Method Selection and Planning

Team 17: Scone Zone
Micheal Calaciuti
Sabrina Djebbar
Yaseen Khan
Tinetariro Muzunzandare
Josh Saunders
Zac Spooner

Justification of Methods

- The software engineering method we used was a structured, sequential method based on Jackson System Development (JSD).
- We integrated this engineering method with the project management method known as the “waterfall approach” to create an overall approach to the whole project.
- We realised very soon during the initial method selection that JSD’s structured methodology mapped very well with our desired linear approach.
- However, there were a few steps outlined in the methodology that was optimised towards very large organisations and teams; we streamlined these for our much smaller group of six people.
- The first stage of JSD is abstracting aspects of the software by modelling the relationships between the actual system and the intended game world. [1]
- Each process, class and entity in the software has a one-to-one relationship with aspects of the game world.

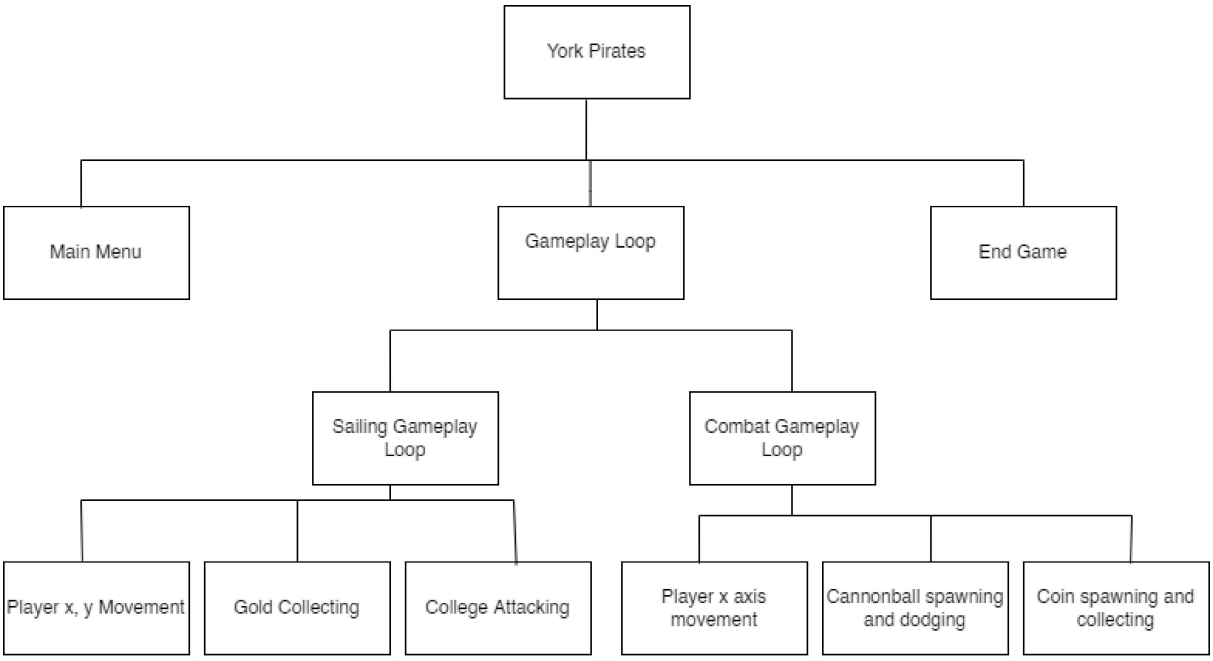

Fig. 1, our initial JSD diagram to plan and map out the different processes that will be taking place in the game

We used a range of collaborative tools to aid in the development of the program, which was important as most of the development was done on separate networks and machines.

- The first of these was GitHub, a version control system. All team members were co-collaborators on a GitHub repository, and the team members working on the game development as well as the website would work on their own branches, and then gradually merge them into the main repository when the requirements for that stage in development was met.
- GitHub lent itself well to JSD because it meant that team members could simply look at the JSD entity structure diagrams, and then create a new branch in the shared repository and work on a particular entity.

- Later, when that feature set was complete, these branches could then be merged into the main branch; this was our general developmental workflow, and the diagrams created by JSD were key in this endeavour.
- Another collaborative tool that was key during development was Google Drive. Google Drive allowed us to upload all documentation, design diagrams, illustrations and tables that would guide the development for the entire team.
- Another important feature was the seamless integration of Google Docs into the Drive ecosystem, which allowed multiple users to work on the same text document at the same time; an important feature for working on a large document together.
- The main communication tool that we used was Discord, which we used to engage in voice call team meetings, communicate ideas and discuss the tasks that were ongoing throughout each phase of the project.
- We chose a structured development method as opposed to a flexible, iterative approach [3] such as Agile because the engineering requirements were not subject to repeated changes by the client - we had a fixed set of requirements that we received at the start of the project.

Team Organisation

- With regards to team organisation, we took a range of approaches into account.
- Organisation regarding teamwork needed to happen early in the project, so during the first meeting, we created a google drive document to split up the project into different tasks and highlight the key members for each role that was to be filled.
- In the initial stages of the project, we decided to split up into a group of three people to work on the game, three on the documentation, 1 on the website and two on the art (with some people working on multiple roles), with roles being assigned loosely so that members could work on different aspects of the project if they wished and as needs arose.
- The roles were assigned naturally, as people assigned themselves to the roles, they felt they had some level of ability in.
- In addition to this, we had weekly meetings both in the labs and on Discord calls so that we could continuously re-assess the progress we were making towards the project goals, and re-focus our efforts on any aspects of the project that needed attention.
- For each of these meetings, we created a table in the shared team meetings document which outlined the present members and which aspects of the project was being worked on.
- All these weekly meetings corresponded to the progress Gantt chart that was made early in the project lifecycle to ensure that we made good progress towards our goals.
- In addition to these methods of synchronous communication, we also sent messages on the Discord group chat periodically to update each other on our progress regarding each of our tasks and also communicate any concerns or questions.
- The team maintained a flat hierarchy, with no designated team leader - any ideas were put forward in meetings and the group worked together to go through concepts and plans and achieved consensus on major decisions by going through this process.
- In addition, once the broad aims and objectives were established, team members had the freedom to achieve those project goals as they saw fit.
- The idea behind this was to break down the tasks available to the extent that each person works independently and efficiently within their domain, and then with members working together during the integration phase when individual tasks were combined, such as in the case of multiple members working on multiple different classes in a Java project.
- This gave autonomy to members which reduced the overhead associated with a single team leader but still maintained implicit deadlines as small groups of 2-3 people worked together in parallel to solve a larger problem within the scope of the project.
- This meant that we did not run into any major issues regarding deadlines or participation, as progress would be updated regularly

Task Plan

NB: The weekly snapshots of our plan can be downloaded from the website as the pdf file named "Weekly Snapshots"

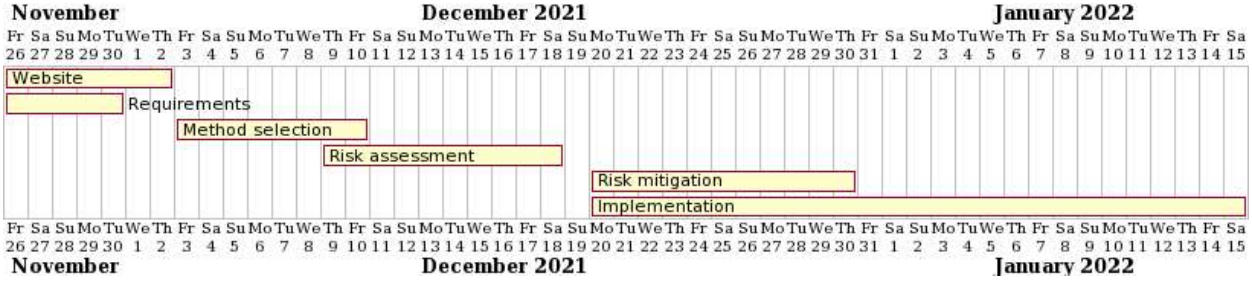

Fig. 2, an initial Gantt chart created at the end of November for the project's tasks

- The schedule for the project was identified within the first two weeks of the project starting, as we were able to look at the project scope, get to grips with the library that we used and thus be able to construct a realistic timeframe for the project.
- The overarching task priorities were to create a suitable architecture for the project, and then be able to implement it as effectively as possible. This meant that even though the actual implementation phase only began on 20/12/21, the team experimented with and learnt the development framework, re-acclimated themselves with the Java environment for weeks beforehand.
- However, the main week-to-week priorities corresponded to the contents of the Gantt chart; it was only when each member had some additional time, they engaged with the software tools described above.
- The layout of the Gantt chart implicitly shows the dependencies that the team decided on during the project.
    - Method selection is dependent on requirements
    - Risk assessment is dependent on method selection
    - Implementation and risk mitigation are both dependent on the risk assessment.
- In addition, the team created weekly agenda tables which mapped the larger goals to a set of smaller subtasks that were designed to be completed for that week.
- This form of short term planning helped to immediately identify if a task was going to be delayed due to unforeseen difficulties and allowed problems to be resolved and keep the project on track.
- The initial plan was created by consulting the JSD diagram that we created and by having an in-depth discussion and reading into the questions asked by the assignment and then creating a plan based on our initial estimations.
- However, the plan did change throughout the lifecycle of the project, as while undertaking each of the subtasks we were able to have a better understanding of dependencies we had not considered earlier, or dependencies we incorrectly assumed would exist.
- An example of this is much of the assumed dependencies of risk assessment and risk mitigation. Ideally, the risk mitigation phase could have occurred earlier and before the implementation phase rather than concurrently, as the team ran into slight issues in the early stages of implementation as there were few risk mitigation strategies with regards to the programming aspect of things.

- For example, the team changed the proposed graphics library from JFrame to LibGDX around 1 week into the implementation phase - our initial plans were changed, and there were no real mitigation strategies developed for this risk we had identified, so we lost sometime early in the project.
- Another aspect that changed the rigid presuppositions regarding the planning was the fact that we would repeatedly discover new risks, identify new ways of mitigating these risks and alter our proposed architecture throughout the lifecycle of the project.
- This meant that while we largely used the waterfall approach, we would periodically review and make changes to previous stages of the project. This meant that throughout the middle and later stages of the project, we would work with the mindset that the existing plan was a good guide, but if we found a possible better approach, we would pause, discuss with each other, then alter the plan going forward (and making whatever necessary changes to previously written documentation).
- This was a slight deviation from the exclusively sequential [2] waterfall approach, but it was an effective change that reflected the higher level of adaptability of our small team.
- One aspect of this was the coin game mechanics. The initial idea was for the player to contend with a timer, but to make the coin collection game mechanics more coherent with the combat mechanics, we decided to make the player fire at the islands to release coins.
- This changed our overall plan quite late in the development cycle, but we were able to consult the risk mitigation table and evaluate that it would be an effective addition and without adding much risk, as this feature's implementation was a simple extension of other tools in LibGDX we were already very familiar with.

Bibliography

[1] M. A. Jackson, *System Development.* Prentice Hall, 1983
[2] K. Peterson, *Waterfall Model in Large Scale Development.* Springer, 2009
[3] A. Sondra and R. Kristin, *Introduction to Agile Methods.* Addison-Wesley, 2014