Method Section and Planning

Outline of software engineering methods

We followed a spiral model approach for the planning, coding and analysis of the project. This particular model was chosen due to the high volatility of change that needed to occur, as we first brainstormed a basic plan, event ideas and risk analysis based on the product brief we received. However, we knew that we needed to hold a stakeholder interview where we would need to change multiple requirements to satisfy the needs of the stakeholder. Therefore, the spiral method approach that focuses on incrementing and flexibility is extremely important, since we were able to refine our plan and ideas to better fit the stakeholder's vision. Allowing us to conduct a more in-depth risk analysis and delegate the tasks more effectively, as we had a clear understanding of what we should do next. Another advantage of the spiral method is the focus on prototypes and testing, as this allowed us to create working software which was much easier to show in group meetings compared to lines of text. For example, a prototype of the events was created, which was useful as the visual demo helped us to refine a better list of events. This was because some events felt underwhelming in the demo, so we brainstormed better alternatives or enhancements, as we were able to better identify the positives and negatives of each event. Moreover, the frequent demonstrations helped test the code, as all functionality was clearly displayed in each demo to see if the game worked as intended.

An alternative to the spiral model that we considered was the waterfall model. However, we decided not to choose it, as the waterfall approach is quite inflexible, which was problematic since we were already expecting a large amount of modifications to aspects such as the planning and requirements due to the stakeholder interview. Moreover, the heavy focus on the predecessor task could cause quality issues. For example, if one of the stages was completed poorly, such as missing some steps in the plan, then the subsequent stages might also suffer in quality due to the influence of the previous tasks, such as missing a key requirement in the game. In addition, we are required to submit a prototype of the game. However, in the waterfall approach, working software is only built in the late stages of the development cycle, meaning that if there are any unforeseen circumstances, then we may not have enough time to develop and test the prototype, leading to an incomplete game with many bugs or missing elements. Furthermore, there could be an inefficient allocation of resources, as each stage often needs to be finished, or nearly finished, before moving on to the next stage. This could result in some group members working in areas that they are less skilled at, or having to wait until the other group members have finished their tasks to complete the stage in the waterfall model before we can begin the subsequent task.

We used the agile software development methodology, as it is the most effective for our team due to the focus on flexibility that the method has, which is extremely important since it allows us to plan and create effective tasks, enabling us to modify our plans and documentation. This is crucial, as it has a high synergy with the spiral approach that we are using, and does not conflict with advantages such as the ability to quickly adapt our plans and tasks. Another important aspect of agile development is the ability to clearly delegate and create accurate tasks that need to be completed. This is highly effective, as we held face-to-face discussions on Wednesday and Friday, where the work assigned to each member was based on our current situation and the preferences of each group member.

Moreover, it also assured that group members would not be repeating any work, as we only needed to complete the tasks that were assigned to us. Furthermore, if we finished our tasks early, then we could easily assist other team members, as we knew who was responsible for each task and who to approach for information regarding each task. A different methodology we considered was RUP; however, this contrasted with the flexibility of the spiral model, while the agile and spiral approach have better synergy together.

To facilitate the agile software development methodology, we chose to use the tool Jira, as it is a commonly used tool for agile development in the industry, ensuring that it has all the features necessary for us to properly use the agile software development methodology. For example, tasks were delegated via tickets, which easily allowed any group member to see what tasks each member was working on and the stages they were at. An alternative we considered was Trello; however, we decided to choose Jira instead due to its focus on sprints, which closely mirrors our workflow of delegating tasks to complete on a weekly basis.

Google Docs and Drive, as well as PlantUML, were chosen for the documentation and the creation of diagrams. Google Docs and Drive were specifically chosen due to their well-connected ecosystem that the team was familiar with, while containing important features such as real-time collaboration and cloud saving, allowing the team to create the needed documentation remotely on the same document at flexible times, or even simultaneously. A substitute we considered for Google Docs and Drive was Microsoft OneDrive and Word, but we decided against it due to potential syncing issues, where it either does not properly sync all the work done or takes a long time to sync the file when multiple people collaborate at once. PlantUML was selected for diagram creation due to its text-based UML generation, as it was much easier to modify compared to GUI-based tools. Another option we considered over PlantUML was Lucidchart; however, the limited free-tier options and the GUI-based UML creation made it more difficult to create and modify the UML diagrams than we expected, which contrasts with the spiral model approach we adopted that requires frequent incremental change to be made.

To develop the game, we used GitHub and LibGDX. We chose to use GitHub because it is a version control system that would allow us to develop the same piece of code at flexible times and ensure everyone had access to the latest version. In addition, the entire team was already familiar with GitHub, so we could save some time as we did not need to learn a new tool. An alternative we considered was BitBucket, as it is owned by the same company that owns Jira; however, it has a limited free tier plan of 5 members per private repository, which is not enough so we chose GitHub instead. LibGDX was picked, as it has a large, active community with plenty of documentation, making it easier to use and learn compared to other, less documented and known game engines. A different game engine we considered was jMonkeyEngine; however, we decided against it as it primarily focuses on 3D games, but we are creating a 2D game. This means that there may be useful features not implemented in jMonkeyEngine. Furthermore, LibGDX will be more optimised for 2D games, hence improving the gaming experience due to faster load times.

Approach to organisation

We decided to split the team group members into main roles and sub-roles, where each team member would have at least one main role and two sub-roles per week. The main role corresponds to a specific type of task assigned on Jira, with the group members in this role being the primary contributors of the task. They are also responsible for updating the team on the progress of the task and its completion. This system of team organisation was specially chosen due to the high synergy it has with the Agile methodology, as it provides an effective way to delegate work in a task-based system. The sub-roles are distributed to group members who will assist the main contributor, either by providing specialist information about their tasks or helping them complete the tasks. This also highlighted another strength of the Agile methodology of flexibility, as if the completion of a task was taking too long or the task had a higher workload than anticipated, we could assign the sub-role or main role to more group members to accelerate its completion.

This organised role-based system was also extremely helpful in reducing risks such as team members being unable to attend meetings. This is because it was easy to associate specific types of tasks with certain group members, meaning if some group members could not attend a meeting, we immediately knew if there were any other group members capable of updating the group on the task. Coupled with the idea of multiple people collaborating on each task, it was highly unlikely for all the members working on a task to be unavailable on the same meeting day, allowing us to get an accurate and up-to-date briefing at each meeting.

The final aspect of team organisation was having group members do a quality check on the work completed and participate in frequent reviews. This was to try and spot any potential issues or generate new ideas from a fresh perspective, which aligns with the key aspects of the Agile methodology's focus on continuous feedback and the spiral model's approach of prototyping and refinement. These regular reviews ensured that the work we created was of a high standard and met the stakeholder's needs.

Project Plan and snapshots

During the first week of the project, we had a very relaxed plan, with every group member participating in every task due to the small number of tasks available for the week. We then held a discussion, and by reflecting on the tasks we did, we gained a pretty good understanding of each group member's preferences, strengths and weaknesses, which helped us identify which types of tasks would best fit each group member.

In the second week, we created a proper project plan where we transitioned to a more structured, role-based system, assigning group members responsibility for specific types of tasks. This led to a clearer understanding of task ownership, helping us to avoid the repetition of work that had occurred last week. We were also able to begin developing a plan for the code and documentation, as we now had a clearer understanding of the requirements we needed to meet based on the product brief.

By the third week of the project, we modified our plans a bit by further refining the roles given to each team member, adding sub-roles to support the completion of tasks as we had fallen behind schedule on some of the tasks. We began focusing more on the technical aspects, with two prototypes being built, as advised by the spiral model, where we built one general game prototype and one specifically for all event ideas made at the time. To ensure the quality of work under the new role system, we made sure to assign group members to do periodic checks on works that they were not actively working on, allowing for unbiased reviews to be done.

In the fourth week, we were able to catch up on work with the improved role-based system by reassigning group members who had completed tasks early. We also decided to continue having team members perform periodic checks on the work that they did not actively work on. Our focus then shifted to the final game prototype, including the creation of assets and required documentation, as some team members had to wait for the final prototype to begin development before they could proceed with their tasks. Furthermore, the previous prototypes we created allowed us to refine our current game concepts and ideas, so we now also had all the game concepts we wanted to implement in the final prototype.

By the fifth week, we decided to shift our focus on reviewing the entire project and conducting a quality check on all the work completed so far, to ensure a high standard of quality has been maintained. This was important, as while significant improvements had been made, the addition of an extra member to perform another quality check led to an even higher level of consistency, causing a reduction in the amount of editing that was normally made. Next we also assigned all the remaining tasks to every member and compiled a list to clearly outline the tasks that each person was responsible for during week 5 and the reading week. This is important as many group members were unavailable at certain times during the reading week. This was crucial as many team members were unavailable at various times during the reading week, and having a list helped ensure everyone knew what tasks were still to be completed.

During the reading week, we continued with the plan and focus of completing the assigned tasks, while also adding in the finishing touches and new modifications made for some aspects of the project. The development of the final prototype progressed, and new

implementation ideas emerged, requiring updates to both the prototype and the documentation. To accommodate these changes, we modified the plan to include frequent updates on task completion whenever significant progress was made, as we were unable to hold meetings during the read week.

In the sixth week, we continued following our plan by changing our approach to adding the final finishing touches and reviewing all the completed work to prepare for submission. This was essential to ensure that all the required information and diagrams were documented and ready for submission. We accomplished this by first completing any outstanding tasks remaining, followed by group members performing another quality check to ensure that all the completed work being submitted was of a high quality. This approach of double checking was effective, as it helped us to catch some small issues that we overlooked.