System design document for Maze (SDD)

1 Introduction 2
1.1 Design goals.......... 2
1.2 Definitions, acronyms and abbreviations 2

2 System design 3
2.1 Overview 3

2.2 Software decomposition 3
2.2.1 General 3
2.2.2 Decomposition into subsystems 3
2.2.3 Dependency analysis.......... 4
2.3 Concurrency issues 4
2.4 Persistent data management.......... 4
2.5 Access control and security 4
2.6 Boundary conditions 4

3 References 4

4 Appendix 5

Version: Maze 0.1

Date:

Author:
Matilda Andersson
Karin Wibergh
Johan Ärlebrandt
Jonathan Johansson

1 Introduction

1.1 Design goals

We aim to have a self-contained model which is fully testable. Further, it should be possible to use the same model regardless of what the GUI is like.

1.2 Definitions, acronyms and abbreviations

- **GUI**, graphical user interface.
- **Java**, platform independent programming language.
- **Model–view–controller** (**MVC**) is a software architectural pattern for implementing user interfaces.

2 System design

2.1 Overview

The design is easy to use, it is easy to come to the game view and play a level and from there play next level. On every view the player can go back to the mainview. Every player has the possibility to see the the score for every slot and a high score list over all players that have play the game.

2.2 Software decomposition

2.2.1 General

The application is decomposed into five packages.

Main
This package only contains the main class which starts the application.

Model
This package's functionality is exposed through the IGame interface.

View
This package contains the application's GUI. The package has a class for every view

Controller
This package contains the controller part of the MVC. As in the view package there are a controller class for every view.

Util
Contains IO-classes and constants. The IO - classes take care of the saving and the reading of the map for every level.

To see class diagrams for every package see appendix.

2.2.2 Decomposition into subsystems
The only thing remotely resembling a subsystem are the IO-files in Util, but these are not a unified subsystem.

2.2.3 Dependency analysis
See Appendix.

2.3 Concurrency issues
NA - single-threaded application.

2.4 Persistent data management
Information about the save slots and highscores is saved in a file called Game.ser.

2.5 Access control and security
NA

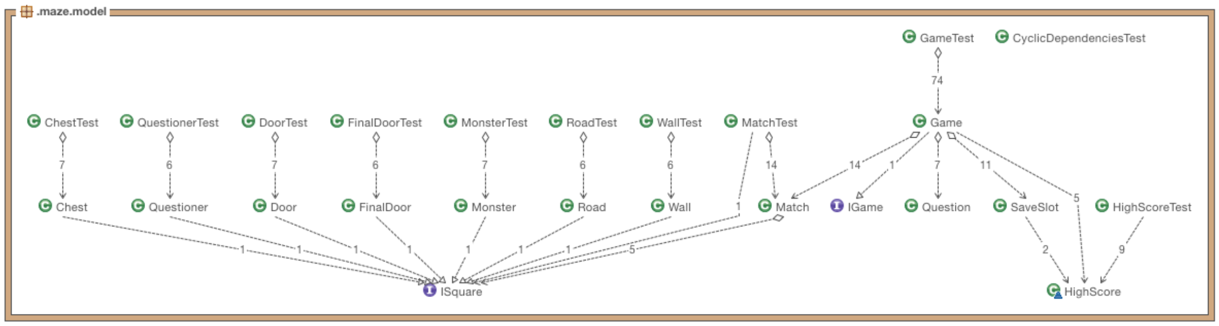
2.6 Boundary conditions
NA. Application launched and exited as normal desktop application

3 References
MVC: Wikipedia

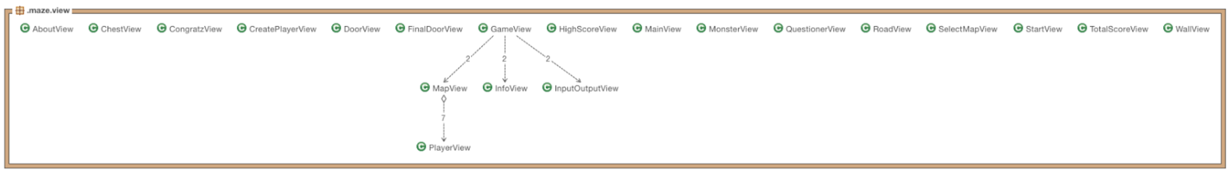
APPENDIX

Diagram for the dependency:

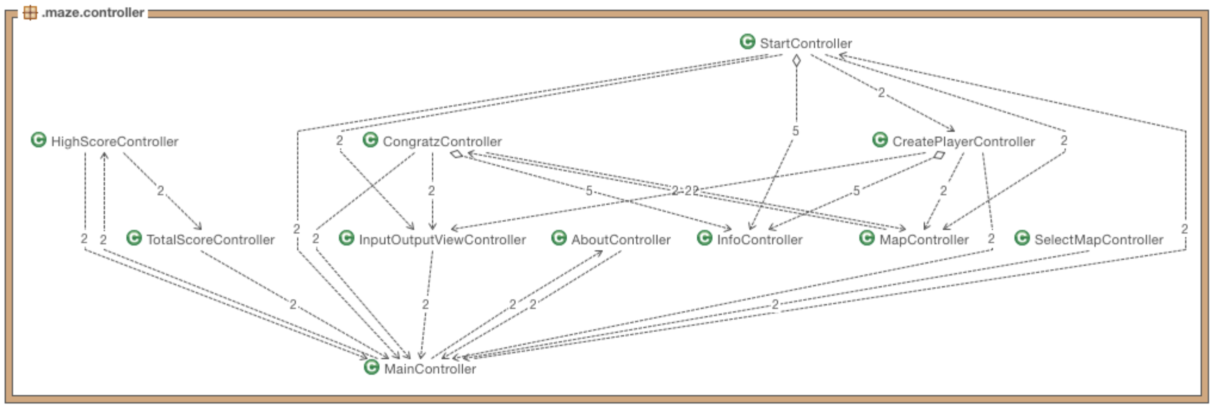
Model:



View:



Controller:

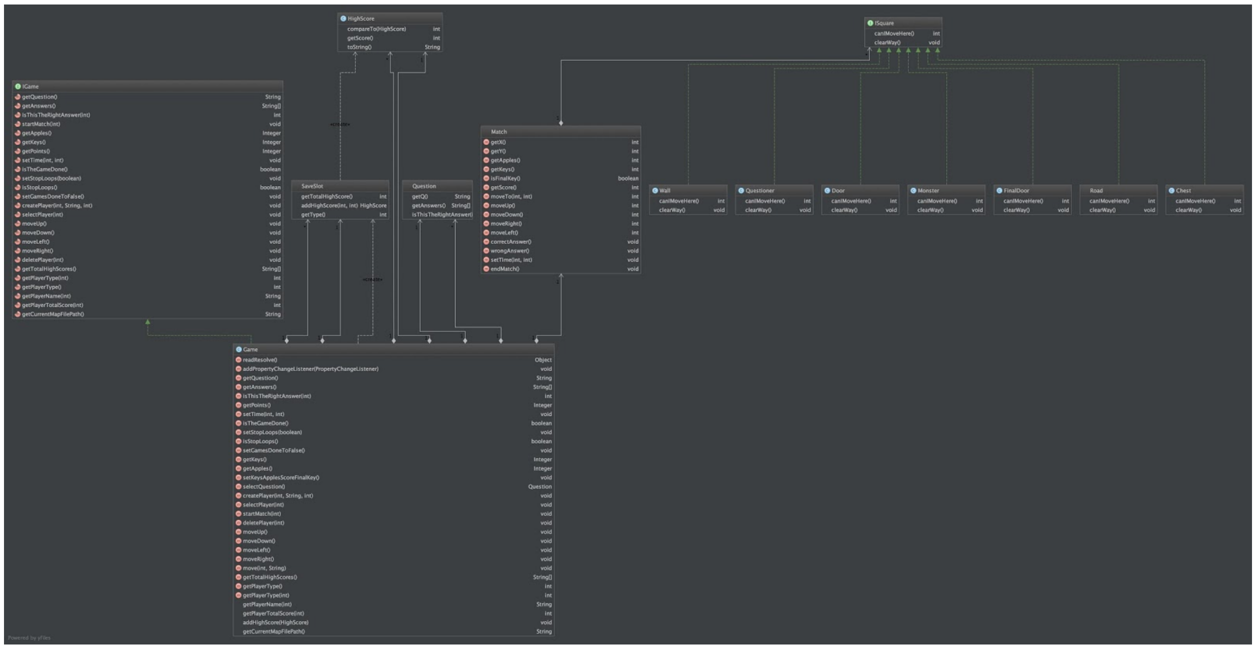


Util:

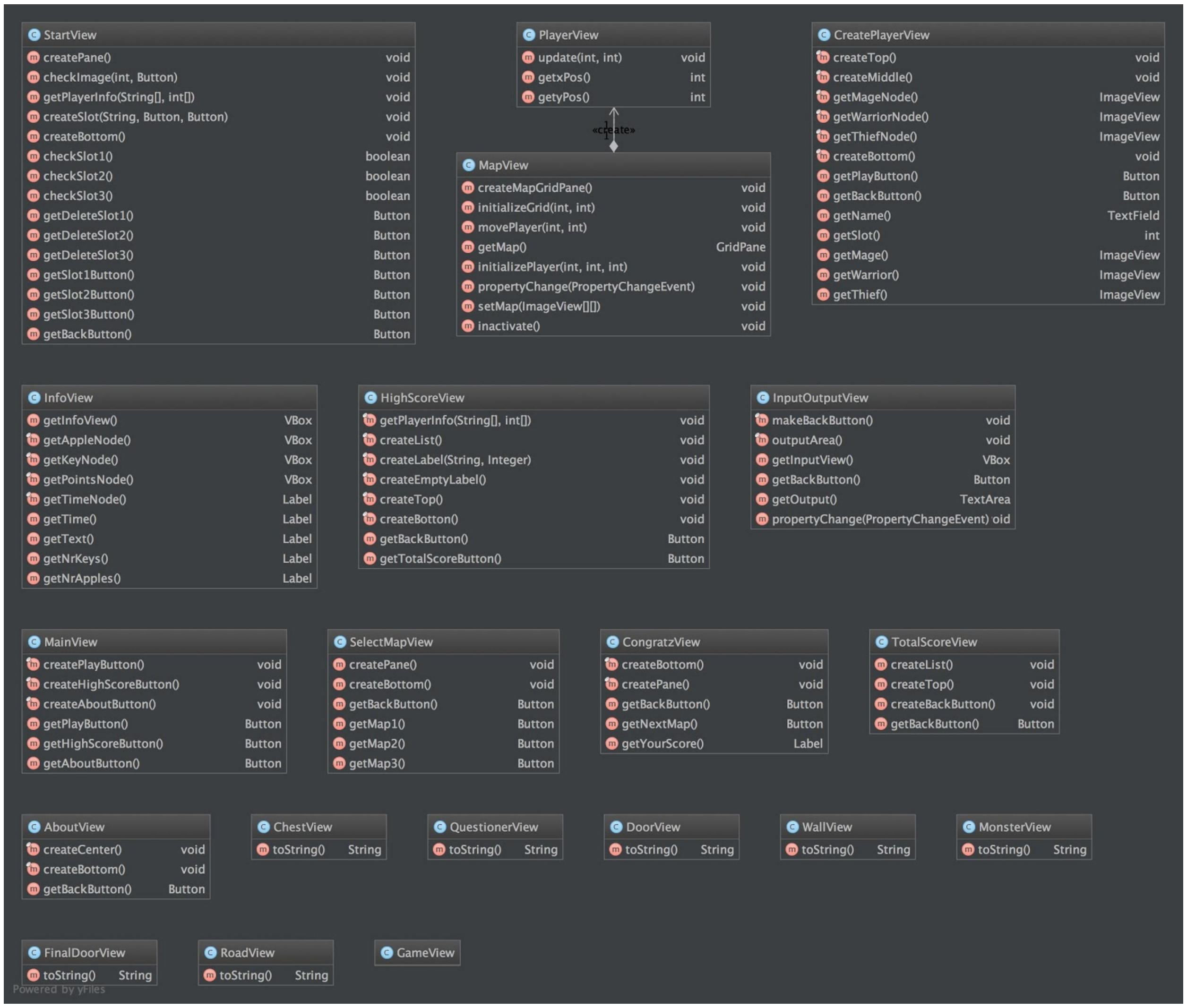
.maze.util
Constants
ResourceReader
SavedInformationHandler

Uml diagram for every package:

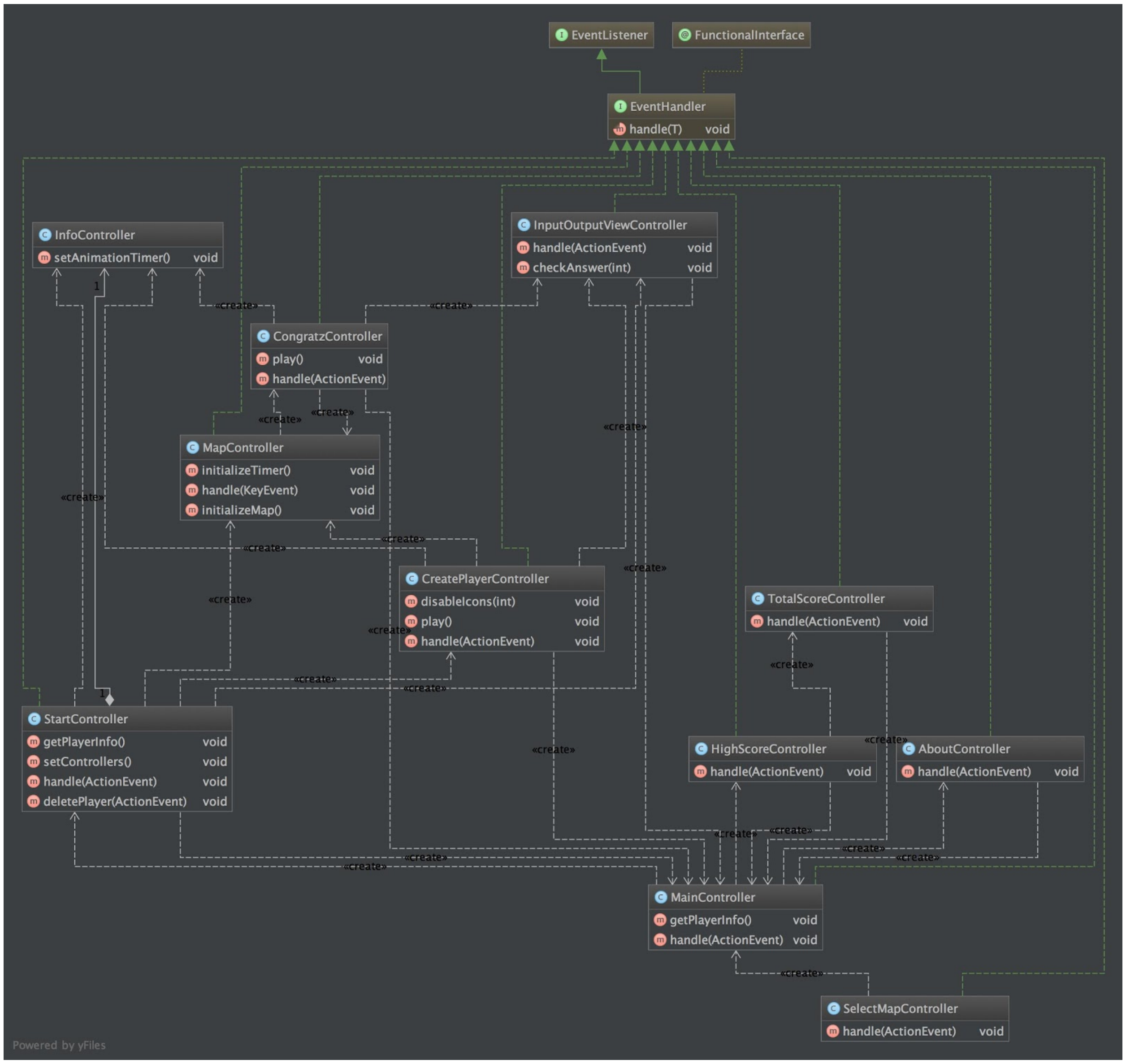
Model:



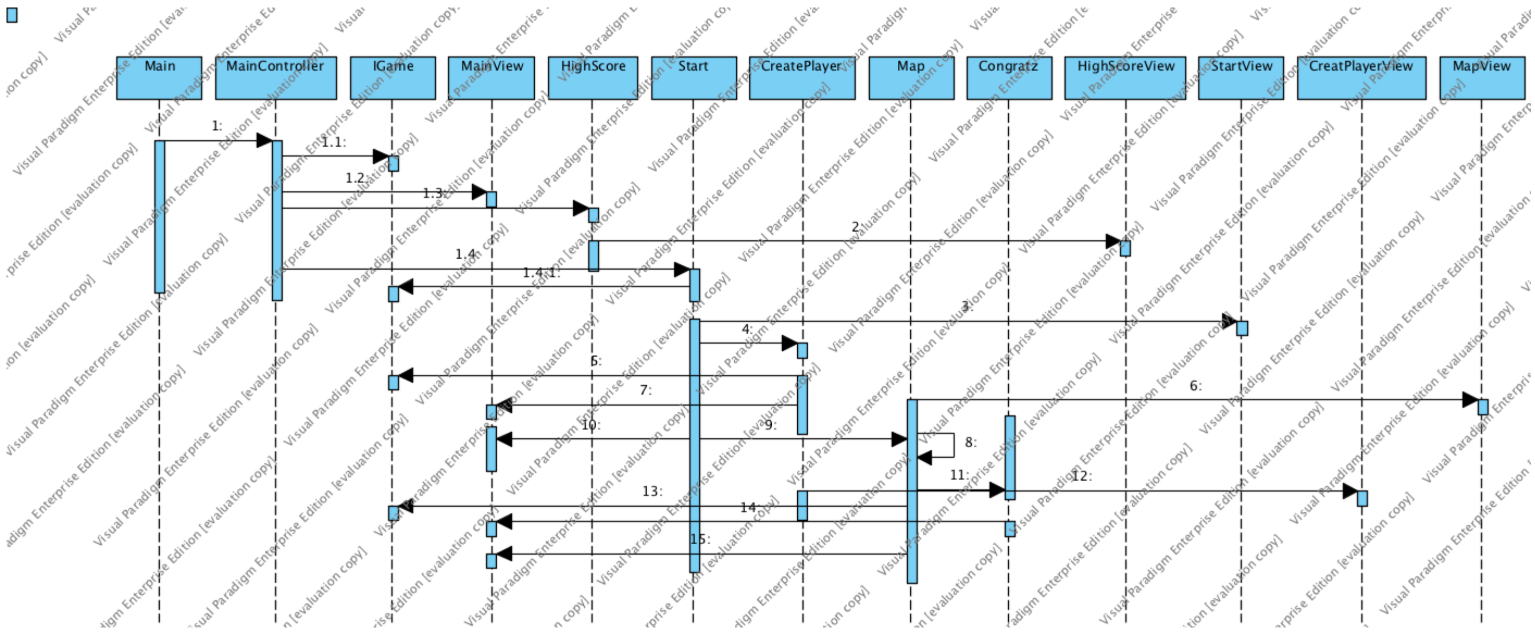
View:


StartView
createPane() void
checkImage(int, Button) void
getPlayerInfo(String[], int[]) void
createSlot(String, Button, Button) void
createBottom() void
checkSlot1() boolean
checkSlot2() boolean
checkSlot3() boolean
getDeleteSlot1() Button
getDeleteSlot2() Button
getDeleteSlot3() Button
getSlot1Button() Button
getSlot2Button() Button
getSlot3Button() Button
getBackButton() Button
PlayerView
update(int, int) void
getxPos() int
getyPos() int
«create»
1
1
MapView
createMapGridPane() void
initializeGrid(int, int) void
movePlayer(int, int) void
getMap() GridPane
initializePlayer(int, int, int) void
propertyChange(PropertyChangeEvent) void
setMap(ImageView[][]) void
inactivate() void
CreatePlayerView
createTop() void
createMiddle() void
getMageNode() ImageView
getWarriorNode() ImageView
getThiefNode() ImageView
createBottom() void
getPlayButton() Button
getBackButton() Button
getName() TextField
getSlot() int
getMage() ImageView
getWarrior() ImageView
getThief() ImageView
InfoView
getInfoView() VBox
getAppleNode() VBox
getKeyNode() VBox
getPointsNode() VBox
getTimeNode() Label
getTime() Label
getText() Label
getNrKeys() Label
getNrApples() Label
HighScoreView
getPlayerInfo(String[], int[]) void
createList() void
createLabel(String, Integer) void
createEmptyLabel() void
createTop() void
createBotton() void
getBackButton() Button
getTotalScoreButton() Button
InputOutputView
makeBackButton() void
outputArea() void
getInputView() VBox
getBackButton() Button
getOutput() TextArea
propertyChange(PropertyChangeEvent) oid
MainView
createPlayButton() void
createHighScoreButton() void
createAboutButton() void
getPlayButton() Button
getHighScoreButton() Button
getAboutButton() Button
SelectMapView
createPane() void
createBottom() void
getBackButton() Button
getMap1() Button
getMap2() Button
getMap3() Button
CongratzView
createBottom() void
createPane() void
getBackButton() Button
getNextMap() Button
getYourScore() Label
TotalScoreView
createList() void
createTop() void
createBackButton() void
getBackButton() Button
AboutView
createCenter() void
createBottom() void
getBackButton() Button
ChestView
toString() String
QuestionerView
toString() String
DoorView
toString() String
WallView
toString() String
MonsterView
toString() String
FinalDoorView
toString() String
RoadView
toString() String
GameView
Powered by yFiles

Controller:


EventListener
FunctionalInterface
EventHandler
handle(T) void
InfoController
setAnimationTimer() void
InputOutputViewController
handle(ActionEvent) void
checkAnswer(int) void
CongratzController
play() void
handle(ActionEvent)
MapController
initializeTimer() void
handle(KeyEvent) void
initializeMap() void
CreatePlayerController
disableIcons(int) void
play() void
handle(ActionEvent) void
TotalScoreController
handle(ActionEvent) void
StartController
getPlayerInfo() void
setControllers() void
handle(ActionEvent) void
deletePlayer(ActionEvent) void
HighScoreController
handle(ActionEvent) void
AboutController
handle(ActionEvent) void
MainController
getPlayerInfo() void
handle(ActionEvent) void
SelectMapController
handle(ActionEvent) void
«create»
1
Powered by yFiles

Sequence Diagram:



1: Main calls the mainController with the IGame and mainview.
2: From the MainView The user can call either highscore or start.
3: The start calls the startView. If the user never played the create calls otherwise the map calls and the game begins.
4: To get info about the player the IGame interface calls.
5: Every Controller calls a View.
6: When the player finished the first level he/she can play next level.
7: From every view the player has the possibility to go back to the mainview.