JKG

Software Engineering Project

Assessment 1

Project Document

Introduction

We are going to make a two player, turn-based game. The game will be competitive and focused around train management within a given area. The players must be able to direct their trains in order to try and complete objectives given to them by the game before the other player whilst also avoiding any obstacles on the map. The map will be modelled on a futuristic version of Europe to give the player a sense of familiarity whilst also leaving room for any creative changes that need to be made.

This document details the user and system requirements, how the system requirements fulfil the user requirements and how the system will fulfill them. This document also contains our development plan, including our predicted time management for the entire project to ensure that the product will be delivered to the customer on time. Also, we included a team management plan so that the team can carry on with this project if something goes wrong.

Requirements & Specification

Assumptions

In order to greatly simplify the requirements and specification of the project, we have made a small number of assumptions about the players and the platform the game will run on. We have made the assumption that the players are all English speakers, and can operate a mouse and keyboard in order to interact with the game. The platform that the game will be played on is a standard issue computer in the Computer Science labs, running Windows 7 with a 1680x1050 screen resolution, a standard mouse and keyboard. The computer will have the Java Runtime Environment installed.

Feasibility

Overall, our current design for this system is highly feasible. We have constructed a Gantt chart in order to ensure that the project is completed on time. We have also written up a section on risk mitigation to ensure that if something out of our control does go wrong, it will affect the progress of the team as little as possible.

Some sections of our design will entail a lot of work to make sure that they meet the requirements, we have allocated time accordingly in our Gantt chart. Some areas of the design have been changed during the design process, to make them more feasible to complete in the specified time-frame, such as the fault generation engine. The original design had the map divided up into different zones, then the fault rate and type of the tracks would be determined based on which zone the track is in. We decided that this would add a lot of complexity while not necessarily delivering a better user experience, so we have replaced it with this much simpler model, instead we are having the fault rate of the track as a set value for each segment of track, and a lookup table to determine the type of fault if one occurs.

Risks

We have tried to avoid risks to the project as a whole, however there are still some risks in the user and system requirements that we could not avoid. One such risk in our design was to put a lot of emphasis on the GUI. If the GUI is not well designed, we could lose a lot of the usability, potentially making the game unnecessarily complicated.

Another risk is that our design for our fault generation engine may be overly complex. When testing to see if a fault occurs, the engine must take into account a great many factors drawn in from different sources such as different objects in the game and databases separate from the game. If we are unable to code the fault generation engine in its current form in time then we will have to reduce its complexity. This could result in the game being less fun for the player as it may be too simplistic or too random.

Objectives:

Our objective for the project is to create a fun and appealing game that follows the customers specification as close as possible. It will be a competitive, turn-based, two player train management game, that results from the following exact requirements:

User requirements:

1. The player must be able to play the game at the same time as one other person
2. The player must be able to play on a map based on Europe, which has at least 5 cities
3. The player must be given three goals by the game
4. The player must be able to select one goal to keep at the end of a turn if none of the goals have been completed
5. At all points during the game, the player should be able to find out what these goals are.
6. The player must be able to accumulate in game resources
    a. The player must be able to get at least 10 different in game resources
    b. such as different trains and money
7. The player must be presented with a GUI
    a. The player must be able to use the GUI to choose their start location
    b. The player must be able to see their start location
    c. The player must be able to use the GUI to set routes for their trains
    d. The player must be able to use the GUI to edit routes for their trains
    e. The player must be able to see the faults on the map
8. The player should be able to save the game state to a file
9. The player should be able to load a game state from such a file.
10. The player must be able to buy resources, such as trains, at a store
11. The player must be able to maintain and repair, track and trains
12. The player must be able to see the other player's updated location after their turn has ended
13. The player must be able to earn resources as a reward for completing goals
14. The player must be able to quit the game at any time.
15. The player must be able to create a new game with default values
16. The player must be able to suffer from track faults
    a. There must be at least 2 different track faults
    b. One of the faults must be a signal failure
17. The player must have the number of turns taken shown by the game
18. The player must have their score shown by the game

System Requirements:

1. The system must be able to track which player's turn it is currently
2. The system must be able to keep track of the number of turns that have passed
3. The system must be able to store both players' information at the same time
4. The system must be able to store the resources belonging to each player
    a. The system must be able to allocate the player additional resources
    b. The system must be able to remove resources from the player
5. The system will store the map in a graph data structure
    a. The map must include multiple nodes which represent junctions
6. The system must store a graphical image of Europe displayed to the players
7. The system must feature a goal engine
    a. The goal engine must be able to generate new goals for the players
        i. The goal generation must be partly pseudo-random
        ii. The goal generation must take account of the current game state (e.g where trains are/ which tracks have been destroyed)
        iii. The goal engine must be linked to a database to fetch goal types and possible objectives etc.
    b. The goal engine must be able to detect when goals are completed
        i. The goal engine must give rewards to the first player that completes a particular goal
        ii. Should two players complete the goal in the same turn-set, the goal engine must split this reward between both players
    c. The goal engine must be able to delete goals when they have been completed
    d. The goal engine must never have more than 3 goals active at the same time
8. The system must ensure that the players have 3 goals at the start of each turn
9. If the players both complete their turns without completing any goals, the system must allow the players to each chose a goal to keep and then discard a goal that neither player chose to keep
    a. Should both players choose the same goal a random one is selected from the remaining two to be removed
10. The system must have a database with all the details about the different in-game resources
11. The system must be able to keep track of the score for each player
12. The system must be able to check the player's score against a victory condition
13. The system must have a GUI
    a. The GUI must display the player's trains
    b. The GUI must allow the player to select a start location that then can not be changed
    c. The GUI must display the player's start location
    d. The GUI must allow the player to specify the route of a train

    e. The GUI must display the current routes of the player’s trains
    f. The GUI must allow the player to edit the current route of the trains
    g. The GUI must display all obstacles the the players
    h. The GUI must display the goals to the player

14. The system must be able to store a start node for each player
    a. The start node must not be able to be changed after it has been set
15. The system must be able to store the route for each train specified by the player
    a. The routes stored by the system must be editable.
16. The system must feature a fault engine
    a. The fault engine must be able to create faults on trains and tracks
    b. The fault engine must be able to remove faults from trains and tracks to restore them to working order
    c. The fault generation must be partly pseudo-random
    d. The fault generation must be context sensitive (not generate faults for things that are already suffering from a fault)
    e. The fault engine must be able to generate at least 2 different types of fault with one of them being a signal failure
17. The system should be able to save the current game state
    a. The system should be able to convert all facets of its current state into a unique format (unique to the system state)
18. The system should be able to load a save file
    a. The system should be able to restore itself to a previous state in every detail from a previously generated save file
    b. The system should be able to verify that a save file is syntactically correct before attempting to load the game from the file
19. The system must feature an exit mechanic for the players to stop the game
20. The system must be able to generate the starting conditions for a new game
    a. The starting conditions must allow each player to place their start node
    b. The starting conditions must allocate both players equal starting resources

See Figure 1

	User Requirements	1.0	2.0	3.0	4.0	5.0	6.0	6.1	7.0	7.1	7.2	7.3	7.4	7.5	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0	16.0	16.1	16.2	17.0	18.0
System Requirements																											
1.0		X																									
2.0																										X	
3.0		X																									
4.0		X					X										X			X							
4.1		X					X													X							
4.2		X															X										
5.0			X																								
5.1			X																								
6.0			X																								
7.0.0				X																							
7.1.0				X																							
7.1.1				X																							
7.1.2				X																							
7.2.0				X																							
7.2.1				X																X							
7.2.2				X																X							
7.2.3				X			X													X							
7.3				X																							
7.4				X																							
8.0					X																						
9.0					X																						
9.1					X																						
10.0							X	X									X										
11.0																											X
12.0																											X
13.0						X			X	X	X	X	X	X					X								
13.1																			X								
13.2										X																	
13.3													X														
13.4											X																
13.5											X																
13.6												X															
13.7																											
13.8						X																					
14.0										X																	
14.1										X																	
15.0											X																
15.1												X															
16.0																							X	X	X		
16.1																							X				
16.2																		X									
16.3																							X				
16.4																							X				
16.5																								X	X		
17.0															X												
17.1															X												
18.0																X											
18.1																X											
18.2																X											
19.0																					X						
20.0																						X					
20.1																						X					
20.2																						X					

Abstract

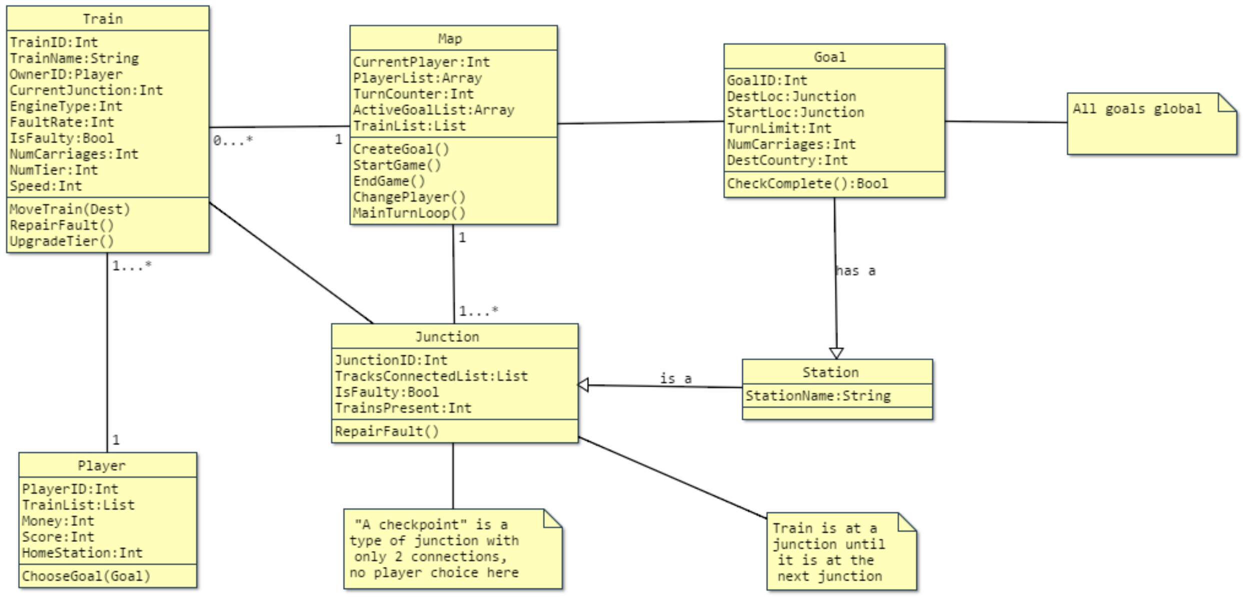


Figure 2

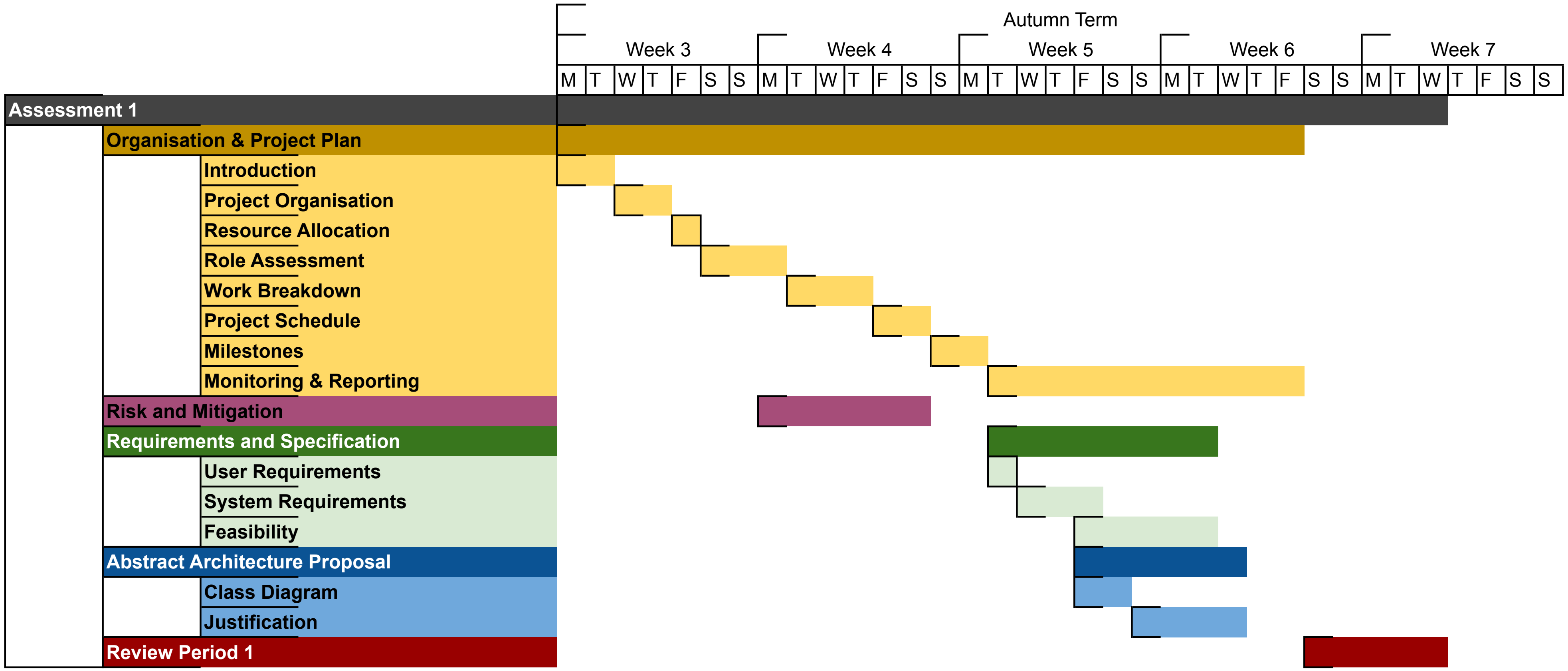
Justification

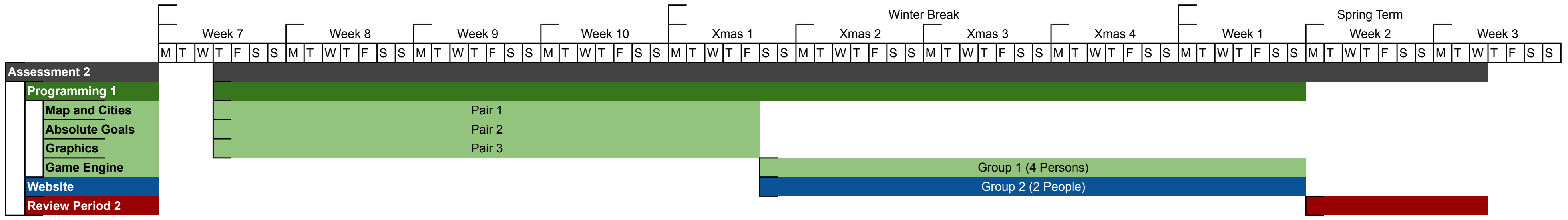
We created a class diagram to ensure that we all understood the design ideas of the project and so that some basic implementation methods could be agreed and explained throughout the group.

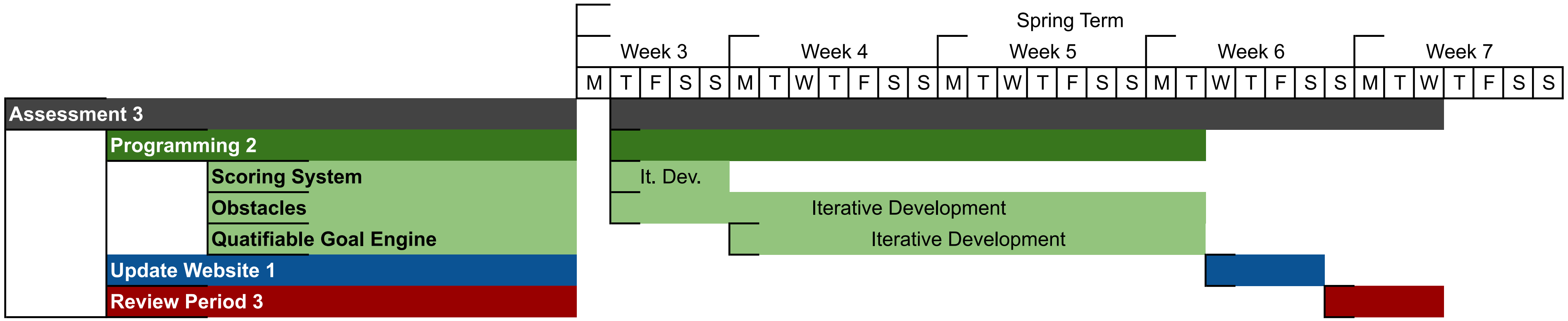
‘Map’ is the core object of the structure, so all other objects (with the exception of ’Player’) are directly associated. ‘Station’ is a subclass of ‘Junction’ as it must have all the same functionality, with an addition- stations will be named. We have decided that an instance of ‘Goal’ will have at least one ‘Station’ instance in it’s definition so that they are easier for players to understand and prioritise. It was decided that all goals should be related to at least one station, as in real life, trains would not pick up/drop off passengers at any other points along the track. It was decided that goals are global to ‘Map’ which should encourage competition, and so, we had to ensure that players are not directly associated with any goals active at any time.

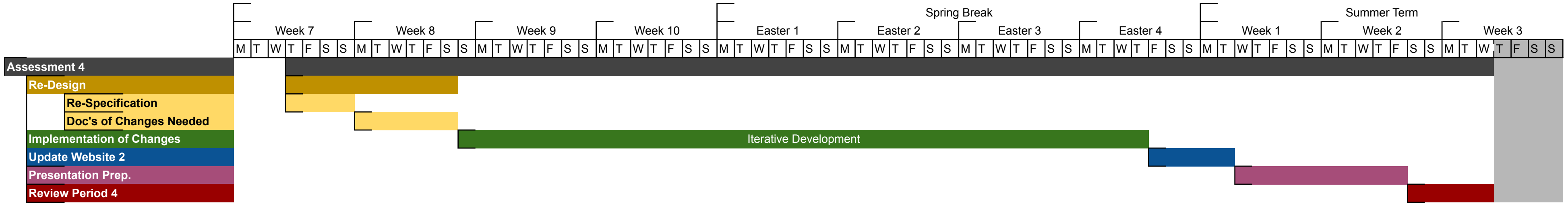
Players are related only to trains, as they will not own stations or track, and because all goals are global.

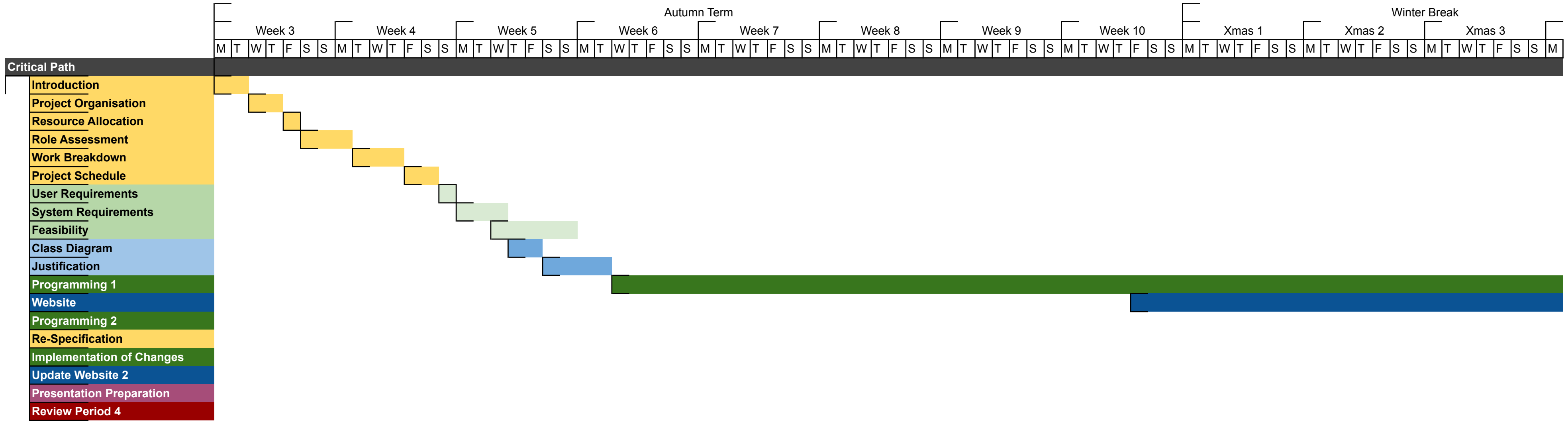
A use case diagram was also created but quickly discarded, as all of the environments we created made for trivial interactions, that a use case diagram failed to explain any further. We also considered creating a sequence diagram, however no further clarity could have been gained from another diagram.

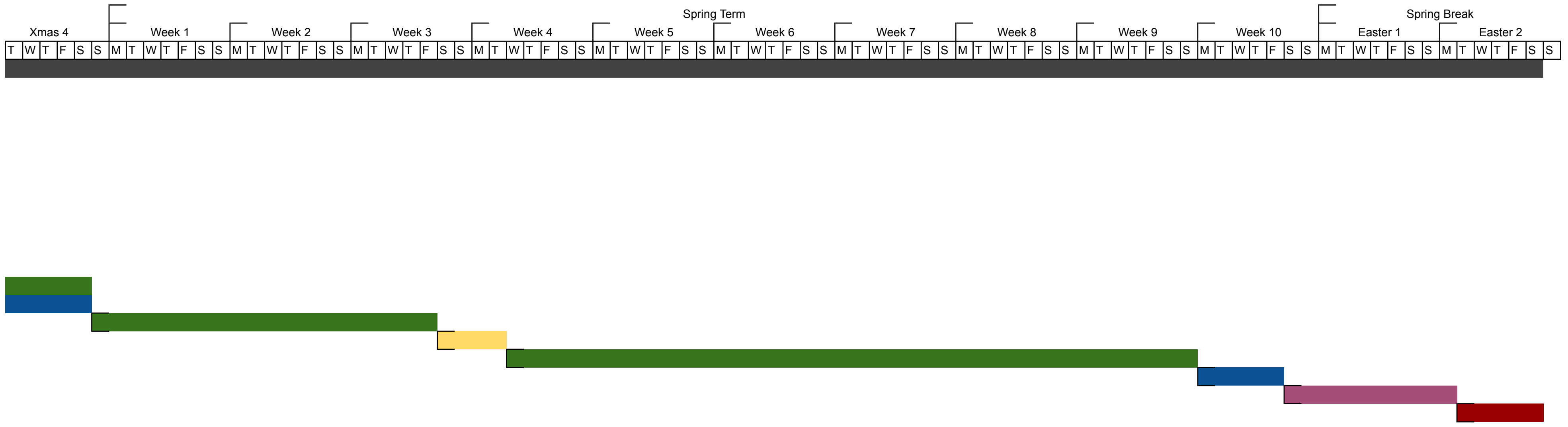

Autumn Term
Week 3
Week 4
Week 5
Week 6
Week 7
M T W T F S S
Assessment 1
Organisation & Project Plan
Introduction
Project Organisation
Resource Allocation
Role Assessment
Work Breakdown
Project Schedule
Milestones
Monitoring & Reporting
Risk and Mitigation
Requirements and Specification
User Requirements
System Requirements
Feasibility
Abstract Architecture Proposal
Class Diagram
Justification
Review Period 1


Winter Break
Spring Term
Week 7
Week 8
Week 9
Week 10
Xmas 1
Xmas 2
Xmas 3
Xmas 4
Week 1
Week 2
Week 3
M T W T F S S
Assessment 2
Programming 1
Map and Cities
Absolute Goals
Graphics
Game Engine
Website
Review Period 2
Pair 1
Pair 2
Pair 3
Group 1 (4 Persons)
Group 2 (2 People)


Spring Term
Week 3
Week 4
Week 5
Week 6
Week 7
M T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S
Assessment 3
Programming 2
Scoring System
Obstacles
Quatifiable Goal Engine
Update Website 1
Review Period 3
It. Dev.
Iterative Development
Iterative Development


Spring Break
Summer Term
Week 7
Week 8
Week 9
Week 10
Easter 1
Easter 2
Easter 3
Easter 4
Week 1
Week 2
Week 3
M T W T F S S
Assessment 4
Re-Design
Re-Specification
Doc's of Changes Needed
Implementation of Changes
Update Website 2
Presentation Prep.
Review Period 4
Iterative Development


Autumn Term
Winter Break
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Xmas 1
Xmas 2
Xmas 3
M T W T F S S
Critical Path
Introduction
Project Organisation
Resource Allocation
Role Assessment
Work Breakdown
Project Schedule
User Requirements
System Requirements
Feasibility
Class Diagram
Justification
Programming 1
Website
Programming 2
Re-Specification
Implementation of Changes
Update Website 2
Presentation Preparation
Review Period 4


Spring Term
Spring Break
Xmas 4
Week 1
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Easter 1
Easter 2
T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S

Team Organisation & Project Plan

Task Priorities, Critical Path, Task Dependencies, etc.

Team Roles-

We defined our team roles from the Belbin Team Inventory[1]. We chose these team roles as we felt that it fitted with our team dynamic better than more traditional structures. We also felt that the Belbin team inventory worked better with the agile development process we chose due to its inherently flatter team hierarchy with the use of the co-ordinator to help resolve any disagreements in the team.

The roles we defined for the group are as follows:

Plant:	Everyone
Resource Investigator:	Richard
Co-ordinator:	Richard
Shaper:	Eashan
Teamworker:	Oliver, Sam
Implementer:	Mark, Daniel
Specialist:	Sam

A Gantt chart (See *Figure 3*) consisting of the development schedule for each of the assessments and the critical path (see end of report) was created. This helped to organise the team and ensure that we kept on schedule to avoid missing deadlines. Each task in the Gantt chart has been assigned a task priority as follows:

Task	**Priority**
Organisation	High
Risk and Mitigation	Low
Requirements and Specification	High
Abstract Architecture Proposal	High
Review Period 1	Medium
Programming 1	High
Website	High
Review Period 2	Medium
Programming 2	High

Update Website 1	Low
Review Period 3	Medium
Re-Design	High
Implementation of Changes	High
Update Website 2	High
Presentation Preparation	High
Review Period 4	High

Some tasks depend on the completion of other tasks in order to be started, the task dependencies are as follows:

Assessment 1 -> Review 1
Assessment 2 -> Review 2
Assessment 3 -> Review 3
Assessment 4 -> Review 4

Role Assessment -> Work Breakdown -> Project Schedule -> Milestones

User Requirements -> System Requirements

Class Diagram -> Justification

Scoring System -> Quantifiable Goal Engine

Re-Design -> Implementation of Changes
Re-Specification -> Documents of Changes Needed

Implementation of Changes -> Website Alterations

Requirements and Specification -> Programming 1 -> Programming 2

Website -> Update Website 1 -> Update Website 2

Proposed Software Engineering Approaches

Agile Development Approach - Refers to the combination of software development methods in which software solutions can be adapted quickly to suit changes in the requirements and specification. This is achieved by close and frequent communication between the development team and the customer. We are told by the customer that the requirements might undergo major/minor changes as we are developing the project. Our top priority is to deliver the softwares in phases and engage the customer in all parts of the development process via face-to-face and electronic communications and ensure their agreement. Rather than spending a lot of time on planning, heavy documentation and fixed contracts, we are constantly working towards rapid delivery of useful software in phases and regular review of circumstances.

Pair Programming - We have decided to use the pair programming technique of the agile software development methodology as it significantly reduces code generation errors and avoids major bugs being discovered later on in the development process. This technique will also allow less experienced programmers in the group to benefit from those with more experience and knowledge.

Development & Collaboration Tools

jsUML2 - We are using jsUML2 as the software for developing our UML Class diagram. Despite the fact that the user interface is not very easy to use, it is available for free online, meaning that it can be accessed from any operating system and from anywhere. Which is useful if various members of the group want to work on the diagram at different times. jsUML2 also allows us to export the UML diagram as xml, meaning that we can import it into other programs.

Eclipse - We have decided to use Eclipse for this project. We chose to use Eclipse specifically because it is very easily extensible, this will allow us to use tools to work with UML to convert our class diagram into Java code. This will help us to better structure our code.. We have decided to use Java, this is because it is an object oriented language which is necessary for our software architecture. Java also has a wide variety of built in libraries which may be useful when writing our game.

GitHub - We are using GitHub as our version control system for keeping track of changes, enhance collaboration and also for backing up all of our software code. We have to chosen GIT to encourage small commits to the project by the team members as well as the fact that it supports branching of the main repository.

Google Drive - We are using Google Drive as our primary backup for documentation and file sharing tool between the group. It allows us to access and collaborate in editing our documentation files at any time and is platform and tool independent.

Risk Assessment & Mitigation

Risk	**Likelihood**	**Impact**	**Mitigation**
Team member unavailable	Very High	Low, if managed	All assessment documents are stored in the shared Google Drive. They can therefore be accessed and edited by anyone in the group.
Poor productivity of individual group members	Medium	Medium	A form of agile development will be used. This ensures short development cycles and therefore a sense of urgency is implied. Generous, but strict deadlines will be given for each task. Each member is expected to adhere to these deadlines, and ask for help in good time, if required.
Total loss of team member	Very Low	Medium	Agile software development will be used, so that a working prototype is available at all stages of development. If a team member is lost, features may be removed from the project, but it should still work as a whole.
Incorrect system requirements	Medium	Very High	Keep communication between the developers and the customers open so that the requirements are fully understood by both parties at each stage.
Requirements inflation	Medium	High	Keep communication between the developers and the customers open so that the requirements are fully understood by both parties at each stage. If a particular feature is added to the requirements later on in the project, explain to the customer that this may come at the expense of other features.
Organisation structure fall-down	Low	High	There will be regular team meetings to check team members are happy in roles, leadership structure etc. Re-organisation may then occur if required.
Loss of	Low	Medium	Use Version Control (GitHub), Documents are

Data/No Backup			stored on Google Drive as a backup.
Failure to keep deadlines	Low	Very High	Strict use of Gantt charts to maintain schedules. Co-ordinator to ensure teams get support if struggling
Copyright Claims	Low	Medium	Check existing copyrights, ideas implemented already
Tool issues	Medium	Medium	e.g. jsUML2, IDE differences. These can be mitigated by ensuring the team use the same tools, and we choose reliability by precedence in our tools.
Conflict between members	Medium	Medium	Project Co-ordinator would attempt to resolve conflicts through compromise.

References

[1] Belbin. *“Team Role Theory - Belbin Team Roles”,* belbin.com. [Online] Available: http://www.belbin.com/rte.asp?id=8. [Accessed: Oct. 17, 2014].