1. General

We were asked to build a F1 manager game in java. You should be able to buy drivers and engines, change strategy, upgrade staff, manage a budget and run a race. There is a total of five sprints of two weeks. At the end of three of those sprints there was a demo. First we created a UML to generally map out the structure of our code. For version control we used Github with Zenhub as an extension. We also used a slack channel and a Whatsapp group to communicate.

Generally the collaboration went well. All of us listened to others and were heard. For us version control was a necessary evil. The first few weeks were almost solely devoted to getting to know Git, Github and maven. Even though a lot of time was spent managing/assigning issues and trying to merge branches, without Git we could not have made the game.

2. Design process

problems

On our way to a finished project, we encountered a lot problems.

For example, a problem that has taken a lot of time to solve is the “infinite json loop” problem. We have a team with a list of drivers as an attribute. These drivers in turn had a team as one of their attributes. When saving the game, these classes had to be converted to json. But because of this circular reference this could not be done. A lot of things were tried to fix this but nothing worked as we wanted it to. Eventually we decided to change the driver.team attribute to a string: driver.teamid. The problem with this is that we had to change a lot of our already written code to make this work, which took a lot of time.

Another problem we ran into had to do with reaching classes. We never really thought about making a central place where we could reference to our player list and save/load methods. Our solution was a central Game class that has static methods(load/new game) and could also be instantiated with attributes like the list with drivers and the list with al teams. Basically “the game state”. This “game” object is first instantiated in the controller and then assigned to the static “main” class so it could be referenced by any class.

At the end of week 7 we had some branches that had a lot of improvements over the master but were never merged. Those branches had stepped off of the idea of “one branch per issue”. These branches became very hard to merge, because the differed to much from the master. This was not very productive when you wanted to add upon the newly added changes of those branches. Because you could not simply split a new branch from the master.

Technologies

graphics

for graphics we considered a lot of options but we decided to use javaFX pretty quickly because there seemed the most documentation and forum posts about it. Also both the presentation and our Project mentor mentioned javaFX as a good choice.

File System

For saving our files json was the obvious choice. It is easily readable and converts very easily to java classes with attributes. For This conversion we settled with Gson. Gson is made by google and is used a lot to handle json files in java. Both of these facts made us confident that Gson is a solid and well-made tool to reliably convert to and from json.

Testing

To test our application we went with Junit. We all were already familiar with Junit for testing because of the Object Oriented Programming course in the first quarter. We did however need some more functionality to test some methods. That is why we also used Mockito to mock classes in our tests

Other

We also used a framework called “Guava” to be easily implement preconditions to methods in our code. This makes our code a lot more reliable and readable.

UML

We used UML to map out the structure of our program.

insert UML here

This was good at the start of the project because it helped us visualize the project in our heads. However, along the way we encountered a lot of things that had to be changed and added. We had to make certain classes static and add a game class. This also meant that, close to the end of the project, we lost the overview a little. It often was not very clear where certain methods were and how to access them. But eventually we always managed to figure it out.

3. Points for improvement.