

# OBJECTION



SWOP 2012-2013

ITERATION 2

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>General Information</b>	<b>3</b>
2.1	Team Work . . . . .	3
2.2	Iterations . . . . .	3
2.2.1	Late Submission Policy . . . . .	4
2.2.2	When Toledo Fails . . . . .	4
2.3	The Software . . . . .	4
2.4	User Interface . . . . .	4
2.5	Testing . . . . .	5
2.6	Tools . . . . .	5
2.7	What You Should Hand In . . . . .	5
2.8	Evaluation . . . . .	5
2.8.1	Presentation Of The Current Iteration . . . . .	6
2.8.2	Planning Tests . . . . .	6
2.9	Peer/Self-assessment . . . . .	6
<b>3</b>	<b>Objectron</b>	<b>8</b>
3.1	Actions . . . . .	8
3.2	The Grid . . . . .	8
3.2.1	Walls . . . . .	9
3.2.2	Power Failure . . . . .	9
3.3	Items . . . . .	10
3.3.1	Light Grenades . . . . .	10
<b>4</b>	<b>Domain Model</b>	<b>10</b>
<b>5</b>	<b>Use Cases</b>	<b>13</b>
5.1	Use Case: Start New Game . . . . .	13
5.2	Use Case: Move . . . . .	13
5.3	Use Case: Pick Up An Item . . . . .	14
5.4	Use Case: Use An Item From The Inventory . . . . .	15
5.5	Use Case: End Turn . . . . .	15

# 1 Introduction

For the course *Software Ontwerp: project*, you will develop a board game in which players move on a grid with the goal to reach a target before the opponents reach their targets.

In Section 2, we explain how the project is organized, discuss the quality requirements for the software you will develop and how we evaluate the solutions. In Section 3, we explain the problem domain of the application, followed by a diagram of the domain model in Section 4. We describe the use cases in detail in Section 5.

## 2 General Information

In this section, we explain how the project is organized, what is expected of the software you will develop and the report you will write.

### 2.1 Team Work

For this project, you will work in groups of four. Each group is assigned an advisor from the educational staff. If you have any questions regarding the project, you can contact your advisor and schedule a meeting. When you come to the meeting, you are expected to prepare specific questions and have sufficient design documentation available. *If the design documentation is not of sufficient quality, the corresponding question will not be answered.* It is your own responsibility to organize meetings with your advisor and we advise to do this regularly. Experience from previous years shows that groups that regularly meet with their advisors produce a higher quality design.

If there are problems within the group, you should immediately notify your advisor. Do not wait until right before the deadline or the exam!

To ensure that every team member practices all topics of the course, a number of roles are assigned to team members at the start of each iteration (or shortly thereafter in case of the first iteration). A team member that is assigned a certain role will give the presentation or demo corresponding to that role at the end of the iteration. That team member, however, is *not supposed to do all of the work concerning his task!* But he must take a leading role in that activity, and be able to answer most questions on that topic during the evaluation.

The following roles will be assigned round-robin. The iterations during which a role is used is shown between square brackets.

**Lead Designer [1...6]** The lead designer has an active role in making the design of your software. In addition to knowing the design itself, he knows *why* these design decisions were taken, and what the alternatives are.

**Lead Tester [1...6]** The lead tester has an active role in planning, designing, and writing the tests for the software. Know which kinds of tests are needed (scenario, unit) for testing various parts of the software, which concrete tests are chosen (success cases, failure cases, corner cases,...), and which techniques are used (mocking, stubbing, ...).

**Lead Domain Modeler [3...6]** The lead domain modeler has an active role in defining the domain model. He knows which parts of the domain are relevant for the application.

### 2.2 Iterations

The project is divided into six iterations. In the first iteration, you will implement the base functionality of the software. In subsequent iterations, new functionality will be added and/or existing functionality will be changed. For this year, the deadlines are:

- Iteration 1: 18/02/2013 - 01/03/2013 at 18:00
- Iteration 2: 04/03/2013 - 15/03/2013 at 18:00

- Iteration 3: 18/03/2013 - 29/03/2013 at 18:00
- Iteration 4: 15/04/2013 - 26/04/2013 at 18:00
- Iteration 5: 29/04/2013 - 10/05/2013 at 18:00
- Iteration 6: 13/05/2013 - 24/05/2013 at 18:00

### 2.2.1 Late Submission Policy

If the zip file is submitted  $N$  minutes late, with  $0 \leq N \leq 240$ , the score for all team members is scaled by  $(240 - N)/240$  for that iteration. For example, if your solution is submitted 30 minutes late, the score is scaled by 87.5%, so the maximum score for the iteration is 1.75. If the zip file is submitted more than 4 hours late, the score for all team members is 0 for that iteration.

### 2.2.2 When Toledo Fails

If the Toledo website is down -- and *only* if Toledo is down -- at the time of the deadline, submit your solution as follows. First, make sure that the departmental subversion repository of your group is up to date. Then, send an email to both Prof. Bart Jacobs and your project advisor to report the problem and provide the url of your repository. We will then do a checkout based on the *timestamp* of the deadline. This means that any changes after the deadline will not be seen by us.

Even if you use a different repository during your project, we will only do a checkout from departmental repositories in case of problems with Toledo. Therefore, be sure to test that you have properly configured access to your repository.

## 2.3 The Software

The focus of this course is on the *quality* (maintainability, extensibility, stability, readability,...) of the software you write. We expect you to use the development process and the techniques that are taught in this course. You are also required to provide extensive class and method documentation, as taught in previous courses.

When designing and implementing your system, you should use a *defensive programming style*. This means that the *client* of the public interface of a class cannot bring the objects of that class, or objects of connected classes, in an inconsistent state. You can optionally use tools such as *Contracts For Java*, and *NonNullCheckWeaver* to enable runtime checking of pre- and postconditions. These tools can help you to reduce the workload for testing and debugging. Instructions to use these tools are given in the appendices of the assignment of iteration 1.

Unless explicitly stated in the assignment, you do not have to take into account persistent storage, security, multi-threading, and networking. If you have doubts about other non-functional concerns, please ask your advisor.

## 2.4 User Interface

We provide a library that allows you to draw the grid of the game board and draw objects at certain grid locations. You are encouraged to use this library, but it is not mandatory. You are allowed to create your own visualization code, but the amount of time allocated for each iteration does *not include* the workload for such custom visualization code. You will not gain any points for making a nicer user interface. You will, however, lose many points if the quality (design, testing, ...) of your software is lacking because you spent too much time on the user interface.

## 2.5 Testing

All functionality of the software should be tested. For scenario tests, this means that not only success scenarios should be tested, but also scenarios for all kinds of faulty user input. Similarly, in addition to confirming a few cases of desired behavior, unit tests should try to break class invariants and postconditions.

Tests should have good coverage, i.e. a testing strategy that leaves large portions of a software system untested is of low value. Several tools exist to give a rough estimate of how much code is tested. One such tool is Eclemma<sup>1</sup>. If this tool reports that only 60% of your code is covered by tests, this indicates there may be a serious problem with (the execution of) your testing strategy. However, be careful when drawing conclusions from both reported high coverage and reported low coverage (understand why you should be careful).

The lead tester is expected to use a coverage tool and briefly report the results during the evaluation of the iteration.

## 2.6 Tools

Instructions to run Visual Paradigm in the computer labs is described in the following file:  
`/localhost/packages/visual_paradigm/README.CS.KULEUVEN.BE`

This file also contains the location of the license key that you can use on your own computer.

## 2.7 What You Should Hand In

You hand in an electronic ZIP-archive via Toledo. **The archive contains the items below and follows the structure defined below:**

- `groupXX` (where XX is your group number (e.g. 01, 12, ...))
  - `design`: a folder containing **all** your design diagrams as image files (PDF, PNG, JPG, ...), including those not used in the presentation
  - `doc`: a folder containing the Javadoc documentation of your entire system
  - `src`: a folder containing your source code
  - `presentation.pdf`: a PDF version of your presentation
  - `system.jar`: an executable JAR file of your system

When including your source code into the archive, make sure to *not include files from your version control system*. If you use subversion, you can do this with the `svn export` command, which removes unnecessary repository folders from the source tree.

Make sure you choose relevant file names for your analysis and design diagrams (e.g. `SS-DsomeOperation.png`). You do **not** have to include the project file of your UML tool, only the exported diagrams.

We should be able to start your system by executing the JAR file with the following command:  
`java -jar system.jar`.

## 2.8 Evaluation

After iterations 1-5 there will be an intermediate evaluation of your solution. An intermediate evaluation consists of a presentation about the design and the testing approach, accompanied by a demo of the tests. The final part of the intermediate evaluation consists of a presentation of the testing approach for the next iteration.

The intermediate evaluation of an iteration will cover only the part of the software that was developed during that iteration. Before the final exam, the *entire* project will be evaluated. It is your own responsibility to process the feedback, and discuss the results with your advisor.

---

<sup>1</sup><http://www.eclemma.org>

### 2.8.1 Presentation Of The Current Iteration

The main part of the presentation should cover the design. The motivation of your design decisions *must* be written in terms of GRASP principles. Use the appropriate design diagrams to illustrate how the most important parts of your software work. Your presentation should cover the following elements. Note that these are not necessarily all separate sections in the presentation.

1. A discussion of the high level design of the software.
2. A more detailed discussion of the parts that you think are the most interesting in terms of design.
3. Which GRASP and design patterns you used.
4. A discussion of the extensibility of the system. Briefly discuss how your system can deal with a number of change scenarios (e.g. extra constraints, additional domain concepts,...).
5. A discussion of the testing approach used in the current iteration.

Your presentation should not consist of slides filled with text, but of slides with clear design diagrams and keywords or a few short sentences. The goal of giving a presentation is to communicate a message, not to write a novel. All design diagrams should be *clearly readable* and use the correct UML notation. It is therefore typically a bad idea to create a single class diagram with all information. Instead, you can for example use an overview class diagram with only the most important classes, and use more detailed class diagrams to document specific parts of the system. Similarly, use appropriate interaction diagrams to illustrate the working of the most important (or complex) parts of the system.

### 2.8.2 Planning Tests

The evaluation of an iteration is planned between 2 and 5 days after that iteration. Immediately after an iteration ends, the assignment for the next iteration is released. The lead tester for the new iteration then starts planning and/or writing the *scenario* tests for the new iteration. The tests should compile, and they should all fail, so that it is clear that the implementation still has to be made (unless of course you manage to already implement one or more scenarios in that short period). It is of course allowed to change these tests throughout the iteration, but they should already provide a clear criterium to decide when the next iteration is finished. He will present his work at the end of the evaluation of the previous iteration.

## 2.9 Peer/Self-assessment

In order for you to critically reflect upon the contribution of each team member, you are asked to perform a peer/self-assessment within your team. For each team member (including yourself) and for each of the criteria below, you must give a score on the following scale: *poor/below average/adequate/above average/excellent*. The criteria to be used are:

- Design skills (use of GRASP and DESIGN patterns, ...)
- Coding skills (correctness, defensive programming, documentation,...)
- Testing skills (approach, test suite, coverage, ...)
- Collaboration (teamwork, communication, commitment)

In addition to the scores themselves, we expect you to briefly explain for each of the criteria why you have given these particular scores to each of the team members. The total length of your evaluation should not exceed 1 page.

Please be fair and to the point. Your team members will not have access to your evaluation report. If the reports reveal significant problems, the project advisor may discuss these issues with you and/or your team. Please note that your score for this course will be based on the quality of the work that has been delivered, and not on how you are rated by your other team members.

The deadline for the peer/self-assessment for each iteration is the Sunday after the deadline for that iteration at 18:00. Submit your peer/self-assessment by e-mail to both Prof. Bart Jacobs and your project advisor, using the following subject:

**[SWOP] peer-/self-assessment of group \$groupnumber\$ by \$firstname\$ \$lastname\$**

### 3 Objectron

The domain of this year's project is a game that is played on a grid. Each of two players try to reach their target before the other player does.

#### 3.1 Actions

The game is turn-based. During each turn, a player is allowed to perform 3 individual actions. The kind of each action can be chosen by the player. The following kinds of actions are supported:

1. The first kind of action is to move.
  - (a) During a single action, a player can move one square in any direction (vertical, horizontal, and diagonal).
  - (b) No two players can occupy the same square at the same time.
  - (c) A player cannot occupy a square that is occupied by a wall.
  - (d) A player cannot leave the grid.
  - (e) When a player moves, he leaves a light trail behind on his previous square. The light trail remains active during 2 additional actions *of that player*. After that, the light trail disappears. Therefore, the maximum length of the light trail is 3 squares.
  - (f) A player cannot cross any light trail: he cannot enter a square that contains a light trail, and he cannot pass through a diagonal light trail. **For this rule, a player is considered to be part of his light trail. A player cannot pass diagonally between another player and the most recent part (the head) of the light trail of the other player. A player can diagonally pass behind the oldest part (the tail) of the light trail of the other player, unless other constraints prevent this. Think of the light trail as a chain that the player drags along.**
  - (g) A player is not allowed to end the turn on the square that he started the turn on.
  - (h) If a player cannot move **in an action of his turn**, he is trapped and loses the game. **If a player has no actions in a turn, due to the explosion of a light grenade (see further), he skips this turn but he does not automatically lose the game.**
  - (i) If a player reaches the starting position of his opponent, he wins the game.
2. The second kind of action is to pick up an item.
  - (a) A player can only pick up an item that is placed on the same square as the player.
  - (b) A player can carry at most six items.
3. The third kind of action is to use an item.
  - (a) A player can only use an item that is in his inventory.
4. The fourth kind of action is to end the turn. In this case, an empty action is performed for each remaining action in the turn.

#### 3.2 The Grid

The grid on which the game is played has a rectangular shape for which the size can be configured when the game is started. The minimum grid size is 10x10. Player 1 starts on the bottom left corner of the grid, player 2 starts on the top right corner. The finish for each player is the starting location of the other player.



### 3.2.1 Walls

At the start of the game, walls are placed on the grid. A wall forms a barrier that a player cannot cross. To ensure that the game is still playable, placement of the walls is constrained.

1. Walls are placed either horizontal or vertical, and have a width of 1 square.
2. The direction of each wall is chosen randomly.
3. The minimal length of a wall is two squares.
4. The maximum length (in terms of the numbers of squares that it covers) of a vertical (horizontal) wall is 50% of the vertical (horizontal) size of the grid, rounded up to an integer value.
5. The number of walls is chosen randomly.
6. There is at least 1 wall.
7. At most 20% of the squares in the grid, rounded up to an integer value, is covered by a wall.
8. Walls can neither touch each other nor intersect. *If the largest vertical wall in Figure 1 is shifted 1 square up, it touches the horizontal wall. If we extend the largest vertical wall upward by 3 squares, and the horizontal wall to the right by 2 squares, the walls intersect.*
9. A wall cannot cover the starting position of a player.

### 3.2.2 Power Failure

At random moments in the game, squares can experience a power failure. Moving through a square affected by a power failure slows a player down. Power failures behave according to the following rules:

1. At the start of each turn, each square has a 5% chance of losing power.
2. When a power failure occurs on a square, the squares directly adjacent to this square lose their power as well. The effect of that particular power failure does not spread farther than that, but a square can be affected by multiple power failures.
3. A square that is affected by a power failure loses power for 3 turns starting from the occurrence of that power failure.
4. Squares containing active light grenades can be lose power, and this increases the impact of the light grenade on detonation (see further).
5. Power failures have no influence on other items (such as inactive light grenades) or the light trail.
6. If a player enters a square without power and that square does not contain an active light grenade, his turn ends.
7. If a player enters a square without power that contains an active light grenade, that light grenade explodes and the player loses his next 4 actions.
8. If a player is in a square without power at the start of his turn, he can only perform 2 actions in this turn, instead of 3.

### 3.3 Items

#### 3.3.1 Light Grenades

Light grenades are items that can be picked up and used by a player.

1. When a player uses a light grenade, the light grenade is removed from his inventory and placed on the current square of the player. Initially, the light grenade is inactive. The light grenade becomes active when the player leaves the square.
2. An active light grenade is invisible and cannot be picked up.
3. A player cannot use multiple light grenades on the same square.
4. When any player enters a square with an active light grenade, it explodes. The player that drove over the light grenade is blinded and loses his next 3 actions (or his next 4 actions if the light grenade is on a square without power).
5. An exploded light grenade is not active.
6. Active and exploded light grenades cannot be picked up.

At the start of the game a number of inactive light grenades is placed on the grid.

1. The placement of light grenades is random.
2. At the start of the game, 5% of the squares contains a light grenade, rounded up to an integer value.
3. A light grenade cannot be placed on a wall.
4. There can be at most a single light grenade on each square.
5. The starting position of a player cannot contain a light grenade.
6. For each player, at least one light grenade is placed in the area of 3x3 squares that covers the starting position of that player.

## 4 Domain Model

Figures 2 and 3 show the domain model of Objectron.

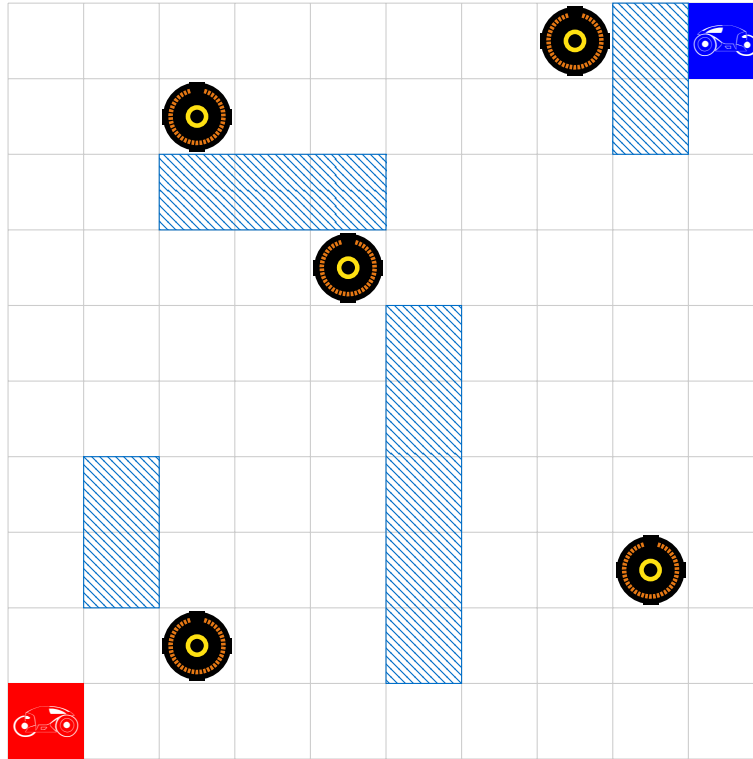


Figure 1: Example of an initial game setup.

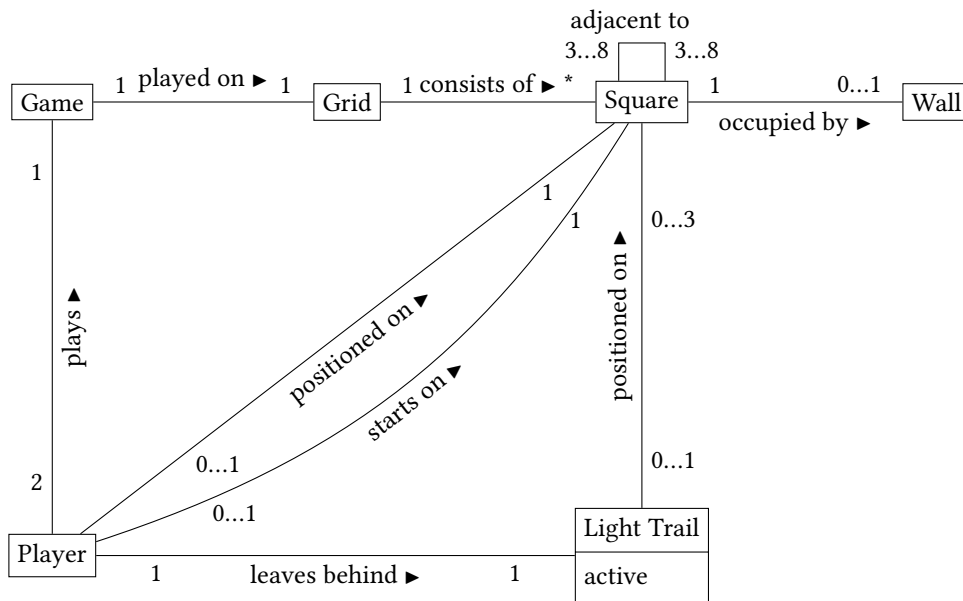


Figure 2: Domain model, **part 1**.

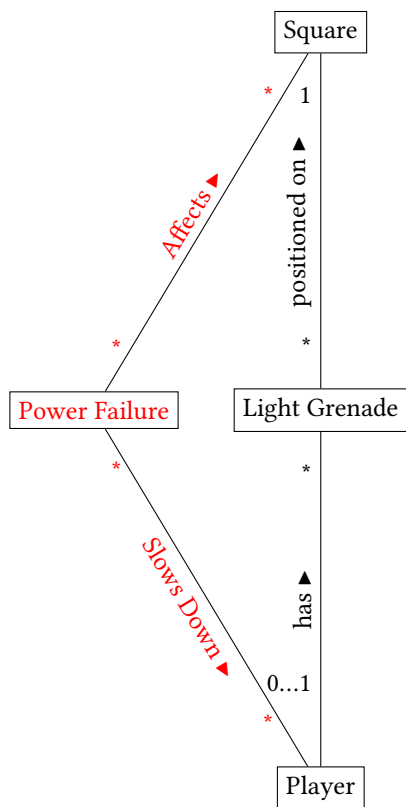


Figure 3: Domain model, part 2.

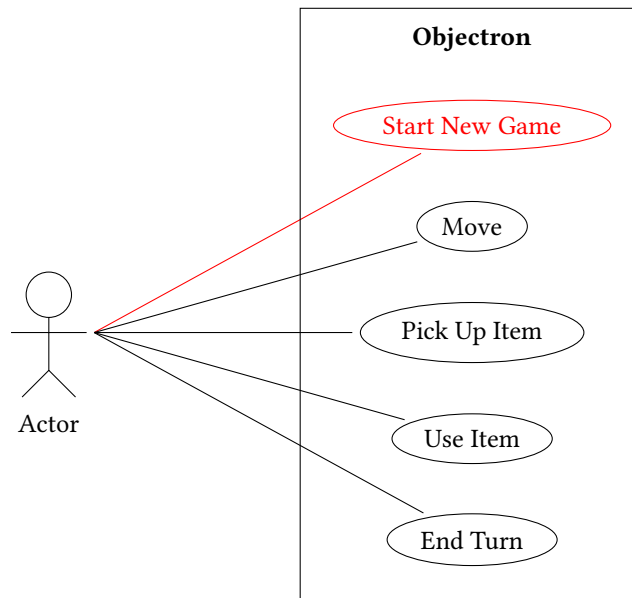


Figure 4: Use case diagram for Objectron.

## 5 Use Cases

Figure 4 shows the use case diagram for Objectron. The following sections describe the various use cases in detail.

The following use case was missing from the assignment of iteration 1:

### 5.1 Use Case: Start New Game

**Primary Actor:** Any player.

**Success End Condition:** A new game has been started.

**Main Success Scenario:**

1. The user indicates he wants to start a new game.
2. The user indicates the dimensions of the grid he wants to play on.
3. The system starts a new game with a grid of the given dimensions, according to the rules described in Section 3.

**Alternate Flow:**

2. (a) The user indicates invalid dimensions for the grid.
3. The system notifies the user that the specified dimensions are invalid.
4. The use case returns to step 2.

### 5.2 Use Case: Move

**Primary Actor:** The player whose turn it is.

**Precondition:** It is the player's turn, and the player has performed less than 3 actions in the current turn.

**Success End Condition:** The player has moved.

**Main Success Scenario:**

1. The player selects a direction in which he wants to move.
2. The system moves the character of the player 1 square in the selected direction.
3. The system adds 1 to the number of actions that the player has performed during this turn.

**Alternate Flow:**

1. (a) The user selects a direction that would position his character on a square that is covered by a wall.
2. The use case ends.

**Alternate Flow:**

1. (b) The user selects a direction that would move his character off the grid.
2. The use case ends.

**Alternate Flow:**

1. (c) The user selects a direction that would position his character on a square that is covered by the other player.
2. The use case ends.

**Alternate Flow:**

1. (d) Moving in the selected direction would let the character of the player cross a light trail.
2. The use case ends.

**Alternate Flow:**

3. (a) The player has reached the starting position of the other player.
4. The player wins the game.

### **5.3 Use Case: Pick Up An Item**

**Primary Actor:** The player whose turn it is.

**Precondition:** It is the player's turn, and the player has performed less than 3 actions in the current turn.

**Precondition:** If the player has not yet moved during this turn, he must have more than one action left.

**Success End Condition:** The user receives one of the items on the current square.

**Main Success Scenario:**

1. The user indicates he wants to pick up an item.
2. The system presents a list of items on the current location.
3. The user selects an item from the list.
4. The system removes the item from the gameboard at the position of the player and puts it in the inventory of the player.
5. The system adds 1 to the number of actions that the player has performed during this turn.

**Alternate Flow:**

1. (a) There are no items on the current square of the player.
2. The use case ends.

**Alternate Flow:**

1. (b) The user already has six items in his inventory.
2. The use case ends.

## 5.4 Use Case: Use An Item From The Inventory

**Primary Actor:** The player whose turn it is.

**Precondition:** It is the player's turn, and the player has performed less than 3 actions in the current turn.

**Precondition:** If the player has not yet moved during this turn, he must have more than one action left.

**Success End Condition:** The user has used an item.

**Main Success Scenario:**

1. The player selects an item from the inventory.
2. The user indicates he wants to use an item.
3. The system lets the character of the player use the item.
4. The system adds 1 to the number of actions that the player has performed during this turn.

**Alternate Flow:**

1. (a) There are no items in the inventory of the player.
2. The use case ends.

**Alternate Flow:**

3. (a) The user decides he no longer wants use an item and cancels the use case.
4. The use case ends.

## 5.5 Use Case: End Turn

**Precondition:** It is the player's turn, and the player has performed less than 3 actions in the current turn.

**Primary Actor:** The player whose turn it is.

**Success End Condition:** The turn of the player ends.

**Main Success Scenario:**

1. The user indicates that he wants to ends his turn.
2. The system asks for confirmation and warns the user that we will lose the game if he did not yet move during this turn.
3. The user confirms
4. The system lets the character of the player perform an empty action for each remaining action in this turn.

**Alternate Flow:**

3. (a) The player does not confirm

4. The use case ends.

Good luck!

The SWOP Team members