TECHNOPOLY

Technopoly Game Software Engineering Project

Hugo Harbinson, Ethan Harbinson, Luke Kelly, James MacCreanor, Christopher Healey

Group 6

Concept

The Technopoly project will develop a Monopoly style game with a technology theme. Players will seek to acquire companies from various sections of the technology industry, developing offices and campuses for their companies in order to extract higher and higher subscription fees from opposition players in order to become the ultimate tech tycoon! With a minimum of two players competing, the goal will be to earn the highest number of "Techcoins", the in-game currency, while avoiding hitting zero at which point the game will end and players will be ranked first to last by the value of their assets owned.

Implementation

The game will be developed in java using the Eclipse console. Game play will also be through the console using plain English text interactions

Game Guide

Between 2-5 players.
Players will play turnabout.
Each player starts with 1500 "techcoins".
Each turn a player must 'roll' 2 virtual die, which generates a number between 2 and 12 that will move them round the virtual board (see virtual board on next page).
The board is comprised of a Funding Round, Company, Utility, Chance, a Holiday square and a "Hacked" square.
When you pass the funding round you'll receive a venture capital investment of 100 techcoin.
Utility squares comprise 2 "Techcoin Mines" and 2 Data Centres. If a player lands on the Utility, they can buy it. If it's already owned they have to pay the subscription fee to the player who owns it. Each utility has a subscription fee of 30 but this is multiplied by the number of utilities owned, so if 4 utilities are owned by one player the subscription fee will be 120 techcoins.
When a player lands on a chance square they will get a randomly generated action that could help or hinder them.
Holiday square is a free space to take a breather.
If you land on a Hacked square you must pay the hackers 100 or 200 techcoins in ransom!
Company squares are divided into 4 sectors; Streaming, retail, social media and tech giants. When you own all of a certain sector you can build offices for your companies. You can build up to 4 offices per company, with each office built the subscription fee of the square increases. Once 4 offices are built you may develop further by building a campus, the four offices are then removed, and the company square has been fully developed. Any properties owned can be sold back to the bank for half the price, all campuses and offices will be removed from the sold property.
Subscription fees are charged if a player lands on a company that another player owns at 20% of the company value e.g. Player 1 owns Netflix which they bought for 60, when player 2 lands on Netflix they must pay Player 1 a fee of 12 techcoins. As houses are added they will increase the subscription fee by a set amount per sector, as described in the game guide on the next page.

Virtual Board

HOLIDAY	260 Twitter	260 Instagram	150 TECHCOIN MINE	280 Facebook	CHANCE ?
160 Amazon			TECHNOPOLY		HACKED PAY 100
140 Alibaba		CHANCE			350 Apple
150 DATA CENTRE					150 DATA CENTRE
140 eBay					400 Microsoft
CHANCE ?	HACKED PAY 200	Hulu 60	TECHCOIN MINE 150	Netflix 60	Funding Round Collect 100 Investment

Streaming	Retail	Social Media	Tech Giants
Office cost:40 Sub Increase per office: 25 Campus cost: 40 Sub Increase per campus: 125	Office cost:105 Sub increase per office: 50 Campus cost: 105 Sub increase per campus: 250	Office cost:195 Sub increase per office: 75 Campus cost: 195 Sub increase per campus: 375	Office cost:300 Sub increase per office: 100 Campus cost: 300 Sub increase per campus 100
Subscription Without Office: 20% Value	Subscription Without Office: 20% Value	Subscription Without Office: 20% Value	Subscription Without Office: 20% Value

User Story for Game Play

Upon starting the game, the user will enter the number of players, then each player will enter their name and play will then commence in the order in which the players entered their name. The player will be given a menu of options:

1. Roll Die
2. Sell Property
3. Grow business
4. View resources
5. Display board
6. Forfeit game

At this point in the game the player can sell any properties owned, buy offices or campuses for companies when a sector is controlled, view your current resources meaning bank account, companies and utilities owned, display the "board" i.e. the position of each player in the game. Finally, they can quit the game and forfeit to last position.

When a player rolls the die, they will be given a number which will move them to a square where they interact with it. For example, the user rolled a 3 and lands on Hulu, they are given details about the square and the option to buy the company. They elect to buy and now own the property, but resources will decrease. The game will then move onto the next player.

Use Case Descriptions

Primary player selects number of players	
Objective	***To set the number of players in the game***
Precondition	*There is an input request for number of players*
Main Flow	*1. The user is prompted to enter the number of players between 2 and 5* *2. The player enters a valid input* *3. A confirmation question is displayed* *4. Player confirms* *5. Number of players is set*
Alternative Flows	*At 2 the player could enter an invalid input, so they would be asked to re-enter* *At 4 the player could say no so will be asked to re enter*
Post-condition	*The number of players is set*

Display square details	
Objective	***Square details and options are displayed***
Precondition	*A player has rolled die and moved to square*
Main Flow	*1. The details of the square are displayed* *2. Includes any compulsory actions that must be taken and any options that can be taken*
Post-condition	*Square details are displayed*

Each player enters their name	
Objective	***To set the player names***
Precondition	*There is an input request to enter the player name, number of players has been set*
Main Flow	1. *The user is prompted to enter a name* 2. *The player enters a valid input* 3. *A confirmation question is displayed* 4. *Player confirms* 5. *Player name is set* 6. *Continues until all players have entered a name*
Alternative Flows	*At 2 the player could enter a duplicate of a previous player name and will be asked to re-enter* *At 4 the player could say no so will be asked to re enter*
Post-condition	*Each player's name is set*

Display Player Options Menu	
Objective	***Display player options menu on screen***
Precondition	*It is the beginning of a players turn*
Main Flow	1. *At the start of a players turn they will have a menu displayed with the turn options* 2. *They can select any of these options* 3. *They can return to menu for the transactional options once transaction is complete.* 4. *Will move onto next step and menu will disappear until the start of the next players turn for other options*
Alternative Flows	*At 2 can quit game in which case menu will not reappear.*
Post-condition	*Player is allowed to action all options and complete their turn*

Quit Game	
Objective	***Player can quit the game***
Precondition	*It is the beginning of a players turn*
Main Flow	1. *At the start of a players turn they have the option to quit the game* 2. *Select option* 3. *Confirmation message displayed* 4. *Player confirms* 5. *Game is quit by player*
Alternative Flows	*At 3 player can change their mind and go back to menu*
Post-condition	*Game is over*

Pay Rent	
Objective	***If a player lands on a company or utility square they must pay subscription to the player who owns it***
Precondition	*Company or utility square owned by one player and landed on by another*
Main Flow	*1. Subscription fee is displayed* *2. Player who lands on the square will pass the value of the subscription to the player who owns the square* *3. Each players resource is adjusted* *4. The players turn is over* *5. If the player runs out of techcoins the game is over*
Alternative Flows	
Post-condition	*Both players resources are adjusted by the subscription amount*

Sell Property	
Objective	***A player can sell a company back to the bank for half the cost of the company***
Precondition	*Company or utility square owned by a player*
Main Flow	*1. Menu at the start of a turn select sell company* *2. Option to type in company to sell* *3. Input valid property name* *4. Confirmation message* *5. Company is sold back to game* *6. Player techcoin supply increases* *7. Option to sell another property*
Alternative Flows	*At 1 if no property owned revert back to menu* *At 2 if invalid property entered option to enter again* *At 4 option to return to menu if you change your mind*
Post-condition	*Player techcoin increases. Company is available to buy for other players*

Pass Funding Round	
Objective	***A player passes the funding round and collects 100 techcoins***
Precondition	*Player rolls die*
Main Flow	*1. Player rolls a number that will land them on or take them past funding round* *2. Message is displayed stating that* *3. Player resources increase*
Alternative Flows	
Post-condition	*Player techcoin increases.*

Buy Square	
Objective	***A player buys an unowned company or utility***
Precondition	*Player lands on an unowned company or utility*
Main Flow	1. *Option is displayed to buy property with square details* 2. *Player selects Y* 3. *Player owns the square*
Alternative Flows	*At 2 Player selects to not buy the square, turn ends*
Post-condition	*Player techcoin decreases. Company/ Utility added to player resources, not able to be bought by other players.*

Buy Office/Campus	
Objective	***A player develops an office or campus on an owned company***
Precondition	*Player owns a complete sector of companies, it is the beginning of their turn.*
Main Flow	1. *Select grow business from menu* 2. *Prompted to type in name of business to develop* 3. *Valid company input* 4. *Asked if they would like to develop an office or campus or go back* 5. *Office input* 6. *Confirmation requested* 7. *Confirmed* 8. *Office added to the company*
Alternative Flows	*At 1 if no complete sector owned then message displayed stating this and returned to player menu* *At 2 invalid company name input asked to re-enter* *At 4 if campus entered and not 4 offices owned then cannot build campus, back to office/ campus option* *At 6 if not confirmed, back to office/ campus/ back selection*
Post-condition	*Player techcoin decreases. Office/campus added to player resources on particular company.*

Use Case Diagram

UML Class Diagram JM EH

Game
- board: Board
- confirmed: boolean
- correctNumberOfPlayers: boolean
- currenPlayer: Player
- currentSquare: Square
- numberOfPlayers: int
- playerList: ArrayList<Player>
- +Game(scanner: Scanner)
- +getter and setter methods
- +buildCompany(company: String, officeCost: int, campusCost: int):
- +buildCompanyBusiness(canSellProperty: boolean, i: int): boolean
- +confirmNumberOfPlayers():
- +confirmSellBusiness(canSellProperty: boolean, i: int): boolean
- +displayFinalRankings():
- +displayTurnOptions():
- +displayWelcomeMessage():
- +endGame():
- +endGame(currentPlayer: player):
- +endTurn():
- +growCompany():
- +requestNumberOfPlayers():
- +requestPlayerNames(numberOfPlayers: int):
- +setUp():
- +rollDice():
- +sellBusiness():
- +startGame():
- +updatePlayerPosition(movement: int, player: Player

Board
- alibaba: Retail
- amazon: Retail
- apple: TechGiant
- chance1: Chance
- chance2: Chance
- companies: ArrayList<Company>
- dataCentre1: Utility
- dataCentre2: Utility
- digitalTax: Tax
- ebay: Retail
- facebook: SocialMedia
- fundingRound: GO
- hacked: Tax
- holiday: Holiday
- hulu: StreamingService
- instagram: SocialMedia
- microsoft: TechGiant
- nine1: Utility
- nine2: Utility
- netflix: StreamingService
- squares: ArrayList<Square>
- twitter: SocialMedia
- +Board(squares: ArrayList<Square>)
- +getter and setter methods
- +displayBoard(player: Player):
- +multiplyUtilitySubscription(player: Player):

Player
- name: String
- numberOfRetailOwned: int
- numberOfSocialMediaOwned: int
- numberOfStreamingServicesOwned: int
- numberOfTechGiantOwned: int
- numberOfUtilitiesOwned: int
- ownedCompanies: ArrayList<Company>
- ownedSquares: ArrayList<Square>
- playerNumber: int
- position: int
- resource: int
- +Player(name: String, position: int, playerNumber: int)
- +getter and setter methods
- +addTotalResource():
- +display(OwnedSquares():
- +display(Resources():

Square
- field: String
- name: String
- position: int
- value: int
- +Square(field: String, name: String, position: int, value: int)
- +getter and setter methods
- +sendSquareDetails(player: Player, playerList: ArrayList<Player>, scanner: Scanner):
- +updateResource(value: int, player: Player):

Die
- rollNumber: int
- +Die()
- +getter and setter methods
- +rollDie():

Company
- hasCampus: boolean
- numberOfCampuses: int
- numberOfOffices: int
- squareOwnership: int
- +Company(hasCampus: boolean, numberOfCampuses: int, numberOfOffices: int, squareOwnership: int)
- +getter and setter methods
- +addSubscription(numberOfHouses: int, numberOfCampuses: int): int

Office
- +Office()
- +updateNumberOfOffices (company: Company):

Campus
- +Campus()
- +updateNumberOfCampuses(company: Company):

Chance
- chanceResponse: String
- +Chance(chanceResponse: String)
- +getter and setter methods
- +generateChanceResponse(player: Player):

Tax
- +TAX_COST: int
- +Tax()

Investment
- +INVESTMENT_BONUS: int
- +Investment()

Holiday
- +Holiday()

Utility
- squareOwnership: int
- subscription: int
- +Utility(squareOwnership, int, subscription: int)
- +getter and setter methods
- +buyCompany(player: Player, company: Company):
- +multiplySubscription(player: Player):
- +updateSquareOwnership(player: Player):
- +updateUtilitiesOwned(player: Player):

Retail
- campusCost: int
- officeCost: int
- subscription: int
- +Retail(campusCost: int, officeCost: int, subscription: int)
- +getter and setter methods
- +buyCompany(player: Player, scanner: Scanner):
- +updateRetailOwned(player: Player):

TechGiant
- campusCost: int
- officeCost: int
- subscription: int
- +TechGiant(campusCost: int, officeCost: int, subscription: int)
- +getter and setter methods
- +buyCompany(player: Player, scanner: Scanner):
- +updateTechGiantOwned(player: Player):

StreamingService
- campusCost: int
- officeCost: int
- subscription: int
- +StreamingService(campusCost: int, officeCost: int, subscription: int)
- +getter and setter methods
- +buyCompany(player: Player, scanner: Scanner):
- +updateStreamingOwned(player: Player):

SocialMedia
- campusCost: int
- officeCost: int
- subscription: int
- +SocialMedia(campusCost: int, officeCost: int, subscription: int)
- +getter and setter methods
- +buyCompany(player: Player, scanner: Scanner):
- +updateSocialMediaOwned(player: Player):

Relationships: plays, has, Contains, Extends, Interacts with, lands on, rolls, has, becomes

The UML class diagram used in our design follows a notation that Priestly sets out in 'Practical Object Orientated Design With UML' – "the top section of the class icon contains the name of the class, the second section contains its attributes and the third section its operations… Constructors are underlined to distinguish them from normal instance methods of the class" (17), abstract classes have also been defined by placing the class name in italics, while inheritance is shown by a white-headed arrow. The getter and setter methods of each class have been omitted and replaced with "+getter and setter methods" as it is expected that these would be implemented as standard and so do not require unnecessary notation.

As well as this, multiplicity notation has been placed beside classes to indicate how many of each class should exist within the system, for example, only one Board class, one Game class, and between two and five Player classes should exist at any given time to prevent the system from falling into an illegal state. The notation also indicates that a square can interact with between one and two players, for example, when a player lands on an unowned square, the Square class will send details of that square to the player, but when Player 1 lands on a square owned by Player 2, the square will interact with both of those Player classes by deducting the subscription fee from Player 1's resources, and adding it to Player 2's resources via the updateResource(int value, Player player) method.

Due to the text-based nature of the game, appropriate communication with the player via the console is a priority. Players are therefore provided with options to view their own personal resources which includes their in-game currency and the number of properties they own, and to view the state of the board with player positions marked at relevant squares. This ensures that all players are kept up to date with the state of play and so can get the most out of the game.

Appropriate exception handling is used mainly InputMismatchException's to handle errors with user input which is where the primary risk of failure arises within the system. Users are informed of errors with relevant messages, such as "Number of players can only be 2, 3, 4, or 5. Try again!" in the case of when a user tries to set the number of players outside of the stipulated range. Other such error messages include whenever the user gives unexpected input to a yes or no option such as buying a property - "Sorry, that's not a valid input! Please type Y for yes or N for no." The use of such exception handling greatly enhances the system as user's are quickly and reliably informed if and when they have made a mistake, and how to rectify it.

Furthermore, two players are prevented from having the same name to prevent confusion. If a player tries to use a previously selected name then an error message appears reminding them that the name has already been taken so they must choose another. To further prevent user error, confirmation methods are in place across the system for major events. For example, the confimSellCompany method in Game ensures a player does not accidentally sell one of their companies.

Principles of good design are in place across the system. The concept of encapsulation has been adhered to as the data stored inside objects is kept private and can only be accessed and manipulated via public class methods. This provides our code system with security, flexibility, and maintainability. There are also instances of class variables declared as final, such as the *INVESTMENT_BONUS* and the *TAX_COST*. This design choice was made as the bonus received from passing the Funding Round and the fine received from landing on a Tax square will be the same throughout a game. It also allows these numbers to be quickly and easily updated should the use wish to increase the Investment Bonus or Tax Cost, as they need only be updated once which adds to the maintainability and cohesion of the system.
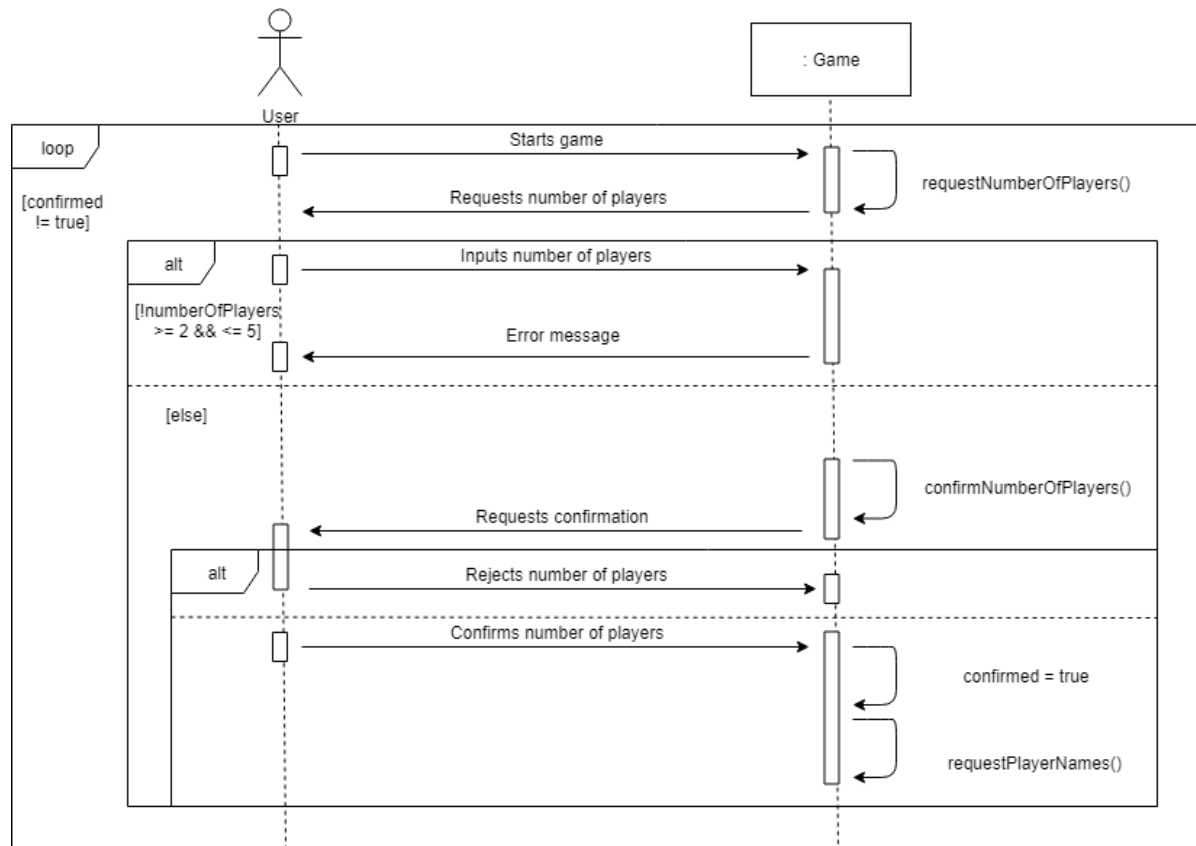
The system also benefits from the loose coupling of modules which is achieved by passing arguments through parameters and method calls. This makes the classes more

independent of each other. For example, the parent Square class updates the player resources, players do not directly interact with the instance of the square that they land on. This adds to the extensibility of the system, as a new square can be added without a new updateResource method having to be coded for it.

This is due to the structure of inheritance that is implemented throughout. The use of inheritance within the system enhances code reuse and limits the amount of duplicate code. For example, a new type of square or company can easily be added. The extensibility of the system is further improved by the concept of abstraction. Square and Company are declared as abstract classes, and they provide common implementation methods for their subclasses, which can be overridden where necessary, while also allowing the addition of specialised methods in each class. For example, the subclasses of Company each inherit and implement the addSubscription method slightly differently as each subclass has a different cost of subscription, and each subclass also adds their own update methods which register with the player class upon purchase. As a result of these factors, the system is easily maintained and could be easily enhanced or extended.
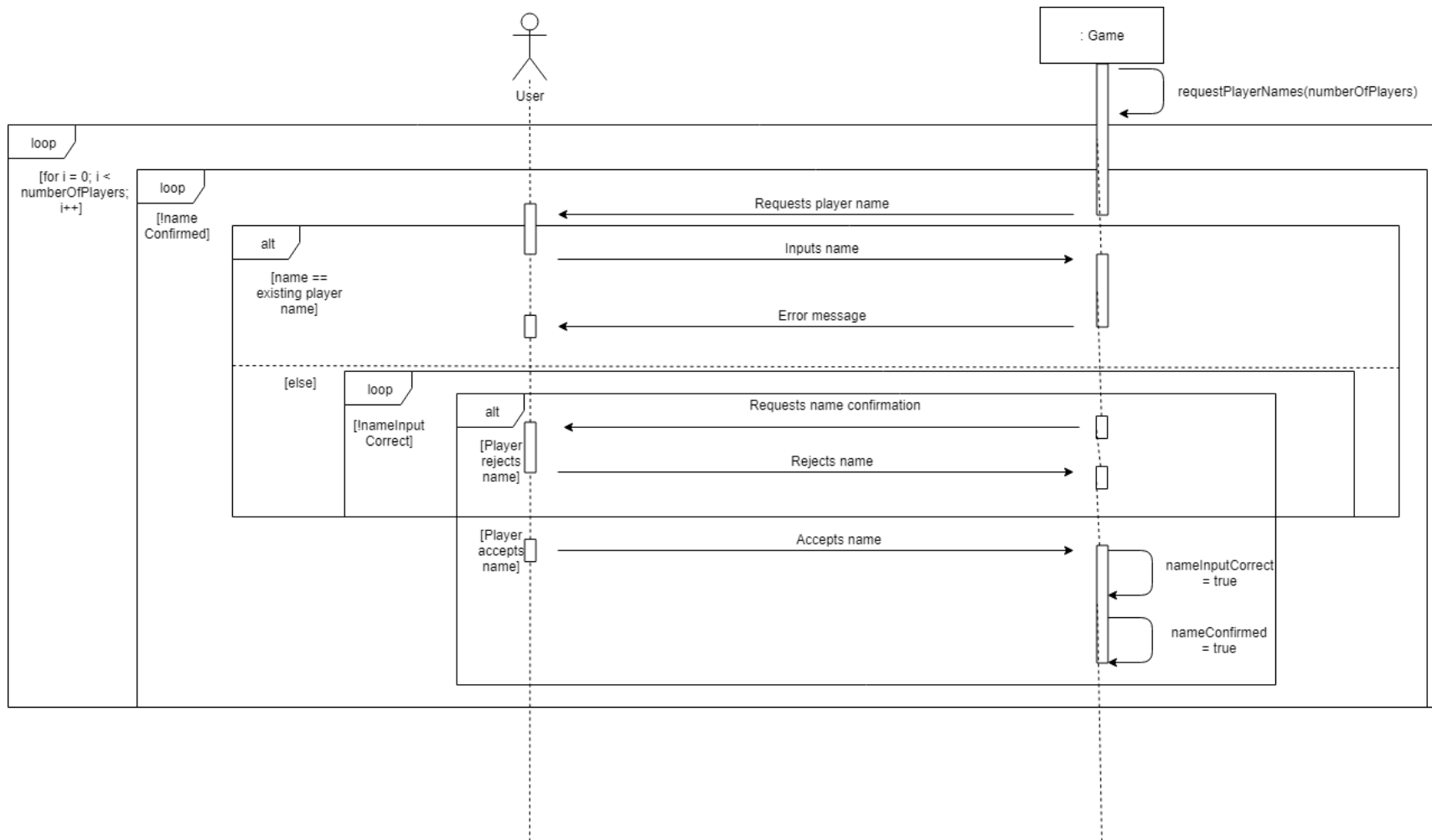
Priestly, Mark. *Practical Object Orientated Design With UML.* 2nd edition. McGraw-Hill Education: Maidenhead, 2003.
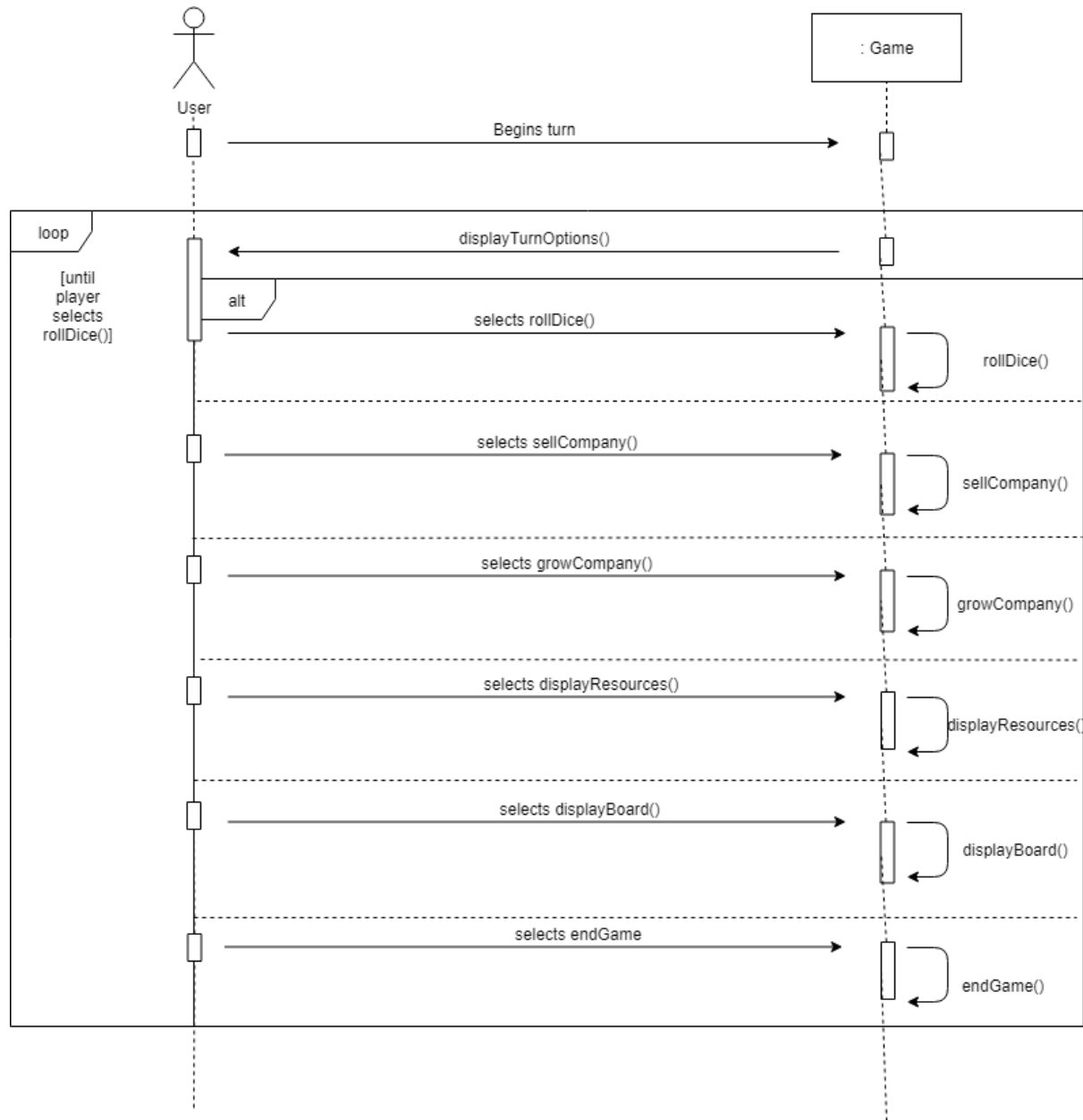
REALISATION LK

Diagram labels:

- User (actor)
- : Game (object)
- loop
- [confirmed != true]
- Starts game
- Requests number of players
- requestNumberOfPlayers()
- alt
- [!numberOfPlayers >= 2 && <= 5]
- Inputs number of players
- Error message
- [else]
- confirmNumberOfPlayers()
- Requests confirmation
- alt
- Rejects number of players
- Confirms number of players
- confirmed = true
- requestPlayerNames()

1. **Select Number of Players.** The system loops through a series of console outputs, asking for the number of players and then confirmation of the number of players. If the user has entered the incorrect number of players, they can indicate this and the loop will restart. Only when the user inputs confirmation of the number of player will the loop break and requestPlayerNames() be called.

REALISATION LK

: Game

requestPlayerNames(numberOfPlayers)

loop

[for i = 0; i < numberOfPlayers; i++]

loop

[!name Confirmed]

alt

Requests player name

Inputs name

[name == existing player name]

Error message

[else]

loop

[!nameInput Correct]

alt

Requests name confirmation

[Player rejects name]

Rejects name

[Player accepts name]

Accepts name

nameInputCorrect = true

nameConfirmed = true

2. Request Player Names. The system loops through a series of console outputs asking for the players' names. The outer loop is executed as many times as there are players and the inner loop breaks only when each name is confirmed. If the player inputs an existing name or fails to confirm their input, they are placed back at the beginning of the inner loop.

REALISATION LK

3. Display Turn Options. The system displays the turn options to the user. Each of these options, when selected, will result in a method call. The menu loops until the player selects rollDice().

REALISATION LK
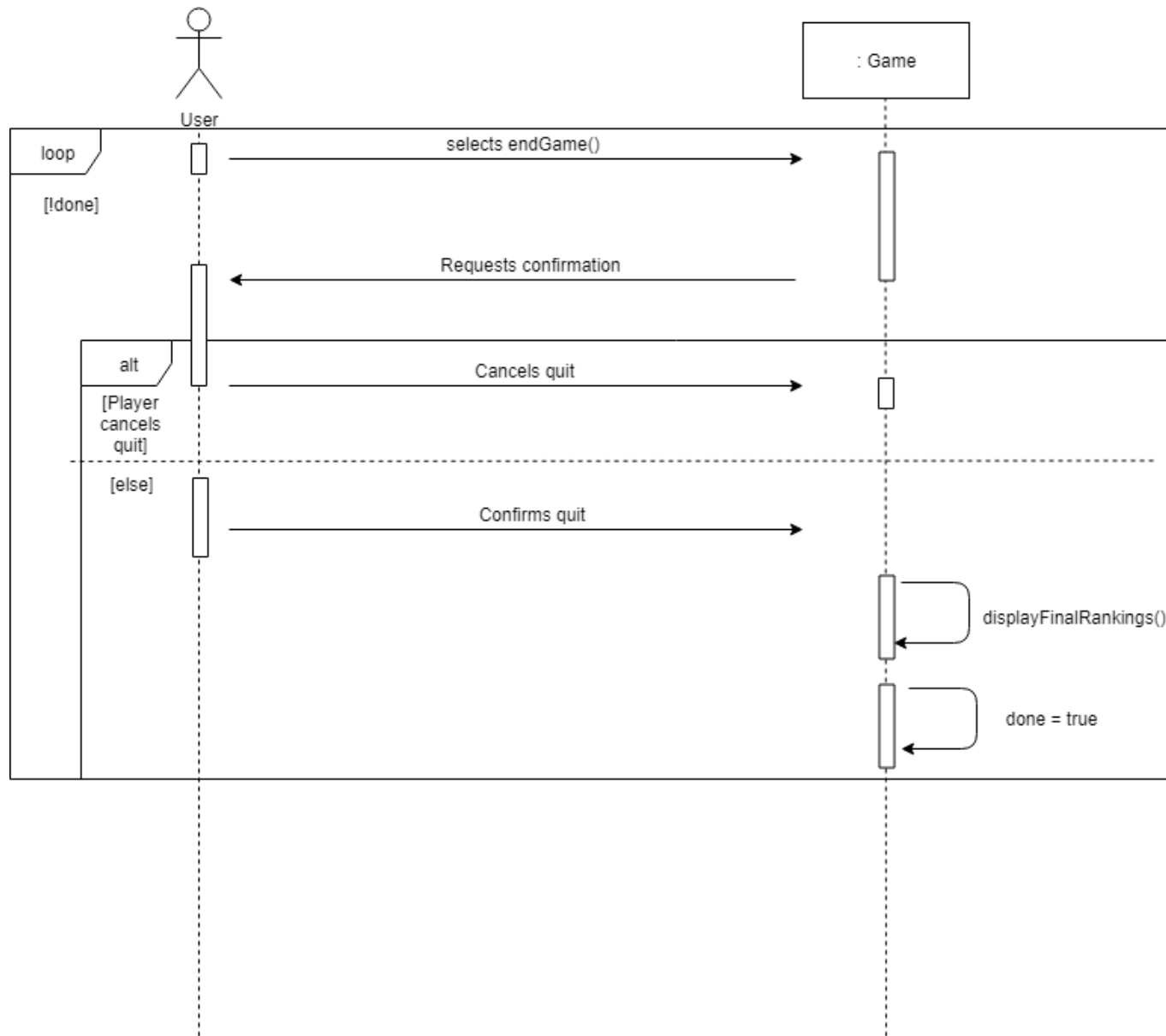
4. Quit Game. If the player selects endGame() from the displayTurnOptions() menu, the system will request confirmation via a system output.
If the player inputs "N", the quit will be cancelled. If the player inputs "Y", then the quit will be confirmed. displayFinalRankings() is called to show the final scores of the players and done is set to true to terminate the endGame() menu loop.

REALISATION LK
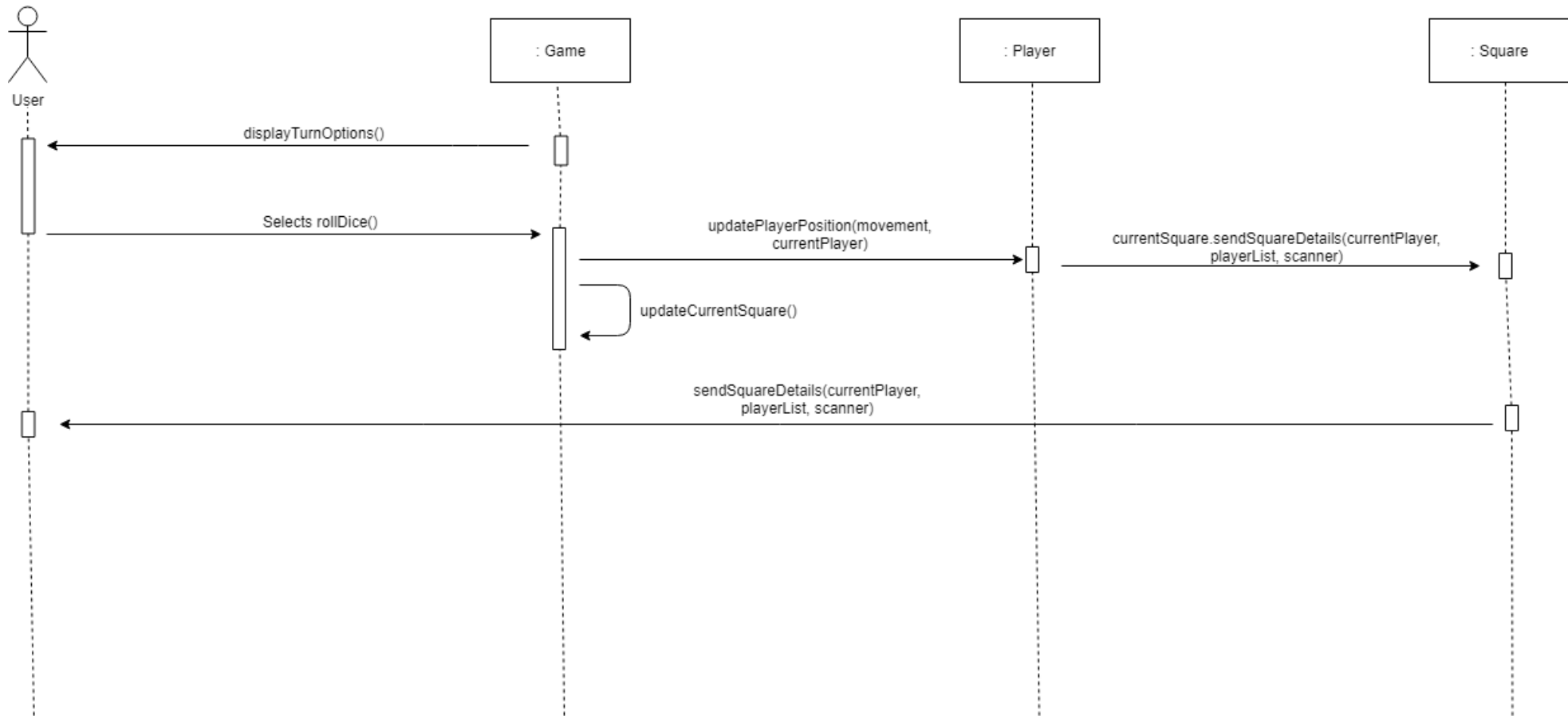
5. Movement and Display Square Details. If a player selects rollDice() from the displayTurnOptions() menu, then two dice are rolled to produce a number between 1 and 12. updatePlayerPosition() is called to move the player the requisite number of spaces and updateCurrentSquare() is updated to contain the square on which that player has landed. currentSquare.sendSquareDetails() is then called to produce the square's details on console for the user and to begin any associated menu logic, such as buying the company on the square/paying
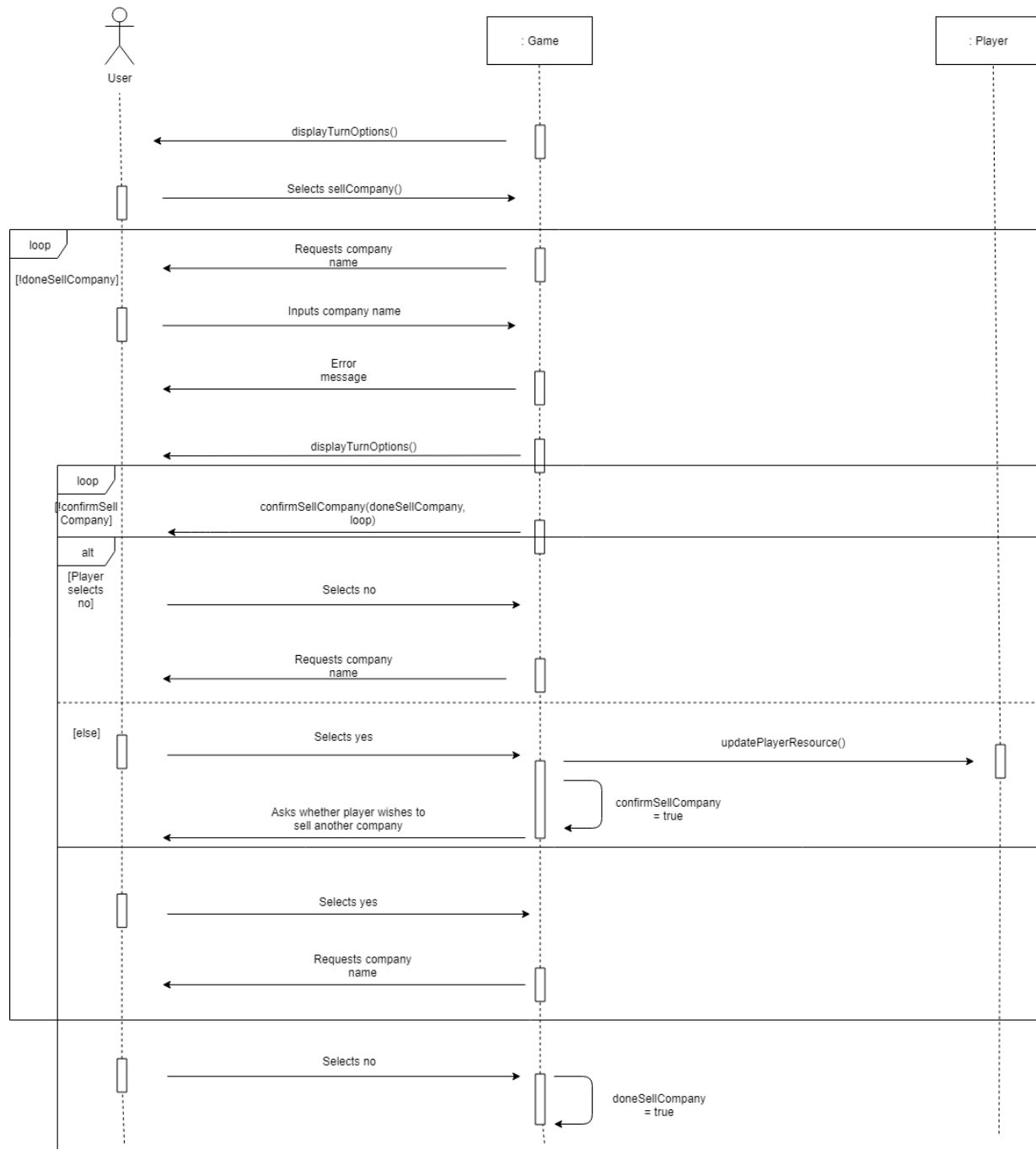
REALISATION LK
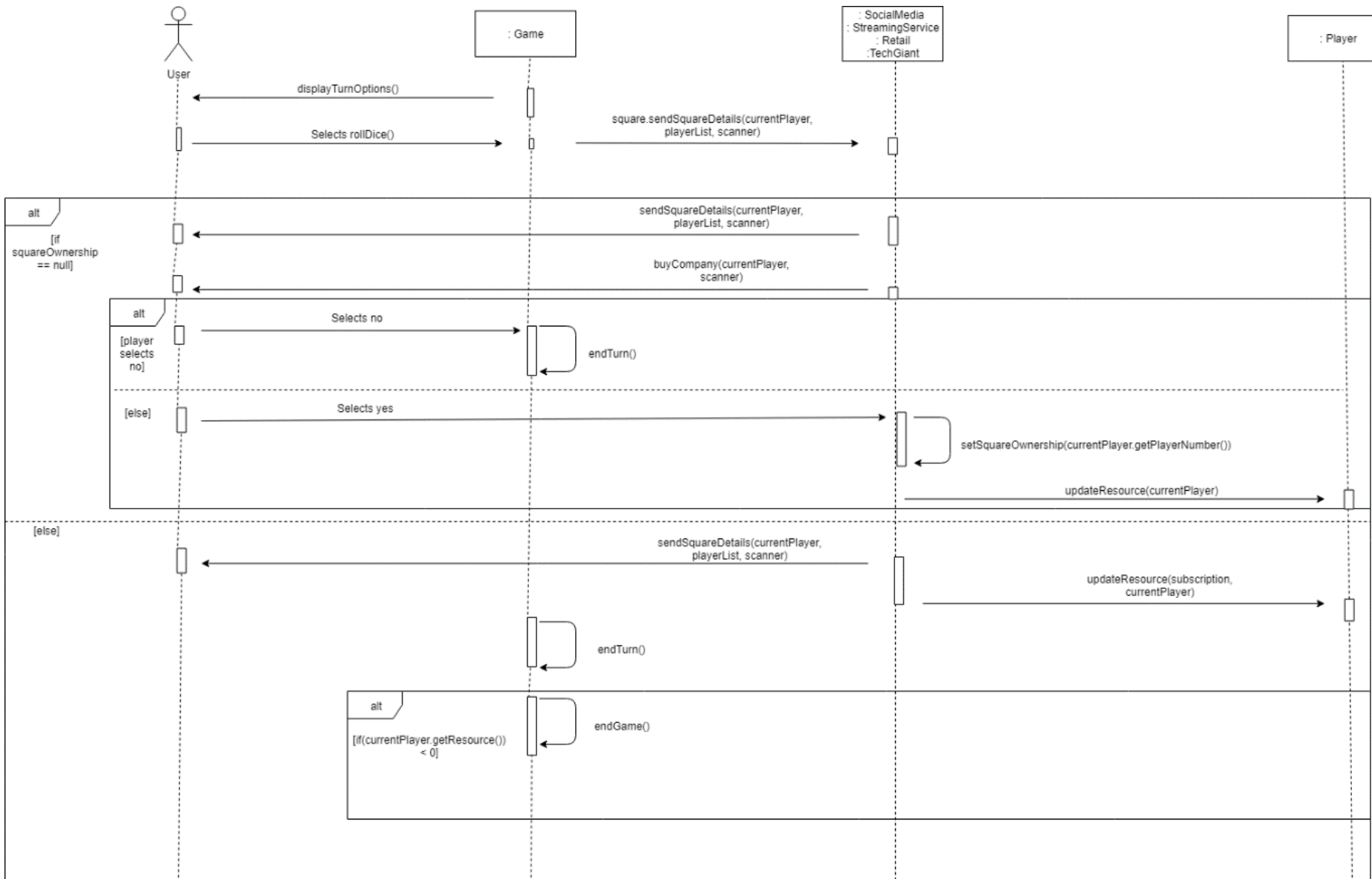
6. Pay Rent and Buy Company. Once a player lands on a square, if it is unowned then the buyCompany() method will be called to give the user the opportunity to buy it. If they opt not to buy it, then endTurn() is called; otherwise, setSquareOwnership() is called to change the square owner to the player and updateResource() is called to update that player's resources. If the square is already owned, then updateResource() is called to deduct the subscription owed to the square owner, depending on the number of offices/campuses on the square.
If the player's resources drop below zero as a result, endGame() is called.

REALISATION LK

Sequence diagram with actor **User**, and lifelines **: Game** and **: Player**.

- displayTurnOptions()
- Selects sellCompany()

loop [!doneSellCompany]
- Requests company name
- Inputs company name
- Error message
- displayTurnOptions()

loop [!confirmSellCompany]
- confirmSellCompany(doneSellCompany, loop)

alt [Player selects no]
- Selects no
- Requests company name

[else]
- Selects yes → updatePlayerResource()
- confirmSellCompany = true
- Asks whether player wishes to sell another company

- Selects yes
- Requests company name

- Selects no
- doneSellCompany = true

7. Sell Company. If a user selects sellCompany() from the displayTurnOptions() menu, then a series of system outputs are looped until the player indicates they are done selling companies.
The user is asked to input the company name, which is checked against the companies owned by the player. If the input does not match the name of any owned companies, then an error message is displayed and the loop continues.
If the user input does match an owned company, the user is asked to confirm sale of this company. If they select no, then they are returned to the top of the outer loop. If the player selects yes, then updatePlayerResource() is called to update the player's resources accordingly.
confirmSellCompany is set to true, closing the inner loop requesting confirmation of the sale. The system outputs a message asking if the player would like to sell another company. If they select yes, then the outer loop continues. If they select no, then doneSellCompany is set to true and the loop closes, returning the player to the displayTurnOptions() menu.

REALISATION LK

User

: Game

: Player

Lands on
Investment Round

updatePlayerPosition(movement,
currentPlayer)
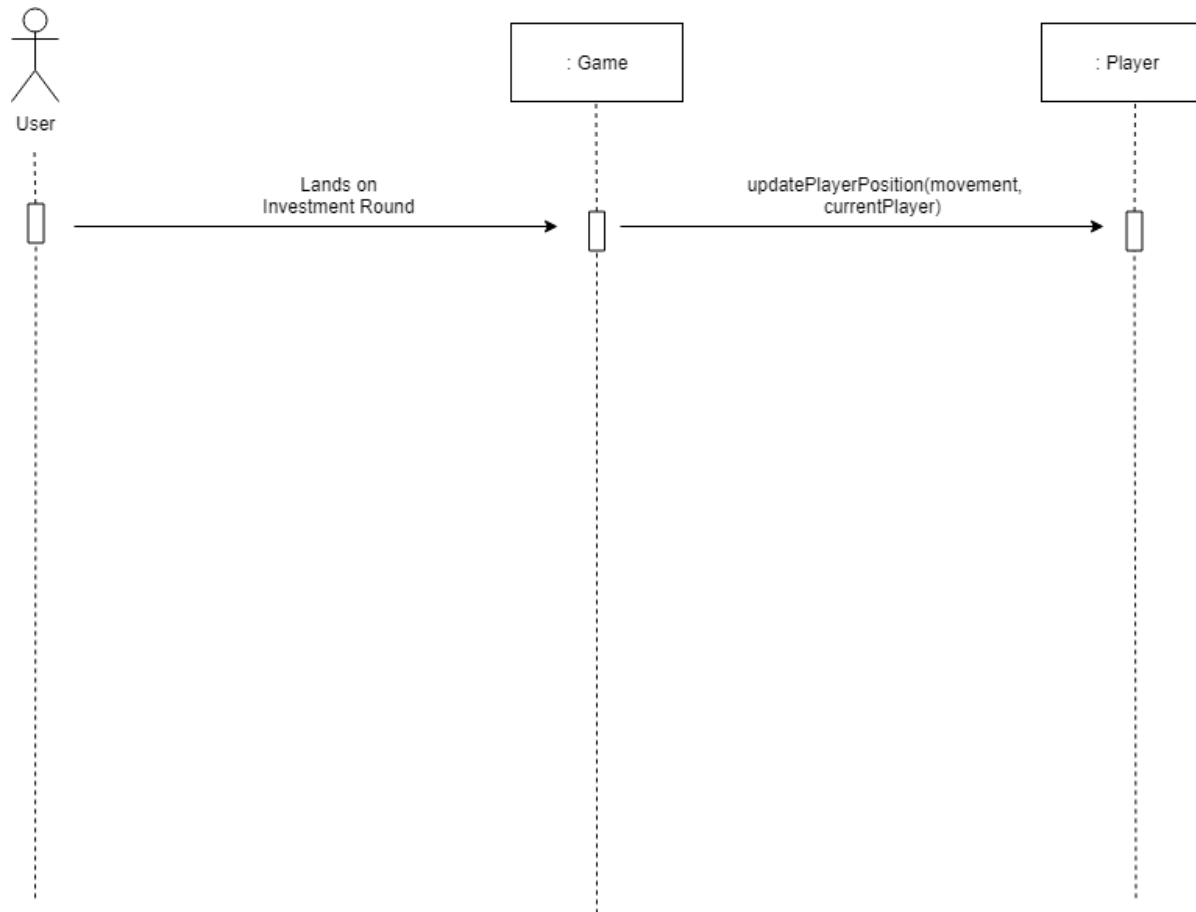
8. Investment Round. updatePlayerPosition() is called during each movement. If the player passes the Investment Round during this repositioning, then they are given more resources.

<<TECHNOPOLY >>

Test Plan

Document Change History

Version Number	Date	Contributor	Description
V1.0	06/02/2019	C.H, J,M. L,K. E,H. H,H.	

1   Introduction

The Test Plan's objective is to communicate the testing approach to team members. This includes the Projects objectives, scope, schedule, risks, and approach. This document will clearly identify what the test deliverables will be and what is deemed in and out of scope.

1.1   Objectives

Technopoly is a text-based game based on the traditional monopoly game, of which is to be played via the console of the development environment. The game is a new product written with Java. The test team is responsible for developing & testing the product, and ensuring a working system that satisfies the customer's requirements.

2 Scope

Phase one will begin with the creation of unit tests before the game-code is written. These unit tests will comprise of individual units/components of the Technopoly software, in order to validate that each unit is functional. Each class will have their own individual class that will test the individual units.

Phase two will consist of Peer testing, users other than the developer will QA the code to give a fresh perspective. Each person within the team shall QA the work of another developer. Any respective changes will be conducted by the original developer.

Phase three will consist of Usability testing, whereby the Technopoly will undergo its Alpha testing stage. The Alpha testers will play the game for one hour.

2.1   Assumptions/Risks

Assumptions

Delivery of the product is in format that the test team can check, i.e. .java files.

2.2   Risks

#	Risk	Impact	Trigger	Mitigation Plan

1	The usability testing focus group may require greater functionality within the system.	High	Delays in implementation date	Each iteration, functionality will be closely monitored. Priorities will be set and discussed by stakeholders. Since we are constrained by a fixed deadline, implementation of such improvements will be prioritized based on simplest to implement.
2	Changes to the functionality may negate the unit tests already written.	High	Loss of all unit tests.	
3	Developers may take different approaches towards the code development, and thus integration may become troublesome when tested.	Medium	Product did not get delivered on schedule	Unit testing each area will allow us to identify which areas are causing the failures. Allowing developers to correct them quickly.

3 Test Approach

The project is following an agile approach, with iterations weekly. Furthermore, exploratory testing will be heavily used within the testing team. Benefits of this approach include:

1. Find bugs that automated tests (JUnit) miss.
2. Improve the speed of our test cycles as it reduces the unnecessary documentation as per the Agile Manifesto- "*working software over comprehensive documentation*".

JUnit testing will play a large part of the testing as the team has never used this type of tool and will be learning as they go.

3.1   Test Automation

Automated unit tests are part of the development process, all automated tests regarding the Technopoly project consist of Junit testing. The fundamental idea behind this approach is to '*first test and then code*'.

3.2   Test Environment

- Java IDE console
- JUnit with Mockito

4   Milestones / Deliverables

Task Name	Start	Finish	Effort
Test Planning	03/03/2019	05/03/2019	1 d
Review Requirements documents	03/03/2019	04/03/2019	1 d
Create initial test estimates	04/03/2019	05/03/2019	1 d
First deploy to test environment	05/03/2019	06/03/2019	1 d
Functional testing – Iteration 1	06/03/2019	07/03/2019	1 d
Iteration 2 deploy to test environment	07/03/2019	08/03/2019	1 d
Functional testing – Iteration 2	08/03/2019	09/03/2019	1 d
Usability testing	09/03/2019	11/03/2019	2 d
Resolution of final defects and final build testing	11/03/2019	13/03/2019	2 d
Deploy to Staging environment	13/03/2019	14/03/2019	1 d
Release to Production	14/03/2019	14/03/2019	1 d

5   Test Methodology

5.1   Purpose

5.1.1   Overview
The two primary forms of testing carried out will be 1) Usability testing, and 2) Unit testing.

5.1.2   Usability Testing
Usability testing is to ensure that the features will meet the standards expected by potential users. This testing will be carried out by non-testers, a focus group will be used and their feedback will assist with our report to the project team.

5.1.3   Unit Testing
Unit Testing is carried out by the developer during code development to safeguard working functionality, and code coverage. Each Unit Test will test the smallest testable components. It is conducted before any code is written as per Test Driven Development principles.

6   Roles Responsibilities

Name	Role
Chris Healey	Test Manager
Luke Kelly	Tester
Ethan Harbinson	Tester
Hugo Harbinson	Tester
James MacCreanor	Tester

Meeting Minutes

Meeting Title: Planning Meeting	**Meeting #:** 1

Date & Location:	24/01/2019, CSB (First Derivatives Project Room)
Attendees:	Hugo Harbinson, Ethan Harbinson, James MacCreanor, Luke Kelly, Chris Healey
Apologies:	
Absent:	

Agenda:

#1: Project Timeline

Summary of Discussion:

Project Timeline

Luke raised the issue of the project timeline. He noted that the following tasks would need to be planned:

i.   Use-case modelling;
ii.   Domain modelling;
iii. Sequence modelling;
iv.   Class modelling;
v.   Coding;
vi.   Component integration;
vii. Test plan creation and testing;
viii. Drafting of the final report; and
ix.   Recording of the demo video.

Luke suggested that the steps i.-vii. will likely need to be iterated a number of times in line with an iterative approach and the group agreed. Ethan suggested that use-case modelling and domain modelling could be completed in Week 3.

Hugo noted that completion of the sequence models and class model could be done in early-mid Week 4, with a view to beginning coding at the end of Week 4. Hugo also noted that the use-case modelling work should include dividing components amongst the group.

The group agreed these points. Component integration was scheduled for the start of Week 5, with the end of Week 5/all of Week 6 being slotted for iteration/redrafting of the various documentation and/or code.

Weeks 6 and 7 were set aside for drafting the test plan, running tests and adapting the code/design documents as necessary.
The end of Week 7 and beginning of Week 8 were assigned to drafting the final report and the end of Week 8 assigned to making the demo video.

Luke noted that this schedule leaves a week for slippage in the various tasks.

The next meeting was agreed to be Monday 28 January at 10:00 in the First Derivatives room, to develop use-cases.

Action Points:
- Luke to publish a draft timeline for comment;
- Luke to circulate minutes of Planning meeting for comment.

Meeting Minutes

Meeting Title: Planning Meeting   **Meeting #: 2**

Date & Location:	28/01/2019, CSB (First Derivatives Project Room)
Attendees:	Hugo Harbinson, Ethan Harbinson, James MacCreanor, Luke Kelly, Chris Healey
Apologies:	
Absent:	

Agenda:

#1 Use Case Diagram
#2 Domain Modelling

Summary of Discussion:

Project Timeline

Ethan mocked up the beginning of a use case diagram.

James pointed out there are 2 types of actor, administrator and player.

Hugo clarified administrator will be for system maintenance etc. and player will be playing the game.

Only 2 actors, player and administrator

Use Case diagram:

1. Player enters number of players
2. Player enters name
3. Player rolls dice
4. Player moves to square – decided this is included
5. Show resources - decided this is separate ellipse to sq. details and options
6. Show Square details and options.
7. Discussion on base square case, Luke suggests non property squares are extensions of the property squares. Hugo points out that the base sq. is a free parking type sq. because there are no options so the display sq. details is actually an extension of land on sq.
8. Display square details extends to Transaction
9. Transaction includes; buy sq. collect pass go, pay fine etc.
10. A player must make a decision on some transactions e.g. buy sq. but not others e.g. pay rent
11. James points out we must display the resources again
12. Resources display the properties owned by the player
13. End game extends from transactions when a player goes bust
14. Also end game always an option for player
15. Decided that display sq. details includes what group the sq. is in and what is owned in this group.
16. Need to include develop sq. from the player
17. Decided the player should always have to roll the dice before they do anything
18. Then have an end turn option to allow for property development in a turn.

Domain model includes the objects and the action for each.

Objects include:

1. Players
2. Dice
3. Squares
4. Properties of squares
5. Player Resources i.e. money
6. Board object, displaying the current state of play

Using a skeleton use case diagram to begin mapping.

James makes the point that a transaction will interact with the player resources.

Squares in the domain diagram will therefore interact with resources to represent transactions.

Square has an owner so a square cannot be owned by multiple players.

Start made to domain diagram

Action Points:
   1. **Meet 29th January 2pm**
   2. **Finish Domain diagram**
   3. **Sequence model**

Meeting Minutes

Meeting Title: Domain Diagram Meeting	**Meeting #: 3**

Date & Location:	29/01/2019, CSB (First Derivatives Project Room)
Attendees:	Hugo Harbinson, Ethan Harbinson, James MacCreanor, Chris Healey
Apologies:	Luke Kelly
Absent:	

Agenda:

#1 Domain Modelling

Summary of Discussion:

Brief meeting to design the domain model.
Agreed to use generic terms i.e. house/ hotel until we have designed the game.
From the use case diagram, we added a player which interacts with resources.
Decided the player should interact with the game class which will hold the game play modules, e.g. number of players etc.
This interacts with the Die class as a player has to roll the die to participate in the game.
Board will hold the squares and their positions.
Squares class will be a child of board with the square specific details.
Houses and hotels will be conditional on the property square so will interact with the square class.
This gives us a first iteration to begin sequence modelling, will need to be updated as we develop the game logic.

Action Points:
1. **Sequence diagrams**
2. **Class Diagram UML**

Meeting Minutes   C.H.

Meeting Title: Sprint Meeting	**Meeting #: 4**

Date & Location:	11/03/2019, CSB (First Derivatives Project Room)
Attendees:	Chris Healey, Hugo Harbinson, Ethan Harbinson, James MacCreanor, Luke Kelly
Apologies:	
Absent:	

Agenda:

#1 Bugs
#2 Reports

Summary of Discussion:

19. Ethan, Hugo, and Chris were discussing how they could not find the location of the bug whereby Netflix square is assigned ownership to Player 1 immediately. Luke arrives and states that he put it in as dummy test data, and proceeds to remove it from the code.
20. Luke discussed his bug fix for when the game would crash if you tried to sell another property when you have none left.
21. Luke discussed whether sendSquareDetails() method is required, Hugo pointed out that it needs to be in each subclass to send to array every square has their own table.
22. Luke mentions that he stills needs to refactor the Game class.
23. Discussion between Luke and Chris regarding testing the Game class.
24. Hugo and James discussed the implementation of business rules for the game.
25. James mentioned how DevelopBusiness method is no longer working.
26. James mentioned how he made an alteration on the class diagram.
27. James and Chris had a discussion on how we will set up the presentation, i.e. three laptops adjacent to each other. Each laptop at a different stage of the game, such as Laptop 1-Start, Laptop 2- Middle of the game, and Laptop 3- Nearing final stage.
28. Ethan volunteered to record the demo as he has the recording software OBS already installed.
29. Chris stated that the Test Report is near completion, and he can create some diagrams if needed.
30. Hugo asked if someone can review his Requirement specifications, Chris offered to do so.

31. James volunteered to write the design report, he and Hugo discussed the formatting.

Action Points:
   1. System Test the game
   2.Finish Design report
   3.Finish Test Report