Software Engineering Project:

Technopoly

Group 3:
Barbara Murtland
Colette Casey
Dave Kennedy
Ismael Florit
Tim Lewis

CSC7053 Peer Assessment: Technopoly

This Assessment Document is intended to provide you and your assessor with an overview of each group member's involvement delivery of the CSC7053 Project.

Each group should complete <u>one</u> Assessment Document and its content must be agreed by all group members. The completed form should be included as hard copy at the start of your group's report. ***Don't forget to fill in the Group Number.***

There are two main parts to the Assessment Document – the Evaluation and the Declaration. Both parts must be completed – otherwise your group's report will not be marked. Arrange a group meeting to discuss the <u>evaluation, and</u> see the note below!

Evaluation	Group Number:	Group 3		
Name	Contribution of time and effort[1]	Contribution to team-working and motivation[1]	Contribution to the deliverables [1,2]	Peer Score (Range 85 – 115)
Barbara Murtland	5	5	5	115
Colette Casey	5	5	5	115
Dave Kennedy	5	5	5	115
Ismael Florit	5	5	5	115
Tim Lewis	5	5	5	115

[1]Values: 1 = Less than average; 2 = Slightly less than average; 3 = Average; 4 = Slightly more than average; 5 = More than average
[2]This value should consider contributions in the round – direct contributions to required deliverables, and contributions that have made the deliverables possible.

Declaration		
"I declare that I have read the Queen's University regulations on plagiarism, and that any contribution I have made to this assignment is my own original work, except for any elements that I have clearly attributed to third parties. I understand that this assignment will be subject to an electronic test for plagiarism and will also be subject to the University's regulations concerning late submission if it is received after the deadline."		
Name	Date	Confirmation *(use the words shown in the example below!)*
Barbara Murtland	14/03/2019	*I agree to the terms of the declaration*
Colette Casey	14/03/2019	*I agree to the terms of the declaration*
Dave Kennedy	14/03/2019	*I agree to the terms of the declaration*
Ismael Florit	14/03/2019	*I agree to the terms of the declaration*
Tim Lewis	14/03/2019	*I agree to the terms of the declaration*

A note on the Evaluation:

The Contribution columns in the Evaluation table are intended to help group members quantify each other's input to the project, before they award agreed Peer Scores. There will not necessarily be a precise correlation between the Peer Score and the Contribution values. However, high Contribution scores, as an indicator of the importance of the group member's work to the success of the assignment, should normally result in a high Peer Score for a group member. <u>Likewise</u> a low Peer Score would be the expected outcome if Contribution values are low.

Each group member's overall score for the project will be calculated according to the following formula, where S_i is Group Member i's overall score, P_i is the peer score received by Group Member i, N is the number of members in the group, and M is the raw mark awarded to the assignment by the assessor.

$$S_i = \frac{P_i}{\frac{1}{N}\sum_{j=1}^{N} P_j} \times M$$

The following guidelines will help you award appropriate peer scores. If the group agrees that Group Member 1's overall contribution to the project was much weaker than the average contribution, a peer score of 85 would be appropriate for Group Member 1. If Group Member 1's contribution was much stronger than average, consider a peer score of 115. If Group Member 1 did what was expected and shared the effort equally with their fellow group members, they could expect to receive a Peer Score of 100. Any mark within the range 85 – 115 will normally be accepted by the module Lecturer. Marks outside this range may require that the Group discuss its decision with the module Lecturer in order to agree a fair distribution of marks. Where group members cannot agree a distribution, the module Lecturer's judgement will be final. ***Please inform the module Lecturer if a group member has left your group or has ceased to play an active role in the group.***
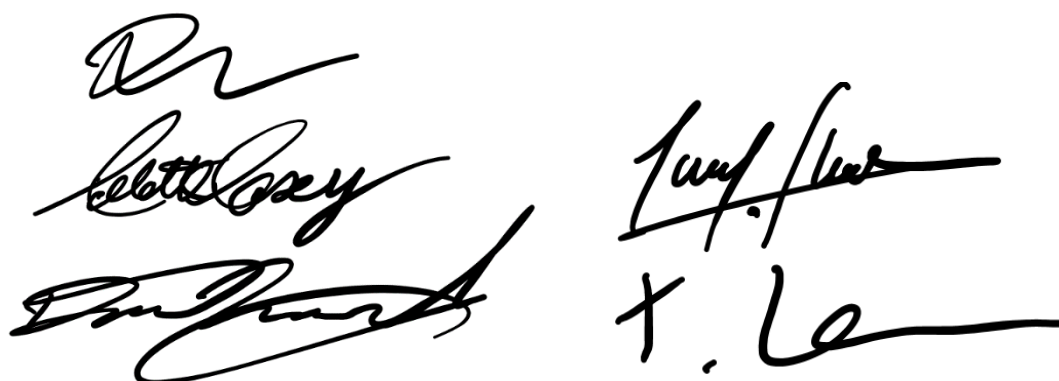
Table of Contents

Appendix 1

A. Photos of use case and sequence diagrams
B. Full user guide and graphical representation of board
C. Test Plan
D. Test Plan – Junit tests conducted
E. Glossary of Key Terms

Appendix 2

A. Weekly Team Meeting Minutes

Introduction

The aim of this report is to document the planning, design, development and implementation of a Monopoly style interactive game known as Technopoly. Technopoly has been developed in Java, using the console to create a textual interface. A use case driven development process was adopted by the team in order to effectively manage and meet the requirements established by the customer.

The group sought to follow the best practices of software engineering by adhering to the key stages in the software development process. The Requirements Analysis section details the group's analysis of the system requirements using use case descriptions and a use case diagram. This proved to be a valuable tool for defining the Technopoly system's external behaviour. The Realisation section of the report outlines how the most important use case descriptions were used to create detailed sequence diagrams which reflect the high-level interactions between use cases.

Finally, the Design section highlights how the group's focus shifted from the requirements of the system to the structure of the Technopoly software, transforming the analysis model into a functioning game. The responsibilities of each object were considered in more detail. The UML Class Diagram illustrates the final design of the working Technopoly game. A key goal of the Design section was to ensure high cohesion and low coupling within the game system. A test plan was devised and JUnit tests were implemented alongside the development of the software, see Appendix 1. Regular scrum meetings were held and documented to keep development on track, see Appendix 2.

Requirements Analysis

The team met to fully discuss the uses cases which were needed, running through a mock up the game. The use cases were then split amongst the team to write and develop before being collated and referenced.

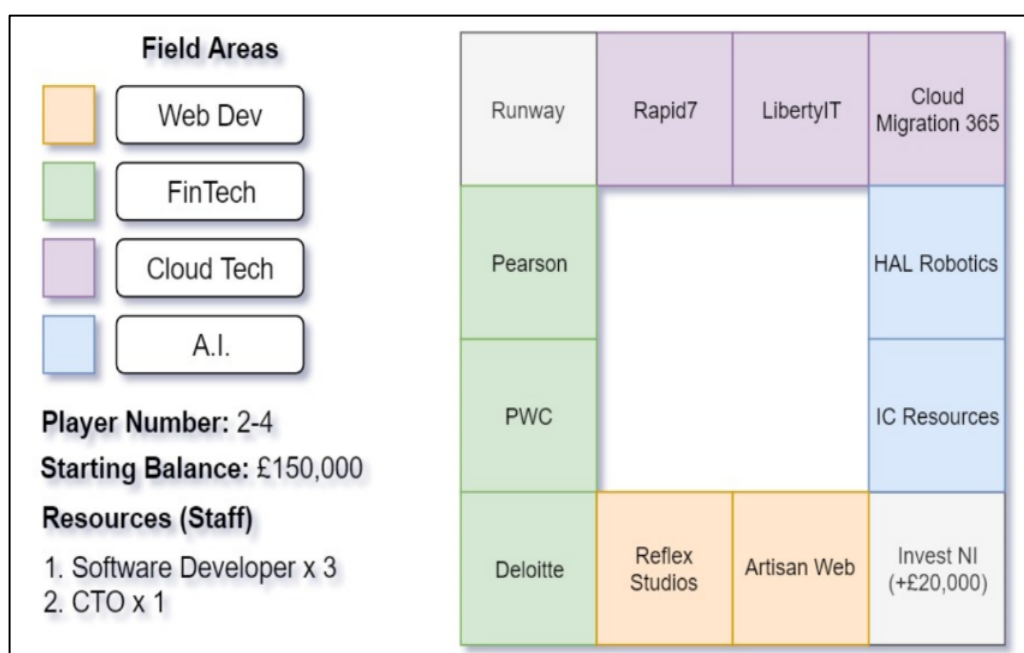## i. Game Guide and Graphical representation (TL, IF)

Figure 1 Technopoly Board

Technopoly Guide

Object
- The object of the game is to become the wealthiest player through buying start-ups, hiring staff and selling start-ups.

Preparation
- The game is allowed up to four players.
- Each player enters their username Players cannot share the same username.
- The players are then randomly assigned a starting order in the game.
- Each player is given £150,000.

The Play
- Each player starts the game on the InvestNI square.
- The virtual dice is rolled for Player 1 and they are moved along the board the number of spaces indicated by the dice.
- After a player has completed their turn, the next player takes their turn.
- Two or more players can be on the same space at the same time.
- According to the space you land on, you may be entitled to purchase the start-up or obliged to pay a licence fee.
- If a player's resources have changed, they will be informed of their new balance and given a reason for the change.
- If you throw doubles, you are moved as usual, the sum of the two virtual dice, and are subject to any privileges or penalties pertaining to the space on which you land. Retaining the virtual dice, throw again. If a player throws doubles three times they are taxed £1000 and are not permitted to roll again. <u>Furthermore</u> the player is still subject to any privileges or penalties pertaining to the space on which they land.
- When one player runs out of money and cannot pay a licence fee to another player, the player with the most resources is declared the winner.
- If a player no longer wishes to play they can terminate the game and the game ends. The player with the most resources is declared the winner. If the player with the most resources terminates the game, the player with the second highest amount of resources is declared the winner.

"Invest NI"
- Each time a player passes over the "Invest NI" space they receive a £20,000 investment.
- The £20,000 investment is only paid once each time around the board.

Purchasing Start-up
- Whenever you land on an unowned start-up you may buy that start-up at its printed price. For a full list of prices please see the Technopoly Pricing Guide included in the appendix of the report.

Paying licence Fee
- When you land on a start-up owned by another player, the system collects a licence fee from you in accordance with the cost assigned to the start-up.
- It is advantageous to have software developers or CTOs working at the start-ups because the licence fee is much higher than for unstaffed start-ups.

"Runway"
- A player landing on the "Runway" space does not receive any money, start-up or reward of any kind.
- This is just a resting place.

Hiring Staff
- When you own all the start-ups in a field you may hire staff and employ them at those start-ups.
- Once you hire staff for one of your start-ups you are no longer allowed to sell the start-up or any other start-up from that field to another player.
- There are two types of staff that can be hired for each start-up. They are the Software Developer and the Chief Technology Officer (CTO).

Software Developers
- If you hire a software developer, you may employ him in any of the start-ups in the field you own.
- The price you must pay to hire each software developer will be highlighted to the player at the time of purchase and varies depending on the field.
- You may hire up to a limit of three software developers per start-up.

Chief Technology Officers (CTOs)
- When a player has three software developers working at each start-up in a complete field he/she may hire a Chief Technology Officer (CTO) and employ them in any start-up of that field.
- Only one CTO can work at any one start-up.

Takeover Start-up
- If a player wishes to purchase a start-up from another player, they may attempt a takeover by making a request to the owner of the start-up.
- The owner of the start-up must then choose whether to approve the takeover attempt or deny it.
- If approved, the owner will be paid a 20 percent mark-up on the original cost of the start-up by the player wishing to takeover the start-up.

Bankruptcy
- You are declared bankrupt if you owe more than you can pay to another player. Once a player is declared bankrupt the game ends and the player with the most resources is declared the winner.

Miscellaneous
- No player may borrow from or lend money to another player

Figure 2 Technopoly Game Guide

ii. Use Case Descriptions (BM collated. CC, TL, DK, IF contributed)

The use cases below describe the externally visible behaviours and structures of Technopoly. Each use case represents a task or interaction with the game performed by the actor(s) listed. These descriptions therefore, will form the basis for the system requirements. Please refer to the Use Case Diagram. In order to assist the reading of the below descriptions all use cases have been listed numerically. All extension and inclusion points have been assigned a letter, red for Extension Points, blue for inclusions. Actors have been italicised.

Use Case 1	Starts Game
Summary	Establish the number of players(2-4) and their names. Randomly generate the order of play. Game starts.
Actors	*Player1, Player2,* (*Player3, Player4* are optional)
Triggers	2-4 players wish to set-up Technopoly
Pre Conditions	There are a minimum of 2 and a maximum of 4 players
Post Conditions	• The valid number of players selected is displayed. • A random order is generated. The players can either view the rules [Use case 2 Views Rules] or press start to begin the game.
Flow	1. The *player(s)* are prompted to enter the number of valid players(between 2 and 4 players) 2. The *players* will be prompted in turn to enter a valid name (2-15 characters) 3. Once all player names are added to the game a random order is assigned to the players and the game begins.
Alternative Flow	• At flow point 1:- If the number of players is invalid, the *player(s)* will be prompted again and reminded that the valid number of players is between 2 and 4 • At flow point 2:- If the name of the player is invalid or already exists in the game, the *player* will be prompted again and reminded that a valid name is between 2 and 15 characters(including – and ') excluding whitespace.
Extension Points	None

Inclusions Points	None
Use Case 2	**Views Rules**
Summary	The *player* views the rules of the game before selecting 'Start Game'.
Actors	Player(s)
Triggers	[USE CASE 1 Start Games] After selecting entering a valid number of players and entering valid name the players can view the rules.
Pre Conditions	• The game has not yet started
Post Conditions	• The rules of the game are displayed.
Flow	1. The *player* selects the "View Rules" option. 2. The system displays the rules of the game to the player(s).
Alternative Flow	The players do not select 'View Rules' in which case the only other option available to them is to 'Start Game'.
Extension Points	None
Inclusions Points	None
Use Case 3	**Takes Turn**
Summary	*Player* rolls dice and moves to the corresponding space. Three options are then posed to the player
Actors	Player(s)
Triggers	The game starts OR the player before has ended their turn
Pre Conditions	• [USE CASE 1 Starts Game] :- The game is set up • It's the *player* in question's turn
Post Conditions	• [USE CASE 11 Ends Turn] :- The *player* must select 'end turn' once they have completed their turn. The next player will repeat this operation.
Flow	1. The player is shown: Player Number, Name, Balance, Properties Owned including Staff, the dice score thrown and the space they have landed on. Eg, Player 1 | Joe Bloggs | £xx.xx | Owns: [AProperty, BProperty] | Rolled [2,6] | Landed on X. 2. The *player* is 'moved' to the corresponding space. Depending on the space landed on the *player* can: 3. Purchase Start-up [Extension D] 4. Pay Licence fee [Extension B] 5. The *player* can view menu [Inclusion A]
Alternative Flow	1. At flow point 1 and 2:- If the *player* rolls a double (same value on both dice) the *player* is moved to the corresponding space. They finish that turn and take another go immediately. 2. At flow point 1 and 2:- If doubles is rolled three times in a row; on the third turn the *player* suffers a penalty (tax or jail depending on the game iteration) 3. At flow point 4:- If the *player* has insufficient funds to pay a Licence fee, then that *player* is declared bankrupt and the game is over. The player with the highest net worth is deemed the winner. [Extension C]
Extension Points	D. Purchase Start-Up – [Use Case 7] B. Pay Licence Fee –[Use Case 5] C. Declares Bankruptcy – [Use Case 6] A. Lands on Runway – [Use Case 4] E. Passes Invest NI – [Use Case 8]
Inclusions Points	A. Views Menu – [Use Case 9]
Use Case 4	**Lands on Runway**
Summary	Player lands on the 'Runway' field
Actors	*Players*
Triggers	The Player has moved onto the "Runway" field
Pre Conditions	• It's the *player* in question's turn [Use Case 3 Takes Turn]
Post Conditions	• The *player's* turn continues
Flow	1. The *player* lands on "Runway". This space cannot be owned. No licence fee is applicable.
Alternative Flow	None
Extension Points	None
Inclusions Points	None
Use Case 5	**Pays Licence Fee**

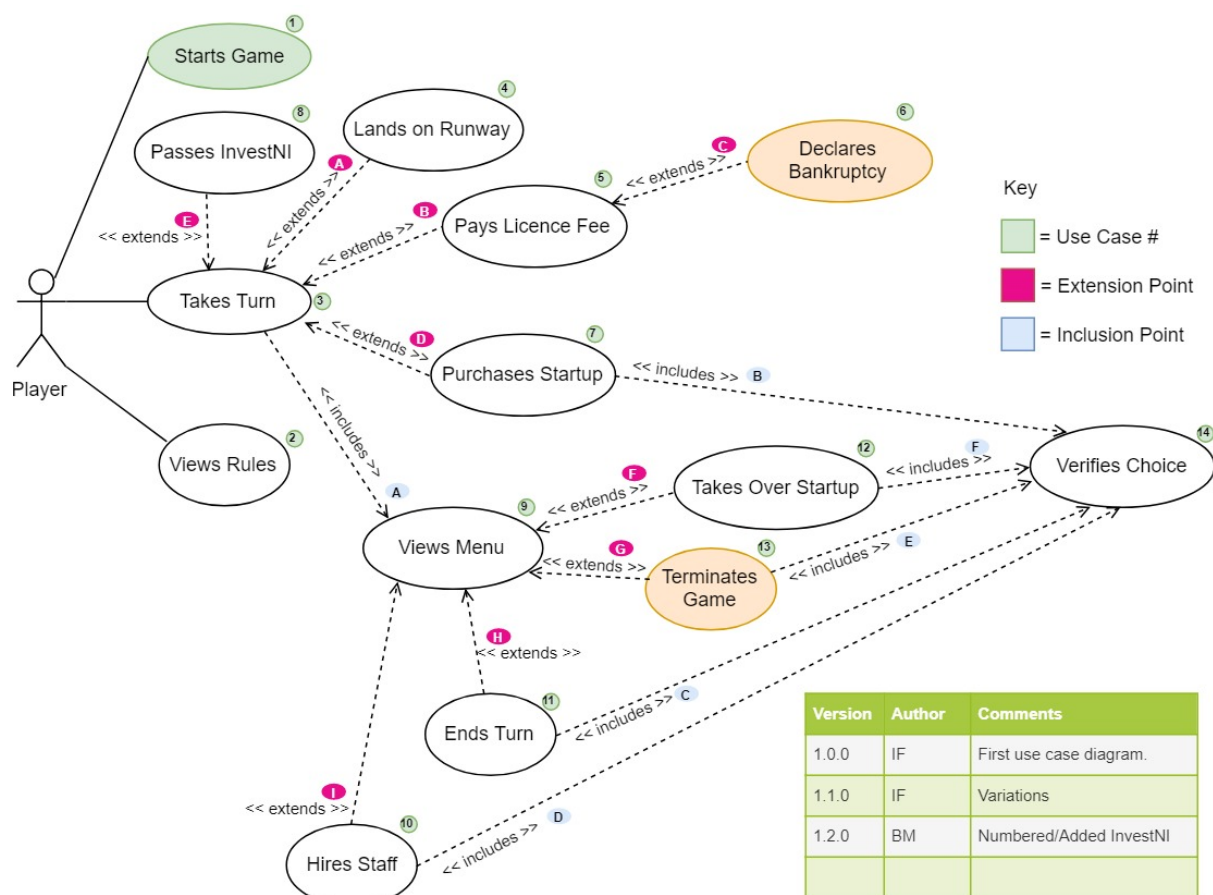| Summary	The player lands on a space with a start-up owned by another player. The player must pay the appropriate licence fee to the other player.
Actors	Players
Triggers	A Player lands on a square owned by another player.
Pre Conditions	• It's the player in question's turn [Use Case 3 Takes Turn] • The start-up on the space is owned by another player • The player has sufficient funds to pay the licence fee
Post Conditions	• The player's balance is updated to reflect paying the licence fee • The new balance is displayed by the system • The player has sufficient funds to continue to play (Balance >=0)
Main Flow	1. The player lands on a square for a start-up owned by another player 2. The system displays the licence fee and the other players name/number 3. The player must have the funds to pay the appropriate licence fee 4. Player's balance is reduced 5. Other players balance is increased 6. The updated balances for each player are displayed by the system
Alternative Flow	1. At main flow point 3:- the player does not have sufficient funds to pay the licence fee [Extension C].
Extension Points	C. Declares Bankruptcy – [Use Case 6]
Inclusions Points	None
Use Case 6	**Declares Bankruptcy**
Summary	Player is declared bankrupt by the system.
Actors	Player(s)
Triggers	The player lands on an start-up owned by another player without sufficient funds to pay the licence fee.
Pre Conditions	• It's the player in question's turn [Use Case 3 Takes Turn] • The player has landed on any owned start-up that is not their own [Use Case 5 Pay Licence Fee] • The player has less funds than necessary to pay a Licence Fee.
Post Conditions	• The player is declared bankrupt. The game ends and a winner is declared.
Flow	1. The player lands on an owned start-up that is not his own. 2. When unable to pay the fee, the system declares the player bankrupt. 3. The system ends the game and declares the winner (player with most resources).
Alternative Flow	None
Extension Points	None
Inclusions Points	None
Use Case 7	**Purchase Start-up**
Summary	The player lands on a space with an available start-up. If they decide to purchase the start-up, they pay the listed price. If they buy all the start-ups in the same colour group(field), they have a tech monopoly (technopoly) and can charge double the listed license fee.
Actors	Player(s)
Triggers	Player lands on a new space with an available start-up
Pre Conditions	• It's the player in question's turn [Use Case 3 Takes Turn] • The start-up on the space is available to purchase • The player has sufficient funds to buy the start-up
Post Conditions	• The player's balance is updated to reflect the purchase • The player's properties are updated to list the company • The player's properties are updated if they own a complete field

Main Flow	1. The player lands on a space with an available start-up
	2. Player decides to purchase start-up by selecting "Purchase" from the menu [Inclusion A]
	3. The system checks the player has sufficient funds, then:
	    I. Reduces player's balance by the listed price
	    II. Adds the start-up to the player's list of companies
	    III. Displays a message to confirm the purchase and the new balance
	4. The player can either return to the menu/end their turn [Extension H]

Alternative Flow	1. At main flow point 1:- The *player* decides not to purchase the start-up, Views Menu and ends turn [Inclusion A and Extends H] 2. At main flow point 3:III:- If the purchase completes the *player's* ownership of a colour field, the system will display a message to announce this. The player is brought to the menu [Inclusion A] 3. At main flow point 3:- The *player* does not have sufficient funds. The System displays a warning message. The player cannot purchase the space. Player must end their turn. [Extension H]
Extension Points	H. Ends Turn [Use Case 11]
Inclusions Points	A. Views Menu - [Use Case 9]
Use Case 8	**Passes Invest NI**
Summary	The *player* receives a £20,000 investment each time they are on or pass the Invest NI space. Their balance is updated to reflect the changes
Actors	*Player*
Triggers	*Player* is on or passes the Invest NI space
Pre Conditions	• The game has been started [Use Case 1 Starts Game] • It is the *player's* turn [Use Case 2 Takes Turn]
Post Conditions	• The *player's* balance updates to reflect an investment of £20,000
Main Flow	1. The *player* is on or passes the Invest NI space 2. The player receives an investment of £20,000
Alternative Flow	None
Extension Points	None
Inclusions Points	None
Use Case 9	**Views Menu**
Summary	Player views menu, which describes the options available to take during their turn.
Actors	*Player*(s)
Triggers	• It is the *player's* turn [Use Case 2 Takes Turn] • Player has finished any actions other than "Quits Game".
Pre Conditions	• It is the *player's* turn [Use Case 2 Takes Turn] • The player has 'rolled' and if necessary paid any licence fees applicable
Post Conditions	• Player will either perform the selected action or quit the game.

Flow	1. The *player* views the menu.
	2. The player may choose any option on the menu:
	  i. Purchases Start-up – [Extension D].
	  ii. Takes Over Start-up – [Extension F].
	  iii. Hires Staff – [Extension I].
	  iv. Ends Turn – [Extension Point H].
	  v. Terminates Game – [Extension G].

Alternative Flow	None
Extension Points	D. Purchases Start-up – [Use Case 7] F. Takes Over Start-up – [Use Case 12] I. Hires Staff – [Use Case 10] H. Ends Turn – [Use Case 11] G. Terminates Game – [Use Case 13]
Inclusions Points	None
Use Case 10	**Hires Staff**
Summary	*Player* hires a new member of staff for their Start-up space. The player must own all spaces in that field. They can then hire a software developer OR if three developers are present on the space, then they can purchase one CTO to place on that space. Each space can only have a maximum of three developers OR one CTO present.
Actors	*Player*
Triggers	*Player* selects 'Hire Staff' from the menu[Use Case 9 Views Menu]
Pre Conditions	• It is the *player's* turn [Use Case 2 Takes Turn]. • The player is on <u>any</u> space. • The player owns all corresponding spaces relating to that field.

	• The player has sufficient funds to employ the corresponding Developer/CTO.
	• The number of staff on the relevant field will not exceed the limit of staff once this hiring process is complete.
Post Conditions	• Player will have relevant amount of money debited from their account.
	• Player will have a new member of staff on the desired field.
	• Player will be shown new account balance.
	• Player will be told they have hired the relevant staff member at the relevant start-up.
Flow	1. The *player(s)* selects the 'Hires Staff' menu option.
	2. The player selects the area they wish to develop.
	3. The player confirms they wish to proceed with the hiring of staff [Inclusion D]
	4. The relevant amount of money is deducted from the players account.
Alternative Flow	1. At flow point 1:- If the player has insufficient funds to hire a member of staff at this start up a message is displayed stating so and player is returned to view menu [Use Case 9 Views Menu].
	2. At flow point 2:- If the player already has too many staff hired at the startup in question; a message is displayed stating so and the player is returned to the Views Menu [Use Case 9 Views Menu].
Extension Points	None
Inclusions Points	D. Verifies Choice [Use Case 14]
Use Case 11	**Ends Turn**
Summary	A player ends their turn in the game. The game continues.
Actors	Player(s)
Triggers	The *player* selects the "end turn" option from the menu.
Pre Conditions	• It is the *player's* turn [Use Case 2 Takes Turn].
	• The *player* has already 'rolled' the dice and landed on a new space.
	• If required, the player has paid the necessary licence fee to another *player*.[Use Case 7 Pays Licence Fee]
	• The *player* has selected the "view menu" option [Use Case 9 Views Menu].
Post Conditions	• The player's turn ends
	• The system produces a message declaring that it is the next player's turn.
Flow	1. The *player* chooses the "view menu" option. [Inclusion A]
	2. The *player* chooses the "end turn" option.[Extension H]
	3. The system asks the *player* if they are sure they want to end their turn. [Inclusion C]
	4. The *player* chooses the "yes" option.
	5. The *player's* turn ends.
Alternative Flow	1. At flow point 4:- The player chooses the "No" option and they are returned to the menu. Their turn has not ended.
Extension Points	H. Ends Turn [Use Case 11]
Inclusion Points	C. Verifies Choice [Use Case 14]
	A. Views Menu [Use Case 9]
Use Case 12	**Takes over Startup**
Summary	Player stages a Takeover, offering the initial price of the start-up(they wish to Take Over) plus 20% on top. Takeovers need to be agreed by **both** parties.
Actors	*Players (Always 2 players)*
Triggers	The *Player(whose turn it is)* has selected the "Trades Start-up" option on the Menu.
Pre Conditions	• It is the *player* who wishes to perform the takeover's turn – [Use Case 3].
	• The player selects "Takeover Start-up" option from the play menu.
Post Conditions	• The current player has the required money for the transaction removed from their inventory.
	• The relevant player, who sold their start-up with the current player, has the start-up in question removed from their inventory.(If they agree to the Take Over).
	• The current player gains the Start-up from the other player and it is added to the current players inventory.
	• The relevant player, who sold their start-up, gains the relevant amount of money and it is added to their inventory.
Flow	1. The current *player* selects the other player they wish to purchase a start-up from in a sub menu (each player is numbered, current player is not shown in sub menu).

	2. The current player selects the start-up they wish to acquire from the previously selected player from a sub menu (each owned start-up is numbered).
	3. The current player is prompted with a message displaying the cost of the property they are about to purchase and asked if they wish to proceed.[Inclusion F]
	4. The other player involved in the transaction is asked to confirm the sale of said startup. [Inclusion F]
	5. The trade is completed.
	6. The current player is shown the Menu – [Use Case 9]
Alternative Flow	1. At flow point 1:- The current player may choose to cancel the takeover from the provided sub menu.
	• Current player Views Menu [Use Case 9].
	2. At flow point 2:- The current player may choose to cancel the takeover from the provided sub menu.
	• Current player Views Menu [Use Case 9].
	3. At flow point 3:- The current player may choose to cancel the trade from the provided verification message.
	• Current player Views Menu [Use Case 9].
	4. At flow point 4:- The other player involved may choose No when prompted if they wish to proceed with the takeover.
	• Current player Views Menu [Use Case 9].
Extension Points	None
Inclusions Points	F. Views 'Verifies Choice' [Use Cases 14].
Use Case 13	**Terminates Game**
Summary	A player terminates the game and the player with the most resources is declared the winner.
Actors	Player(s)
Triggers	The *player* selects the "terminate game" option from the menu.
Pre Conditions	• It is the *player's* turn [Use Case 2 Takes Turn].
	• The *player* has already 'rolled' the dice and landed on a new space.
	• If required, the *player* has paid the necessary licence fee to another player.[Use Case 7 Pays Licence Fee]
	• The *player* has selected the "view menu" option [Use Case 9 Views Menu].
Post Conditions	• The system produces a message declaring that a *player* has terminated the game and the game is over.
	• The *player* with the most resources is declared the winner.
Flow	1. The *player* chooses the "View Menu" option.[Inclusion A]
	2. The *player* chooses the "Terminate Game" option.
	3. The system asks the *player* if they are sure they want to terminate the game. [Inclusion E]
	4. The *player* chooses the "yes" option.
	5. The game is terminated, and the winning *player* is displayed
Alternative Flow	• At flow point 3:- The player chooses the "No" option and is returned to the menu.
Extension Points	None
Inclusions Points	A. Views Menu [Use Case 9]
	E. Verifies Choice [Use Case 14]
Use Case 14	**Verifies Choice**
Summary	The *player* verifies a selected action from the menu presented to them. They can only select yes or no. No returns them to the menu. Yes carries out the required action.
Actors	*Player(s)*
Triggers	• The *player* attempts to use the "Purchase Start-up" option.[Use Case 7]
	• The *player* attempts to use the "Takes Over Start-up" option.[Use Case 12]
	• The *player* attempts to use the "Terminate Game" option.[Use Case 13]
	• The *player* attempts to use the "Ends Turn" option. [Use Case 11]
	• The *player* attempts to use the "Hires Staff" option. [Use Case 10]
Pre Conditions	• The *player* attempts to use the "Purchase Start-up" option.[Use Case 7]
	• The *player* attempts to use the "Takes Over Start-up" option.[Use Case 12]

	• The *player* attempts to use the "Terminate Game" option.[Use Case 13] • The *player* attempts to use the "Ends Turn" option. [Use Case 11] • The *player* attempts to use the "Hires Staff" option. [Use Case 10]
Post Conditions	The selected option is confirmed and the system notifies the *player* with the corresponding confirmation message.

Flow	1. The player attempts to use one of the following options in the menu:
	   i. "Purchase Start-up"[Use Case 7]
	   ii. "Takes Over Start-up" Use Case 12]
	   iii. "Terminate Game" Use Case 13]
	   iv. "Ends Turn" [Use Case 11]
	   v. "Hires Staff" [Use Case 10]
	2. The system prompts the user to verify the selected choice.
	3. The *player* confirms the choice.
	4. The system displays a confirmation message to the *player.*

Alternative Flow	1. At flow point 3:- The *player* may choose to cancel the selected choice. • Current player Views Menu [Use Case 9].
Extension Points	None
Inclusions Points	None

iii. Use Case Diagram (BM/IF drew. TL, DK, CC contributed)

Version	Author	Comments
1.0.0	IF	First use case diagram.
1.1.0	IF	Variations
1.2.0	BM	Numbered/Added InvestNI

The final use case diagram was created to reflect the use cases and clarify the requirements of the system. It details the different interactions between use cases and the system as a whole, while also setting out the boundaries for the game system. One actor has been used to represent each player.

It was noted that a Domain Model could have been implemented to further the planning and analysis stage. However due to time constraints on the project this was deemed unnecessary.

Realisation (TL/BM led. DK/CC/IF contributed.)

Upon completion of the use cases and use case diagram, sequence diagrams were developed in order to fully realise the system design before development. The diagrams below highlight the main object interactions, arranged in time sequence, within the Technopoly system. They focus on lifelines and the objects that interact simultaneously, as well as the messages passed between them, before the lifeline ends or is destroyed.

The group focused on the key events in the use case descriptions to realise in their sequence diagrams:

• TurnEngine was envisioned as being made up of Use Case 3 [Takes Turn]. Turn Engine continues until it reaches Use Case 13 [Terminates Game] or Use Case 11 [Ends Turn]. Yet by its nature it also contains elements from Use Cases 7 [Purchase StartUp], 5[Pay Licence Fee], 6 [Declares Bankruptcy] and 9 [Views Menu].
• GameAdmin was a realisation of Use Case 1 [Starts Game] and Use Case 3 [Views Rules].
• UserInput was a realisation of Use Case 14 [Verifies Choice]. Despite initially being designed solely to verify a user's choice, upon further development it was decided that this class should manage all user input.
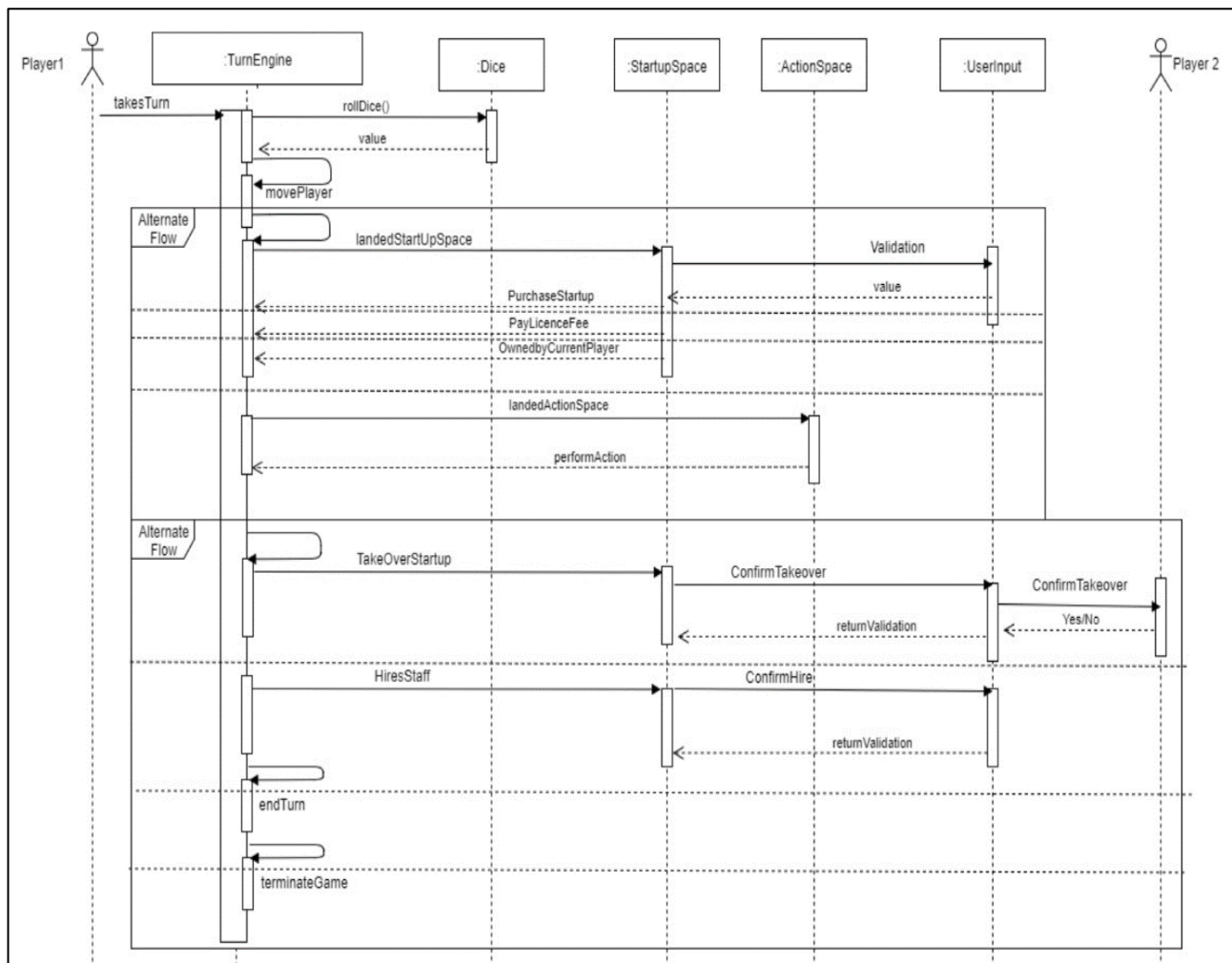
i. Sequence Diagrams

Figure 3 Turn Engine Sequence Diagram

The TurnEngine is the largest object within the Technopoly game. All behaviours that are contained within a turn are controlled by the TurnEngine, while all behaviours relating to the management of each turn are controlled by the GameEngine object. In order to control all the behaviour of each turn, the TurnEngine is required to interact with several other objects within the Technopoly game, as shown in figure 4. For example, the TurnEngine is responsible for rolling the dice and moving players around the board. In order to achieve this, it must interact with the Dice object during each player's turn before it can move a player around the board.
Having moved a player, the TurnEngine must then interact with the StartupSpace object to determine whether a licence fee needs to be paid or if a start-up is available for purchase. The TurnEngine also checks the StartupSpace object to see if a startup can be developed, enabling the hiring of staff.

The TurnEngine contains all the functionality relating to the in-game menu. As a result it must interact with the UserInput object throughout the game in order to act on each player's decisions. Each time the menu is printed to the console screen, the TurnEngine makes a call to the UserInput object. If a player wishes to terminate the game, for example, they choose the "Terminates Game" option from the menu and that value is then returned to the TurnEngine by the UserInput object. Once a player has requested to terminate the game, the TurnEngine calculates the winner, and outputs a message to screen.
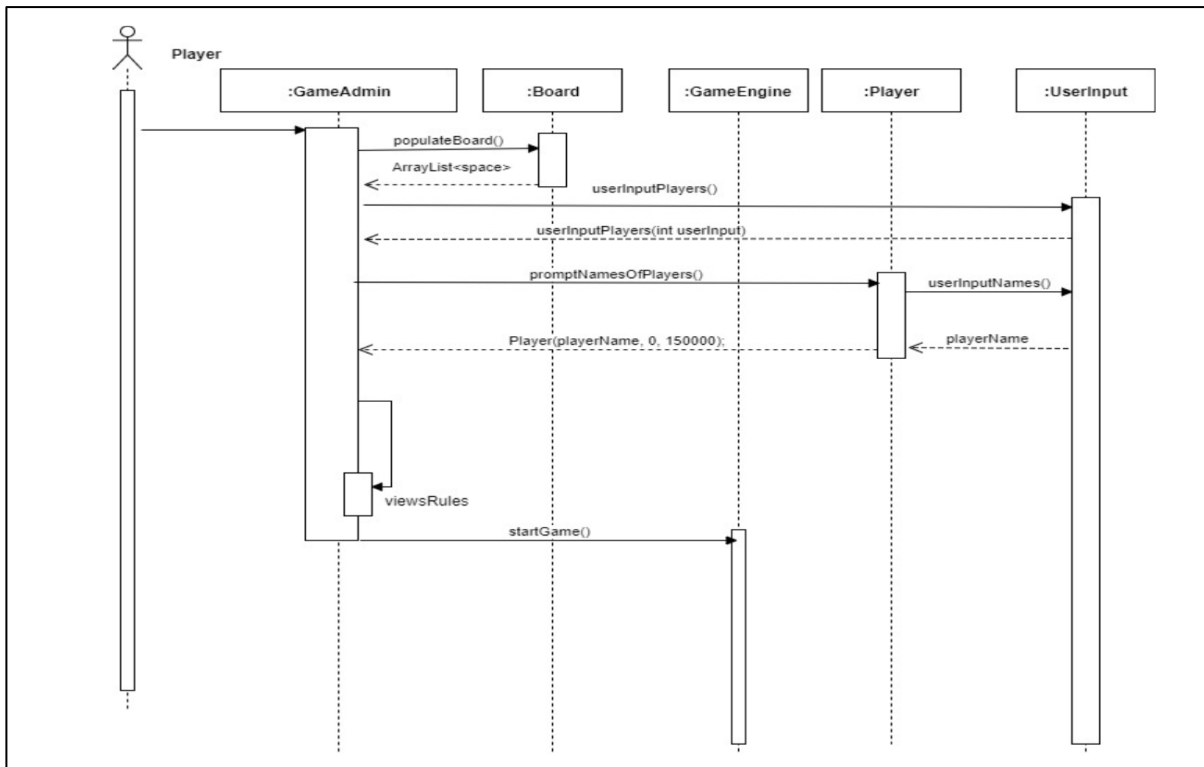
Figure 5 Game Engine Sequence Diagram

As figure 5 shows, the entire Technopoly game is launched from the GameAdmin object. Once instantiated, GameAdmin begins by interacting with the Board object to populate the virtual board with all twelve spaces before the actual game begins. After this, the GameAdmin object then makes a call to the UserInput object to allow the player setting up the game to establish the number of players, before then making another call to UserInput to get each player's username.

After populating the board, assigning the number of players and individual usernames, GameAdmin then instantiates the GameEngine object through the startGame() method and the game begins.
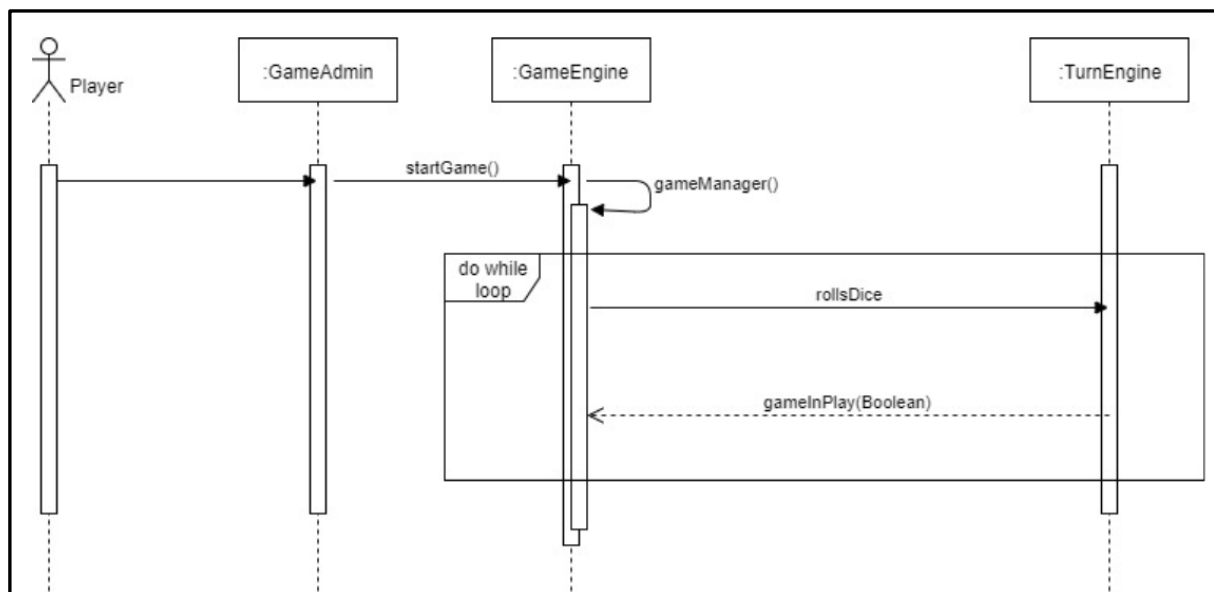
Figure 6 Game Admin Sequence Diagram

Figure 6 reflects one of the major design decisions of the Technopoly project – the GameEngine object. Having initially decided that the management of each turn would be carried out by the TurnEngine object it was then agreed that it would be better to create a separate object, known as GameEngine, that was solely responsible for managing each turn. This meant that all behaviour associated with a turn would be kept within the TurnEngine, whilst the management of each turn and round would be controlled by the GameEngine object.

As Figure 6 highlights, the GameEngine is instantiated by the GameAdmin object through the startGame() method. The GameEngine manages the game through the gameManager() method which is designed to monitor and store the following: the count of the round the game is on, which player's turn it is to roll the dice (via the rollDice() method in the TurnEngine) and how many turns have been completed. Furthermore, the gameManager() method tracks doubles, and checks if a player has rolled three in a row.

The gameManager() method interacts with the TurnEngine object by continuously calling the rollDice() method from within the TurnEngine, while the gameInPlay Boolean returns true. The gameManager will continue to generate a new round after each player has taken their turn as long as no player has run out of resources or terminated the game.
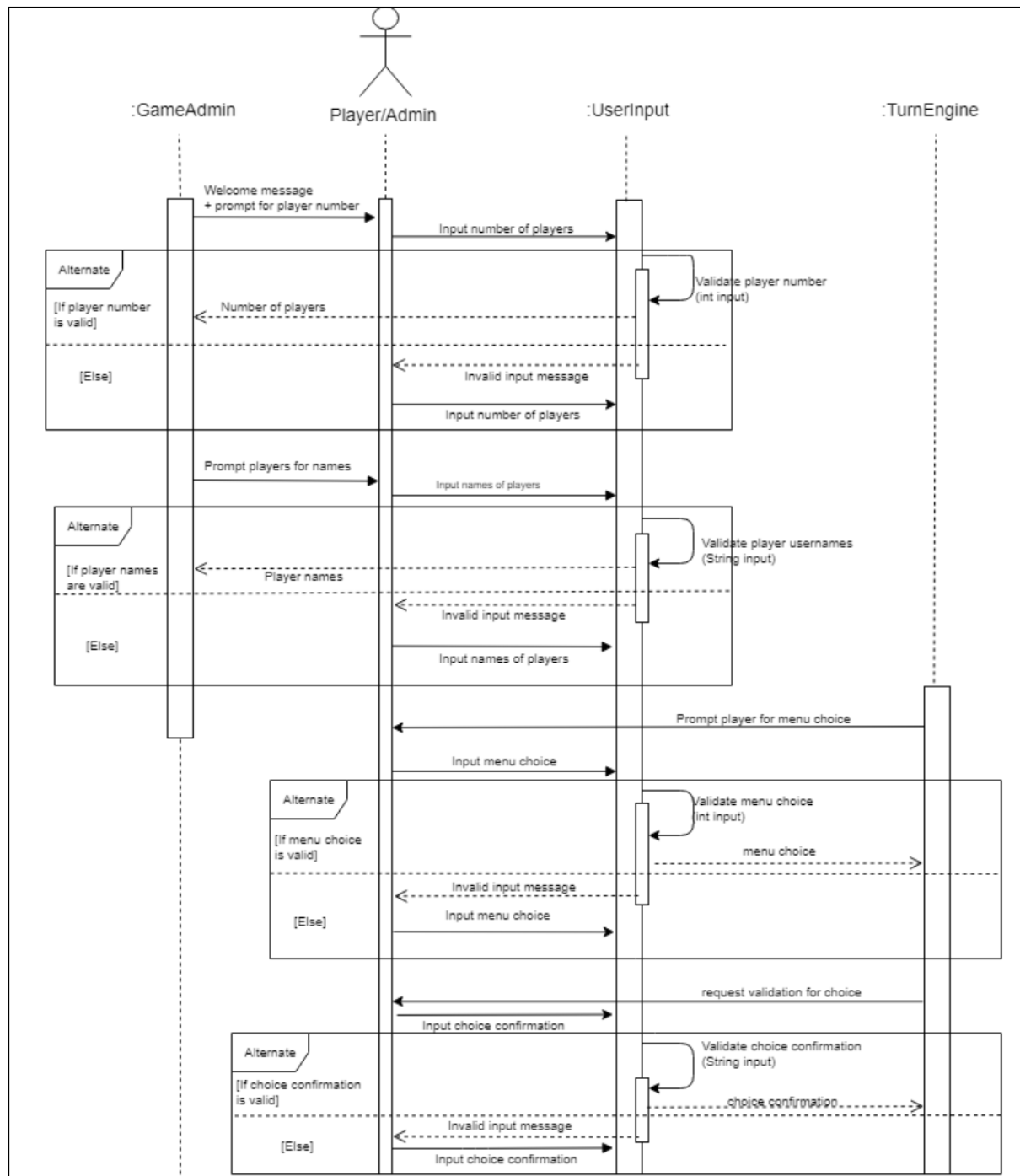
Figure 7 UserInput Sequence Diagram

The UserInput object is concerned with scanning and validating a player's input before returning the value to either the GameAdmin or TurnEngine objects. The UserInput object is essential for all interactions between players and the system throughout the game.

As highlighted in figure 7, the first instance of UserInput within the game is during initial game set up. The player setting up the game is asked to input the number of participating players, a method within the UserInput object is called from the GameAdmin object and scans in the user's response. It is here that the function of UserInput is vital, as the user's response will be validated before being returned to the GameAdmin object. If the user inputs an invalid value, the UserInput object does not return the value, instead asking the user to re-enter the value until a valid answer is entered. This system of validation is repeated throughout the UserInput object for each interaction between the player and the system.

Design

i. Traceability table

Use Case Description	Final UML Class Name
1.Starts Game	GameAdmin
2.Views Rules	Method in GameAdmin - printGameRules()
3.Takes Turn	GameEngine (Class) has gameManager() method which creates and instance of TurnEngine and then rollDice() method is called from that instance
4.Lands on Runway	Runway class extends Space class
5.Pays Licence Fee	paysLicenceFee() is a method in TurnEngine class. It first checks Bank Class checkFunds() method before using the Bank Class subtract() methodto deduct from the paying player and then calls the Bank add() method to add those funds to the receiving player balance. If Bank Class checkFunds() method returns false then paysLicenceFee() method will call the declareWinnerMethod() from TurnEngine
6.Declares Bankruptcy	Bank Class checkFunds() method returns false then the declareWinner() method is called and the game ends
7.Purchase Startup	purchaseStartUp() method in TurnEngine Class interacts with UserInput Class userInputValidation() method. If it is returned a 'Y' it will then it will use the Bank Class subtract() Method to reduce the balance of the current player by the Startup price. The startup space owner is then updated and the listOwned() method in TurnEngine class is called to dislay a revised list of current players owned startups. It then calls the viewsMenu() method in Turn Engine Class
8.Passes Invest NI	InvestNI Class extends the Space Class it exists to hold the investment amount that a player gets upon passing it. When the player lands on or passes the InvestNI space the addInvestment() method is called which in turn calls the Bank Class .add method. Passing or landing on InvestNI is handled by the movePlayer() method in the TurnEngine class.
9.Views Menu	viewsMenu() method is located in the TurnEngine class. viewsMenu() method uses a menuList ArrayList to figure out what options need to be displayed in the dynamic menu. The ArrayList values are set by every other method in the TurnEngine class depending on what needs to be shown
10.Hires Staff	hiresStaff() is a method in TurnEngine Class. It uses a switch statement of determine how many staff can be hired on the give startup and then sets the menuList ArrayList to reflect that in the dynamic menu. This also utilises the Bank Class subtract() method to debit the current players balance after purchase.
11.Ends Turn	endTurn() is a method in TurnEngine class. It interacts with the userInputValidation method in the UserInput class to determine if the current player wishes to end their turn
12.Takes Over Startup	takesOverStartup() method in TurnEngine class. It populates an ArrayList with other startups owned by players other than the current player. This displays via the pushScreenContent() method in the MessagePrinter Class. The userInputMenu() method from the UserInput Class is then used to choose which statartup to takeover. It also ustilises the Bank Class methods and UserInput methods to complete its actions.
13.Terminates Game	terminatesGame() is a method in the TurnEngine Class. It uses the userInputValidation() method in the UserInput classto check if the player wants to quit or not. If they do it sets the GameAdmin class setGameInPlay variable to false and then calls the declareWinner() method in the turn engine.
14.Verifies Choice	Became UserInput class. It contains four key methods these take in a Yes or No, a String name, number check and number of players. Validation methods ensure all input stays within specified bounds.

ii. UML Class Diagram (DK/IF/CC lead but the whole team contributed fully to the design and implementation of the game)

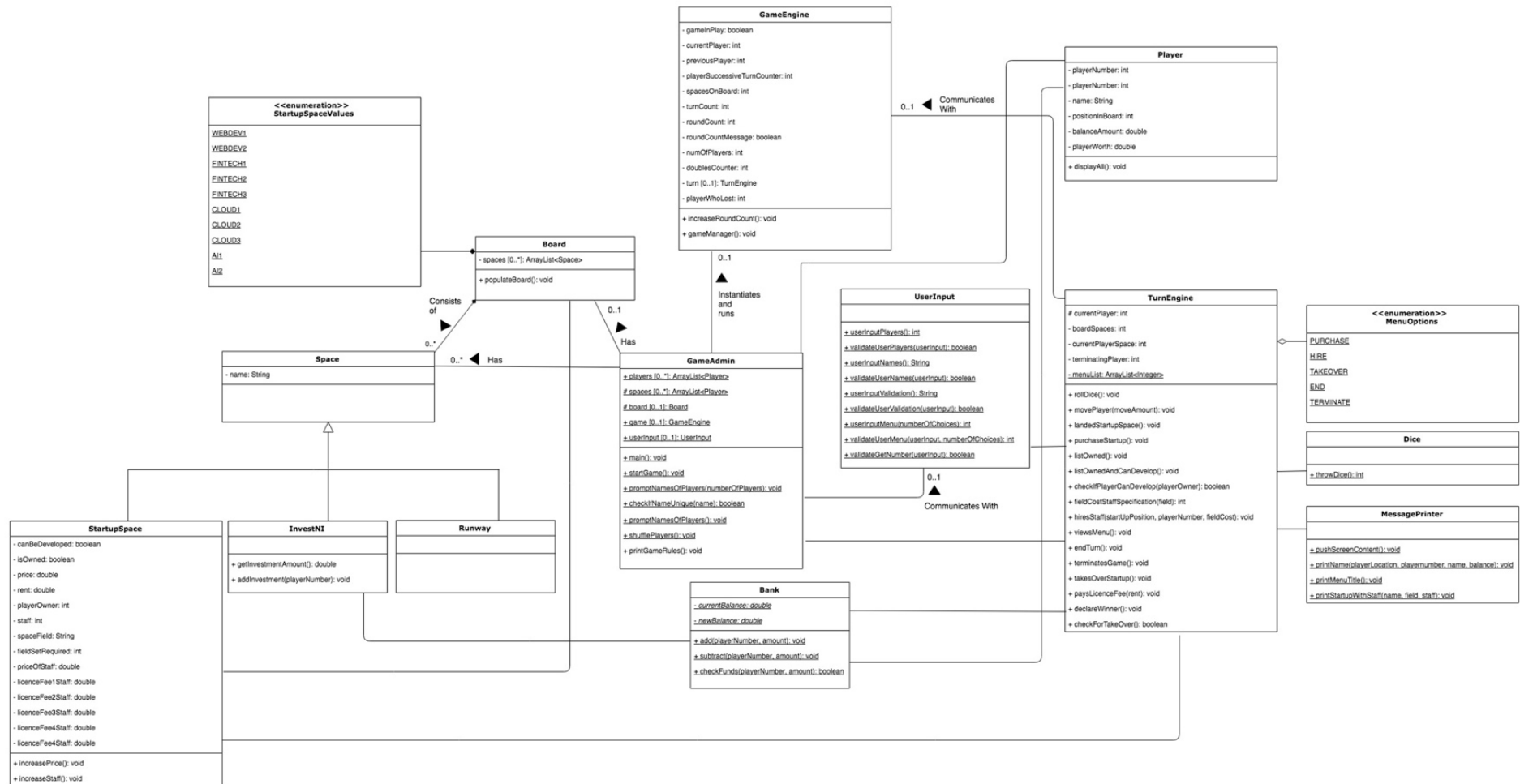The Group 3 UML Class diagram gives an overview of the Technopoly system's working components.

Figure 8: Group 3 Technopoly UML Class Diagram

Initial design of the game involved the discovery of candidate classes through noun spotting of the Game Guide and Game Requirements documents. Whilst many of the classes had already been accounted for, some nouns gave way to new classes. Some noteworthy considerations included:

- **Tax**: it was decided that rolling three doubles in a row would incur a tax of £1,000.
- **Winner**: the group considered implementing a leader board system and decided that it should be a feature of a future iteration of the game.
- **Takeover**: to incentivise players to part with their startups during takeover attempts, it was agreed that a 20% markup price should be attached to all startups upon their purchase. This would improve the playability of the game.

The class diagram shown in Figure 8 corresponds closely with the final implementation of the Technopoly game. Principles of maintenance and extensibility were constantly considered throughout the design process. This is evidenced by the use of the Enum data type in StartupSpaceValues, wherein each value holds its own information. The board size is dynamically generated depending on however many Enum values are listed. Having the values in a separate list makes them more accessible and easier to maintain if updates or changes are required. For extensibility, the size of the list can simply be increased if future iterations of the game require more spaces on the board.

In the class diagram, shows a composition relationship between the spaces and the board. The spaces can only belong to one board at a time and will be destroyed when the board is terminated at the end of the game.
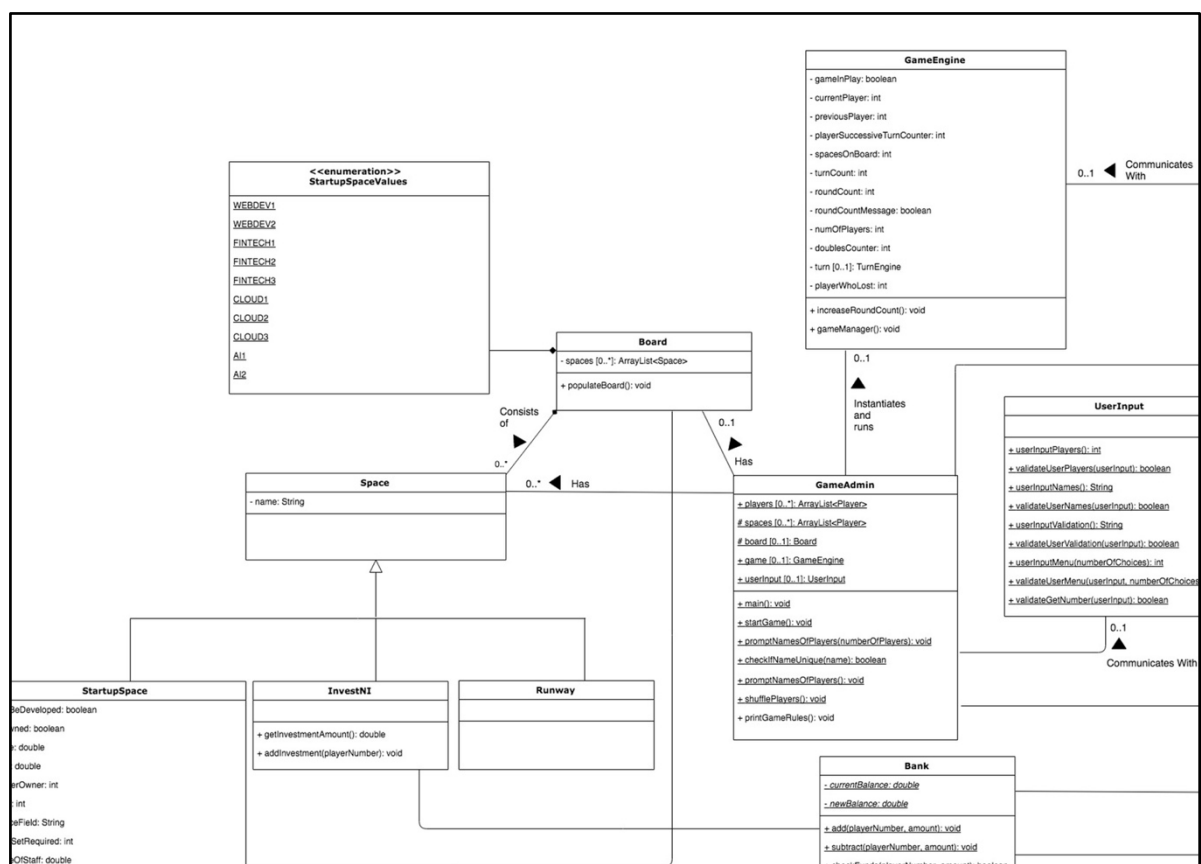

Figure 9: Examples of Composition and Inheritance Relationships in Technopoly UML Class Diagram

Generalisation (inheritance) has also been used to construct the different types of spaces on the board. The general superclass Space is placed at the head of the open arrow and is extended by the more specialised classes StartupSpace, InvestNI and Runway. Each subclass 'is a kind of' space on the Technopoly board (p157, Priestly). Using this relationship improves ease of maintenance and would allow extensibility by making it simpler to develop other subclasses in the future.

While coupling is useful in these circumstances, it is loose and limited to this one area of the diagram; the overall design aimed to achieve a higher degree of cohesion. In doing so, careful consideration was given to the responsibilities each class would have and how well they would fit together. When a key class such as TurnEngine became expansive during development, a new GameEngine class was created to ensure a better division of responsibilities. This also helped make the code easier to develop and maintain.
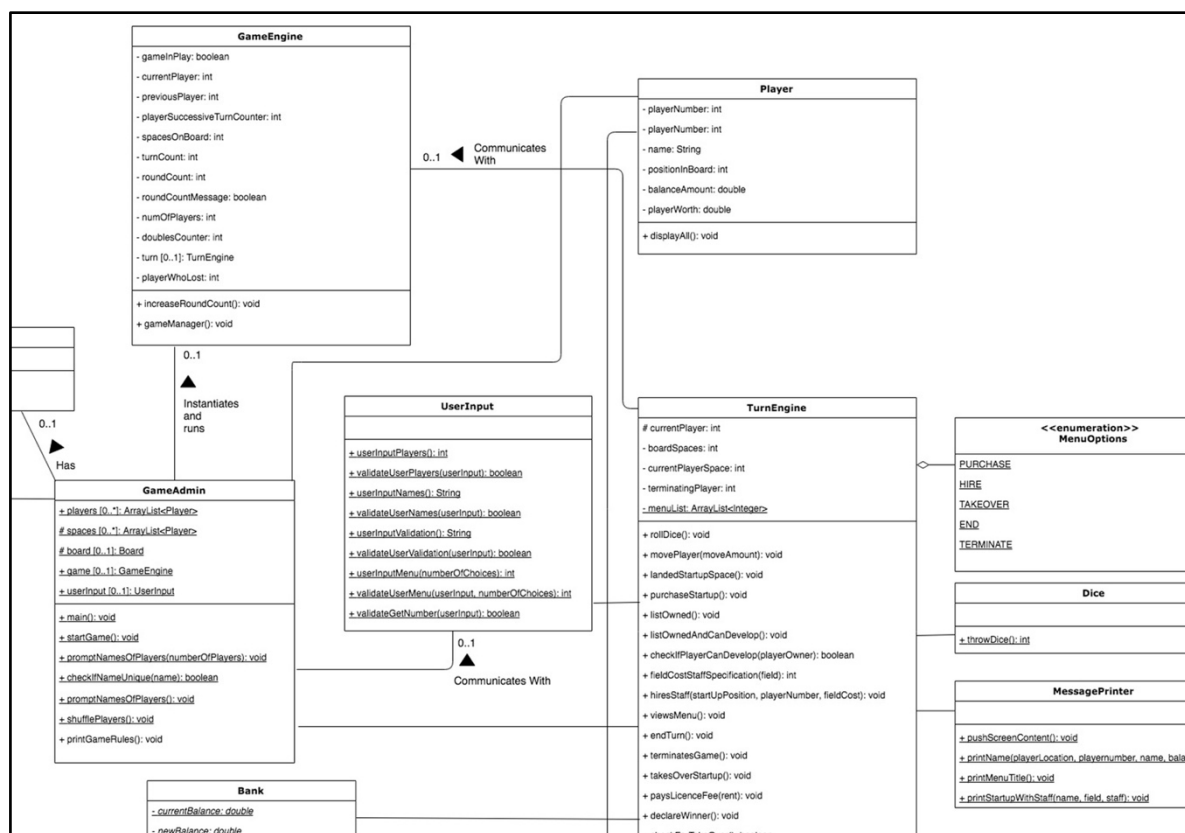

Figure 10: Closer look at TurnEngine and GameEngine in the Technopoly UML Class Diagram

The GameEngine class enables the system to keep track of each round, turn and how many doubles the current player has rolled in a row. The round counter was added with future extensibility in mind as very little extra work would be required to add a round limit to any game. This idea fuelled further adaptations in other classes, such as the StartupSpace class. This class has an alternative constructor, which may be used in a future version of the game, to assign ownership of startups to players at the beginning of the game. This could lead to a 'Quick Game' option being given at the start of the program, where players are already assigned a certain amount of assets and a round limit is set.

Other important design considerations include:

- The SecureRandom class being used in the Dice class, as it is more secure and there is less chance of returning the same number twice.
- A separate UserInput class to handle all user input operations. This furthers the extensibility and maintainability by limiting all user messages to one class. There are four methods in this class to deal with different types of input from the user and another four additional methods to handle validation of the inputs. More types of input could be accommodated with little work and the existing methods in the UserInput class can be called by any new class or method easily.
- The viewsMenu method in the TurnEngine class is dynamic, only showing the options available to the user/player at the appropriate times. The menu will not display the option to purchase a startup that is already ownedand will not display the option to perform a takeover if no other player owns a startup.
- The value of startups increase in price when purchased. This offers a financial incentive to the potential seller during a takeover attempt.
- A separate Bank class to process all balance transactions.
- A separate MessagePrinter class to handle formatting of key message outputs. For example, the name of the player's turn along with their balance.

An option to view the game rules at the start of the game was included to provide the user/player with everything they need while in the domain of the game. A map of the board would be an obvious future inclusion if implementing a GUI.

The language was carefully considered to be playful at big decision points, such as the termination of the game and for points of no control such as being taxed. For the vast majority of the game the language is purposely direct, informative and sparing. It was decided that the user should not feel like they are overloaded with messages but rather are able to understand where they are and what they own at a glance.

Ideally, given more time, there would be further de-coupling of the TurnEngine class. Due to time constraints and with a willingness not to veer too far from the initial concept design, the group included the declareWinner(), purchaseStartup() and hiresStaff() methods in the TurnEngine. However, the group feels these methods may be better represented as their own classses in a future iteration of the game, as they perform a complex function in their own right.

Conclusion

By following the process of requirement analysis, realisation and design, Group 3 were able to test and implement a working Technopoly game with the appropriate functionality required by the customer. The theme has a technology twist based on local companies and investors. All the final functionality of the working game can be viewed in the accompanying video attached with this report. In the meantime, Group 3 invite you to become the actor in their game of Technopoly. Try for a triumph by becoming the wealthiest player through buying and selling, hiring and firing at local technology companies.

https://gitlab.eeecs.qub.ac.uk/csc7053-1819/csc7053-1819-g3