METHOD SELECTION AND PLANNING

ENG1 Team 7
Broken Designers

Adam Brown
Morgan Francis
Oliver Johnstone
Shabari Jagadeeswaran
Laura Mata Le Bot
Rebecca Stone

We approached the software engineering aspect of this project by first having a group meeting in which we discussed the requirements, and began brainstorming how we would structure the classes in our project. We used the requirements we had elicited after a meeting with our customer to guide the creation of UML diagrams that represented all the classes present in the game. This helped us to begin programming the game as it gave a good starting point to build the “scaffolding” of the game - the general outline of everything included in the game, without necessarily writing the specific implementation of those parts as we went.

Building the rest of the game off of a skeleton made separating tasks among team members more manageable. One person could take one section or class of the game that needed to be implemented and write the relevant code. Their code could then be merged into the existing skeleton using a version control system.

This would have been a hard task to split up between multiple members of the team. Although the initial discussion took place in a meeting, everyone had slightly different ideas in their heads on how everything in the project was connected and how the requirements would get translated into a working code base. It would have also been tricky to merge separate parts together using version control or otherwise. Therefore, by giving the task of creating a skeleton of code that everyone could work off to one person ensured nothing was repeated.

Once multiple team members were working on the game simultaneously, collaboration was very important. Different aspects needed to be split among team members to allow equal collaboration and help us meet the deadline. Some of our coding was done during meeting times, allowing direct, in-person communication surrounding the implementation of code. Those with more experience could help teach others and we could also work together to solve uncertainty or issues with the game.

Pair programming is a software engineering method that involves two developers working together on one workstation. Not only did this help those who were less experienced to familiarise themselves with libGDX and other tools used in development, it also helped us get past minor hiccups where one person’s knowledge around a particular aspect of implementation was stronger than the other.

A downside to pair programming is that it may take more time - two people are working on one part of the game when their time could be split among two different tasks. For that reason, we mainly used pair programming when there was a part of the game that was difficult to code and the pair could discuss the issues they were facing. It was especially useful when there were fewer programming tasks currently in progress than there were people available to work on them, and this suited the team well.

As we completed sections of the project individually, we found it both helpful and important to update each other on the progress we’d made. One way in which this helped is that it mitigated the risk of a team member becoming unavailable as we would be more able to take over from them. It also let others build off any code we’d written or changed. For example, once the skeleton of the game was written, we had a meeting to go through the project in its current state and ensure everyone understood the code so that they could contribute to it. This was done during an online meeting, so the use of screen sharing helped us to explain while going through each section. Discord was used to achieve this, which worked well for the team as it was something we were all familiar with. It worked well for the project as we could keep track of important messages that had been sent and keep meeting notes in one easily accessible place to refer back to as needed.

Outside of pair programming and presenting our work to the group, we were all working on separate parts of the project on our own local copies of the game code. In order to combine our work together, we used git as our version control system. This allowed us to pull the project from github, make any changes required locally and then push them to the main file. Without version control, we would have had to manually copy changes made on one machine over to all the others. One small discrepancy could result in errors on one person's computer while the game runs fine on another.

Another benefit to using a version control system is that previous versions of the game can be accessed, and branches can be made that expand on the current version of the game. If the game stops working during implementation of a new feature, a previous, working version of the game can be restored.

When choosing a version control system, we considered both git and SVN. Git is a distributed version control system, meaning contributors work independently on local copies of the product and commit any changes made with the central repository. On the other hand, SVN is a centralised version control system. While it is easier to learn and contribute to, it doesn't allow us to work offline on our own workstations, which was most beneficial to the team and the way we were able to work. SVN also allows strict hierarchies in terms of security and access to the project, but this wasn't necessary for our project. Therefore, git was the best choice for our team and the project.

For the documentation sections of the project, where essays were split among team members, we found google docs and google drive a useful tool to aid teamwork, especially over long distance. It allowed us to simultaneously work on the documents, and add comments on sections that could be addressed by the other team members working on the document. Another way this was helpful was in the final checks; we proof-read each other's essays and gave feedback via comments that the original author could consider. We also considered writing reports on our own text editors and sharing them once complete. However, this wouldn't have allowed as much co-operation and the files wouldn't have been easily accessible from the same place. Therefore, google drive and google docs were a good fit for our team and the project.

We approached team organisation by discussing the current plan for the project and what we aimed to achieve in the next week at the end of each meeting. There were very few strict deadlines imposed by the project itself, placing the responsibility on us. We decided to set flexible deadlines so that we could more easily keep track of what everyone was working on and what still needed to be done, however we thought that individuals could decide when to do their set piece of work when it best suited them, as long as they weren't leaving everything to the last minute. Regular check-ups during meetings ensured we were all getting on okay.

There were three main ways that we planned the work for the project. These were the gantt chart, the website Trello and a list of work to be done and by who, included in the logbook for each meeting. While Trello and the gantt chart were the primary representations for our plan, the logbook was useful for making sure everyone was on the same page and knew what was going on. It also incentivised everyone to contribute to the project.

The gantt chart was useful for giving an overview of the project and all of the elements that needed completing. It was a good visual representation of the jobs, who they were being done by, when they were being completed as well as the priority of those tasks. Updating it every week to put on the website allowed us to see how much progress had been made and evaluate how we might wish to change deadlines or priorities going forward.

Trello is a project planning website that allows groups to make boards that they can all access from their own workstations. Tasks can be set for specific people or groups of people, alongside start and end dates and the option for an email reminder when the set deadline is approaching. Comments can be added to the tasks to allow communication between group members.

We had two Trello boards, one for general project management and planning and one for work that needed to be done on the game implementation itself. We split tasks into four categories: "Future", "To-do", "In-progress" and "Complete", assigning a different colour to each of the deliverables so it was clear how each task fit into the project as a whole. The main downfall of Trello was that it didn't show dependencies between tasks - for example the UML diagrams had to be created before work on the Architecture essays could begin. The best feature that gantt charts didn't allow was the ability to update the rest of the team on progress that one member had made as soon as it was done. It was also an especially useful tool for working over the Christmas break; the lack of regular meetings made it harder to communicate about work that was in progress and needed doing.

Completing this project as a team of 6 required us to split up the work into sections that could be tackled by individual team members in between our meetings. We could keep track of which tasks were in progress or had been completed via Trello or our weekly gantt charts as needed. The separation of various elements of the project was necessary for our team to allow us to complete work when we weren't all together. It also suited the project well because members were able to research their assigned section of the project and specialise in that area, feeding back their findings to the team and resulting in a higher quality end product.

Weekly gantt chart snapshots are available at:
https://ajbrown-york.github.io/html/gantt_chart.html

At the beginning of the project, we weren't entirely sure how we might all fit roles in the team and split work between us based on our strengths and weaknesses. Therefore, the plan was short term for the first week - we knew we had a strict deadline for the requirements gathering meeting with our customer. As the rest of the work was dependent on this meeting and having a general understanding of what was required from us, the plan outlined these important tasks up to the meeting date, with the idea that we would have a meeting to split the rest of the tasks.

When putting together the initial gantt chart that encompassed the entire project, we separated the tasks into the sections outlined in the assessment document. This made the gantt charts clearer to look at and helped find, add and edit the gantt chart in the future iterations of it. Our gantt chart used colour coding to specify the priority for each task (green for low priority, orange for medium priority and high priority). At the start of the project, there were only a few high priority tasks, mainly due to strict deadlines, such as any preparation required for the interview. As the project progressed, more tasks were given a high priority ranking to ensure we would be able to finish everything in the given time frame.

Priorities allowed us to represent flexible and more strict deadlines on the chart and prioritise certain tasks over others. For example, between the plans for autumn term week 3 and spring term week 2 there is a very noticeable difference. Autumn week three contains loose deadlines for the system justification essay in the Architecture section. After beginning work on the game and looking into what was expected from us for this section, we realised that we would need to make more progress into the game before we could adequately write about its architecture and any changes we had to make to it. We also swapped the person responsible for finishing this essay to more evenly distribute the work we had left to do.

Various sections of the project were pushed back during this time frame due to the Christmas break. We planned around this by setting long term, low priority deadlines that allowed team members to work as much or as little as they were able to over Christmas. One example of this is the method selection and planning section of the gantt chart; the flexible deadline was pushed back and the priority increased between autumn and spring term.

We had to allow time for unforeseen circumstances, so that we would still have time to finish if anything went wrong. One of our team members' laptops had issues that meant they were unable to bring it to practical sessions and meetings. To work around this, we didn't assign them any of the implementation tasks when we were distributing the rest of the work at the beginning of the spring term; their efforts were focussed on the documentation sections of the project instead.

There was a delay in the merging of code that had been written over Christmas. This was a very important part of the project, as the gantt chart shows that it has multiple other tasks dependent on it. While initially planned for week 2 of spring term, it was pushed back until the following week. The implementation of tasks dependent on the merge could be researched by the team members responsible, however they couldn't yet push their code to the repository. Therefore, a large part of the project had to be pushed back towards the end of the project. This took away the flexibility of our initial deadlines and meant that anything going wrong in the last week may lead to the inability to meet all of the specified requirements.

Due to a difference in expertise and experience, the task of implementing the main menu of the game got transferred to a different team member. This was a significant inconvenience as the main menu had been left to the last week of the project. Luckily, we were ahead of schedule enough that we could manage despite the setback.