Assessment 1 Report

Software Engineering Project (SEPR)

Team "EEP"

Richard Cosgrove, Yindi Dong, Alfio E. Fresta, Andy Grierson, Peter Lippitt, Stefan Kokov
Department of Computer Science
University of York

Table of Contents

1 **Introduction**
- 1.1 Single Statement of Need (SSON)
- 1.2 Stakeholders
- 1.3 Purpose
- 1.4 Objective and Scope

2 **Requirement Considerations**
- 2.2 Assumptions and dependencies
- 2.3 Constraints
  - 2.3.1 Project Constraints
  - 2.3.2 Design Constraints
  - 2.3.3 Process Constraints

3 **User Requirement Specification**

4 **System Requirement Specification**
- 4.1 Functional Requirements
- 4.2 Non-Functional Requirements
- 4.3 Traceability Matrix

5 **Use Cases**
- 5.1 Starting a Game
- 5.2 Completing a goal
- 5.3 Winning a Game

6 **Architecture**
- 6.1 Game Architecture
  - 6.1.1 Games and Players
  - 6.1.2 Resources Inventory and Store
  - 6.1.3 Regions, Vertices, Junctions and Stations
  - 6.1.4 Trains, Running Status and Goals
  - 6.1.5 Ages and Environment Obstacles
- 6.2 Network Architecture
  - 6.2.1 Master/Slave Network
  - 6.2.2 Network Technology

7 **Planning**
- 7.1 Team Roles
- 7.2 Software Engineering Approaches
- 7.3 Programming Languages
- 7.4 Collaboration and Other Software Tools
- 7.5 Gantt Chart

8 **Risk Assessment**

1 Introduction

1.1 Single Statement of Need (SSON)

Trains Across Europe (TaxE) is a competitive turn-based game. Two players take in turns to route trains between different European cities to complete various game goals. Players may strategically use in-game resources to meet these diverse goals. Upon a goal's completion, the player is awarded a score based on their efficiency and the goal's difficulty. The player with the highest score after completion of all the goals is considered to be the winner [Appendix A.2].

1.2 Stakeholders

Richard Paige, a computer science lecturer at the University of York, will be a direct stakeholder for the game and our main client. We will be in constant communication with Richard, feeding back our progress and ideas whilst acquiring his feedback to cater the game to his needs. We will make use of prototypes to demonstrate possible game functionality. Our team shall also involve other potential users of the game as direct stakeholders in providing input (e.g. through surveys) to improve the game. We may involve the other two lecturers, Tim Kelly and Fiona Polack, as indirect stakeholders to advise our team on different aspects of the project to do with the game or any group problems.

1.3 Purpose

The following requirements document constitutes an agreement between ourselves and our client as to what the users of the game will require and what is needed from the proposed system to meet these requirements.

1.4 Objective and Scope

The objective of the project is to create a fun, playable game of TaxE for users to enjoy by the 29th April 2015 with the inclusion of other project deadlines throughout the year [Appendix A.1]. We are limited by some initial game requirements defined in the assessment briefing [Appendix A.2]. Any further limitations through additional requirements will be defined by the client throughout the software development process through meetings and emails. We will provide the client with ideas for potential features within the game and acquire their feedback on the idea, determining whether to include, adjust or not include in the game.

2 Requirement Considerations

2.2 Assumptions and dependencies

1. Users will run the game on a Windows or Linux operating system.
2. Users will have a mouse and keyboard when playing the game.
3. Users will a monitor of minimum size 1366x768, a standard laptop screen.
4. Users will have Java Runtime Environment installed on their operating system.
5. Users are expected to have at least played games before or be a 'casual' gamer.

2.3 Constraints

Before beginning to formulate specific requirements, we started to consider constraints that would be applied to our project. These are based upon the assessment briefing [Appendix A] and practical considerations.

2.3.1 Project Constraints

Deadlines have been imposed by the direct stakeholder(s), as specified in the assessment briefing document. Workload constraints exist due to the formation of our team:

1. All work on this project must be completed by only a team of six people, each with their own skillsets. We are all 2nd year computer science students with other study obligations alongside the software engineering project.
2. Our team has never worked on a large scale software engineering project such as this before, hence the project will be a learning experience for our team. This will mean our team will be working at a slower pace than a team of experienced software engineers.
3. As time is a precious commodity throughout the project, each team member will be allocated a task suitable to their skillset.

2.3.2 Design Constraints

1. The game is expected to be compilable and executable on any personal computer within the York Computer Science labs that uses either Windows 7 or Linux Ubuntu 14.04 for its operating systems. The game will need to be able to run on at least one operating system to be playable and, due to time

constraints, it might be too time consuming enabling the game to run on all platforms. In addition, the game will be tested by the client on a university computer that contains Windows or Linux, hence it would be best to make the game for one of these two operating systems.
2. The software resources we have available include any software available on the University of York's computers and any we may deem useful in aiding us to complete the project. The main programming language of our game will need to be one covered by our team in the past, preferably Python (version 2.7.8) or Java (version 7) as these were taught to all Computer Science students. This would otherwise take too much time and effort in making everyone learn a new language and potentially delay the project.

Constraints specified in assessment briefing:
1. The game is expected to include at least 5 cities.
2. The game is expected to support at least two possible routes when routing a train between any two cities.
3. The game is expected to support at least 10 different goals.
4. The game is expected to allow a player to have a maximum of 3 goals assigned simultaneously.
5. The game is expected to support at least 10 different resources.
6. The game is expected to support at least two junctions (routes that intersect).
7. The game is expected to have at least two different types of obstacle.

2.3.3 Process Constraints

The following constraints will be held throughout development of the system:
1. We have chosen to use IEEE830 'Recommended Practice For Software Requirements Specifications' [1] as a template for our requirements document.
2. We are to use the IEEE citation for any referencing made within our documentation.
3. We will follow the appropriate programming specification for our chosen language (Python [2] or Java [3]).
4. The system must be sufficiently documented such that a different team can continue developing for assessment 3 and beyond.
5. We are not allowed to pay people to do work for us. However, we are able to use freely available open source libraries as part of our system.

3 User Requirement Specification

- Description - Details of what both players (users) should be able to do within the game.
- Necessity - The priority of involving the requirement in the game. The priority level has been agreed with the client and is ranked in the following manner:
  - Essential - A feature the game must have in order to be considered complete by the client
  - Preferable - A feature the client would like in the game, but is not required
  - Optional - A feature worth considering as a point of expansion for the game in the future.
- Origin - Where the requirement originated from for backward traceability. As part of game research we considered games, such as Civilization, and discussed different features with potential users to discover what they thought about them. The features that our group or potential users felt would be a good addition to the game were included in our user requirements. Evidence of interviews with the client is included in the appendix.
- Justification and possible alternatives - Reasons for involving the requirement, including an investigation into possible alternatives.

UR No.	Description	Necessity	Origin	Justification, feasibility, possible alternatives
	Menu Screens and Starting a Game			
1.1	Players must have access to a menu screen when initially running the game application with the following navigation options e.g. (1) Play Game, (2) How to Play, (3) Credits, (4) Quit	Essential	Game research and interview with client	We agreed with the client that a fundamental feature is allowing players the freedom to enter the game via a menu screen. Players may not yet be ready, and choose "How to play" the game or discover who made the game (credits). The alternative option would be to immediately place the player into the game, which would likely cause confusion and annoyance for the player.
1.2	Players must be able to play on a network	Essential	Game research	Based upon research into other games, we settled on creating a network based game. This allows players in

	against a player on a different computer.		and interview with client	different locations to play against each other and players can explore the game world and strategize even when it is not their turn. It will be kept simple to avoid adding unnecessary complexity to the game with the help of programming libraries. Alternatively, two players play on the same computer (hotseat), which would require no networking. However, both have to be present in the same room to play, which can be hard to do regularly if the two players do not live together or nearby.
1.3	Players should be able to enter their nicknames to be used during the game.	Preferable	Game research and interview with client	Most multiplayer games allow a player to choose a nickname as it adds a level of personalisation to the player's experience. There is a risk of players choosing offensive names, however. This could be mitigated through an offensive language filter.
1.4	Players must be able to create a new game or join an already created one from a list of available games.	Essential	Game research	We chose to create a simplistic lobby system where players can either join another player's game or create their own game. There is a risk the server may not be able to host too many games at once. A full lobby would require more functionality (e.g. sending and receiving game requests between players) and take more development time. It is possible to implement this in the future.
1.5	Players should have an escape/pause game option which brings them to a menu with options e.g. (1) Resume Game (2) How to Play (3) Save Game (4) Abandon Game	Preferable	Game research	The client suggested that each player should be able to pause the game if they need to take a break with a time limit in case the player who pauses does not return. Not being able to pause could mean the other player races ahead in the game and unfairly wins. To let the game progress at a reasonable pace whilst still incorporating a pause function, limits to the duration and the frequency of the pauses could be implemented.
1.6	Players should be able to save and load games so they can resume gameplay at a future point.	Preferable	Interview with client	The client suggested this as a feature he would like to see in the game. This may take a significant amount of time to implement. Alternatively, we do not have to implement this requirement, resulting in our team spending less time developing the game and not being as risky to implement. It is a requirement to include if there is free time.
1.7	Players should be able to set the difficulty/length of the game.	Preferable	Interview with client	The client liked the idea of being able to pre-set the length of the game, e.g. how many objectives must be completed to progress through each age within the game. To keep in line with UR 1.4, players will only be able to join other players who have chosen the same difficulty.
	Turns and Train Routing			
2.1	Players must take it in turns to play the game and should know when it is their turn.	Essential	Assessment briefing	A player needs to know when it is their turn, as the alternative would result in a confusing game experience. The turns could be taken within a time limit (e.g. 60 seconds) to stop players delaying the game by taking too long with their turn or moving away from their keyboard.
2.2	Players must start with one train at the beginning of the game.	Essential	Interview with client	We agreed with the client that the player cannot be placed in a position where they cannot progress in the game. Therefore, the player needs to be able to start with a train in order to start progressing.
2.3	Players must be able to route a train between the European cities visible on a map; e.g.	Essential	Assessment briefing	We have chosen to use real city names and most likely capital cities for a more immersive experience, as players should be familiar with them, and to fit with our historic goal-driven story. The alternative would be to use fictional

	London, Paris, Berlin, Rome and Warsaw			cities, which would not fit within the historical theme of our game.
2.4	Players should be able to purchase (using resources) a contract, giving them permission to send their trains along specific routes.	Preferable	Interview with client	The client liked the idea when we shared it with him. The ability to purchase certain routes between cities would reward players for progressing through the game and leaves players with the strategic choice as to where to invest their resources. The alternative is to immediately give players permission to use all available train routes, which seems less interesting.
2.5	Players could be allowed to have their train stop by way-points between the routes connecting cities.	Optional	Interview with client	The client agreed it would make the game more interesting and strategic for players to have stops in between the main cities (e.g. smaller cities). However, this would take more time to develop and the assessment briefing states that specifying advanced routings is optional.
2.6	Players must come across at least two junctions (train routes that intersect).	Essential	Assessment briefing and interview with client	Keeping with the design constraints specified in the assessment briefing, the game should have at least two junctions. The junctions would allow players to alternate their route if they changed their mind.
2.7	Players must come across at least two different types of obstacles, in which one must be a junction failure.	Essential	Assessment briefing and interview with client	The game must have at least two different types of obstacles. We have chosen to link in this requirement with 2.6 to allow obstacles to impede players on junctions (e.g. signals are not working). The client has requested fun and wacky obstacles, such as yetis in the colder regions of Europe and using the weather based on the region.
2.8	Players must be able to see all of their current trains and their respective route and destination.	Essential	Assessment briefing and interview with client	The assessment briefing and client have made it clear that the player must be able to see the progress they are making towards goals. Thus, players need to be able to keep track of their trains' movements in case they wish to change their route.
2.9	Players should be able to see all of their opponents' trains and possibly their respective route and destination.	Preferable	Interview with client	The client suggested that it would make the game more interesting for players to be able to make decisions based upon what their opponents are doing. The alternative is to require a player to buy upgrades in order to see what their opponent is doing or use a fog of war where the player has to have "vision" over a region, however this would add greater complexity to the game.
2.10	Players must be able to see how many turns it would take a train to travel between any two cities.	Essential	Assessment briefing	We consider the number of turns for a train to travel between cities to be a crucial part of the route's description (required by assessment briefing). The alternative would be to leave the player guessing how long it would take to reach a destination, which would make it hard to plan their moves and goals to complete.
	Game Progression			
3.1	Players must be assigned a new goal each turn. They must be visible at all times during the game.	Essential	Assessment briefing	A player will only be assigned a new goal if they are within the constraint of 3 goals simultaneously. The goals must be visible to ensure a player always knows what they should be trying to do. The goals can be quantifiable (e.g. reach London from Paris in 3 turns) or absolute (e.g. send a train from Berlin to Brussels).
3.2	Players could be able to choose a goal from a list of possible goals, and also be able to abandon current goals.	Optional	Interview with client	The client suggested this as an idea to allow players to choose whether to try and complete more challenging goals or stick with easier goals. Players need to be able to abandon goals in case they cannot complete a goal due to insufficient resources.
3.3	Players must be given a score after completing a	Essential	Assessment briefing	Players will be given a score for completing a goal and will be rewarded more points for goals that require more

	goal. The score will be based upon how efficiently a goal was completed and its relative difficulty.			resources and time. The alternative would be giving players the same reward for all goals, which would create no incentive for players to choose harder goals or take more risks in the game.
3.4	Players should progress through different ages upon reaching minimum scores. E.g. the steam age will allow access to steam trains.	Preferable	Game research and interview with client	The client really liked the idea of progressing through history during the game. Players are more immersed into the game through means of a historical story through the different ages. Resources, trains and goals will be themed based upon what age the player is currently in.
3.5	Players must be able to see their current score, and possibly their opponents too.	Essential	Assessment briefing	The assessment briefing states a player should be able to see their score. We feel that being able to see your opponents score will make the game more competitive and exciting, something that the client desires. This could help players strategise about their next move, whether to improve their own score or try and delay their opponent.
3.6	Players' games must end once either player has completed their goals. The player with the higher score wins.	Essential	Assessment briefing	The assessment briefing states a player should win the game when all goals are completed. The alternative would be to wait for both players to have completed their goals, which would be boring for whichever player has to sit waiting for their opponent to finish.
	Resources			
4.1	Players must receive two spendable resources (Gold and metal in our case) each turn. Players with a higher score will receive more of both resources.	Essential	Assessment briefing and game research	In a similar way to other games, the client agrees with us that rewarding players who earn a higher score allows them to experience more interesting gameplay. These spendable resources are crucial to allow players to purchase more resources such as upgrades.
4.2	Players must be able to spend their gold in a store where they can buy upgrades for themselves and traps to use against their opponent. The player should have their resources stored for future use.	Essential	Assessment briefing and interview with client	These upgrades and traps will meet the constraint of there being at least 10 different resources and players will be allowed to hold 7 at a time (5 minus the gold and metal). The client really liked the idea of players being able to choose what resources they would like to buy, rather than being allocated them randomly. A player may choose to purchase resources to sabotage their opponents' progress, or they may wish to try and speed up their own progression. This adds a greater element of strategy and competitiveness to the game, making a unique and diverse experience each time.
4.3	Players should be able to spend their gold and metal on buying new trains.	Preferable	Interview with client	Whilst the briefing does not specify there must be more than one train, we agreed with the client that it may make the game more interesting if a player can control multiple trains at the same time. This would add an extra element of strategy and uniqueness to each game.
4.4	Players must have their resources removed from their inventory once they have been spent or used.	Essential	Assessment briefing	Once gold and metal have been spent or upgrade/traps have been used, it would not make for interesting gameplay if these same resources could keep being used.

4 System Requirement Specification

The system will need to meet the user requirements specified. To do this, our group has developed some design decisions for our system. These requirements will be adhered to in the development process and used to create a testing framework.

4.1 Functional Requirements

FR No.	Necessity	Description (Inputs and transformations)	Invariants and Failures	Meets UR
1.1	Essential	The system shall provide a menu listing options which upon being selected [using mouse or keyboard] by a user will navigate them to the required screen. The options will include: ● Resume - player navigates back to the current game (if one has been started) ● Play Game - player navigates to server list screen (if game has not yet been started) ● How To Play - player navigates to the game manual ● Credits - player navigates to a screen with a list of the people who created the game and their respective roles ● Quit - player quits the game	System must always navigate to the screen the user requests.	1.1
1.2	Essential	The system shall use a server that acts as a relay between two players, and will synchronise all the game variables between both players whenever a turn has been made. E.g. if player 1 has routed their train to a destination, then player 2's map should update to display this.	If a player loses their connection the game must allow up to a minute for a connection to be re-established, otherwise the disconnected player will lose by default.	1.2 and 1.4
1.3	Preferable	The system shall request players input a suitable nickname when they are creating or joining a game, which will be stored on the server and made visible to both the player and their opponent throughout the game.	Nicknames considered offensive will be forbidden and blocked by a bad language filter.	1.3
1.4	Essential	The system shall allow players to connect to a server which will output a list of available games and the nickname of the player who created the game. The server shall act as an intermediary between players, allowing them to add new games to the list or join already created games.	Games which have been filled (i.e. it has two players) must not be displayed on the available games list.	1.4
1.5	Preferable	The system shall allow a player to request the game to be paused for a maximum of five minutes. Both players' game interfaces will be frozen until the player has returned.	A player can only request the game be paused once during the game, otherwise they will not be allowed. If the game is paused for longer than 5 minutes the game will continue regardless of whether the player is ready or not.	1.5
1.6	Preferable	The system shall automatically save the game on the server at every turn. The players shall be able to interrupt a game at any moment to continue at a later time. The system shall keep track of started games and allow the players to choose a game to continue.	The player should have no control on the auto saving functionality as this could make an unfair advantage if a player could revert back to an earlier save.	1.6
1.7	Preferable	The system shall allow the host player to set the initial difficulty of the goals before the game starts (Easy, Medium and Hard initially, with potentially more later). The goals in game shall adjust in difficulty depending on the distance between the two cities in question and how realistically the	The host player cannot change difficulty once the game has started as this would be hard to implement.	1.7 and 3.3

		quantifiable value is achievable.		
2.1	Essential	The system shall use a round-robin pattern to switch between both player's turn. It will follow this sequence: (1) System determines the player to go first (2) Player is given a maximum of 2 minutes to make their turn - if they exceed the time limit they forfeit their turn. They can also end their turn before the time limit if they are done by pressing "end turn" button. (3) Next player is informed that it is their turn and they are given 2 minutes.	A turn must never exceed 2 minutes and a player must never have two turns in a row.	2.1
2.2	Essential	The system shall assign both players a default starter train at the beginning of the game.	The train will always be of the lowest quality in the starting age (e.g. Steam train for Steam age).	2.2
2.3	Essential	The system shall allow players to select a train and specify a city on a map where it should start and then the cities it should pass in order to reach its destination. The train's route shall remain displayed on the player's interface.	A player can only select a train that belongs to them. They can only specify a route that is unlocked for them - the system will block them from making routings they do not have permission to use.	2.3
2.4	Preferable	The system shall allow players to buy contracts to use special routes that will be highlighted a different colour on the map. The player will exchange gold for the system to give them the ability to send trains along these routes.	Special routes must not be able to be used if the player has not bought them using gold.	2.4
2.5	Optional	The system shall allow players to specify more specific routes involving more than the 5 primary cities in the game, e.g. smaller cities nearby to the primary cities.	The cities and their locations must remain consistent for both players throughout the entire game.	2.5
2.6	Essential	The system shall allow players to interact with at least two junctions, whereby a player will be given the choice of changing the routing of their train. If a player chooses to change the routing the map will update to reflect this.	A player must be able to change their route at a junction.	2.6
2.7	Essential	The system shall use a random function such that there is a chance that when passing a junction a random obstacle will occur, e.g. the signals fail and the player's train is delayed for several turns. A player will be informed when this has happened.	Games that are of an easy difficulty will have a 5% chance of a random obstacle occurring; medium difficulty will be 10%; and hard difficulty will be 20%.	2.7
2.8	Essential	The system shall show a player all their trains and their respective locations on a map. Upon selecting a train the player will be able to see the train's destination.	A player must see their train at all times.	2.8
2.9	Preferable	The system shall show a player all their opponents' trains and their respective locations on a map.	The system must make it easy to distinguish between a player's trains and their opponents, e.g. opponents trains could be highlighted red.	2.9
2.1 0	Essential	The system shall calculate how many turns it would take a player's train to travel between any two cities on the map. The result of this calculation will be displayed to the player before they have confirmed	The number of turns calculated will always be a whole number. The calculation must be consistent, such that any player with the same train	2.10

		their train's route.	and same routing would take the same number of turns.	
3.1	Essential	The system shall assign a player a new goal each turn, unless they already have the maximum of 3 goals. The goal will be visible on the player's interface at all times.	The starting location of a goal must always be a city where a player currently has a train stationary.	3.1
3.2	Optional	The system shall allow a player to choose from a list which goal they are to be assigned. At the beginning of each turn a player may abandon a current goal and choose a new one.	Player must remain within constraint of 3 simultaneous goals. Players must be able to abandon goals for which it is impossible for them to complete.	3.2
3.3	Essential	The system shall calculate a score to award a player upon their completion of a goal. E.g. Score = Goal difficulty coefficient / Number of turns to complete goal	Any player who completes any specific goal with the same efficiency must always receive the same consistent score.	3.3
3.4	Preferable	The system shall allow players upon completing a specific number of goals to progress through to a new age, which will have new goals with relevant historical backstory and resources/trains available to buy.	The minimum goals required to complete to progress to a new age must remain consistent for both players throughout the game.	3.4
3.5	Essential	The system shall make both players scores clearly visible at all times on their interface.	The score must be updated at the end of each turn.	3.5
3.6	Essential	The system shall end the game once a player has completed a specific number of goals. The player will be shown a congratulatory or consolation message and then returned back to the menu.	The player must only win the game if all their goals are completed.	3.6
4.1	Essential	The system shall grant each player two resources (metal and gold) upon the start of their respective turn. The amount of gold and metal will be based upon the player's score.	Gold and metal must be equal to a constant value multiplied by the player's score.	4.1
4.2	Essential	The system shall use a store to allow players to spend their gold on resources. Metal will be used in combination to purchase additional trains or train upgrades.	Purchased resource must be placed in player's inventory over an empty resource slot.	4.2 & 4.3
4.3	Essential	The system shall use an inventory system to allow players to store up to seven resources.	Players must use a maximum of five additional resources and replace an empty slot with the purchased resource.	4.2 & 4.3
4.4	Essential	The system shall remove resources once they have been used from the player's inventory.	Resource must be removed and replaced by an empty slot in the inventory.	4.4

4.2 Non-Functional Requirements

NFR No.	Description	Function Constrained
1	The list displaying available games should never be left out-of-date for longer than ten seconds, i.e. it must periodically refresh at least every ten seconds.	1.4
2	A maximum of five seconds delay is allowed between each turn, e.g. for the system to calculate	1.2, 2.1

	the new location and trajectory of a train and to communicate this to the other player over the server.	
3	If a player loses connection from the server, the other player must be informed within 30 seconds that connection has been lost.	1.2, 2.1
4	The number of turns calculated for a train to travel between any two cities must always be an integer.	2.10
5	All quantities of resources (e.g. gold, metal and boosts) should always be an integer.	4.1, 4.2

4.3 Traceability Matrix

FR No.	Starting/Joining a game	Turns and train routing (via a map)	Goals and score (Game Progression)	Resources
1.1	x			
1.2	x	x	x	x
1.3	x			
1.4	x			
1.5	x	x	x	
1.6	x		x	
1.7	x	x	x	
2.1		x		
2.2		x		x
2.3		x		
2.4		x		x
2.5		x		
2.6		x		
2.7		x	x	
2.8		x		
2.9		x		
2.10		x		
3.1		x	x	
3.2			x	
3.3		x	x	
3.4			x	x
3.5			x	
3.6		x	x	
4.1		x	x	x
4.2			x	x
4.3				x
4.4			x	x

5 Use Cases

5.1 Starting a Game

Primary Actor: Charlie (Player)
Supporting Actor: Fred (Player), Game Server
Precondition: Charlie and Fred both have the game downloaded and installed on their respective computers.
Trigger: Charlie and Fred have both started the game application and navigated to the ‘Play Game’ menu option.
Main Success Scenario:

1. Charlie’s and Fred’s interface displays an empty list of games to join - clearly indicating that nobody else has yet created a game for somebody to join.
2. Charlie decides that he will create a game, and presses a ‘Create Game’ button. He is given the option of choosing the difficulty of the game, ‘Easy’, ‘Medium’ or ‘Hard’. He chooses to create an easy game.
3. Charlie starts waiting for somebody to join the game he has created.
4. Within ten seconds of Charlie creating a game, Fred’s interface is updated to show that a new game has been created which is of easy difficulty and has been created by a player with nickname Charlie.
5. Fred decides to join the game by presses a ‘Join Game’ button.

6. Charlie is alerted that Fred has joined his game, and they are sent to a loading screen whilst the game is prepared by the system.
7. Within ten seconds the game should have started.

Secondary Scenarios:

1. **Server is offline -** Upon Charlie and Fred both pressing ‘Play Game’ they are both shown an error message informing them that a connection cannot be made to the server hosting the list of games.
2. **Charlie’s internet connection is down -** Upon Charlie pressing ‘Play Game’ he is shown an error message informing him that the game is unable to make a connection to the internet.
3. **Charlie loses his internet connection after creating a game -** Despite just joining the game that Charlie has created, Fred is removed from the game and the game is removed from the list of available games. Fred is informed that Charlie had lost connection.
4. **Charlie and Fred both decide to create a game -** Fred notices that Charlie has also created a game and decides to delete his game and join Charlie’s game instead.

Success Postcondition: Charlie and Fred have both started a game against each other and the first player is starting their turn.

5.2 Completing a goal

Primary Actor: Ben (Player)
Supporting Actor: Laura (Player)
Precondition: Ben and Laura have both successfully started a game against each other.
Trigger: Ben has been given the goal - “Transport soldiers from Paris to London”
Main Success Scenario:

1. Ben selects his stationary steam train, which he can clearly see on his map located in Paris.
2. The game asks him to select on the map where he would like to send his train and Ben selects on the map London. He is informed that it should take approximately 4 turns to reach London from Paris using a steam train. He confirms the routing.
3. His map is updated with a direct route from Paris to London glowing in green, so he can visibly see the direction the train will be heading.
4. Ben’s 30 second turn is up and it becomes Laura’s turn, who is given a different goal to Ben.
5. Once Laura has completed her turn, Ben’s map updates to show his train is making progress along the route from Paris to London.
6. This continues for two more turns, until Ben’s steam train finally reaches London successfully.
7. Ben is awarded 100 points for completing his goal, and he is assigned a new goal with a starting location of London.

Secondary Scenarios:

1. **Ben comes across an obstacle** - Upon reaching a junction an explosion causes damage to the track, meaning Ben’s train is delayed for 2 turns until the track can be fixed. Once Ben finally reaches London he is awarded a lower score due to taking longer to reach his destination.
2. **Ben purchases a boost for his train using gold** - The boost he purchases makes his train temporarily faster, meaning he reaches London in only 2 turns rather than 4 turns. He is awarded a higher score for reaching his destination more quickly. The boost is removed now that he has completed his goal.
3. **Laura purchases an impeder to sabotage Ben’s train** - The impeder she purchases temporarily slows his train down, meaning he reaches London in 6 turns rather than 4 turns. He is awarded a lower score for reaching his destination more slowly. The trap is removed now that he has completed his goal.
4. **Laura wins the game** - Before Ben can complete his goal, the game has ended and Laura has the higher score and wins.
5. **Laura pauses the game** - Ben’s game is frozen and he is informed that Laura has paused the game and a countdown timer is shown for 5 minutes. Laura returns after 2 minutes and the game is resumed.
6. **Laura loses connection/quits the game** - Ben is informed that Laura has lost connection and the game must be abandoned.

Success Postcondition: Ben has completed the goal he was set and has been awarded a score.

5.3 Winning a Game

Primary Actor: Mark (Player)
Supporting Actor: Tom (Player)
Precondition: Mark and Tom both have the game downloaded and installed on their respective computers. They also have started the game and played some turns.
Trigger: Mark only needs to complete one last goal to finish the current Age. It's Tom's turn.
Main success scenario:

1. Tom sends two trains, one from London to Madrid and another from Munich to Rome. The turn ends;
2. Mark is asked to play his turn;
3. One of Mark's trains end its journey achieving one of Mark's final goal meaning Mark has progressed beyond the final age of the game meaning the game is complete.
4. It is clear that Mark has gained a higher score than Tom and therefore Mark is declared the winner and Tom is consoled for his loss.
5. They both click on "Close". This brings them back to the main game menu.

Secondary scenarios:

1. **At the end of the game, Mark's and Tom's scores are identical** - The game invites Mark and Tom to play another turn until their score is different and a winner can be identified.
2. **Despite Mark finishing all his goals first Tom still wins due to a higher score** - Mark chose to complete only easy goals whilst Tom decided to go for more difficult goals, meaning in the end Tom still earned a higher score.

Success Postcondition: Once a player has completed all their goals (i.e. they have completed the final age) the game decides who the winner is based upon who has the highest score.

6 Architecture

The architecture explains the structure and behaviour of a system. For our system, we will describe the main concepts, the entities and the network architecture of our proposed system, justifying our design decisions during the development process.

6.1 Game Architecture

The full UML class diagram for the proposed Architecture of the Game can be found at the end of the report. In order to facilitate understanding of the architecture, the following sections refer to individual parts of the class diagram.

6.1.1 Games and Players

The two main entities for a two-player game are the "Game" and the "Player". The Game class represents any current game being played by two Players which will hold all the information about the progress of the game, such as the turn being played and its code (UID). The Player class represents the two players of the Game which includes the properties of the Player's nickname, their score and the number of objectives accomplished by the Player.
Justification: A master "Game" class is needed as a unique point of access to all of the Game information and will facilitate the collection of data exchanged in the network. Each Player will be able to edit its own Player object by specifying their nickname and playing the game (e.g. by buying a resource or advancing an age).
Related Requirements: FR no. 1.2, 1.3 and 1.4.

6.1.2 Resources Inventory and Store

We classified Resource as any type of Spendable or Usable resource within the game. All Resources have a short name and a description associated with them. The two Spendable Resources Gold and Metal are essential elements of the game that can be used by the player. They are used to buy other Usable resources when the player can afford them in the store.
All Usable Resources have a price property that is expressed in quantities of Gold and Metal, and are usable on Trains of both players. For example, a Train Speed Modifier is a Resource that specifies a Speed Factor to be applied on a train. Examples of Modifiers could either be a speed boost for a Player's own Train, or any

Impeder that slows down an opponent's train. The Store class represents the in-game shop that contains a collection of Usable Resources available for the Players to buy.
The "Inventory" of a Player is defined as the set of all the Resources currently owned by the Player - both those that are currently being used and those that are not.
Justification: This Architecture allows us to have some common properties for objects that are very different (e.g. Gold and a Speed Upgrade). The Store class is very helpful to keep a reference of all the available Resources that can be bought in the Game.
Related Requirements: FR no. 4.1, 4.2, 4.3 and 4.4.

6.1.3 Regions, Vertices, Junctions and Stations

The Game's map will be internally represented as an undirected weighted graph, a collection of Vertices and Arcs. A Vertex in the map can either be a Station with a recognisable name (e.g. "London King's Cross) or a simple Junction.
A Route is an arc between two vertices and has a variable length that represents the distance between the two vertices. This length will be used to calculate estimations for journey times in turns based upon a given train's speed, or calculate the shortest path between any two vertices in the graph whenever the Player decides to move a train.
In order to make the game more interesting, we decided to introduce Regions. These will be areas of the map, represented as collections of Vertices. Each vertex in the map will belong to one Region. The Regions of the Game will have a name (e.g. North Europe or Britain) to categorise the Random Obstacle Events, so that the game will be able to easily choose an adequate natural phenomenon or historical event for the region.
Justification: The nature of the railway network could not lead to anything other than having a graph and separate classes. For simplicity, we decided to avoid including route directions, and instead decided that an undirected graph will best fit our needs.
Related Requirements: FR no. 2.3, 2.5, 2.6 and 2.8.

6.1.4 Trains, Running Status and Goals

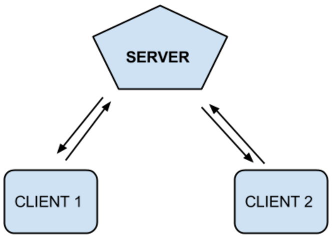
Goals represent the objectives that the Player needs to achieve in order to be awarded points, advance to the next stage or age and finish the game. Each Goal has properties such as a general description, a reward in score, an age of the game that the player needs to have reached before being assigned the Goal, and, more importantly, at least two Stations that a Player's train needs to start travelling at and end at to achieve the Goal. All Goals need to be checked for accomplishment at the end of every turn.
Status represents the current position, if any, of a Train on the map, and its position relative to a journey. It is represented by a set of three elements: an ordered list of routes of the whole journey, a number that indicates the current route of the list and a Route Completion figure that specifies at which point of the current route the train is positioned.
The Train class is used to represent a single Player's train. A train is strictly related to an Age (e.g. can either be a Steam Train or an Electric Train), some properties such as a Make and a Model, a Base Speed (that can be altered using an Upgrade) and when it was placed on the map, as opposed to being idle in the player's Trains Depot. It also has a Status object representing the current position and route, if any.
Related Requirements: FR no. 2.3, 2.5, 2.8 and 2.10.

6.1.5 Ages and Environment Obstacles

The Age class represents an Age of the Game. An Age is related to a number of Random Obstacle Events that can occur throughout the Game, while the Player is in that particular Age, and a set of Trains built during that Age (e.g. Nuclear Trains for the Nuclear Age).
Random Obstacle Events have a description that can either be of a historic or natural event, which help to compose the story of a particular Game. As a consequence of this, a Player's Train would have a higher or lower Speed Factor than its original value. The Obstacles are chosen randomly by the Game from the set of obstacles based on a Player's stage. The probability that a Random Obstacle Event occurs at each turn is defined by the Game difficulty (higher difficulty meaning more probability). An event may affect a train for a number of turns, and the "Turn Penalty Remaining" property keeps track of how many are needed for the effect of the Obstacle to end.
Justification: The Game is based upon the concept of having a Player progress through different ages from the past to a future to give the Players a sense of creating their own story and ideal train company. It can give

the player a sense of progression and satisfaction when seeing their train company grow from being recently established to a monopoly (i.e. fully control the train business of Europe when they win the game). An immersive story can keep Players playing the game, especially one which evolves, adapts and brings a unique experience each time. This, in response, creates the need for an object that represents any playable Age which can be used to classify a number of other entities in the game, such as Trains and Random Obstacle Events.
Related Requirements: FR no. 2.7.

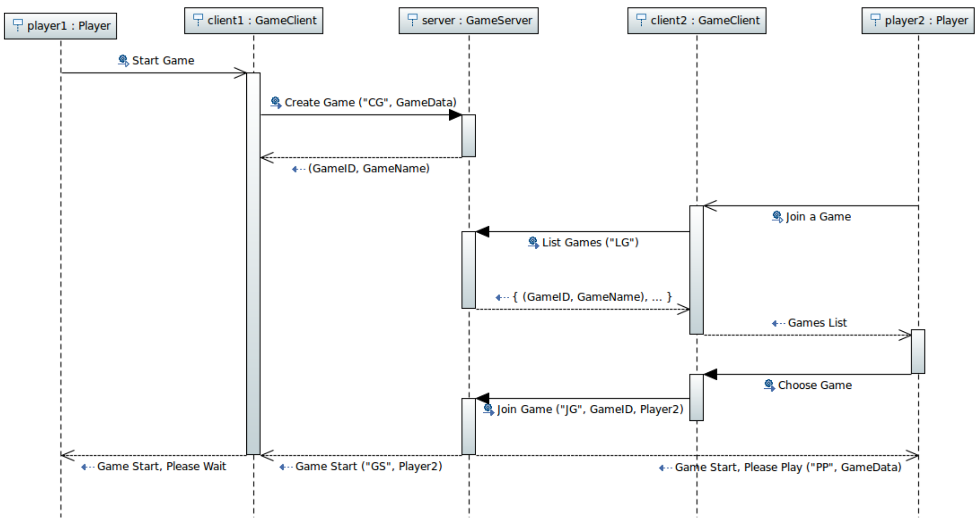
6.2 Network Architecture



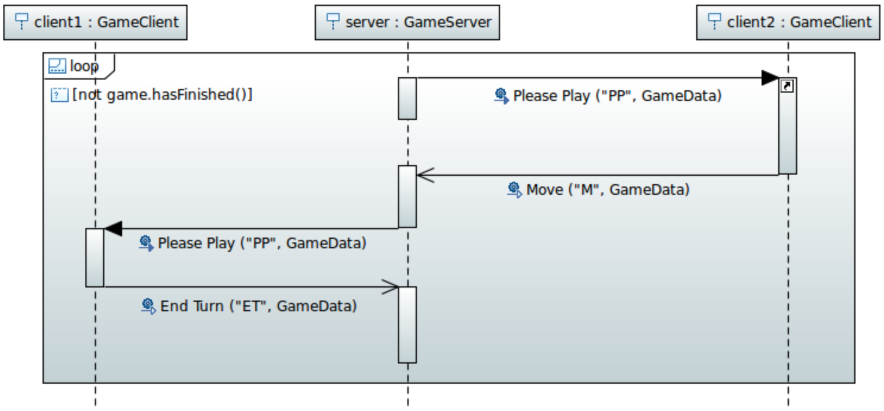
We decided to adopt a very simple Client-Server Network architecture to allow Players to compete against each other in two-player. Games. We also want to keep the Server architecture as simple as possible to avoid adding unnecessary complexity to the Project.
The reason for choosing a Network solution is to allow the players to interact without obligation to be in the same room and to provide more playing comfort, removing the need for players to switch seats at the end of each turn.
Moreover, we decided for a Client-Server-Client infrastructure to remove the need for the User to carry out any network configuration (e.g. setting up port forwarding for playing via the Internet or remembering a long IP to connect to an opponent). In fact, assuming the Server is always active and reachable via a known host address, the Players will only require an Internet connection which is required to access the game website in the first place. After establishing the communication link, the Server will pair the Players and forward any game data each Player sends to the other one.

6.2.1 Master/Slave Network

In order to keep the Server architecture very simple, and to keep the Server ignorant of the actual Game logic, we chose a Master-Slave network architecture. In our architecture, the server's only duty is to facilitate data exchange between the two players. All of the Game computation (e.g. creating the Game's initial environment, calculate score, random obstacles, etc.) is done by one of the two Game clients which will be called the "Master" Client.
When the game is started, one of the two Players' Clients is designated as the Master Client (host) and the other as the Slave Client. The Slave Client's Player will always play the first half of each turn, sending the move data to the Master Client through the server. The Master Client will then wait for the local Player to move, compute any end of turn calculations (the end of turn scores, assign resources, environment obstacles and so on), and communicate this computed game data to the other player. This process will then repeat until the game eventually finishes.
The following UML Sequence Diagram describes the interactions between the Players, their respective Game Clients and the Server when Starting a game:



Successively, while the game is in progress, the process described in the following UML Sequence Diagram is repeated, until the Game is finished.



6.2.2 Network Technology

While in the process of choosing the Network technology to adopt for the Game, we thought of the following requirements we needed it to satisfy:

- We need a Network technology that is simple for the Player to set up (does not need any configuration or have any particular requisite apart from a working Internet connection);
- We need a Network technology that allows real time communication. This is because Players will play in turns and expect the system to be ready right after the opponent plays. Examples could be TCP and UDP sockets, as opposed to *pull-type* communication over HTTP requests;
- We need a Network technology that takes care of data loss, data integrity check and retransmission. As we don't have enough resources to build it ourselves, our options are narrowed to those based on the TCP protocol;
- We need a Network technology that is able to check the health of the connection, quickly detect disconnection, attempt to reconnect automatically in case of a temporary network failure and, when reconnection is not possible (e.g. when a client goes offline, or the Wi-Fi is disconnected for a long time), promptly notify the server and the other player of the event;

After considering a couple of options, we decided on Socket.io [7] as our network infrastructure framework. Socket.io is an open source project [8] built on top of various technologies such as TCP sockets. It will allow us to create a simple Server application with NodeJS, an interpreter for the JavaScript language, while taking away all of the complexity that generally comes with a network application. The underlying framework of NodeJS will take care of all low-level duties such as keeping connections alive, correcting data-transmission errors and catching disconnection events.
Many widely-used and tested implementations for Socket.io already exist for the language of our Game, Java [9]. Being very famous in the open source community, it also comes with excellent community support and, hence, lower risk for us in the adoption of an external library.

7 Planning

7.1 Team Roles

Within any software engineering project, there are team roles allocated to individuals. The team roles tend to be based on their technical or leadership abilities, hence our team will replicate the same idea. Distributing team roles will help each team member focus on a particular aspect of the game and be able to continually progress without becoming lost or unsure with what to do next. Team members could be allocated tasks outside their role if more team members are required due to a lack of time or a high workload for a particular task. The team roles are based on those specified in OpenSeminar [6] as follows:

- Team Leader, Andrew Grierson - Andrew has previous experience in leading teams in projects through the HACS module from first year to relatively high standard. The team leader role may rotate round to another team member if someone is more knowledgeable on a particular task. The team leader role will coincide with being the secretary and the Scrum master (removing any impediments).
- Technical Lead, Alfio Fresta - Alfio has previous experience in using the proposed collaboration tools and vast amounts of knowledge in the hardware and software fields by learning in his spare time. He is the person to go to for any technical problems within the project, such as how a particular feature could be made, and knows about any hardware or software we may need for our project.

- Designer, Stefan Kokov - Stefan has previous experience in creating project plans through the HACS module and is familiar in breaking up a project into allocated slots with their priorities and dependencies.
- Lead Programmers, Alfio Fresta and Peter Lippitt - Alfio and Peter have vast amounts of knowledge in the programming field. Having two lead programmers will allow our team to split into at least two different areas to program such as one sub-team working on the gameplay and another on the map.
- Technical Directors, All - All team members will write the documentation in relation to their given task, if appropriate. Every team member will review all pieces of work before an assessment date.
- Configuration Management, Stefan Kokov. Maintaining the code base - Stefan has knowledge in maintaining working environments for projects in the past.
- Quality Assurance, Richard Cosgrove - Richard has knowledge in maintaining code and is most familiar with the requirements specification.
- Lead Artist, Yindi Dong - Yindi has a large amount of knowledge in the creative and artistic fields.

In addition, each team member has been placed into a pair to help keep each other focused on their task and to catch up their work if they happen to be unavailable for a particular reason. This would help the project to not be delayed for extended periods of time, increasing the amount of time to improve the quality of our work and keep to the specified deadlines. The pairs are as follows: Andrew Grierson and Richard Cosgrove, Peter Lippitt and Yindi Dong, Alfio Fresta and Stefan Kokov.

7.2 Software Engineering Approaches

Before we can progress into the main development of the project, we need to investigate different software engineering approaches and conclude which would be most suitable to our project needs.

The first software engineering approach we considered was the plan-driven method using IBM's Rational Unified Process (RUP) [4]. The plan-driven method involves a planned route to take throughout the software's life cycle. The user requirements are defined before starting the project and are taken into account at each stage of the lifecycle, however, this approach is best in a stable environment with a clear model of the end product. As the project will involve a lot of communication with the direct stakeholder about their ideas for the game and ours, the requirements may alter a lot over the development of the software and can be sudden, creating an unstable environment. Implementing a requirement change into an existing model would be possible in RUP, but could end up being complex and many changes would disrupt the entire process in changes to the contract, according to Boehm [5]. The entire project will also be documented for future reference and for other assessments when passing our finished project onto other teams to help them understand and assess our work.

The second software engineering approach we considered was the agile method using the Scrum approach. The agile method's main focus is on the client throughout the software development cycle. It involves short iteration cycles called "sprints" which require an initial sprint meeting to understand what work needs to be completed, daily sprint meetings to acquire updates on how the work is progressing and sprint reviews to acknowledge what we have completed. These meetings will help our team develop the software at a regular pace by collaborating routinely with each other in accordance with the client. We can also allocate tasks appropriately to work in parallel with one another and merge and review our work continuously throughout to make sure we are thinking similarly and producing a high quality piece of work. As the requirements are more volatile with a client being progressively involved, the Scrum method would allow much easier backtracking if additional requirements are acquired or any need updating.

The final software engineering approach we considered was the incremental build model using a waterfall model. The idea is to follow a strict sequence of stages throughout development: Planning, Analysis, Design, Code, Test and Maintenance. This will be easier in understanding what needs to be completed next in the project life cycle, but makes it not as flexible when planning the project. The client can review and provide feedback at any stage, however, any additional functionality to the game may cause errors in the documentation in relation to the system architecture. It is much harder for us as developers to backtrack and make changes if required.

To conclude, our group has decided to use the Scrum agile software development method due to its flexibility when adapting to the client's needs and its larger focus on meeting the requirements of the client. In addition, the agile method is more advantageous with the time constraint imposed on our team, the constraints already given in our requirements and the large amounts of UML modelling produced.

7.3 Programming Languages

We decided on using Java as our main programing language for the development of the game. One of the main reasons was that we have all had experience with it in our first year in university. It is also one of the most popular programming language with lots of open source libraries and support available. Also Java makes implementing cross platform applications really simple as it can virtually run on any type of machine and this may help us overcome some limitations of the university lab computers on which we are supposed to run the game. Other languages considered were Python and C#, however we decided that Python isn't suitable for large scale projects and the lack of experience in C# would require too much time.

7.4 Collaboration and Other Software Tools

In this section we will discuss the collaboration tools, software and programing languages we decided to use in our game:

- **Git -** Git will allow us to easily keep track of changes in the source code, UML diagrams and code documentation through time stamps and commented notes. We will be using the GitHub service, as it provides many web tools to navigate and edit code without the need to enter Git commands yourself. It will also be useful in helping developers work on different aspects of the game simultaneously, automatically resolving most of the conflicts. It has been taught in lectures, widely used in open source projects and has an easy to read and understand documentation. Git is more complicated than its counterpart SVN and might require more time to get used to, however, it is more reliable as the repository is distributed and in the unlikely event of GitHub failing we won't lose any files. There are some team members who are experienced in using GitHub who can teach the other team members in how to use it, so we would spend less time as a group learning GitHub.
- **Facebook -** It is accessible at any time with an Internet connection, making it a great way to communicate with the group at all times if there are any problems or anything needs to be discussed. Currently, all team members use Facebook as an everyday communication method to friends therefore it would be straightforward to integrate it.
- **Eclipse -** Eclipse is one of the most popular IDE's for development in Java. All of our team members have experience in it as we all worked with it in our first year programing module and is available on the university's system should we need to use them.
- **Papyrus -** It is integrated with Eclipse. It provides functionality for saving the diagrams and exporting them into an image format. It is also provides OCL validation functionalities during the implementation of the game to allow us to update our UML diagrams at the same time.
- **Google Docs / Drive** - It is a good collaboration tool for documenting work and creating charts. It allows everyone in the team to work simultaneously on a single document without the need to distribute the document locally as it uses Google's Cloud system, Google Drive. It allows us to suggest edits to each person's work.
- **Excel -** We use Microsoft Excel for the Gantt chart as it provides much more functionality, customization options and ready templates we can use. This would allow us to spend less time planning and distribute the time between the other tasks.

7.5 Gantt Chart

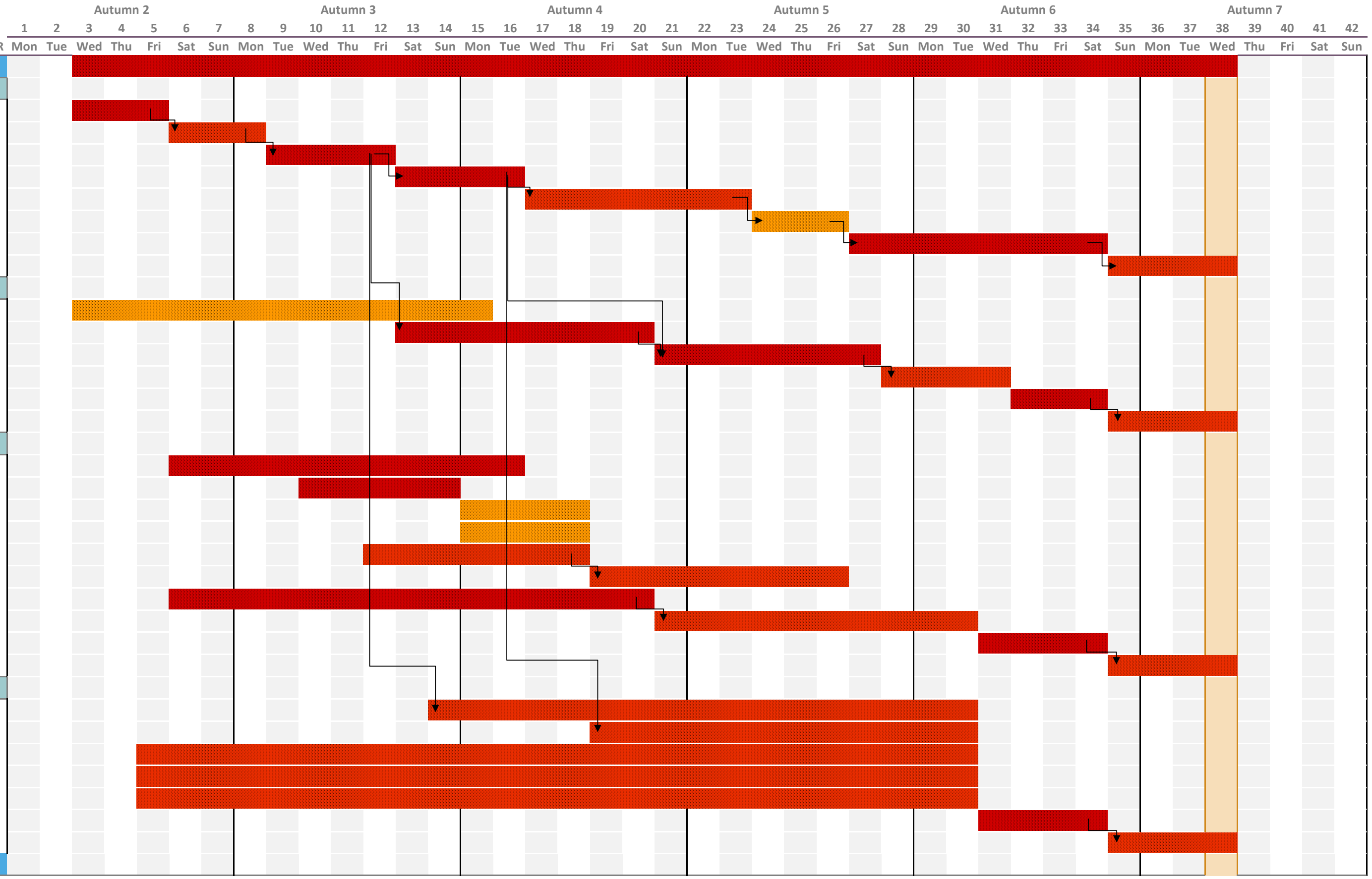
For the task scheduling and task allocation of each assessment, we have constructed a Gantt chart in a Microsoft Excel format. Assessment 1's Gantt chart is contained within this section, while the other three are in the Appendix [Appendix C]. An Excel template was used as a starting point, and was later modified to suit our team's needs. The Gantt chart is a way of helping to organise our team, in order to save time on building a complex project plan from scratch. The whole assessment was split into different categories and each category was split into several tasks. Tasks are then assigned a starting and an end date. In the chart each task is represented by a bar in the corresponding row. The part of the task where time has already passed is coloured orange and the part yet to be completed is represented by a striped line. There is also a column showing the percentage of the time allocated for a specific task that has passed. Dependencies between different tasks are represented by black arrows where the arrow points towards the dependent task. Tasks are allocated using a column in which team members responsible for it are assigned. There will be a different Gantt chart for each assessment for simplicity. Gantt charts are updated regularly in team meetings and they reflect our whole development process. Gantt charts are subject to change during the whole project.

Assessment 1

Current Date: 38

High Priority | Medium Priority | Low Priority | % Complete | No Priority

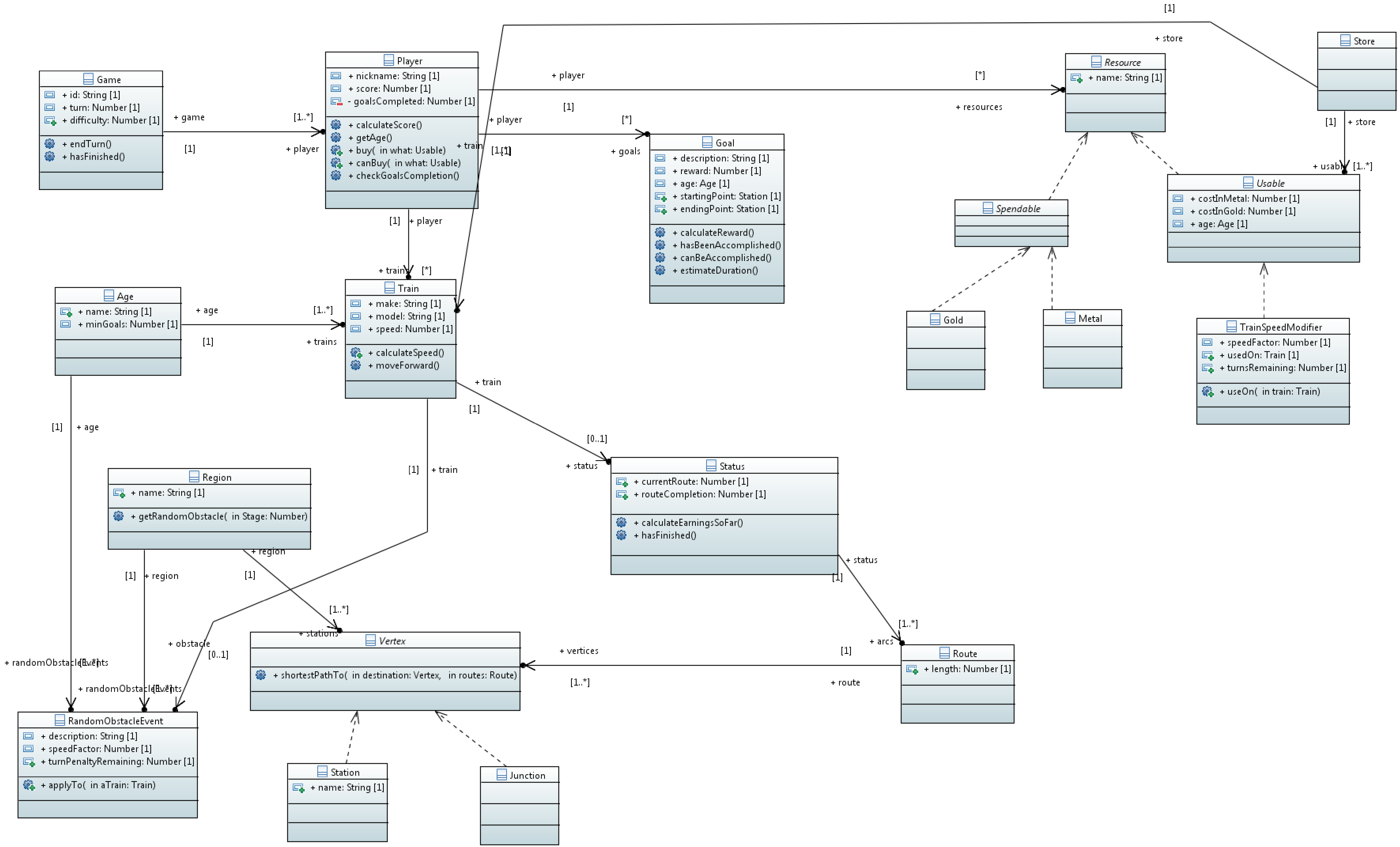
ACTIVITY	START	END	DURATION	PERCENT COMPLETE	PRIORITY	TEAM MEMBER
ASSESSMENT 1	3	38	36	100%	1	
REQUIREMENTS						
Requirements elicitation	3	5	3	100%	1	AG/RC
Constraint requirements	6	8	3	100%	2	AG/RC
User requirements	9	12	4	100%	1	AG/RC
Functional requirements	13	16	4	100%	1	AG/RC
Non-functional requirements	17	23	7	100%	2	AG/RC
Use cases	24	26	3	100%	3	ALL
Formalizing reqirements document	27	34	8	100%	1	RC
Review	35	38	4	100%	2	ALL
ARCHITECTURE						
Learning UML	3	15	13	100%	3	ALL
Identifying high level architecture	13	20	8	100%	1	AF
Building architecture models	21	27	7	100%	1	AF
Justification	28	31	4	100%	2	AF/SK
Formalizing architecture document	32	34	3	100%	1	AF
Review	35	38	4	100%	2	ALL
PLANNING						
Gantt chart, time scheduling for assessment 1	6	16	11	100%	1	SK
Gantt chart, time scheduling for assessment 2	10	14	5	100%	1	SK
Gantt chart, time scheduling for assessment 3	15	18	4	100%	3	SK
Gantt chart, time scheduling for assessment 4	15	18	4	100%	3	SK
Software engineering approach research	12	18	7	100%	2	PL
Software engineering approach justification	19	26	8	100%	2	PL
Collaboration tools research	6	20	15	100%	1	SK
Collaboration tools justification	21	30	10	100%	2	SK
Formalising planning document	31	34	4	100%	1	AG/SK
Planning review	35	38	4	100%	2	ALL
RISK ASSESSMENT						
Risk analysis based on user requirements	14	30	17	100%	2	PL/YD
Risk analysis based on functional requiremen	19	30	12	100%	2	PL/YD
Identifying project risks	5	30	26	100%	2	PL/YD
Product risks	5	30	26	100%	2	PL/YD
Business risks	5	30	26	100%	2	PL/YD
Formailising risk assessment document	31	34	4	100%	1	PL/YD
Review	35	38	4	100%	2	ALL
ASSESSMENT 1						



Assessment 1 - There are two paths with the longest path, but the critical path is the one specified here due to the dependencies:
Requirements elicitation (3 days) > Constraint Requirements (3 days) > User Requirements (4 days) > Functional Requirements (4 days) > Non-Functional Requirements (7 days) > Use Cases (3 days) > Formalising Requirements Document (8 days) > Review (4 days) = Assessment 1 Deadline reached and completed (36 days)

Risk ID	Type	Likelyhood (1-3)	Impact (1-3)	Risk Factor (1-9)	Risk	Mitigation strategy	Action	Responder
1	Project	3	2	6	**Team has a disagreement which halts progress.**	Create team decision making process to make decisions run smoothly. Any time the design needs to be altered we must have a team meeting to discuss and collect everyone's ideas. If there are differing opinions, members need to be vocal about them so that they can be discussed to ensure we get the best result possible. This will help avoid further disagreements later in development.	Have a team meeting to discuss ideas and try to reach a decision. If a decision still cannot be made, contact Tim and Fiona.	Andy
2	Project & Product	2	3	6	**Network solution is found to be too complicated to implement.**	Try to keep the network design as simplistic as possible throughout development.	Hold a team meeting and try to find a simpler solution to the problem. If none can be found, the product will need to be changed to allow local play instead (This would be a severe set back).	Alfio
3	Project	2	3	6	**Plan is overly optimistic and we are not able to keep up with it.**	Leave extra time in the team's timetable so that, if parts of the project do take longer than expected, there is time for us to complete the project by the deadline. The most important features of the project should be implemented first.	If this occurs and we do not have enough time to complete the project, we must consider removing unnecessary features from our design.	Andy and Stefan
4	Project	2	2	4	**A team member is ill and so cannot work for a short period of time (i.e. a week)**	All source code and work must be shared. At least two people at a time should be able to work on a single part of the project. We always need at least one person (e.g. the leader) to be familiar with the work each team member is doing.	Change the plan and the Gantt chart as soon as possible. Reallocate the tasks they have failed to do due to illness amongst the rest of the group. Plan the rest of the work that the ill member must do when they come back. Other members or the leader must manage the work clearly so that an ill member can continue the work more easily and quickly.	The team has been split into pairs. If one of the pair is ill, the other person in the pair will manage their work, as in the action section. If both people in the pair are ill, Andy will manage their work. The pairs are: Andy and Richard; Alfio and Stefan; Peter and Yindi.
5	Project	2	2	4	**Team member loses their work**	Team member's should back up their work whenever possible.	Try to find the backup from them or someone else. Otherwise, do the work again as quickly as possible.	The team member that lost their work
6	Product	2	2	4	**Part of the game (e.g. animation) does not work on the given equipment**	Take into account the limitations of the department's computers when making the game. Test the product on the given hardware before submitting the project.	Edit game to make it compatible.	Alfio and Peter
7	Project & Product	2	2	4	**Team changes mind on how to do something**	Be cautious when making big decisions. Consider the alternatives in detail before making a decision.	Team meeting to discuss the implications of the change, and work out what needs to be edited to accommodate the change.	Andy
8	Project	2	2	4	**Person's code is not compatible with another person's code**	Be careful when writing code that you do not stray too far from the plan. Check the class diagrams regularly and make sure you have the correct variable, method and class names. The code should be tested for compatibility regularly so that the likelihood of a major problem is reduced.	The person concerned should first try to edit their code themselves to make it compatible. Otherwise, they should seek help from the configuration manager (Stefan).	Stefan
9	Project	3	1	3	**A team member is ill and so cannot work for a day or two**	All source code and work must be shared. At least two people at a time should be able to work on a single part of the project. We always need at least one person (e.g. leader) to be familiar with the work each team member is doing.	See if they can catch up with the work they have missed. If not, another team member can help do some of the work.	The team has been split into pairs. If one of the pair is ill, the other person in the pair will manage their work, as in the action section. If both people in the pair are ill, Andy will manage their work. The pairs are: Andy and Richard; Alfio and Stefan; Peter and Yindi.
10	Project	1	3	3	**A team member is ill and so cannot work for an extended period of time (i.e. a month)**	Always leave enough extra space in the planning timetable for any accidents. All source code and work must be shared. At least two people at a time should be able to work on a single part of the project.	Change plan and Gantt chart as soon as possible. Reallocate the tasks they have failed to do due to illness amongst the rest of the group. Plan the rest of the work that the ill member must do when they come back. Other members or the leader must manage the work clearly so that an ill member can continue the work more easily and quickly.	The team has been split into pairs. If one of the pair is ill, the other person in the pair will manage their work, as in the action section. If both people in the pair are ill, Andy will manage their work. The pairs are: Andy and Richard; Alfio and Stefan; Peter and Yindi.
11	Project	1	3	3	**Team member drops out of course, meaning fewer people to work on the project**	If team member is unhappy or struggling with the course they should seek help from administration/their supervisors. Throughout the project, the leader should appease everyone's emotions and encourage everyone to continue their efforts.	Replan the rest of the work and reallocate tasks amongst the remaining team members. Try to get more help from the administrator and supervisors.	Andy; Stefan if it is Andy that left
12	Project & Product	1	3	3	**Client changes their mind on one of the requirements, meaning the product no longer satisfies all the requirements**	Leave enough extra time in the team's timetable so that the necessary changes can be incorporated into the project.	Have a meeting to discuss the new requirement and gather new ideas. Replan the work. Make necessary changes to both the program and documentation. Try to change the project in the most efficient way possible.	Andy and Richard
13	Business	1	3	3	**The game another group is making is too similar to ours**	Use original ideas in our design.	Find an administrator or supervisor to discuss the situation so that we can find a good solution for everyone.	Andy
14	Project & Product	1	3	3	**We find out some of the requirements have not been achieved late in development**	Ensure the project plan satisfies all the requirements and ensure team members follow the plan during development. Monitor the work regularly to make sure it is going according to the plan.	Have a team meeting as quickly as possible to find a solution. Make sure that the team has enough time to implement the solution.	Andy and Richard

Risk ID	Type	Likelyhood (1-3)	Impact (1-3)	Risk Factor (1-9)	Risk	Mitigation strategy	Action	Responder
15	Business	1	2	2	**During development we infringe copyright. (eg. we may use an image which is copyright protected)**	Be careful when using any online resources in the game (eg. images). Make sure all the resources we take from websites can be used for business. Prepare the resources ourselves when possible (e.g. draw pictures and illustrate icons).	Remove copyrighted materials from the program ASAP. Find or make a new resource instead.	Yindi
16	Project & Product	1	2	2	**Find out part of the game design is too difficult to program**	Design the game based on the technology we are using (i.e. the programming languages used), the time we have available and the skill of the team members.	Have a team meeting to go through the design to find a solution. First try to find out if there is an easier method we can use to solve the problem. If not, consider removing the feature (this may mean we have to find an alternative way to satisfy a requirement).	Peter and Alfio
17	Project	2	1	2	**Team member's computer stops working**	Make sure antivirus software is installed and updated. Be careful when carrying laptops.	Use department's computers instead until a repair is made.	The owner of the computer
18	Product	2	1	2	**Find that the game is unbalanced or unfair during development**	Keep the idea of balance in mind during development. Be careful not to implement anything which would give a single player an unfair advantage or make it impossible for a player to catch up to the leading player.	Have a team meeting to decide on the best course of action to take to make the game more balanced. Edit the code as necessary to ensure the game is balanced.	Richard
19	Project	1	1	1	**Collaboration system (GitHub) goes down, stopping us from both accessing and uploading to the project**	Prepare more than one way to gather the project together.	A team member will set up a GIT server while GitHub is offline.	Alfio
20	Project	1	1	1	**Communication system (Facebook) goes down**	Prepare more than one method of communication.	Use an alternative communication system instead (e.g. email, google drive).	Alfio
21	Project	1	1	1	**Team member is unable to attend a meeting (e.g. they leave university for a while)**	Avoid doing things that may negatively influence the project planning where possible.	If the meeting is very important then have the meeting online, such as using Facebook or possibly Skype. If it is not essential that every single team member attends then have the meeting as normal and update the person whom could not attend.	Andy
22	Project & Product	1	1	1	**An idea is lost when implementing a change in design late in development**	Always note down any ideas we have during meetings. Tick off the ideas that we have already implemented and check the rest every time we having a meeting about the design.	Change the design and plan as quickly as possible. Try to add the idea into the current plan or program.	Stefan
23	Product	1	1	1	**A resource is lost (e.g. picture, icon, background)**	Check the game folders every time they are copied or moved.	Locate the original file and copy it.	Yindi



Game
+ id: String [1]
+ turn: Number [1]
+ difficulty: Number [1]
+ endTurn()
+ hasFinished()
+ game
[1]
[1..*]
+ player
Player
+ nickname: String [1]
+ score: Number [1]
- goalsCompleted: Number [1]
+ calculateScore()
+ getAge()
+ buy(in what: Usable)
+ canBuy(in what: Usable)
+ checkGoalsCompletion()
+ player
[*]
+ resources
Resource
+ name: String [1]
[1]
+ store
Store
[1]
+ store
+ usable
[1..*]
+ player
[1]
[*]
+ goals
+ train
[1..*]
Goal
+ description: String [1]
+ reward: Number [1]
+ age: Age [1]
+ startingPoint: Station [1]
+ endingPoint: Station [1]
+ calculateReward()
+ hasBeenAccomplished()
+ canBeAccomplished()
+ estimateDuration()
Usable
+ costInMetal: Number [1]
+ costInGold: Number [1]
+ age: Age [1]
Spendable
Gold
Metal
TrainSpeedModifier
+ speedFactor: Number [1]
+ usedOn: Train [1]
+ turnsRemaining: Number [1]
+ useOn(in train: Train)
[1]
+ player
+ trains
[*]
Age
+ name: String [1]
+ minGoals: Number [1]
+ age
[1..*]
[1]
+ trains
Train
+ make: String [1]
+ model: String [1]
+ speed: Number [1]
+ calculateSpeed()
+ moveForward()
+ train
[1]
[0..1]
+ status
[1]
+ age
[1]
+ train
Region
+ name: String [1]
+ getRandomObstacle(in Stage: Number)
+ region
[1]
+ region
[1]
Status
+ currentRoute: Number [1]
+ routeCompletion: Number [1]
+ calculateEarningsSoFar()
+ hasFinished()
+ status
[1]
[1..*]
+ arcs
[1..*]
+ stations
+ obstacle
[0..1]
+ randomObstacleEvents
[1..*]
+ randomObstacleEvents
[1..*]
Vertex
+ shortestPathTo(in destination: Vertex, in routes: Route)
+ vertices
[1..*]
[1]
+ route
Route
+ length: Number [1]
RandomObstacleEvent
+ description: String [1]
+ speedFactor: Number [1]
+ turnPenaltyRemaining: Number [1]
+ applyTo(in aTrain: Train)
Station
+ name: String [1]
Junction

Bibliography

Assessment 1

1. CIO Staff, “How to define the scope of a project”, http://www.cio.com.au/article/401353/how_define_scope_project/, [Nov. 2nd 2014]
2. JKinfoLine, “Software Project Planning”, http://www.jkinfoline.com/software-project-planning.html [Nov. 2nd 2014]
3. IEEE Computer society, (1998, June 25th), “IEEE Recommended Practice for Software Requirements Specification” [PDF]. Available: http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf [Nov. 4th 2014]
4. Python Software Foundation, “The Python Language Reference” [PDF]. Available: https://docs.python.org/2/reference/introduction.html [Nov. 10th 2014]
5. Oracle, (2013, March 3rd), “The Java Language Specification Java SE 7 Edition” [PDF]. Available: https://docs.oracle.com/javase/specs/jls/se7/jls7.pdf [Nov. 10th 2014]
6. IBM, “Rational Unified Process” [PDF]. Available: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf [Nov. 11th 2014]
7. F. Polack. Class Lecture, Topic “Software Engineering Introduction”, University of York, York, Oct. 9th 2014.
8. OpenSeminar, “Team Management”, http://openseminar.org/se/modules/11/index/screen.do [Nov. 11th 2014].
9. G. Rauch, “Socket.IO 1.0 IS HERE”, http://socket.io/ [Nov. 3rd 2014]
10. GitHub, “Automattic / socket.io”, https://github.com/Automattic/socket.io [Nov. 3rd 2014]

GitHub, “Gottox / socket.io-java-client”, https://github.com/Gottox/socket.io-java-client [Nov. 3rd 2014]