Assignment 2

by

Group #6

Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science.

Mekelweg 4,

Delft

Name	StudentId	Email
Michiel Doesburg	4343875	M.S.Doesburg@student.tudelft.nl
Matthijs Halvemaan	4353803	M.J.W.Halvemaan@student.tudelft.nl
Dmitry Malarev	4345274	D.R.Malarev@student.tudelft.nl
Sunwei Wang	4345697	S.Wang-11@student.tudelft.nl
Clinton Cao	4349024	C.S.Cao@student.tudelft.nl

TA:

Valentine Mairet V.A.P.Mairet@student.tudelft.nl

Course: *Software Engineering Methods (TI2206)*

TU Delft

Table of contents

A small update

In this section, we will quickly go through some small updates regarding the classes and report of last iteration (sprint #1). These updates were made after receiving feedback from our TA for our work of last iteration.

Based on the feedback of our TA, we have changed the class of *Entity* to an abstract class. Therefore it is only possible to make an instance of *Entity* using on the subclasses (*PlayerFish* or *EnemyFish*) of Entity. We tried changing the Entity class into an interface. But after some consideration, we came to the conclusion that an Entity interface would serve little purpose. When we built Entity as an interface (and had PlayerFish and EnemyFish implement it), the interface contained just only intersection method. Leaving it as an abstract class means PlayerFish and EnemyFish can inherit common attributes like movement speed and sprite. This is why we have decided to leave Entity as a class.

In our last report, in the section where we describe the main classes of our project, we said that the class *BoundingBox* forms part of the main classes of our project. It has been pointed by our TA, that the class cannot stand alone by itself, as it is a part of the *Entity* class. Therefore *BoundingBox* is not part of the main classes anymore.

Also in our last sprint, we forgot to update the class diagram after implementing the logger for the game. The class diagram has been updated and it is in our repository.

Furthermore, we have changed *Game* class and *MainScreenController* class slightly. This fixes the *NullPointerException* that we kept getting when the player clicks on the menu or quit button, without actually playing the game. We also adapted the *Logger* class and added a JUnit testclass, *LoggerTest*, for testing the *Logger* class.

CRC card(s) for the new class(es)

Going through our requirements of sprint plan #2, we think the only word in the requirements that may represent a class are the words "power-ups" and "bomb". This is because in our own requirements, we have new features (that will be implementing this sprint) that does need us to create new classes for these new features.

We will try to explain this with an example. In our must haves, we have a requirement that the game should be able to play music. It is not needed to create a new class for music; we do not see how a music object/instance will be used between other instances like i.e *PlayerFish*. We just simply going to add a music theme for our game, so that the player can enjoy some music.

As for the words "power-ups" and "bomb", these are from the requirements of our TA. As bomb is a power-up, the only word that may represent a class is "power-ups". And the CRC card is given below.

Power-ups	
Superclass(es):	
Subclasses:	
Responsabilities	Collaborations
Represent an item that the player can use i.e to kill enemy fishes.	PlayerFish

Deriving responsibilities and collaborations for power-ups

The power-ups is an item that the player can use in game to i.e kill the other fishes (this is if the power-up is a bomb). Therefore the responsibility of this class is to represent an item that the player can use. As the player is the only one that can be able to the power-ups, this class will only collaborate with *PlayerFish*.

Comparison with actual implementation

We had orginally implemented the *Item* class to model a power-up. After FishBomb was completely implemented, the Item class contained only one attribute - a sprite. Obviously, that is pretty useless. That is why we have decided to remove the Item class and make FishBomb stand on its own. For now, FishBomb is the only power-up to be available, and we have no idea what any possible future power-ups might look like, so trying to predict common attributes and forcing in some inheritance structure simply is not beneficial. If more power-ups are implemented and there is opportunity for a parent class, we will look into re-introducing an item class. For now however, it is gone.

We also implemented the *FishBomb*. This class represents the bombs that the player will be able to use in the game to kill the other fishes. The main difference with what we had derived in the previous paragraph is that FishBomb models a fishbomb specifically, instead of the more general item.

Notes on test coverage

We have noticed that our test (line) coverage of our project is not very high, as seen from the .html files that are generated by Maven. The reason behind this low coverage is because we have one test class (*PlayerFishTest)* in the main package, not in the test package.

We know that a test class is not supposed to be in the main package, but in that test class we launch the game to test if the right image has been set to the player. If we do not launch the game, then we are not able to test it, as it gives us an error. Therefore we decided to launch the game for testing the class.

But by launching the game, you would have to close it manually for the test to finish. You can already see the problem of what will happen if we put this testclass in the test package and try building it on Travis; Travis will just stall and we might get a red build! Therefore we put the class in the main package so that Travis will not stall.

We also noticed that when the game is launched and you actually play the game, the test coverage gets very high after closing the the game! We have tried to do a “smoke test” to see if it will work on Travis, but so far no luck. We are still searching for a solution for this problem.