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\_01)

Студент: Марченко С.С.

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

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

**Задание**

**на курсовую работу**

Студент: Шашков Д. А.

Группа: ПрИн-367

1. Тема: «Проектированиепрограммы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №01\_01)

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

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

3. Содержание пояснительной записки:

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

4. Перечень графического материала:

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

5. Дата выдачи задания « » февраля 2024 г.

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

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

« » февраля 2024 г.

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

**Вариативность:** на поле могут появляться объекты, которые могут или не могут быть передвинуты козой. Кроме того, неперемещаемые объекты можно активировать, применив ранее собранные объекты (неперемещаемые объекты). Результатом активации является изменение положения существующих объектов, появление новых объектов, уничтожение имеющихся.

**НЕ изменяя** ранее созданные классы, а используя **точки расширения, реализовать:** телепортатор, который активируется ключом.

# 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.

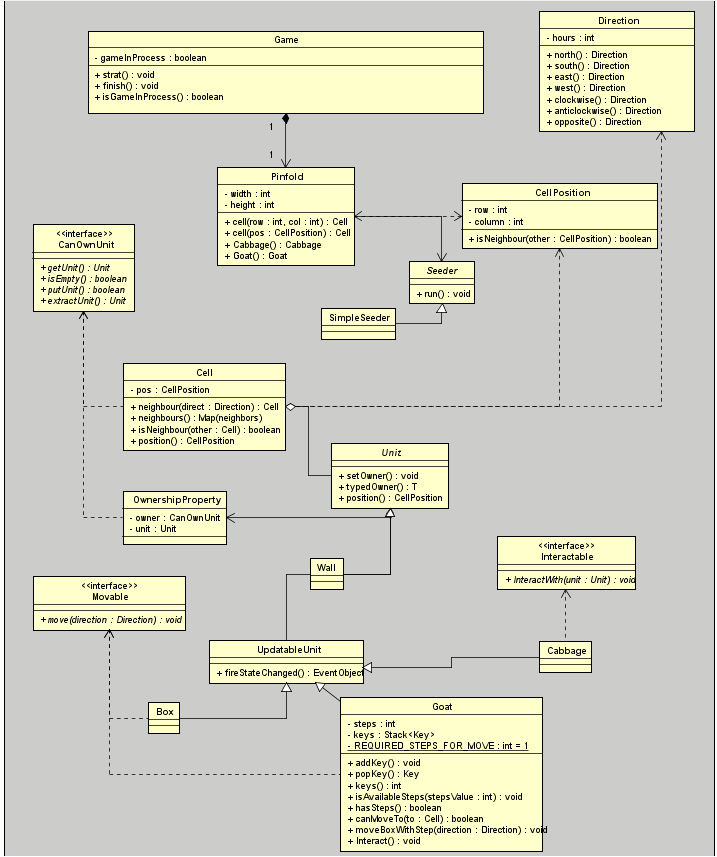
**Загон** – умеет создавать Стены, Козу и Капусту и размещать их на Поле в Ячейках. Позиции этих сущностей Загон определяет самостоятельно. Он знает расположение Капусты.

**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.

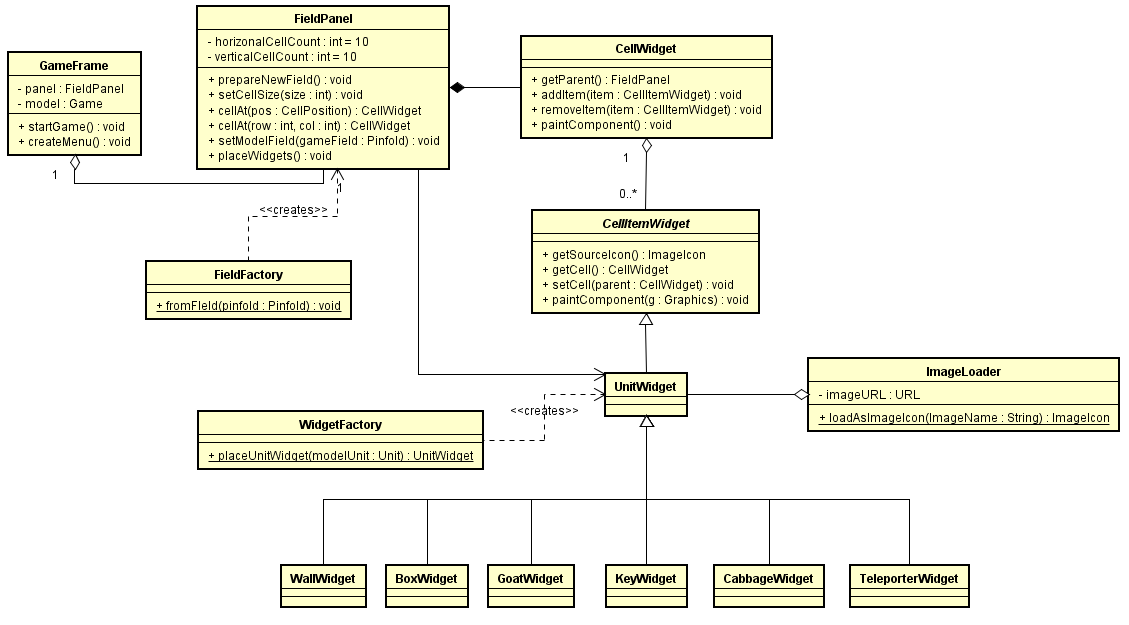
**Капуста** – разновидность объекта взаимодействия. Когда в Ячейку, где находится Капуста, становится Коза, Капуста уничтожается и сообщает о конце игры.

**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**

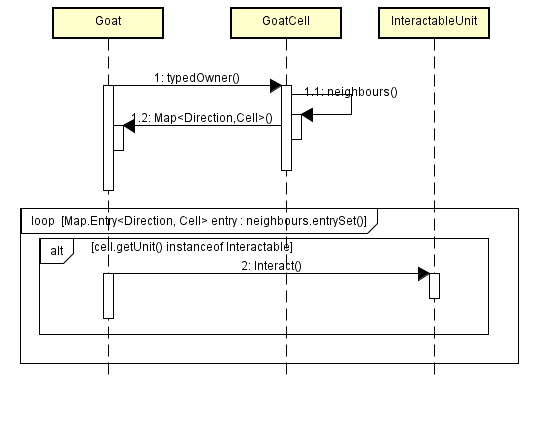


**Диаграмма классов модели**

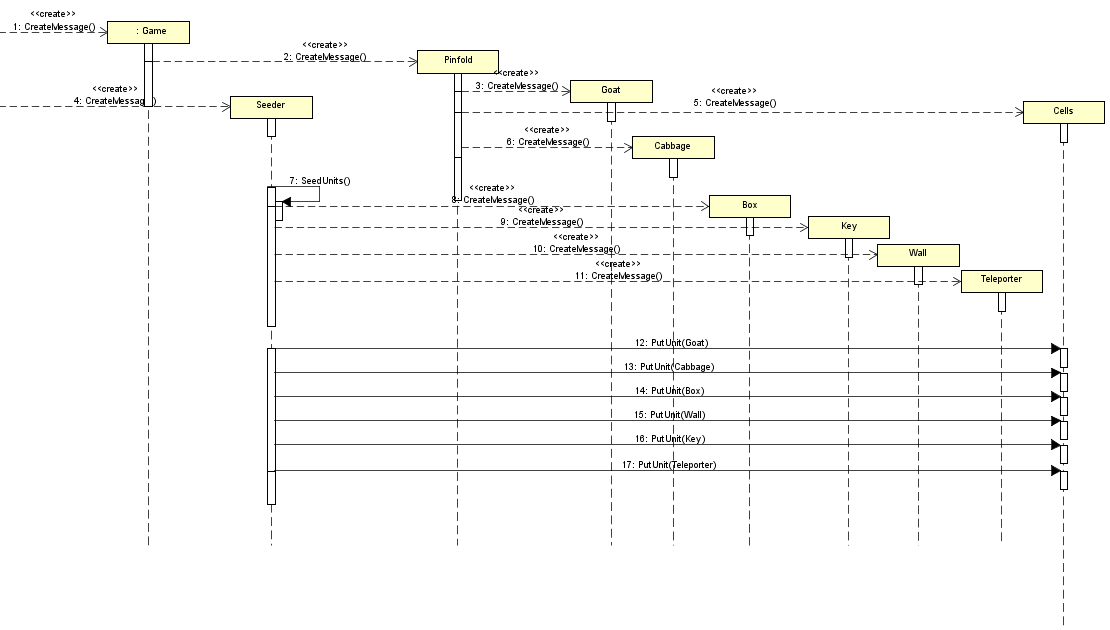


**Диаграмма классов представления**

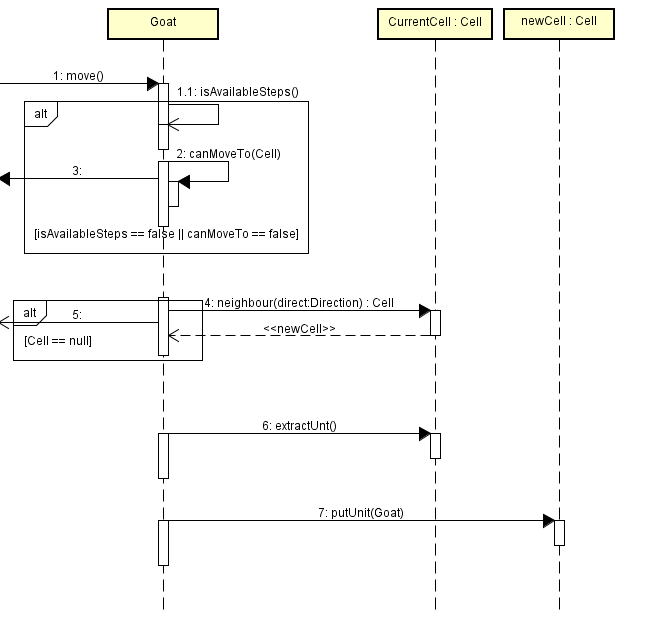
**3.5 Типовые процессы в программе**

****

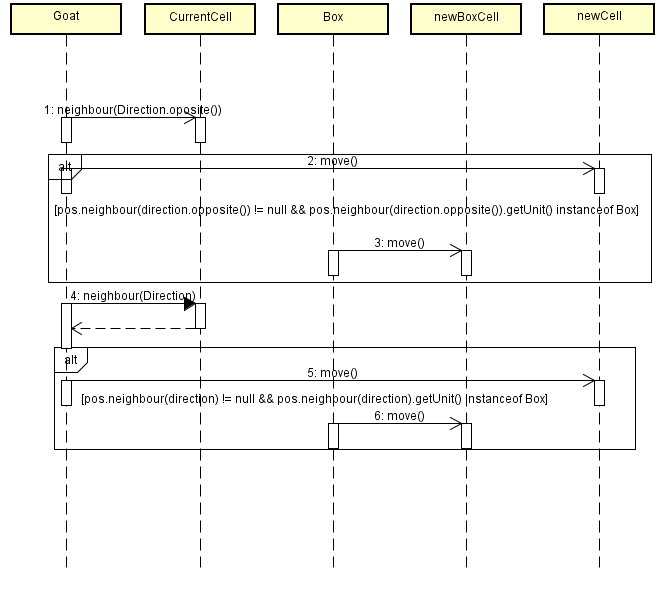
**Рисунок 2. Взаимодействие с объектами**

****

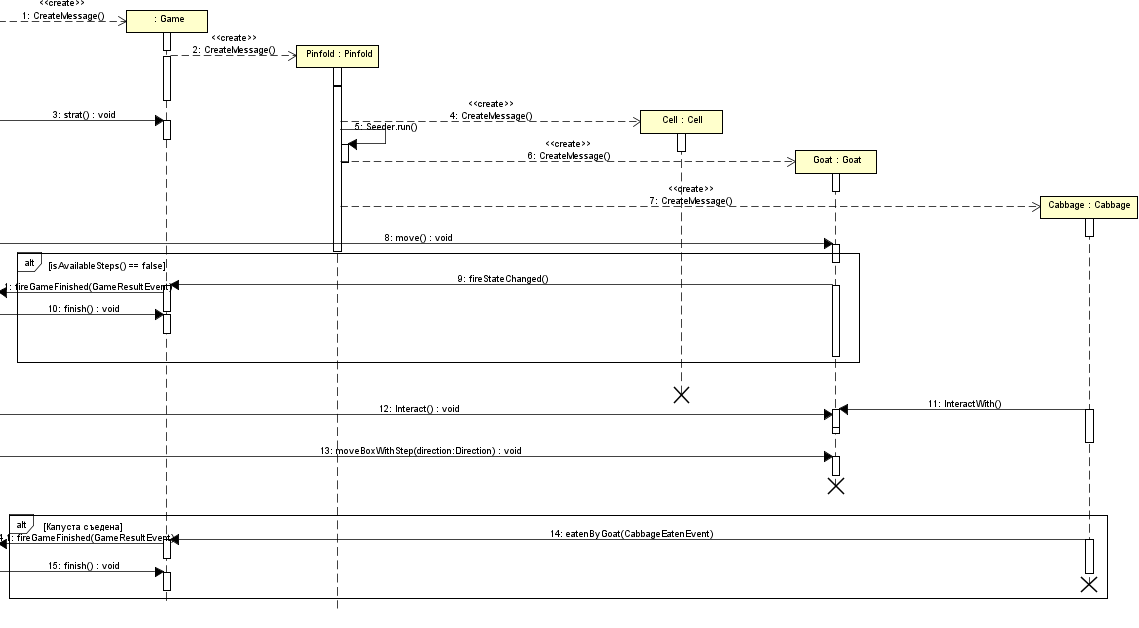
**Рисунок 3. Заполнение загона**

****

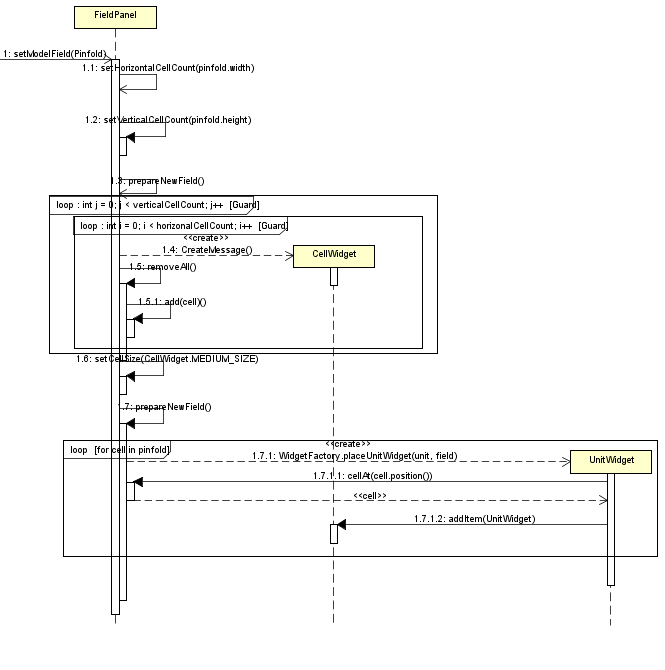
**Рисунок 4. Перемещение**

****

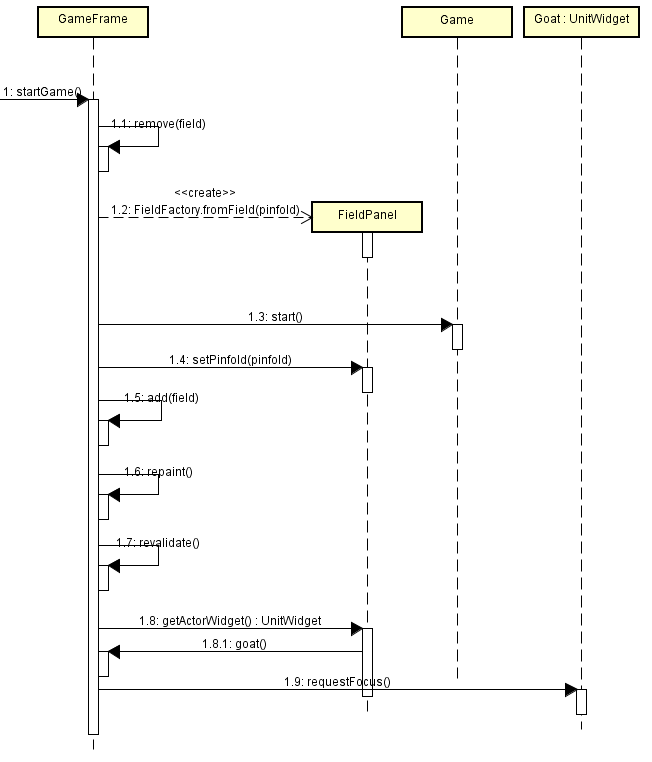
**Рисунок 5. Перемещение коробки**

****

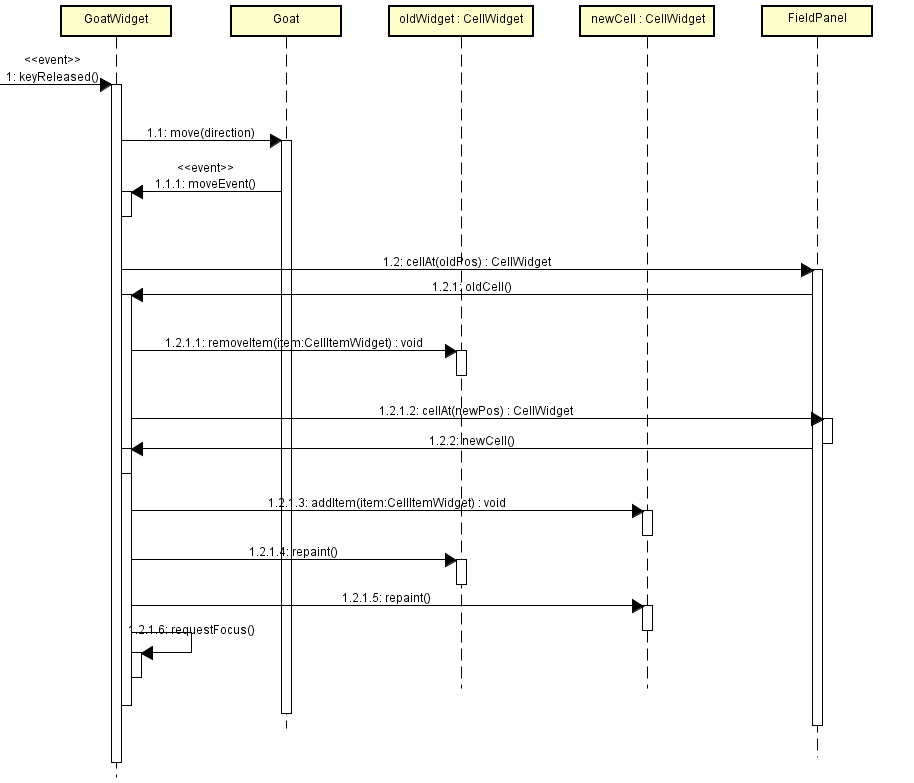
**Рисунок 6. Съедение капусты**

****

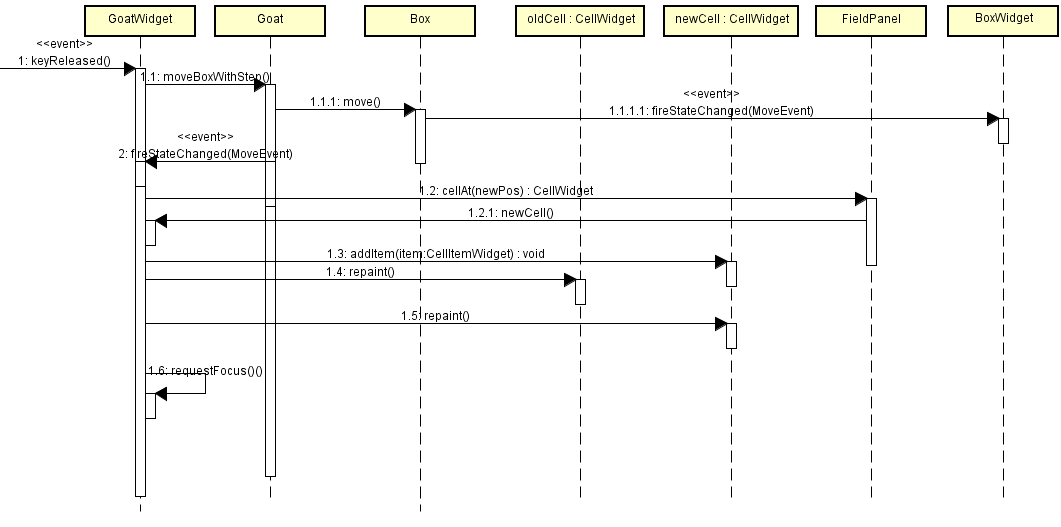
**Рисунок 7. Заполнение поля**

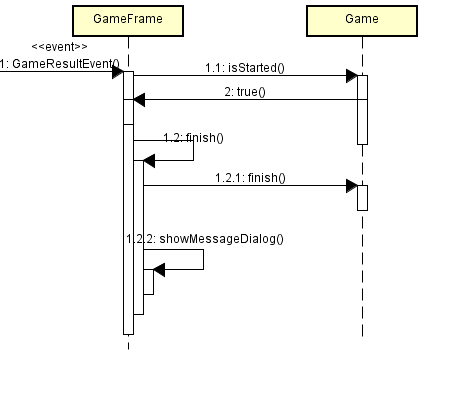
****

**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.

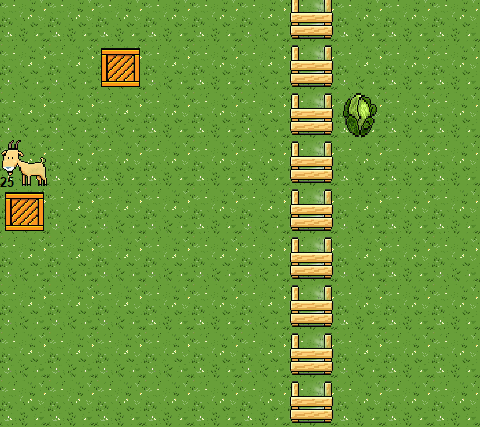


Рисунок 12. Вид загона в представлении

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

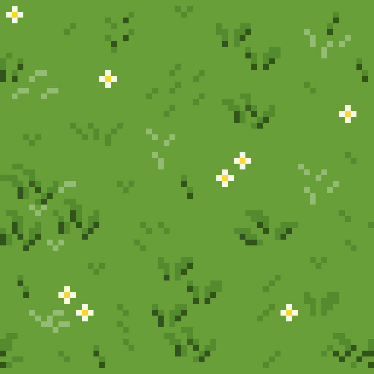


Рисунок 14. Фоновое изображение клетки загона

Также имеются изображения для Коробки, Стен и Капусты

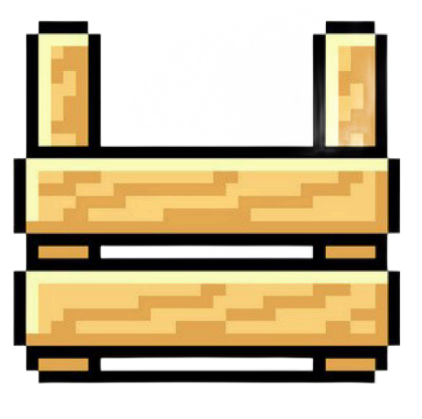
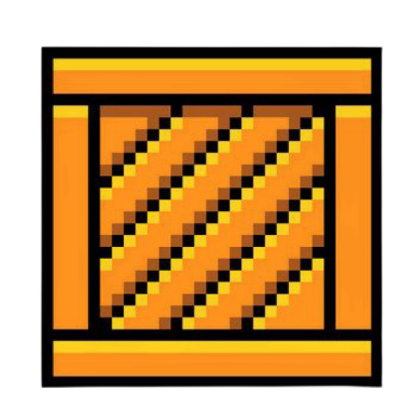


Рисунок 15. Изображение Стены



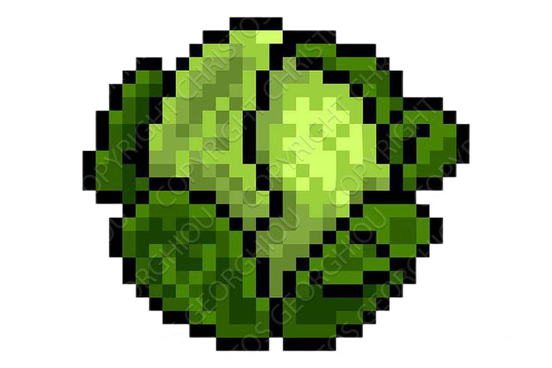
Рисунок 16. Изображение Коробки

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 GameFIeldTest **{**

@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 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**());**

**}**

**}**

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 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**()));**

**}**

**}**

# 4 Вторая итерация разработки

**4.1 Функциональные требования (сценарии)**

1. **Сценарий** «Коза использует Яму с помощью лопаты»

1. В ответ на запрос Козы, Яма запрашивает количество Лопат у Козы

2. Если Лопаты есть, Лопата изымается у Козы

3. Яма перемещает Козу рядом с другой Ямой, связанной с текущей

* 1. **Альтернативный Сценарий** «Коза использует Яму без Лопаты с собой»

1. В ответ на запрос Козы, Яма запрашивает количество Лопат у Козы
2. Если Лопаты нет, взаимодейтсвие Козы с Ямой заканчивается

**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.

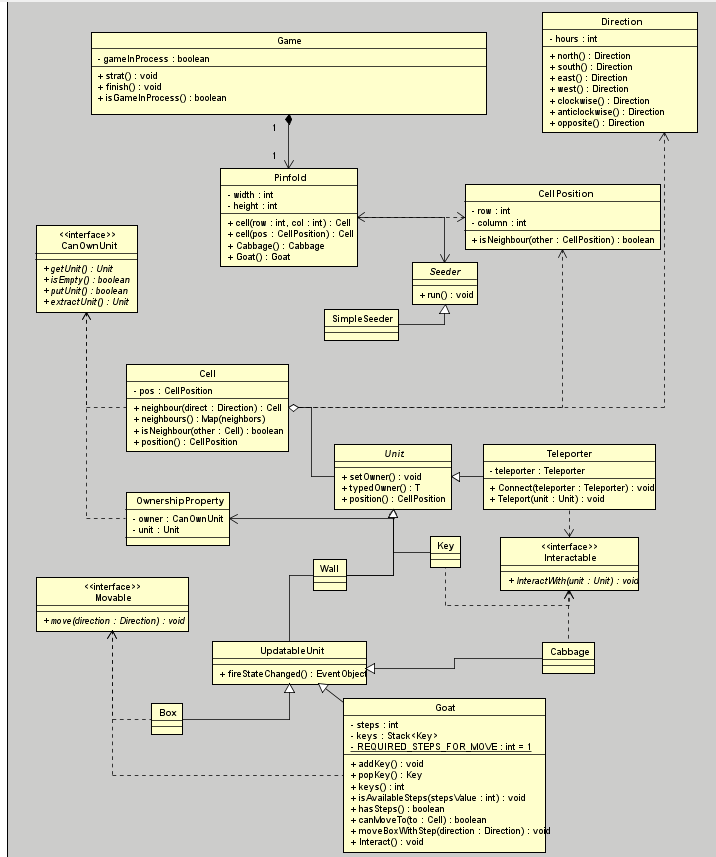
**Коза** – умеет *перемещаться* и двигать одну коробку на себя или от себя. Коза перемещается в соседнюю Ячейку, затрачивая Шаги, но не может пройти через Стену.

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

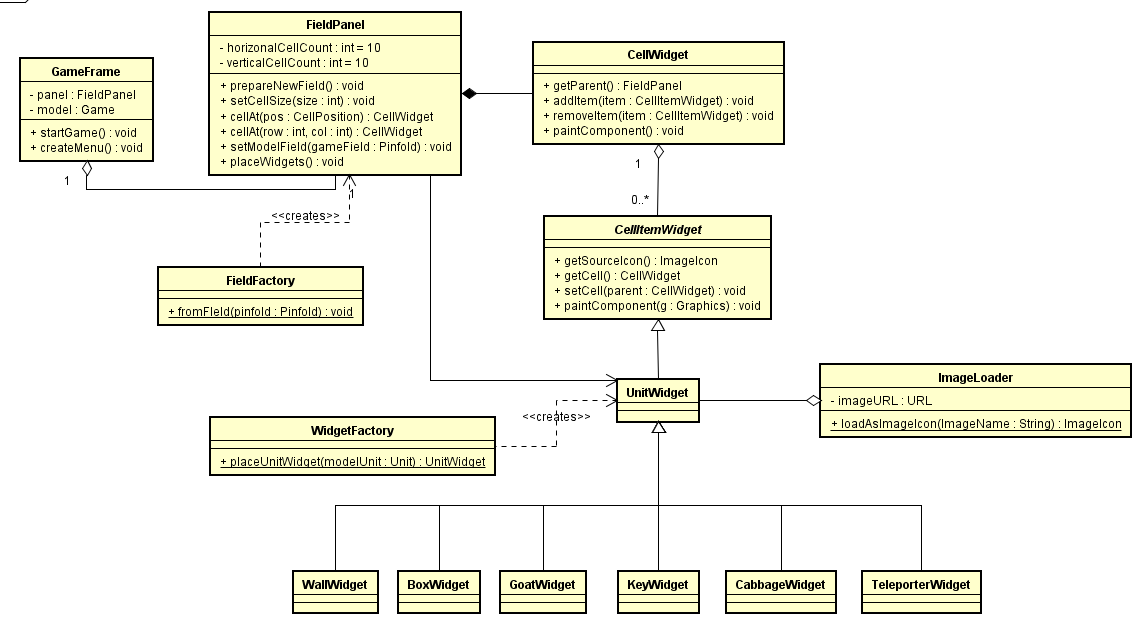
**Ящик** – разновидность объекта взаимодействия. Непреодолимое препятствие для Козы. Когда рядом Коза, она может взять Ящик и потянуть вперед или назад.

**Ключ (в документе Лопата)** – разновидность объекта взаимодействия. Непреодолимое препятствие для Козы. Когда рядом Коза, она может подобрать ключ с поля себе.

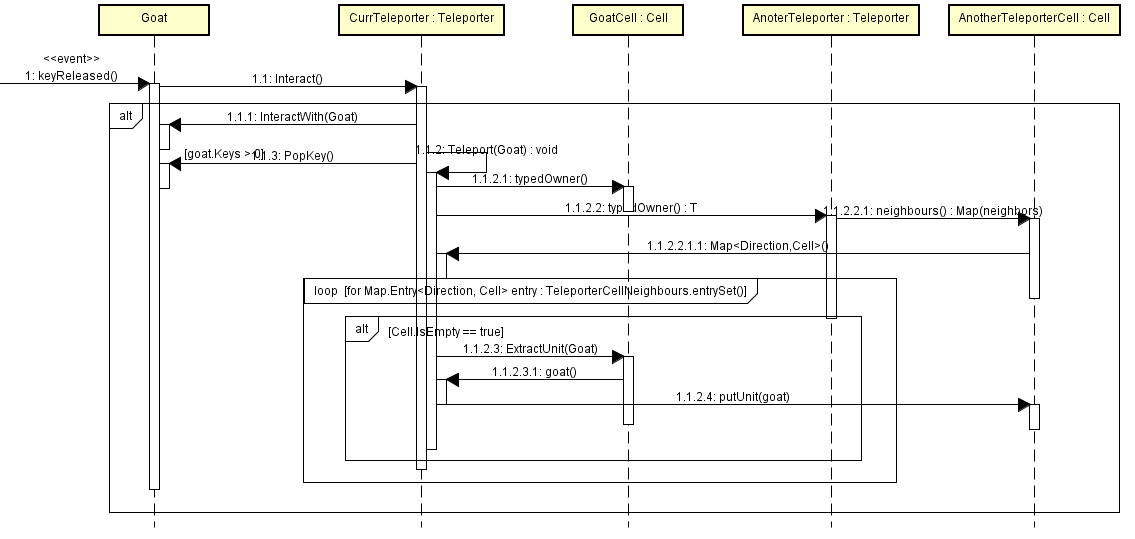
**Телепорт (в документе Яма)** – разновидность объекта взаимодействия. Непреодолимое препятствие для Козы. Когда рядом Коза, она может Переместиться от одного телепорта к другому ценой одного Ключа(Лопаты).

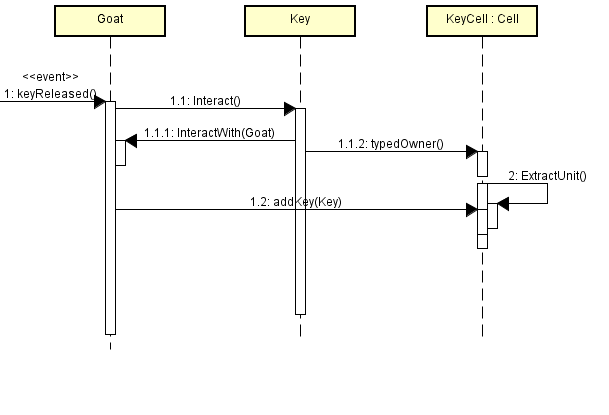
**4.3 Структура программы на уровне классов**

**Рисунок 18. Диаграмма классов модели**



**Диаграмма классов представления**

**4.4 Типовые процессы в программе**

**Рисунок 19. Перемещение козы через телепорт**

**Рисунок 20 .Подбор ключа**

**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

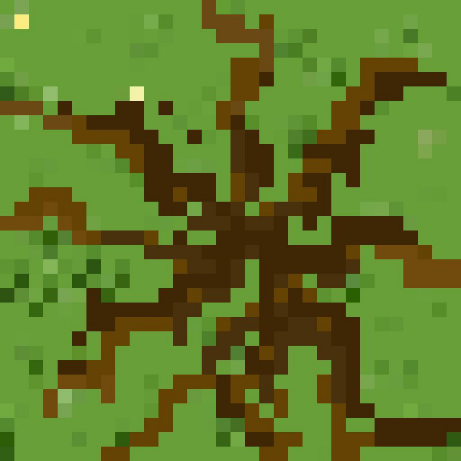


Рисунок 21. Изображение ямы

Рисунок 22. Изображение лопаты

**4.6 Реализация ключевых классов**

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

private Teleporter teleporter**;**

public Teleporter getTeleporter**()** **{**

**return** teleporter**;**

**}**

private void setTeleporter**(**Teleporter teleporter**)** **{**

**this.**teleporter **=** teleporter**;**

**}**

public void Connect**(**Teleporter teleporter**)** **{**

**if(this** **==** teleporter**)** **{**

**return;**

**}**

**this.**setTeleporter**(**teleporter**);**

teleporter**.**setTeleporter**(this);**

**}**

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

**if(**teleporter **==** **null)** **return;**

Cell TeleporterCell **=** teleporter**.**typedOwner**();**

Cell UnitCell **=** unit**.**typedOwner**();**

Map**<**Direction**,** Cell**>** TeleporterCellNeighbours **=** TeleporterCell**.**neighbors**();**

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

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

**if(**cell**.**isEmpty**())**

**{**

Unit extractedUnit **=** UnitCell**.**extractUnit**();**

cell**.**putUnit**(**extractedUnit**);**

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

**return;**

**}**

**}**

**}**

@Override

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

**if(**unit **instanceof** Goat **&&** **((**Goat**)**unit**).**Keys**()** **>** 0**)** **{**

**((**Goat**)**unit**).**PopKey**();**

Teleport**(**unit**);**

System**.**out**.**println**(**"Keys remain: " **+** **((**Goat**)**unit**).**Keys**());**

**}**

**}**

**}**

// Ключ

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

@Override

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

**return** "K"**;**

**}**

@Override

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

**{**

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

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

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

goat**.**AddKey**((**Key**)**c**.**extractUnit**());**

System**.**out**.**println**(**"Keys: " **+** goat**.**Keys**());**

**}**

**}**

**}**

**4.7 Реализация ключевых тестовых случаев**