Space Invaders : Reflection

Group 22
TI2206 Software Engineering Methods

Ege de Bruin
Bryan van Wijk
Dorian de Koning
Jochem Lugtenburg

October 22, 2015

Supervisor: Dr. A. Bacchelli
TA: Danny Plenge

Delft University of Technology
Faculty of EEMCS

Introduction

This document contains a reflection on the source code evolution and the route followed to create the final version.

Reflection

Requirements

In the early stages of the project, selection of the game to implement took place during the first lab hours. After selecting Space Invaders as the game to implement, the team started defining requirements using the MoSCoW method. During the project these requirements were extended, as new features were added to the game. Defining requirements in a clear and ambiguous way helped to distinguish tasks to implement and, without requiring long discussions about how to implement a certain feature. During the following sprints, the requirements were defined after the creation of the sprint plan.

Team

As a team, the project was started with five persons. Losing one person in during the first iteration, we had to carry out the work of one extra missing person. Because of this, the team quickly realized we had to maintain a clear organization. Making sure the organization of the project was defined using clear documents and following SCRUM we could implement features relatively fast with minimum overhead.

Sprint Iterations

The sprints were carried out using the SCRUM methodology where each iteration consisted of a meeting on Tuesday and a daily status report using the Whatsapp messaging service. Each Tuesday meeting started with an evaluation session of the previous sprint, resulting in a sprint reflection document. During this evaluation the team focused on improving the weekly iterations. Among the most important things the team improved were time management, making it more easy to approximate the required time for a task. Also the addition of smaller features to the game became an improvement to the process. The team discovered that implementing small features increased the quality of the product. Large features or major code refactors, such as a refactor of sprite loading, increased the amount of bugs on the final day of each sprint, causing risks to handing in a stable deliverable. During the final sprint, Code Review was also improved. Since a huge amount of Checkstyle errors slipped into the stable branch, the team decided not to merge pull requests containing Checkstyle errors.

After evaluation the new sprint was defined in a Sprint Plan which clearly states the tasks required to solve the weekly assignments. As more iterations followed, the Sprint Plan was improved with prioritization and a more clear definition of the responsibility per task. After defining the sprint plan the team began working on individual tasks together during the remaining lab hours. Each team member kept in touch using Whatsapp on a daily basis, posting the performed tasks, todo tasks and problems that occurred.

Code Review

Each feature addition or refactor was merged into a develop branch using pull requests. In these pull requests at least two team members had to approve the code change. The code change were reviewed on implementation, CheckStyle and understanding of the code. Code Review helped the team understand each others code, and solved a small amount of bugs. Before handing in the final version for a sprint, the develop branch was merged in to the master branch. The team decided to work with a separate develop branch, to make sure the master branch always contains a well-tested, stable version of the game.

Implementation

During the implementation of the product the team focused on maintainability and easy extension of the game. This payed off in later iterations, because it became very easy to add new features to the game, such as the Boss level using Waves. Design patterns such as the strategy pattern made it more easy to achieve our goal of easy extension of the game. Comparing the first version to the current master branch, it becomes clear that a lot of features have been added. Comparing version v1.0 of the game, it becomes clear this version had a good performance, and high test coverage. The current master branch also performs well, and has high test coverage, but the system has grown in size. Also the system is still extendable and maintainable in an easy way. Running inCode on the first version of the game, detects the Game class as a God class. In the current master branch this has been solved by sharing responsibilities over different classes. Sharing responsibilities makes it more easy to extend and maintain the game in the way it is intended. This proves that the use of tools such as inCode help detect these problems in an early stage. While the God class design flaw was resolved in another iteration, this design flaw could have been detected before the first iteration using a code analysis tool, increasing code quality. The first version of the game also didn't make use of design patterns, which could have helped to make the code more extendable, which was solved later by implementing the Shootable and Movable interfaces.

Conclusion

During the Software Engineering Methods lab project, the team learned how to develop a Game by using standards and techniques taught during the lectures. It helped greatly in understanding SCRUM, design patterns, GitHub and the overall organization of a project. Knowledge was gained about methods which help to improve iterations, which can be used to make future projects more easy to manage.