Method selection and planning

<u>HardGForGifs</u>

Dragos Stoican

Rhys Milling

Samuel Plane

Quentin Rothman

Bowen Lyu

Jack Gerhard

a) Software engineering methods

We choose to do an agile "SCRUM" (Atlassian and Drumond) method approach [1]. We believe it is the ideal candidate for this for the following reasons:

- Since this was a first for a lot of us and there would be a lot of continuous learning, and the requirements that we have defined earlier might need to be revised, which is great about scrum where the "product backlog" is constantly revisited and maintained to ensure that the requirements are still correct.
- Gave us the opportunity to focus every sprint by allocating a certain amount of tasks to our "sprint backlog" and we would attempt to have that fundamental goal implemented by the end of the sprint.

An alternative perhaps, could have been a plan-driven approach. But due to how we work as a group this wouldn't be nearly as efficient and would handicap our performance. Since our requirements were changing throughout as we better understood the tools we were using, the excessive documentation would be of no use since we could have changed our approach entirely and rendered time loss. If we were working as a larger group or been given a larger scope project, then it would become more appropriate to use. But since the scope isn't too large it makes agile scrum ideal.

Having all these tools is great, but the main thing that is needed in any project or team is communication. Unfortunately, with COVID-19, there are no real ways to have a lab to work in together in person. And even if we did, it wouldn't be enough. So we spread our communication across three areas, each being played to their strength:

1. Our online meetings were done through Zoom and were effective for important discussions, not as good documentation.
2. Our instant messaging application, Discord, provided the nice in-between the two extremes, providing both text and voice functionality, and letting you also organise them into channels, which is quite powerful for keeping our communication organised.
3. Our website acted as the long term memory, important information that occurred in the other platforms were eventually merged onto it, So it gave quick access to documentation, meeting minutes and links to our repositories.

To aid development, we took advantage of a couple of tools:

- Git, provided version control, the branching functionality allowed features to be developed separately from the stable codebase, then after being safely implemented could be merged back. The commit history is also useful as it made all committed code still accessible even after being deleted. An alternative could be Mercury, which does have a simpler user interface, git has also become ubiquitous in the industry, and provided a better real-world experience.

- GitHub offers git as a base, but it adds a lot more functionality that makes software development easier. One feature would be the issue system, as it allowed us to convert our system requirements into a trackable object, where we could comment on it, assign it different statuses so we could see how complete the issue is. This integrates excellently with GitHub projects which allows you to put those issues on boards, so we could visualise how each issue has been progressing, who has been assigned, how old, etc. An alternative could be Trello, which does have the card system and lets you transfer it between boards also, but we decided not to adopt it since GitHub allows a much tighter integration with the code, and being able to reference directly and with some provided assistance is very useful.
- Google Drive, provided collaborative document editing, this was particularly useful since we often did paired work. Also, the version history is useful for being able to retrieve previous versions of work. There are alternatives like Office 365 / OneDrive, but this wouldn't probably wouldn't work since our University accounts only provided the basic tier and we needed to be able to share with the lecturers.

Those tools were great at help for assisting. But for developing the game we also used some other tools to accelerate the progress:

- LibGDX, provided an easy to use API built on top of openGL and quite cross compatible. We chose to use libgdx since there was a rich amount of tutorials available to beginners and let us get up to speed in a rather short amount of time. A possible substitute that we considered was lwjgl, which is lower level and could offer more functionality but that does come with a lot more learning than what we had time for. So ultimately decided against it due to us wanting to get familiar with what everyone else was using so our skills would be more transferable.
- GitHub Pages, which allows us to host our own website for a more public view of our project, without the lengthy and hard process of making our own website.

b) Team organisation

Since we are a small group of students where all our work is being done remotely, and with one of our members being in a largely different timezone it required us to be quite flexible with how we were organised as a group. Consequently we used an agile method, specifically a scrum as it suited our small team size, need for flexibility and use of customer meetings.

To be effective, we set out to accomplish a few preliminary goals before getting into the scrum cycle. Firstly, we had ice-breaking activities so we could find out what people were best at doing and also seeing how we interacted with each other as a team. This proved useful as it allowed us to delegate our roles out easily and efficiently, although we didn't always strictly follow the roles that we set out for ourselves, and people gravitated towards what initially wasn't expected, but ended up performing well at the task. So at the base we had our set roles, and we used that to our advantage by assigning tasks to people who had the most appropriate role, allowing for further flexibility in development

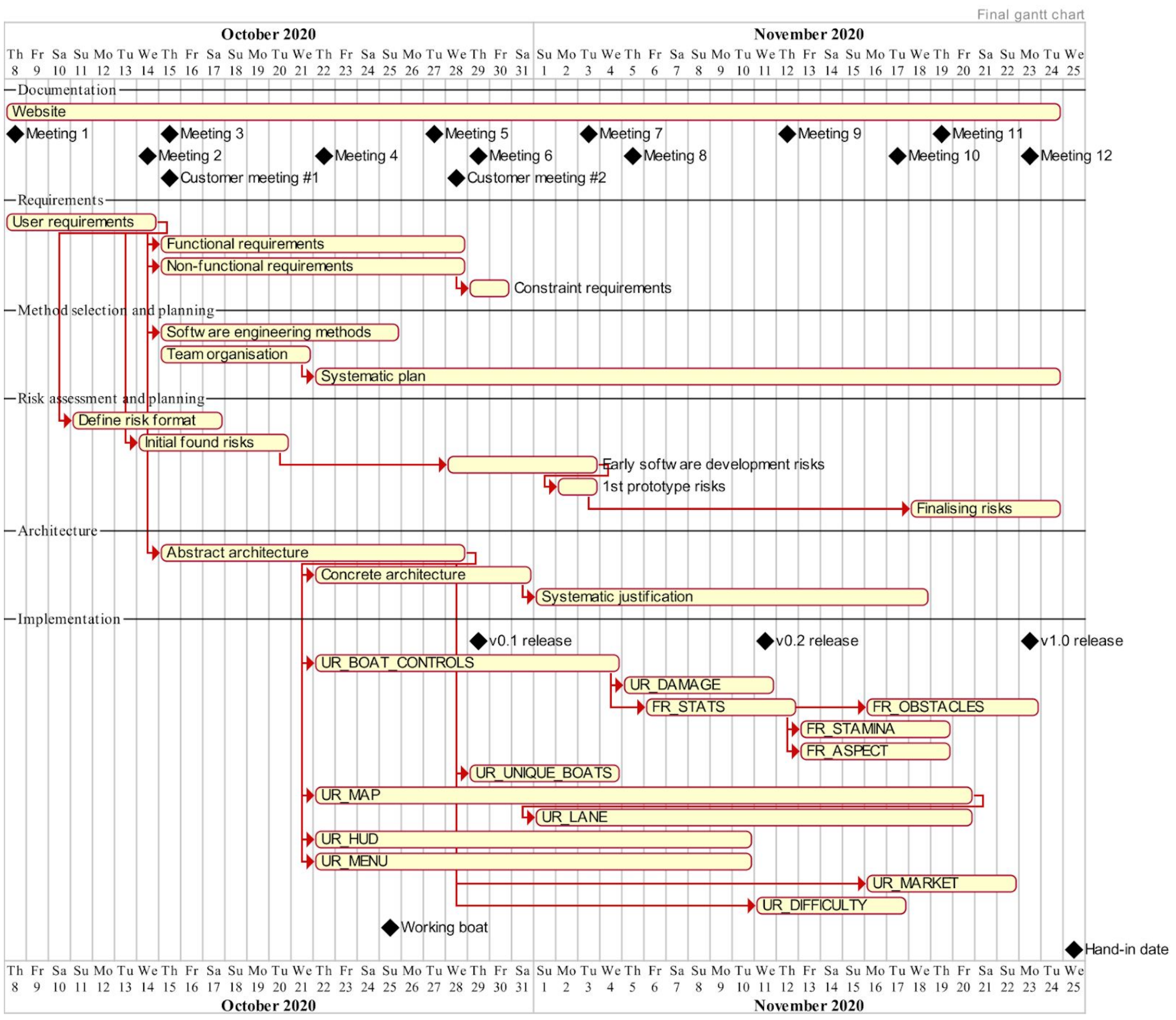
Furthermore, we also organised set dates occuring every week, allowing us to define sprints, and partake in a scrum. We also held sub meetings where people working on similar tasks would meet and work together to bounce ideas off each other and help develop the project further.

These meetings provided some core functionality to how we organise ourselves and allowed us to be the most productive.

- It allowed us as a team to make large, educated, decisions about the approaches / tasks we set out to do due to the scrums, and gave everyone a chance to raise concerns and resolve any conflicts before we went ahead and implemented that item.
- We also used them to catch-up and check progress with what people had been working on since the last meeting, enabling us to document progress in the minutes and let the group reflect on what was produced and see if any improvements could be made. This provided a good incentive to create high-quality work, acting like a positive feedback loop, whilst also fitting in well with the need for frequent releases in agile methods
- The most important factor from these meetings was giving us a solid platform to delegate tasks out, which aided us heavily in completing the majority of the project. After discussing a task, and breaking it down, we each would choose a sub-task. In some cases two people would collaborate on the same task in a pair programming fashion, which was useful for larger and complex tasks and allowed us to complete them within a set timeframe we assigned it.

Overall, these meetings helped us focus on what needed attention, where we would define clear goals, keep within scope, and maintain a high level of efficiency throughout the project life.

C)



The plan for the project was to build the game using an agile method approach, allowing us to modify existing elements as we progress through the project. So with our agile approach, we decided to put ourselves on two week sprints.
To map our progress we also produced a new gantt chart every two weeks. The one provided here is the final one, but on our website there are multiple available.

First two weeks:

- In the first two weeks, starting on 8th October, the key tasks were creating the means of communication within the team, designing the user requirements and deciding which game engine we're going to use, and getting used to it.
- The main focus here was understanding the project, getting to know each other and coming up with ideas for some of the key features of the game.

Third week:

- After these two weeks, we had everything ready to create a rough abstract architecture of the game. This was a very important step, as now we were ready to distribute some tasks between the team members, in order to be able to create small bits of code that fulfill some of the tasks in the abstract architecture. We decided to follow the same tutorial and build some of the requirements we designed last weeks as standalone, small projects. These included a moveable object script, a map rendering script, a UI rendering script and a collision detection script.

- These small projects were meant to help us understand how we want to build the game and how things work together best.
- This is also when we created the team's website, where all the deliverables will be published.
- After a week of work we were ready to move to the next step.

Fourth week:

- Starting in the fourth week, we already had a lot of scripts that were working great by themselves, but we now had to focus on putting everything together.
- Since we were now a lot more experienced with the game engine, we were able to start designing a concrete architecture, based on inheritance and polymorphism, that we thought would fit best with what he had done already.
- The main priority was designing code that is highly reusable, focusing a lot on simplicity and readability.

Fifth week:

- In the fifth week, the first prototype of the game was ready. We now needed to focus on the documentation aspect of the project, which included the systematic justification, a first draft of the risk assessment and more work on method selection and planning.
- This is where the agile "SCRUM" approach came in handy, as we were modifying some information based on what we learned since we first drafted it.
- Meanwhile, the development of the game continued with adding new features and merging core aspects of the game with UI elements.

Sixth week:

- The sixth week, which corresponds to week seven of the term, was mainly focused on completing the development of the game, as we were planning to have a release with all the mandatory features implemented by the end of this week.
- On top of that, we were always updating the documentation as necessary

Seventh week:

- In the seventh week we had everything ready for the 1.0 release of the game, so we focused on bug fixing, revising deliverables, adding code documentation and updating whatever is necessary so the project accurately matches the concrete architecture, including variable names and method names.

Eighth week:

- In the eighth week, we worked on the deliverables to improve get them to their best state, whilst also trying to polish the product, changing the scenery, and attempting to add user-friendly transitions between game states, although the latter was unsuccessful

Bibliography

1 Atlassian, and Claire Drumond. “What is Scrum?” *Scrum - what it is, how it works, and why it's awesome*, https://www.atlassian.com/agile/scrum. Accessed 24 11 2020.