Model and Method Selection

In keeping with a formal developmental process, subsequent work will need to be fit around an engineering model and carried forward by a supplementary methodology. The model will set "the order of the stages involved in software development" [9] and the overall criteria for moving between those stages [9], the method will detail exactly how a team can navigate through those stages [9]. An appropriate model was chosen before implementing and a suitable methodology was adapted around the rest of the project.

Research was made into a variety of models such as the waterfall [9] [10], spiral [9], V-model [10] [11] and evolutionary models [9] [12]; considerations were made on which of them would fit optimally around the project's scope and the upcoming commitments of the development team's members. Each model had benefits: the waterfall model wouldn't have required a great deal of planning to implement because of how linear it is [9] [10], whereas the evolutionary and spiral models cater quite well to changes in customers' requirements on account of their iterative components. [9] [12] However, in spite of their advantages, it was agreed that all of these traditional models were too rigid to take up, for a team of our size, experience and physical proximity, without compromising heavily on working efficiency.

The project's requirements are going to be altered as it progresses, given that many game-design decisions are typically made after stakeholders test prototypes - and being unable to respond to directional changes because of some rigorous engineering model's guidelines would severely impact upon the likelihood for a working, well-design game to be delivered within the project's time-constraints. Furthermore; while documentation will be needed to create reference points from which further work and decisions can be derived, most traditional models call for excessive amounts of documentation [9] [10] [11] [12] (whereby some concepts and designs may be unused in the final project) but pending design decisions can still be suitably informed with less documentation than what those models require.

As the project concerns the development of a game: unlike a strictly functional application, a game is assessed subjectively and will differ in perceived "quality" from person-to-person. The game that comes out of this project will only be as good as what the project's primary stakeholder deems it to be, so a significant part of the work that the project carries will be dictated by that stakeholder. Traditional models limit stakeholders' influences during design and implementation phases [9] [10] [11] [12]: hence, if one was to be adopted for this project, it would be more likely for our primary stakeholder's requests to be ignored and for precious time to be wasted on implementing unrequested features instead.

Rather than fitting the rest of the project around a traditional model, it would be more suitable to direct it ourselves by a set of principles with the aim of making it feel as efficient and as flexible as possible to get through. Hence, the next stage of the project will be fit around the **agile** philosophy, which is described in the figure that follows. This is a direct response to the traditional issues of software development that were raised in the last two paragraphs - as is exhibited by the "agile manifesto" [13] - along with enabling all of us to direct and work on the project in any manner that we or our primary stakeholder see fit, it should leave the door open for another team to pick up the project and run with it in whatever manner they may choose to.

AGILE MODEL FOR SOFTWARE DEVELOPMENT [13] [14]

Manifesto [13]	*Justification [14]*
● **Individuals and interactions** over processes and tools *This refers to adapting projects around frequent communications and decisions rather than arbitrary rigors* ● **Working software** over comprehensive documentation ● **Customer collaboration** over contract negotiation ● **Responding to change** over following a plan	● It encourages teams to drop formalities and do what they think would be the best for their respective projects… ● …and we can do this sensibly because we're able to communicate frequently, both in text and in person ● We don't want to restrict ourselves to specific roles and/or disciplines when some tasks may take priority over others at certain times ● The chances of our requirements changing over time is quite high, and we'd like to adapt to such changes without having to waste so much prior work and/or documentation ● The project's limited time-frame necessitates delivering a working game over writing excessive documentation ● We don't want to dictate the development practices that our successors

Once an agile model was chosen for the project's procedure, a methodology needed to be fit around it. Once again, numerous methodologies - including the extreme programming [15] [16], dynamic systems development [17] [18], feature-driven development [19] [20] and agile unified process methods [21] - were considered, and each had their appreciable advantages: XP's focus on responding to user stories is suitably client-centric enough to satisfy our obligations to the project's primary stakeholder [15] [16], whereas the DSD and FDD methods each encourage prototyping and testing to such an extent that could potentially allow for the project's requirements to be finalised quite early on [17]

[18] [19] [20]. In the end, though, the **scrum** methodology [22] was chosen for how well it can leverage frequent communication for productive benefit and for how it could enable the rest of the project to be fit around the team's shared university schedule and individual commitments.

SCRUM METHODOLOGY FOR AGILE MODEL [22] [23]

Description [22]	*Advantages [23]*	*Disadvantages*	*Justification*
- A backlog of tasks to be done is created and prioritised appropriately - The team takes some of those tasks and decides on how to complete them - A "sprint" (in which the team tries to complete their selected tasks) begins - Daily progress meetings take place during sprints - Sprints end with new product iterations being shipped, sprint reviews taking place and new sprints being planned - Sprints ensue until all	- Allows team members to remain flexible in completing different types of tasks - Priorities ensure that the most critical work is done first - Can quickly respond to the addition or modification of requirements by following additional iterations - Development can be fit around changing timings	- Additional requirements can spiral out of control - Places less emphasis on customer relations - Requires meetings to be held very frequently	The scrum methodology was designed to help small, tightly-knit teams of developers to power through projects, so it fits very comfortably around the context of this project. By meeting so regularly (which isn't an issue for us, given that we're undergraduate students who follow a shared timetable), critical decisions can be made more quickly and the work of individual team-members can be built directly off of one-another, thereby accelerating the pace of development. The main disadvantage of scrum - that being how poorly it reacts to drastic changes in customers' requirements - doesn't present such a big issue for this project because the project's core requirements have already been set in stone by a supplementary brief, thereby eliminating the risk of the team

LIST OF PROJECT RESOURCES

Task	Product	Use(s) in Project
Version Control	[GitHub](#)	Allows coding work to be reverted if stakeholder requirements change; prevents unauthorised changes to work (and requires the entire team to approve of any changes to the game's master copy); facilitates prototyping through branching.
File-Sharing		Repository acts as an online source for implementation work, enabling the entire team to access one-another's contributions flexibly and independently
	[Google Drive](#)	Acts as an online source for all non-implementation work and resources, including documentation and meeting records.
Documentation		Includes a web-hosted word-processor through which documentation can be accessed and edited collaboratively.
Burn-Down Analysis	[ZenHub](#)	Automatically measures task completions over sprints' durations and uses them to generate "burndown charts" showing whether or not sprints are on-track.
Task Management		Allows task-lists and sprint-lists to be logged directly within our chosen VCS and our game's repository, providing flexible access to those backlogs and assisting in assigning tasks to particular team members.
Communication	[Google Hangouts](#)	Provides a reliable way for team members to remain in contact by maintaining a private online chat-room for the team to access at any time.
UML Modelling	[LucidChart](#)	Partially automates the creation of UML diagrams and use-case diagrams, which will be required to describe our game's internal architecture and how the game's players will ideally interact with it.

Project Planning	Smartsheet	Facilitates the construction of Gantt charts, such as the prototype time-planning chart that's referred to in the "Project Plan" section on the next page.
Testing *(Following Continuous Integration)**	Travis CI	Augments our chosen VCS with external servers on which new commits can be tested prior to being pulled (through an online terminal), preventing the need for the files changed in such commits to be downloaded, compiled and tested manually instead.
Build automation system	Gradle	A build automation system. It was used to support incremental builds by determining which parts of the build tree are up to date and prevents any tasks dependant on those parts of build from being re-executed.
Java code editor	IntelliJ	A Java integrated development environment. It is a convenient tool to edit Java program code.
Image Editing	Pixelmator	An image editing software for mac, with the ability to do Pixel drawing and export to all the file types we needed.
Image Editing	Paint.NET	An image editing software for windows. This tool was useful when developing the map for the game screen as it allows images with multiple layers (for buildings, land, tile grid etc).

*The outcome of each task will be tested individually through *Travis CI* as such outcomes will be required to be committed to our VCS once they take form

---

Method Implementation

DRAFT FORMAT OF CHOSEN DEVELOPMENT METHOD | *Agile - Scrum*

Sprint Length: 6 Days
Review Period Length: 1 Day

Sprint Starting Day: Wednesday
Review Period Day: Tuesday

Sprint Meeting Days:
Thursday and Monday

Review Period Tasks:
- Confirm that all sprint tasks have been completed successfully
- Review sprint outcomes and determine any additional tasks that may now need to be completed
    - Use the requirements document to do this
- Present outcomes of sprint tasks to all relevant stakeholders and confirm their acceptance/feedback
- Select tasks for next sprint and assign them to developers
    - Use "burn-down" statistics to benchmark progress and factor this into workload decisions
- Set the length, review period and scrum-master for the next sprint
- Perform tests on the whole project by following agreed testing methodologies

Daily Meeting Tasks:
- Scrum-master reviews each team member's tasks and inquires into the progress made on those tasks
- Scrum-master adds and/or removes sprint tasks from the sprint itinerary based on correspondences
- Team members request assistance from peers or from the scrum-master if it's needed

Scrum-Master Responsibilities:
- Prevent team members from falling behind in sprints (either due to underperformance or working issues)...
- ...or from completing additional work that's unnecessary in the current sprint
- Assist team members with issues - whether they're small or large - either by request or personal intuition

Justifications for Implementation Decisions
- Sprints ensue from week-to-week so that they align with the team's shared university time-table
- Sprints begin on Wednesdays because the team generally has few other commitments to meet on that day, enabling each sprint to begin with a burst of work
    - Also allows review meetings to be scheduled for Tuesdays, on which there are many time-slots over which the team is typically available to meet
- Sprint meetings are to be held on Thursdays and Mondays, allowing the team to remain synchronised and up-to-date while also leaving enough time for considerable progress to be made between meetings
- Different scrum-master set each week to balance additional scrum-master workloads between colleagues
- Tasks will be set such that each team-member will have roughly the same amount of work to do during each

sprint; this obviously means that different numbers of tasks may be assigned to different team-members (as some tasks will take more work to complete than other)
- ○ The combined workload warranted by each sprint will be judged using burn-down statistics
- ○ Each task in the project's backlog will be assigned priorities and weights to help judge individual task workloads

---

Project Plan

**The complete timetabled plan for this project is too large to be shown here, so it has been left in the appendix.
What follows on this page is a textual transcription of the plan, complete with priority numbers.**

A list of instructions with associated priorities has been provided for sprints, including tasks for the scrum team and scrum master; this has also been done for sprint reviews and progress meetings. It is important to add this detail so that it is clear what each team member, of a specific team role, must do during sprints of assessment 3 and to ensure that tasks with a higher priority are completed first. The plan was designed to be more general, in terms of completing specific tasks, so that it allows room for replanning if something does not work out how it was intended.

Assessment 4
- Determine and Select Another Project (22/02/17 →27/02/17)
- Complete Supplementary Work (27/02/17 →07/03/17)
  - ○ (1) Devise methods for justifying and implementing changes
  - ○ (2) Review code and GUI of inherited project
  - ○ (2) Review development methods, tools and approaches
  - ○ (2) Review management approaches
  - ○ (3) Update project risk assessment
  - ○ (3) Review testing methods
- Hold Sprint Periods to Implement Required Changes (Wednesday →Tuesday of each week from 01/03/17 → 11/04/17)
- Update Architecture Report for Inherited Project (12/04/17 →18/04/17)
  - ○ (1) Justify any changes made to final solution architecture
  - ○ (1) Create supplementary models (inc. UML/sequence diagrams)
- Complete Final/Acceptance Tests on Inherited Solution (19/04/17 →25/04/17)
  - ○ (1) Describe formal approach to these tests (against requirements and for quality)
  - ○ (2) Carry out and report on resultant tests
- Create final presentation for inherited game (26/04/17 →03/05/17)
- Update website and make sure all the technical information is available (26/04/17 →03/05/17)
- Write up a Project Review of the team management, approach, methods, tools, etc. (26/04/17 →03/05/17)