ENG1 Group 2 Marlin Studios
Part 2

Method Selection and Planning

SKYE FULLER

JONATHAN HAYTER

ALEXANDER MIKHEEV

MARKS POLAKOVS

FREDERICK SAVILL

HAOWEN ZHENG

Introduction:

This document will outline the software engineering methodologies and tools used throughout the project, in addition to the efforts made in organisation and planning.

Development and Collaboration Tools:

The following tools were utilised throughout the project to aid in collaboration and development:

- **Discord**
  This was used by all our team members as the primary method of communication – meetings, discussions and sharing of small resources was all performed using this platform.
  We heavily utilised features such as screen sharing and voice channels to collaborate and discuss in real time during development.
- **GitHub**
  Centralised place for all code – allows everyone in our team to view and contribute to the development of the project (source code). Branches and versioning were utilised heavily to ensure collaborative efforts were constructive and not destructive. GitHub Desktop was also used to minimise teaching necessary to those unfamiliar with the command line utility. *In the second part of the project, we added GitHub Actions to automatically run tests, calculate code coverage, and generate executable JARs and javadocs.*
- **Tiled Map Editor**
  Map editor designed specifically for development of video game maps. Free and easy to use, allows the use of efficient editing tools to minimise development time such as stamp, fill, and layers.
- **Libgdx**
  Cross platform Java game development framework. Well documented used for game design specifically.
- **Java**
  Object oriented programming language used for the project. Easily made cross platform and easy to develop with, alongside compatibility with libgdx made this ideal for our project.
- **Neovim / Visual Studio Code**
  Development tools used to write and run the project's source code, using Gradle to build the project. Offers features such as syntax highlighting, folder organisation, debugging tools and other programming specific features which make development faster compared to a traditional text editor.
- **Google Suite (Drive, Docs, Slides, Gmail, *Sheets*)**
  Google applications such as Google Docs, Drive and Gmail were used for file sharing and development of documentation. Allows for real time collaboration on documents meaning files could be stored in the cloud and accessed by any member of the team. *In the second part of the project, we continued to use Google applications, with the addition of Google sheets to create the Gantt chart for our systematic project plan*
- **Adobe Suite**
  Adobe applications such as Photoshop and Illustrator were used to develop and edit graphics for the game, such as sprites, logos, and other visual matter.
- **Paint.net**
  In part two of the project, the team used Paint.net to create new graphics, which were added to finish the game. Paint.net was the graphic design tool that the new team members were most experienced with, and thus was the clear choice.
- **Microsoft Powerpoint**
  Microsoft Powerpoint was the tool used to create the Gantt chart for the systematic project plan, in the first part of the project. Powerpoint provides functionality "add-ons" and Gantt chart templates.
- **Zoom**
  In part two of the project, the team used Zoom as the primary method of communication; the platform over which all meetings took place. Free zoom accounts were provided by the University and ENG1 practicals already took place over Zoom, therefore it was the obvious choice.
- **WhatsApp**
  In part two of the project, a group chat on WhatsApp was a secondary tool for communication. This was used mainly to schedule Zoom meetings, and stay connected on progress made between meetings.
- **IntelliJ IDE**

In part two of the project, the team used IntelliJ IDE to develop the project code. IntelliJ was the IDE the team used in the first part of the project, and the recommended IDE in the LibGDX documentation. IntelliJ is easy to install, and provides efficiency and collaboration benefits the team made use of.

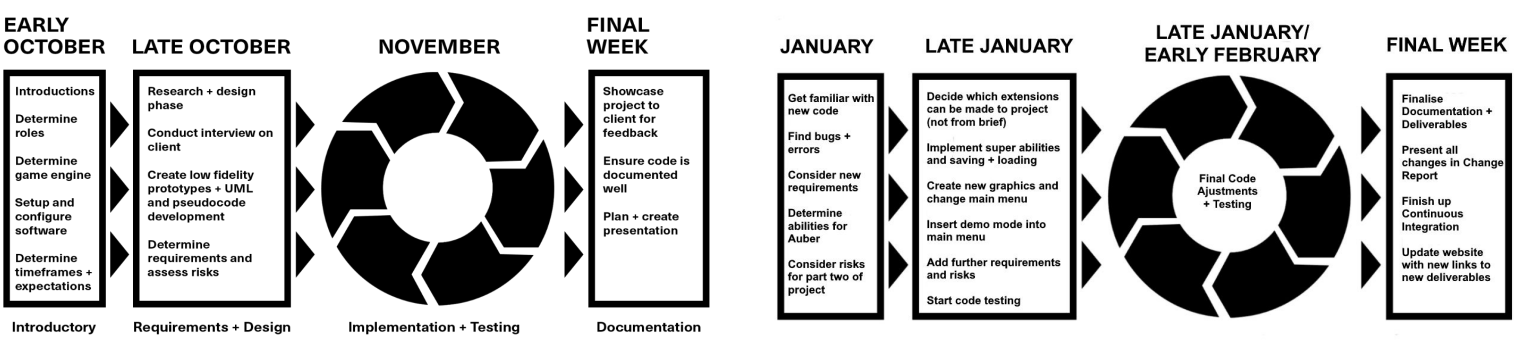
- **gdx-testing**
  In part two of the project, we added the GdxTestRunner from the open source gdx-testing project to emulate a headless environment as we had trouble setting it up ourselves (discussed in Testing2)
- **Coveralls and JaCoCo**
  In part two of the project, we added Coveralls and JaCoCo to track code coverage (discussed in Testing2).
- **GitHubPages + Jekyll**
  In part two of the project, we used GitHub Pages to host our website, in addition with Jekyll as it would build automatically and offered a wide range of themes and functionality

Before deciding on the above tools and software, we also considered alternatives:

- **Unity**
  Allows for development of top down 2D games and is also well documented, but for the scale and complexity of our project this would have been overkill (examples include 3D models and lighting engines, which we would not be using in the project). In addition, this engine uses C# which not all our team are familiar with.
- **Microsoft Teams/Zoom**
  Other software for communication were also considered early in the project's development but features such as text channels for discussions and voice calls without a URL make Discord more suited to purpose. Discord also offers screenshare which is easy to use and easily accessible.
- **Lwjgl**
  Another game focused library for Java – however we chose libgdx over this as it offers functions necessary for our project that lwjgl does not. This saves us time in development as we can use pre-programmed and well tested code in our project.
- **Microsoft PowerPoint**
  The team initially considered using PowerPoint to design the Gantt Chart for the project plan. However, our team found that Google Slides had similar Gantt features, but was easier to add to our shared drive folder.
- **Bootstrap CSS**
  Jekyll was the simpler framework compared to Bootstrap CSS, and therefore we opted for Jekyll due to time constraints.

Software Engineering Methodologies:

Our team worked in a scrum methodology throughout the development process – this involved weekly sprints where work was completed, and then discussions and new tasks were assigned at the end of each mini-sprint. The length of the sprints was limited due to the timeframe given to us for the project. This helped us to keep on track with our progress, communicate more often, and support others working on different parts of the project. In addition, the flexibility of this approach allowed us to reorganise and succeed when behind on certain timeframes/deadlines internally.



Team organisation approach

Team organisation was a strong subject in our regular weekly meetings, which involved progress updates, new assignments, and task delegation. The main idea behind our team organisation approach was to delegate tasks to the strengths of each member, by assigning them to their strongest section. Nonetheless we realise this is an educational project and therefore we still wanted to give members the opportunity to work on skills they are less familiar with.

To counteract the potential reduction in productivity when taking this approach, we often paired less experienced members with a more experienced member in the same sub-team; the most experienced member would be the leader of that section. The section leader would take the largest portion of the work and would be responsible for organising the sub-team, ensuring that deadlines were met. This approach maintained efficiency whilst also providing an educational benefit.

We regularly had over two meetings a week, therefore we could quickly identify any issues with delegated assignments, and keep up to date with the progress made. We never assigned a single member to a specific task; this would increase pressure on the individual and result in slower task completion.

When larger project issues formed within a sub-group, the issue would be reported at the next group meeting, and thus the whole team could contribute to a solution. A solution often involved team switching or simply an extra opinion. Critiquing and analysing each other's work was another measure taken, both for code and documentation; this ensured the highest quality was produced, benefiting from ideas of other group members.

Within our Discord server, we had different channels allocated to discrete sections of our project (Fig. 1); these proved valuable as it allowed sub-teams to focus on their particular assignment. To maintain work-life balance, we also opted to include a #memes channel which allows for non-project related matters to be posted, helping to build a bond early in the project. We also had additional channels such as #general, #important-to-remember and #licensing for other miscellaneous topics and discussions during calls.

In part two of the project, through the WhatsApp group, team leaders would enquire and chase up on uncompleted assignments. Likewise, private calls within sub-teams would be arranged over WhatsApp.

During the meetings, we also referred to the six thinking hats thought of by Edward De Bono (Fig. 2). These allowed us to keep the discussion focused, gather useful information from other team members, and logically consider if certain proposals are feasible. Using this system ensures that we make the 2-hour timeframe of our usual weekly meeting as efficient and to the point as possible.

The teams for the deliverables of the second part of project were as follows:

- **Url2** (Website):
  - ***Marks (leader), JJ***
- **Change2** (Change Report):
  - Introduction: ***JJ (leader), Alex***
  - Requirements: ***Alex (leader), JJ***
  - Planning: ***Alex (leader), Freddie***
  - Risk Assessment: ***Alex (leader), JJ***
  - Architecture: ***JJ (leader), Marks***
- **Impl2** (Implementation):
  - New features: ***JJ (leader), Marks***
  - Changed Menu Screen: ***JJ (leader), Alex***
  - Minimap: ***JJ (leader), Marks***
- **Test2** (Testing):
  - ***Marks (leader), JJ***
- **CI2** (Continuous Integration):
  - ***Marks (leader), JJ***

Fig. 2

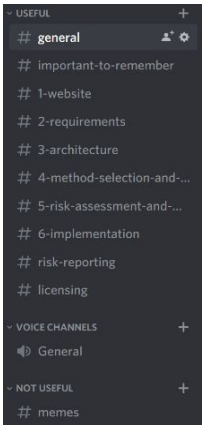


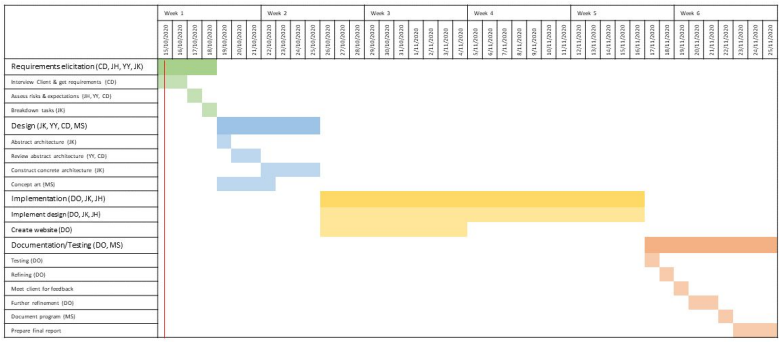
Fig. 1

Systematic Project Plan

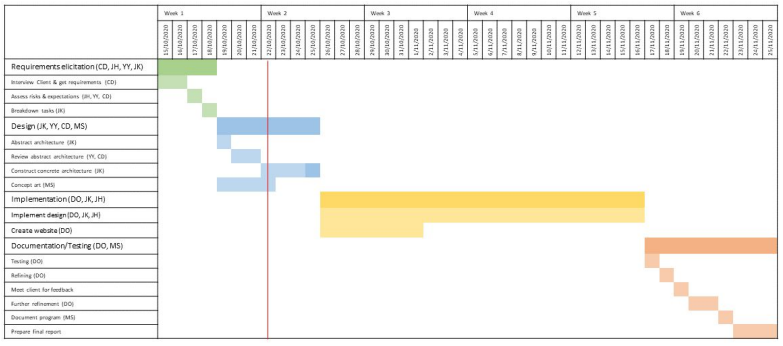
The critical path for this first part of the project was: Requirements → Design → Implementation → Documentation → Testing/Feedback. This approach was taken because each section depends on the completion of its predecessor.
In the second part of the project the critical path was: Risks → Requirements → Testing → Implementation → Documentation. Likewise, each section depended on the completion of the predecessor, however we make a note that testing before implementation may seem unexpected, but this approach was taken because after the elicitation of new requirements, test coverage could be started on sections the group decided not to modify.

The gantt chart was the model recommended in ENG1 Lecture 4; it is clear, and easily updated. The implementation section took the longest time at around three weeks, whereas requirements, design, and testing were planned to take shy of a week each. As the project went on, all sections had taken slightly longer than initially anticipated, however this was not a major issue as we were prepared for minor delays. During the implementation stage, we anticipated a more significant delay however still achieved completion within the original timeframe. Overall, our original prediction was fairly accurate. Shown below are the Gantt charts for the six weeks of our project:

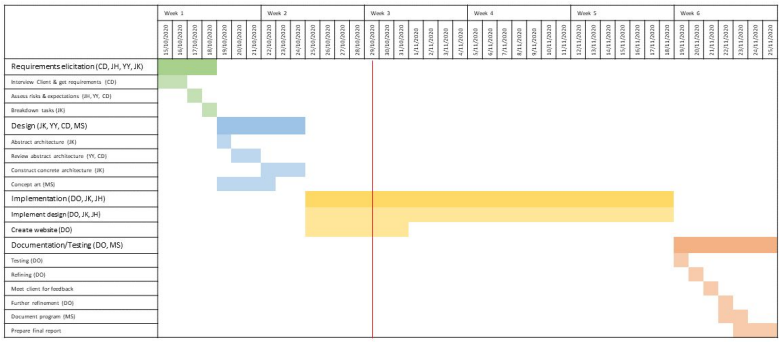
Week 1 (15/10/20 – 21/10/20



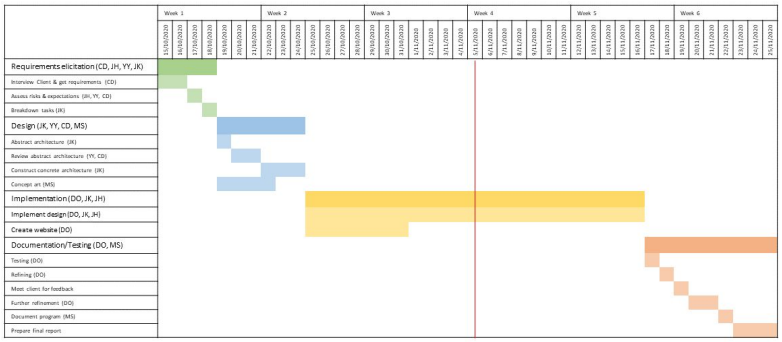
Week 2 (22/10/20 – 28/10/20)



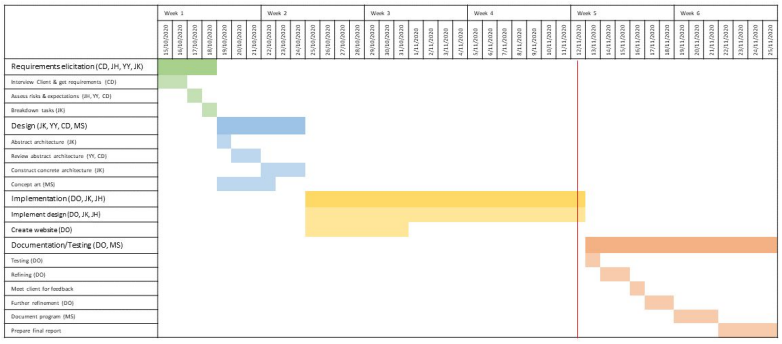
Week 3 (29/10/20 – 4/11/20



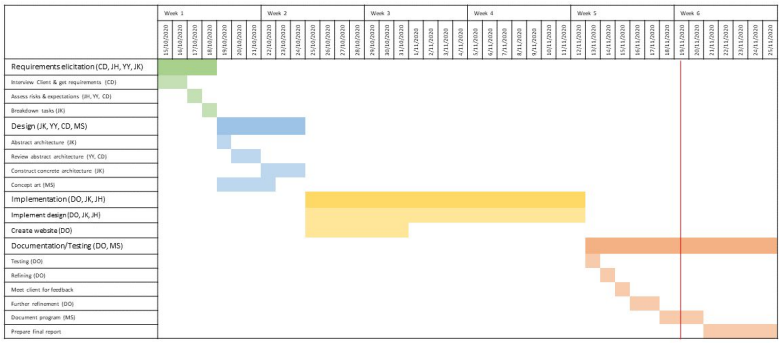
Week 4 (5/11/20 – 11/11/20)



Week 5 (12/11/20 – 18/11/20)

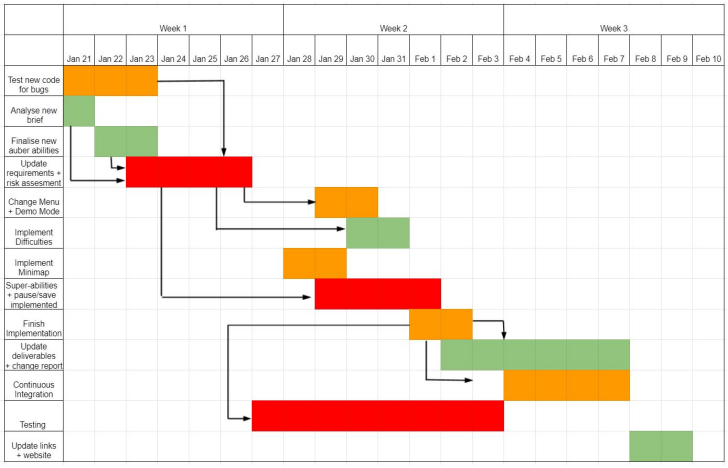


Week 6 (19/11/20 – 25/11/20)



For the second part of the project, after our original plan was drafted, we only made one update due to the short duration of the project. In the updated plan we added a slight increase to implementational task duration which we had previously slightly underestimated, and a fairly substantial increase to testing which the group faced an issue with.

Original:



Final: