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 №01\_02)

Студент: Керимов Т.И.

Группа: ПрИн-367

Работа зачтена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ « »  июня  2024 г.

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

Волгоград 2024 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

и. о. зав. кафедрой \_\_\_\_\_\_\_\_\_\_ Сычев О.А.

**Задание**

**на курсовую работу**

Студент: Керимов Т.И.

Группа: ПрИн-367

1. Тема: «Проектированиепрограммы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №01\_02)

Утверждена приказом от « 15 » января 2024 г. № 37-ст

2. Срок представления работы к защите « 04 »   июня  2024 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 « » февраля 2024 г.

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

Задание принял к исполнению: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Марченко С.С.

« 14 » февраля 2024 г.

**Содержание**

[1 Формулировка задания 4](#_Toc168491878)

[2 Нефункциональные требования 5](#_Toc168491879)

[3 Первая итерация разработки 6](#_Toc168491880)

[3.1 Формулировка упрощённого варианта задания 6](#_Toc168491881)

[3.2 Функциональные требования (сценарии) 6](#_Toc168491882)

[3.3 Словарь предметной области 10](#_Toc168491883)

[3.4 Структура программы на уровне классов 12](#_Toc168491884)

[3.5 Типовые процессы в программе 14](#_Toc168491885)

[3.6 Человеко-машинное взаимодействие 22](#_Toc168491886)

[3.7 Реализация ключевых классов 25](#_Toc168491887)

[3.8 Реализация ключевых тестовых случаев 32](#_Toc168491888)

[4 Вторая итерация разработки 36](#_Toc168491889)

[4.1 Функциональные требования (сценарии) 36](#_Toc168491890)

[4.2 Словарь предметной области 37](#_Toc168491891)

[4.3 Структура программы на уровне классов 39](#_Toc168491892)

[4.4 Типовые процессы в программе 41](#_Toc168491893)

[4.5 Человеко-машинное взаимодействие 42](#_Toc168491894)

[4.6 Реализация ключевых классов 44](#_Toc168491895)

[4.7 Реализация ключевых тестовых случаев 46](#_Toc168491896)

[5 Список использованной литературы и других источников 48](#_Toc168491897)

**1 Формулировка задания**

Игра "Коза и капуста".

Правила игры:

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

- ящики и стены, возможно, образуют непроходимый лабиринт;

- цель козы — добраться до капусты, двигая ящики;

- коза может двигать только один ящик от себя или на себя;

- количество шагов козы ограничено.

**Подвариант 2:** необходимо предусмотреть в программе **точки расширения**, используя которые можно реализовать вариативную часть программы (в дополнение к базовой функциональности).

**Вариативность:** коза может двигать от себя больше одного ящика в зависимости от общего "веса" и своей "силы". "Сила" козы может изменяться по некоторому закону в течение некоторого, в том числе и бесконечного, периода времени в зависимости от типа съеденной травы. Разные виды травы должны быть визуально различимы.

**НЕ изменяя** ранее созданные классы, а используя точки расширения, реализовать: в течение 5 шагов коза может двигать сразу два ящика, если поест волшебной травы.

# 2 Нефункциональные требования

1. Программа должна быть реализована на языке Java SE 17 с использованием стандартных библиотек, в том числе, библиотеки Swing.
2. Форматирование исходного кода программы должно соответствовать Java Code Conventions, September 12, 1997.

# 3 Первая итерация разработки

**3.1 Формулировка упрощённого варианта задания**

Игра "Коза и капуста".

Правила игры:

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

-         ящики и стены, возможно, образуют непроходимый лабиринт;

-         цель козы — добраться до капусты, двигая ящики;

-         коза может двигать только один ящик от себя или на себя;

-         количество шагов козы ограничено.

**3.2 Функциональные требования (сценарии)**

**1) Сценарий** «Играть»

1. **По указанию пользователя,** Игра стартует.
2. **По указанию** Игры, Поле создаёт Ячейки и формирует из них себя. При этом Поле узнает позицию для Капусты у Загона.
3. **По указанию** Игры, создаетя Загон, после чего Загон создаёт и размещает на Поле Стены, Ящики, Капусту и Козу.
4. **В ответ на запрос** Игры, Поле **сообщает** о Козе, которая находится на нём.
5. **Делать** 
   1. **По указанию** **пользователя,** Коза двигает Ящик от себя или на себя, расходуя Шаги.
   2. **По указанию** **пользователя,** Коза перемещается на соседнюю Ячейку, расходуя Шаги.
   3. **Если** Коза находится в клетке с Капустой, она съедает Капусту
   4. Коза сообщает Игре о том, что съела Капусту
   5. Игра завершается победой пользователя

# Пока у Козы есть доступное количество шагов и Игра не завершена

1. **Сценарий завершается**.

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

1. Загон создаёт Козу.
2. Загон помещает на Поле созданную Козу.
3. Загон создаёт Капусту и помещает ее на Поле.
4. Загон создаёт и расставляет последовательности Стен по периметру Поля.
5. **Сценарий завершается**.

**3) Дочерний сценарий** «Коза перемещается на соседнюю Ячейку расходуя Шаги»

1. **В ответ на запрос** Козы, Загон **сообщает**, что Препятствия нет в направлении движения Козы
2. **Если** взята Коробка, Загон **сообщает** о том, что в соседней для Коробки Клетки в направлении движения нет Стены или Коробки
3. Если взята Коробка, Загон **извлекает** Коробку из соседней Клетки
4. **В ответ на запрос** Козы, соседняя **Клетка** в направлении движения Козы сообщает, свободна ли она
5. Коза решает, что может переместиться в соседнюю Клетку
6. **По указанию** Козы, шаг, необходимый для движения в направлении, изымается из общего доступного количества шагов
7. **По указанию** Козы, Клетка с Козой извлекает ее из себя
8. Соседняя Клетка помещает Козу в себя, так как Клетка пуста
9. Если была взята Коробка, то Коза перемещает Коробку в соседнюю свободную по направлению движения (к себе или от себя) Клетку
10. **Сценарий завершается**.

3.1) **Альтернативный сценарий** «Между соседней Клеткой и текущей находится Стена». Сценарий выполняется с п. 1 сценария 3

1. **В ответ на запрос** Козы, Соседняя Клетка сообщает, что Стена имеется в направлении движения Козы

2. Коза решает, что не может переместиться в соседнюю Клетку

3. **Сценарий завершается**

3.2) **Альтернативный сценарий** «В соседней для взятой Коробки Клетке находится Коробка или Стена». Сценарий выполняется с п. 1 сценария 2

1. **По указанию** Козы, соседняя Клетка сообщает, что в ней есть Коробка или Стена

2. Коза решает, что не может переместиться в соседнюю Клетку

3. **Сценарий завершается**

3.3) **Альтернативный сценарий** «У Козы недостаточно шагов для совершения хода». Сценарий выполняется с п. 6 сценария 3

1. Коза **сообщает**, что у нее недостаточно шагов для совершения хода

2. **Сценарий завершается**

**4) Альтернативный сценарий** «Перемещение Козы с Ящиком вдоль направления». Сценарий **выполняется с** п. 2 сценария 3.2

1. **По указанию** Пользователя, Коза сообщает наличие ящика.

2. Если направление, **указанное** Пользователем, вдоль, Коза запрашивает наличие препятствия у Клетки, в которую сместится ящик и Коза.

3. Если Клетка свободна, коза переходит в нее, коробка следом за ней.

3. **Сценарий завершается**

**5) Альтернативный сценарий** «Перемещение Козы с Ящиком не вдоль направления». Сценарий **выполняется с** п. 2 сценария 3.2

1. **По указанию** Пользователя, Коза сообщает наличие ящика.

2. Если направление, **указанное** Пользователем, не вдоль, Коза сообщает, что передвижение невозможно.

3. **Сценарий завершается**

**6) Дочерний сценарий** «Игра считает победителем Козу, забравшую Капусту»

1. Игра **запрашивает** у Поля Капусту.
2. Поле **запрашивает** Козу.
3. Поле **сообщает** Игре, что капусты нет на поле.
4. Игра считает победителем Козу, т.к. она съела Капусту.
5. **Сценарий завершается.**

**6.1) Альтернативный сценарий** «Поражение, так как Коза израсходовала все шаги, не достигнув капусту». Сценарий **выполняется с** п. 3 сценария 5

1. Игра **запрашивает** у Поля Капусту.
2. Поле **запрашивает** Козу.
3. Поле **сообщает** Игре, что капуста есть на поле.
4. Коза сообщает, что у нее нет шагов на совершение действия
5. Игра считает проигравшей Козу, т.к. она не забрала Капусту и не может двигаться.
6. **Сценарий завершается.**

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

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

**3.3 Словарь предметной области**

**Игра** – знает о Поле и Загоне. Игра инициирует создание Поля и расстановку всех сущностей на нем с помощью Загона. Игра определяет окончание игры.

**Поле** – прямоугольная область, состоящая из Ячеек. Между Ячейками может располагаться Стена. По границе Поля также могут располагаться Стены. Знает о Козе и Капусте, находящейся на Поле.

**Ячейка** – квадратная область Поля. Знает о четырёх соседних Ячейках и граничащих с ней Стенах. На ней может располагаться Коза, или Капуста, либо сразу оба.

**Стена** – непроходимое Препятствие для Козы, располагающееся между Ячейками.

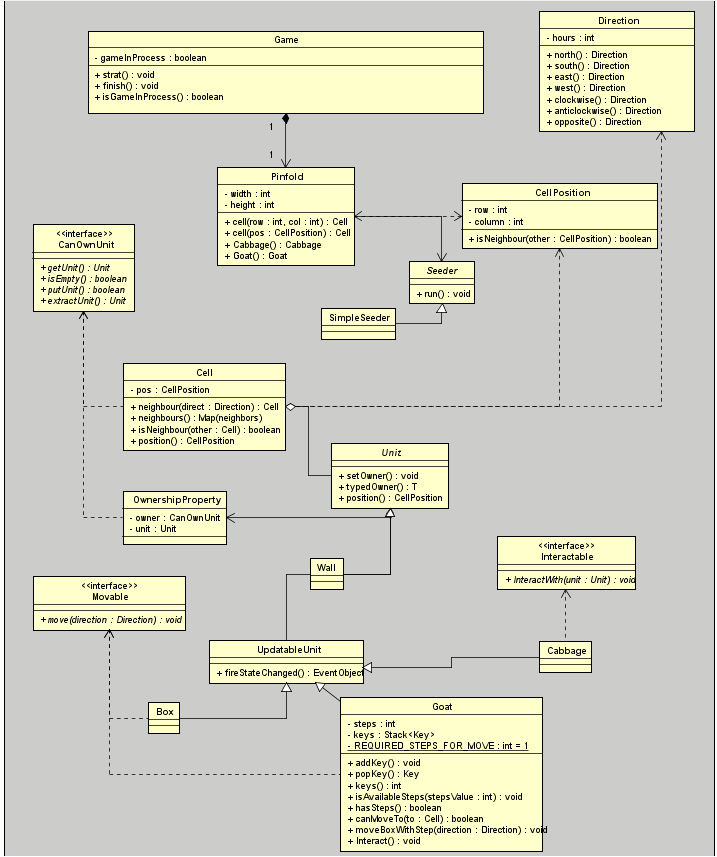
**Загон** – умеет создавать Стены, Козу и Капусту и размещать их на Поле в Ячейках. Позиции этих сущностей Загон определяет самостоятельно. Он знает расположение Капусты.

**Коза** – умеет *перемещаться* и двигать одну коробку на себя или от себя. Коза перемещается в соседнюю Ячейку, затрачивая Шаги, но не может пройти через Стену.

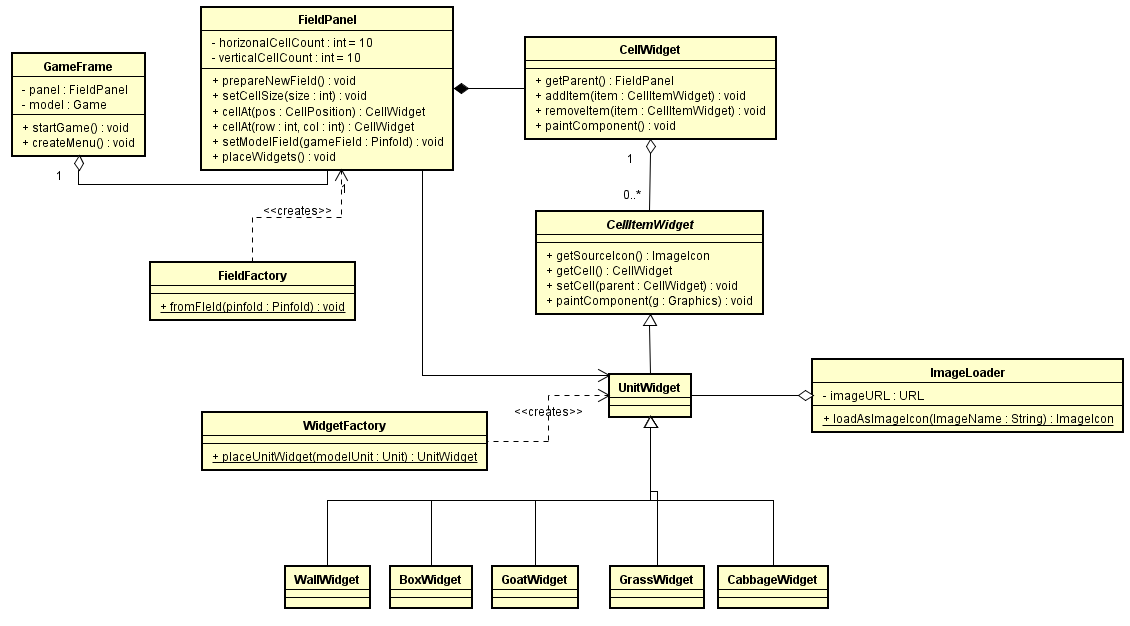
**Капуста** – разновидность объекта взаимодействия. Когда в Ячейку, где находится Капуста, становится Коза, Капуста уничтожается и сообщает о конце игры.

**Ящик** – разновидность объекта взаимодействия. Непреодолимое препятствие для Козы. Когда рядом Коза, она может взять Ящик и потянуть вперед или назад.

**3.4 Структура программы на уровне классов**

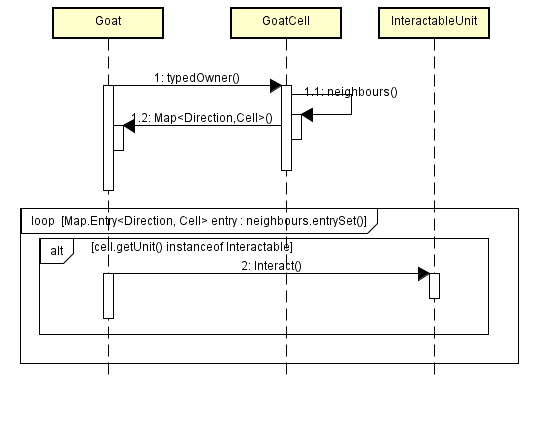


**Диаграмма классов модели**

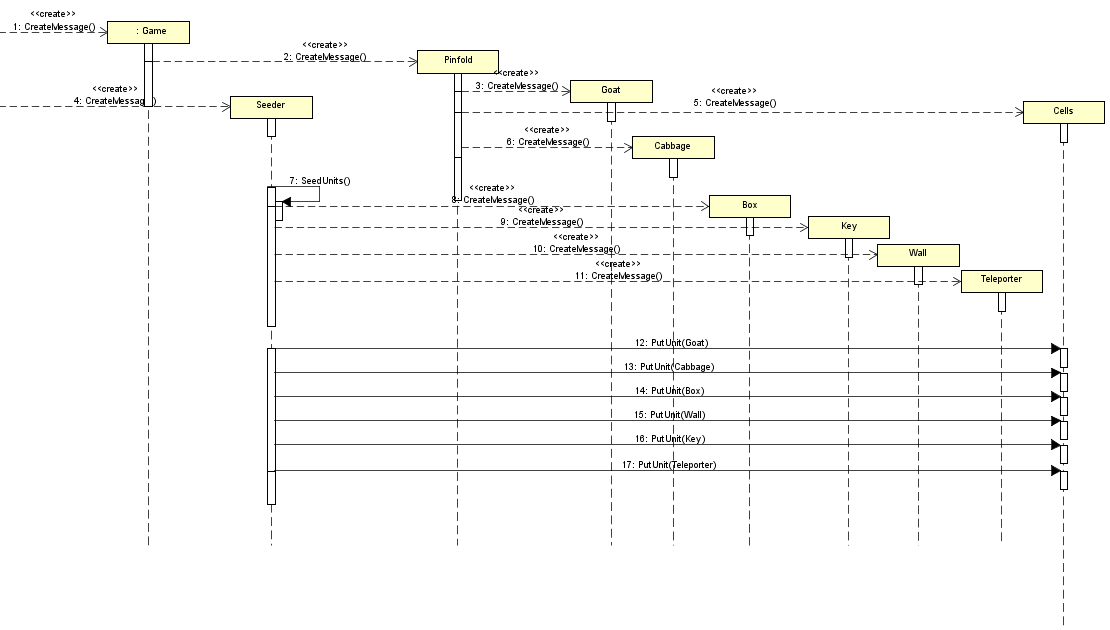


**Диаграмма классов представления**

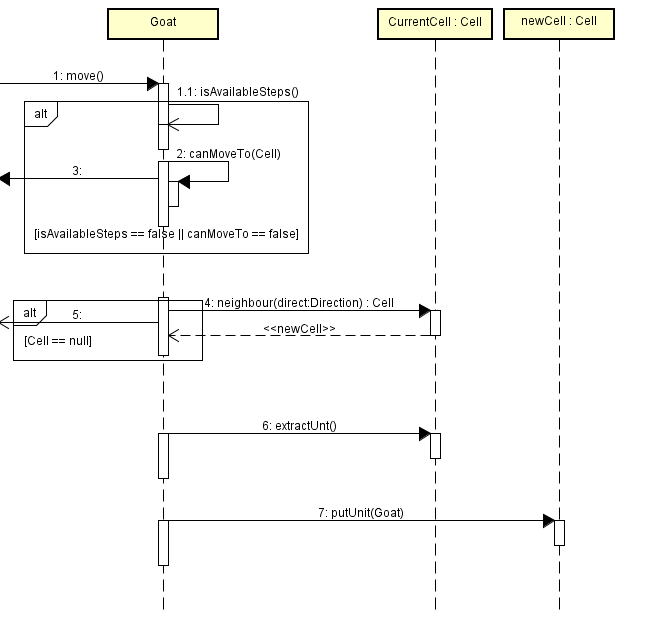
**3.5 Типовые процессы в программе**

****

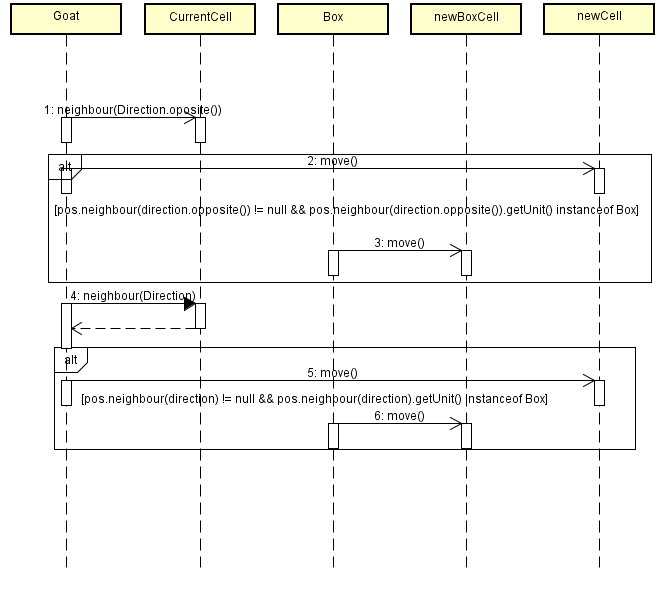
**Рисунок 2. Взаимодействие с объектами**

****

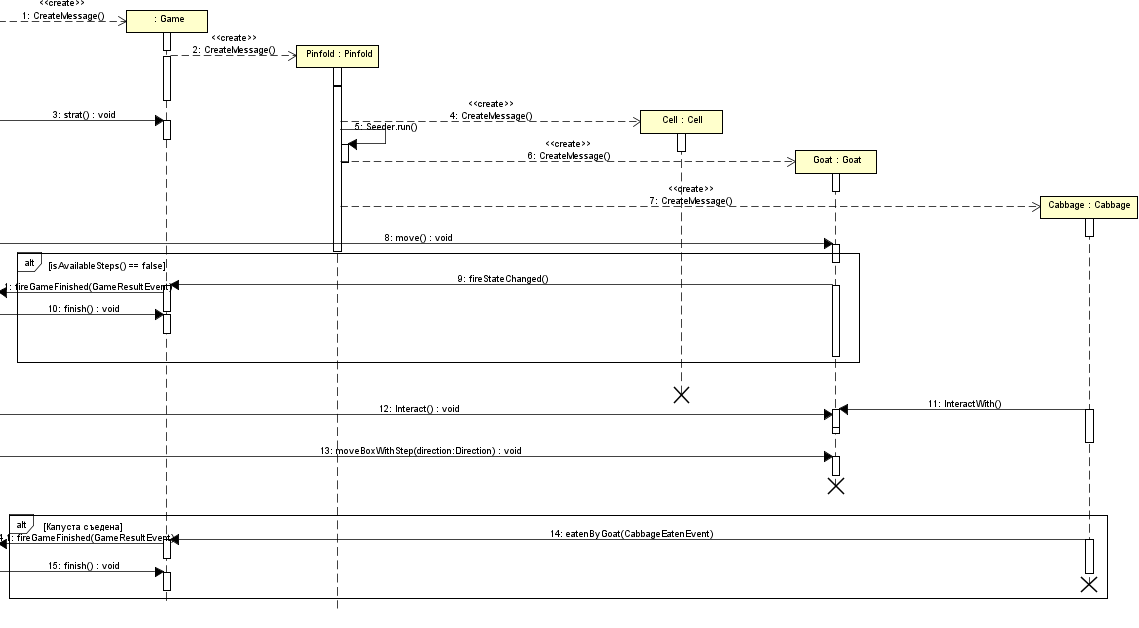
**Рисунок 3. Заполнение загона**

****

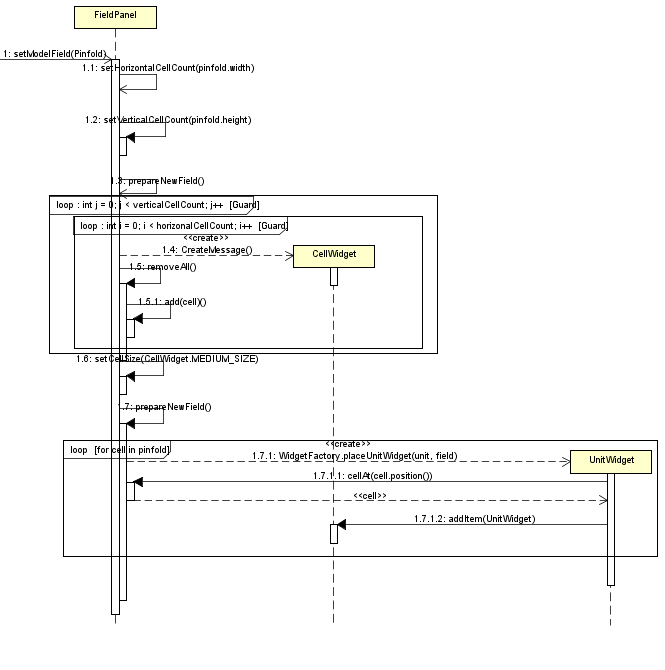
**Рисунок 4. Перемещение**

****

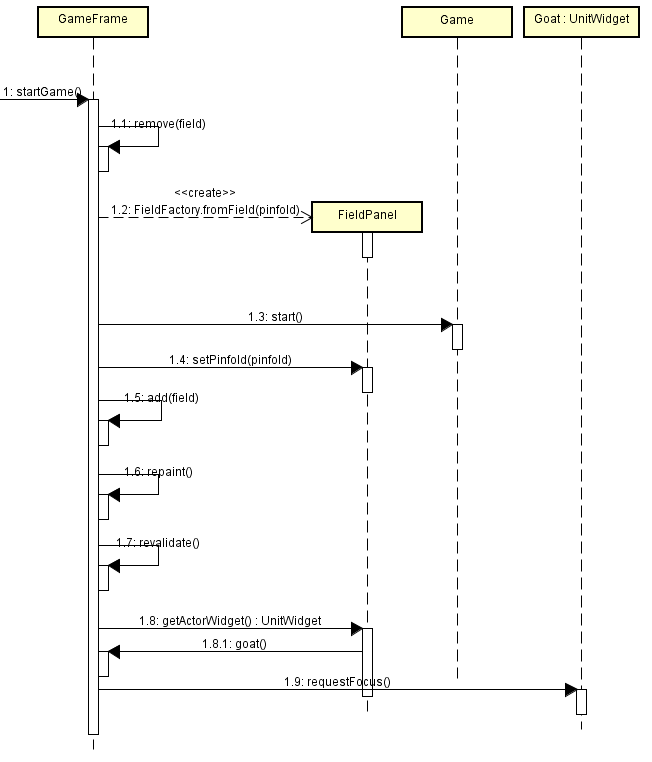
**Рисунок 5. Перемещение коробки**

****

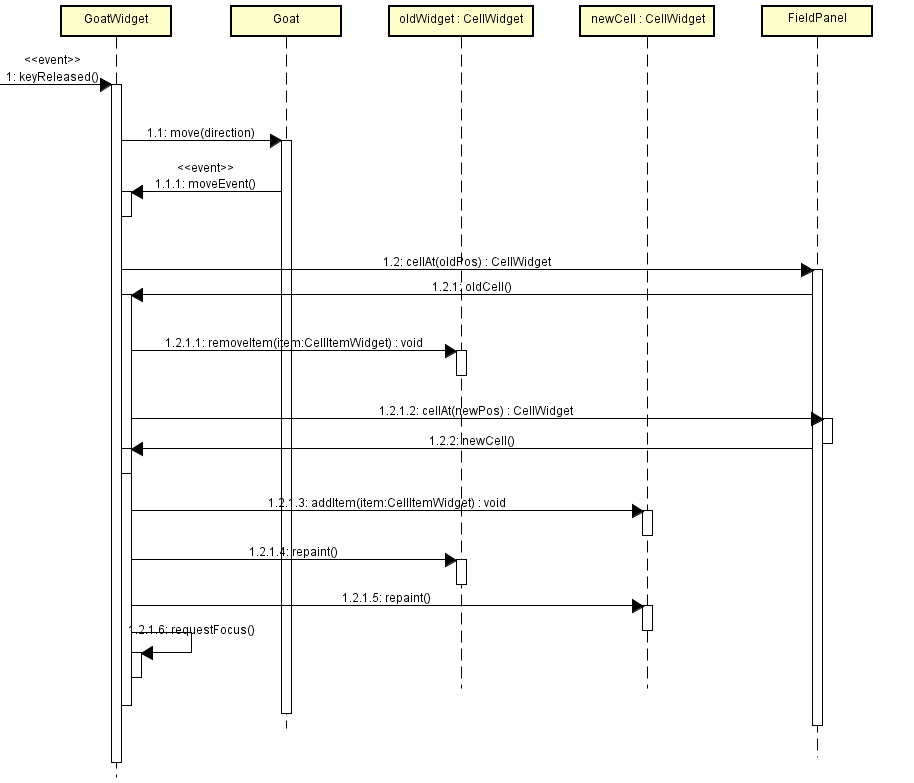
**Рисунок 6. Съедение капусты**

****

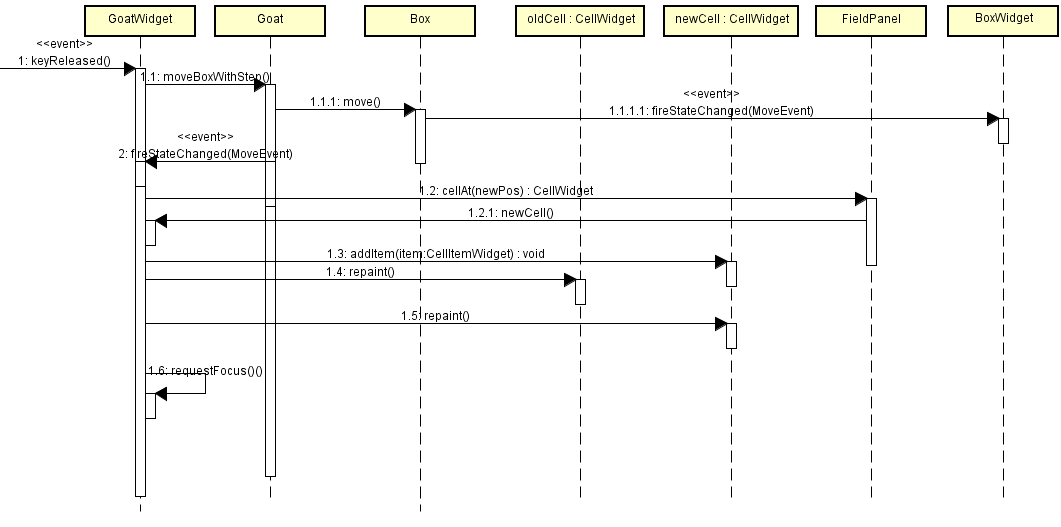
**Рисунок 7. Заполнение поля**

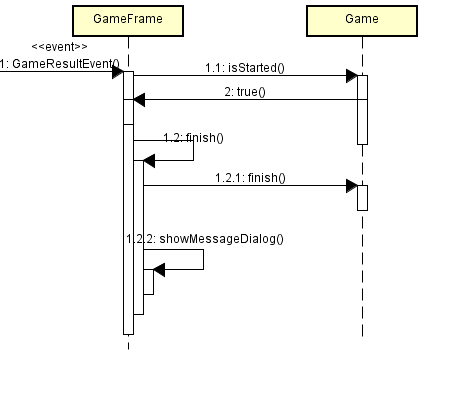
****

**Рисунок 8. Начало игры**

****

**Рисунок 9. Перемещение козы**

**Рисунок 10. Перемещение коробки**

****

**Рисунок 11. Съедение капусты**

**3.6 Человеко-машинное взаимодействие**

Общий вид главного экрана программы представлен ниже. На нём располагается игровой загон, на котором изображена Коза и Капуста, коробки и стены.

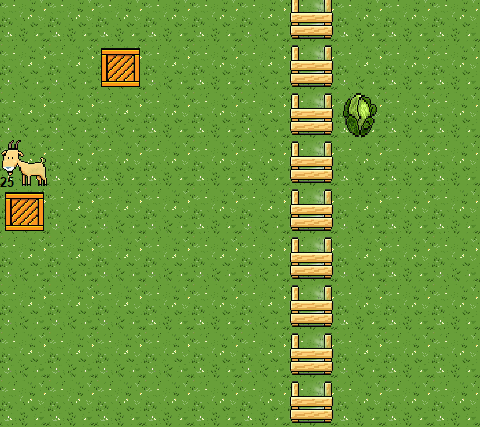


Рисунок 12. Вид загона в представлении

Управление Козой пользователь осуществляет с помощью клавиатуры.

Стрелка вверх – движение вверх.

Стрелка вниз – движение вниз.

Стрелка влево – движение влево.

Стрелка вправо – движение вправо.

Пробел – взаимодействие с объектами

Клавиша Ctrl удерживается и при совместном применении с любой из клавиш стрелок Коза тянет коробку к себе или толкает ее в направление движения.



Рисунок 13. Изображение Козы

В левом нижнем углу от Козы располагается надпись, информирующая о кол-ве шагов, доступных Козе, на данный момент.

Каждая клетка имеет свое фоновое изображение

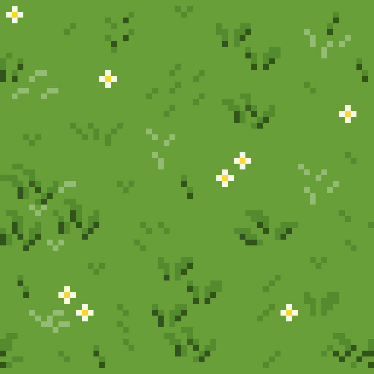


Рисунок 14. Фоновое изображение клетки загона

Также имеются изображения для Коробки, Стен и Капусты

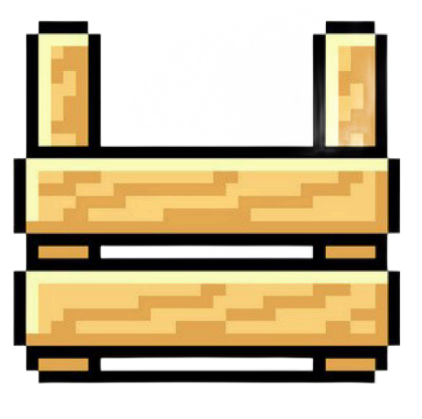
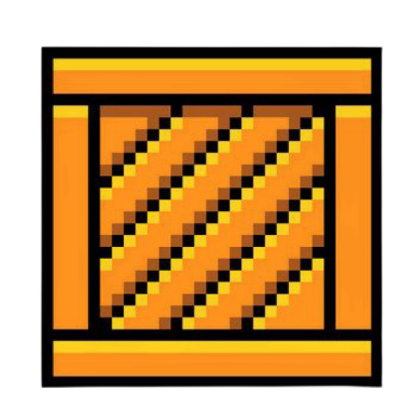


Рисунок 15. Изображение Стены



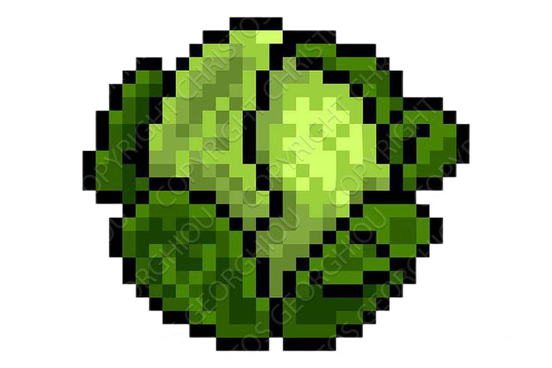
Рисунок 16. Изображение Коробки

Рисунок 17. Изображение Капусты

**3.7 Реализация ключевых классов**

// Коза

public class Goat **extends** UpdatableUnit **implements** Movable **{**

public Goat**()** **{**

**}**

public Goat**(**int steps**){**

setSteps**(**steps**);**

**}**

private int \_steps **=** 25**;**

public void setSteps**(**int steps**)** **{**

**this.**\_steps **=** steps**;**

**}**

private static final int REQUIRED\_STEPS\_FOR\_MOVE **=** 1**;**

private Stack**<**Key**>** \_keys **=** **new** Stack**<**Key**>();**

public void AddKey**(**Key key**)** **{**

\_keys**.**push**(**key**);**

**}**

public Key PopKey**()** **{**

Key key **=** \_keys**.**pop**();**

**if(**key **!=** **null)** **return** key**;**

**return** **null;**

**}**

public int Keys**()** **{** **return** \_keys**.**size**();** **}**

public int steps**()** **{**

**return** \_steps**;**

**}**

public boolean isAvailableSteps**(**int stepsValue**)** **{**

**return** stepsValue **<=** \_steps**;**

**}**

protected int reduceSteps**(**int stepsValue**)** **{**

int retrievedSteps **=** Math**.**min**(**\_steps**,** stepsValue**);**

\_steps **-=** retrievedSteps**;**

**return** retrievedSteps**;**

**}**

public boolean hasSteps**()** **{** **return** \_steps **>=** REQUIRED\_STEPS\_FOR\_MOVE**;}**

// --------------------------- Перемещение ------------------------------

public boolean canMoveTo**(**Cell to**)** **{**

**return** to**.**isEmpty**();**

**}**

@Override

public void move**(**Direction direct**)** **{**

Cell pos **=** typedOwner**();**

CellPosition prevPosition **=** pos**.**position**();**

**if(!**isAvailableSteps**(**REQUIRED\_STEPS\_FOR\_MOVE**))** **{**

**return;**

**}**

Cell newPos **=** pos**.**neighbour**(**direct**);**

**if(**newPos **==** **null)** **{**

**return;**

**}**

**if(!**canMoveTo**(**newPos**))** **{**

**return;**

**}**

Unit unit **=** pos**.**extractUnit**();**

newPos**.**putUnit**(**unit**);**

reduceSteps**(**REQUIRED\_STEPS\_FOR\_MOVE**);**

fireStateChanged**(new** MoveEvent**(this,** newPos**.**position**(),** prevPosition**));**

**}**

public void MoveBoxWithStep**(**Direction direction**)**

**{**

Cell pos **=** typedOwner**();**

Box boxForward **=** **null,** boxBehind **=** **null;**

**if(**pos**.**neighbour**(**direction**.**opposite**())** **!=** **null** **&&** pos**.**neighbour**(**direction**.**opposite**()).**getUnit**()** **instanceof** Box**)**

**{**

boxBehind **=** **(**Box**)** pos**.**neighbour**(**direction**.**opposite**()).**getUnit**();**

System**.**out**.**println**(**"boxBehind: " **+** boxBehind**);**

**}**

**if(**boxBehind **!=** **null)**

**{**

move**(**direction**);**

boxBehind**.**move**(**direction**);**

**}**

**if(**pos**.**neighbour**(**direction**)** **!=** **null** **&&** pos**.**neighbour**(**direction**).**getUnit**()** **instanceof** Box**)**

**{**

boxForward **=** **(**Box**)** pos**.**neighbour**(**direction**).**getUnit**();**

System**.**out**.**println**(**"boxForward: " **+** boxForward**);**

**}**

**if(**boxForward **!=** **null)** **{**

boxForward**.**move**(**direction**);**

move**(**direction**);**

**}**

**}**

public void Interact**()**

**{**

Cell c **=** typedOwner**();**

Map**<**Direction**,** Cell**>** neighbours **=** c**.**neighbors**();**

**for** **(**Map**.**Entry**<**Direction**,** Cell**>** entry **:** neighbours**.**entrySet**())** **{**

Cell cell **=** entry**.**getValue**();**

System**.**out**.**println**(**cell**.**position**().**column**()** **+** " " **+** cell**.**position**().**row**());**

**if(**cell**.**getUnit**()** **instanceof** Interactable**)** **{**

**((**Interactable**)** cell**.**getUnit**()).**InteractWith**(this);**

**return;**

**}**

**}**

**}**

@Override

public String toString**()** **{**

String msg**;**

msg **=** "G(" **+** steps**()** **+** ")"**;**

**return** msg**;**

**}**

**}**

// Ячейка поля

public class Cell **implements** CanOwnUnit **{**

private final CellPosition \_pos**;**

public CellPosition position**()** **{**

**return** \_pos**;**

**}**

public Cell**(**CellPosition position**)** **{**

\_pos **=** position**;**

**}**

private final Map**<**Direction**,** Cell**>** \_neighbors **=** **new** HashMap**<>();**

public Cell neighbour**(**Direction direct**)** **{**

**if(**\_neighbors**.**containsKey**(**direct**))** **{**

**return** \_neighbors**.**get**(**direct**);**

**}**

**return** **null;**

**}**

public Map**<**Direction**,** Cell**>** neighbors**()** **{**

**return** Collections**.**unmodifiableMap**(**\_neighbors**);**

**}**

void setNeighbor**(**Direction direct**,** Cell neighbor**)** **{**

**if(**neighbor **!=** **this** **&&** **!**isNeighbor**(**neighbor**))** **{**

\_neighbors**.**put**(**direct**,** neighbor**);**

neighbor**.**setNeighbor**(**direct**.**opposite**(),** **this);**

**}**

**}**

public boolean isNeighbor**(**Cell other**)** **{**

**return** \_neighbors**.**containsValue**(**other**);**

**}**

private final OwnershipProperty \_ownership **=** **new** OwnershipProperty**(this);**

public Unit getUnit**()** **{**

**return** \_ownership**.**getUnit**();**

**}**

public boolean isEmpty**()** **{**

**return** \_ownership**.**isEmpty**();**

**}**

public boolean putUnit**(**Unit unit**)** **{** **return** \_ownership**.**putUnit**(**unit**);** **}**

public Unit extractUnit**()** **{** **return** \_ownership**.**extractUnit**();** **}**

**}**

public class Game **implements** StateChangeListener **{**

private Pinfold \_field**;**

private boolean \_gameInProcess **=** **false;**

private ArrayList**<**GameStateListener**>** \_listeners **=** **new** ArrayList**<>();**

public boolean is\_gameInProcess**()** **{return** \_gameInProcess**;}**

public void finish**()** **{**\_gameInProcess **=** **false;}**

public Pinfold getField**()** **{return** \_field**;}**

public void start**(**int width**,**int height**,** Seeder seeder**)** **{**

\_field **=** **new** Pinfold**(**height**,** width**,** seeder**);**

\_field**.**Goat**().**addListener**(this);**

\_field**.**Cabbage**().**addCabbageEatenListener**(new** CabbageEatenListener**()** **{**

@Override

public void cabbageEaten**(**CabbageEatenEvent event**)** **{**

fireGameFinished**(true);**

**}**

**});**

\_gameInProcess **=** **true;**

**}**

@Override

public void stateChanged**(**EventObject e**)** **{**

**if** **(**e**.**getSource**()** **instanceof** Goat**)**

**{**

**if(!((**Goat**)** e**.**getSource**()).**hasSteps**())**

fireGameFinished**(false);**

**}**

**}**

private void fireGameFinished**(**boolean result**)** **{**

**for(** GameStateListener listener **:** \_listeners**)**

listener**.**GameFinished**(new** GameResultEvent**(**result**));**

**}**

public void addGameStateListener**(**GameStateListener listener**)** **{**

\_listeners**.**add**(**listener**);**

**}**

public void removeGameStateListener**(**GameStateListener listener**)** **{**

\_listeners**.**add**(**listener**);**

**}**

public boolean hasGameStateListeners**()** **{return** **!**\_listeners**.**isEmpty**();}**

**}**

// Прямоугольное поле, состоящее из ячеек

public class Pinfold **implements** Iterable**<**Cell**>,** CabbageEatenListener **{**

// ---------------------- Размеры -----------------------------

private final int \_width**;**

private final int \_height**;**

public int width**()** **{**

**return** \_width**;**

**}**

public int height**()** **{**

**return** \_height**;**

**}**

// --------------------------- Ячейки ----------------------

private final HashMap**<**CellPosition**,** Cell**>** \_cells **=** **new** HashMap**<>();**

public Cell cell**(**CellPosition pos**)** **{**

**return** \_cells**.**get**(** pos **);**

**}**

public Cell cell**(**int row**,** int col**)** **{**

**return** cell**(new** CellPosition**(**row**,** col**));**

**}**

@Override

public Iterator**<**Cell**>** iterator**()** **{**

**return** **new** GameFieldIterator**(** **this** **);**

**}**

// ---------------------------- Порождение ---------------------

public Pinfold**(**int height**,** int width**,** Seeder seeder**)** **{**

**if** **(**width **<=** 0 **||** height **<=** 0**)** **{**

**throw** **new** IllegalArgumentException**();**

**}**

\_width **=** width**;**

\_height **=** height**;**

buildField**();**

\_cabbage**.**addCabbageEatenListener**(this);**

seeder**.**setField**(this);**

seeder**.**run**();**

**}**

private void buildField**()** **{**

// Создаем ячейки

**for** **(**int row **=** 0**;** row **<** height**();** row**++)** **{**

**for** **(**int col **=** 0**;** col **<** width**();** col**++)** **{**

CellPosition pos **=** **new** CellPosition**(**row**,** col**);**

\_cells**.**put**(**pos**,** **new** Cell**(**pos**));**

**}**

**}**

// Связываем ячейки

**for** **(**int row **=** 0**;** row **<** height**();** row**++)** **{**

**for** **(**int col **=** 0**;** col **<** width**();** col**++)** **{**

Cell cell **=** cell**(**row**,** col**);**

**if** **(**height**()** **>** 1 **&&** row **<** height**()** **-** 1**)** **{**

cell**.**setNeighbor**(**Direction**.**south**(),** cell**(**row **+** 1**,** col**));**

**}**

**if** **(**row **>** 0**)** **{**

cell**.**setNeighbor**(**Direction**.**north**(),** cell**(**row **-** 1**,** col**));**

**}**

**if** **(**width**()** **>** 1 **&&** col **<** width**()** **-** 1**)** **{**

cell**.**setNeighbor**(**Direction**.**east**(),** cell**(**row**,** col **+** 1**));**

**}**

**if** **(**col **>** 0**)** **{**

cell**.**setNeighbor**(**Direction**.**west**(),** cell**(**row**,** col **-** 1**));**

**}**

**}**

**}**

**}**

private final Goat \_goat **=** **new** Goat**();**

public Goat Goat**()** **{**

**return** \_goat**;**

**}**

private final Cabbage \_cabbage **=** **new** Cabbage**();**

public Cabbage Cabbage**()** **{return** \_cabbage**;}**

@Override

public void cabbageEaten**(**CabbageEatenEvent event**)** **{**

// Обработка события съедания капусты

System**.**out**.**println**(**"Коза съела капусту!"**);**

**}**

private class GameFieldIterator **implements** Iterator**<**Cell**>** **{**

private Cell \_cell **=** **null;**

private final Pinfold \_field**;**

public GameFieldIterator**(**Pinfold field**)** **{**

\_field **=** field**;**

**}**

@Override

public boolean hasNext**()** **{**

**return** nextCell**(** \_cell **)** **!=** **null;**

**}**

@Override

public Cell next**()** **{**

\_cell **=** nextCell**(**\_cell**);**

**return** \_cell**;**

**}**

private Cell nextCell**(**Cell cell**)** **{**

Cell next\_cell **=** **null;**

**if(**cell **==** **null)** **{**

next\_cell **=** \_field**.**cell**(**0**,** 0**);**

**}** **else** **{**

next\_cell **=** cell**.**neighbour**(** Direction**.**east**()** **);**

**if(** next\_cell **==** **null** **&&** cell**.**position**().**row**()** **<** \_field**.**height**()-**1 **)** **{**

next\_cell **=** \_field**.**cell**(** cell**.**position**().**row**()** **+** 1 **,** 0 **);**

**}**

**}**

**return** next\_cell**;**

**}**

**}**

**}**

**3.8 Реализация ключевых тестовых случаев**

public class PinfoldTest **{**

@Test

public void testCellCreation**()** **{**

Pinfold pinfold **=** **new** Pinfold**(**3**,** 3**,** **new** TestSeeder**());**

**for** **(**int row **=** 0**;** row **<** pinfold**.**height**();** row**++)** **{**

**for** **(**int col **=** 0**;** col **<** pinfold**.**width**();** col**++)** **{**

CellPosition pos **=** **new** CellPosition**(**row**,** col**);**

assertNotNull**(**pinfold**.**cell**(**pos**));**

**}**

**}**

**}**

@Test

public void testGoatCreation**()** **{**

Pinfold pinfold **=** **new** Pinfold**(**3**,** 3**,** **new** TestSeeder**());**

assertNotNull**(**pinfold**.**Goat**());**

**for** **(**Cell cell **:** pinfold**)** **{**

assertNotEquals**(**cell**.**getUnit**(),** pinfold**.**Goat**());**

**}**

**}**

@Test

public void testCabbageCreation**()** **{**

Pinfold pinfold **=** **new** Pinfold**(**3**,** 3**,** **new** TestSeeder**());**

assertNotNull**(**pinfold**.**Cabbage**());**

**for** **(**Cell cell **:** pinfold**)** **{**

assertNotEquals**(**cell**.**getUnit**(),** pinfold**.**Cabbage**());**

**}**

**}**

@Test

public void testIterator**()** **{**

Pinfold pinfold **=** **new** Pinfold**(**3**,** 3**,** **new** TestSeeder**());**

int count **=** 0**;**

**for** **(**Cell cell **:** pinfold**)** **{**

assertNotNull**(**cell**);**

count**++;**

**}**

assertEquals**(**9**,** count**);**

**}**

@Test

public void neighbourTest**()** **{**

Pinfold field **=** **new** Pinfold**(**6**,**6**,** **new** TestSeeder**());**

Assertions**.**assertTrue**(**field**.**cell**(**3**,** 3**).**isNeighbor**(**field**.**cell**(**3**,**2**)));**

Assertions**.**assertTrue**(**field**.**cell**(**3**,** 3**).**isNeighbor**(**field**.**cell**(**3**,**4**)));**

Assertions**.**assertTrue**(**field**.**cell**(**3**,** 3**).**isNeighbor**(**field**.**cell**(**4**,**3**)));**

Assertions**.**assertTrue**(**field**.**cell**(**3**,** 3**).**isNeighbor**(**field**.**cell**(**2**,**3**)));**

**}**

@Test

public void paddockFillTest**()** **{**

Pinfold field **=** **new** Pinfold**(**6**,**6**,** **new** TestSeeder**());**

Box box **=** **new** Box**();**

field**.**cell**(**3**,** 3**).**putUnit**(**box**);**

Assertions**.**assertFalse**(**field**.**cell**(**3**,**3**).**isEmpty**());**

Assertions**.**assertInstanceOf**(**Box**.**class**,** field**.**cell**(**3**,** 3**).**getUnit**());**

**}**

**}**

public class GoatTest **{**

@Test

public void moveToDirectionsTest**()** **{**

Goat gt **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**3**,** 3**).**putUnit**(**gt**);**

gt**.**move**(**Direction**.**east**());**

Assertions**.**assertEquals**(**gt**.**position**(),** field**.**cell**(**3**,**4**).**position**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**3**,** 4**));**

gt**.**move**(**Direction**.**west**());**

Assertions**.**assertEquals**(**gt**.**position**(),** field**.**cell**(**3**,**3**).**position**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**3**,** 3**));**

gt**.**move**(**Direction**.**south**());**

Assertions**.**assertEquals**(**gt**.**position**(),** field**.**cell**(**4**,**3**).**position**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**4**,** 3**));**

gt**.**move**(**Direction**.**north**());**

Assertions**.**assertEquals**(**gt**.**position**(),** field**.**cell**(**3**,**3**).**position**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**3**,** 3**));**

**}**

@Test

public void connectionWithCellTest**()** **{**

Goat gt **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**3**,** 3**).**putUnit**(**gt**);**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**3**,** 3**));**

**}**

@Test

public void goatCollideWithInnerWallTest**()** **{**

Goat gt **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**4**,** 3**).**putUnit**(**gt**);**

field**.**cell**(**5**,** 3**).**putUnit**(new** Wall**());**

gt**.**move**(**Direction**.**south**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**4**,** 3**));**

**}**

@Test

public void GoatInteraction**()** **{**

Goat g **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**3**,** 3**).**putUnit**(**g**);**

field**.**cell**(**3**,** 4**).**putUnit**(new** Cabbage**());**

g**.**Interact**();**

**for** **(**Cell cell **:** field**)** **{**

assertNotEquals**(**cell**.**getUnit**(),** field**.**Cabbage**());**

**}**

**}**

@Test

public void goatCollideWithOuterWallTest**()** **{**

Goat gt **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**1**,** 1**).**putUnit**(**gt**);**

gt**.**move**(**Direction**.**west**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**1**,** 1**));**

**}**

@Test

public void goatCollideWithBoxNearOuterWallTest**()** **{**

Goat gt **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**9**,** 8**).**putUnit**(**gt**);**

field**.**cell**(**9**,** 9**).**putUnit**(new** Box**());**

gt**.**move**(**Direction**.**east**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**9**,** 8**));**

**}**

**}**

public class CellTest **{**

@Test

public void setNewWallTest**()** **{**

Cell cell **=** **new** Cell**(null);**

cell**.**putUnit**(new** Wall**());**

Assertions**.**assertFalse**(**cell**.**isEmpty**());**

Assertions**.**assertInstanceOf**(**Wall**.**class**,** cell**.**getUnit**());**

**}**

@Test

public void setExistingWallTest**()** **{**

Cell cell **=** **new** Cell**(null);**

cell**.**putUnit**(new** Wall**());**

cell**.**putUnit**(new** Cabbage**());**

Assertions**.**assertInstanceOf**(**Wall**.**class**,** cell**.**getUnit**());**

**}**

@Test

public void removeWallTest**()** **{**

Cell cell **=** **new** Cell**(null);**

Wall wall **=** **new** Wall**();**

Assertions**.**assertTrue**(**cell**.**putUnit**(**wall**));**

cell**.**extractUnit**();**

Assertions**.**assertTrue**(**cell**.**isEmpty**());**

**}**

@Test

public void neighbourOnBorderTest**()** **{**

Pinfold field **=** **new** Pinfold**(**5 **,**5**,** **new** TestSeeder**());**

Assertions**.**assertNull**(**field**.**cell**(**0**,** 4**).**neighbour**(**Model**.**gamefield**.**Direction**.**east**()));**

Assertions**.**assertNull**(**field**.**cell**(**0**,** 0**).**neighbour**(**Model**.**gamefield**.**Direction**.**west**()));**

Assertions**.**assertNull**(**field**.**cell**(**4**,** 4**).**neighbour**(**Model**.**gamefield**.**Direction**.**south**()));**

Assertions**.**assertNull**(**field**.**cell**(**0**,** 0**).**neighbour**(**Model**.**gamefield**.**Direction**.**north**()));**

**}**

**}**

public class GameTest **{**

@Test

public void GameWinTest**()** **{**

Game game **=** **new** Game**();**

game**.**start**(**10**,** 10**,** **new** SimpleSeeder**());**

game**.**getField**().**Goat**().**addListener**(**

**new** StateChangeListener**()** **{**

@Override

public void stateChanged**(**EventObject e**)** **{**

**if(**e **instanceof** CabbageEatenEvent**)** **{**

Assertions**.**assertEquals**(**game**.**getField**().**Cabbage**().**position**(),**

**((**CabbageEatenEvent**)** e**).**getCabbagePosition**());**

**}**

**}**

**}**

**);**

game**.**addGameStateListener**(** result **->** **{**

Assertions**.**assertTrue**(**result**.**GetResult**());**

**});**

Assertions**.**assertTrue**(**game**.**is\_gameInProcess**());**

game**.**getField**().**Goat**().**Interact**();**

**}**

@Test

public void GameDefeatTest**()** **{**

Game game **=** **new** Game**();**

game**.**start**(**10**,** 10**,** **new** SimpleSeeder**());**

game**.**addGameStateListener**(** result **->** **{**

Assertions**.**assertFalse**(**result**.**GetResult**());**

**});**

game**.**getField**().**Goat**().**setSteps**(**1**);**

game**.**getField**().**Goat**().**move**(**Direction**.**east**());**

**}**

**}**

# 4 Вторая итерация разработки

**4.1 Функциональные требования (сценарии)**

1. **Сценарий** «Коза пытается сдвинуть несколько коробок вряд»

1. Коза рассчитывает количество ящиков перед ней и их общий вес

2. Если у Козы Силы больше, чем ссумарный Вес коробок, то она сдвигает их.

* 1. **Альтернативный Сценарий** «Козе не хватает сил сдвинуть ящики перед ней»

1. Коза рассчитывает количество ящиков перед ней и их общий вес

2. Если у Козы Силы меньше, чем суммарный Вес коробок, то она не сдвигает их.

**4.2 Словарь предметной области**

**Игра** – знает о Поле и Загоне. Игра инициирует создание Поля и расстановку всех сущностей на нем с помощью Загона. Игра определяет окончание игры.

**Поле** – прямоугольная область, состоящая из Ячеек. Между Ячейками может располагаться Стена. По границе Поля также могут располагаться Стены. Знает о Козе и Капусте, находящейся на Поле.

**Ячейка** – квадратная область Поля. Знает о четырёх соседних Ячейках и граничащих с ней Стенах. На ней может располагаться Коза, или Капуста, либо сразу оба.

**Стена** – непроходимое Препятствие для Козы, располагающееся между Ячейками.

**Загон** – умеет создавать Стены, Козу и Капусту и размещать их на Поле в Ячейках. Позиции этих сущностей Загон определяет самостоятельно. Он знает расположение Капусты.

**Коза** – умеет *перемещаться* и двигать одну коробку на себя или от себя. Коза перемещается в соседнюю Ячейку, затрачивая Шаги, но не может пройти через Стену.

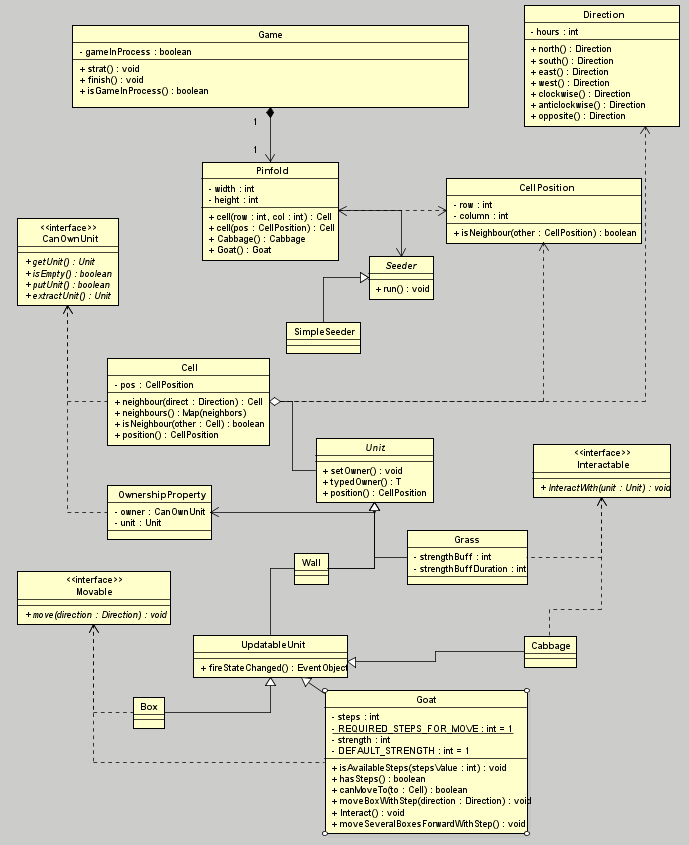
**Капуста** – разновидность объекта взаимодействия. Когда в Ячейку, где находится Капуста, становится Коза, Капуста уничтожается и сообщает о конце игры.

**Ящик** – разновидность объекта взаимодействия. Непреодолимое препятствие для Козы. Когда рядом Коза, она может взять Ящик и потянуть вперед или назад.

**Трава** – разновидность объекта взаимодействия. Непреодолимое препятствие для Козы. Когда рядом Коза, она может съесть Траву и получить силы.

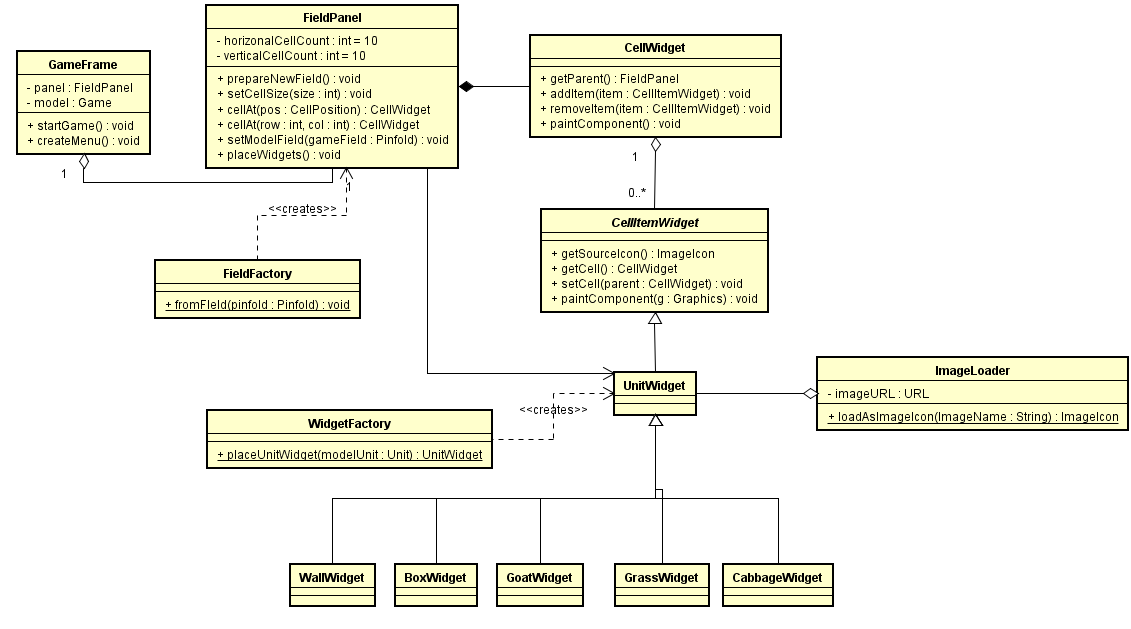
**Сила** – значение, исходя из которого можно узнать, сколько коробок вряд сможет сдвинуть Коза за раз.

**Вес** – значение, характеризующее вес каждой коробки.



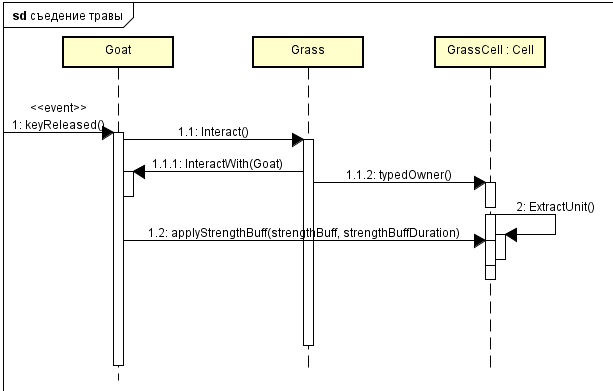
**4.3 Структура программы на уровне классов**

**Рисунок 18. Диаграмма классов модели**

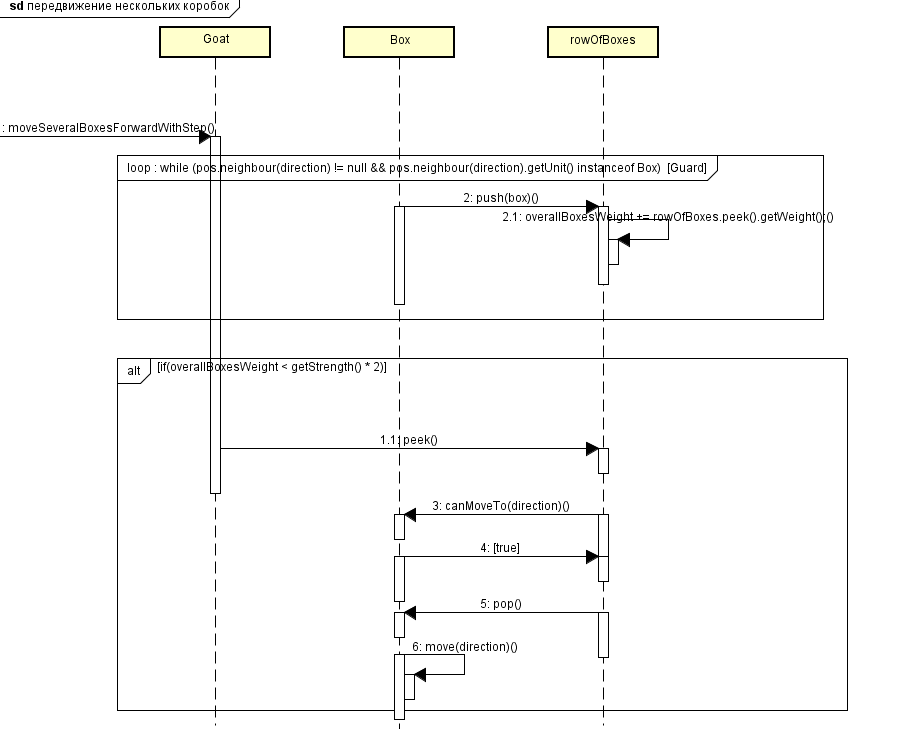


**Диаграмма классов представления**

**4.4 Типовые процессы в программе**



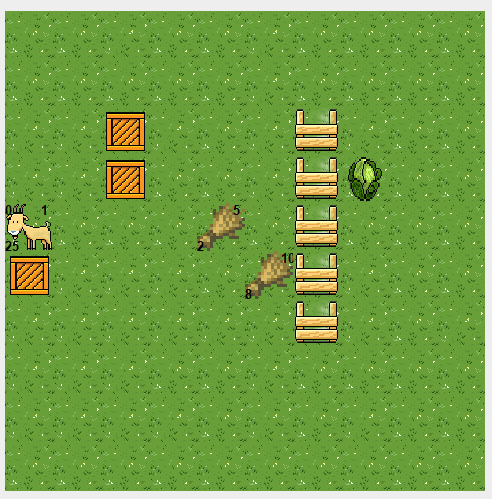
**Рисунок 19. Съедение травы**



**Рисунок 20 .** **передвижение нескольких коробок**

**4.5 Человеко-машинное взаимодействие**

Общий вид главного экрана программы представлен ниже. На нём располагается игровой загон, на котором изображена Коза и Капуста, коробки и стены.



Есть на поле ямы и лопаты для перемещения по ним

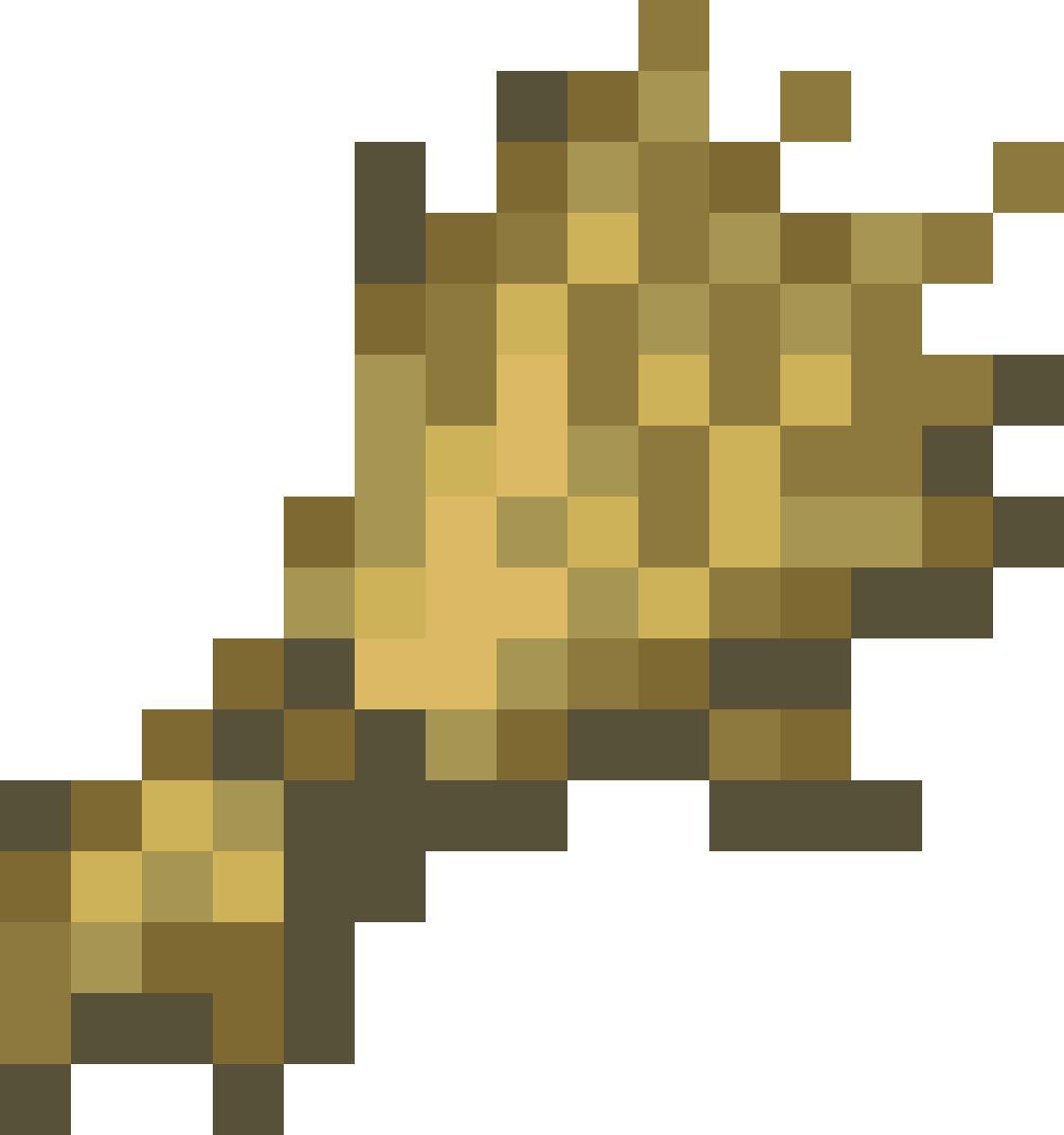


Рисунок 21. Изображение травы

**4.6 Реализация ключевых классов**

package Model**.**units**;**

**import** Model**.**gamefield**.**Cell**;**

**import** Model**.**ownership**.**Unit**;**

**import** Model**.**updatableunit**.**Interactable**;**

public class Grass **extends** Unit **implements** Interactable **{**

final int strengthBuff**;**

public int getStrengthBuff**()** **{**

**return** strengthBuff**;**

**}**

final int strengthBuffDuration**;**

public int getStrengthBuffDuration**()** **{**

**return** strengthBuffDuration**;**

**}**

public Grass**()**

**{**

**this.**strengthBuff **=** 2**;**

**this.**strengthBuffDuration **=** 5**;**

**}**

public Grass**(**int strengthBuff**,** int strengthBuffDuration**)**

**{**

**this.**strengthBuff **=** strengthBuff**;**

**this.**strengthBuffDuration **=** strengthBuffDuration**;**

**}**

@Override

public void InteractWith**(**Unit unit**)** **{**

**if(**unit **instanceof** Goat**)** **{**

Goat goat **=** **(**Goat**)**unit**;**

//TODO

goat**.**applyStrengthBuff**(**strengthBuff**,** strengthBuffDuration**);**

**((**Cell**)** typedOwner**()).**extractUnit**();**

System**.**out**.**println**(**"Goat strength: " **+** goat**.**getStrength**()** **+** " Goat buff duration: " **+** goat**.**getStrengthBuffDuration**());**

**}**

**}**

**}**

package Model**.**units**;**

**import** Model**.**gamefield**.**CellPosition**;**

**import** Model**.**gamefield**.**Direction**;**

**import** Model**.**gamefield**.**Cell**;**

**import** Model**.**ownership**.**Unit**;**

**import** Model**.**updatableunit**.**Interactable**;**

**import** Model**.**updatableunit**.**Movable**;**

**import** Model**.**updatableunit**.**MoveEvent**;**

**import** Model**.**updatableunit**.**UpdatableUnit**;**

**import** java**.**util**.**Map**;**

**import** java**.**util**.**Stack**;**

// Коза

public class Goat **extends** UpdatableUnit **implements** Movable **{**

public Goat**()** **{**

**}**

public Goat**(**int steps**)** **{**

setSteps**(**steps**);**

**}**

private static final int DEFAULT\_STRENGTH **=** 1**;**

private int \_steps **=** 25**;**

private int \_strength **=** DEFAULT\_STRENGTH**;**

private int \_strengthBuffDuration **=** 0**;**

public int getStrength**()** **{**

**return** \_strength**;**

**}**

public int getStrengthBuffDuration**()** **{**

**return** \_strengthBuffDuration**;**

**}**

public void applyStrengthBuff**(**int strengthBuff**,** int strengthBuffDuration**)** **{**

\_strength **=** strengthBuff**;**

\_strengthBuffDuration **=** strengthBuffDuration**;**

**}**

public void reduceBuffDuration**()** **{**

**if** **(**\_strengthBuffDuration **==** 0**)**

\_strength **=** DEFAULT\_STRENGTH**;**

**if** **(**\_strengthBuffDuration **>** 0**)**

\_strengthBuffDuration**--;**

**}**

public void setSteps**(**int steps**)** **{**

**this.**\_steps **=** steps**;**

**}**

private static final int REQUIRED\_STEPS\_FOR\_MOVE **=** 1**;**

public int steps**()** **{**

**return** \_steps**;**

**}**

public boolean isAvailableSteps**(**int stepsValue**)** **{**

**return** stepsValue **<=** \_steps**;**

**}**

protected int reduceSteps**(**int stepsValue**)** **{**

int retrievedSteps **=** Math**.**min**(**\_steps**,** stepsValue**);**

\_steps **-=** retrievedSteps**;**

**return** retrievedSteps**;**

**}**

public boolean hasSteps**()** **{**

**return** \_steps **>=** REQUIRED\_STEPS\_FOR\_MOVE**;**

**}**

// --------------------------- Перемещение ------------------------------------

public boolean canMoveTo**(**Cell to**)** **{**

**return** to**.**isEmpty**();**

**}**

@Override

public void move**(**Direction direct**)** **{**

Cell pos **=** typedOwner**();**

CellPosition prevPosition **=** pos**.**position**();**

**if** **(!**isAvailableSteps**(**REQUIRED\_STEPS\_FOR\_MOVE**))** **{**

**return;**

**}**

Cell newPos **=** pos**.**neighbour**(**direct**);**

**if** **(**newPos **==** **null)** **{**

**return;**

**}**

**if** **(!**canMoveTo**(**newPos**))** **{**

**return;**

**}**

Unit unit **=** pos**.**extractUnit**();**

newPos**.**putUnit**(**unit**);**

reduceSteps**(**REQUIRED\_STEPS\_FOR\_MOVE**);**

reduceBuffDuration**();**

fireStateChanged**(new** MoveEvent**(this,** newPos**.**position**(),** prevPosition**));**

**}**

public void Sleep**(**int turns**)**

**{**

**while** **(**turns **>=** 0**)** **{**

fireStateChanged**(new** MoveEvent**(this,** **this.**position**(),** **this.**position**()));**

reduceBuffDuration**();**

turns**--;**

**}**

**}**

public void moveBoxWithStep**(**Direction direction**)**

**{**

**if(**moveBoxBehindWithStep**(**direction**))** **return;**

moveSeveralBoxesForwardWithStep**(**direction**);**

reduceBuffDuration**();**

**}**

public boolean moveBoxBehindWithStep**(**Direction direction**)**

**{**

Cell pos **=** typedOwner**();**

Box boxBehind **=** **null;**

**if(**pos**.**neighbour**(**direction**.**opposite**())** **!=** **null** **&&** pos**.**neighbour**(**direction**.**opposite**()).**getUnit**()** **instanceof** Box**)**

**{**

boxBehind **=** **(**Box**)** pos**.**neighbour**(**direction**.**opposite**()).**getUnit**();**

System**.**out**.**println**(**"boxBehind: " **+** boxBehind**);**

**}**

**if(**boxBehind **!=** **null)**

**{**

move**(**direction**);**

boxBehind**.**move**(**direction**);**

**return** **true;**

**}** **else** **{** **return** **false;** **}**

**}**

public boolean moveSeveralBoxesForwardWithStep**(**Direction direction**)**

**{**

Cell pos **=** typedOwner**();**

int overallBoxesWeight **=** 0**;**

Stack**<**Box**>** rowOfBoxes **=** **new** Stack**<**Box**>();**

**while** **(**pos**.**neighbour**(**direction**)** **!=** **null** **&&** pos**.**neighbour**(**direction**).**getUnit**()** **instanceof** Box**)**

**{**

rowOfBoxes**.**push**((**Box**)** pos**.**neighbour**(**direction**).**getUnit**());**

overallBoxesWeight **+=** rowOfBoxes**.**peek**().**getWeight**();**

System**.**out**.**println**(**"rowOfBoxes: " **+** rowOfBoxes**.**size**());**

pos **=** pos**.**neighbour**(**direction**);**

**}**

**if(**overallBoxesWeight **>** getStrength**()** **\*** 2**)**

**{**

System**.**out**.**println**(**"Не хватает силы"**);**

**return** **false;**

**}** **else** **{**

**while** **(!**rowOfBoxes**.**isEmpty**())**

**{**

Box boxOnTop **=** rowOfBoxes**.**peek**();**

**if(** **((**Cell**)** boxOnTop**.**typedOwner**()).**neighbour**(**direction**)** **!=** **null**

**&&** boxOnTop**.**canMoveTo**(** **((**Cell**)** boxOnTop**.**typedOwner**()).**neighbour**(**direction**)))**

**{**

boxOnTop **=** rowOfBoxes**.**pop**();**

boxOnTop**.**move**(**direction**);**

**}** **else**

**{**

System**.**out**.**println**(**"Некуда двигать коробки"**);**

**return** **false;**

**}**

**}**

move**(**direction**);**

**return** **true;**

**}**

**}**

public boolean MoveBoxForwardWithStep**(**Direction direction**)**

**{**

Cell pos **=** typedOwner**();**

Box boxForward **=** **null;**

**if(**pos**.**neighbour**(**direction**)** **!=** **null** **&&** pos**.**neighbour**(**direction**).**getUnit**()** **instanceof** Box**)**

**{**

boxForward **=** **(**Box**)** pos**.**neighbour**(**direction**).**getUnit**();**

System**.**out**.**println**(**"boxForward: " **+** boxForward**);**

**}**

**if(**boxForward **!=** **null)** **{**

boxForward**.**move**(**direction**);**

move**(**direction**);**

**return** **true;**

**}** **else** **{** **return** **false;** **}**

**}**

public void Interact**()**

**{**

Cell c **=** typedOwner**();**

Map**<**Direction**,** Cell**>** neighbours **=** c**.**neighbors**();**

**for** **(**Map**.**Entry**<**Direction**,** Cell**>** entry **:** neighbours**.**entrySet**())** **{**

Cell cell **=** entry**.**getValue**();**

System**.**out**.**println**(**cell**.**position**().**column**()** **+** " " **+** cell**.**position**().**row**()** **+** " " **+** cell**.**getUnit**());**

**if(**cell**.**getUnit**()** **instanceof** Interactable**)** **{**

**((**Interactable**)** cell**.**getUnit**()).**InteractWith**(this);**

**return;**

**}**

**}**

**}**

@Override

public String toString**()** **{**

String msg**;**

msg **=** "G(" **+** steps**()** **+** ")"**;**

**return** msg**;**

**}**

**}**

**4.7 Реализация ключевых тестовых случаев**

package Test**;**

**import** Model**.**gamefield**.**Pinfold**;**

**import** Model**.**seeders**.**TestSeeder**;**

**import** Model**.**units**.**Goat**;**

**import** Model**.**units**.**Grass**;**

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

**import** org**.**junit**.**jupiter**.**api**.**Test**;**

public class GrassTest **{**

@Test

public void DefaultGrassInteraction**()** **{**

Goat g **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**3**,** 3**).**putUnit**(**g**);**

field**.**cell**(**3**,** 4**).**putUnit**(new** Grass**());**

g**.**Interact**();**

Assertions**.**assertEquals**(**g**.**getStrength**(),** 2**);**

Assertions**.**assertEquals**(**g**.**getStrengthBuffDuration**(),** 5**);**

**}**

@Test

public void CustomGrassInteraction**()** **{**

Goat g **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**3**,** 3**).**putUnit**(**g**);**

field**.**cell**(**3**,** 4**).**putUnit**(new** Grass**(**100**,** 100**));**

g**.**Interact**();**

Assertions**.**assertEquals**(**g**.**getStrength**(),** 100**);**

Assertions**.**assertEquals**(**g**.**getStrengthBuffDuration**(),** 100**);**

**}**

@Test

public void GrassBuffProcess**()** **{**

Goat g **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**3**,** 3**).**putUnit**(**g**);**

field**.**cell**(**3**,** 4**).**putUnit**(new** Grass**(**3**,** 4**));**

g**.**Interact**();**

g**.**Sleep**(**4**);**

Assertions**.**assertEquals**(**g**.**getStrength**(),** 1**);**

Assertions**.**assertEquals**(**g**.**getStrengthBuffDuration**(),** 0**);**

**}**

**}**

package Test**;**

**import** Model**.**gamefield**.**Cell**;**

**import** Model**.**gamefield**.**Direction**;**

**import** Model**.**gamefield**.**Pinfold**;**

**import** Model**.**seeders**.**TestSeeder**;**

**import** Model**.**units**.\*;**

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

**import** org**.**junit**.**jupiter**.**api**.**Test**;**

**import** java**.**util**.\*;**

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

public class GoatTest **{**

@Test

public void moveToDirectionsTest**()** **{**

Goat gt **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**3**,** 3**).**putUnit**(**gt**);**

gt**.**move**(**Direction**.**east**());**

Assertions**.**assertEquals**(**gt**.**position**(),** field**.**cell**(**3**,**4**).**position**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**3**,** 4**));**

gt**.**move**(**Direction**.**west**());**

Assertions**.**assertEquals**(**gt**.**position**(),** field**.**cell**(**3**,**3**).**position**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**3**,** 3**));**

gt**.**move**(**Direction**.**south**());**

Assertions**.**assertEquals**(**gt**.**position**(),** field**.**cell**(**4**,**3**).**position**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**4**,** 3**));**

gt**.**move**(**Direction**.**north**());**

Assertions**.**assertEquals**(**gt**.**position**(),** field**.**cell**(**3**,**3**).**position**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**3**,** 3**));**

**}**

@Test

public void connectionWithCellTest**()** **{**

Goat gt **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**3**,** 3**).**putUnit**(**gt**);**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**3**,** 3**));**

**}**

@Test

public void goatCollideWithInnerWallTest**()** **{**

Goat gt **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**4**,** 3**).**putUnit**(**gt**);**

field**.**cell**(**5**,** 3**).**putUnit**(new** Wall**());**

gt**.**move**(**Direction**.**south**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**4**,** 3**));**

**}**

@Test

public void GoatInteraction**()** **{**

Goat g **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**3**,** 3**).**putUnit**(**g**);**

field**.**cell**(**3**,** 4**).**putUnit**(new** Cabbage**());**

g**.**Interact**();**

**for** **(**Cell cell **:** field**)** **{**

assertNotEquals**(**cell**.**getUnit**(),** field**.**Cabbage**());**

**}**

**}**

@Test

public void goatEatsGrassAndMoveTwoBoxes**()** **{**

Goat gt **=** **new** Goat**();**

Box box1 **=** **new** Box**(),** box2 **=** **new** Box**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**1**,** 1**).**putUnit**(**gt**);**

field**.**cell**(**1**,** 2**).**putUnit**(**box1**);**

field**.**cell**(**1**,** 3**).**putUnit**(**box2**);**

field**.**cell**(**2**,**1**).**putUnit**(new** Grass**(**3**,**5**));**

gt**.**Interact**();**

gt**.**moveSeveralBoxesForwardWithStep**(**Direction**.**east**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**1**,** 2**));**

Assertions**.**assertEquals**(**box1**.**typedOwner**(),** field**.**cell**(**1**,** 3**));**

Assertions**.**assertEquals**(**box2**.**typedOwner**(),** field**.**cell**(**1**,** 4**));**

**}**

@Test

public void goatHaveNotEnoughStrengthToMoveBoxes**()** **{**

Goat gt **=** **new** Goat**();**

Box box1 **=** **new** Box**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**1**,** 1**).**putUnit**(**gt**);**

field**.**cell**(**1**,** 2**).**putUnit**(**box1**);**

field**.**cell**(**2**,**1**).**putUnit**(new** Grass**(**0**,**5**));**

gt**.**Interact**();**

gt**.**moveSeveralBoxesForwardWithStep**(**Direction**.**east**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**1**,** 1**));**

Assertions**.**assertEquals**(**box1**.**typedOwner**(),** field**.**cell**(**1**,** 2**));**

**}**

@Test

public void goatEatsGrassAndMoveSeveralBoxes**()** **{**

Goat gt **=** **new** Goat**();**

List**<**Box**>** boxes **=** **new** ArrayList**<>();**

**for** **(**int i **=** 0**;** i **<** 5**;** i**++)**

**{**

boxes**.**add**(new** Box**());**

**}**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**1**,** 1**).**putUnit**(**gt**);**

**for** **(**int i **=** 0**;** i **<** 5**;** i**++)**

**{**

field**.**cell**(**1**,** 2 **+** i**).**putUnit**(**boxes**.**get**(**i**));**

**}**

field**.**cell**(**2**,**1**).**putUnit**(new** Grass**(**20**,**5**));**

gt**.**Interact**();**

gt**.**moveSeveralBoxesForwardWithStep**(**Direction**.**east**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**1**,** 2**));**

**for** **(**int i **=** 0**;** i **<** 5**;** i**++)**

**{**

Assertions**.**assertEquals**(**boxes**.**get**(**i**).**typedOwner**(),** field**.**cell**(**1**,** 3 **+** i**));**

**}**

**}**

@Test

public void goatCollideWithOuterWallTest**()** **{**

Goat gt **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**1**,** 1**).**putUnit**(**gt**);**

gt**.**move**(**Direction**.**west**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**1**,** 1**));**

**}**

@Test

public void goatCollideWithBoxNearOuterWallTest**()** **{**

Goat gt **=** **new** Goat**();**

Pinfold field **=** **new** Pinfold**(**10**,** 10**,** **new** TestSeeder**());**

field**.**cell**(**9**,** 8**).**putUnit**(**gt**);**

field**.**cell**(**9**,** 9**).**putUnit**(new** Box**());**

gt**.**move**(**Direction**.**east**());**

Assertions**.**assertEquals**(**gt**.**typedOwner**(),** field**.**cell**(**9**,** 8**));**

**}**

**}**

# 5 Список использованной литературы и других источников

1. 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
2. Программирование на языке Java. Конспект лекций. [Электронный ресурс] : учеб. пособие / А.В. Гаврилов [и др.]. — Электрон. дан. — СПб.: НИУ ИТМО, 2015. — 126 с. — Режим доступа: http://e.lanbook.com/book/91488
3. Swing. Эффектные пользовательские интерфейсы. [Электронный ресурс]: учеб. пособие / Портянкин И. А. — 524 с. — Режим доступа: https://ipsoftware.ru/books/swing\_book\_2/