells().values()) {  
 if (cell.getRobot() != null) {  
 \_robots.add(cell.getRobot());  
 }  
 }  
 return \_robots;  
 }  
  
 // ----------------------------------------------- Точка выхода ----------------------------------------------------  
  
 private ExitCell \_exitCell;  
  
 public ExitCell getExitCell() {  
 return \_exitCell;  
 }  
  
 // ----------------------------------------------- Построение поля из ячеек с точкой выхода в exitPoint ------------  
  
 private void buildField(MyPoint exitPoint) {  
 for (int i = 1; i <= \_width; ++i) {  
 for (int j = 1; j <= \_height; ++j) {  
 MyPoint cellCoordinates = new MyPoint(i, j);  
 if (cellCoordinates.equals(exitPoint)) {  
 \_exitCell = new ExitCell(cellCoordinates);  
 \_cells.put(cellCoordinates, \_exitCell);  
 } else {  
 \_cells.put(cellCoordinates, new Cell(cellCoordinates));  
 }  
 }  
 }  
  
 for (var cell : \_cells.entrySet()) {  
 for (var direction : Direction.*values*()) {  
 Cell neighbourCell = \_cells.get(cell.getKey().getNeighbourPoint(direction));  
 if (neighbourCell != null) {  
 cell.getValue().setNeighbourCell(neighbourCell, direction);  
 }  
 }  
 }  
 }  
  
 // ----------------------------------------------- Наблюдение за точкой выхода -------------------------------------  
  
 @Override  
 public void robotTeleported(RobotTeleportEvent e) {  
 \_robots.remove(e.getTeleportedRobot());  
 }  
}

Контроллер сезонов

public class SeasonController implements LittleRobotEndStepListener {  
 // ---------------------------------------------- Сезон ------------------------------------------------------------  
 private static final int *SEASON\_DURATION* = 3;  
 private static final int *NUM\_OF\_SEASONS* = 4;  
 private static final int *SEASONS\_CYCLE\_DURATION* = *SEASON\_DURATION* \* *NUM\_OF\_SEASONS*;  
  
 public enum Season {  
 *SUMMER*(0),  
 *AUTUMN*(*SEASON\_DURATION*),  
 *WINTER*(*SEASON\_DURATION* \* 2),  
 *SPRING*(*SEASON\_DURATION* \* 3);  
  
 private int \_value;  
 Season(int value) {  
 \_value = value;  
 }  
  
 public int getSeasonValue() {  
 return \_value;  
 }  
 }  
  
 private Season getSeasonByValue(int value) {  
 if (0 <= value % *SEASONS\_CYCLE\_DURATION* && value % *SEASONS\_CYCLE\_DURATION* < *SEASON\_DURATION*)  
 return Season.*SUMMER*;  
 else if (*SEASON\_DURATION* <= value % *SEASONS\_CYCLE\_DURATION* && value % *SEASONS\_CYCLE\_DURATION* < *SEASON\_DURATION* \* 2)  
 return Season.*AUTUMN*;  
 else if (*SEASON\_DURATION* \* 2 <= value % *SEASONS\_CYCLE\_DURATION* && value % *SEASONS\_CYCLE\_DURATION* < *SEASON\_DURATION* \* 3)  
 return Season.*WINTER*;  
 else if (*SEASON\_DURATION* \* 3 <= value % *SEASONS\_CYCLE\_DURATION*)  
 return Season.*SPRING*;  
  
 return Season.*SUMMER*;  
 }  
  
 // ----------------------------------------------- Контроллер сезонов ----------------------------------------------  
 private final Field \_field;  
  
 public SeasonController(Field field, Season startSeason) {  
 \_field = field;  
 \_weatherController = new WeatherController(\_field);  
 \_currentSeason = startSeason;  
 \_currentTime = startSeason.getSeasonValue();  
 changeSeason(\_currentSeason);  
 }  
  
 private final WeatherController \_weatherController;  
 private int \_currentTime = 0;  
 private Season \_currentSeason = Season.*SUMMER*;  
  
 public void changeSeason(Season season) {  
 \_currentSeason = season;  
  
 switch (season) {  
 case *SUMMER* -> {  
 \_weatherController.changeWeather(WeatherController.Weather.*DRY*);  
 \_weatherController.changeWeather(WeatherController.Weather.*HEAT*);  
 }  
 case *AUTUMN*, *SPRING* -> {  
 \_weatherController.changeWeather(WeatherController.Weather.*HEAT*);  
 \_weatherController.changeWeather(WeatherController.Weather.*RAINY*);  
 }  
 case *WINTER* -> {  
 \_weatherController.changeWeather(WeatherController.Weather.*DRY*);  
 \_weatherController.changeWeather(WeatherController.Weather.*ICILY*);  
 }  
 }  
 }  
  
 @Override  
 public void littleRobotEndedStep(LittleRobotEndStepEvent e) {  
 \_currentTime = ((\_currentTime + 1) % (*NUM\_OF\_SEASONS* \* *SEASON\_DURATION*));  
 Season newSeason = getSeasonByValue(\_currentTime);  
 if (newSeason != \_currentSeason) {  
 changeSeason(newSeason);  
 fireWeatherChanged();  
 }  
 }  
  
 // ---------------------------------------- Добавление и удаление слушателей WeatherController ---------------------  
 private final List<WeatherChangeListener> \_weatherChangeListeners = new ArrayList<>();  
  
 public void addWeatherChangeListener(WeatherChangeListener l) {  
 \_weatherChangeListeners.add(l);  
 }  
  
 public void addWeatherChangeListener(int index, WeatherChangeListener l) {  
 \_weatherChangeListeners.add(index, l);  
 }  
  
 public void removeWeatherChangeListener(WeatherChangeListener l) {  
 \_weatherChangeListeners.remove(l);  
 }  
  
 private void fireWeatherChanged() {  
 for (WeatherChangeListener l : \_weatherChangeListeners) {  
 WeatherChangeEvent e = new WeatherChangeEvent(this);  
 e.setChangedLandscape(\_weatherController.takeChangedLandscapeCellsInfo());  
 l.weatherChanged(e);  
 }  
 }  
}

Контроллер погоды

class WeatherController {  
 public enum Weather {  
 *RAINY*,  
 *HEAT*,  
 *DRY*,  
 *ICILY* }  
  
 private final Field \_field;  
  
 WeatherController(Field field) {  
 \_field = field;  
 \_currentLandscapeState.addAll(\_field.getCells().values());  
 }  
  
 private final HashMap<Cell, LandscapeSegment> \_previousStateOfModifiedLandscape = new HashMap<>();  
 private final HashMap<Cell, LandscapeSegment> \_stateBuffer = new HashMap<>();  
 private final ArrayList<Cell> \_currentLandscapeState = new ArrayList<>();  
 private final ArrayList<Cell> \_deltaBuffer = new ArrayList<>();  
  
 private final HashSet<Cell> \_delta = new HashSet<>();  
  
 HashSet<Cell> getChangedLandscapeCells() {  
 return \_delta;  
 }  
  
 HashSet<Cell> takeChangedLandscapeCellsInfo() {  
 var temp = (HashSet<Cell>) \_delta.clone();  
 \_delta.clear();  
 return temp;  
 }  
  
 void changeWeather(Weather weather) {  
 switch (weather) {  
 case *DRY* -> makeLandscapeDry();  
 case *RAINY* -> makeLandscapeRainy();  
 case *ICILY* -> makeLandscapeIcily();  
 case *HEAT* -> makeLandscapeHeat();  
 }  
 \_previousStateOfModifiedLandscape.clear();  
 \_previousStateOfModifiedLandscape.putAll(\_stateBuffer);  
 \_stateBuffer.clear();  
 \_deltaBuffer.clear();  
 }  
  
 private void makeLandscapeRainy() {  
 for (var cell : \_currentLandscapeState) {  
 if (!\_deltaBuffer.contains(cell) && cell.getLandscapeSegment() instanceof SwampSegment) {  
 makePuddle(cell);  
 }  
 }  
 }  
  
 private void makePuddle(Cell cell) {  
 for (var neighbour : cell.getNeighbourCells().entrySet()) {  
 setSwampSegment(neighbour.getValue());  
 if (neighbour.getKey() == Direction.*NORTH* || neighbour.getKey() == Direction.*SOUTH*) {  
 setSwampSegment(neighbour.getValue().getNeighbourCell(Direction.*WEST*));  
 setSwampSegment(neighbour.getValue().getNeighbourCell(Direction.*EAST*));  
 }  
 }  
 }  
  
 private void setSwampSegment(Cell cell) {  
 if (cell == null) return;  
 if (!(cell.getLandscapeSegment() instanceof SwampSegment)) {  
 \_stateBuffer.put(cell, cell.getLandscapeSegment());  
 \_delta.add(cell);  
 \_deltaBuffer.add(cell);  
 cell.setLandscapeSegment(new SwampSegment());  
 }  
 }  
  
 private void makeLandscapeDry() {  
 if (\_previousStateOfModifiedLandscape.isEmpty()) return;  
 for (var landscape : \_currentLandscapeState) {  
 if (landscape.getLandscapeSegment() instanceof SwampSegment  
 && \_previousStateOfModifiedLandscape.containsKey(landscape) && !(\_previousStateOfModifiedLandscape.get(landscape) instanceof SwampSegment)) {  
 LandscapeSegment curLandscape = landscape.getLandscapeSegment();  
 landscape.setLandscapeSegment(\_previousStateOfModifiedLandscape.get(landscape));  
 \_stateBuffer.put(landscape, curLandscape);  
 \_delta.add(landscape);  
 \_deltaBuffer.add(landscape);  
 }  
 }  
 }  
  
 private void makeLandscapeIcily() {  
 for (var landscape : \_currentLandscapeState) {  
 if (landscape.getLandscapeSegment() instanceof SwampSegment) {  
 LandscapeSegment buf = landscape.getLandscapeSegment();  
 landscape.setLandscapeSegment(\_previousStateOfModifiedLandscape.get(landscape) instanceof IceSegment ?  
 \_previousStateOfModifiedLandscape.get(landscape) : new IceSegment());  
  
 \_stateBuffer.put(landscape, buf);  
 \_delta.add(landscape);  
 \_deltaBuffer.add(landscape);  
 }  
 }  
 }  
  
 private void makeLandscapeHeat() {  
 for (var landscape : \_currentLandscapeState) {  
 if (landscape.getLandscapeSegment() instanceof IceSegment) {  
 LandscapeSegment buf = landscape.getLandscapeSegment();  
 landscape.setLandscapeSegment(\_previousStateOfModifiedLandscape.get(landscape) instanceof SwampSegment ?  
 \_previousStateOfModifiedLandscape.get(landscape) : new SwampSegment());  
  
 \_stateBuffer.put(landscape, buf);  
 \_delta.add(landscape);  
 \_deltaBuffer.add(landscape);  
 }  
 }  
 }  
}

Ивент изменения погоды

public class WeatherChangeEvent extends EventObject {  
 public WeatherChangeEvent(Object source) {  
 super(source);  
 }  
  
 private HashSet<Cell> \_changedLandscape;  
  
 public void setChangedLandscape(HashSet<Cell> changedLandscape) {  
 \_changedLandscape = changedLandscape;  
 }  
  
 public HashSet<Cell> getChangedLandscape() {  
 return \_changedLandscape;  
 }  
}

Интерфейс слушателя погоды

public interface WeatherChangeListener {  
 void weatherChanged(WeatherChangeEvent e);  
}

Контроллер изменения погоды в UI

private class WeatherController implements WeatherChangeListener {  
  
 @Override  
 public void weatherChanged(WeatherChangeEvent e) {  
 for (var cell : e.getChangedLandscape()) {  
 CellWidget cellWidget = \_fields.get(\_game.getGameField()).getWidgetFactory().getWidget(cell);  
 LandscapeSegmentWidget oldLandscape = (LandscapeSegmentWidget) cellWidget.getItem(CellWidget.Layer.*BOTTOM*);  
 if (oldLandscape != null) {  
 cellWidget.removeItem(oldLandscape);  
 \_fields.get(\_game.getGameField()).getWidgetFactory().remove(oldLandscape);  
 }  
 if (cell.getLandscapeSegment() != null) {  
 CellItemWidget newLandscape = \_fields.get(\_game.getGameField()).getWidgetFactory().create(cell.getLandscapeSegment());  
 cellWidget.addItem(newLandscape);  
 }  
 }  
 updateUI();  
 }  
}

Робот

public abstract class Robot extends CellItem implements WeatherChangeListener {  
  
 // ----------------------------------------------- Позиция ---------------------------------------------------------  
 public boolean setPosition(Cell cell) {  
 if (cell == null) return false;  
  
 if (\_position == cell) return true;  
  
 boolean isRobotCanToBeLocatedAtNewPosition = canBeLocatedAtPosition(cell) || cell.getRobot() == this;  
 if (!isRobotCanToBeLocatedAtNewPosition) return false;  
  
 \_position = cell;  
  
 return cell.setRobot(this);  
 }  
  
 public void setNullPosition() {  
 \_position = null;  
 }  
  
 public Cell getPosition() {  
 return \_position;  
 }  
  
 public boolean canBeLocatedAtPosition(Cell cell) {  
 return cell != null && cell.getRobot() == null;  
 }  
  
 // ----------------------------------------------- Перемещение -----------------------------------------------------  
 protected Cell canMove(Direction direction) {  
 if (direction == null || \_position == null) return null;  
  
 Cell neighbourCell = \_position.getNeighbourCell(direction);  
  
 boolean canMoveAtNeighbourCell = neighbourCell != null  
 && canBeLocatedAtPosition(\_position.getNeighbourCell(direction))  
 && \_position.getWallSegment(direction) == null;  
  
 if (canMoveAtNeighbourCell) {  
 return neighbourCell;  
 }  
  
 return null;  
 }  
  
 protected boolean makeStep(Direction direction) {  
 if (move(direction)) {  
 processIfLandscapeSegment();  
 return true;  
 }  
 return false;  
 }  
  
 private boolean move(Direction direction) {  
 Cell cellToMove = canMove(direction);  
  
 if (cellToMove == null) return false;  
  
 Cell fromCell = getPosition();  
 getPosition().takeRobot();  
  
 boolean isRobotSuccessfullySetAtNewPosition = setPosition(cellToMove);  
  
 if (isRobotSuccessfullySetAtNewPosition) {  
 fireRobotMove(fromCell, cellToMove);  
 }  
  
 return isRobotSuccessfullySetAtNewPosition;  
 }  
  
 protected boolean canMoveAfterProcessingMoveCharacteristic() {  
 boolean isRobotStuck = robotHasViscosityCharacteristic();  
 return !(isRobotStuck);  
 }  
  
 // ----------------------------------------------- Работа с ландшафтом ---------------------------------------------  
 protected MoveCharacteristic \_characteristic = null;  
  
 public abstract void processIfLandscapeSegment();  
  
 protected void processBeforeMovingCharacteristic() {  
 boolean isBeforeMovingCharacteristic = robotHasViscosityCharacteristic();  
  
 if (!isBeforeMovingCharacteristic) return;  
  
 boolean isCharacteristicSetButLandscapeSegmentNotUnderTheRobot = (\_position.getLandscapeSegment() == null);  
  
 boolean isLifeTimeOfCharacteristicEnded = !\_characteristic.decrementLifeTime();  
  
 if (isCharacteristicSetButLandscapeSegmentNotUnderTheRobot || isLifeTimeOfCharacteristicEnded) {  
 \_characteristic = null;  
 }  
 }  
  
 protected void processAfterMovingCharacteristic() {  
 boolean isAfterMovingCharacteristic = robotHasSlipperinessCharacteristic();  
  
 if (!isAfterMovingCharacteristic) return;  
  
 boolean isCharacteristicSetButLandscapeSegmentNotUnderTheRobot = (\_position.getLandscapeSegment() == null);  
  
 boolean isLifeTimeOfCharacteristicEnded = !\_characteristic.decrementLifeTime();  
  
 if (isCharacteristicSetButLandscapeSegmentNotUnderTheRobot || isLifeTimeOfCharacteristicEnded) {  
 \_characteristic = null;  
 }  
 }  
  
 protected void setLandscapeCharacteristic(MoveCharacteristic characteristic) {  
 boolean newAndOldCharacteristicAreSlipperiness = (robotHasSlipperinessCharacteristic()  
 && characteristic instanceof SlipperinessCharacteristic);  
  
 if (characteristic == null || newAndOldCharacteristicAreSlipperiness) return;  
  
 \_characteristic = characteristic;  
 }  
  
 protected boolean robotHasViscosityCharacteristic() {  
 return \_characteristic instanceof ViscosityCharacteristic;  
 }  
  
 protected boolean robotHasSlipperinessCharacteristic() {  
 return \_characteristic instanceof SlipperinessCharacteristic;  
 }  
  
 protected abstract MoveCharacteristic createMoveCharacteristic(LandscapeSegment landscapeSegment);  
  
 // ----------------------------------------------- Генерация события при перемещении -------------------------------  
 private final List<RobotMoveListener> \_moveListeners = new ArrayList<>();  
  
 public void addRobotMoveListener(RobotMoveListener l) {  
 if (l == null) return;  
 \_moveListeners.add(l);  
 }  
  
 public void addRobotMoveListener(int index, RobotMoveListener l) {  
 if (l == null) return;  
 \_moveListeners.add(index, l);  
 }  
  
 public void removeRobotMoveListener(RobotMoveListener l) {  
 \_moveListeners.remove(l);  
 }  
  
 protected void fireRobotMove(Cell fromCell, Cell toCell) {  
 for (RobotMoveListener l : \_moveListeners) {  
 RobotMoveEvent e = new RobotMoveEvent(this);  
 e.setRobot(this);  
 e.setFromCell(fromCell);  
 e.setToCell(toCell);  
 l.robotMadeMove(e);  
 }  
 }  
  
 public void weatherChanged(WeatherChangeEvent e) {  
 processIfLandscapeSegment();  
 }  
  
 // ----------------------------------------------- Коэффициенты характеристик передвижения -------------------------  
 protected static class MoveCharacteristicCoefficients {  
 protected static final int *SwampViscosityCoefficient* = 0;  
 protected static final int *SandViscosityCoefficient* = 0;  
 protected static final int *IceSlipperinessCoefficient* = 0;  
 }  
}

Маленький робот

public class LittleRobot extends Robot implements Destroyable {  
  
 // ----------------------------------------------- Перемещение -----------------------------------------------------  
  
 @Override  
 public boolean makeStep(Direction direction) {  
 boolean isSuccessfulStep = false;  
 boolean isRobotSlides;  
 do {  
 processBeforeMovingCharacteristic();  
  
 if (!canMoveAfterProcessingMoveCharacteristic()) return isSuccessfulStep;  
  
 isSuccessfulStep = super.makeStep(direction) || isSuccessfulStep;  
  
 processAfterMovingCharacteristic();  
  
 isRobotSlides = robotHasSlipperinessCharacteristic();  
 } while (isRobotSlides);  
  
 fireLittleRobotEndStep();  
  
 return isSuccessfulStep;  
 }  
  
 // ----------------------------------------------- Работа с ландшафтом ---------------------------------------------  
 @Override  
 public void processIfLandscapeSegment() {  
 if (\_position == null) return;  
  
 setLandscapeCharacteristic(createMoveCharacteristic(\_position.getLandscapeSegment()));  
  
 if (\_position.getLandscapeSegment() instanceof SwampSegment) {  
 destroy();  
 }  
 }  
  
 @Override  
 protected void processBeforeMovingCharacteristic() {  
 super.processBeforeMovingCharacteristic();  
 if (robotHasViscosityCharacteristic()) {  
 fireLittleRobotEndStep();  
 }  
 }  
  
 protected MoveCharacteristic createMoveCharacteristic(LandscapeSegment landscapeSegment) {  
 if (\_position.getLandscapeSegment() instanceof SandSegment) {  
 return new ViscosityCharacteristic(MoveCharacteristicCoefficients.*SandViscosityCoefficient*);  
 } else if (\_position.getLandscapeSegment() instanceof IceSegment) {  
 return new SlipperinessCharacteristic(MoveCharacteristicCoefficients.*IceSlipperinessCoefficient*);  
 }  
 return null;  
 }  
  
 // ----------------------------------------------- Уничтожение -----------------------------------------------------  
 @Override  
 public void destroy() {  
 Cell whereDestroyedCell = \_position;  
 \_position.takeRobot();  
 fireRobotDestroy(whereDestroyedCell);  
 }  
  
 // ----------------------------------------------- Порождение события при уничтожении ------------------------------  
  
 private final List<RobotDestroyListener> \_destroyListeners = new ArrayList<>();  
  
 public void addRobotDestroyListener(RobotDestroyListener l) {  
 if (l == null) return;  
 \_destroyListeners.add(l);  
 }  
  
 public void addRobotDestroyListener(int index, RobotDestroyListener l) {  
 if (l == null) return;  
 \_destroyListeners.add(index, l);  
 }  
  
 public void removeRobotDestroyListener(RobotDestroyListener l) {  
 \_destroyListeners.remove(l);  
 }  
  
 protected void fireRobotDestroy(Cell whereDestroyedCell) {  
 for (RobotDestroyListener l : \_destroyListeners) {  
 RobotDestroyEvent e = new RobotDestroyEvent(this);  
 e.setWhereDestroyedCell(whereDestroyedCell);  
 e.setDestroyedRobot(this);  
 l.robotDestroyed(e);  
 }  
 }  
  
 // ----------------------------------------------- Порождение события об окончании хода ----------------------------  
  
 private final List<LittleRobotEndStepListener> \_endStepListeners = new ArrayList<>();  
  
 public void addRobotEndStepListener(LittleRobotEndStepListener l) {  
 if (l == null) return;  
 \_endStepListeners.add(l);  
 }  
  
 public void addRobotEndStepListener(int index, LittleRobotEndStepListener l) {  
 if (l == null) return;  
 \_endStepListeners.add(index, l);  
 }  
  
 public void removeRobotDestroyListener(LittleRobotEndStepListener l) {  
 \_endStepListeners.remove(l);  
 }  
  
 protected void fireLittleRobotEndStep() {  
 for (LittleRobotEndStepListener l : \_endStepListeners) {  
 l.littleRobotEndedStep(new LittleRobotEndStepEvent(this));  
 }  
 }  
  
 // ----------------------------------------------- Коэффициенты характеристик передвижения -------------------------  
 protected static class MoveCharacteristicCoefficients {  
 protected final static int *SandViscosityCoefficient* = 1;  
 protected final static int *IceSlipperinessCoefficient* = 1;  
 }  
}

Большой робот

public class BigRobot extends Robot implements LittleRobotEndStepListener {  
 private PathFinder \_pathFinder;  
  
 public void setPathFinder(PathFinder pathFinder) {  
 if (pathFinder == null) return;  
 if (\_pathFinder == pathFinder) return;  
 dropPathFinder();  
 \_pathFinder = pathFinder;  
 \_pathFinder.setRobot(this);  
 }  
  
 public PathFinder getPathFinder() {  
 return \_pathFinder;  
 }  
  
 public void dropPathFinder() {  
 var pathFinder = \_pathFinder;  
 \_pathFinder = null;  
 if (pathFinder != null) pathFinder.forgetRobot();  
 }  
  
 // ----------------------------------------------- Цель ------------------------------------------------------------  
 private Destroyable \_target = null;  
  
 private final int TARGET\_SIGHT\_DISTANCE = 2;  
  
 public void setTarget(Destroyable target) {  
 \_target = target;  
 }  
  
 private void catchTarget() {  
 if (\_target != null) {  
 \_target.destroy();  
 }  
 }  
  
 private boolean checkIfTargetInNeighbourCell(Direction direction) {  
 return \_target != null && \_position.getNeighbourCell(direction) == getTargetPosition();  
 }  
  
 private Cell getTargetPosition() {  
 if (\_target == null) return null;  
 return ((CellItem)\_target).getPosition();  
 }  
  
 // ----------------------------------------------- Перемещение -----------------------------------------------------  
 private void makeStep() {  
 if (\_target == null) return;  
 Direction directionToMove = calculateStepDirection(getTargetPosition());  
 makeStep(directionToMove);  
 }  
  
 @Override  
 protected boolean makeStep(Direction direction) {  
 boolean isSuccessfulStep = false;  
  
 boolean isRobotSlides;  
 do {  
 processBeforeMovingCharacteristic();  
  
 if (!canMoveAfterProcessingMoveCharacteristic()) return isSuccessfulStep;  
  
 isSuccessfulStep = super.makeStep(direction) || isSuccessfulStep;  
  
 processAfterMovingCharacteristic();  
  
 if (!isSuccessfulStep && direction != null) {  
 if (checkIfTargetInNeighbourCell(direction)) {  
 catchTarget();  
 return true;  
 }  
 }  
  
 isRobotSlides = robotHasSlipperinessCharacteristic();  
 } while(isRobotSlides);  
  
 return isSuccessfulStep;  
 }  
  
 private Direction calculateStepDirection(Cell cell) {  
 if (cell == null) return null;  
  
 MyPoint distanceToTarget = \_position.getDistanceFromCellToCell(cell);  
  
 boolean isDistanceToTargetInRangeOfSight = distanceToTarget.getX() <= TARGET\_SIGHT\_DISTANCE  
 && distanceToTarget.getY() <= TARGET\_SIGHT\_DISTANCE;  
  
 Direction directionToMove;  
 if (isDistanceToTargetInRangeOfSight) {  
 directionToMove = getDirectionToMoveToCellUsingPathFinder(cell);  
 } else {  
 directionToMove = getDirectionToMoveHorizontally(cell);  
 }  
 return directionToMove;  
 }  
  
 private Direction getDirectionToMoveToCellUsingPathFinder(Cell cell) {  
 var pathToMove = \_pathFinder.findPath(getPosition(), cell);  
  
 Direction directionToMove = null;  
 if (!pathToMove.isEmpty()) {  
 pathToMove.pop();  
 directionToMove = \_position.getNeighbourDirection(pathToMove.pop());  
 }  
  
 return directionToMove;  
 }  
  
 private Direction getDirectionToMoveHorizontally(Cell cell) {  
 Direction calculatedDirectionToMove = \_position.getDirectionsToMove(cell).pop();  
  
 if (calculatedDirectionToMove == Direction.*EAST* || calculatedDirectionToMove == Direction.*WEST*) {  
 return calculatedDirectionToMove;  
 }  
  
 return null;  
 }  
  
 // ----------------------------------------------- Работа с ландшафтом ---------------------------------------------  
  
 @Override  
 public void processIfLandscapeSegment() {  
 if (\_position == null) return;  
 setLandscapeCharacteristic(createMoveCharacteristic(\_position.getLandscapeSegment()));  
 }  
  
 protected MoveCharacteristic createMoveCharacteristic(LandscapeSegment landscapeSegment) {  
 if (\_position.getLandscapeSegment() instanceof SwampSegment) {  
 return new ViscosityCharacteristic(MoveCharacteristicCoefficients.*SwampViscosityCoefficient*);  
 } else if (\_position.getLandscapeSegment() instanceof SandSegment) {  
 return new ViscosityCharacteristic(MoveCharacteristicCoefficients.*SandViscosityCoefficient*);  
 } else if (\_position.getLandscapeSegment() instanceof IceSegment) {  
 return new SlipperinessCharacteristic(MoveCharacteristicCoefficients.*IceSlipperinessCoefficient*);  
 }  
  
 return null;  
 }  
  
 // ----------------------------------------------- Наблюдение за перемещением маленького робота --------------------  
 @Override  
 public void littleRobotEndedStep(LittleRobotEndStepEvent e) {  
 makeStep();  
 }  
  
 // ----------------------------------------------- Коэффициенты характеристик передвижения -------------------------  
 protected static class MoveCharacteristicCoefficients {  
 protected static final int *SwampViscosityCoefficient* = 3;  
 protected static final int *SandViscosityCoefficient* = 1;  
 protected static final int *IceSlipperinessCoefficient* = 1;  
 }  
}

## 4.7 Реализация ключевых тестовых сценариев

Тест на изменение сезонов с ходами маленького робота

@Test  
public void changingSeasonsTest() {  
 Field field = new Field(10, 10, new MyPoint(1, 1));  
 MyPoint initialSwampSegmentPosition = new MyPoint(5, 5);  
 SwampSegment initialSwampSegment = new SwampSegment();  
 field.getCell(initialSwampSegmentPosition).setLandscapeSegment(initialSwampSegment);  
  
 field.setSeason(SeasonController.Season.*SUMMER*);  
  
 // Лето  
 int numOfStep = 0;  
 for (; numOfStep < 3; ++numOfStep) {  
 Assertions.*assertTrue*(field.getCell(initialSwampSegmentPosition).getLandscapeSegment() instanceof SwampSegment);  
  
 for (int i = initialSwampSegmentPosition.getX() - 1; i <= initialSwampSegmentPosition.getX() + 1; ++i) {  
 for (int j = initialSwampSegmentPosition.getY() - 1; j <= initialSwampSegmentPosition.getY() + 1; ++j) {  
 MyPoint curPosition = new MyPoint(i, j);  
 Cell currentCell = field.getCell(curPosition);  
 if (!(curPosition.getY() == initialSwampSegmentPosition.getY() && curPosition.getX() == initialSwampSegmentPosition.getX())) {  
 Assertions.*assertNull*(currentCell.getLandscapeSegment());  
 }  
 }  
 }  
  
 int cnt = 0;  
 for (var cell : field.getCells().values()) {  
 if (cell.getLandscapeSegment() != null) {  
 cnt++;  
 }  
 }  
  
 Assertions.*assertEquals*(1, cnt);  
  
 field.getSeasonController().littleRobotEndedStep(new LittleRobotEndStepEvent(this));  
 }  
  
 // Осень  
 for (; numOfStep < 6; ++numOfStep) {  
 Assertions.*assertEquals*(initialSwampSegment, field.getCell(initialSwampSegmentPosition).getLandscapeSegment());  
  
 for (int i = initialSwampSegmentPosition.getX() - 1; i <= initialSwampSegmentPosition.getX() + 1; ++i) {  
 for (int j = initialSwampSegmentPosition.getY() - 1; j <= initialSwampSegmentPosition.getY() + 1; ++j) {  
 MyPoint curPosition = new MyPoint(i, j);  
 Cell currentCell = field.getCell(curPosition);  
 if (!(curPosition.getY() == initialSwampSegmentPosition.getY() && curPosition.getX() == initialSwampSegmentPosition.getX())) {  
 Assertions.*assertNotNull*(currentCell.getLandscapeSegment());  
 Assertions.*assertEquals*(SwampSegment.class, currentCell.getLandscapeSegment().getClass());  
 }  
 }  
 }  
  
 int cnt = 0;  
 for (var cell : field.getCells().values()) {  
 if (cell.getLandscapeSegment() != null) {  
 cnt++;  
 }  
 }  
  
 Assertions.*assertEquals*(9, cnt);  
 field.getSeasonController().littleRobotEndedStep(new LittleRobotEndStepEvent(this));  
 }  
  
 // Зима  
 for (; numOfStep < 9; ++numOfStep) {  
 Assertions.*assertTrue*(field.getCell(initialSwampSegmentPosition).getLandscapeSegment() instanceof IceSegment);  
  
 for (int i = initialSwampSegmentPosition.getX() - 1; i <= initialSwampSegmentPosition.getX() + 1; ++i) {  
 for (int j = initialSwampSegmentPosition.getY() - 1; j <= initialSwampSegmentPosition.getY() + 1; ++j) {  
 MyPoint curPosition = new MyPoint(i, j);  
 Cell currentCell = field.getCell(curPosition);  
 if (!(curPosition.getY() == initialSwampSegmentPosition.getY() && curPosition.getX() == initialSwampSegmentPosition.getX())) {  
 Assertions.*assertNull*(currentCell.getLandscapeSegment());  
 }  
 }  
 }  
  
 int cnt = 0;  
 for (var cell : field.getCells().values()) {  
 if (cell.getLandscapeSegment() != null) {  
 cnt++;  
 }  
 }  
  
 Assertions.*assertEquals*(1, cnt);  
 field.getSeasonController().littleRobotEndedStep(new LittleRobotEndStepEvent(this));  
 }  
  
 // Весна  
 for (; numOfStep < 12; ++numOfStep) {  
 Assertions.*assertEquals*(initialSwampSegment, field.getCell(initialSwampSegmentPosition).getLandscapeSegment());  
  
 for (int i = initialSwampSegmentPosition.getX() - 1; i <= initialSwampSegmentPosition.getX() + 1; ++i) {  
 for (int j = initialSwampSegmentPosition.getY() - 1; j <= initialSwampSegmentPosition.getY() + 1; ++j) {  
 MyPoint curPosition = new MyPoint(i, j);  
 Cell currentCell = field.getCell(curPosition);  
 if (!(curPosition.getY() == initialSwampSegmentPosition.getY() && curPosition.getX() == initialSwampSegmentPosition.getX())) {  
 Assertions.*assertNotNull*(currentCell.getLandscapeSegment());  
 Assertions.*assertEquals*(SwampSegment.class, currentCell.getLandscapeSegment().getClass());  
 }  
 }  
 }  
  
 int cnt = 0;  
 for (var cell : field.getCells().values()) {  
 if (cell.getLandscapeSegment() != null) {  
 cnt++;  
 }  
 }  
  
 Assertions.*assertEquals*(9, cnt);  
 field.getSeasonController().littleRobotEndedStep(new LittleRobotEndStepEvent(this));  
 }  
  
 // Лето  
 Assertions.*assertTrue*(field.getCell(initialSwampSegmentPosition).getLandscapeSegment() instanceof SwampSegment);  
  
 for (int i = initialSwampSegmentPosition.getX() - 1; i <= initialSwampSegmentPosition.getX() + 1; ++i) {  
 for (int j = initialSwampSegmentPosition.getY() - 1; j <= initialSwampSegmentPosition.getY() + 1; ++j) {  
 MyPoint curPosition = new MyPoint(i, j);  
 Cell currentCell = field.getCell(curPosition);  
 if (!(curPosition.getY() == initialSwampSegmentPosition.getY() && curPosition.getX() == initialSwampSegmentPosition.getX())) {  
 Assertions.*assertNull*(currentCell.getLandscapeSegment());  
 }  
 }  
 }  
  
 int cnt = 0;  
 for (var cell : field.getCells().values()) {  
 if (cell.getLandscapeSegment() != null) {  
 cnt++;  
 }  
 }  
  
 Assertions.*assertEquals*(1, cnt);  
}

Тест на перемещение робота на клетку с замерзшим болотом

@Test  
public void littleRobotMoveOnIceSegmentTest() {  
 int exitPointX = 1, exitPointY = 1;  
 MyPoint exitPointCoordinates = new MyPoint(exitPointX, exitPointY);  
 int width = 5, height = 5;  
 Field field = new Field(width, height, exitPointCoordinates);  
  
 LittleRobot robot = new LittleRobot();  
  
 MyPoint littleRobotStartPosition = new MyPoint(3, 3);  
  
 // Установить робота на поле  
 Assertions.*assertTrue*(field.getCell(littleRobotStartPosition).setRobot(robot));  
 Assertions.*assertEquals*(field.getCell(littleRobotStartPosition), robot.getPosition());  
 Assertions.*assertEquals*(field.getCell(littleRobotStartPosition).getRobot(), robot);  
  
 // Установить справа лед  
 MyPoint icePosition = new MyPoint(4, 3);  
 field.getCell(icePosition).setLandscapeSegment(new IceSegment());  
  
 MyPoint resultPosition = icePosition.getNeighbourPoint(Direction.*EAST*);  
  
 // Переместиться в ячейку со льдом  
 Assertions.*assertTrue*(robot.makeStep(Direction.*EAST*));  
  
 Assertions.*assertNull*(field.getCell(littleRobotStartPosition).getRobot());  
 Assertions.*assertNull*(field.getCell(icePosition).getRobot());  
 Assertions.*assertEquals*(field.getCell(resultPosition).getRobot(), robot);  
 Assertions.*assertEquals*(field.getCell(resultPosition), robot.getPosition());  
}

Тест на перемещение маленького робота на ячейку с песком

@Test  
public void littleRobotMoveOnSandSegmentTest() {  
 int exitPointX = 1, exitPointY = 1;  
 MyPoint exitPointCoordinates = new MyPoint(exitPointX, exitPointY);  
 int width = 5, height = 5;  
 Field field = new Field(width, height, exitPointCoordinates);  
  
 LittleRobot robot = new LittleRobot();  
  
 MyPoint littleRobotStartPosition = new MyPoint(3, 3);  
  
 // Установить робота на поле  
 Assertions.*assertTrue*(field.getCell(littleRobotStartPosition).setRobot(robot));  
 Assertions.*assertEquals*(field.getCell(littleRobotStartPosition), robot.getPosition());  
 Assertions.*assertEquals*(field.getCell(littleRobotStartPosition).getRobot(), robot);  
  
 // Установить справа песок  
 MyPoint sandPosition = new MyPoint(4, 3);  
 field.getCell(sandPosition).setLandscapeSegment(new SandSegment());  
  
 // Переместиться в ячейку с песком  
 Assertions.*assertTrue*(robot.makeStep(Direction.*EAST*));  
  
 Assertions.*assertNull*(field.getCell(littleRobotStartPosition).getRobot());  
 Assertions.*assertEquals*(field.getCell(sandPosition), robot.getPosition());  
 Assertions.*assertEquals*(robot, field.getCell(sandPosition).getRobot());  
  
 Assertions.*assertFalse*(robot.makeStep(Direction.*EAST*));  
  
 MyPoint resultPosition = sandPosition.getNeighbourPoint(Direction.*EAST*);  
  
 Assertions.*assertNull*(field.getCell(resultPosition).getRobot());  
 Assertions.*assertEquals*(field.getCell(sandPosition), robot.getPosition());  
 Assertions.*assertEquals*(robot, field.getCell(sandPosition).getRobot());  
  
 Assertions.*assertTrue*(robot.makeStep(Direction.*EAST*));  
  
 Assertions.*assertNull*(field.getCell(sandPosition).getRobot());  
 Assertions.*assertEquals*(field.getCell(resultPosition), robot.getPosition());  
 Assertions.*assertEquals*(robot, field.getCell(resultPosition).getRobot());  
  
}

Тест на перемещение маленького робота на клетку со льдом, стена по направлению скольжения

@Test  
public void littleRobotMoveOnIceSegmentButMeetWallTest() {  
 int exitPointX = 1, exitPointY = 1;  
 MyPoint exitPointCoordinates = new MyPoint(exitPointX, exitPointY);  
 int width = 5, height = 5;  
 Field field = new Field(width, height, exitPointCoordinates);  
  
 LittleRobot robot = new LittleRobot();  
  
 MyPoint littleRobotStartPosition = new MyPoint(3, 3);  
  
 // Установить робота на поле  
 Assertions.*assertTrue*(field.getCell(littleRobotStartPosition).setRobot(robot));  
 Assertions.*assertEquals*(field.getCell(littleRobotStartPosition), robot.getPosition());  
 Assertions.*assertEquals*(field.getCell(littleRobotStartPosition).getRobot(), robot);  
  
 // Установить справа лед  
 MyPoint ice1Position = new MyPoint(4, 3);  
 field.getCell(ice1Position).setLandscapeSegment(new IceSegment());  
 field.getCell(ice1Position).setWallSegment(new WallSegment(), Direction.*EAST*);  
  
 MyPoint resultPosition = ice1Position;  
  
 // Переместиться в ячейку со льдом  
 Assertions.*assertTrue*(robot.makeStep(Direction.*EAST*));  
  
 Assertions.*assertNull*(field.getCell(littleRobotStartPosition).getRobot());  
 Assertions.*assertEquals*(field.getCell(resultPosition).getRobot(), robot);  
 Assertions.*assertEquals*(field.getCell(resultPosition), robot.getPosition());  
}

Тест на перемещение большого робота в клетку с замерзшим болотом

@Test  
public void moveOnIceSegmentBigRobotTest() {  
 int exitPointX = 3, exitPointY = 3;  
 Field field = new Field(5, 5, new MyPoint(exitPointX, exitPointY));  
  
 LittleRobot littleRobot = new LittleRobot();  
 field.getCell(new MyPoint(5, 5)).setRobot(littleRobot);  
  
 PathFinder pathFinder = new PathFinder(field);  
 BigRobot bigRobot = new BigRobot();  
 bigRobot.setPathFinder(pathFinder);  
  
 bigRobot.setTarget(littleRobot);  
 field.getCell(new MyPoint(1, 1)).setRobot(bigRobot);  
 field.getCell(new MyPoint(2, 1)).setLandscapeSegment(new IceSegment());  
  
 littleRobot.addRobotEndStepListener(bigRobot);  
  
 Assertions.*assertTrue*(littleRobot.makeStep(Direction.*SOUTH*));  
 Assertions.*assertNull*(field.getCell(new MyPoint(2, 1)).getRobot());  
 Assertions.*assertEquals*(bigRobot, field.getCell(new MyPoint(3, 1)).getRobot());  
}

Тест на перемещение большого робота в ячейку с замерзшим болтом, во время скольжения встретил стену

@Test  
public void moveOnIceSegmentButMeetWallBigRobotTest() {  
 int exitPointX = 3, exitPointY = 3;  
 Field field = new Field(5, 5, new MyPoint(exitPointX, exitPointY));  
  
 LittleRobot littleRobot = new LittleRobot();  
 field.getCell(new MyPoint(5, 5)).setRobot(littleRobot);  
  
 PathFinder pathFinder = new PathFinder(field);  
 BigRobot bigRobot = new BigRobot();  
 bigRobot.setPathFinder(pathFinder);  
  
 bigRobot.setTarget(littleRobot);  
 field.getCell(new MyPoint(1, 1)).setRobot(bigRobot);  
 field.getCell(new MyPoint(2, 1)).setLandscapeSegment(new IceSegment());  
 field.getCell(new MyPoint(2, 1)).setWallSegment(new WallSegment(), Direction.*EAST*);  
  
 littleRobot.addRobotEndStepListener(bigRobot);  
  
 Assertions.*assertTrue*(littleRobot.makeStep(Direction.*SOUTH*));  
 Assertions.*assertEquals*(bigRobot, field.getCell(new MyPoint(2, 1)).getRobot());  
 Assertions.*assertNull*(field.getCell(new MyPoint(3, 1)).getRobot());  
}

Тест на перемещение большого робота в ячейку с замерзшим болтом, во время скольжения встретил маленького робота

@Test  
public void moveOnIceSegmentCatchLittleRobotWhileSlidesBigRobotTest() {  
 int exitPointX = 3, exitPointY = 3;  
 Field field = new Field(7, 7, new MyPoint(exitPointX, exitPointY));  
  
 LittleRobot littleRobot = new LittleRobot();  
 field.getCell(new MyPoint(3, 2)).setRobot(littleRobot);  
  
 PathFinder pathFinder = new PathFinder(field);  
 BigRobot bigRobot = new BigRobot();  
 bigRobot.setPathFinder(pathFinder);  
  
 bigRobot.setTarget(littleRobot);  
 field.getCell(new MyPoint(1, 1)).setRobot(bigRobot);  
 field.getCell(new MyPoint(2, 1)).setLandscapeSegment(new IceSegment());  
  
 littleRobot.addRobotEndStepListener(bigRobot);  
  
 Assertions.*assertTrue*(littleRobot.makeStep(Direction.*SOUTH*));  
 Assertions.*assertNull*(field.getCell(new MyPoint(1, 1)).getRobot());  
 Assertions.*assertEquals*(bigRobot, field.getCell(new MyPoint(2, 1)).getRobot());  
 Assertions.*assertNull*(field.getCell(new MyPoint(3, 1)).getRobot());  
}

Тест на перемещение большого робота в ячейку с песком

@Test  
public void moveOnSandSegmentBigRobotTest() {  
 int exitPointX = 3, exitPointY = 3;  
 Field field = new Field(5, 5, new MyPoint(exitPointX, exitPointY));  
  
 LittleRobot littleRobot = new LittleRobot();  
 field.getCell(new MyPoint(5, 5)).setRobot(littleRobot);  
  
 PathFinder pathFinder = new PathFinder(field);  
 BigRobot bigRobot = new BigRobot();  
 bigRobot.setPathFinder(pathFinder);  
  
 bigRobot.setTarget(littleRobot);  
 field.getCell(new MyPoint(1, 1)).setRobot(bigRobot);  
 field.getCell(new MyPoint(2, 1)).setLandscapeSegment(new SandSegment());  
  
 littleRobot.addRobotEndStepListener(bigRobot);  
  
 Assertions.*assertTrue*(littleRobot.makeStep(Direction.*SOUTH*));  
 Assertions.*assertEquals*(field.getCell(new MyPoint(2, 1)), bigRobot.getPosition());  
 Assertions.*assertTrue*(littleRobot.makeStep(Direction.*NORTH*));  
 Assertions.*assertEquals*(field.getCell(new MyPoint(2, 1)), bigRobot.getPosition());  
 Assertions.*assertTrue*(littleRobot.makeStep(Direction.*SOUTH*));  
 Assertions.*assertEquals*(field.getCell(new MyPoint(3, 1)), bigRobot.getPosition());  
}

# 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/91