OBJECTRON

SWOP 2012-2013

ITERATION 4

Contents

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 five 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 - 12/04/2013 at 18:00
- Iteration 4: 15/04/2013 - 03/05/2013 at 18:00
- Iteration 5: 06/05/2013 - 24/05/2013 at 18:00

2.2.1 Late Submission Policy

If the zip file is submitted N minutes late, with $0 <= N <= 240$, 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.

For certain functionality you are required to provide class and method documentation as taught in previous courses. Section 3 indicates which functionality should be documented. The documentation is limited to the domain layer. Overriding methods do not have to be documented, but the corresponding overridden methods should be documented.

When designing and implementing your system, you should enforce class and representation invariants. 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. Note that not all invariants are stated explicitly in the assignment. You are expected to think about reasonable invariants that forbid program states that make no sense. Make the distinction between implicit and explicit rules in the assignment. Implicit rules are not written down and should therefore not have a large impact on your design when they change. An example of an implicit rule is typically: ``there is only one instance of the system running at a time''.

You can optionally use tools such as *Contracts For Java*, and *NonNullCheckWeaver* to enabled 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 each use case, there should be a dedicated scenario test class. For each use case flow, there should be at least one test method that tests the flow. Scenario tests should not only cover success scenarios, but also scenarios for all kinds of faulty user input.

You determine to which extent you use unit testing. The lead tester briefly motivates the choice during the evaluation of the iteration.

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[1]. 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.

Group all tests in different source trees or in different packages depending on their scope. Using separate source trees allows you to still define the test classes in the packages of your domain such that you can still access package accessible functionality. We distinguish the following scopes.

1. Unit tests: the scope is a single class
2. Intermediate scenario test: the scope contains multiple classes, but less than an entire use case.
3. Use case scenario test: the scope is a single use cases
4. User session scenario tests: the scope contains multiple use cases

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

Exactly *one* person of the team hands in an electronic ZIP-archive via Toledo. The archive contains the items below and follows the structure defined below. *Make sure that you use the prescribed directory names!*

- `groupXX` (where XX is your group number (e.g. 01, 12, ...))
  - `doc`: a folder containing the Javadoc documentation of your entire system
  - `src`: a folder containing your source code
  - `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 the `svn export` command, which removes unnecessary repository folders from the source tree.

[1] http://www.eclemma.org

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.

The evaluation of an iteration is planned in the week after that iteration, unless that week is part of a holiday. In the latter case, the defense is either in the week after the holiday, or during the holiday if all group members and the advisor agree.

Immediately after the evaluation is done, you mail the PDF file of your presentation to your advisor. Include the design diagrams that you do not discuss during the presentation after the last slide of your presentation (like an appendix).

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. An updated version of the domain model that includes the added concepts and associations.
2. A discussion of the high level design of the software.
3. A more detailed discussion of the parts that you think are the most interesting in terms of design.
4. Which GRASP and design patterns you used.
5. 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,...).
6. A discussion of the testing approach used in the current iteration.
7. An overview of the project management. Give an approximation of how many hours each team member worked. Use the following categories: group work, individual work, and study (excluding the classes and exercise sessions). In addition, insert a slide that describes the roles of the team members of the current iteration, and the roles for the next iteration. Note that these slides do not have to be presented, but we need the information.

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 For The Next 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 4 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 between the other player and 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 does not move in an action of his turn, he is trapped and loses the game. If a player has no actions left in a turn due to movement penalties (see further), he does not automatically lose the game. As a consequence, when a player gets a penalty during his turn and he still had at least one action left, he does not 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

At the start of the game, the players can choose to either play on an automatically generated grid (as in the previous iterations) or on a grid that is read from a file.

3.2.1 Automatically Generated 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.

3.2.2 File-based grid

The shape of the grid on which the game is played, is read from a file. This file has a simple text format:

1. Each line of the file corresponds to a row of the grid and each character in a line corresponds to a cell in a row.
2. Different characters correspond to different kinds of cells:
   - A space corresponds to a free square that is part of the grid.
   - An asterisk `*' corresponds to a location that is not part of the grid.
   - A number sign `#' corresponds to a square that is part of the grid and contains a wall segment.
   - An integer number 1 or 2 corresponds to a square that is part of the grid, on which a player can start.
   - Other characters (except for newlines) are invalid.
3. When reading a grid from a file, the game must check that the grid adheres to the following rules:
   - There must be a path from each free square that is part of the grid to each other free square that is part of the grid. That is, there can be no unreachable `islands' of squares that are part of the grid.
   - There must be exactly two starting locations.
   - There is a path between the two starting locations.

3.2.3 Walls

If the game is played on a grid read from a file, the walls are specified in the grid input file (see above). Otherwise, 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.

3.2.4 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. Some examples of power failures are shown in orange on the grid at the end of this section. Power failures behave according to the following rules:

1. Power failures are visible on the grid.

2. At the start of each turn, each square that is part of the grid has a 1% chance of losing power. These power failures are called primary power failures.

3. As long as a primary power failure occurs on a square, this causes a secondary power failure on a single adjacent square as follows. When the primary power failure starts, one randomly chosen surrounding square has a secondary power failure. This secondary power loss then starts rotating around the source of the power loss. After every 2 actions, the secondary power loss moves by one square in the direction of the rotation. The direction of the rotation is chosen randomly at the start of the power failure (with a 50% chance of clockwise rotation and a 50% chance of counter clockwise rotation).

4. A secondary power failure causes a tertiary power failure. A power failure does not spread any further than that. A tertiary power failure occurs on a single square randomly chosen from the following three squares (T1, T2, T3) adjacent to the secondary power failure (F2). T1 is on a line with the primary (F1) and secondary (F2) power failures. T2 and T3 are adjacent to both F2 and T1, and are each either north, east, south, or west of T1. The duration of a tertiary power failure is 1 action.

5. A square can be affected by secondary and tertiary power failures from primary power failures.

6. The selection procedures for the secondary and tertiary power failures can select ``squares'' that are off the grid. When an off-grid square is selected for a power failure, that power failure does not occur since there is no grid to affect.

7. A primary power failure lasts for 3 turns.

8. A secondary power failure lasts for 2 actions.

9. A tertiary power failure lasts for 1 action.

10. Squares containing active light grenades can lose power, and this increases the impact of the light grenade on detonation (see further).

11. Power failures have no influence on other items (such as inactive light grenades) or the light trail.

12. If a player enters a square without power and that square does not contain an active light grenade, his turn ends.

13. 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.

14. If a player is in a square without power at the start of his turn, he loses an additional action on top of any existing penalties.

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, 2% of the squares that are part of the grid contains a light grenade, rounded up to an integer value. If this is impossible due to other constraints, then less than 2% of the squares can contain a light grenade, but the number of squares having one should be as close as possible to 2%.
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.

3.3.2 Identity discs

Identity discs are items that can be picked up and used by a player to shoot at other players.

1. When a player uses an identity disc, the disc is shot in a direction chosen by the user. The direction can be up, down, left or right (not diagonal).
2. The maximum range of a shot is 4 squares: if there is no grid boundary, wall, or player in the trajectory of the identity disc, it will end up on the 4th square from the player in the direction the disc was shot. Consequently, the identity disc can hit players or walls that are at most 4 squares away from the player that launches it.
3. If the identity disc hits a player, that player skips his next turn and the disc ends up on the square of the player that was hit.
4. If the identity disc hits a wall or a grid boundary, it ends up on the square in front of the wall/boundary. In this case, no player is hit by the disc.
5. If the identity disc enters a square with a teleporter (see below), it is teleported to the square of the corresponding destination teleporter and continues on its path.

6. Each square without power on the trajectory of the identity disc, including the square from which the disc was launched, decreases the total range of the identity disc by 1.
7. A previously launched identity disc that has ended up on some square of the grid can be picked up again by any player.
8. Identity disks stop, no matter what, after they have teleported through the same teleporter twice. This prevents infinite loops when teleporters form a loop in the flight path of an identity disc. So if we have teleporters A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A that form a loop, the charged disk would follow that path and stop after going through A for the second time (so it lands on B).
9. If a player hits himself with an identity disc (via a teleporter loop), he loses both the remaining actions in the current turn, and his entire next turn.

At the start of the game a number of identity discs are placed on the grid.

1. The placement of identity discs is random.
2. At the start of the game, 2% of the squares that are part of the grid contains an identity disc, rounded up to an integer value. If this is impossible due to other constraints, then less than 2% of the squares can contain an identity disc, but the number of squares having one should be as close as possible to 2%.
3. An identity disc cannot be placed on a wall.
4. Initially, there can be at most a single identity disc on each square.
5. The starting position of a player cannot contain an identity disc.
6. For each player, at least one identity disc is placed in the area of 5x5 squares that covers the starting position of that player.

3.3.3 Charged identity disc

Charged identity discs are items that can be picked up and used by a player to shoot at other players.

1. A charged identity disc behaves the same as an uncharged identity disc, except that it has an unlimited maximum range.

At the start of the game, at most one charged identity disc is placed on the grid.

1. The charged identity disc is placed randomly on the grid.
2. The charged identity disc must be reachable in about the same number of actions by each player. More precisely, the disc must be placed such that the length of the shortest path from each player to the disc differs by at most 2 squares.
3. A charged identity disc cannot be placed on a wall.
4. Initially, there can be at most a single charged identity disc on each square.
5. The starting position of a player cannot contain a charged identity disc.
6. If it is impossible to place the disc such that it meets the above constraints, no disc is placed on the grid.

3.3.4 Teleporter

Teleporters are items that can be used by a player, but they cannot be picked up.

1. Each teleporter has a fixed destination teleporter. When a player steps onto a teleporter, that player is relocated to the square of the destination teleporter. His light trail travels with him trough the teleporter, so it can be partly at the source location and partly at the destination location.
2. Since a player is immediately teleported to the destination teleporter, he has no chance to pick up any items from the source teleporter location, but he can pick up items on the destination teleporter location (if he has an action left in his turn).
3. The destination of a teleporter cannot be that teleporter itself.
4. If the destination of a teleporter A is teleporter B, then the destination of teleporter B is *not necessarily* teleporter A.
5. A player is unable to step onto a teleporter if another player is on the square of the corresponding destination teleporter.
6. A player takes only one hop at a time. If the destination of teleporter A is teleporter B, and the destination of teleporter B is teleporter C, then the player is teleported to B when stepping on A. To use teleporter B, the player must first leave and the re-enter the square of teleporter B.

At the start of the game, a number of teleporters is placed on the grid.

1. Teleporters are placed randomly on the grid.
2. The destination of each teleporter is chosen randomly.
3. At the start of the game, 3% of the squares that are part of the grid contains a teleporter, rounded up to an integer value. If this is impossible due to other constraints, then less than 3% of the squares can contain a teleporter, but the number of squares having one should be as close as possible to 3%.
4. A teleporter cannot be placed on a wall.
5. There can be at most a single teleporter on each square.
6. The starting position of a player cannot contain a teleporter.

3.3.5 Force Field Generators

Force field generators are items that can be picked up by a player.

- A force field generator can be picked up by a player.
- When the player uses the force field generator, it is put on the current square of the player.
- A force field generator is activated automatically when there is another force field generator on a square that is at most 3 squares away from it in a straight line in the eight main directions: N,E,S,W,NE,SE,SW,NW. Only those exact directions count.
- Force field generators in the inventory of a player are always inactive and do not influence other force field generators.
- When two force field generators are within 3 squares of each other in the main directions, and the line segment between both generators does not cross a wall, a force field is generated between them.

- A force field forms a straight line between its two generators, and includes the squares of these generators.
- A force field switches on and off after every 2 actions that have been performed. When a player uses a force field generator, all potential force fields generated by that generator are in the off state.
- A player cannot pas through a force field, or enter one of the squares of the force field.
- When a force field is activated and a player is present on one of its squares, that player cannot move until the force field deactivates.
- When a player is released from a force field, and is trapped again by that force field the next time it is activated, the player loses the game. Note that this is an exception to the rule that prevents loss due to penalties.
- An identity disc is destroyed when it hits a force field.

At the start of the game, a number of portable force fields is placed on the grid.

1. Force field generators are placed randomly on the grid.
2. At the start of the game, 7% of the squares that are part of the grid contains a force field generator, rounded up to an integer value. If this is impossible due to other constraints, then less than 7% of the squares can contain a force field generator, but the number of squares having one should be as close as possible to 7%.
3. A force field generator cannot be placed on a wall.
4. There can be at most a single force field generator on each square, excluding those in the inventory of a player.
5. The starting position of a player cannot contain a force field generator.

3.4 Required Documentation

The domain functionality with respect to movement and how the side effects of movements are triggered should be documented.

4 Domain Model

Figures 2 and 3 show the domain model of Objectron.

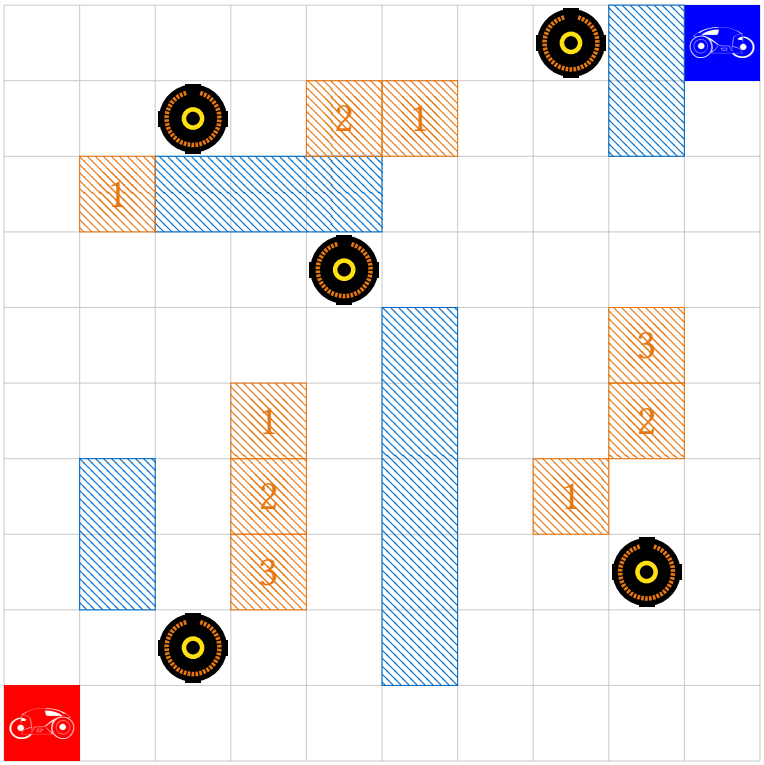

Figure 1: Example of an initial game setup.

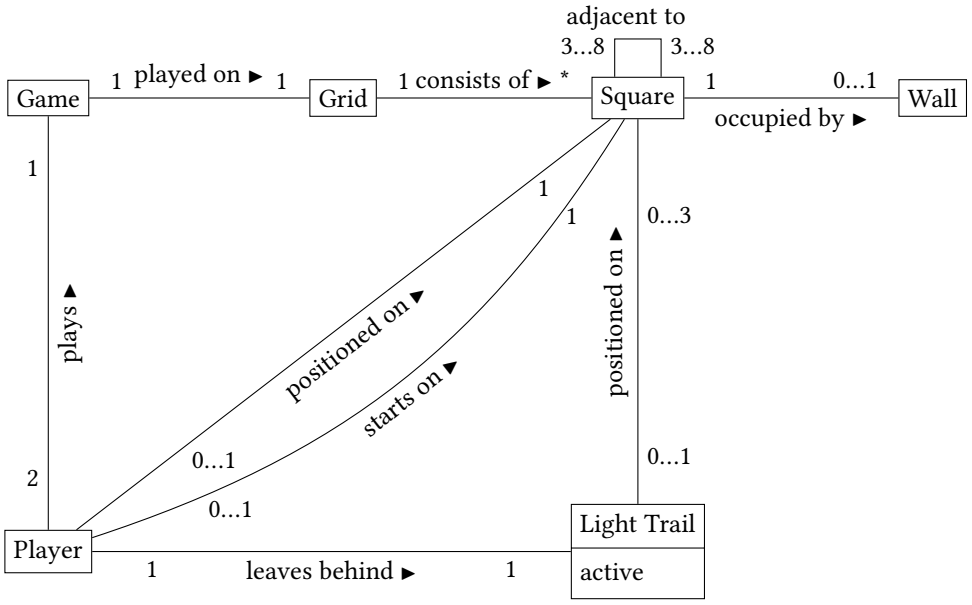

Figure 2: Domain model, part 1.

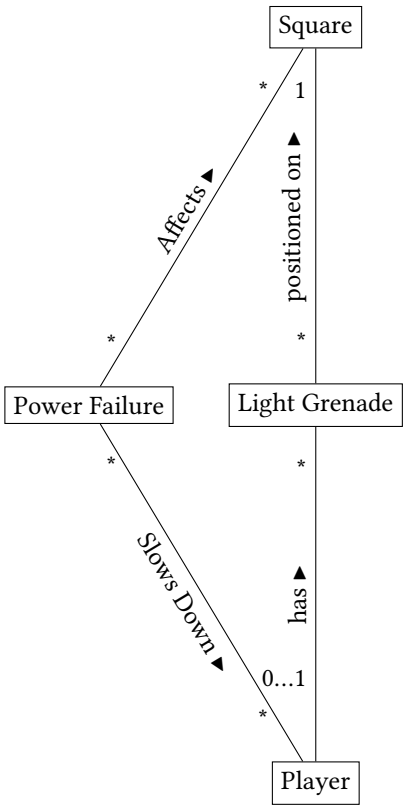

Figure 3: Domain model, part 2.

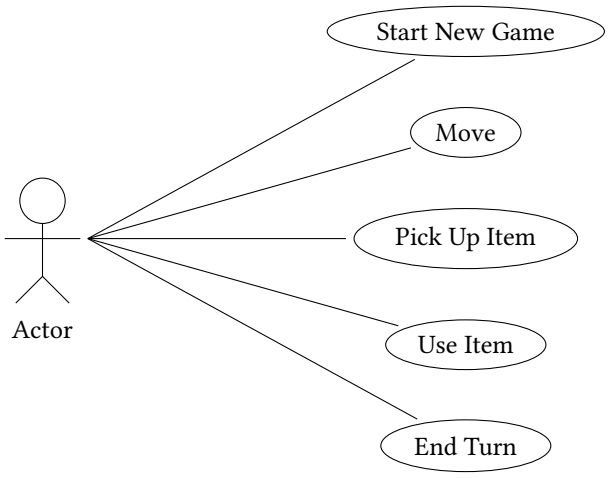

Figure 4: Use case diagram for Objectron.

5 Use Cases

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

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 on an automatically generated grid.
2. [Include use case 'Choose Generated Grid']
3. The system starts the new game.

Alternate Flow:

1. (a) The user indicates he wants to start a new game on a grid loaded from file.
2. [Include use case 'Choose Grid From File']
3. The use case continues at step 3 of the main flow.

5.2 Use Case: Choose Generated Grid

Main Success Scenario:

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

Alternate Flow:

1. (a) The user indicates invalid dimensions for the grid.
2. The system notifies the user that the specified dimensions are invalid.
3. The use case returns to step 1.

5.3 Use Case: Choose Grid From File

Main Success Scenario:

1. The user indicates the file to load.
2. The system constructs a grid based on the specified file, according to the rules described in Section 3.2.2.

Alternate Flow:

1. (a) The user specifies an invalid file.
2. The system notifies the user that the selected file is invalid.
3. The use case returns to step 1.

5.4 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.

5.5 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.

5.6 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. [Extension point: "Choose (Charged) Identity Disc Direction"]
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.

5.7 Use Case: Choose (Charged) Identity Disc Direction

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 chosen a direction for a (charged) identity disc

Main Success Scenario:
After step 2 in the basic flow of "Use An Item From The Inventory", if the item to use is an identity disc or a charged identity disc, then

1. The player indicates the direction (up, down, left, right) in which to shoot the disc.
2. The basic flow of "Use An Item From The Inventory" is resumed at step 3.

Alternate Flow:

1. (a) The user decides he no longer wants use an item and cancels the use case.
2. This use case and the "Use An Item From The Inventory" use case end.

5.8 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