Assignment 4 Bust-a-Move

By
Calvin Nhieu
Jason Xie
Winer Bao
Justin Segond
Maurice Willemsen

TI2206 Software Engineering Methods

Supervisor:
Dr. A. Bacchelli

Teaching Assistants:
F. C. A. Abcouwer
R. van Bekkum
M. M. Beller
T. Boumans
A. Ferouge
J.E.Giesenberg

Exercise 1 - Refactor If-else statements

Refactor 10 if-else statements. Explain why you changed it and why not ?

1. Public Update() method in Cannon Class:

The method had several reasons to be changed, First the method was quite long. Second the method had a lot of duplicate code.
A big if-else statement had to select the key input for player 1 or player 2. But in fact it copied the whole code with other arguments. That is fixed with creating an 2D array which contains all key inputs for each player. Now the input is chosen simply with player id. Also the action for going left and going right where quite the same which are now taken together in 2 looping for-loop where a simply extra method controls the cannon movement.
Reduced from 90 lines to 30 of code.

2. Public Update() method in PausedScreen Class:

The Update method had to check if the mouse was pressed and if it was inbounds of every button. So for every button it had to an additional if statement needed to be given. In those if statements where all actions for when the button was pressed. Now every button stores its actions in its object while all buttons now are in an arraylist. Now there is an if statement that checks if the key is pressed and then loops all on all buttons and if its inbound that button is performs its action stored in its object.
This does not only simplify the if update method but also the drawing and centering function while they just have to loop over all buttons in the arraylist instead of calling each button on its own line of code.
This simplifies the screens for future improvement when buttons are added. Only in the init method now a button can be created, its actions can be added and then it can be added to the list while you don't have to concern on the other function any more for a call to the button.

3.Public Update() method in DefeatScreen Class:
Same story as for the PausedScreen.

4.Public Update() method in VictoryScreen Class:
Same story as for the PausedScreen.

5.Public Update() method in NameScreen Class:
Same story as for the PausedScreen.

6.Public Update() method in NamesScreen Class:
Same story as for the PausedScreen.

7.Public Update() method in MainMenu Class:
Same story as for the PausedScreen.

8. PowerUp apply method
The else ifs had an additional conditional to check whether the random number was higher than the chance of the previous if statement. This was redundant because the

previous ifs catches all cases where the random number is below that, and leaves the random numbers that are higher through.

9. BubbleStorage
removeBubble method
Rewrote a small bit to actually break outside the for loop.
getColumn method
Removed the if block, because that information was available from the array length.

10. Game Class
There is only 1 button to check for, so in this class we don't really need to apply the same technique as used in the previous screens, as it would not reduce the if statements in this class. What could be done is the moving of the if statements from the screen classes altogether and move it to the Button class. Where a method like Handle(Input) would check if the mouse was clicked and in the right spot. This would further reduce the amount of if statements in the BasicGameState classes.

Exercise 2 - Highscore feature

To start off we created a new requirements document in which we highlighted every noun.

RDD requirements

Although the current game is fun to play, it is not very competitive. To solve this dilemma, the game need a highscore system that keep the top 10 scores. A new highscore is a score that has at least a larger value than the 10th highscore. This new highscore is saved when the VictoryScreen or DefeatScreen is displayed. The name and the score of the player is recorded in a file stored in the project folder. The highscore screen can be accessed by pressing the highscore button located at the MainMenuScreen. The highscore screen displays a list of the top 10 scores and a back/main menu button. The back/ main menu button is used to go back to the MainMenuScreen.

Later, we went on to divide every noun in a category.

Conceptual entities:
Highscore, HighscoreScreen

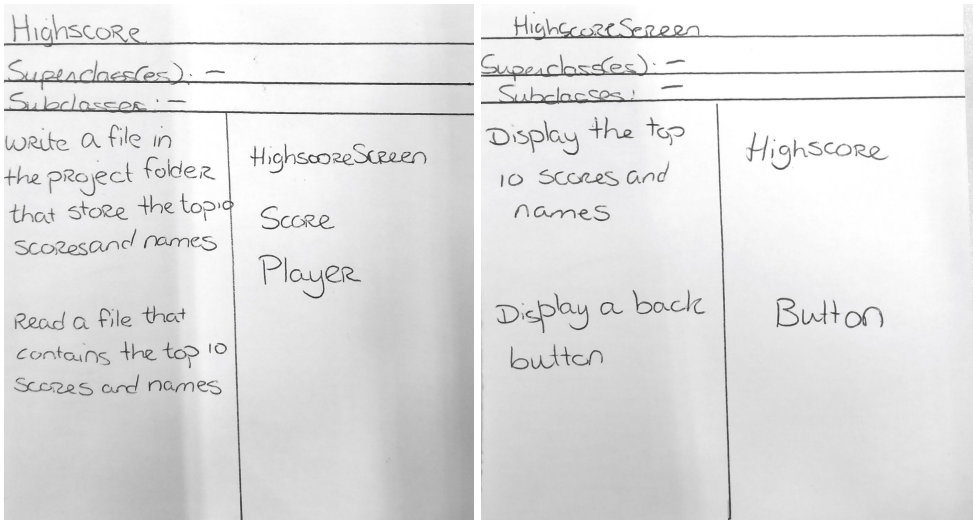
We also removed some of the nouns because they meant the same as others or they already exist:
One word for one concept:
Highscore system -> Highscore
Scores -> Score
Name -> Player
Highscore button -> Button

Back/ main menu button -> Button

Misleading subjects:
Dilemma, file

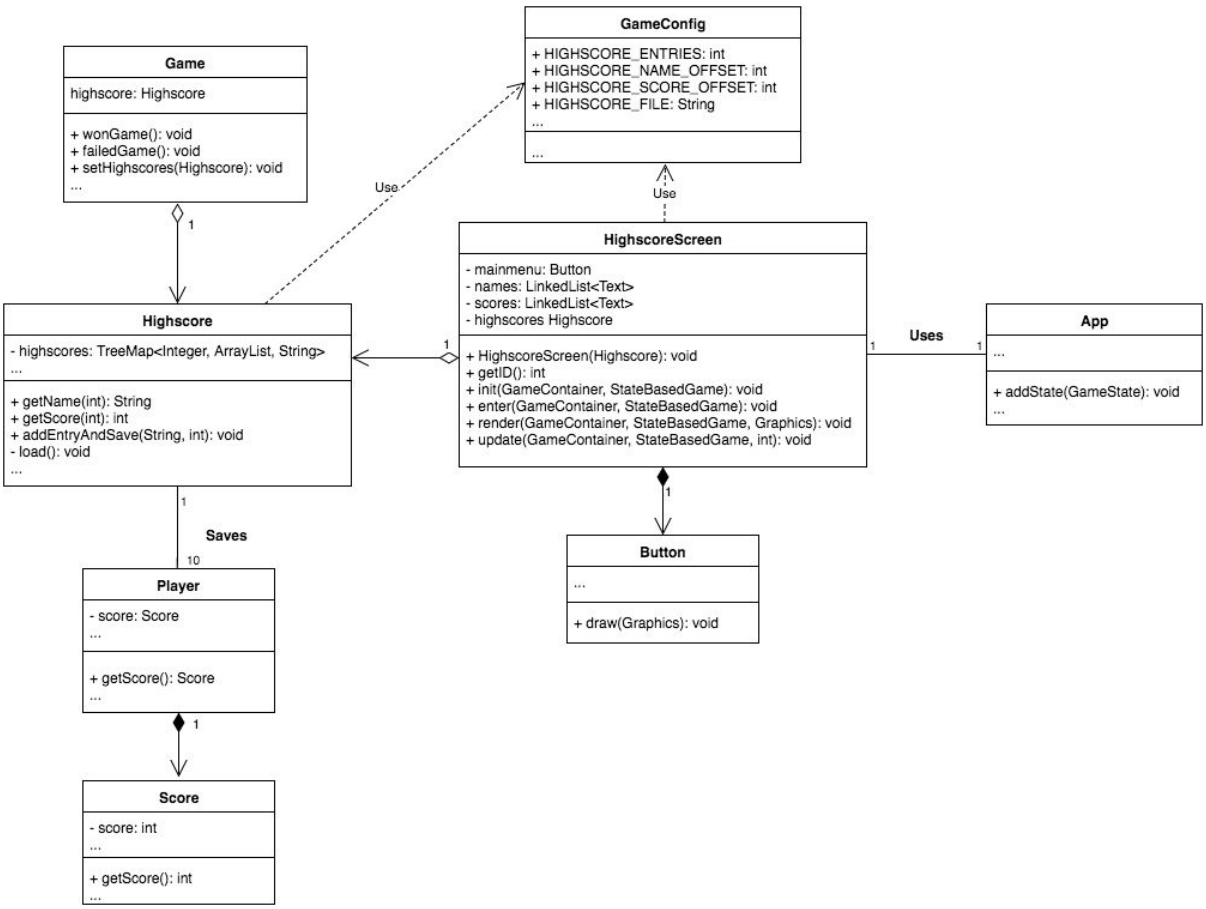
Already existing classes:
Game, Score, VictoryScreen, DefeatScreen

The next step is to define the responsibility of each class and the collaborations between each class. All the derived classes from this RDD redesign session are documented. The CRC cards show each classes with its corresponding responsibilities and collaborations.

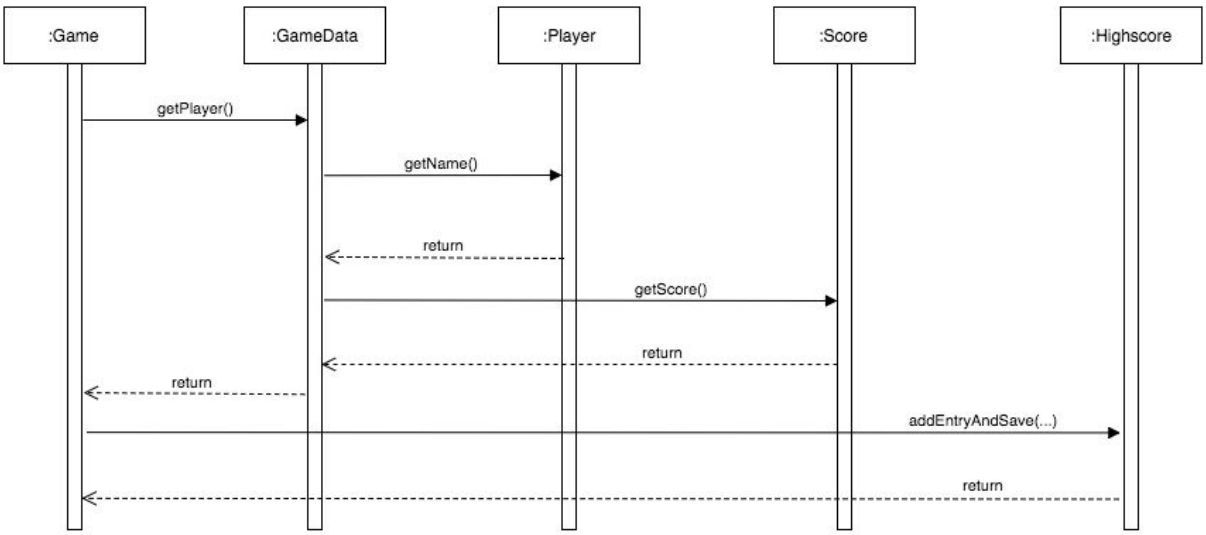


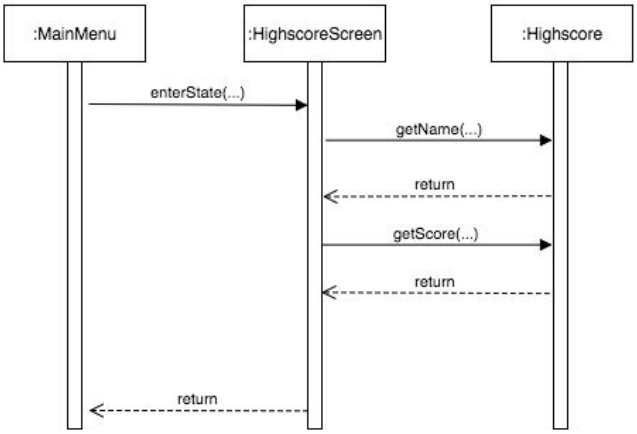
The Highscore class uses a singleton pattern since the class need to be able to be accessed on a global level by other classes. The highscores are also saved at one location; so only one instance of the class exist at a time. When the player wins or loses the game, the name and score of the player is saved in the highscore system.

The following design was implemented:



When the Highscore button in the main menu is clicked, the game state is changed to the the high score screen. The high score screen then calls getName(...) and getScore(...) of the Highscore class to display the top 10 scores. The sequence diagrams below show the methods calls to save and display highscores:





The following requirements are set for the Highscore implementation:

Functional requirements

Must have:

- The game must have a button in the main menu, which allows the user to enter the highscores screen.
- When the button is pressed, a click sound is played.
- The button is under the “2 Player” button.
- The highscores screen shows a list of 10 entries.
- Each entry consist of a player’s name and score.
- The entries are sorted by the score. The highest score is at the top and the other scores has a descending order.
- The name and score are stored in a txt file. This ensures that those data are retained after the program is closed. Those data are read back from the txt file when the program displays the highscore screen.
- When the player gets a highscore, a message is displayed in the VictoryScreen or DefeatScreen.

Should have:

- Achievement system that will reward the player with point if he/she reaches a certain point value.

Could have:

- Separate high score table for both single and multi-player mode.

Won't have:

- When a highscores is achieved, no animation is shown to notify the player.

Non-functional requirements:

Must have:

- Test coverage of at least 30%.
- Finish implementation before Friday Oct 21st 2016 5.55PM.

Should Have:

- Test coverage of 80%

Could Have:

- Test coverage of 100%

Won't Have:

- Finish implementation before Tuesday Oct 18th 2016 11.59PM.

Exercise 3 - Code review

The code review is done based on the following rubrics:
https://docs.google.com/spreadsheets/d/149ORB8l3CNLqk91O8y4xJWrUpRP9D7zW0UrRQwKaEzk/edit#gid=1109733044

Code (change) quality

- Nice use of Strategy pattern in the Bubble classes.
- Redundant if-statement in GameFactory.java line 42 and 50. If-statement is alway true. It is recommended to remove these statements.
- The game is turn-based. There is no apparent reason for this. One would expect a competitive game not to hinder the player controls. Remove the turn variable if this phenomena is not intended.
- Line 136 - 146 of SinglePlayerGame.java has a similar function as the gameOverScreen() of MultiPlayerGame.java. To keep everything consistent; move this to a gameOverScreen() method in SinglePlayerGame.java.
- The Logger class is the perfect class to use the Singleton pattern on. Nice work!
- Good Iterator pattern usage in Highscore classes.
- Good to separate the parsing from the HighScore data.
- The GameFactory is called as such but it doesn't implement the factory pattern.
- Redundant variable at Grid.java line 91. Just return grid directly.
- Redundant if-statement at Grid.java line 105. Simply return grid[i][j]. It will return a Bubble or null anyways.
- Usage of error.printStackTrace() not nice way to deal with errors. Use the logger class to provide a better problem/system tracking.
- Advice: add an iterator to the Grid class since you go through the grid elements many times in the code.

Code readability(Formatting/naming/comments)

Formatting

- Most of the classes are doing fine on line length and when it does happen, it usually occurs by itself and not a whole group. This is not the case in the Collisions class however. The code looks really clustered in that class specifically. Would suggest change the longer variable names to something shorter.
- Found formatting issue in GameFactory.java line 43. Add correct indents.
- Advice: add a variable to store bubble.getGridPosition() in Grid.java line 136. Use this variable to keep the code shorter.
- You guys haven't enabled the DesignForExtension rule check.

Naming

- getLeft/RightWall sounds more natural than getLeft/RightLine(the javadoc says that it returns a well as well)

- In the collision method of the Collision class, the quite long variable names are causing that conditional block to be quite hard to read. I would suggest to use something like newCenterX and gridCenterX as names.
- For the rest, all class/method/variable names are really concise, I like it alot!
- setAmountShot() in pop.java could use a better name. Rename to resetAmountShot().

Comments

- I'm not sure if something went wrong during the zipping or it's the personal preference of the group but every class javadoc contains line breaks in the middle.
- Empty javadoc block in Grid class.
- In general all methods have good javadoc that describes what the method does and what the variables are supposed to be.
- There are classes where there are a bunch of (albeit private) variables that don't have any javadoc/comments.(Ex SinglePlayerScreen) but in render method, there is a comment that says what a group of them is supposed to show/render. This won't result in point lost, but would have liked to see more of it.

Testing

- *For Calvin*

Grading:

Code quality: 7
Code formatting: 8
Code naming: 8
Code comments: 8
Code testing: *For Calvin*
Total: *To be determined.*