Assignment 2

TI2206 Software Engineering Methods

Group 25
Pim Veldhuisen (4153448)
Jan-Gerrit Harms (4163400)
Amritpal Singh Gill (4419820)
Jeroen Vrijenhoef (1307037)

25 september 2015

EXERCISE 1

1. We decided to create a high score system as an extension of the game. The following user story and subsequent requirements describe the new functionality.

   User stories

   - As a user I want the game to remember my high scores.
   - As a user I want to see the current high scores before and while I play the game.

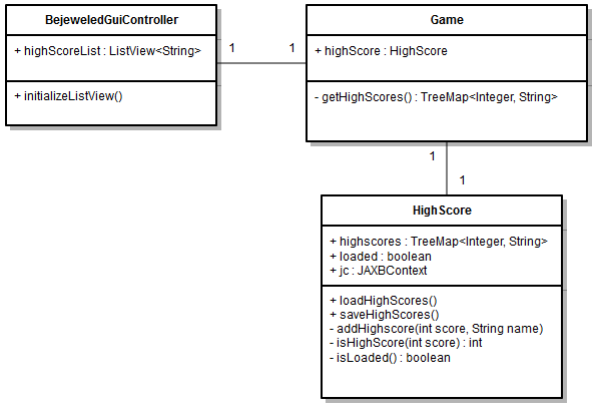
   These user stories can be translated into a number of requirements:

   Highscore requirements

   - Functional requirements
     (a) The game shall check if the current score is a high score when the game ends.
     (b) The game shall ask for the name of the player when the game ends and the player achieved a highscore.
     (c) The game shall save the top 5 high scores in a file.
     (d) The game shall display the top 5 high scores before and during gameplay.
   - Non-functional requirements
     (a) The game shall save the highscores in XML format.
     (b) The implementation should be done before 25th September, 23:55PM.

2. The UML for the HighScore extension is shown in Figure (1). In order to save the high scores after the game is quit we decided to use an XML file. The responsibility for saving and loading from this file is encapsulated in the HighScore class. As high scores are not specific to a game of bejeweled (high scores are a general concept in games that have a scoring system) we decided to add the HighScore class to

the Game class rather than the BejeweledGame class. This way the BejeweledGuiController class which is responsible for loading UI elements can use its reference to the Game class to load the highscores at startup.



Figuur 1: UML class diagram for the Highscore extension

EXERCISE 2

1. This extension will be a 'save' option i.e. functionality where the player can exit the game and continue at any later time.

   User stories

   - As a user I want to be able to quit the game and continue where I left off at a later point in time.

   This user story can be translated into a number of requirements:

   Requirements

   - (a) The game shall save the current state of the game, i.e. the jewels and their position on the grid and the score, when the user presses the exit button.

     (b) If a previous state is available upon the start of the application, the game shall ask the user whether to load the previous save state or start a new game.

     (c) The game shall delete any saved game state when the user starts a new game and doesn't load a previous game state.

     (The above requirements implicitly specify that at any time, there is either no save file or one save file.)

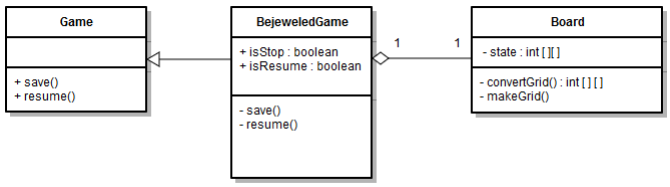
   - Non-functional requirements

     (a) The game shall save the game using Java Serializable Objects .

     (b) The implementation should be done before 25th September, 23:55PM.

2. The UML for the save option is show in Figure (2). We decided not to create a separate class for this functionality and instead give the Board class (which is currently responsible for storing and updating the

grid showing the jewels) and the BejeweledGame class (which is responsible for keeping the score) the responsibilities to save and load their current states. Figure (2) shows the functions and variables that have been added to these classes (and superclass) to achieve this functionality. In order to be able to store the current state of the classes we decided to use serialisation.



Figuur 2: UML class diagram for the Save and Resume extension