# Software Engineering Group Project
# Design Specification AUM group

Author: dkm4, jty, nah37
Config Ref: SE_JC_DS _01
Date: 2018-04-26
Version: 1.5
Status: Released

Department of Computer Science
Aberystwyth University Branch Campus Mauritius
Coastal Road Uniciti
Flic en flac

# Table of Contents

# 1. INTRODUCTION

Clear, consistently followed, document standards are essential in software engineering.

## 1.1 Purpose of this Document

This document describes all the components that make up the JoggleCube Game.

## 1.2 Scope

The reader should be familiar with the User Interface Specification document.

## 1.3 Objectives

The objective of this document is to aid any programmer who wants to understand or maintain or improve the existing code.

# 2. DECOMPOSITION DESCRIPTION

In this section of the document, we will describe the packages that make up the JuggleCube game. The packages have been separated according to their use and role in the application so that the main components dealing with one aspect of the game are in one package. Some of the packages are furthermore divided into sub-packages in order to group the modules accomplishing similar tasks. This is also to allow a clear structure of the code making up the JoggleCube game.

The gameFrames package, as the name suggests, is the package responsible for holding all the modules that make up the interface and also the data that is required for it to function properly. Also, with the help of the other packages, the modules making up the gameFrame package, it will be responsible for *generating* the frames that make up the game and which will be displayed to the user to interact with.

The second main component, the gameIcons package, as its name suggests, is the package that holds the icons referenced in the modules that make up the gameFrames package. This package is mostly to separate the external components that will be loaded apart from the Java code and to hold all of these components in one package for clear structure.

The gameLogic package, the third main component, is the package that holds all the modules that enable the functionality of the game. It interacts with the interface that was made from the gameFrames and gameIcons package. Its main purpose is to provide a structure to build the game on as well as the required functionality to be able to play the game as required. The package is moreover made up of one sub-package, the interfaces package, which holds the interfaces implemented by the modules in the gameLogic package.

## 2.1 Significant Classes

## 2.2 GameMainClass module description

**Background:**

There is quite a bit of functionality, for e.g. enabling the user to check for a word and then adding it to the list of accepted words for that game, clearing a word selection via the assigned button and loading saved games as well as saving played games, that had to work with several components from other modules that a separate module for those was required to keep the project low in coupling. This is also our main module which starts up the game and loads the starting the screen.

**Description:**

| Identification | GameMainClass |
|---|---|
| Purpose | The purpose of this module is to handle all the remaining interactions taking place throughout the game |
| Function | This module was made to hold all the other methods and properties that have to deal with the entire game and not only one grid or other sub-component to maintain a low coupling |

### 2.2.1 Grid Module Description

**Background:**

The Grid module extends the Swing component JPanel as well as makes use of other methods and attributes to create a component. This component is one of the most important components that the user will interact with. It will enable the user to select Tiles to make up a word. It will also responsible for the proper generation and population of the tiles with random letters that make up the grid. However, when a game is loaded, it ensures that the saved tiles are correctly loaded and populated in the grid.

**Description:**

| Identification | Grid |
|---|---|
| Purpose | The aim of this module is to model the grids that are used to make up the game |
| Function | This module has been made to ensure accurate and random generation of random letters at the start of a new game as well as proper population of the grids when a previous game is loaded. It also handles user input via keyboard and mouse and deals with the adjacent tiles that have to be highlighted |

### 2.3 Letter Module Description

**Background:**

A letter is the component that the tile displays on the grid. This has been made so that it is easy to associate the values related to each letter in the population of letters, like the maximum number of occurrences of a letter allowed in a grid and the scrabble score of a letter. It has a toString method that is used when a previously saved game is loaded and only one getter method for the maximum number of a letter allowed in a grid.

**Description:**

| Identification | Letter |
|---|---|
| Purpose | The aim of this module is to model a component that will be represented in each tile and which will have its own properties |
| Function | This module was made to provide a clean and easy way to deal with the component that has to be represented in a tile and since a normal Swing component would not be sufficient, the Letter module has been implemented |

### 2.3.1 LetterPopulation Module Description

**Background:**

This is an enumeration of the letters that make up the letter population. It is a key-value pair where the values stored are the maximum number of occurrences that specific letter can appear in one grid as well as the scrabble value of the letter itself.

**Description:**

| | |
|---|---|
| Identification | LetterPopulation |
| Purpose | The purpose of this module is to provide a collection of all letters that can be used in the game |
| Function | This module was made to provide an easy way to associate the letters with the properties that every letter must have and which must, in turn, be represented on each tile |

### 2.3.2 PositionInGrid Module Description

**Background:**

This module has been made since in order to properly populate the grid at the start of every game, be it loading a previously saved game or a new game, the position of the tiles in each grid have to be taken into consideration and doing so with the modules readily available would have caused a lot of strain on the game overall.

**Description:**

| | |
|---|---|
| Identification | PositionInGrid |
| Purpose | The aim of this module is to model the position of a tile in a grid |
| Function | This module has been made to have an easy way to deal with the positioning of the tiles in the grid |

### 2.3.3 Tile Module Description

**Background:**

The Tile module models a component that extends the swing button component to make it a better fit for the purpose which it is being used for in the JuggleCube game. The added properties include, but are not limited to, defining whether the tile can be selected by the user, whether the tile is currently in the selection made by the player and providing the position of the tile in the grid that it belongs.

**Description:**

| | |
|---|---|
| Identification | Tile |
| Purpose | The aim of this module is to model a component that will extend the properties and functionalities of a standard button |
| Function | This module has been made to provide a suitable component to work with which will make up the grids |

### 2.4 Mapping from requirements to classes

In order to ensure properly fulfilling the requirements needed, a table mapping the functional requirements onto the classes which contribute to those requirements is shown below:

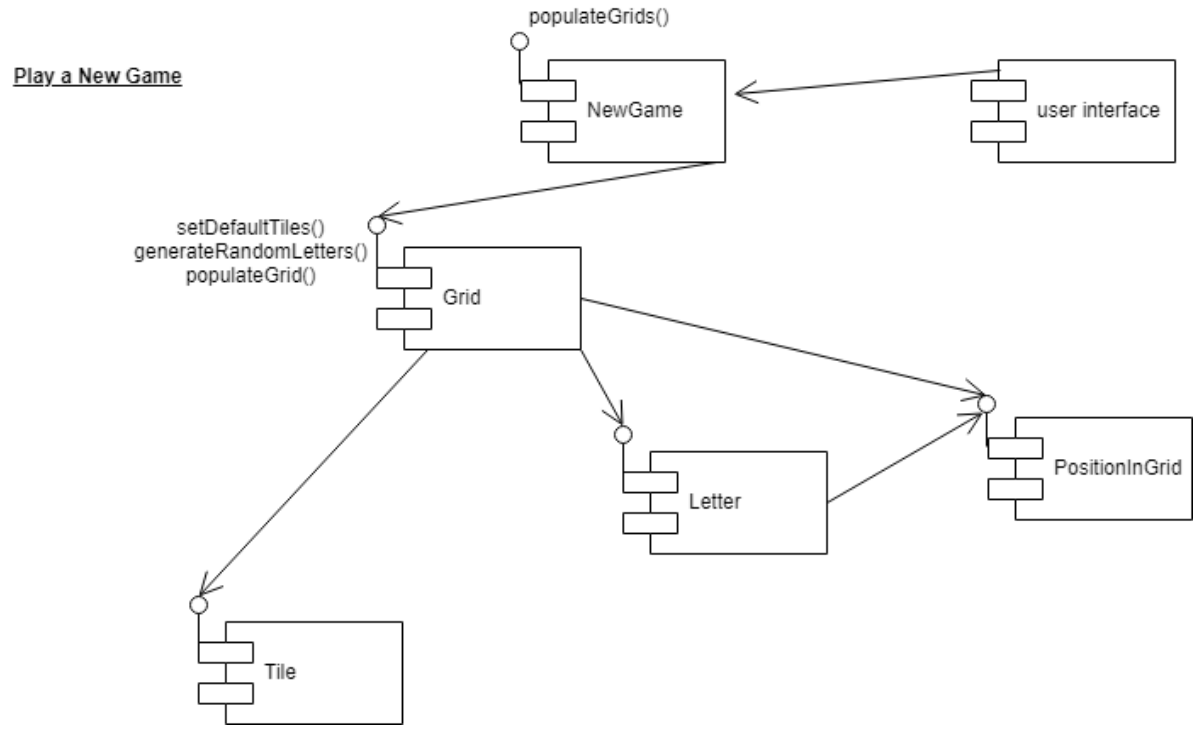
| *Requirement* | Classes providing requirement |
|---|---|
| FR1 | gameMainClass, startMenu |
| FR2 | playGame, Grid, Tile, Letter, LetterPopulation |
| FR3 | loadGame, playGame, Grid, Tile, Letter, LetterPopulation |
| FR4 | playGame, gameMainClass, gameTiming |
| FR5 | selectGame |
| FR6 | gameScoreBoard |
| FR7 | playGame, Grid, Tile |
| FR8 | playGame, gameMainClass |
| FR9 | playGame, gameMainClass |
| FR10 | playGame |
| FR11 | playGame |

# 3. DEPENDENCY DESCRIPTION

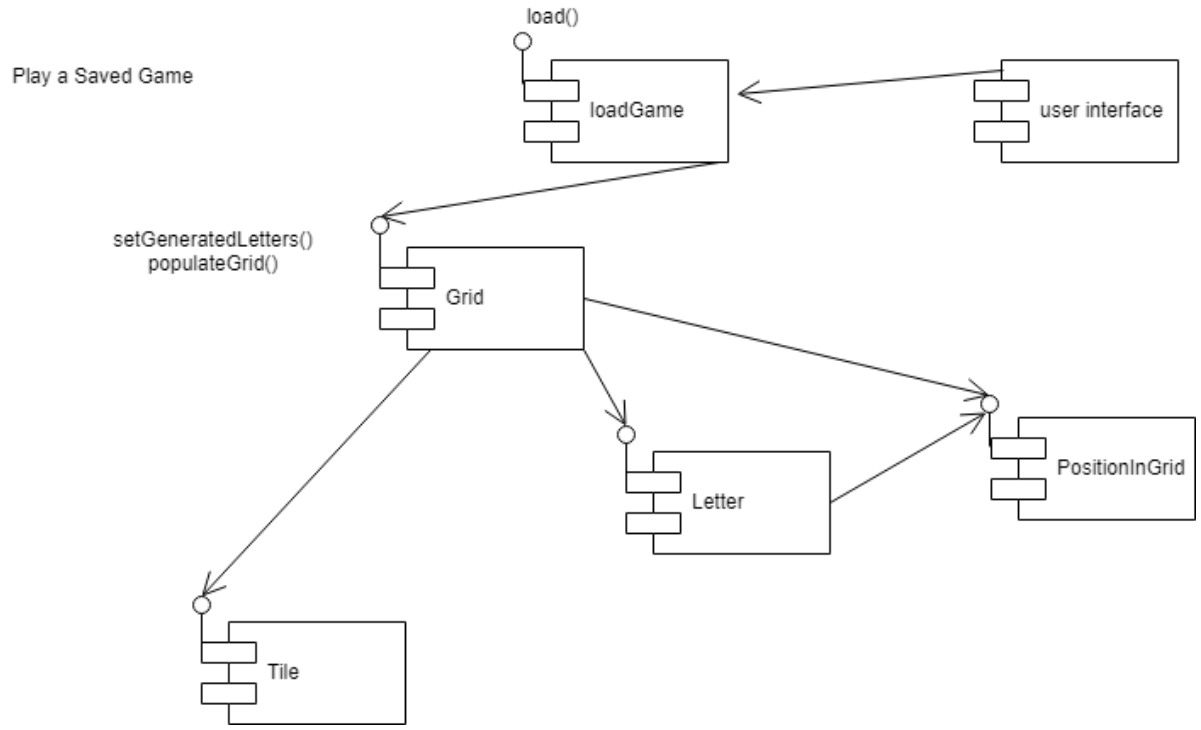
### 3.1 Component Diagrams

Since there are no other programs, and the JoggleCube game was written all in Java, the component diagrams shown below are about describing the relationship between the relevant modules in the project. These also show the inheritance dependencies between the modules.

For playing a new game:



For playing a saved game:




# 4. INTERFACE DESCRIPTION

## 4.1 gameMainClass

```
public class gameMainClass
public static void main(String[] args)
```

gameMainClass is the main class of the program. It is a public class which manage all the back-end function in the JoggleCube . The main class is a public static void type since it there is no object to invoke therefore it has a static method to allow invocation from class and it does not return any value. It accepts arguments of type string in a collection. The main class allow the startMenu to be loaded.

## 4.2 startMenu

```
public class startMenu extends javax.swing.JFrame
```

The startMenu is a public classs which extends the jFrame class to allow the the jFrame design to run in the program.

```
public startMenu()
```

It is a public class which is use to initialize the core components of the jFrame.

```
public void close()
```

It is a public class used to control the termination of the game. Once this function has been clicked on the gameplay will ask the user if he/she wants to end and close the game.

```
private void initComponents()
```

It is a public class with no return value which is use to initialize the core components of the gameplay design.

```
public void actionPerformed(java.awt.event.ActionEvent evt)
```

It is a public class which extends an actionevent evt and returns no values. The actionPerformed is use to monitor the action being performed once pressed or clicked.

```
private void jbnewGameActionPerformed(java.awt.event.ActionEvent evt)
```
It is a public class which returns no values and is use to control the action once the jbutton newGame is pressed.

```
private void jbhelpActionPerformed(java.awt.event.ActionEvent evt)
```
It is a public class which returns no values and is use to control the action once the jbutton help is pressed.

```
private void jbSettingsActionPerformed(java.awt.event.ActionEvent evt)
```
It is a public class which returns no values and is use to control the action once the jbutton settings is pressed.

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt)
```
It is a public class which returns no values and is use to control the action once the jbutton jbutton5 is pressed.

```
public static void main(String args[])
```
The main class is a public static void type since it there is no object to invoke therefore it has a static method to allow invocation from class and it does not return any value. It accepts arguments of type string in a collection. The main class allows the function to be called.

### 4.3 playMenu

```
public class playGame extends javax.swing.JFrame
```
The playGame is a public classs which extends the jFrame class to allow the the jFrame design to run in the program.

```
public playGame()
```
It is a public class which initialize the game components and allows the creation of a new game grid. It create the object for the playMenu.

```
private void initComponents()
```
It is a public class which is use to called the variables and return no values since it is void type.

```
public int getSize()
```
It is a public class which is required to return a integer value for the grid panel to compute.

```
public String getElementAt(int i)
```
It is a public class with a parameter integer i and it returns a string value from the grid for the user to know what element a single grid contains.

```
public void actionPerformed(java.awt.event.ActionEvent evt)
```
It is a public class with parameter evt as an ActionEvent and outputs no value. It performs all the action to be performed once a button is clicked.

```
private void populateGrids()
```
It is a private class to be called only in a specific scope of the code and it is use to generate the game grid with letter for the user to play.

```
private void handleTileAction(java.awt.event.ActionEvent evt)
```
It is a private class with parameter ActionEvent evt to handle the selection of the user once a game tile has been clicked the other grid will also light up the respective game tile.

```
public void run()
```
It is a public class with no values to be output and it is use to link the gameplay button to the main menu of the game.

### 4.4 gameScoreBoard

```
public class gameScoreBoard extends javax.swing.JFrame
```
It is a public class which extends the jFrame and it is the main class for the game board.

```
public gameScoreBoard()
```
It is a public class use to build object needed for the main class.

`private void initComponents()`
It is a private class for the scoreboard scope only and is used to initialize the variables needed for the gameScoreBoard class.

`public boolean isCellEditable(int rowIndex, int columnIndex)`
It is a public class with parameter rowIndex and columnIndex which return true or false for whether the game tile is clickable.

`public void run()`
It is a public class which return no values and is use to create and displays the game score board for the game

### 4.5 gameExit

`public class gameExit extends javax.swing.JFrame`
It is a public class which extends the jFrame and is the main class for the gameExit class.

`public gameExit()`
It is a public class which initialize the game components and create the object for the class gameExit

`public boolean endGame()`
It is a public class of type Boolean which ask the user if he/she really wants to exit the class

### 4.6 selectGame

`public class selectGame extends javax.swing.JFrame`
It is a public class which extends the jFrame and is the main class for the selectGame class.

`public selectGame()`
It is a public class which initialize the game components and create the object for the class selectGame

`public boolean run()`
It is a public class of type Boolean which ask the user if he/she really wants to exit the class

### 4.7 scoreMenu

`public class scoreMenu extends javax.swing.JFrame`
It is a public class which extends the jFrame and is the main class in scoreMenu.

`public scoreMenu()`
It is a public class used to initialize and build the object for the scoreMenu class

`private void initComponents()`
It is a public class with no return value and is use to initialize the components needed for the creation of the scoreMenu class.

### 4.8 selectGame

`public class selectGame extends javax.swing.JFrame`
It is a public class which extends the jFrame and is the main class in selectGame.

`public selectGame ()`
It is a public class used to initialize and build the object for the `selectGame` class

`private void initComponents()`
It is a public class with no return value and is use to initialize the components needed for the creation of the `selectGame` class.

`public void run()`
It is a public class which output no value and is use to create and display the `selectGame` form and linked to the gameMenu.

### 4.9 gameHelp

`public class gameHelp extends javax.swing.JFrame`
It is a public class which extends the jFrame and is the main class in gameHelp.

`public gameHelp()`
It is a public class used to initialize and build the object for the gameHelp class

`private void initComponents()`
It is a public class with no return value and is use to initialize the components needed for the creation of the gameHelpclass.

`public void run()`
It is a public class which output no value and is use to create and display the gameHelp form and linked to the gameMenu form.

# 5. DETAILED DESIGN

In order to choose between JavaFX and Swing as graphics and media packages to use for the JoggleCube game, two experimental projects have been undertaken; one in JavaFX and one in Swing. After working on both experimental projects for a little while, it was decided that for the purpose of this project, Swing would be a better choice. However, this also meant that in order to implement the designs that we were working on, we would have to make custom Swing components. This is what made up most of the second experimental project.

### 5.1 Significant algorithms

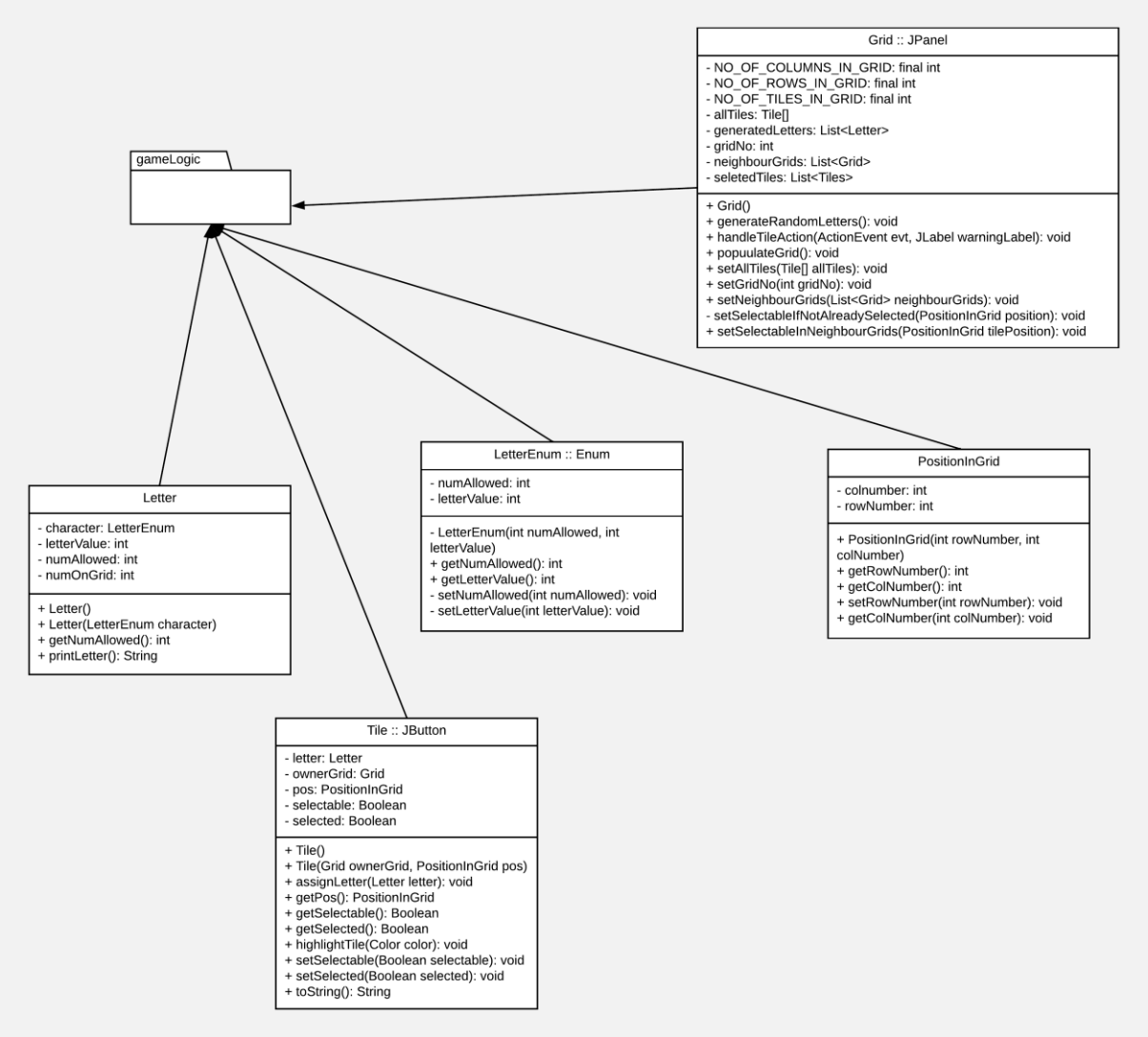
Making the custom components so that they are an appropriate fit for the needs of our project meant creating custom Swing components. This was made possible by using the already existing Swing components which had the most resemblance to the components we would require and implementing our own functionality on top of that. And so, the Tile and Grid modules were made to do exactly that, extending the JPanel and JButton components respectively.

In order to deal with the letter population which each had their own properties (maximum number of occurrences allowed in one grid, scrabble score of letter), we tried to implement those through Lists. But after experimenting, we realised that it would quickly become a very tedious task to keep up with that. That is when we found Enumerations to be a perfect fit for that element of the project.

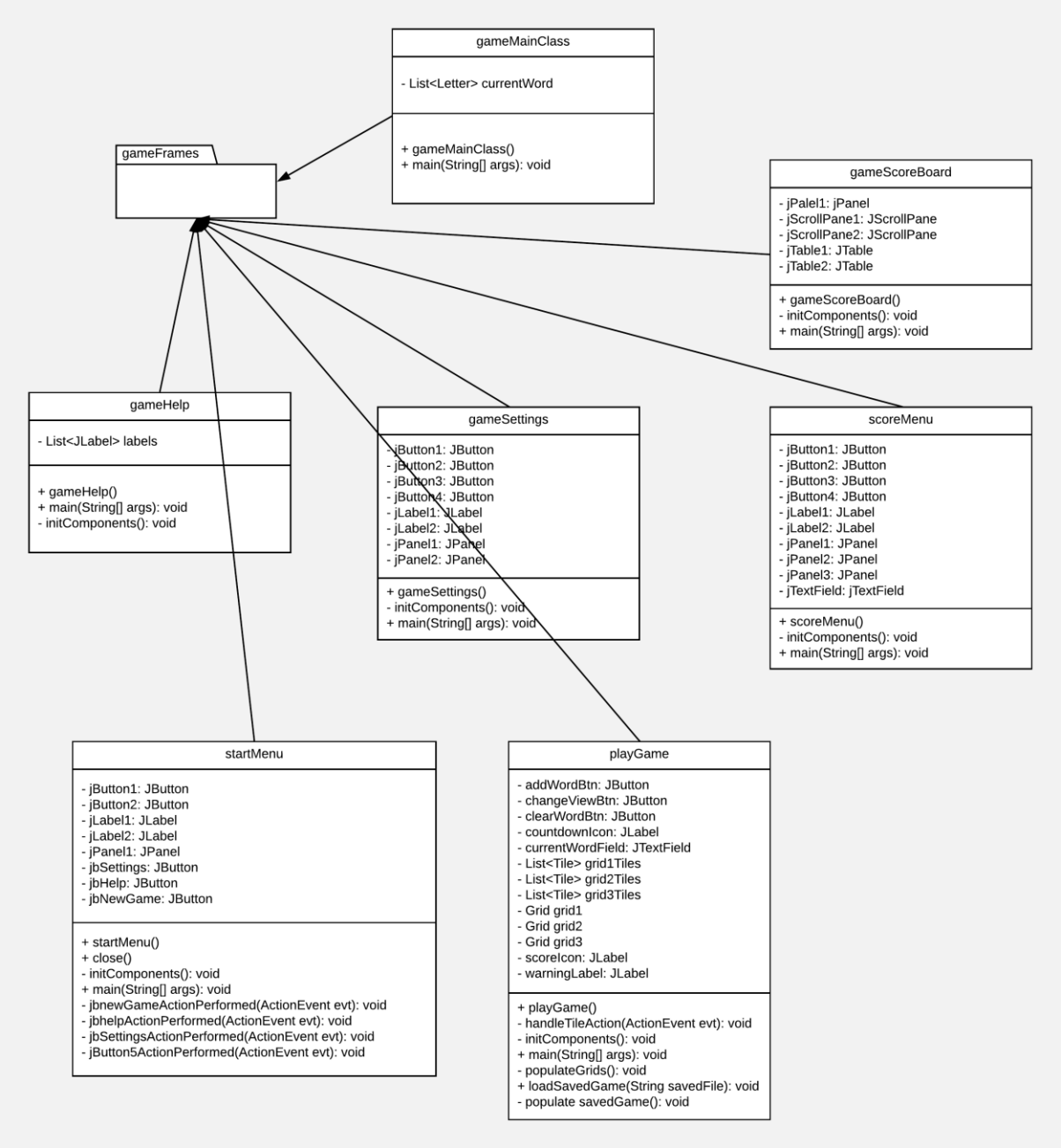
Implementing the required methods to enable the proper handling of the tiles in the grid when the user selects one tile was the task that required the most time to implement since it was working with several components at once. Eventually, we were able to write a short and maintainable algorithm for that purpose which is also extended on in the project.

## 5.2 Significant data structures

A class relationship showing the entity relationships between classes is shown below in the gameLogic package is shown below.



The one for the gameFrames package is below.



## 5.3 Files structure

### 5.4 High score file

The high score file is a simple text file and it stores the name and the score of the ten best players. The first line is the numbers of players details stored in the file. The next two lines is the name of a player followed by its score. The players details are sorted in descending order. The example below stores the score and name of the 6 best players.

```
6
Niman
930
Gin527
800
Minajdg
552
Nxgmkl37
550
Mila
400
Ndbg9
205
```

### 5.5 Saved game file

The saved game file is a simple text file. The letters in each row in the grids are stored on the same line. Each group of three lines corresponds to the letters in a grid, starting from the begin of the file. It is followed by the details of all the players who have played the game. The players details are stored in two lines and they are sorted in descending order. The first line is the player's name and the next one is its score. The example below shows the structure of the saved game file.

```
abc
def
ghi
jkl
mno
pqr
stu
vwx
yza
Niman
930
Gin527
800
Minajdg
552
Nxgmkl37
550
Mila
400
Ndbg9
205
```

# REFERENCES

[1] Software Engineering Group Projects: General Documentation Standards. C. J. Price, N. W. Hardy, B.P. Tiddeman. SE.QA.03. 1.8 Release

# DOCUMENT HISTORY

| *Version* | *CCF No.* | *Date* | *Changes made to document* | *Changed by* |
|---|---|---|---|---|
| 1.0 | N/A | 2018-04-19 | Main parts | nah37 |
| 1.1 | N/A | 2018-04-26 | Interface description | jty0 |
| 1.2 | N/A | 2018-04-26 | Files structure | nah37 |
| 1.3 | N/A | 2018-04-26 | Decomposition description | dkm4 |
| 1.4 | N/A | 2018-04-26 | Dependency description | dkm4 |
| 1.5 | N/A | 2018-04-26 | Significant algorithms & data structures | dkm4 |