Method Selection and

Planning

Group 28

Piazza Panic

By OuseWorks

Ben Harris
Joshua Gill
Niamh Hanratty
Amy Raymond
Matthew Czyzewski
Matt Rohatynskyj

Part A
One of our main priorities has been collaboration, however, because we only have meetings once or twice a week, we need to do lots of work outside of our meetings. Therefore one of our main software engineering methods has been agile development. This means that during our meetings, we can focus mainly on discussing what each member has worked on over the past sprint (this could range from a number of days to a number of weeks) and prioritise what needs to be improved or finished and then assign tasks for the next sprint. Ben was our scrum master as the meeting chair, facilitating the weekly stand ups and progress development.

We considered using waterfall development, like devCharles, but decided against this as we were already familiar with agile development and the methods such as scrum. We also felt isolating phases of the software development was restrictive in the project.

LibGDX [1]
The Java framework we decided to use was libGDX. We also considered a few others - JmonkeyEngine, LWJGL, GODOT.
libGDX was the most:

- Mature
- Widely Used
- Extensible

LWGJL was another option, but libGDX uses it in its framework and is a higher level method of accessing it. LibGDX also consists of mainly regular Java, which the team was already familiar with.

VSCode [2] and IntelliJ [3]
For our IDEs we decided to use IntelliJ and Visual Studio Code. Visual Studio Code because:

- Everyone in the team used this IDE
- Its live share feature was incredibly convenient for collaborating together during meetings even whilst not physically together

IntelliJ was also used as IntelliJ is developed specifically for Java and so had some features more useful for this Java project. We considered using Eclipse however as the team was not familiar with it, we decided against it. Since GitHub allows for different IDEs to be used easily, we believe that both ideas would be best used in tandem.

Tiled [4]
Tiled was used as a map creator for libGDX as it is the most supported map editor so this was chosen due to the fact it had more tutorials and was a simpler tool then any competing product.

GitHub [5]
We then decided that we needed a version control system to upload code and enable every member of the team working on implementation to be capable of developing the

code simultaneously.

We decided on GitHub for this purpose.

- Easy design
- Its high market share means that the team could have experience with real world tools required in software development.
- We could sign up for free (as we are students) to GitHub premium. This allowed us to have web based support in case of any issues we encountered.
- Crucial features like being able to have protected branches and having required reviewers, reducing the chance of losing changes or introducing merge conflicts . This reduced our risk of:
    - R4: A file is accidentally deleted or corrupted
    - R11: A group member accidentally commits directly to main
- Tracking of who and when team members create changes to their work, which allows for a simple way to track which members have contributed individual parts of the code.

PlantUML [6]
To create gantt charts, we used the google doc extension of PlantUML as this was easier to do on a smaller scale (i.e. one team member can do it). It was easier to duplicate the gantt charts on google docs, to show the evolution of the plan every week, which we regularly looked over to see what to improve and do next during our project. This worked well for the Waterfall development method as everyone had access to the gantt chart so they knew what they would need to complete at every point in the project before group meetings or deadlines.

An alternative to PlantUML was Lucidchart, however this requires a subscription to be able to collaborate on it and to use some of the features. After testing PlantUML after our lecturer recommended it, we decided that it was easier to use, and there isn't a limit for free UML diagrams, unlike with Lucidchart.

GitHub also has a Kanban Board management system we utilised (work items are shown as cards, which are moved left to right to show progress)due to the team's now frequent usage of GitHub (and therefore familiarity).

Discord [7]
Because we planned to have regular meetings, we required a tool to communicate. We discussed several different platforms:

- Slack - some members of the team were unfamiliar with slack and slack had some features barred unless we used a premium service we decided against it.
- Whatsapp - less tools for communication.
- Discord

After some discussion we agreed on discord.
Discord allows for:

- Different channels to be used for different topics and whatsapp would only allow for one group chat.
- Sharing of desktop screens allows us to have much more productive calls with significantly more collaboration.
- Voice calls for meetings .
- Messaging for organisation both to individuals and the group as a whole.

Overall we decided on Discord due to these reasons and since everyone already had a discord account, so most of the team thought it was the best option

Because of discord we were able to have regular meetings - towards the beginning this was around twice a week and closer to the deadline this became multiple times a week - both in person and online to:

- Discuss everyone's progress
- Adjust the plan based on this
- Set goals for the next meeting

We would do most of the work together during the meetings but some of our work was done independently.

Google Docs [8]
In order to complete our documentation we decided it would be best to use software such as Microsoft Word. Since every individual in the team had extensive experience with Word we heavily considered using it. We preferred google docs for several reasons:

- Sharing documents was easier
- Live collaboration was simpler
- Our team had more experience with google docs

So whilst it is less capable than word it has significantly better collaboration capabilities. This made it fit with our engineering methods very well which ultimately made us choose this platform over Microsoft's.

Part B

Ouseworks' Roles:

Within our team, we decided to give everyone roles to stay organised. We found this was an important step to do as we knew the shadow roles in particular would play a significant role in our risk assessment document and would increase the bus factor of our project. The roles we came up with are as follows:

Meeting Chair:	Ben Harris/Joshua Gill
Secretary:	Niamh Hanratty / Amy Raymond (shadow role)
Librarian:	Matthew Czyzewski / Ben Harris
Report editor:	Amy Raymond / Niamh Hanratty (shadow role)
Dynamic Shadow Role:	Matt Rohatynskyj and Joshua Gill

Having a meeting chair is appropriate for this project as they will help keep the meetings organised by making sure we don't get off-topic and stay on track with timings for that meeting. The secretary's role is to document what goes on in the meetings and make notes so that if people are absent, which is likely to happen during this project, they will know what they missed. The librarian will provide and document most of the resources we will use throughout the project. The report editor's job is to keep the documentation in the shared drive organised and to keep the deliverables cohesive and consistent. They can also oversee the formatting of the deliverables, which is a requirement for this project. We then have a shadow role for each of these roles, which will oversee the role's tasks in case the assigned team member is absent or falling behind. We also have two dynamic shadow roles who oversee everyone's tasks and pick up missing or behind work. This approach is appropriate for our team as it is likely we will have absences during this project as peoples' schedules clash, so this is a way to stay organised despite having these setbacks.

This method also increases the bus factor, because we would then have multiple people working on each section at once so progression would be as consistent as possible.

We believe that our method of organisation was effective as. The shadow roles particularly worked for when some team members couldn't make it to the meeting. We also tailored it to work for our specific team members' skill levels rather than trying to give tasks to people who are best suited for another part of the project.

Part C
GitHub was used throughout the project, but became the main way to manage tasks when the Gantt chart became less useful for doing tasks as quickly as possible.

The benefits to this method were:

- It was easy to get a grasp on how far people had gotten in work.
- Simple proof-reading
- Easy to raise an issue for help with bug fixing.
- Multiple people can be assigned to tasks both at the start and any time during the progress.
- An integrated method of seeing which members contributed to or completed which tasks.

A plausible downside to this method is the lack of an included given timeframe for each task. This could cause issues in large projects over long periods of time, where each task could have many dependencies.

OuseWorks' Weekly Plan for Assessment 2

We used GitHub Projects to help us progress through the weeks, so this immensely helped us create our weekly plan. However, we use it slightly differently to devCharles.

- devCharles created tasks when they found them and then anyone could pick them up when they had time.

What we did:

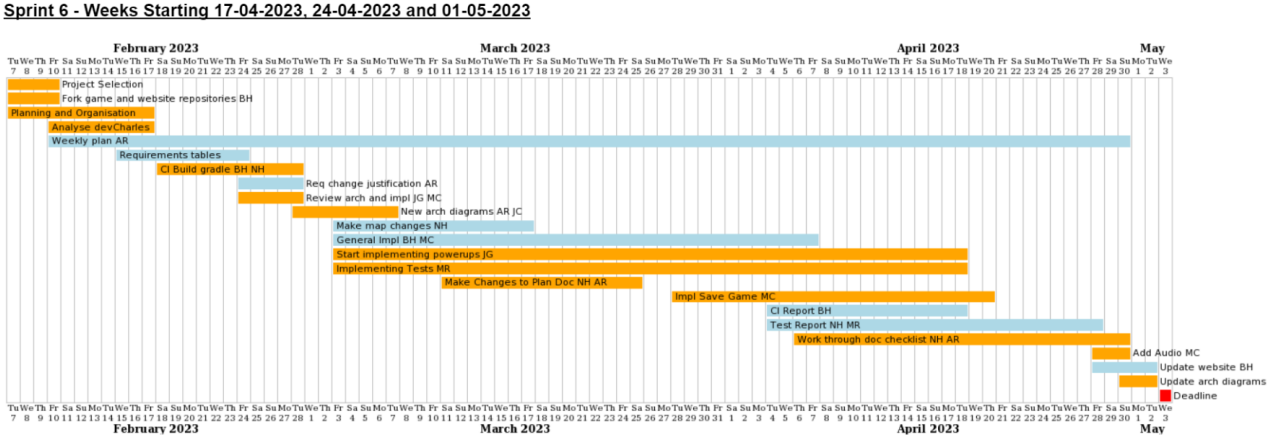
- We would discuss during our meetings if any of the tasks had been completed during that portion of the sprint and then anyone can pitch new tasks during the meeting. Together, we assign the tasks to the people who have the best skill set for that particular task.

Why we did it:

- We can make sure everyone has the same amount of workload and is putting in the same amount of effort. It also gave us the opportunity to see which tasks are falling behind/never getting started and then we could make changes or assign another person to help with that task.

Ouseworks' GitHub Projects
Ouseworks' Weekly plans

Ouseworks' final plan:



Key: Deadlines Deliverables Other
Assume everyone has worked on each task, unless specific initials next to tasks.

Changes and Justification to the Plan during Project:

Sprint 1:
Sprint 1 started off as us discussing the project and taking time to look through devCharles' repositories and documents. We started our weekly plan and in the meetings we discussed what each of us should start working on. We edited devCharles' requirements first using the new product brief as we thought this was a necessary first step as the rest of the project depended on this. Our main priorities were to outline the basic plan for assessment 2.

Sprint 2:
Our main priorities changed in the second sprint as this sprint focused more on the CI Build Gradel and finishing the justification for the requirements. A couple of members of our team started redoing architecture diagrams and making 2 new diagrams, while the others focussed on planning the implementation. One priority was to brainstorm what our 3 new requirements should be and how we could start implementing them. One dependency we had was to add tests for existing code before we started the implementation.

Sprint 3:
In sprint 3 we finalised what our power-ups would be and started the implementation of these. We started on general implementation of the new brief as most tests for existing code were finished.

Sprint 4:
In sprint 4, We considered adding audio and made sure we made the menu screen before we started implementing the second gamemode which included starting the implementation of the code to save the game in and making the stations unlockable and interactive. We continued making changes to the documentation. We focused on the

plan document this week before writing justifications for our changes in the change2 document.

Sprint 5:
In sprint 5, we started the CI report and implemented other features such as adding money, game loss HUD and animating cooking stations whilst in use. We decided to ask if we were allowed access to devCharles' feedback, and then when we realised we were allowed, we analysed it and it helped us make our changes to the documentation. We then used this feedback to write a checklist to help us make progress with the change2 document.

Sprint 6:
We focused on finishing the Test, CI and Change report and relevant associated documentation in regards to the Change Document. Final changes were made to implementation and then we made the last minute architecture changes so that our diagrams were consistent with the code. We then had to add all our documents and supporting diagrams to the website as the final step, adding links to diagrams to relevant documents.

References:

[1] *LibGDX* https://libgdx.com/

[2] *VSCode* https://code.visualstudio.com/

[3] *IntelliJ* https://www.jetbrains.com/idea/download/#section=windows

[4] *Tiled* https://www.mapeditor.org/

[5] *GitHub* https://github.com/

[6] *PlantUML* https://plantuml.com/

[7] *Discord* https://discord.com/

[8] *Google Docs* https://www.google.co.uk/docs/about/