Principles of Software Engineering
2805ICT: System and Software Design
3815ICT Software Engineering
7805ICT Principles of Software Engineering
Milestone One
(MODEL SOLUTION)
Weight 10%

Vladimir Estivill-Castro

July 18, 2020

1 Objectives

1. Understand the differences between conceptual design and physical design.
2. Carry out some initial steps in the process to create conceptual designs.
3. Create appropriate system models for the representation of functional requirements by capturing use-cases.
4. Use a design paradigm to design a simple prototype of a software system.
5. Select an appropriate software architecture as the design basis for a given software requirements specification, justify the selection based on its advantages over alternative architectures.
6. Create software programs that make use of appropriate design patterns.
7. Create user interface software using either event driven or callback based designs
8. Elaborate prototypes building a GUI using a standard widget set.
9. Discuss the properties of good software design including the nature and the role of associated documentation.
10. Create appropriate design documentation for a variety of different designs.

2 Requirements

2.1 Functional Requirements

Functional requirements are descriptions of what does the system achieve, its functionality: a functional requirement describes what is the response to a given input. The set of functional requirements describe what the system can do for its users. Therefore, the functional requirements detail the behaviour of the system. The functional requirements define legal inputs, and the corresponding outputs.

We use a table format to provide requirements similar to the one used by the digital textbook [2, Chapter 2]. There are more formal ways of specifying requirements as per IEEE standards.

Identifier	Priority	Functional Requirement
F-R 1	5	The system offer a graphical user interface (GUI) that allows a player to attempt to complete a challenge as a game, the challenge is to locale all hidden mines in the visual representation of the environment.
F-R 2	5	The system should enable a menu-item option for the player to select different difficulty levels. The levels consist of larger environments of the same type. The density of mines in an environment must be roughly the same across difficulty levels.
F-R 3	5	The system should enable a menu-item option for the player to select different environment. There should be 3 environments: square-grid, hexagonal grid, and a graph layout.
F-R 4	5	The system should display a timer in seconds. Starting the game starts the count down of the timer. The game dose not end if the timer expires.
F-R 5	5	The system should display the total number of mines requiring localisation in the environment.
F-R 6	5	The system should offer a start button so the player commences the challenge. Once the game starts, the choice of environment or level become disabled until the game finishes. Until the start button is pressed actions on the environment are disabled.
F-R 7	5	The game finishes when all mines are located or when the user steps (uncovers) a mine. This halts the timer count down, displays location of all remaining mines.
F-R 8	5	The system should offer a display with the initial level of mines. Every time the user flags a mine correctly, the display decreases the number of mines to be identified. The mine display does not decrease for a wrongly flagged location for a mine.
F-R 9	5	The system should offer an opportunity to reset the game and select a new challenge after the previous game is over.
F-R 10	5	The system should offer visual feedback as to whether the game was completed by all mines being identified or because of stepping on a mine.
F-R 11	4	The score of a player for a level should account for minimising time and minimising wrong placement of flags.
F-R 12	5	Once the game starts, the start button is disabled and the user can click on regions of the environment to 1) step and reveal the environment information or 2) to flag the environment location. Swapping the type of click should be a simple variation of the other: for instance, revealing a location is a simple click, while flagging the location is a SHIFT-click.
F-R 13	1	The game offers a multi-player option where two players receive the same environment and they race against each other to locate all mines. In this case, placing an incorrect flag to stepping on a mine terminates the game. The player who completes the location of the mines before anyone else or is the last player standing wins the game.
F-R 14	3	The game sets the mine in an environment using random choices that are extremely difficult for the players to anticipate. Re-staring the game results in new mine placements, so terminating the application and restarting it does not recycle maps of mine locations.
F-R 15	3	The user can resize the environment and according to this the information for a cell becomes larger and more visible.
F-R 16	2	The user can configure elements of the environment, such as the colours of numbers, or icons in the game.
F-R 17	2	The user can configure musical and sound feedback to main consequences, such as end of the game, completing the mine localisation or restarting the game (timer counting down has some sound feedback).
F-R 18	4	In the hexagonal and squared versions of the game, revealing a location whose hidden information is zero triggers a flooding algorithm that reveals all adjacent locations that also have zero as their number and the boundary of the region with zero.
F-R 19	3	When a user completes a level satisfactorily, they can enter their name and record their best result for that level. Later, players can use a menu-item to review their scores.
F-R 20	5	A configuration of mines results is initially hidden from the player. Cells in the squared and grid environment either hold a mine or the total number of adjacent cells that hold a mine. User revels the hidden number by clicking (stepping) on the cell. In the squared grid the number 1 is presented in blue, the 2 in green, the three in red, 4 in dark grey, five in magenta, 6 in yellow, 7 in cyan, and 8 in pink.
F-R 21	5	At all times there should be an option for the user to exit the game.
F-R 22	1	The user offers and option to configure all parameters of the game, such as the size (and layout) of the environment and the density of mines.
F-R 23	1	There is an *undo* option that allows the user to retire a flag from a previously indicated mine location. Retiring a flag placed on a mine is not a game-ending situation unless the player steps on the location (after the undo).

2.2 One Use Case

We present first an example template for an actor.

Actor: User	
Description: The user of the software (a player). The user interacts with the software using mainly a mouse (but in requirements that record the user name, the keyboard, or when no mouse is available, a keyboard operational option must be available). The user expects the software not to exhibit faults, in particular, disclosed information (sums of neighbouring mines to a cell must be correct).	
Aliases: Player	**Inherits:**

2.2.1 One Documented template

Templates for use cases are available in the lecture notes or in the digital textbook [2, Figure 2-19] and illustrated with examples in the digital textbook [2, Page 101-102].

Use case: Play game on squared grid

Requirements satisfied: Requirement F-R 1

Brief Description: The player interacts indicating a cell to uncover or to flag. Uncovering a cell reveals if there is a mine or not. Uncovering a cell (stepping into a cell) that holds a mine finishes the game. Uncovering a location without a mine reveals the how many adjacent cells hold mines. The player must use information revealed to infer and flag locations of mines until all mine locations are identified in the smallest amount of time.

Actor: User

Priority: 5

Risk: This requirement is essential, anything else depends on this being implemented. Thus, not being able to complete this causes major setback in development. There is also a risk of not being able to interpret the rules for computing the hidden information, and more delicate, the flooding of zero-value regions.

Pre-conditions: The application started correctly or the game was reset correctly after a previous game. The user has selected difficulty level and squared environment.

Post-conditions: The game reaches its end with a player locating all mines, or the player stepping on a mine. In the first case, a score is computed and could be recorded. In the case of stepping on a mine, remaining uncovered mines are revealed.

Basic flow:

1. Initially, All locations (cells) of the game are covered and the timer starts to count down. The number of hidden cells appears in a display.
2. Iteratively, the user changes the state of covered locations. Locations already uncovered cannot change state.
   (a) The user reveals an covered location
      i. If the user reveals a location that holds a mine, the game ends.
      ii. If the user reveals a location that does not hold a mine, the total number of mines in adjacent cells is revealed.
   (b) or flags an covered location.
      i. The user flags the last mine correctly, the game ends and the player earns a score.
      ii. The user flags a mine correctly, the number of un-located mines decreases by one.
      iii. The user flags a location that does not hold a mine, nothing happens. But the user's score, if successful is penalised for abuse of flags.

Activity Diagram: Refer to Figure 1.

Alternative flow: The user exists the game before reaching the end.

2.2.2 One activity diagram

Activity diagrams are common to document use cases and requirements.

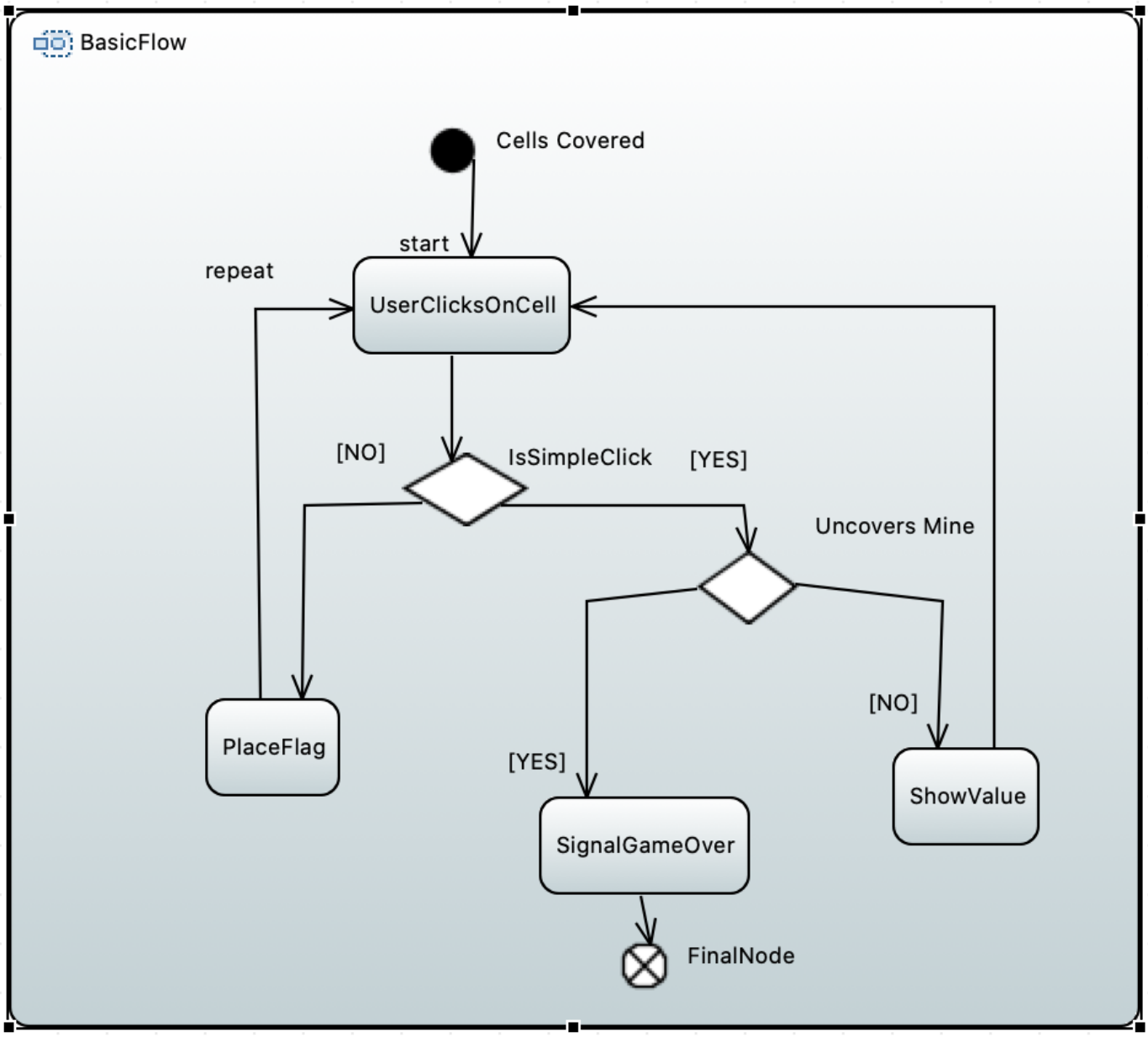


Figure 1: Activity diagram for basic flow of use case Play game on squared grid.

2.3 One Use Case Diagram

We must have at least one actor, at least 4 use cases and some illustration of the *extends* and the *includes* relationship.

2.4 Non-Functional Requirements

See the following from the digital textbook [2, Page 75-76]

> "The term FURPS+ refers to the non-functional system properties:
>
> **Functionality** lists additional functional requirements that might be considered, such as security, which refers to ensuring data integrity and authorized access to information
>
> **Usability** refers to the ease of use, aesthetic, consistency, and documentation — a system that is difficult and confusing to use will likely fail to accomplish its intended purpose
>
> **Reliability** specifies the expected frequency of system failure under certain operating conditions, as well as recoverability, predictability, accuracy, and mean time to failure

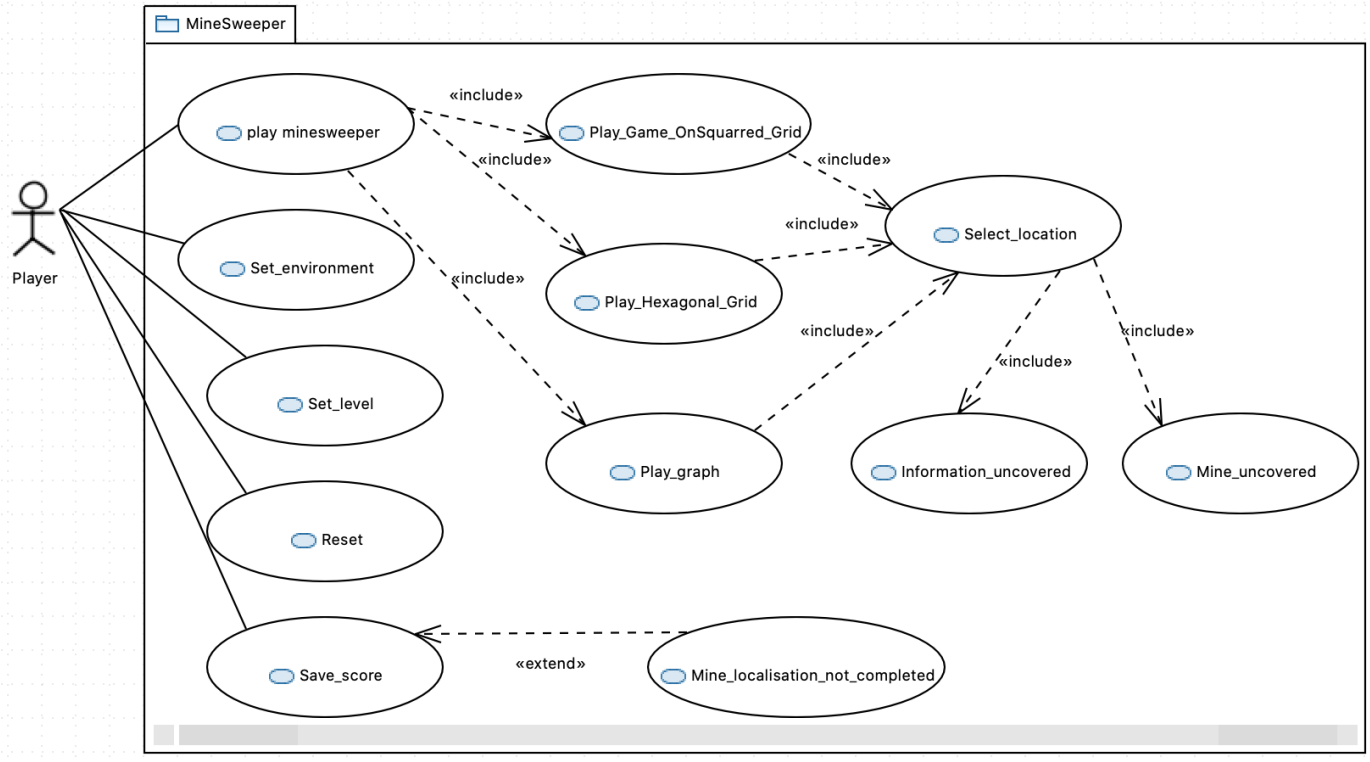


Figure 2: Use case diagram for the package `Minesweeper`.

Performance details the computing speed, efficiency, resource consumption, throughput, and response time

Supportability characterizes testability, adaptability, maintainability, compatibility, configurability, installability, scalability, and localisability.

Therefore, it is important to emphasise the quality property that the non-functional requirement focuses on.

Identifier	Priority	Non-Functional Requirement
NF-R 1	5	*Re-usable*: The classes that are created for the Mine-sweeper project must be designed for maximum re-usability. The algorithm for flooding the zero-number zone should be agnostic to the type of grid.
NF-R 2	5	*Extensible*: The application should include three difficulty levels, but that could increase, and the application should be able to smoothly handle that change. The introduction of a new level should be localised to at most two classes.
NF-R 3	5	*Usability*: The graphical user interface should be intuitive, requiring little or no instruction to use. It should maintain standard conventions that people expect with buttons and menu options. The default action should be the simplest mouse click.
NF-R 4	3	*Enjoyable*: Players shall find the games challenging, enjoyable and not impossible or too easy. The density of mines should allow for some progress in all levels.
NF-R 5	4	*Reliable*: The application should work as it is intended to and not crash or throw unhandled exceptions. Once the game has started, no combination of mouse clicks of text shall crash the game.
NF-R 6	5	*Performance*: Being a small sized game, mine sweeper is designed to respond quickly. The game shall run smoothly with a standard home desktop or a laptop that has a processor of minimum 1.0 GHz or above. The game should respond quickly when user make any move in the game. Every movement shall be processed by less than 0.5 seconds; that is, the user shall not experience any lagging.
NF-R 7	4	*Performance*: There could be complex algorithms to generate the placement of mines or a complex flooding algorithm for connected regions with zero values; nevertheless, the response time experienced by the user should be less that 0.5 seconds.
NF-R 8	5	*Security*: The game will not access any private data user's computer, neither will not change any files (unless its score keeping records). The users are protected from making errors in the game since there are limited components on the game's interface. Even if the game crashes, it will not affect any other files on user's computer since it shall have no files open.
NF-R 9	5	*Appropriate Error handling*: User clicks (or selections) would cause the system to react (or offer a response) only if the click (or selection) is appropriate and constitutes adequate input at that state of the game. For instance, the game area is disabled until the game starts. Once the game starts, the user can only continue playing until a game outcome or leave the game, but the level or the environment cannot be changed.
NF-R 10	4	*Maintainability*: Mine-sweeper should be kept as a small project with well organised modules and classes. If possible, emulate a model-view-controller for separation of responsibilities. Leaving presentation to the view (the GUI).
NF-R 11	4	*Maintainability*: The project shall offer automatic documentation-generation from the source code. All functions, procedures and classes should be documented.
NF-R 12	1	*Universality*: It should be easy to extend the game to multiple languages, or to enable the user to select their preferred language as there will be minimal text feedback on the game.
NF-R 13	4	*Universality*: Size of buttons and displays should be large enough user do not struggle to select locations and to observe the result.
NF-R 14	4	*Quality of design*: Classes and modules should exhibit a high level of cohesion and little coupling. This design quality aspect must be reflected by observing the connections and links between classes.
NF-R 15	4	*Security*: Players scores should be saved encrypted with a difficult mechanisms to falsify or tamper with scores in the user's computer.

2.5 Constraints

Constraints could be considered functional requirements, but in this case, we give details of aspects we have no choice regarding the application's architecture or scope of the functionality.

Identifier	Constraint
C 1	*Schedule Constraint*: The development must meet the three milestone schedule of the course. It is expected that the square grid and the hexagonal grid are completed by the second milestone.
C 2	*Resources Constraint*: The number of developers is limited by the description in the course profile.
C 3	*Platform Constraint*: The application must run in two different platform demonstrating the platform independence of the code.
C 4	*Architecture Constraint*: The application must offer a client-server version by which the GUI is detached from the intelligence that defines the rules if the game, potentially enabling multi-player versions.
C 5	*Installation Constraint*: Installation requirements shall be minimal. It should not be required to install many libraries or packages to be able to enjoy using the game. For instance, if written in python, the most vanilla installation of python should run the game. If developed in java, the most vanilla installation of java swing should tun the game.
C 6	*Version control Constraint*: During the development, a version control system should be used. The data in the version control system should be used to monitor progress against functional requirements or version releases.

3 Project risk

While the risk of something being incomplete or faulty by the due date cannot be entirely eliminated, we can undoubtedly enumerate risks, plan and take precautions such that each risk is satisfactorily mitigated. The planning for this project attempts to accomplish this by focusing on enumeration and risk reduction strategies.

There are two types of risks that we will encounter in this project.

Software process risks. These are risks of the development. We shall look at what could go wrong with the construction of the software. What could cause that the project is not completed in time, or does not achieve its goals.

Software product. These are risks if the project fails or has a fault. What could go wrong if the project does not meet some requirement.

For the software process risks we list the following risks, and we classify them using a table as suggested in Subsection 2.4.4 of the digital textbook [2, Page 101]. The digital textbook defines *ALARP* to mean *as low as reasonably practical* [2, Page 107]. The table ranks the risk, and includes at least one mitigation strategy.

Risk Identification	Type	Reduction Strategy
Risk 1 Difficulty in generating an hexagonal grid that recognises where the mouse clicks.	Intolerable	Carry out some prototyping. In particular, attached to this submission there is a working prototype for the squared environment made by drawing the environment and not by buttons. Although drawing hexagons is more laborious, the basic functionality issues are resolved.
Risk 2 Difficulty in generating a view with a graph where the user can click on nodes and edges and the software detects these events.	ALARP	Carry out some experimenting. Testing of the `mxgraph` for java has been performed and so far this works fine, although restricts the development to java or JavaScript.
Risk 3 Difficulty in creating a client-server application.	ALARP	Carry out some experimenting. Testing successfully the opening and connecting via sockets in java.
Risk 4 Difficulty in meeting deadlines.	ALARP	Following a visible software process. Starting with a prototype, we will proceed in a spiral approach, adding functional requirements, at least two every week. Emulate a Scrum/agile approach. Monitor the version-control log.
Risk 5 Difficulty in meeting deadlines.	ALARP	Dividing into phases. Project partitioned into three milestones. The square grid prototype is completed. Already using java to ensure is multi-platform. Second phase should complete the hexagonal grid and the client-server version. Last milestone adds the graph version.
Risk 6 Difficulty in meeting deadlines.	ALARP	Develop a prototype. A rapidly built prototype is used to discover or experiment with platforms and algorithms. In particular, drawing a grid on a canvas and retrieving the position of a mouse click.
Risk 7 Developer becomes unavailable due to illness.	ALARP	Ensure procedures for applying for special consideration and understood. Contact details of doctor who can issue medical certificate are accessible.
Risk 8 Difficulty in meeting deadlines.	ALARP	Use software metrics. A course metric of the size of the software is the total lines of code (LOC). Assessing the size of the prototype can produce an estimate of the size of the final product and also of the required effort to plan the workload to meet deadlines.
Risk 9 Difficulty in meeting deadlines.	ALARP	Use version control. We will tag working versions of the software. If we run out of time, we can release the last fully tested version.
Risk 10 Difficulty in creating the flooding algorithm.	ALARP	Making use of reusable components. Create a dedicated class that floods the neighbourhood of a graph with methods that do not specify the geometrical layout, just the adjacency of locations that may hold mines. More experimentation required for whether the flooding makes sense in the third version of the game (the graph/topological version).
Risk 11 Difficulty in organising the code.	ALARP	Making use of a software architecture pattern. The prototype is close to a model-view-controller architecture. There are some aspects that may need revision. This is reflected by the class diagram lifted out of the prototype. Some re-factoring may be required.

However, for risk related to faults or malfunction of the produce, we list the following.

Risk Identification	Type	Reduction Strategy
Risk 12 The game becomes either too difficult or too challenging.	Intolerable	Carry out some testing. Identify if it reasonably difficult to complete a full localisation of mines. Experiment with different sizes and densities and allocated amount of initial time.
Risk 13 User claims damages because of use of the software.	Intolerable	Include GNU General Public License note and absolutely no warranty note in file holding the `main` class but retain the copyright and attribution rights.
Risk 14 The product does not meet the stakeholders expectations.	Intolerable	Demonstrate the prototype to the stakeholders at each milestone for those requirements that are ambiguous or details that require resolution during development.
Risk 15 Users require extensions or changes after initial team of developers has left.	Intolerable	Ensure the code has automatically generated documentation, that there are UML models of the main elements, and that the code can be lifted to models by standard IDE's or modelling tools.
Risk 16 Product is not cross platform.	Intolerable	Ensure the coding is performed in a cross platform standardised programming language and conduct regular testing on alternative platforms.
Risk 17 Security risk: Multi-player client-server version open port. Running a server with a socket open creates a security risk. Also, the game may be tampered by malicious intruder.	ALARP	Advice users that networking security must be a consideration if running the multi-player version. and conduct regular testing on alternative platforms. Encrypt protocol between client and server.

4 Feasibility Report

Minesweeper is a game developed that has predecessors back in 1960. Certainly it can be programmed. Moreover, it we have much better technology today. We have a broader understanding of Object-Oriented Technologies. There are no limitations in the non-functional requirements on the foot-print (amount of memory) that the running program can take. So, we can chose a programming language like `java` or `python` and pay a small price in performance for ease of development. There are now rich Integrating Development Environments (IDEs), so it is clear that realising a version of the game is possible.

Moreover, The current prototype is just over one thousand lines of java code (refer to Table 1). This document

Table 1: Detail of files (and classes) and lines of java code (LOC).

Lines of code	File
50	`FloodUncoveringZero.java`
218	`GlobalModel.java`
140	`GridView.java`
96	`Images.java`
323	`MineSweeperPrototype.java`
49	`MouseListenerForGridView.java`
130	`SquareGrid.java`
1006	**TOTAL**

itself is approximately 4,000 words. This data allows to make the following feasibility assessment. Minesweeper is a moderately small sized game. An individual developer can complete the project with a single computer, access to the Internet for a version control repository (for instance github.com) and a study or office space.

We conduct and analysis of the earlier listed functional requirements and their likelihood to be completed.

Identifier	Likelihood to be successfully implemented	
	Score	**Reason**
F-R 1	High	We have a prototype where we have drawn the gird and identified muse clicks on it. The functionality of the timer and counting down mines is already implemented. Test have been successful in drawing graphs.
F-R 2	High	Already implemented in the prototype.
F-R 3	High	While the hexagonal grid has not been drawn, hexagons can be organized in rows and columns to also use a coordinate system (i, j) as the identifier for a hexagon in the environment.
F-R 4	High	Already implemented in the prototype and no major change.
F-R 5	High	Already implemented in the prototype and no major change for hexagons, maybe more challenging for graphs.
F-R 6	High	Already implemented in the prototype and no major change.
F-R 7	High	Already implemented in the prototype and no major change. Already implemented in the prototype and no major change for hexagons, maybe more challenging for graphs.
F-R 8	High	Already implemented in the prototype and no major change for hexagons, maybe more challenging for graphs.
F-R 9	High	Already implemented in the prototype and no major change for hexagons and for graphs.
F-R 10	High	Already implemented in the prototype and no major change for hexagons and for graphs.
F-R 11	Moderate	Not implemented in the prototype.
F-R 12	High	Already implemented in the prototype and no major change for hexagons and for graphs.
F-R 13	Moderate	Not implemented in the prototype.
F-R 14	High	Already implemented in the prototype and no major change for hexagons and for graphs.
F-R 15	Moderate	Not implemented in the prototype although the code to resize has been tested. See video.
F-R 16	Moderate	Not implemented in the prototype.
F-R 17	Low	Nothing tested or experimented in this regard.
F-R 18	High	Already implemented in the prototype and no major change for hexagons. Experimentation needed for graphs.
F-R 19	Low	Nothing tested or experimented in this regard.
F-R 20	High	Already implemented in the prototype and no major change for hexagons and for graphs.
F-R 21	High	Already implemented in the prototype and no major change for hexagons and for graphs.
F-R 22	Moderate	Nothing implemented but just requires a menu option and user to enter values in widget.
F-R 23	Moderate	Nothing implemented.

The following is the analysis of the non-functional requirements.

Identifier	Likelihood to be successfully implemented	
	Score	**Reason**
NF-R 1	High	Already implemented in the prototype and no major change for hexagons. Experimentation needed for graphs.
NF-R 2	High	Already implemented in the prototype as an enumerated type and dynamic array.
NF-R 3	High	
NF-R 4	High	
NF-R 5	High	
NF-R 6	High	Testing of the prototype has been demonstrated this.
NF-R 7	High	Testing of the prototype has been demonstrated this.
NF-R 8	High	Testing of the prototype has been completed, but formal testing would be required.
NF-R 9	High	Testing of the prototype has been completed, but formal testing would be required.
NF-R 10	Moderate	Prototype has some elements of modularity and model-view-controller, but some aspects do not fully comply.
NF-R 11	High	Prototype already implements this.
NF-R 12	Low	Prototype has minimum number of strings in English. But requirement exhibits ambiguity on the required languages. An extremely sophisticated alternative would use Google translator services, but would require on-line connectivity.
NF-R 13	High	Prototype performs well already on this, but formal usability testing would be required.
NF-R 14	High	Lifted class diagram of the prototype shows low coupling (few associations between classes). However, formal analysis with software metrics is required.
NF-R 15	Moderate	Nothing implemented in this regard.

Therefore, we conclude that the feasibility of the projects is high.

5 Prototype

The link to the video demonstrating the prototype is

echo360.org.au/media/a61a94f3-b64b-4509-bf58-07dadf264876/public

6 Conceptual Design

6.1 Automatic generation of documentation

A very powerful tool for generating documentation automatically from source code is `doxygen` [1]. However, since we constructed our prototype with Netbeans IDE 8.1[2] (Build 201510222201), we can directly use the JavaDoc tool to generate documentation. In the released version of the code use a browser to open the index file

`MineSweeperPrototype/dist/javadoc/index.html`

A sample of a page of the generated documentation appears in Figure 3.

Classes and objects

Using NetBeans IDE 8.1, and its facility for `easyUML` the class diagram in Figure 4 was automatically obtained from the code.

The classes used for the prototype are briefly described below in alphabetical order as listed by the automatically generated documentation.

Class FloodUncoveringZero

The `FloodUncoveringZero` class (refer to Figure 5) carries the job of computing a region of cells with value zero and its boundary. It is a variation of Dijkstra's algorithm (or breadth first search) using a queue to explore the adjacent node. This class has no objects and its constructor implements the flooding algorithm. Treats the environment as a graph, and should be generic for different types of environments. It requires a model to 1) obtain adjacent nodes to a node in the environment. 2) obtain the value of the node in the environment. The second parameter (startID) is the starting point to flood the region of zero values.

Class GlobalModel

The `GlobalModel` class (refer to Figure 6) works as the current model of the Minesweeper game.

Class GridView

The `GridView` class (refer to Figure 7) has the responsibility of drawing the environment of a square grid.

Class Images

The `Images` class (refer to Figure 8) loads images (digits) as scaled icons as well as smiling/sad faces.

Class MineSweeperPrototype

The `MineSweeperPrototype` class (refer to Figure 9) creates widgets for the GUI and acts as a pseudo-controller.

Class MouseListenerForGridView

The `MouseListenerForGridView` class (refer to Figure 10) handles the mouse clicks on the game arena and issues a corresponding call-back to the view.

[1] www.doxygen.nl.

[2] netbeans.org.

Class SquareGrid

The `SquareGrid` class (refer to Figure 11) to draw squares, covered an uncovered. Thus with numbers or icons for flags or mines.

6.1.1 The `enum` GlobalModel.DifficultyLevels

The enum that defines the difficulty levels and offers type conversion formats to obtain the enum given the integer ordinal and the inverse. We also use the names in the enum to generate the GUI-Item Menu entries and to iterate to populate a menu.

6.1.2 The `enum` GlobalModel.Environments

The enum that defines the environments and offers type conversion formats to obtain the enum given the integer ordinal and the inverse.

6.1.3 Reflection of class diagram and anticipated design for next phases

The next milestones we would require classes for drawing hexagons and for setting up a graph. Therefore, we anticipate a class analogous to `SquareGrid` for the hexagons (probably named `HexagonGrid`. Similarly, a class `GraphLayout` for the drawing of graphs in the play area. These three classes will have elements in common that will be generalised to a super class `GenericGrid`. Thus, the anticipated class diagram is per Figure 12.

Moreover, there is an overlap of the responsibilities of the class `MineSweeperPrototype`. These class has responsibilities of the view, in that in defines and call the constructors of buttons and menu items (these are view responsibilities) While also relays messages and acts as a controller relaying signals to the model (for example, with the end of the game). Constraint C 4 may be considered considered a functional requirement from the perspective of offering a multi-player functionality. For a client-server architecture that enables multi-player options , clearer separation of view and controller would be required. Most likely, we need one package for the server where the model and the controller would sit, while another package would be required for the client side and the view. Adapter classes would be required to relay the event of the views to the server over the socket connection.

A design of this next phase appears in the package diagram of Figure 13. This will decouple further the view from the controller and the view. This will provide several advantages. It will be possible to test the server side independently of the view. This can enable a headless (no GUI) execution of the game and more detailed testing (for example Test-Driven Development [TDD] [1]). It will also ensure we can set a protocol between the socket adapter on the client side to the server side so that there would be even more security in terms of the effects of the user interaction on the client side. That is, it would be extremely unlikely that interaction on the client side would cause harm. However, when running on client/server mode there could be networking issues as the server would be opening a port in the host where it is running. Also, we may need to use a secure socket layer for the messages if we want to ensure that game sessions is not tampered and destroyed by malicious third parties.

As listed in the risks section (Risk 17), the requirement of a multi-player game from different clients on different hosts opens a series of security issues, we will assume the requirement of multi-player takes precedence over this and trust the set-up of networking privilege will be suitable for the application.

As we mentioned earlier, this re-design has several implications. We obtain generalisations for a several classes which in the prototype are specific for the square grid version.

The separation into a client and server package would increase the modularity of the stand alone version. That is, there is less coupling and better fit to the MVC-pattern even if the player doe s not use a client-server multi-player version, but chooses single player. Figure 14 shows the detail of the server package as part of the planned separation into a client-side package and a server-side package.

Analysis of cohesion and coupling

We already mentioned that there is some lack of cohesion regarding the responsibilities of those classes that make the module for the view, and those that are classes of the controller. This issue is being addressed in the previous section by the client-server architecture while meeting the Constraint C 4. Moreover, the previous analysis for each class show that we can mention the responsibility of focus of each class.

Regarding coupling, we can see that the current class diagram (Figure 4) is planar. There are very few links between classes. Some of these associations are for the main components of the M-V-C software architecture

pattern. It is not unusual that the view need a reference to the model, and that the controller needs reference to both. All other associations are localised and within packages. This shows coupling is low.

Use cases

Reflection on the use case diagram and the detailed use case, plus the demo of the prototype we can see that the application follows closely the documented early version of the popular Minesweeper. This suggests that the application is provided an engaging gaming experience that suggest player (users) obtain value for its use.

Reflection on GUI

Reflection of the user interface suggests that there are many improvements. The drawing of cells is extremely basic, so are the icons for mines and flags. However, as drawn, these icons easily scale (also demonstrated in the video how easy is to modify the code to create different levels and scales of the playing arena).

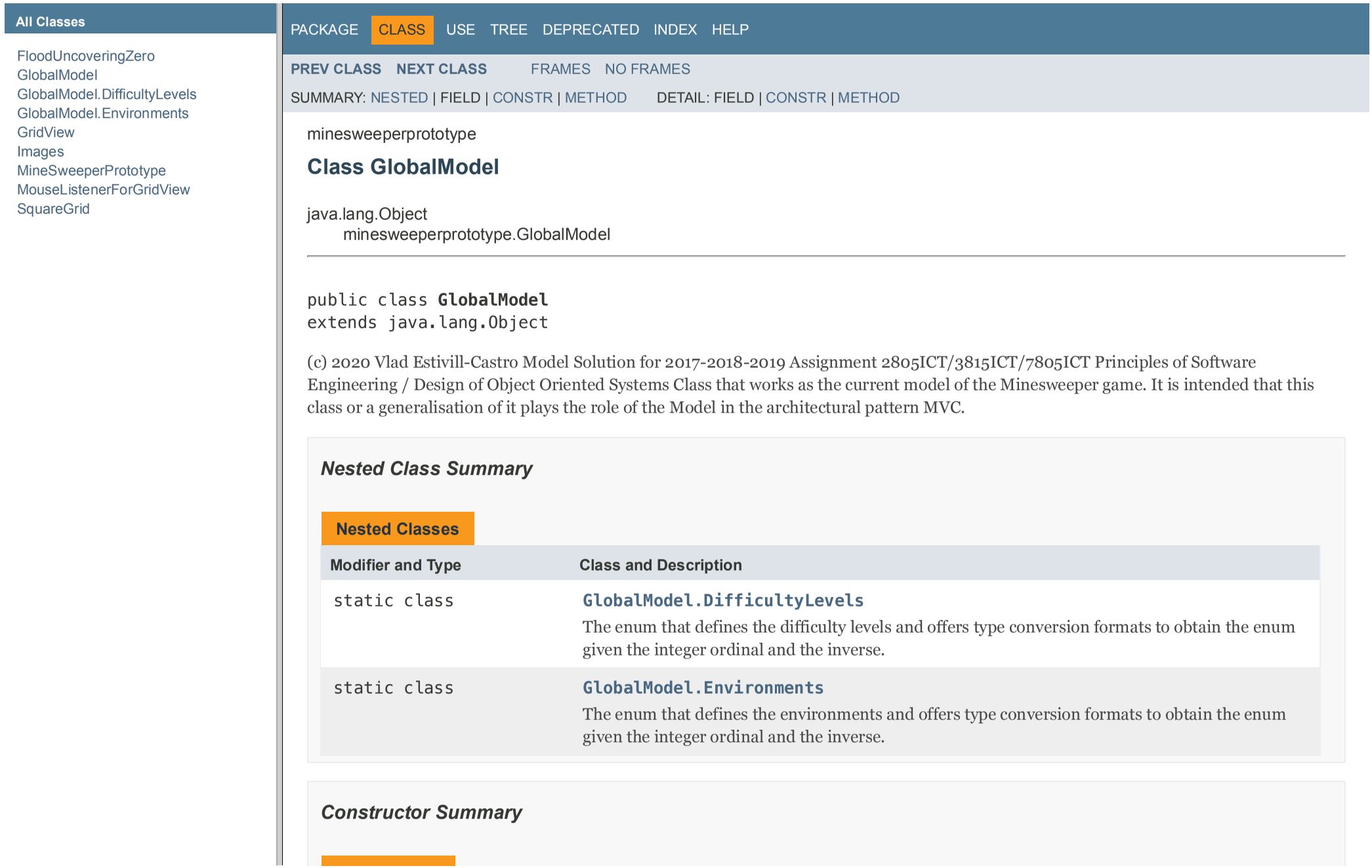
The size of the application is currently fixed. The application is not re-sizable. Maybe this could be another aspect to enhance the GUI.

Naturally, the configuration by the user of the environment and the GUI would enable an even more rewarding engagement with the application.

References

[1] K. Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[2] Ivan Marsic. Software engineering. Rutgers University, September 10th 2012.



All Classes

FloodUncoveringZero
GlobalModel
GlobalModel.DifficultyLevels
GlobalModel.Environments
GridView
Images
MineSweeperPrototype
MouseListenerForGridView
SquareGrid

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

minesweeperprototype

Class GlobalModel

java.lang.Object
minesweeperprototype.GlobalModel

public class **GlobalModel**
extends java.lang.Object

(c) 2020 Vlad Estivill-Castro Model Solution for 2017-2018-2019 Assignment 2805ICT/3815ICT/7805ICT Principles of Software Engineering / Design of Object Oriented Systems Class that works as the current model of the Minesweeper game. It is intended that this class or a generalisation of it plays the role of the Model in the architectural pattern MVC.

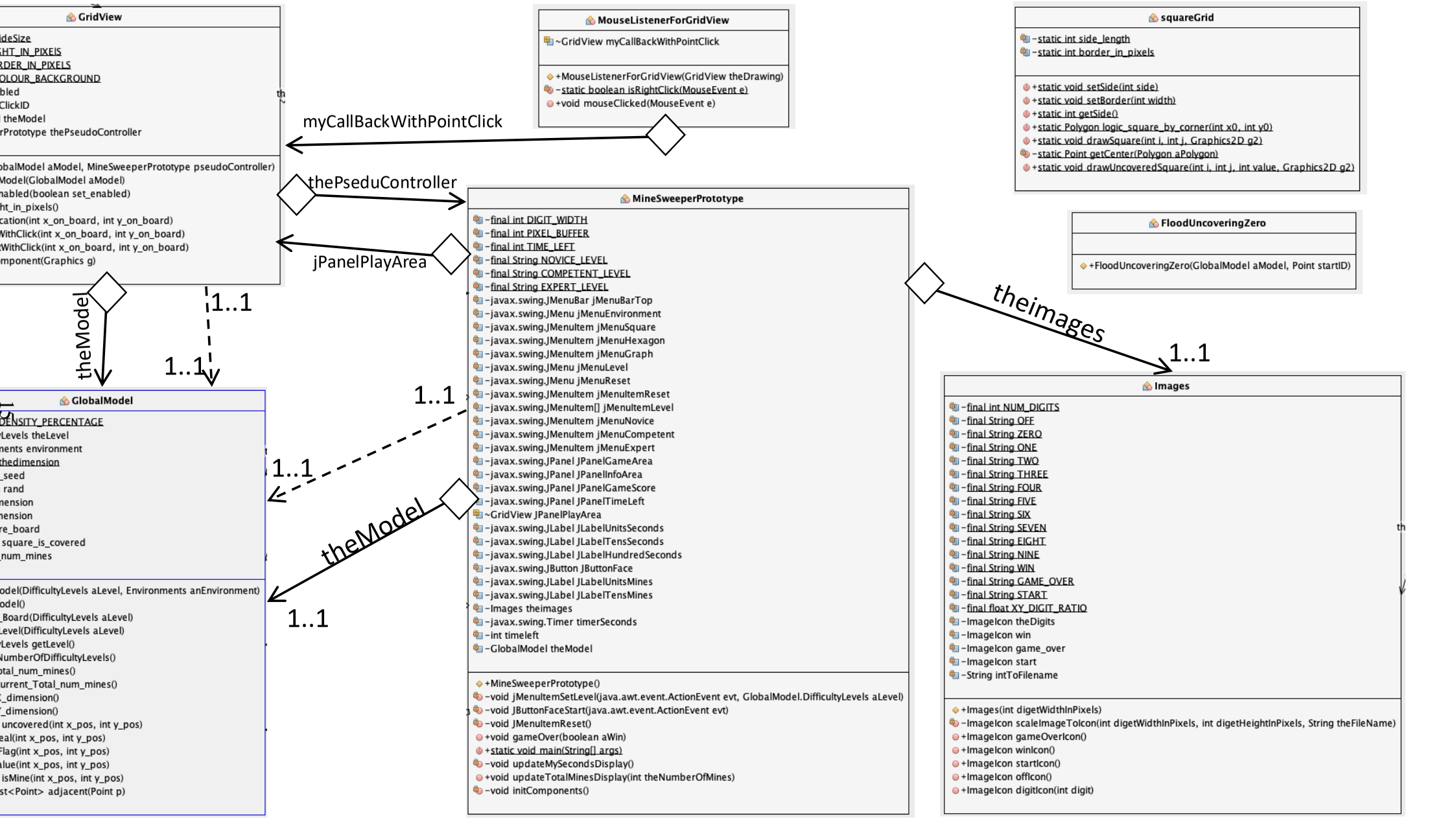
Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
static class	**GlobalModel.DifficultyLevels** The enum that defines the difficulty levels and offers type conversion formats to obtain the enum given the integer ordinal and the inverse.
static class	**GlobalModel.Environments** The enum that defines the environments and offers type conversion formats to obtain the enum given the integer ordinal and the inverse.

Constructor Summary

Figure 3: Sample page of the generated documentation.


GridView
ideSize
GHT_IN_PIXELS
RDER_IN_PIXELS
OLOUR_BACKGROUND
bled
ClickID
theModel
rPrototype thePseudoController
obalModel aModel, MineSweeperPrototype pseudoController)
Model(GlobalModel aModel)
nabled(boolean set_enabled)
ht_in_pixels()
cation(int x_on_board, int y_on_board)
WithClick(int x_on_board, int y_on_board)
tWithClick(int x_on_board, int y_on_board)
mponent(Graphics g)
MouseListenerForGridView
~GridView myCallBackWithPointClick
+MouseListenerForGridView(GridView theDrawing)
-static boolean isRightClick(MouseEvent e)
+void mouseClicked(MouseEvent e)
squareGrid
-static int side_length
-static int border_in_pixels
+static void setSide(int side)
+static void setBorder(int width)
+static int getSide()
+static Polygon logic_square_by_corner(int x0, int y0)
+static void drawSquare(int i, int j, Graphics2D g2)
-static Point getCenter(Polygon aPolygon)
+static void drawUncoveredSquare(int i, int j, int value, Graphics2D g2)
myCallBackWithPointClick
thePseduController
jPanelPlayArea
MineSweeperPrototype
-final int DIGIT_WIDTH
-final int PIXEL_BUFFER
-final int TIME_LEFT
-final String NOVICE_LEVEL
-final String COMPETENT_LEVEL
-final String EXPERT_LEVEL
-javax.swing.JMenuBar jMenuBarTop
-javax.swing.JMenu jMenuEnvironment
-javax.swing.JMenuItem jMenuSquare
-javax.swing.JMenuItem jMenuHexagon
-javax.swing.JMenuItem jMenuGraph
-javax.swing.JMenu jMenuLevel
-javax.swing.JMenu jMenuReset
-javax.swing.JMenuItem jMenuItemReset
-javax.swing.JMenuItem[] jMenuItemLevel
-javax.swing.JMenuItem jMenuNovice
-javax.swing.JMenuItem jMenuCompetent
-javax.swing.JMenuItem jMenuExpert
-javax.swing.JPanel JPanelGameArea
-javax.swing.JPanel JPanelInfoArea
-javax.swing.JPanel JPanelGameScore
-javax.swing.JPanel JPanelTimeLeft
~GridView JPanelPlayArea
-javax.swing.JLabel JLabelUnitsSeconds
-javax.swing.JLabel JLabelTensSeconds
-javax.swing.JLabel JLabelHundredSeconds
-javax.swing.JButton JButtonFace
-javax.swing.JLabel JLabelUnitsMines
-javax.swing.JLabel JLabelTensMines
-Images theimages
-javax.swing.Timer timerSeconds
-int timeleft
-GlobalModel theModel
+MineSweeperPrototype()
-void jMenuItemSetLevel(java.awt.event.ActionEvent evt, GlobalModel.DifficultyLevels aLevel)
-void JButtonFaceStart(java.awt.event.ActionEvent evt)
-void JMenuItemReset()
+void gameOver(boolean aWin)
+static void main(String[] args)
-void updateMySecondsDisplay()
+void updateTotalMinesDisplay(int theNumberOfMines)
-void initComponents()
FloodUncoveringZero
+FloodUncoveringZero(GlobalModel aModel, Point startID)
theimages
1..1
Images
-final int NUM_DIGITS
-final String OFF
-final String ZERO
-final String ONE
-final String TWO
-final String THREE
-final String FOUR
-final String FIVE
-final String SIX
-final String SEVEN
-final String EIGHT
-final String NINE
-final String WIN
-final String GAME_OVER
-final String START
-final float XY_DIGIT_RATIO
-ImageIcon theDigits
-ImageIcon win
-ImageIcon game_over
-ImageIcon start
-String intToFilename
+Images(int digetWidthInPixels)
-ImageIcon scaleImageToIcon(int digetWidthInPixels, int digetHeightInPixels, String theFileName)
+ImageIcon gameOverIcon()
+ImageIcon winIcon()
+ImageIcon startIcon()
+ImageIcon offIcon()
+ImageIcon digitIcon(int digit)
theModel
1..1
1..1
1..1
1..1
1..1
GlobalModel
DENSITY_PERCENTAGE
Levels theLevel
nents environment
thedimension
_seed
rand
nension
nension
re_board
square_is_covered
num_mines
odel(DifficultyLevels aLevel, Environments anEnvironment)
odel()
_Board(DifficultyLevels aLevel)
Level(DifficultyLevels aLevel)
Levels getLevel()
NumberOfDifficultyLevels()
otal_num_mines()
urrent_Total_num_mines()
_dimension()
_dimension()
uncovered(int x_pos, int y_pos)
eal(int x_pos, int y_pos)
Flag(int x_pos, int y_pos)
alue(int x_pos, int y_pos)
isMine(int x_pos, int y_pos)
st<Point> adjacent(Point p)

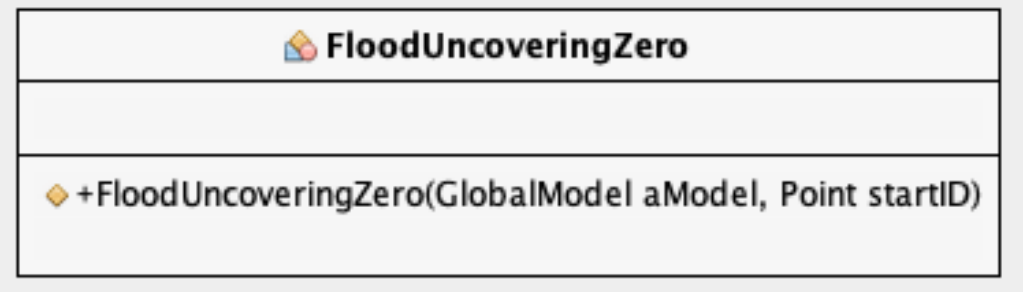


Figure 5: Class for flooding algorithm.

Figure 6: Class to act as the model.

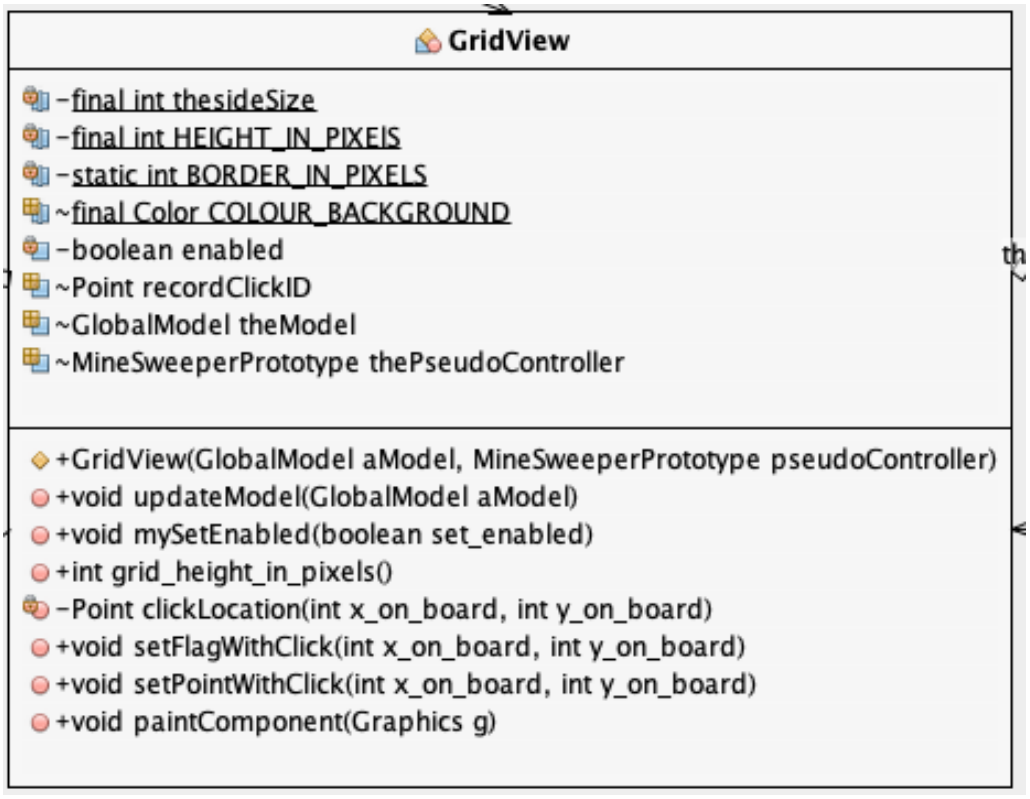


Figure 7: Class to act as the view.

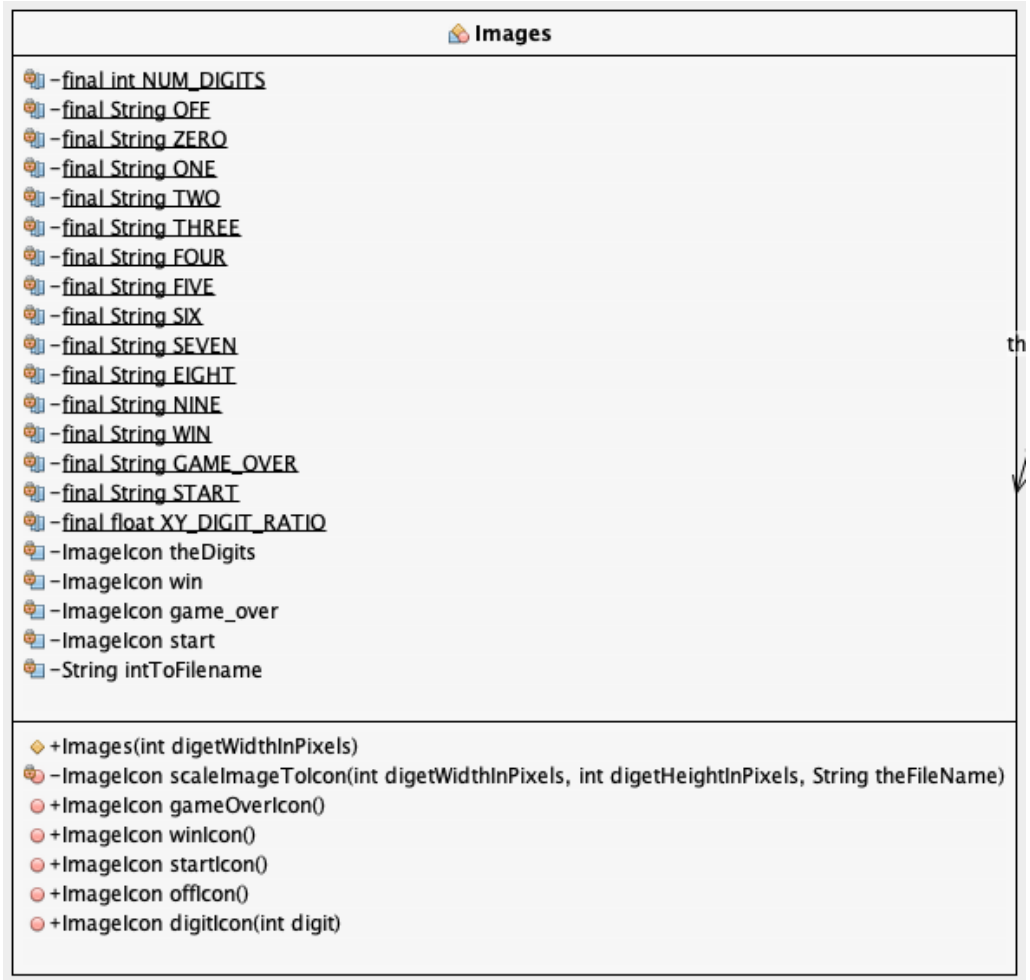


Figure 8: Class for loading icons.

Figure 9: Class for generating widgets and acts as pseudo-controller.

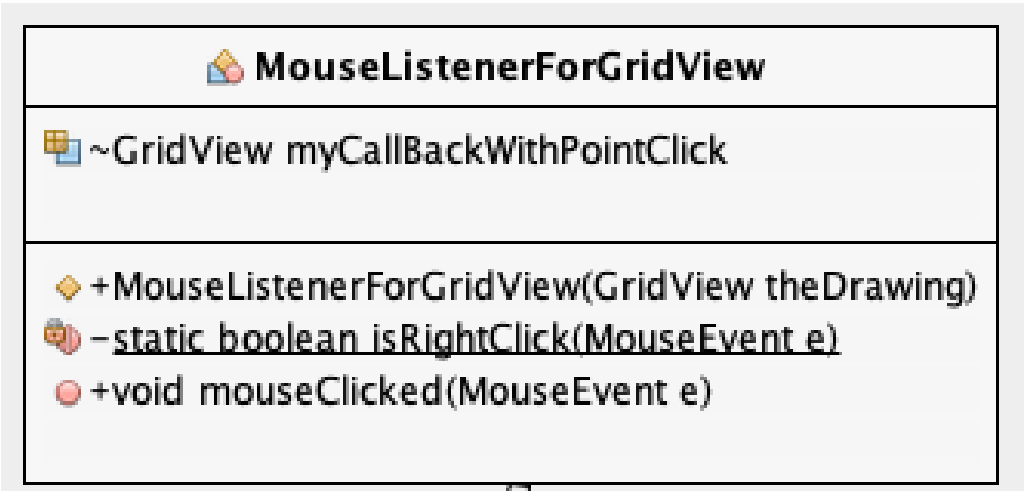


Figure 10: Class for capturing mouse click event on game arena.

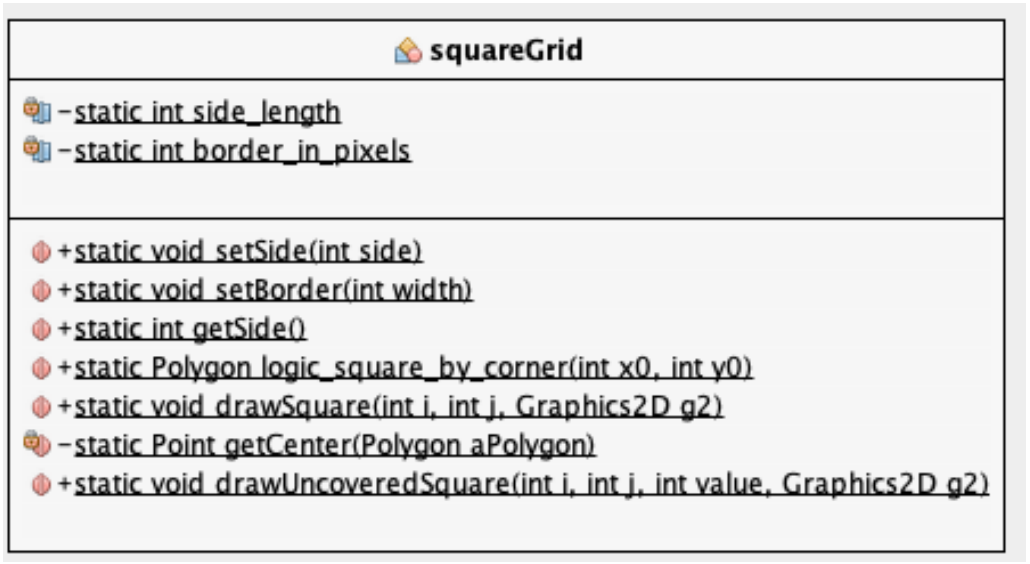


Figure 11: Class to draw squares, covered an uncovered.

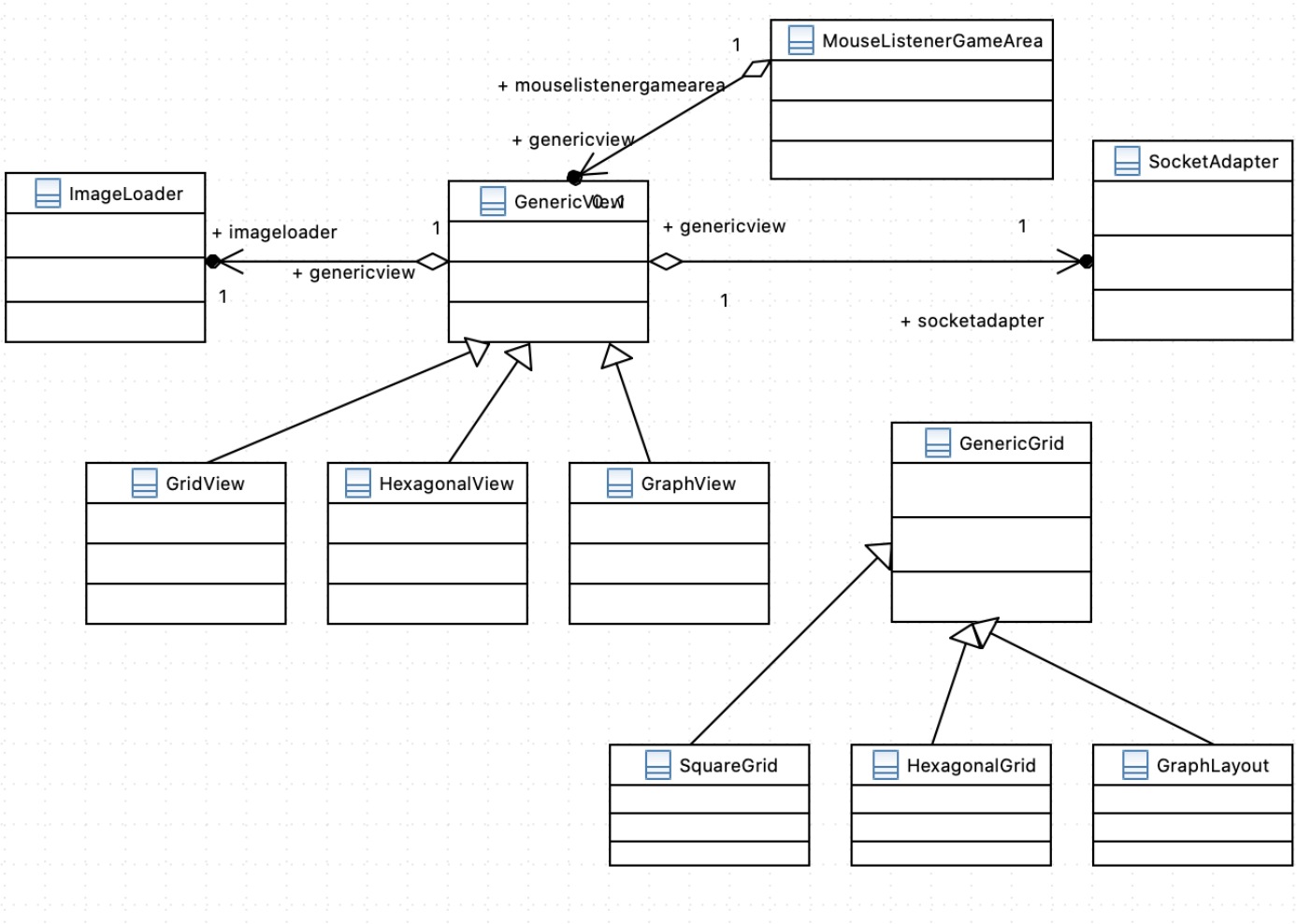


Figure 12: Class diagram of the client-side package for further versions.

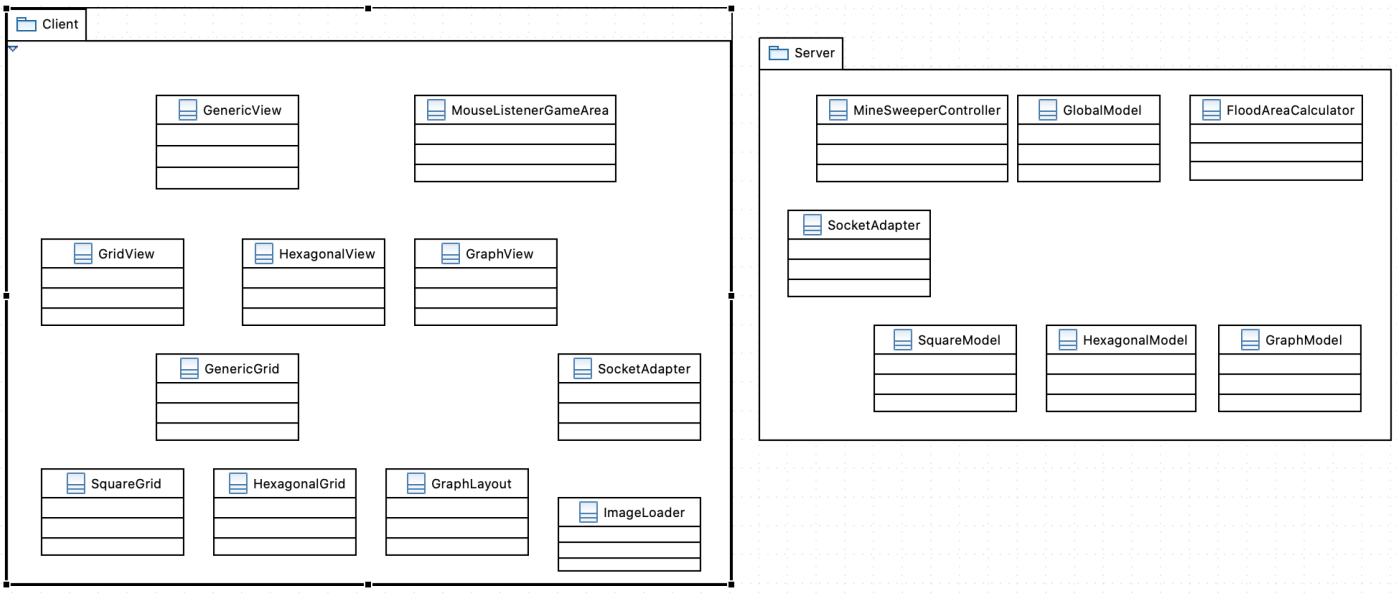


Figure 13: Package diagram for further versions.

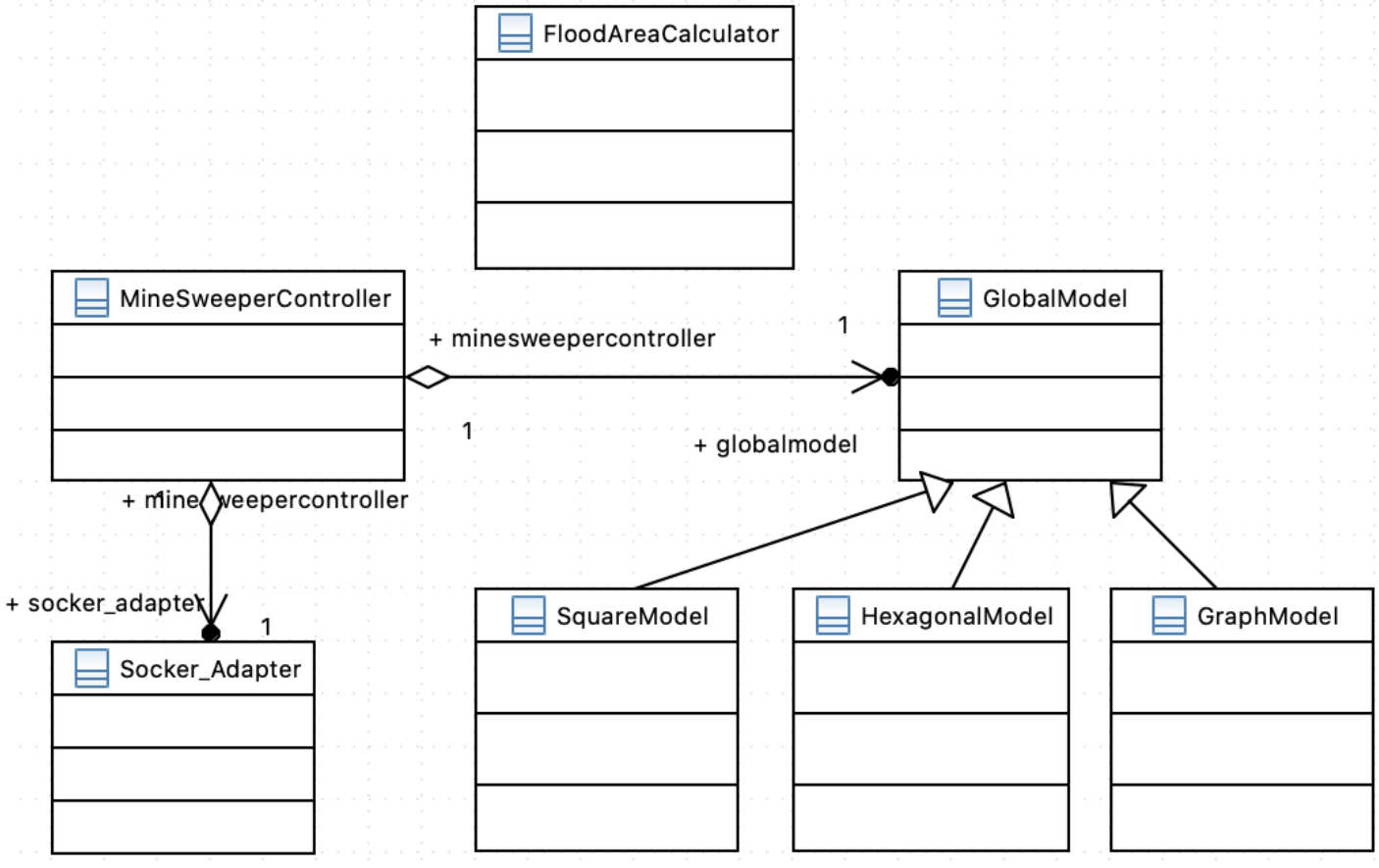


Figure 14: Class diagram of the server-side package for further versions.