Fishy Requirements

TI2206 Software Engineering Methods (2015-2016 Q1)
Date: 11/09/2015
Group Number: 0
Group Members:

Thomas Baars - tcbaars@gmail.com
Lars Ysla - yslars@gmail.com
Boris Schrijver - boris@radialcontext.nl
Adriaan de Vos - adriaan.devos@gmail.com
Dylan Straub - d.r.straub@student.tudelft.nl

Table of Contents

1. Functional Requirements

For the game Fishy, the requirements which describe the systems services and functions have been grouped under functional requirements. These functional requirements have then been divided into four categories based on the MoSCow model for prioritizing requirements; where requirements marked as “Must Have” are crucial for the delivered product to be considered a success, “Should Have” requirements are important but not necessary, “Could Have” requirements are desirable and will be implemented if time permits, and finally “Would/Won’t Have” requirements are of the least importance and will probably not be included in the working version due to time constraints[1].

1.1 Must Have

1. The game shall display a new playing field, before a new game starts.
2. A new playing field shall consist of a rectangular area with zero enemy fish.
3. The game shall spawn a fish in the centre of the playing field, known as the player’s fish, when the game starts.
4. The game shall spawn other fish of various randomly generated sizes, known as the enemy fish, when the game starts.
5. The fish shall be 2-dimensional sprites.
6. The enemy fish shall be spawned at at either the left or the right side of the screen.
7. The enemy fish shall be spawned at different depths.
8. The enemy fish shall move onto the playing field from one side of the screen and exit the playing field from the other.
9. There shall be a minimum and a maximum number of enemy fish on the playing field at one time.
10. When an enemy fish is spawned the number of enemy fish shall be incremented.
11. When an enemy fish exits the playing field the number of enemy fish shall be decremented.
12. If the number of enemy fish is less than the minimum, the game shall spawn more enemy fish.
13. If the number of enemy fish is greater than or equal to the maximum, the game shall stop spawning enemy fish.
14. The player shall be able to control the direction in which the fish moves.
15. The player’s fish shall not be able to leave the playing field.
16. The game shall handle collisions between entities.
17. A fish shall be considered eaten by another fish, if the first fish collides with a larger fish.
18. Enemy fish shall not eat other enemy fish.
19. If a fish is eaten it shall be removed from the playing field.
20. If an enemy fish is eaten, then the number of enemy fish shall be decremented.
21. If an enemy fish eats the player’s fish, then the game shall be considered lost.

[1] MoSCoW method: https://en.wikipedia.org/wiki/MoSCoW_method

22. If a fish eats another fish, its size shall be increased.
23. There shall be a limit to how large a fish can become.
24. If the player's fish reaches the size limit, then every enemy fish eaten by the player shall reduce the number of enemy fish able to spawn.
25. If the number of enemy fish able to spawn is reduced to zero, then the game shall be considered won.
26. The application shall display a "Game Over" screen, when the game is considered won or lost.
27. The player shall be able to exit the application at any time.

1.2 Should Have

1. The player shall have a score.
2. The player's score shall be set to 0, when a new game starts.
3. The score shall be increased when the player's fish eats an enemy fish.
4. The amount, the score is increased by, shall be proportional to the size of the fish that is consumed.
5. The score shall be displayed in-game.
6. The "Game Over" screen shall display the achieved scored.
7. The fish shall face the direction that they are moving.
8. The player's fish shall be able to be controlled by the keyboard.
9. The application shall have a title screen.
10. The application shall display the title screen when it is launched.
11. The application shall have an instructions screen.
12. The instructions screen shall have information on how to play the game.
13. The player shall be able to navigate to the instructions screen from the title screen.
14. The player shall be able to navigate to the title screen from the instruction screen.
15. The player shall be able to start a new game from the title screen.
16. The player shall be able to pause the game, while in-game.
17. The player shall be able to resume the paused game.
18. The player shall be able to return to the title screen from the game screen.
19. The player shall be able to navigate to the title screen from the "Game Over" screen.
20. The player shall be able to start a new game from the "Game Over" screen.

1.3 Could Have

1. The application shall have a background image that adds to the appearance of being underwater.
2. The speed of the fish shall be proportional to its size..
3. The fish shall have animated tails, to simulate the appearance of swimming.

1.4 Would/Won't Have

1. A fish shall only be able eat another fish if its head collides with the latter.
2. The top ten high scores shall be recorded.
3. The current highest score shall be displayed in-game.
4. The "Game Over" screen shall display the current highest score.
5. The player shall be able to associate a 3 character nickname with the achieved high score, once the game is over.
6. The nickname shall be recorded with the associated high score.
7. The application shall have a high score screen.
8. The high score screen shall display the recorded high score information.
9. The player shall be able to navigate to the high score screen from the title screen.
10. The player shall be able to navigate to the title screen from the high score screen.
11. The application shall play background music.
12. The player shall be able to toggle the music on and off.
13. A sound shall play when a fish is eaten.
14. A sound shall play when the player wins.
15. A sound shall play when the player loses.
16. A sound shall play when a menu button is pressed.
17. The player shall be able to toggle the sound on and off.
18. The application shall have bubbles and sea plants to improve the appearance of being underwater
19. There shall be a counter for how many enemy fish the player has eaten.
20. The count shall be reset when a new game starts.
21. If the player's fish eats an enemy fish, then the count is incremented.
22. The player's fish shall decrease in size if it has not eaten an enemy fish within a certain amount of time.
23. The amount of time the player has to eat a fish before it shrinks shall be displayed in-game e.g. a hunger bar or a countdown.
24. The player's fish shall not shrink anymore when it has shrunk to its starting size.
25. The game shall have bonus items, such as pearls, which add to the user's score.
26. The game shall have an online multiplayer mode, where another player can control a second fish in the same playing field.

2. Non-Functional Requirements

For the game Fishy, the requirements which outline the constraints, on the system and the developments process, set forward by our stakeholders have been grouped under non-functional requirements; and are as follows:

1. The game shall be playable on Windows 7 (and higher), Mac Os X (and higher), and Linux Ubuntu 14.04 (and higher).
2. The game shall be implemented in Java 8.
3. Git shall be used as a version control system, via GitHub.
4. JUnit shall be used to perform unit testing.
5. Apache Maven shall be used for project management.
6. Travis shall be used as a continuous integration tool.
7. PMD, CHeckstyle, and FindBugs shall be used as static analysis tools.
8. Octopull shall be used to extend the functionality of the static analysis tools employed.
9. A first fully working version of the game shall be delivered on Friday, 11 September 2015.
10. For the iterations after the delivery of the first fully working version, the SCRUM methodology shall be applied.
11. The final product shall be delivered on Friday, October 30 2015.
12. For each iteration, excluding the first working version, the implementation of the game shall have a minimum of 75% of functional test coverage, which means tests are actually testing the functionality of the game and not just executing the methods involved.