Report
Assignment 5

by

Group 33

Teaching Assistant
B. Reijm

Group members
Wytze Elhorst
Thomas Kolenbrander
Steven Meijer
Bart van Oort
Ruben Vrolijk

Contents

Exercise 1 - 20-Time, Revolutions **2**

Exercise 2 - Design patterns **3**
Factory pattern 3
State pattern 4

Exercise 3 - Reflection **5**

Exercise 1 - 20-Time, Revolutions

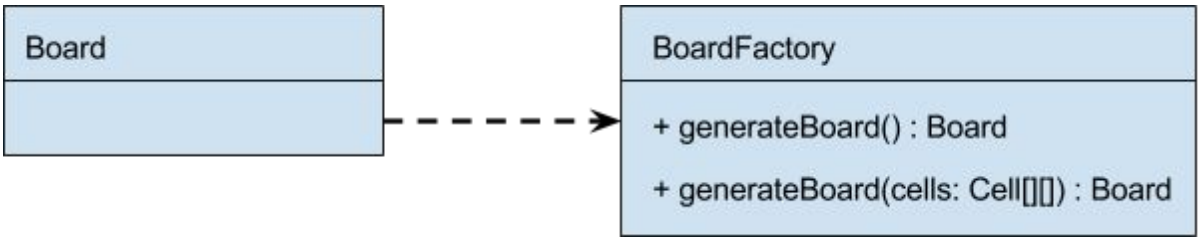
Yet again we have the chance to choose a feature we would like to add to the game. This time we decided to implement a new game mode, the time trial game mode. The time trial is a classic game with a time limit of 2 minutes. In this 2 minutes the player should achieve the highest score possible.

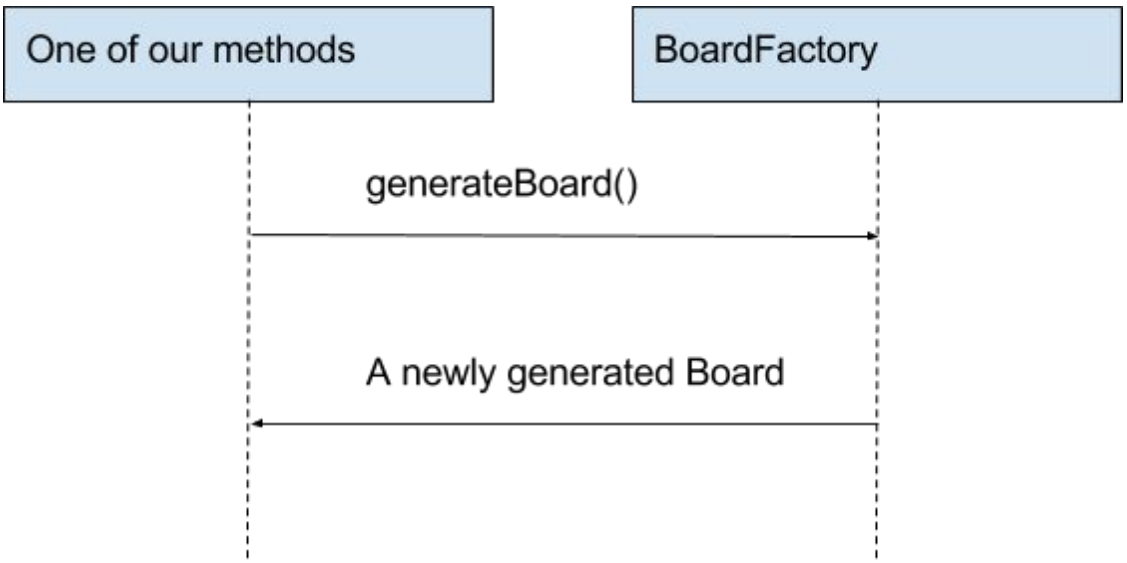
We believe this new game mode is a great feature because it chances the way a user will play the game. The time limit will put pressure on the player to make quick and good decisions, this will result in more active and fast gameplay.

Exercise 2 - Design patterns

Factory pattern

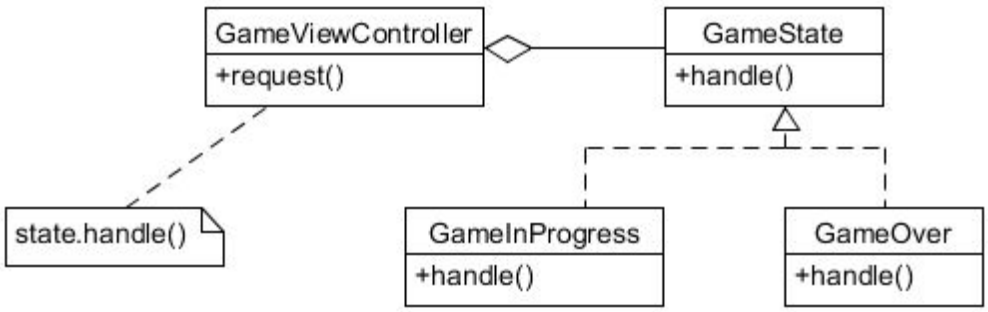
We decided to implement the factory pattern by making a factory for our Board class. The reason why we choose to implement this design pattern is because it allows for expandability. When we would decide to make different types of boards we could have one factory handling all the types instead of each type of board having its own constructor. This allows for really easy implementation as we don't need to specify which type of board we need in advance. Since we don't have different types of boards (yet) the implementation is rather easy. The creation of a board object is simply moved to the factory and instead of calling the constructor the factory is used.

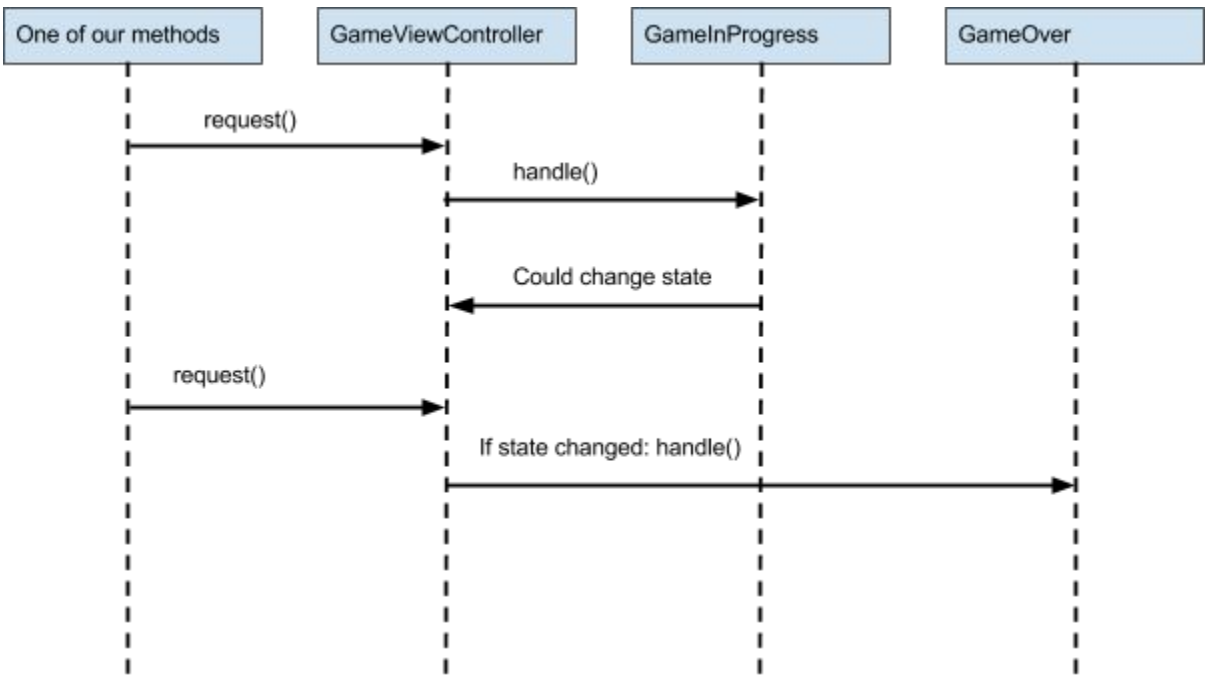




State pattern

To implement the state pattern, we introduced GameStates. These are different states that the game can be in. We chose to use this pattern for this, because it allows for the addition of more states later on. So far we only have two different states: Game In Progress and Game Over. These states are handled by the Game View Controller, which contains a GameState and is able to switch between states. When the Game is in progress, calls amde to the state will be directed to the actual Game object, since a Game is a subclass of a Game In Progress. When the state is Game Over, the game stops functioning except for a restart. Since it has to know which type of game to restart, it also has a field to contain the Game that is over. By checking this field, the right subtype of Game can be restarted. With this field, it is also possible to check the player's score and the board while in the Game Over state. The handling of the states in the Game View Controller happens automatically when a new Game is made and when the Game is set to be over. Every new Game made is a subclass of Game In Progress and therefore starts In Progress. When it is over, the current state is set to Game Over and a new Game can be made.





Exercise 3 - Reflection

There have been a lot of changes in our project during this course. Both in the GUI and in the way our code was designed. Aside from these changes we also learned about working together using SCRUM and to learn from our mistakes. To properly reflect on these things, we divided them into the different categories below.

Graphical User Interface

One of the most major changes has been in the graphical user interface. In our original version we used Swing to set up the interface, but after learning that Swing is getting a bit outdated and that it would be harder to add things like animations to it in the future, we rewrote our entire interface to utilize javafx. Another big change in the user interface and also the rest of the code came when we wanted to add animations to our game. We had never really used animations in java before, so adding animation was new to all of us. On top of that we had to rewrite a lot of our old code because it was not suited for animations.

Mistakes we learned from

We bit off more than we could chew the week we added animation and were unable to release a stable version of the game before the deadline. The following week we learned a lot about how to tackle the bugs still present in the game, we had never really thought of creating a proper document containing all the bugs, what they do and how they are triggered. Due to this we were able to remove all the bugs in the user interface before the deadline of the next week.

There had been one other time where we did not deliver a stable version of our project. This however, we didn't find out until the day after the deadline and the submission of our project. The tricky part of this issue was that while we were working on the code, the program worked perfectly. The issues arose when we all pushed it to the master branch, right before we wanted to hand in our project. We had failed to see that this could create merge issues, so we didn't properly test our master branch before we handed in the project. After learning this the hard way, we don't think we'll forget to test everything properly anytime soon.

Design Patterns

Alongside all the changes we made in the UI and the changes we had to make to make our code work, we also changed a lot of our code to fit certain design patterns. These changes wouldn't really change the game on a visible level, but would do a lot behind the scenes to improve the code by using design patterns that a lot of professionals also use in their projects.

In our original version of the game, we had no idea of these patterns and as a result never used them. If there were any similarities between certain design patterns and our original code, it was most likely a coincidence. After learning about the importance of these design patterns we started changing parts of our code to contain these patterns. The first two patterns we used were the observer design pattern and the strategy design pattern. These two were used to update the score and represent different types of gems respectively. Now in this current assignment we are adding two more design patterns to our project. These design patterns are the state pattern and the factory method pattern. There are still a lot of other patterns that we learned about during these few weeks of software engineering

methods and although they won't all be used in this project, they will definitely be something to keep in mind while working on other projects in the future.

Teamwork and SCRUM

We had all developed projects in teams before, namely last year during the OOP project and Project MAS. So we weren't totally new to working together and using git to share our project. We had never worked in this exact group before though, so at first we had to get to know each other and find out what the best way of communication would be. We never had any real issues with communication though and everybody always delivered what they were assigned to.

The weekly meetings were also something we had experienced before in the two prior big projects listed above. The thing that was new however, was the weekly deadlines. In the other projects we had guidelines as to where we should be every week, but no real deadlines and just one big deadline at the end of the project. This usually meant that you'd slowly get behind on schedule and then you'd have to work harder the last few weeks. These weekly deadlines made it so the workload was distributed evenly throughout the project, as there was no room to fall behind. Although some weeks we would have wanted a bit longer between two deadlines so we could tackle bigger things like animations, which took more time to implement than one week.

Another thing new to this project was the constant requirements documents and UML's we had to make. And although we had used both in project MAS and OOP respectively, we had never used it quite to this scale. In the end this provided us to quickly overview what our program does and how it does it, without actually having to look at the code. Meaning that if someone wasn't working on the code one week, he would have no trouble getting back into it the following week. It also made it easier to get a grasp of the whole project and how you could add additional code.

Conclusion

So in the end we learned a lot about GUI, design patterns, the SCRUM method and how to deal with issues that arise. The main difference between this projects and prior ones are that this project didn't focus on if the product was working, but instead how the product was working. This was something we had not really focused on before so we could learn a lot of new things from it. Since a lot of projects will require using the same design patterns, knowing how to use them will be a huge help in the future. Now we will be able to look at projects as a software developer as well as a software engineer.