Software Engineering Group Project

By Jun, Jihye, Ye-Rang, Georgios, Ibi

Team 32

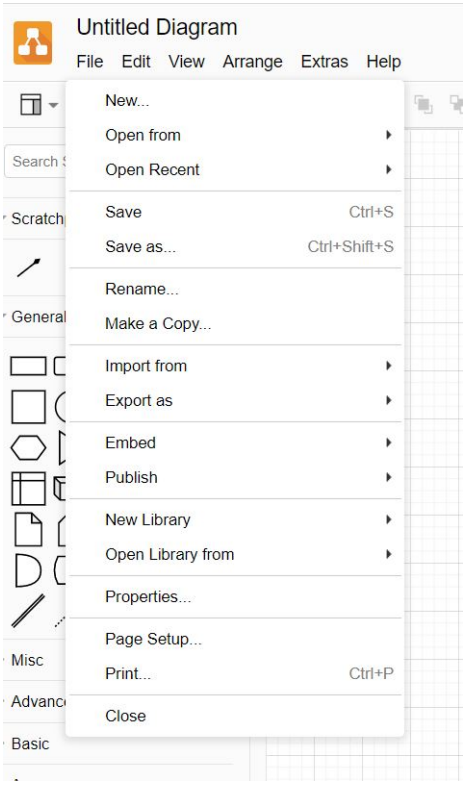
Contents

List of Figures

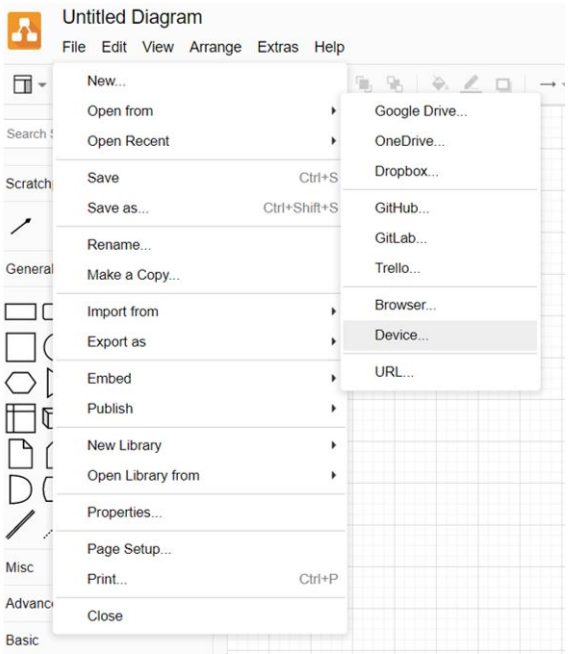
Viewing Class Diagrams

Class diagrams later on in the report may be hard to read. All class diagrams were created using Draw.io[19]. To see a clear view of the class diagram, I highly recommend opening the class diagrams in Draw.io. All class diagram files are located in the folder called "Class Diagrams". The following is a tutorial on how to open these files.

First, navigate to Draw.io and click "File" located on the top left.



Next, hover the mouse over "Open from" and select "Device".



This should open up file explorer. Lastly, select the class diagram to open. The class diagrams have the extension ".drawio".

Planning Phase 1

Team Number: 32
Planning Phase Lead: Jun Baek
Planning start date: 6th February 2020
Planning end data: 13th February 2020

The primary objective of the first week is to get to know each other. Doing so, we built up team cohesion and, together, read through the assignment. Some members did not understand the assignment fully, so members that did understand the assignment tried to answer every question. As a result of very good questions, members that thought they had understood the assignment found that they did not. Here are some of the key sticking points:

- Rule 15 states that "Players ... may not borrow money from the bank", but rule 22 goes to state "The bank will pay the player ... ". These two rules seem to contradict.
- What if a player mortgages a property that has houses/hotels already built on it? Will the bank only pay half the total value of the property, or only half the original price of the property?
- Rule 2 states that a player is designated as the "banker". However, we would argue that being the banker is no fun and quite tedious. As we are creating a electronic implementation of Watson Game, we could have the computer be the banker.
- Under section 7 named "Integrity of the game", it mentions that the "bank is always able to pay the players" as they can "generate new notes". If this is the case then, combined with the point above, is there a point in having a bank in the first place?

Questions surrounding these points have been sent to the customer via an email for clarification.

Whilst explaining the assignment, we decided it would be useful to have a visual reference. Below is the diagram that was created, which is also the first design prototype of our product:

Figure 2.1: First Design Prototype

Figure 2.1 was created to help us explain the assignment. It also gives us a starting point for designing our software. Each colour represents the group that property belongs to.

Overall, this was a successful sprint as by the end of the week everyone understood the assignment and we all got along well. Next week's plan is to finish the planning phase.

2.1 Individual Key Contributions

Team Member	Key Contribution(s)
Jun	Participated in team meeting
Jihye	Participated in team meeting
Ye-Rang	Participated in team meeting
Ibi	Participated in team meeting
Georgios	Participated in team meeting

Planning phase 2

Team Number: 32
Planning Phase Lead: Ye-Rang Lee
Planning start date: 13th February 2020
Planning end date: 20th February 2020

The main objective for this sprint is to finish the planning phase of the project. First we tried to assess risks but found our discussions to be too broad. Instead, we moved on to try and create a PERT chart but we were having difficulty in assigning nodes to members. However, we found the planning process to go very smoothly after assigning roles. Only then did we create a PERT chart, and following this we identified risks and how to manage them.

Following this line of logic made the planning phase to go smoothly because creating the PERT chart was made easier due having assigned roles, which in turn made identifying risks easier due to the PERT chart. For example, a member whose role is to code was given nodes in the PERT chart that was related to coding. From there, we could better identify risks since we know whose doing what tasks e.g., does a member have a planned holiday? If so, then we know which node in the PERT chart will be affected and thus have a better plan on how to mitigate the problem.

Assign Roles $\longrightarrow$ PERT Chart $\longrightarrow$ Identify Risks

Figure 3.1: Planning Logic

Below is the PERT chart that we created:

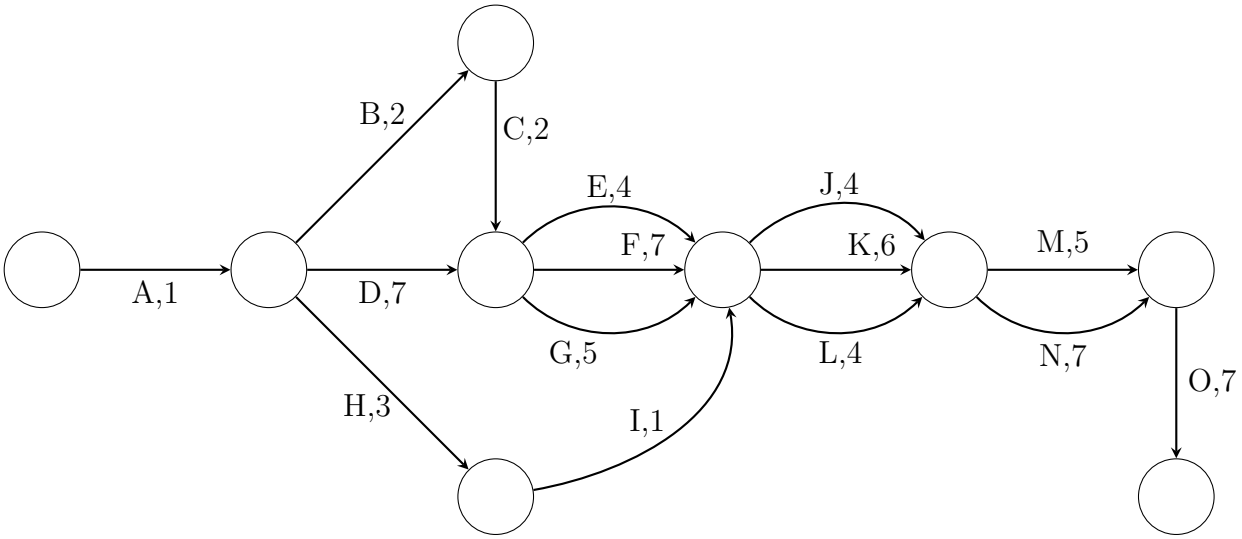


Figure 3.2: Project Plan: PERT Chart

Tasks	Days				Critical Path
	Earliest Start	Earliest Finish	Latest Start	Latest Finish	
A	0	1	0	1	Yes
B	1	3	4	6	No
C	3	5	6	8	No
D	1	8	1	8	Yes
E	8	12	11	15	No
F	8	15	8	15	Yes
G	8	13	10	15	No
H	1	4	11	14	No
I	4	5	14	15	No
J	15	19	17	21	No
K	15	21	15	21	Yes
L	15	19	17	21	No
M	21	26	23	28	No
N	21	28	21	28	Yes
O	28	35	28	35	Yes

Task	Description
A	Meet up and get to know each other
B	Plan out tasks and roles by creating a PERT chart
C	Carry out risk assessment
D	Code first prototype
E	Unit testing
F	Code in game rules
G	Code excel data to board environment
H	Design game board
I	Design characters
J	System testing
K	Code "full" game mode
L	Code "abridged" game mode
M	Create and code menu
N	Code game agent
O	Test the whole game

Figure 3.3: Project Plan Descriptions and Calculations

Critical Path: A, D, F, K, N, O

3.1 Individual Key Contributions

Team Member	Key Contribution(s)
Jun	Assigned roles
Ibi	Helped assigning roles, created PERT chart and calculations
Ye-Rang	Helped assigning roles
Jihye	Helped assigning roles
Georgios	Helped assigning roles

Sprint 1

Team Number: 32
Sprint Technical Lead: Jihye Ahn
Sprint start date: 13th February 2020
Sprint end date: 20th February 2020

4.1 User Stories

Produce the first working prototype of Property Tycoon. This prototype should display the game board and provide the means of the user to roll the dice and move the player's token by the number rolled.

4.2 Task Cards

Scoring scale: 1 to 10, with 10 having the highest priority and 1 the lowest. A task card with priority 10 means that it must be completed this sprint, otherwise the prototype will not work or make enough progress from the previous prototype. Alternatively, a 10 can also be a feature that the customer has recommended. A card with priority 1 means that it does not have to be completed this sprint, and without it, it will not impact the current prototype.

1. Priority: 10, Complexity: 8

   When the game is started, there shall be an 11x11 game board because there are 40 squares, as specified in "PropertyTycoonBoardData.xlsx". The board must facilitate a means of using an image to display the player's character. Each square should be able to have a background image.

2. Priority: 10, Complexity: 5

   There must be a method to roll the dice by pressing a button. The board shall display the updated number rolled on each die. It must be able to display the dice by any common image forms such as png, jpg, etc. In other words, we do not want to draw and redraw the dice by hand.

3. Priority: 7, Complexity: 4

   After the player has rolled the dice, their token must move around the board by the correct amount. More specifically, by the sum of the two numbers shown on each die.

4.3 Requirements Analysis

4.3.1 Functional Requirements

F1 - Mandatory

At the start of the game, the board shall display an 11x11 board. All players must be initialised at the starting position of the game (square 1, "GO").

F2 - Mandatory

Each square on the board shall be a StackPane as it allows us to display all player tokens on a square. Also, we can set the background of a StackPane to a picture of our choice.

F3 - Mandatory

Two dices that randomly chooses a number between 1-6. There shall be a method that returns the sum of the number rolled by each die.

F4 - Mandatory

There must be a button named "Roll Dice" that, when pressed, rolls the dice. The face of each die that is displayed must be appropriately updated.

F5 - Mandatory

When a player presses "Roll Dice" their token shall move around the board by the sum the face of each die. If a player's position is 39 and they roll a 4, then their board position must be updated to $39+4-40 = 3$.

F6 - Mandatory

After rolling the dice, the player's token will automatically move to the correct square.

F7 - Mandatory

The buttons shall be clearly visible, with all buttons located at the bottom of the window.

F8 - Mandatory

The balance of all players shall be located on the right side of the window.

4.3.2 Non-functional Requirements

NF1 - Mandatory

The software shall be written in the Java programming language as all members are comfortable and familiar with Java. Java is widely supported, which allows for maximum portability should the software be extended.

NF2 - Desirable

The software should not crash whilst a game is in progress.

4.3.3 Domain Requirements

D1 - Mandatory

Property Tycoon is a *family* board game, notice the word family. This means that people of all ages should be able to play it. Our interface shall be simple enough for young children to use.

Action needed: The software shall be simple, which means that the game should be as automated as possible. For example, there should not be a button to move the token manually after the dice has been rolled, rather the token should move automatically to the required position. See F6, F7 and F8.

D2 - Mandatory

Property Tycoon is a family game, so the interface shall be clear.

Action needed: Elements of the board should be placed logically, i.e. all buttons on the same area, easy to read text, etc.

4.4 Use Case Diagram

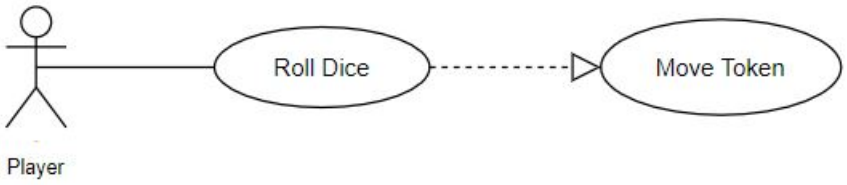


Figure 4.1: First Prototype: Use Case Diagram

4.5 High Level Diagram

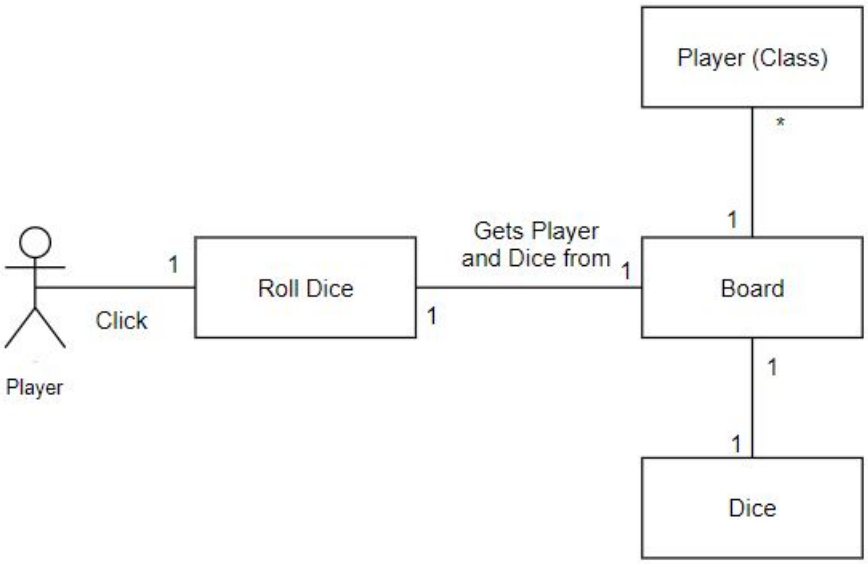


Figure 4.2: First Prototype: High Level Diagram

4.6 Class Diagram

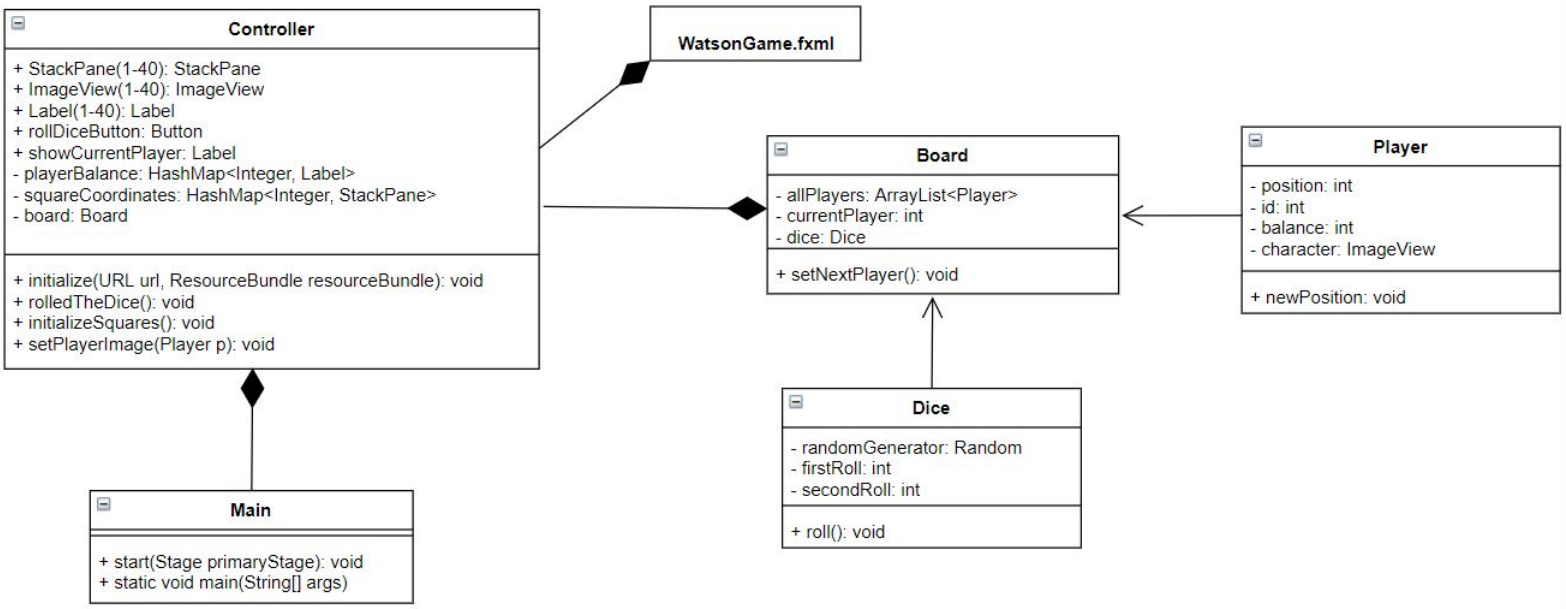


Figure 4.3: First Prototype: Class Diagram

In order to keep the class diagram concise and presentable, getter and setter methods have been omitted with permission from the customer. Note that they are included in the code and used in sequence diagrams even though they do may not appear in the class diagram.

The class diagram can be viewed clearly in Draw.io. To open our class diagrams, please read chapter 1. The first class diagram is located in the "Class Diagrams" folder, called "FirstClassDiagram.io".

4.7 Sequence Diagram

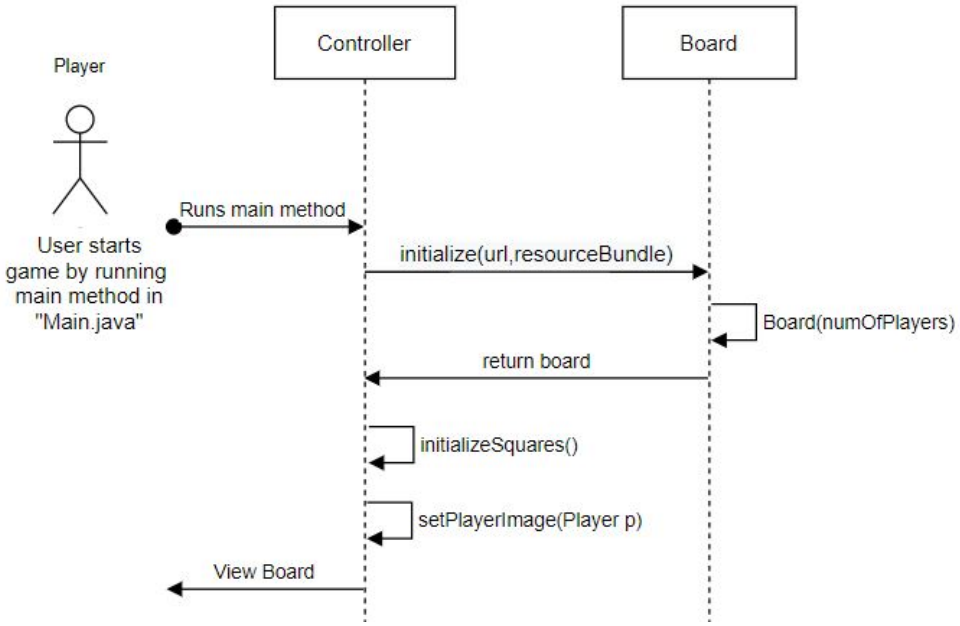


Figure 4.4: First Prototype Sequence Diagram: Starting Game

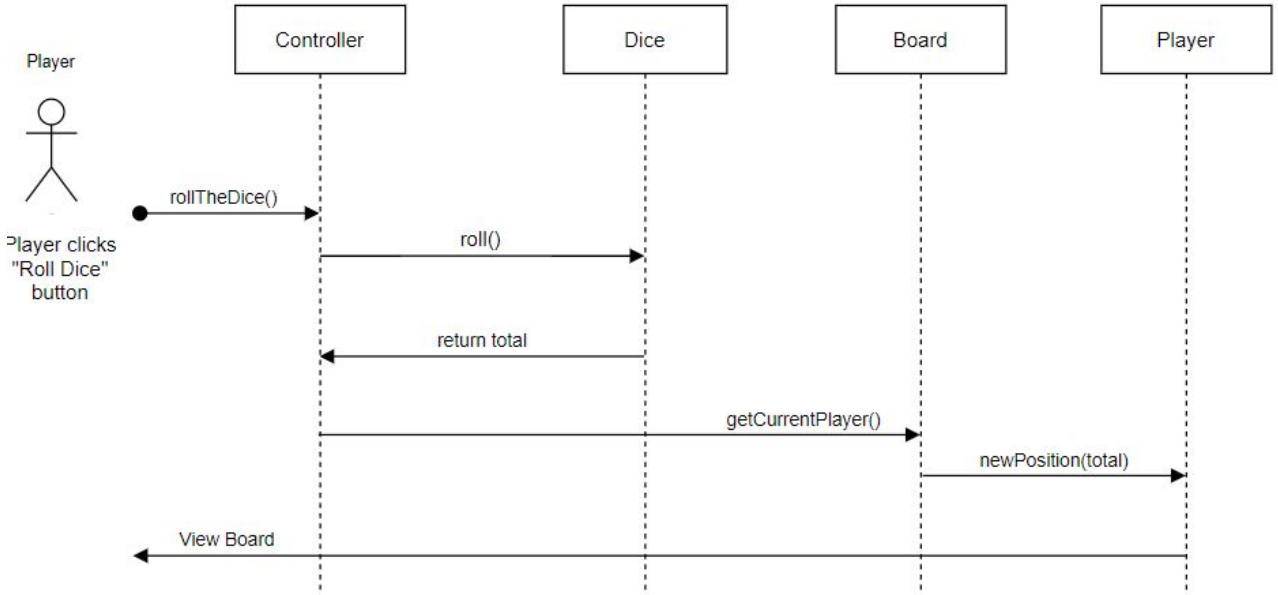
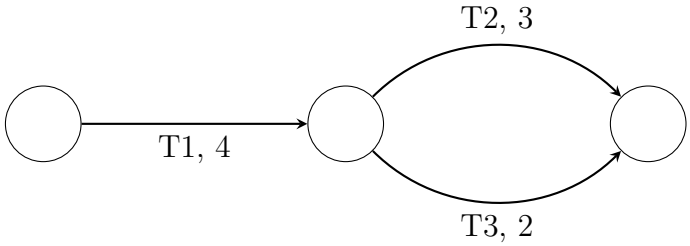


Figure 4.5: First Prototype Sequence Diagram: Clicking "Roll Dice"

4.8 PERT Chart



Task Card	Days				Critical Path
	Earliest Start	Earliest Finish	Latest Start	Latest Finish	
1	0	4	0	4	Yes
2	4	7	4	7	Yes
3	4	6	5	7	No

Figure 4.6: First Prototype: PERT Chart

Critical Path: T1, T2

4.9 Risk Management

Risk Identification	Causes and Likelihood	Mitigation	Monitoring
We will be writing code in Java. Some members may not be comfortable on Java and prefer a different programming language.	Likelihood: 2 Some members may be used to writing in a different programming language.	People who are working on the code base have a greater say on the programming language. If these people are not comfortable working with Java, then we have agreed to switch to Python.	Discuss how the code is going during every team meeting. Technical Director should oversee how Technical Support members are doing.
We will be using JavaFX to create GUI elements of the game. Some members may not be comfortable with using JavaFX.	Likelihood: 3 Members may not have any experience using JavaFX. Some members prefer using Swing over JavaFX.	If not enough progress is being made with the code, swap to using Swing.	Review progress during every team meeting. Technical Supports should update the Technical Director about their progress with using JavaFX.
Software Development may be delayed.	Likelihood: 1 Deadlines for other assessments coming up. As a result, focus of members may be elsewhere near these assessments.	Plan work to take into account future deadlines. Project Manager organises tasks with deadlines in mind.	Project manager discusses future tasks every team meeting.

Likelihood scores are between 1-5, with 1 being highly unlikely and 5 being highly likely.

4.10 Implementation of Task Cards

The work on the first prototype started by implementing the 11x11 board as a GUI. Problems arose when we realised that the GUI had been made using NetBean's built in Swing GUI software [7]. NetBean's Swing GUI software does not let you edit the code that created the image (see figure below), limiting functionality and ease of use.

Consequently, we needed to find a different approach. The problem described above was as a result of using an IDE specific GUI tool, so instead

we decided to code the GUI board by hand. This, however, created another issue where the coupling [8] of our code was far too tight. For example, simply removing a button from the GUI would result in the board's layout messing up, or the player's token going invisible, etc. This made progress hard it made debugging difficult meaning large amounts of time was spent fixing problems. This would not be a sustainable approach as the software becomes more complex.

It became apparent that we needed to separate the GUI of the board from the game logic. Hence, we went on to implement a "View - Controller - Model" structure by using JavaFX Scene Builder [9]. The view being "WatsonGame.fxml", whose task is to display all GUI components. The model being all other ".java" files not called "Controller.java" that make up the game logic. The controller being "Controller.java", which connects the GUI to the game logic.

The board consists of 11x11 StackPanes [10]. A StackPane allows JavaFX components to be placed on top of each other. The main functionality of a StackPane is that it can display multiple objects; this allows us to display every player token on a square. Each StackPane comes with a background attribute that lets us set a background image that will represent a square's colour group.

Every StackPane holds an ImageView [11] object, which in turn holds an Image [12] object. This image will hold the background image of every square. Note that the player class has an ImageView field that holds the image file containing its token. This is why we do not need to explicitly have 6 more ImageView objects in each StackPane.

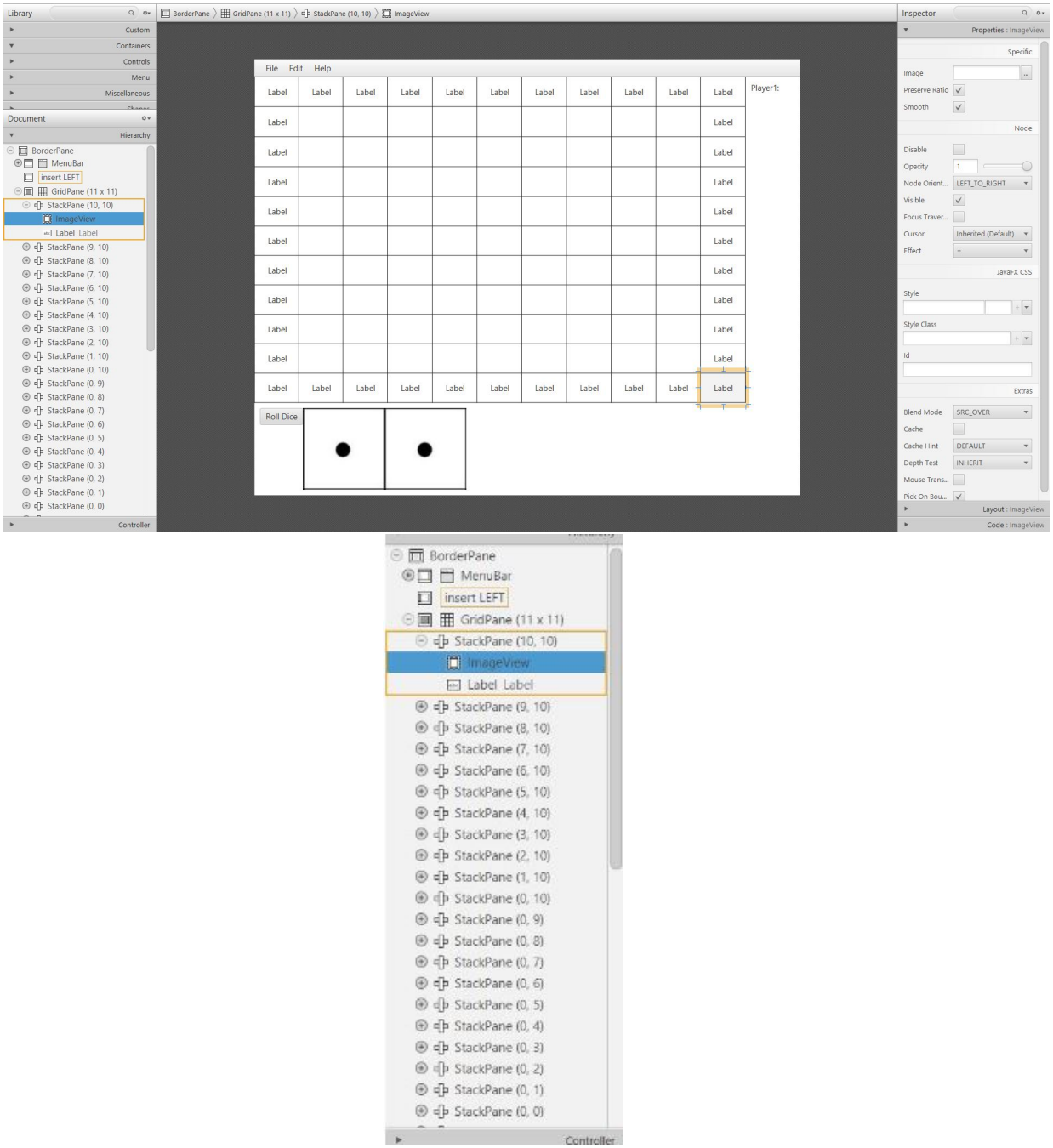


Figure 4.7: Board layout in JavaFX Scene Builder

4.11 Testing

4.11.1 Unit Level

Tests for: T3, F3
Check that a dice does not roll a number less than 1 or greater than 6. Additionally, check that we are summing the dice correctly. This test can be seen below:

```
@Test
public void sumOfDice() {
    Dice dice = new Dice();
    dice.roll();
    int x = dice.getFirstRoll();
    assertTrue( condition: (x >= 1) && (x <= 6)); // check dice does not roll less than 1 or greater than 6
    int y = dice.getSecondRoll();
    assertTrue( condition: (y >= 1) && (y <= 6)); // check dice does not roll less than 1 or greater than 6
    assertEquals( expected: x + y,  actual: dice.getFirstRoll() + dice.getSecondRoll());
}
```

Figure 4.8: Unit Test: Summation of Dice

Tests for: T3, F1, F5
A player's starting position should always be at position 1 ("GO"). Whenever a player rolls the dice, they should move by the correct amount and land on the corresponding square. The test below checks for both requirements.

```
@Test
public void newPosition() {
    Dice dice = new Dice();
    Player p1 = new Player( id: 1);
    int rolled = dice.roll();

    assertEquals( expected: 1, p1.getPosition());
    p1.newPosition(rolled);
    assertEquals( expected: 1 + rolled, p1.getPosition());
}
```

Tests for: F5
There is an edge case when the player rolls the dice. Since the sum of the dice is added to the player's, then what happens when the new position x is over 40? There is no square to facilitate this and we must make sure that

the correct square is chosen. This is achieved by checking if x is over 40, if so, then we do $x = x - 40$.
The smallest value on each die is 1 so the minimum value a player can roll is 2. Therefore, in the unit test, the player's position must be 39 so the player is guaranteed to go over 40. This test can be seen below:

```
@Test
public void newPositionOver40() {
    Dice dice = new Dice(); // create new instance of dice
    Player p1 = new Player( id: 1); // create new instance of player

    int rolled = dice.roll(); // roll the dice
    assertEquals( expected: 1, p1.getPosition()); // check that player's position starts at 1
    p1.setPosition(39); // set player's position to 39
    assertEquals( expected: 39, p1.getPosition());
    p1.newPosition(rolled); // update player's position  based on number rolled
    assertEquals( expected: 39 + rolled - 40, p1.getPosition());
}
```

Figure 4.9: Unit Test: Position Over 40

4.11.2 System Level

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
1	User starts the game.	Running main method in "main" class.	Upon creation, user should see an 11x11 board, with a single player token on the first square. There should also be a button called "Roll Dice", with images of two die beside it.	An 11x11 board created, with a single token on the first square. A button called "Roll Dice" with two die is displayed.	PASS
2	User starts the game.	Running main method in "main" class.	The face of each die should show the 1-sided face.	Face of each die shows 1-sided face.	PASS
3	User presses "Roll Dice" button.	A button click event on "Roll Dice".	The face of each die should equal the sum of the dices rolled (the number of squares the player moves).	The face of each die equals the sum of the dices rolled.	PASS
4	User presses "Roll Dice" button.	A button click event on "Roll Dice".	The player token must move around the board clockwise by the sum of the dices rolled.	Player token moved around the board clockwise by the sum of the dices rolled.	PASS
5	User presses "Roll Dice" button.	A button click event on "Roll Dice" when the player's token is about to pass the first square.	The player token must move to the correct spot on the board by the sum of the dices rolled.	Player token moved to the correct square by the sum of the dices rolled.	PASS
6	User presses "Roll Dice" button.	A button click event on "Roll Dice".	The player's token should be visible when it lands on a square with a background image.	Player's token is visible on top of a background image.	PASS

Test Number	Testing For
1	T1, T2, F1, F4, F6, F7, F8
3	T2, F4
4	T3, F3, F6
5	F5

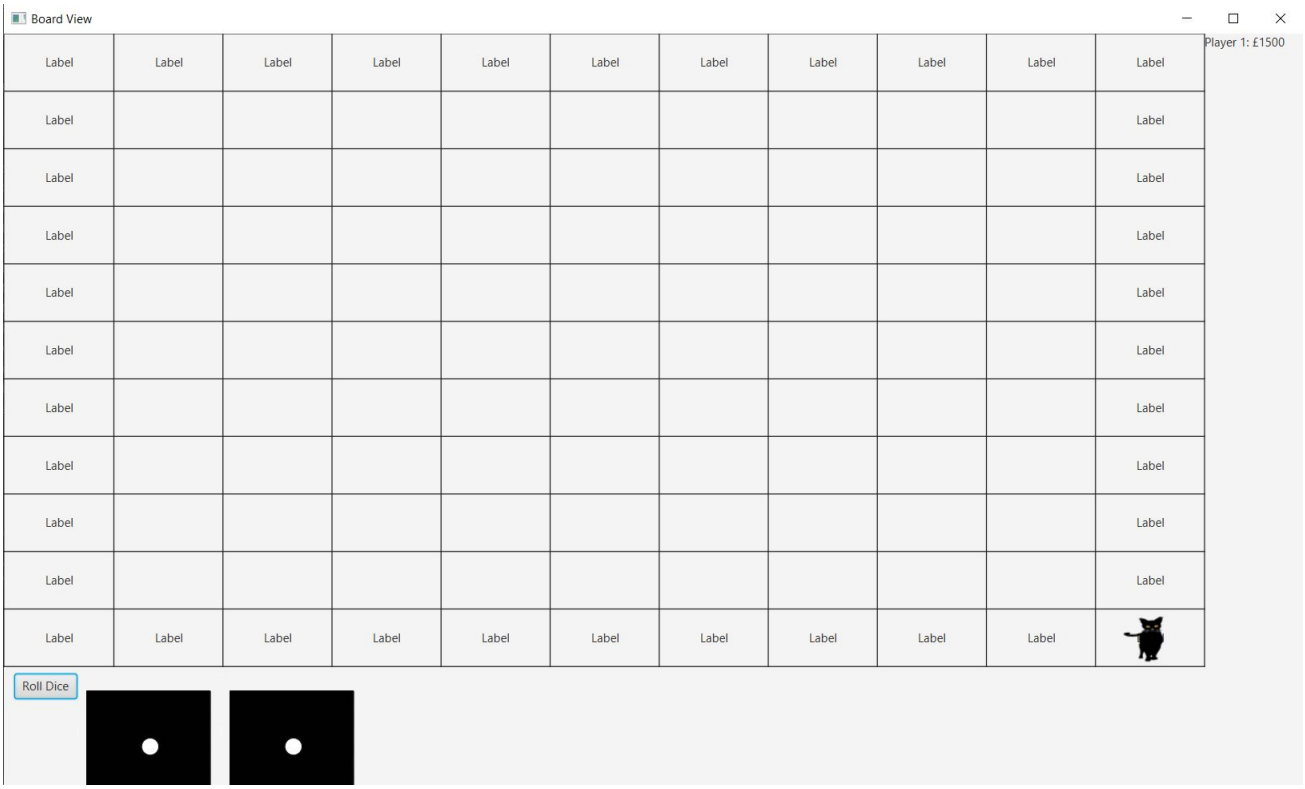


Figure 4.10: Board layout of the first prototype

4.12 Reflection

Apart from a few hurdles during the start of the sprint, this sprint was very successful as we have a working prototype. The prototype fulfilled every task card for the sprint. However, as aforementioned, there were problems with integrating NetBean's Swing GUI tool code with the rest of our code.

This NetBeans problem in 5.22 could have been avoided with better planning. For example, under "Non-Functional Requirements", we could have added the following requirement:

NF3 - Mandatory

The board shall be designed and created in the JavaFX SceneBuilder environment.

Simply stating the programming language did not suffice as designing GUIs often have their own separate package. This was an important learning experience and in the future we will improve our planning by taking into account the coding environments being used.

The code for the first prototype can be found in the submission folder, called "FirstPrototype".

4.13 Individual Key Contributions

Team Member	Key Contribution(s)
Jun	Helped revised PERT chart, created GUI (Task 1)
Ibi	Helped revised PERT chart, created GUI (Task 1), created "Dice" class
Ye-Rang	Helped revised PERT chart, created "Roll Dice" button (Task 2)
Jihye	Helped revised PERT chart, created GUI (Task 1)
Georgios	

Sprint 2

Team Number: 32
Sprint Technical Lead: Ibi
Sprint start date: 20th February 2020
Sprint end date: 5th March 2020

5.1 User Stories

Produce the second working prototype of Property Tycoon, which builds upon the first. This prototype must use an external data file that includes both the Property Tycoon board and card data. Using these external files, it should correctly hold and display the information of a square such as its: name, colour group, position, and so forth. The cards should be placed in a pack, with the ability to shuffle and pick the top card from the pack. Players in the game shall be able to take turns to roll the dice, with the current player clearly displayed. The balance of each player should be displayed on the screen. When starting the application, the user should be greeted with a button called "Create Game" that, when pressed, loads the board view.

5.2 Task Cards

Scoring scale: 1 to 10, with 10 having the highest priority and 1 the lowest. A task card with priority 10 means that it must be completed this sprint, otherwise the prototype will not work or make enough progress from the previous prototype. Alternatively, a 10 can also be a feature that the customer has recommended. A card with priority 1 means that it does not have to be completed this sprint, and without it, it will not impact the current prototype.

1. Priority: 5, Complexity: 7

Design the following characters: boot, smartphone, goblet, hat-stand, cat and spoon.

2. Priority: 10, Complexity: 10

   Use the Excel files to load the board and card data into the game. With respect to these files, correctly display all the information such as the name and colour group of a square on the board. The board data file must be converted into Squares that holds all the information. The card data file must be converted into Cards that hold the description and action of a card.

3. Priority: 8, Complexity: 6

   There shall be a button called "End Turn" that ends a player's turn. Once a player's turn has ended, the board shall choose the next player in line to take their turn.

4. Priority: 7, Complexity: 5

   The board shall be able to display multiple character tokens, with each player taking turns once they have finished their turn.

5. Priority: 4, Complexity: 4

   A player must not be able to end their turn without having rolled the dice. Naturally, a player must also not be able to keep rolling the dice after they have rolled.

6. Priority: 2, Complexity: 5

   The current balance of all players must be displayed, where every player has £1500 at the outset of the game.

7. Priority: 1, Complexity: 9

   When the application starts, there shall be a button called "Create Game" that loads the board view.

8. Priority: 5, Complexity: 2

   The cards, obtained after converting the card data file, shall be placed into a pack. This pack must provide functionality to shuffle and pick the top card from the pack.

5.3 Requirements Analysis

5.3.1 Functional Requirements

F1 - Desirable

Create and draw the following characters: boot, smartphone, goblet, hat-stand, cat and spoon in a drawing software. The images should be in one of the common image formats, jpg or png.

F2 - Mandatory

There shall be a button called "End Turn" that will end the current player's turn and pass the turn over to the next player. A player cannot end their turn without rolling the dice at least once.

F3 - Mandatory

At the start of the game, the board game data shall be loaded into the game. The names of each square shall be displayed, along with a matching background image with respect to their group.

F4 - Mandatory

At the start of the game, the card game data shall be split into "PotLuck" and "Opportunity Knocks", with the description and action to be stored separately in a single card.

F5 - Mandatory

The "PotLuck" and "Opportunity Knocks" card piles shall be shuffled at the outset of the game. A card should be picked from the top of the pile. The card should then be placed at the bottom of the corresponding pile.

F6 - Desirable

When the main method in main.java is called, the user should be greeted with a single button named "Create Game". When pressed, the board view of the game should be loaded.

F7 - Mandatory

At the start of the game, every player shall have £1500 to begin with.

F8 - Mandatory

The current player displayed shall be updated whenever a new player is taking their turn. It shall display the player's ID.

F9 - Mandatory

Every player shall be assigned a unique ID that identifies them from other players.

5.3.2 Non-Functional Requirements

NF1 - Mandatory

Convert external Excel files to Comma-Separated-Values (CSV) files, and use it as the file format from which the board and card data will be loaded from.

NF2 - Desirable

The balance of each player should be displayed using JavaFX labels.

NF3 - Mandatory

All background images of squares must be a JavaFX Image[12] object. A player will hold an ImageView[11] object containing an Image object of their character.

5.3.3 Domain Requirements

D1 - Desirable

All character and group images should be our own intellectual property. This is to avoid any lawsuits by our rivals claiming copyright ownership against us.

Action needed: use a drawing software to create our own images and board, as opposed to using images off the internet.

5.4 Use Case Diagram

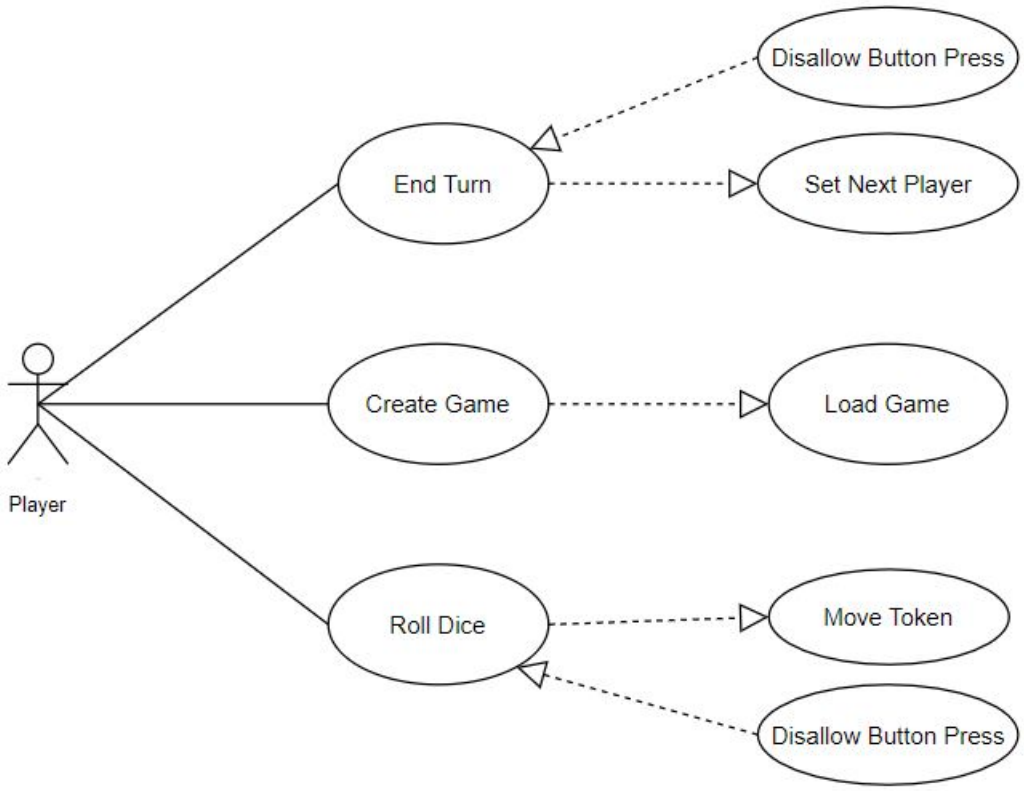


Figure 5.1: Second Prototype: Use Case Diagram

5.5 High Level Diagram

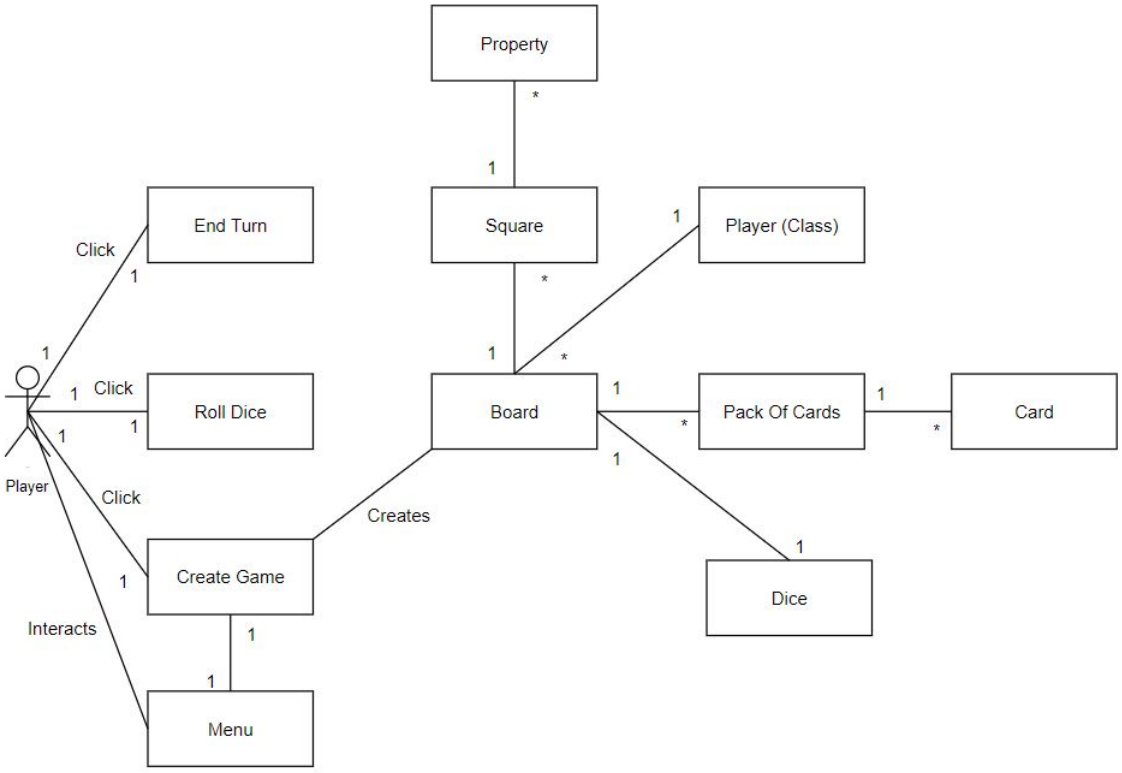


Figure 5.2: Second Prototype: High Level Diagram

5.6 Class Diagram

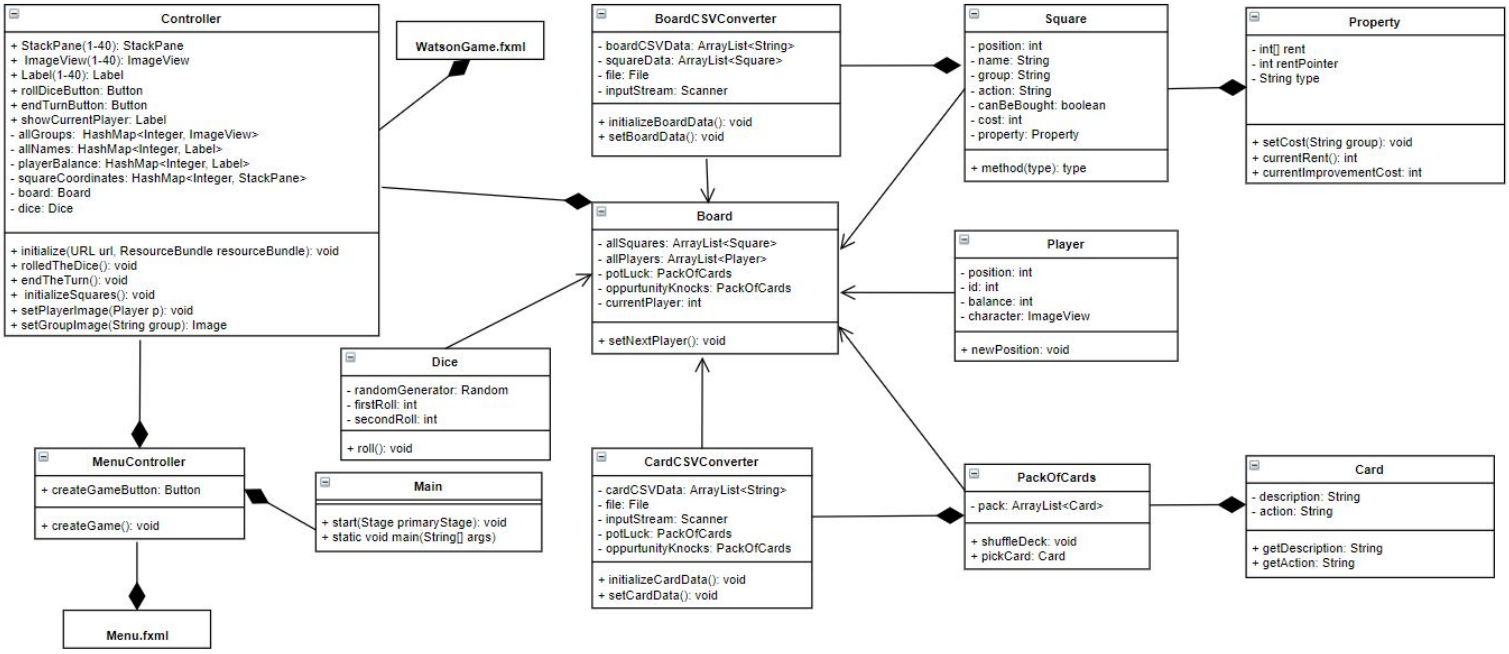


Figure 5.3: Second Prototype: Class Diagram

In order to keep the class diagram concise and presentable, getter and setter methods have been omitted with permission from the customer. Note that they are included in the code and used in sequence diagrams even though they do may not appear in the class diagram.

The class diagram can be viewed clearly in Draw.io. To open our class diagrams, please read chapter 1. The first class diagram is located in the "Class Diagrams" folder, called "SecondClassDiagram.io".

5.7 Sequence Diagram

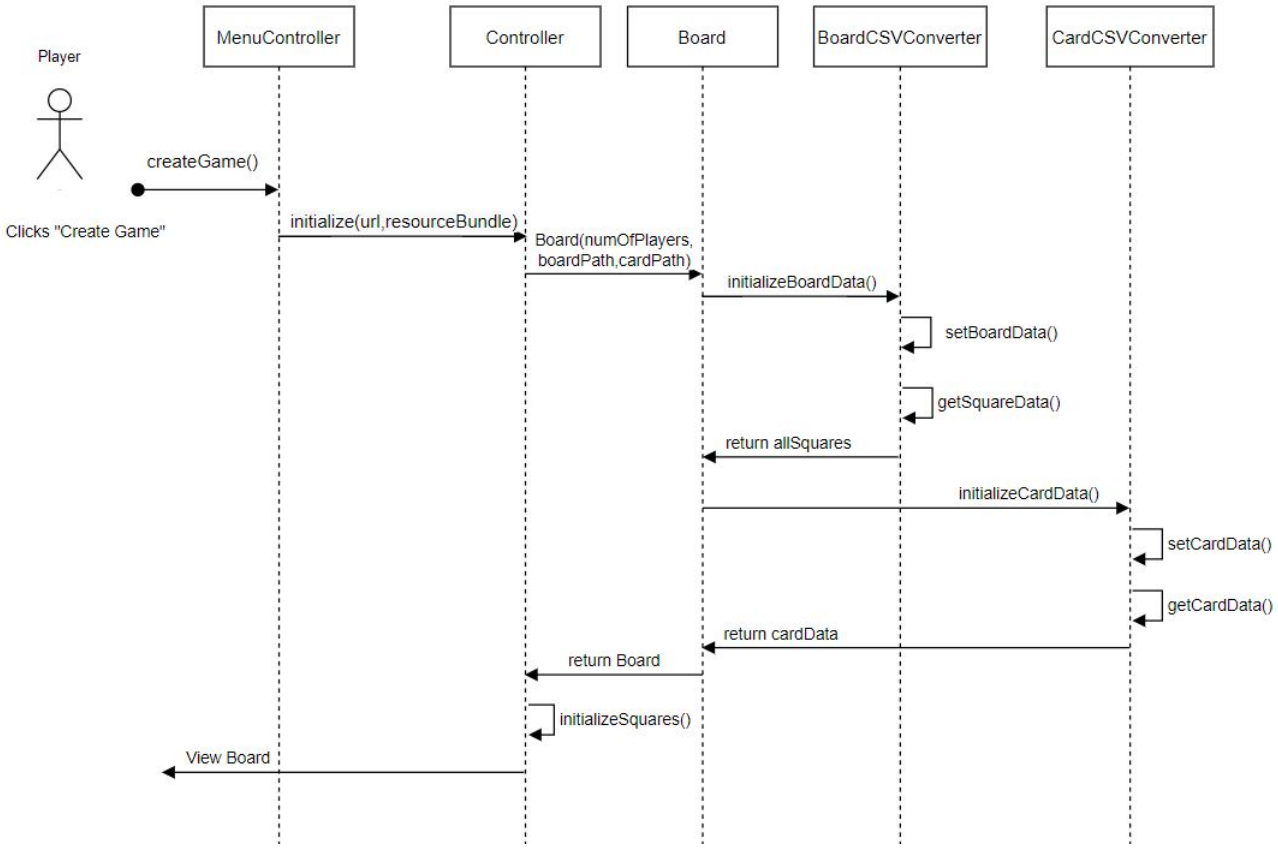


Figure 5.4: Sequence of action for when "Create Game" button is clicked

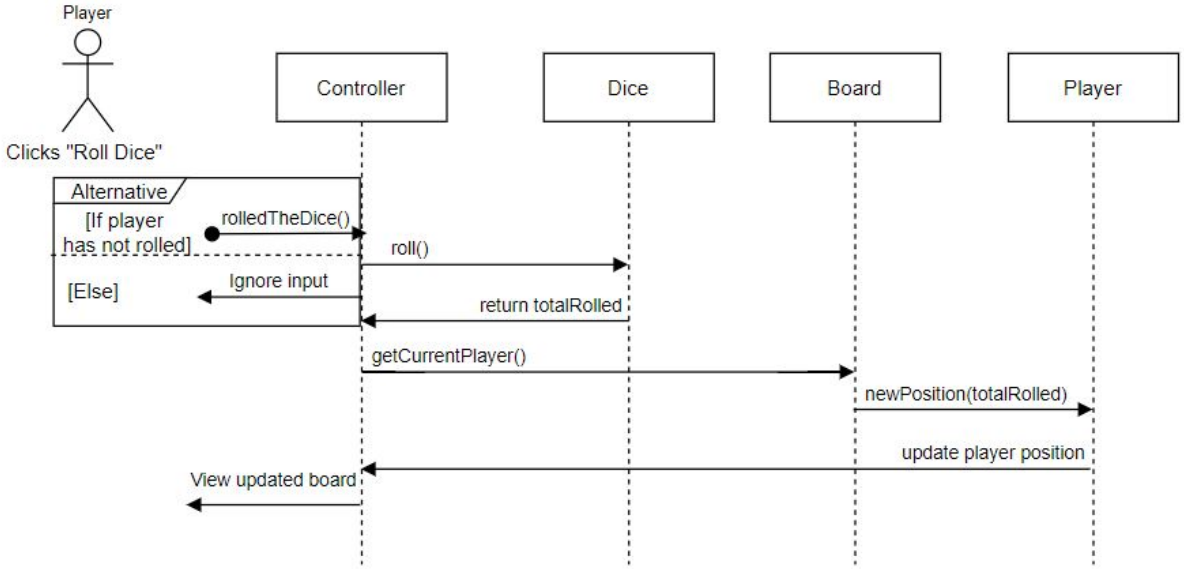


Figure 5.5: Sequence of action for when "Roll Dice" button is clicked

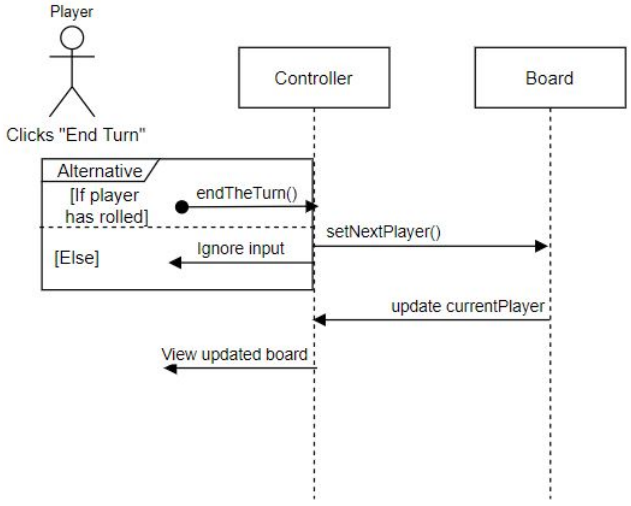
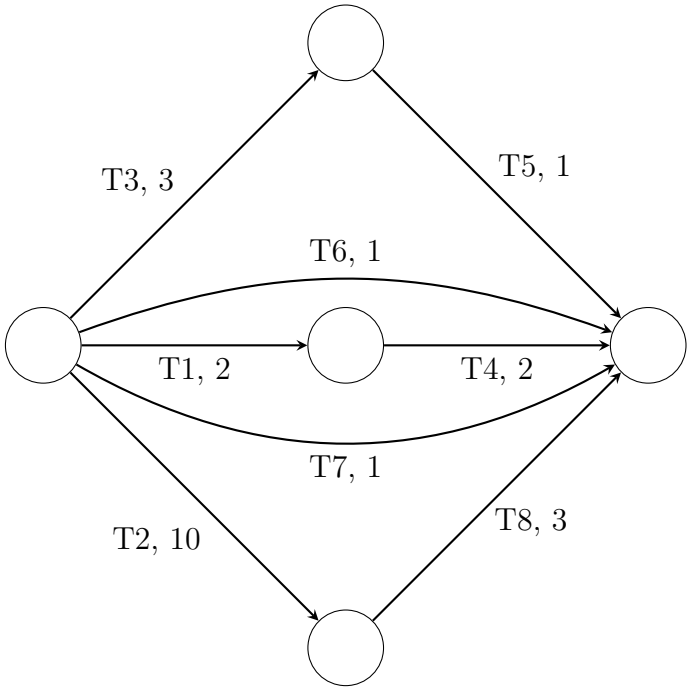


Figure 5.6: Sequence of action for when "End Turn" button is clicked

5.8 PERT Chart



Task Card	Days				Critical Path
	Earliest Start	Earliest Finish	Latest Start	Latest Finish	
1	0	2	9	11	No
2	0	10	0	10	Yes
3	0	3	9	12	No
4	2	4	11	13	No
5	3	4	12	13	No
6	0	1	12	13	No
7	0	1	12	13	No
8	10	13	10	13	Yes

Figure 5.7: Second Prototype: PERT Chart

Critical Path: T2, T8

5.9 Risk Management

Risk Identification	Causes and Likelihood	Mitigation	Monitoring
We will be writing code in Java. Some members may not be comfortable on Java and prefer a different programming language.	Likelihood: 2 Some members may be used to writing in a different programming langage.	People who are working on the code base have a greater say on the programming language. If these people are not making good progress with Java, then we have agreed to switch to Python.	Discuss how the code is going during every team meeting. Technical Director should oversee how Technical Support members are doing.
We will be using JavaFX to create GUI elements of the game. Some members may not be comfortable with using JavaFX.	Likelihood: 1 Members may not have any experience using JavaFX. Some members prefer using Swing over JavaFX.	If not enough progress is being made with the code, swap to using Swing.	Review progress during every team meeting. Technical Supports should update the Technical Director about their progress with using JavaFX.
Software Development may be delayed.	Likelihood: 2 Deadlines for other assessments coming up. As a result, focus of members may be elsewhere near these assessments.	Plan work to take into account future deadlines. Project Manager organises tasks with deadlines in mind.	Project manager discusses future tasks and assigns tasks to each member during team meetings.
Members may have trouble completing a coding task, which can slow progress.	Likelihood: 2 Technical Supports having difficulty with completing a task.	Technical Support should notify the Technical Director for assistance.	Technical Supports will ask for support, if needed, from the Technical Director, during team meetings.

Likelihood scores are between 1-5, with 1 being highly unlikely and 5 being highly likely.

5.10 Implementation of Task Cards

The main feature of this sprint is to load the board and card data into the game logic. For this sprint, we will display the name and group of the square

using an appropriate picture with respect to the board data. As for the card data, we will show that the card data has been correctly loaded by printing the description and action of every card in each pile.

We decided to use the CSV data format because, in our opinion, it was the easiest format to manipulate. An excel file can be converted to ".csv" by using "Save As" and selecting "CSV".

Property Tycoon board data

					If can be bought		When improved				
					Cost	Rent (unimproved)	1 house	2 houses	3 houses	4 houses	1 hotel
Position	Space/property	Group	Action	Can be bought?	£	£	£	£	£	£	£
1	Go		Collect £200	No							
2	Crapper Street	Brown		Yes	60	2	10	30	90	160	250
3	Pot Luck		Take card	No							
4	Gangsters Paradise	Brown		Yes	60	4	20	60	180	320	450
5	Income Tax		Pay £200	No							
6	Brighton Station	Station		Yes	200	See notes					
7	Weeping Angel	Blue		Yes	100	6	30	90	270	400	550
8	Opportunity Knocks		Take card	No							
9	Potts Avenue	Blue		Yes	100	6	30	90	270	400	550
10	Nardole Drive	Blue		Yes	120	8	40	100	300	450	600
11	Jail/Just visiting			No							
12	Skywalker Drive	Purple		Yes	140	10	50	150	450	625	750
13	Tesla Power Co	Utilities		Yes	150	See notes					
14	Wookie Hole	Purple		Yes	140	10	50	150	450	625	750
15	Rey Lane	Purple		Yes	160	12	60	180	500	700	900
16	Hove Station	Station		Yes	200	See notes					
17	Cooper Drive	Orange		Yes	180	14	70	200	550	750	950
18	Pot Luck		Take card	No							
19	Wolowitz Street	Orange		Yes	180	14	70	200	550	750	950
20	Penny Lane	Orange		Yes	200	16	80	220	600	800	1000
21	Free Parking		Collect fines	No							

Figure 5.8: Board data in Excel format

```
1,Go,,,Collect £200,No,,,,,,,,
2,Crapper Street,,Brown,,Yes,,60,2,,10,30,90,160,250
3,Pot Luck,,,Take card,No,,,,,,,,
4,Gangsters Paradise,,Brown,,Yes,,60,4,,20,60,180,320,450
5,Income Tax,,,Pay £200,No,,,,,,,,
6,Brighton Station,,Station,,Yes,,200,See notes,,,,,,
7,Weeping Angel,,Blue,,Yes,,100,6,,30,90,270,400,550
8,Opportunity Knocks,,,Take card,No,,,,,,,,
9,Potts Avenue,,Blue,,Yes,,100,6,,30,90,270,400,550
10,Nardole Drive,,Blue,,Yes,,120,8,,40,100,300,450,600
11,Jail/Just visiting,,,,No,,,,,,,,
12,Skywalker Drive,,Purple,,Yes,,140,10,,50,150,450,625,750
13,Tesla Power Co,,Utilities,,Yes,,150,See notes,,,,,,
14,Wookie Hole,,Purple,,Yes,,140,10,,50,150,450,625,750
15,Rey Lane,,Purple,,Yes,,160,12,,60,180,500,700,900
16,Hove Station,,Station,,Yes,,200,See notes,,,,,,
```

Figure 5.9: Board data in CSV format

Each value in the excel sheet is separated by commas in the CSV format. If a cell is empty then the character in between two commas is empty. Every value in each line in the CSV format can be obtained by using regex, split(",") [13]. This separates the values from commas and stores the result in an array, where each index is the value in order of appearance. The meaning of a value is determined by cross-referencing with the Excel file. For example, the first value is always an integer and it represents the position of the square.

The only difference between converting the board and card data is the final step of creating their respective game logic, "Square" and "PackOfCards"; loading the CSV data is almost identical.

On start-up, in order to be greeted with the menu, we changed the "Main.java" method to instead load "Menu.fxml". This file has its own controller class, "MenuController.java". Instead of creating a new controller class, we could have added all the functionality of the menu into our "Controller.java" class as this would not break the "View-Controller-Model" structure. However, lessons from Sprint 1 taught us that more abstraction makes it easier to maintain and debug the code base. Therefore, we implemented a further "View-Controller" structure where every view will have its own controller class.

5.11 Testing

5.11.1 Unit Level

Tests for: F9
Every player is assigned a number that uniquely identifies them from other players. Hence, we must check that every player has a unique ID.

```
@Test
public void uniquePlayerId() {
    ArrayList<Integer> unique = new ArrayList<~>();
    boolean allUnique = true;
    for (Player p : allPlayers) {
        if (!unique.contains(p.getId())) {
            unique.add(p.getId());
        } else {
            allUnique = false;
        }
    }
    assertTrue(allUnique);
}
```

Figure 5.10: Unit Test: Unique Player ID

Tests for: T3
A player ending their turn requires the board to set the next player to take their turn. Test whether the board correctly updates the next player. There is an edge case when current player is the last player to take their turn, i.e. last element of the array list, in which case the pointer must loop around to the beginning of the array list.

```
@Test
public void nextPlayer(){
    assertEquals(board.getCurrentPlayerPointer(), actual: 0);
    board.setNextPlayer();
    assertEquals(board.getCurrentPlayerPointer(), actual: 1);
    board.setNextPlayer();
    assertEquals(board.getCurrentPlayerPointer(), actual: 2);
    board.setNextPlayer();
    assertEquals(board.getCurrentPlayerPointer(), actual: 3);
    board.setNextPlayer();
    assertEquals(board.getCurrentPlayerPointer(), actual: 4);
    board.setNextPlayer();
    assertEquals(board.getCurrentPlayerPointer(), actual: 5);
    board.setNextPlayer();
    assertEquals(board.getCurrentPlayerPointer(), actual: 0);
}
```

Figure 5.11: Unit Test: Set Next Player

Check if a card object behaves as expected, correctly storing the given description and action.

```
@Test
public void descriptionAndAction(){
    Card c = new Card( description: "hello",  action: "i like cats");
    assertEquals( expected: "hello", c.getDescription());
    assertEquals( expected: "i like cats", c.getAction());
}
```

Figure 5.12: Unit Test: Description and Action of a Card

Tests for: F5
Whenever a card is picked from a pack, it must return the top card of the pack.

```
public void topOfPack() {
    Card top = pack.getPack().get(0);
    Card testTop = pack.pickCard();

    assertEquals(top, testTop);
}
```

Figure 5.13: Unit Test: Top of Card Pack

Tests for: F5
Check whether a pack of cards has been shuffled. Note that there is a very small chance that the test fails if, after the shuffle, all cards remain in the same place in the array list.

```
@Test
public void shuffle() {
    PackOfCards original = pack;
    pack.shuffleDeck();

    assertNotEquals(pack.getPack(), original);
}
```

Figure 5.14: Unit Test: Shuffle Pack of Cards

Tests for: F5
Whenever a card is picked, it must return the card at the top of the pack. This card must then be replaced at the bottom of the pack. After picking a card, check that the card at the top of the pack does not equal the card that was just picked. Now, the card at the top of the pack must be that card that was second in the pack before picking a card. Lastly, the card now at the bottom of the pack must be the card that was picked.

```java
@Test
public void pickCard(){
    pack.shuffleDeck();
    Card first = pack.getPack().get(0);
    Card second = pack.getPack().get(1);
    Card testFirst = pack.pickCard();

    assertEquals(first,testFirst);
    assertNotEquals(first,pack.getPack().get(0));

    Card bottomCard = pack.getPack().get(pack.getPack().size()-1);
    assertEquals(first,bottomCard);
    assertEquals(pack.getPack().get(0),second);
}
```

Figure 5.15: Unit Test: Pick Card from Pack

Tests for: T2
The size of Pot Luck and Opportunity Knocks card packs, based on the given initial card data, are 16 each.

```java
@Test
public void sizeOfPacks() {
    ArrayList<Card> potLuckPack = potLuck.getPack();
    ArrayList<Card> oppurtunityKnocksPack = oppurtunityKnocks.getPack();

    assertEquals(potLuckPack.size(), actual: 16);
    assertEquals(oppurtunityKnocksPack.size(), actual: 16);
}
```

Figure 5.16: Unit Test: Size of Packs

Tests for: T2
Both tests check whether, given the initial card data file, the card data has correctly been converted into packs of cards.

```
@Test
public void potLuck() {
    Card first = potLuck.getPack().get(0);
    assertEquals( expected: "You inherit £100", first.getDescription());
    assertEquals( expected: "Bank pays player £100", first.getAction());
```

Figure 5.17: Unit Test: Pack of Pot Luck Cards

```
@Test
public void opportunityKnocks() {
    Card first = oppurtunityKnocks.getPack().get(0);
    assertEquals( expected: "Bank pays you divided of £50", first.getDescription());
    assertEquals( expected: "Bank pays player £50", first.getAction());
```

Figure 5.18: Unit Test: Pack of Opportunity Knocks Cards

Tests for: T2
The BoardCSVConverter converts data into Square objects, each square represents a square on the board. There must be 40 squares in the board.

```
@Test
public void sizeOfBoard() {
    assertEquals(allSquares.size(),  actual: 40);
}
```

Figure 5.19: Unit Test: Size of Board

Tests for: T2
The test below checks, given the initial board data, if the data has correctly been converted into Square objects.

```
@Test
public void squares() {
    assertEquals(allSquares.get(0).toString(),  actual: "Position: 1, Name: Go, Group: Go, Action: Collect £200, Buyable: No");
```

Figure 5.20: Unit Test: Squares on the Board

5.11.2 System Level

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
1	User starts the game.	Running main method in "main" class.	User should be greeted with a window called "Watson Games". There should be a single button called "Create Game"	A window containing a single button called "Create Game".	PASS
2	User clicks "Create Game" button.	A button click event on "Create Game".	There should be two dice. There should be two button called "Roll Dice" and "End Turn". The current player label should identify "Player 1".	Images of two dice visible. "Roll Dice" and "End Turn" button present. Current player is "Player 1".	PASS
3	User clicks "Create Game" button.	A button click event on "Create Game".	Board should correctly name each square. The background of each square should be correct depending on the group of the square.	Board view correctly names each square. Background images correctly displayed only for squares with a group of a colour. The background images of squares not belonging to a colour is empty.	FAIL
4	User clicks "Create Game" button.	A button click event on "Create Game".	Based on the number of players, the correct number balances of each player should be displayed. Each player should have £1500, and their ID's should be unique.	The correct number of labels displayed, with the ID of players' being unique and having £1500 in the bank.	PASS
5	User clicks "Create Game" button.	A button click event on "Create Game".	The "Roll Dice" button should be clickable. The "End Turn" button not be clickable.	"Roll Dice" button clickable. "End Turn" button not clickable.	PASS
6	User clicks "Create Game" button.	A button click event on "Create Game".	Based on the number of players, the correct number of player tokens should be displayed. Each token should be unique.	Correct number of tokens displayed, each of them unique.	PASS
7	User presses "Roll Dice" button.	A button click event on "Roll Dice".	"Roll Dice" button should not be clickable. The "End Turn" button should be clickable.	"Roll Dice" button is not clickable. "End Turn" button is clickable.	PASS
8	User presses "Roll Dice" button.	A button click event on "Roll Dice".	The current player's token should move clockwise around the board by the sum of the two dice.	Moves the correct player token clockwise around the board by the sum of the two dice.	PASS
9	User presses "End Turn" button	A button click event on "End Turn".	Update the current player label to the next player to take their turn.	Correctly updates the next player to take their turn.	PASS
10	User presses "End Turn" button.	A button click event on "End Turn".	"End Turn" button should not be clickable. The "Roll Dice" button should be clickable.	"End Turn" button is not clickable. "Roll Dice" button is clickable.	PASS

Test Number	Testing For
1	T7, F6
2	T3, F6, NF2
3	F3, F6, NF3
4	F6, F7, NF2
5	T3, T7, F2
6	T1, F1, F6
7	T5, F2
9	F8
10	T5, F2

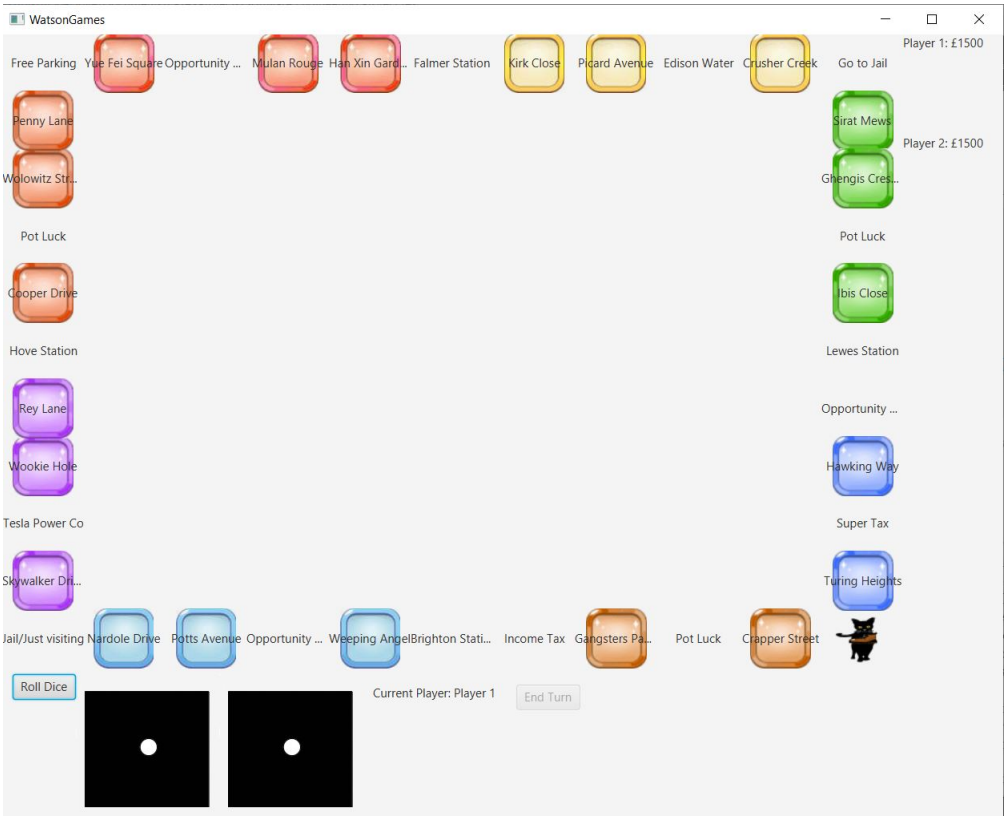


Figure 5.21: Board view of second prototype

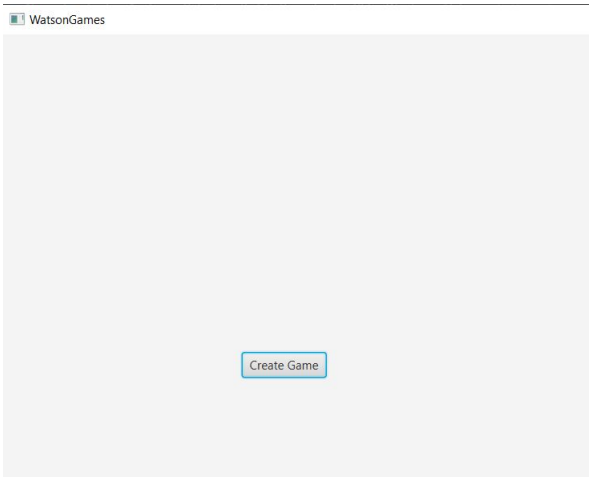


Figure 5.22: Menu after running "Main.java"

5.12 Reflection

The reason why system level test number 3 was not rectified this sprint is because it requires feedback from the customer. The background images of squares are generated based on the group the square belongs to. However, in the initial board data file provided, squares that do not belong to a colour group have empty entries under the group column. We believe that using names to assign appropriate images would not be sensible as the user should ideally be able to name the square to any value they choose. We will propose, to the customer, to let us populate the group column of the board data.

Overall, this sprint was a success as it accomplished all task/requirements bar one (system level test 3). This sprint flowed much smoother than Sprint 1 due to a more thorough requirement analysis.

Feedback from the customer was overall positive. The customer recommends adding the following functionalities and documentation during the next sprint:

- Complete missing Javadocs on classes
- Player receiving £200 when the pass go
- Extending the turn when a player rolls a double
- Make initializing squareCoordinates, allNames and allGroups more concise
- Explain the pointing scale on task cards

- Link system tests back to task cards

The customer has confirmed that we are allowed to populate the group column of the board data file. We will amend the problem of background images not being displayed for non-colour grouped squares next sprint.

The code for the first prototype can be found in the submission folder, called "SecondPrototype".

5.13 Individual Key Contributions

Test Member	Key Contribution(s)
Jun	T4, T5, T6
Jihye	T1, System Testing
Ye-Rang	T2, T8
Ibi	T2, T3, T7, Unit Testing

Sprint 3

Team Number: 32
Sprint Technical Lead: Ye-Rang Lee
Sprint start date: 5th March 2020
Sprint end date: 19th March 2020

6.1 User Stories

This sprint will produce the third prototype. Taking on feedback from sprint 2, the prototype will include the buying and selling of properties. The user should be able to improve these properties by buying houses and hotels, according to the rules given by the customer. When a player passes "GO", the player receives £200. If a player rolls a double, then their turn should be extended.

Furthermore, after rolling, if a player lands on a property that is owned by another player, they pay the rent of that property. There should also be the option of seeing the net worth of the player; this includes the current balance and worth of the properties the player owns. If a player is unable to pay the rent, then they should be removed from the game. In addition, there should be a method of viewing the current state of the board, which is to see who owns which properties. The current rent and worth of the properties should be visible.

The history of the game, i.e. important actions during a turn, should be recorded and visible. Though this is not a requirement, a game log will greatly help all players to keep track of their own, and others, actions.

Finally, the user should be greeted with a proper menu where they can choose the number of players playing the game and specify the board/card data files they want to play with. Again, a menu part of the requirements given by the user, but we believe that creating a menu will make it easy for

the user to customize the game.

6.2 Task Cards

Scoring scale: 1 to 10, with 10 having the highest priority and 1 the lowest. A task card with priority 10 means that it must be completed this sprint, otherwise the prototype will not work or make enough progress from the previous prototype. Alternatively, a 10 can also be a feature that the customer has recommended. A card with priority 1 means that it does not have to be completed this sprint, and without it, it will not impact the current prototype.

1. Priority: 10, Complexity: 2

   When a player passes "GO", £200 must be added to their balance. The balance labels must be updated to display their new balance.

2. Priority: 10, Complexity: 4

   When a player rolls a double, their turn must be extended.

3. Priority: 10, Complexity: 10

   Refactor squareCoordinates, allGroups and allNames to initialize the array lists using the root of the window.

4. Priority: 8, Complexity: 9

   There shall be a button called "Status" that shows the current properties that every player owns. Listed, the net worth of the player and current rent of the properties must be displayed.

5. Priority: 6, Complexity: 10

   When the user starts the application, they shall be greeted with an interactive menu. The user shall be able to choose the number of players, along with the board and card data they wish to play with. Once they have created the game, the board view shall be created with the respective information.

6. Priority: 4, Complexity: 5

   Create a game log that stores the history of important actions during the game.

7. Priority: 10, Complexity: 7

   When a player lands on a property that can be bought, they shall have the option to buy the property. All relevant information of the property, such as the: position, name, group, price, rent prices and improvement costs shall be visible.

8. Priority: 7, Complexity: 5

   If a player lands on a square that is owned by a different player, they shall pay the rent of the square. If they are unable to pay the rent, then remove them from the game.

9. Priority: 4, Complexity: 10

   There shall be a button called "Improve/Sell", that when pressed, lists all the properties that player owns. The player shall be able to buy houses/hotels to improve the properties they own.

10. Priority: 10, Complexity: 2

    Upon clicking "Create Game", the board must display the background images of all squares correctly.

11. Priority: 1, Complexity: 6

    The text fields in the menu must be responsive, meaning, the text field shall increase in width should a long text be entered.

6.3 Requirements Analysis

6.3.1 Functional Requirements

F1 - Mandatory

When a player pass "GO", £200 shall be added to their balance. Refactor the newPosition method of Player.java to update the player's balance every time they pass the end of the board.

F2 - Mandatory

In Controller.java, add a check in the rolledDice method to see whether the player has rolled a double.

F3 - Mandatory

The net worth of a player is calculated by summing the player's balance and worth of their properties. The worth of a property is calculated by

summing the cost of all houses/hotels on the property, along with the cost of buying the property.

F4 - Desirable

When a player clicks "Status", a new window should open on top of the board view. In this new window, for every player remaining in the game, there should be a label containing the player's ID, balance and net worth. Next to this label, there should be a list of all the properties the player owns. The name, group, total worth and current rent of the property shall be displayed.

F5 - Mandatory

A player can buy a property only if they have enough money in their balance. Ignore the net worth of the player.

F6 - Mandatory

When a player lands on a property that can be bought, a new window shall open. No other actions, such as selling houses, may be allowed whilst this window is open. There shall be three buttons, namely: "Yes", "No" and "Confirm Choice". Clicking "Yes" signals that the player wants to buy the property, "No" otherwise. "Confirm Choice" shall buy, or not buy, the property. Pressing "Confirm Choice" shall close the window.

F7 - Desirable

If the player has successfully bought the property, then information including the player's ID, name and cost of said property should be added to the game log. If they are unsuccessful, then add to the game log a message signalling the failure, along with the player's ID and name of the property.

F8 - Desirable

Clicking "Buy/Sell" will open a new window. This window should have four distinct sections:

1. A list of all the properties the player owns. The player should be able to select the property they wish to improve/sell.
2. The cost of buying/selling houses, hotels or the property itself of the selected property.

3. The result of the proposed transaction. For example, was the player successful in buying a house? If not, then list the possible reasons of why. If they were, then let them know.
4. Two buttons called "Buy" and "Sell" that do their respective actions.

Here is the flow of action that should occur when a user interacts with the improve/sell window. First, select a property from the list of properties they own. Once a property has been chosen, the options of buying/selling houses, hotels and the property itself should automatically be updated. Second, select an option, such as buying a house. Finally, click either "Buy" or "Sell" to complete the transaction. Whether the transaction was a success or not should be displayed.

F9 - Mandatory

A player may buy houses/hotel on a property if and only if they own all the properties in that set.

F10 - Mandatory

Whenever a player wants to buy or sell houses/hotel on a property, there may never be a difference of more than 1 house in that set.

F11 - Mandatory

A player may sell a property if and only if there are no houses/hotel on properties in that set.

F12 - Mandatory

The maximum rank on a property is one hotel.

F13 - Mandatory

The only time a player is able to buy a property is if they land on that property. They shall only be able to purchase it during this turn.

F14 - Mandatory

If a player lands on a square that is owned by a different player, they pay the rent of the square. If they are unable to pay their rent by selling assets, they are delcared bankrupt by removing their token and balance labels from the board. All properties the player owned are put back onto the market. The bankrupt player pays the player due rent their total net worth.

F15 - Desirable

In the main menu, the default option for the number of players should be 2. There should be a default path to the board and card data files.

F16 - Desirable

When choosing the number of players in the main menu, the user should only be able to choose between 2-6 players.

6.3.2 Non-Functional Requirements

NF1 - Desirable

Starting the application, the user should be greeted with an interactive menu system. Use JavaFX TextField[14] to allow the user to enter values.

NF2 - Desirable

The game log should be a ListView[15] object. There should be a method to add a message to the game log.

NF3 - Mandatory

Clicking "Status" will show the list of all properties a player owns. This list shall be a ListView[15] object.

NF4 - Desirable

There user should select the number of players from a ChoiceBox[17] object containing values from 2-6.

6.3.3 Domain Requirements

D1 - Mandatory

The game must be easy to use as Watson Games create "board games designed to appeal to a wide range of ages". Hence,

- Viewing properties of all players must be clear and easy to read. See NF3.
- Buying and selling houses, hotels and properties must be intuitive. See F7.
- The actions of players' shall be displayed throught the course of the game. See T6 and NF2.

6.4 Use Case Diagram

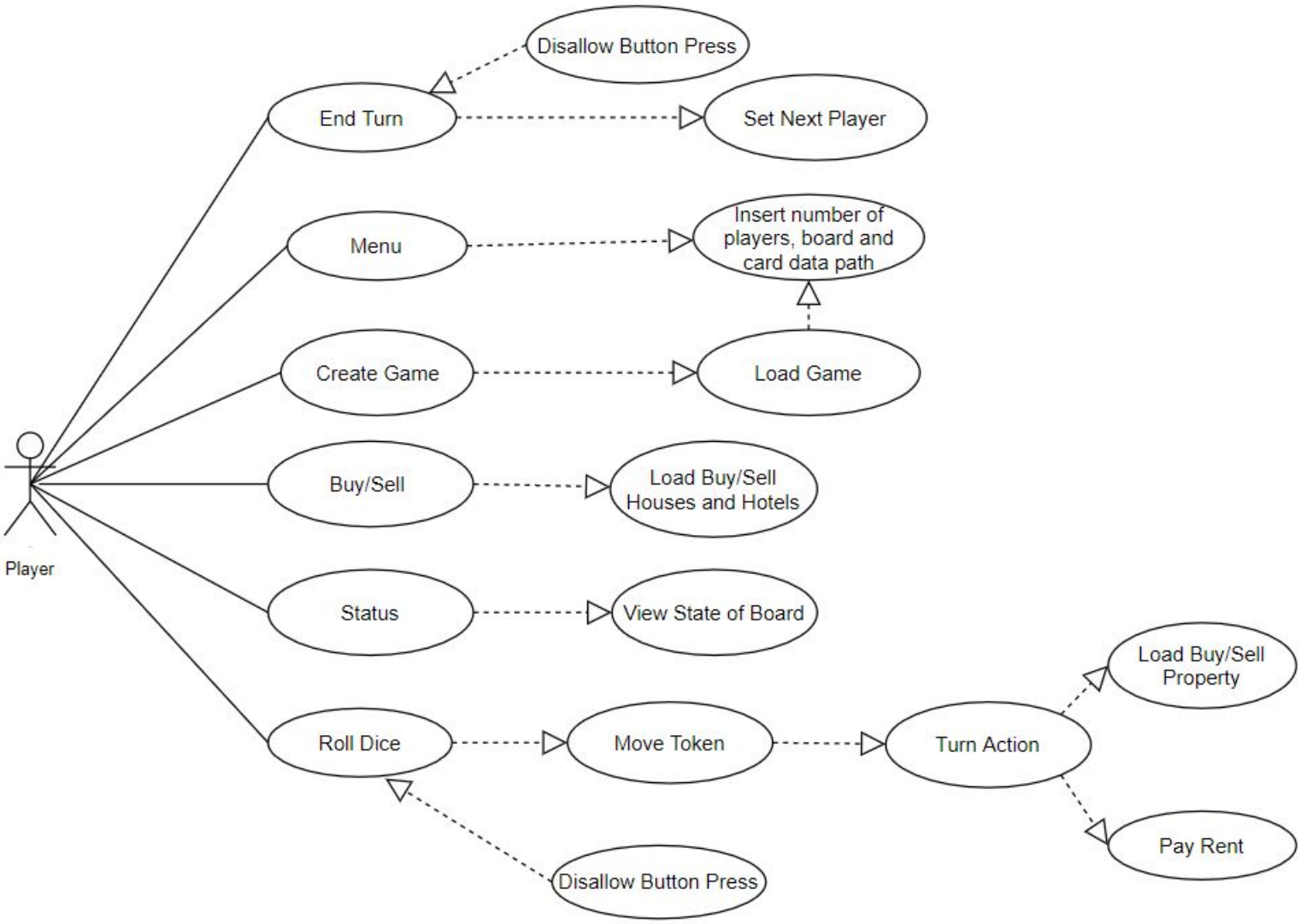


Figure 6.1: Third Prototype: Use Case Diagram

6.5 High Level Diagram

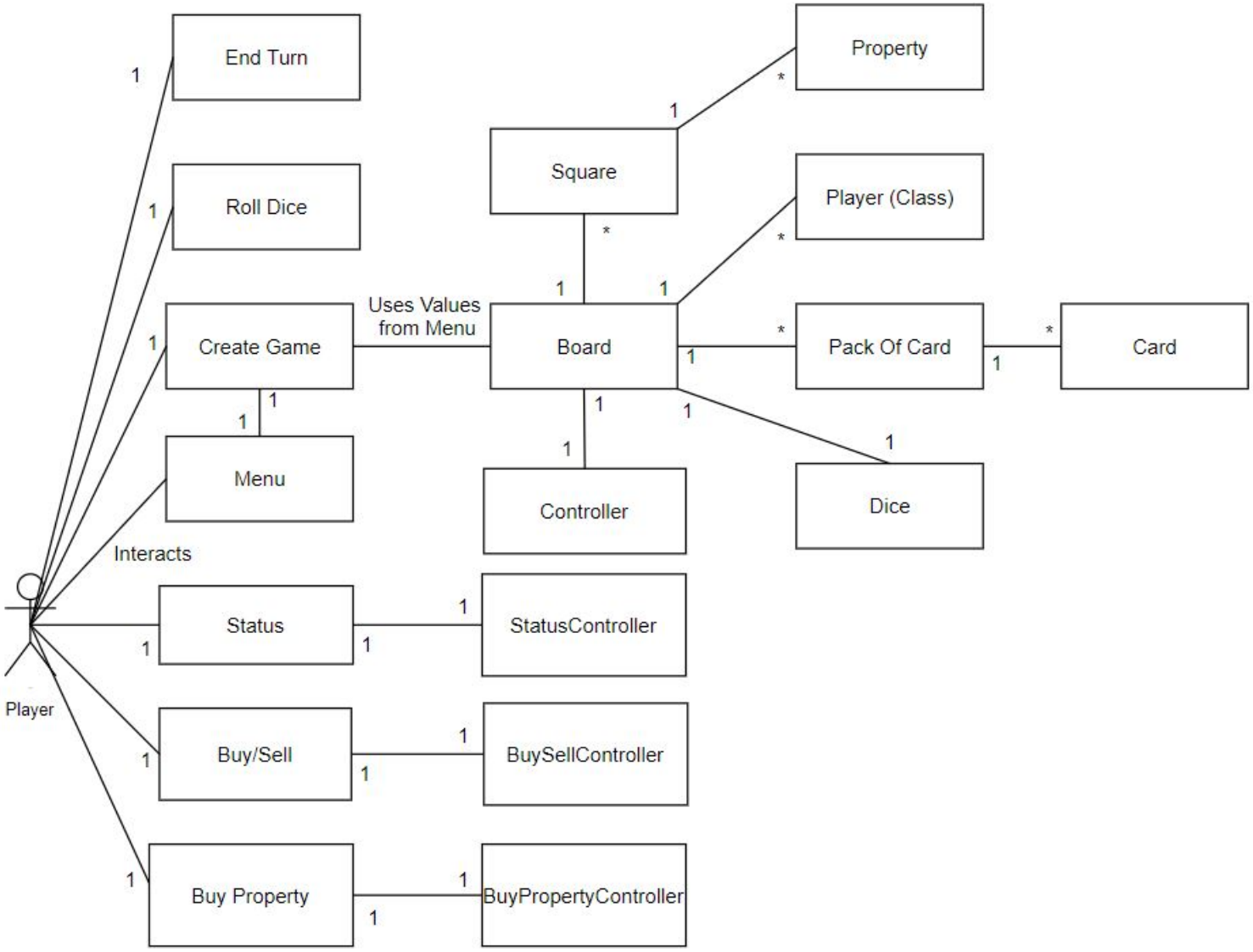


Figure 6.2: Third Prototype: High Level Diagram

6.6 Class Diagram

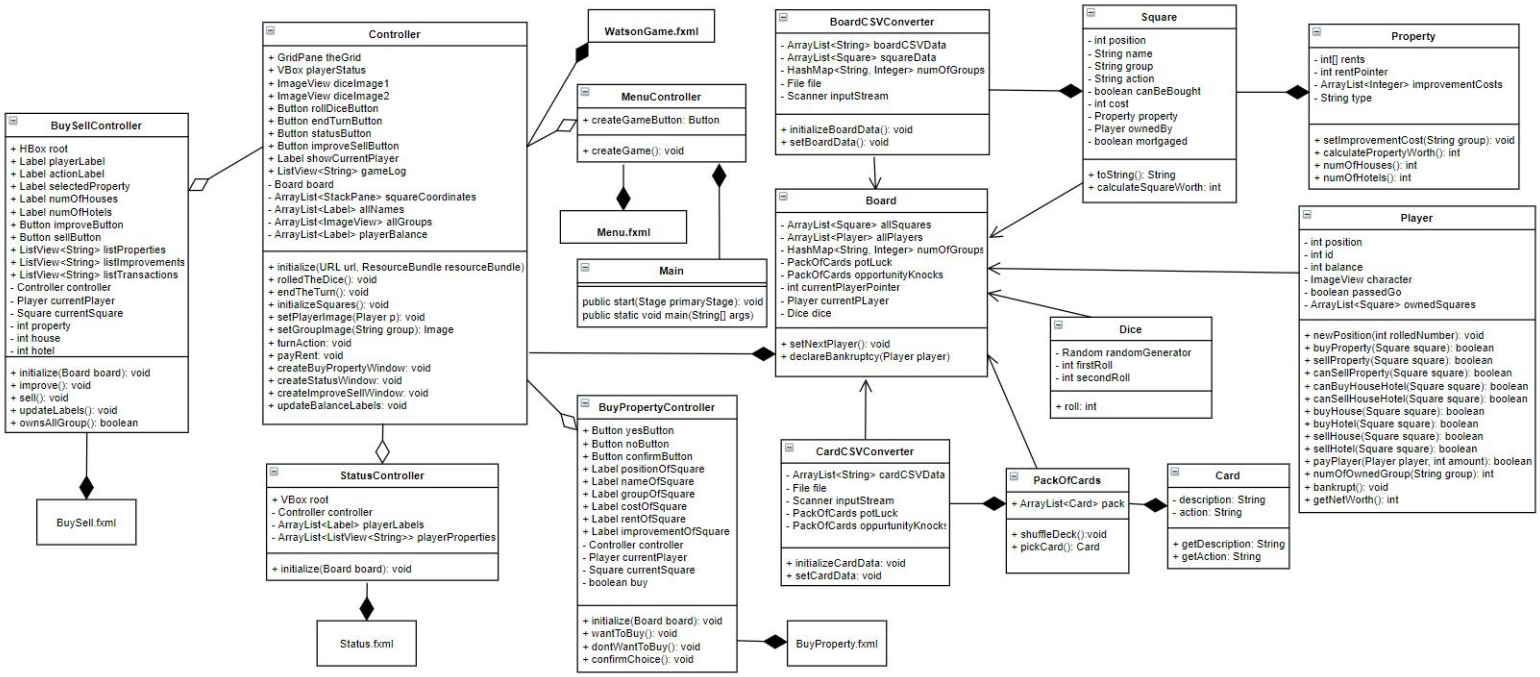


Figure 6.3: Class Diagram

In order to keep the class diagram concise and presentable, getter and setter methods have been omitted with permission from the customer. Note that they are included in the code and used in sequence diagrams even though they do may not appear in the class diagram.

The class diagram can be viewed clearly in Draw.io. To open our class diagrams, please read chapter 1. The first class diagram is located in the "Class Diagrams" folder, called "ThirdClassDiagram.io".

6.7 Sequence Diagrams

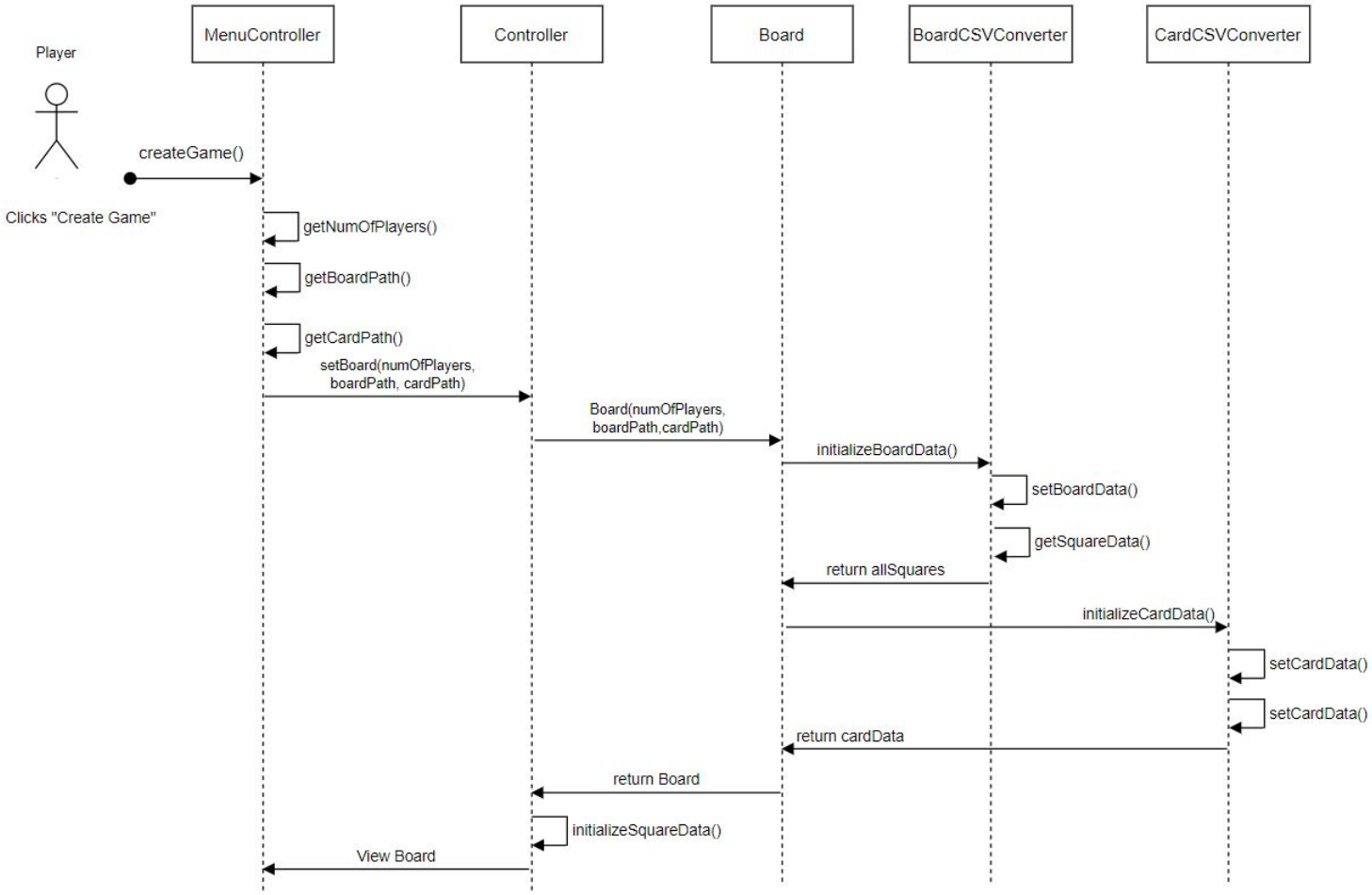


Figure 6.4: Sequence of actions when "Create Game" button is clicked

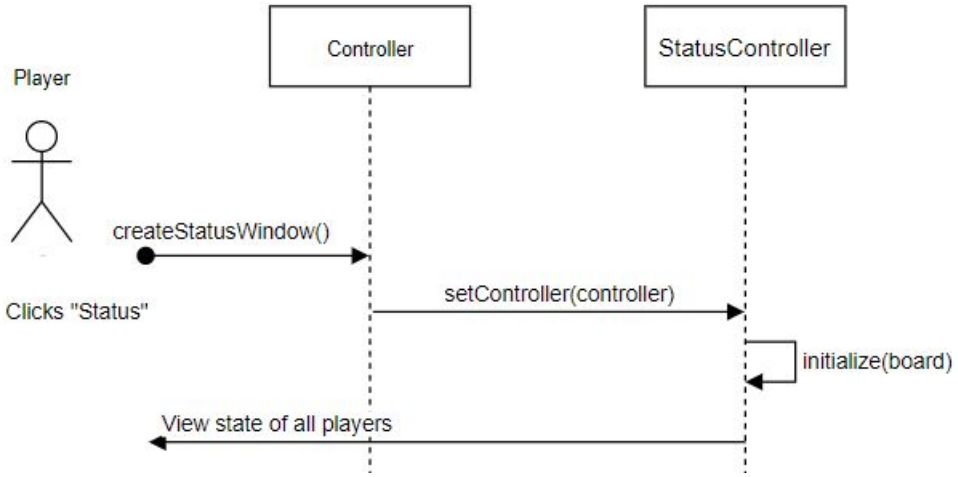


Figure 6.5: Sequence of actions when ”Status” button is clicked

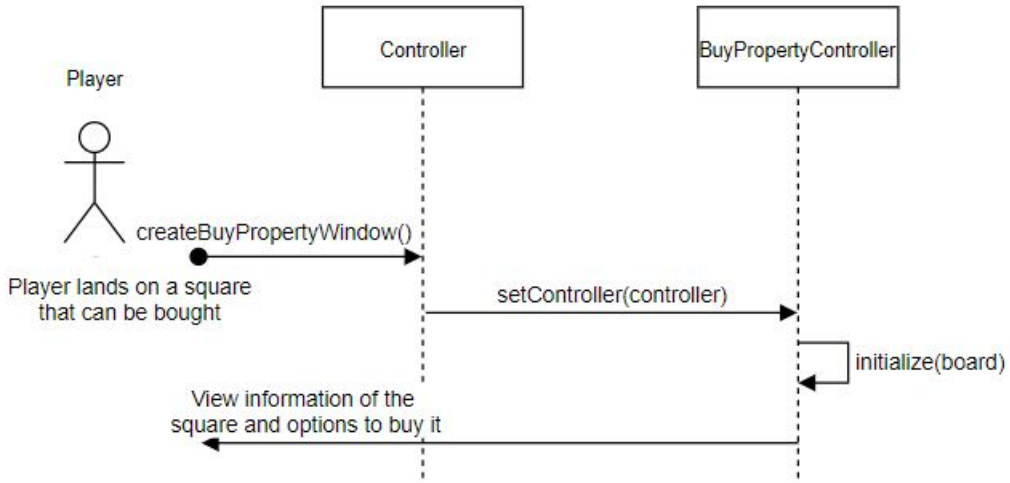


Figure 6.6: Sequence of actions when a player lands on a square that can be bought

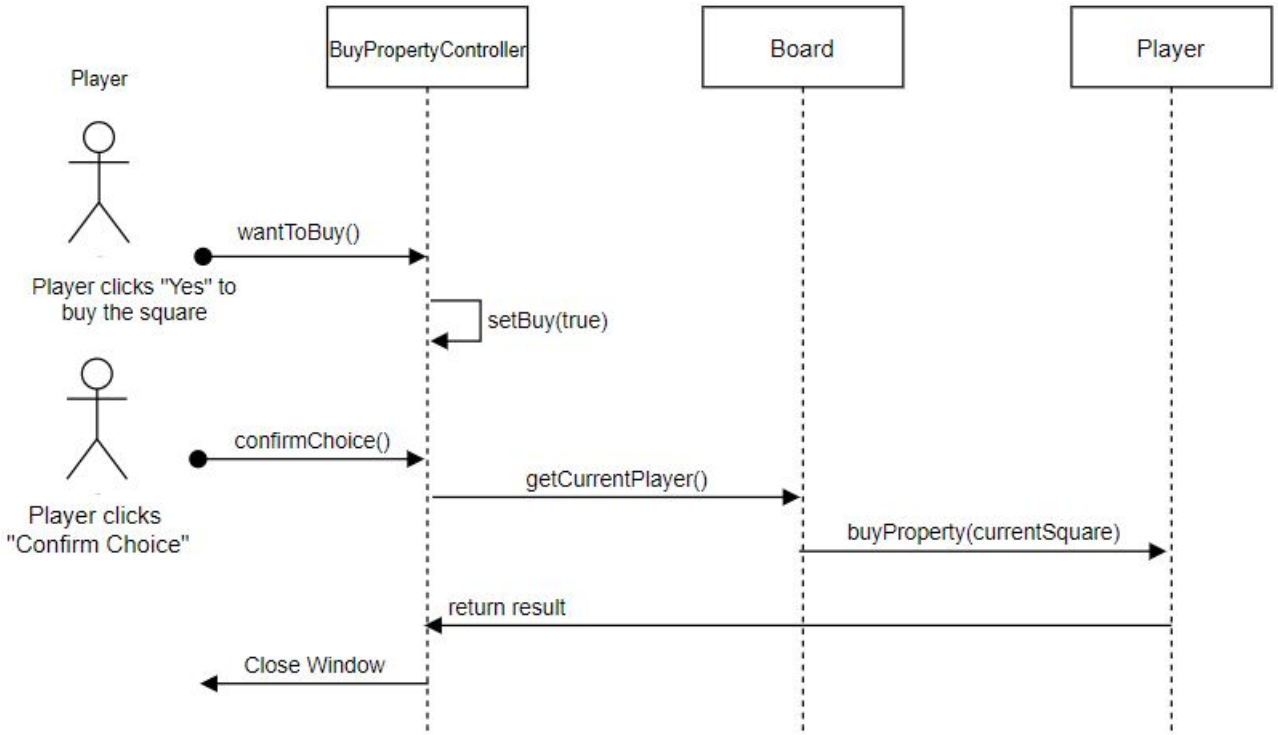


Figure 6.7: Sequence of actions when a player want to buy a square

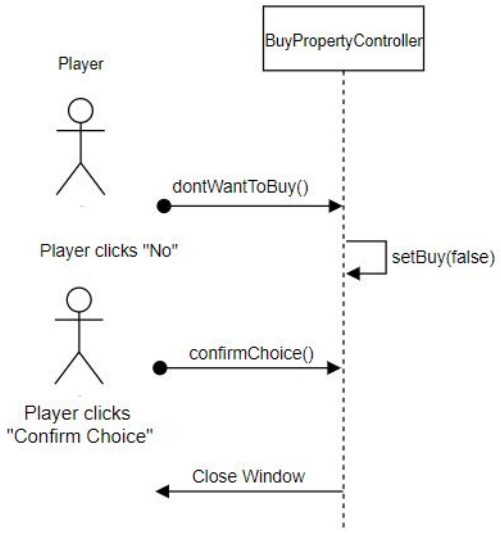


Figure 6.8: Sequence of actions when a player does not want to buy a square

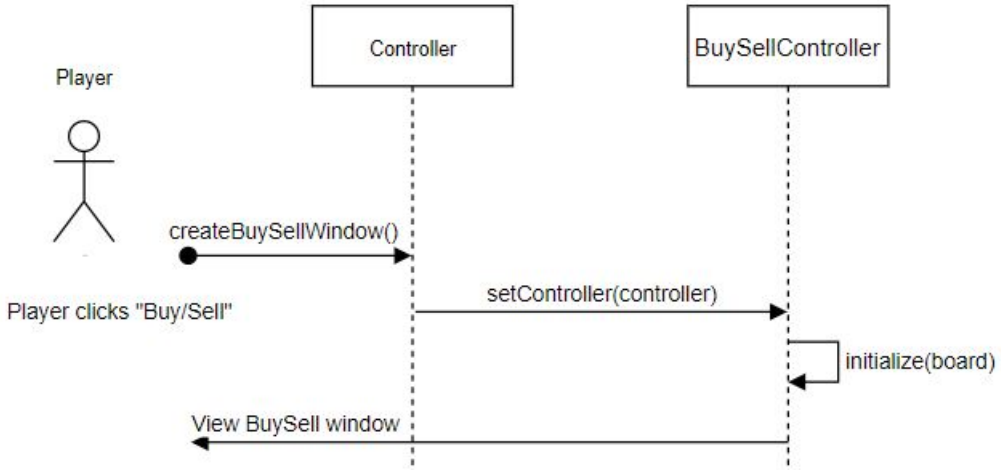


Figure 6.9: Sequence of actions when a player clicks "Buy/Sell"

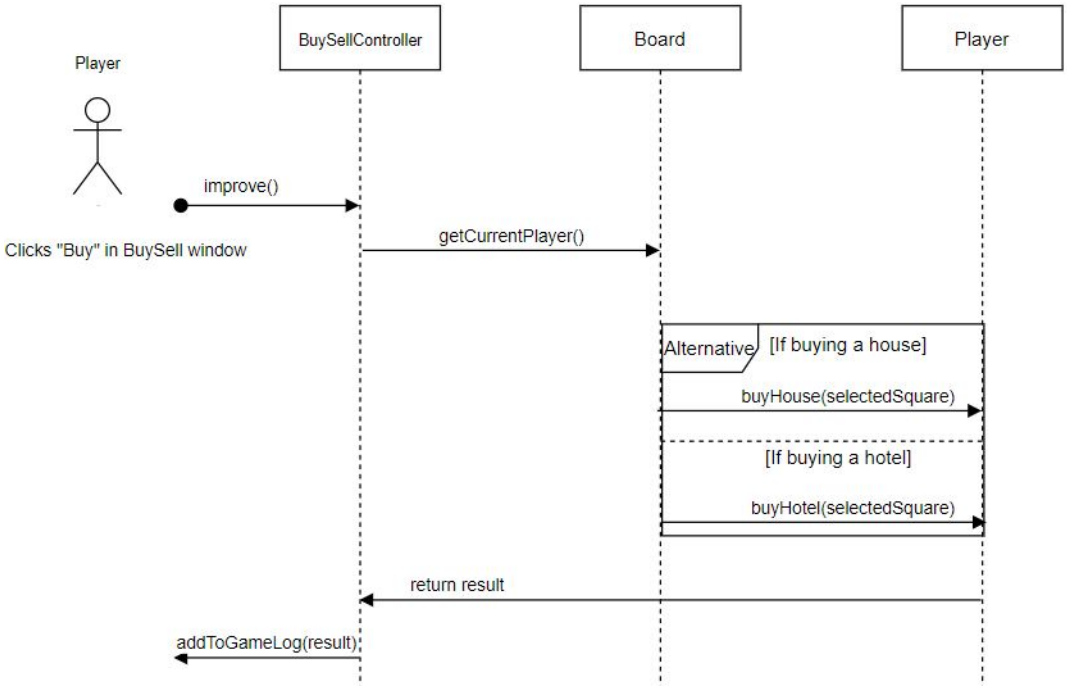


Figure 6.10: Sequence of actions when a player wants to buy a house/hotel

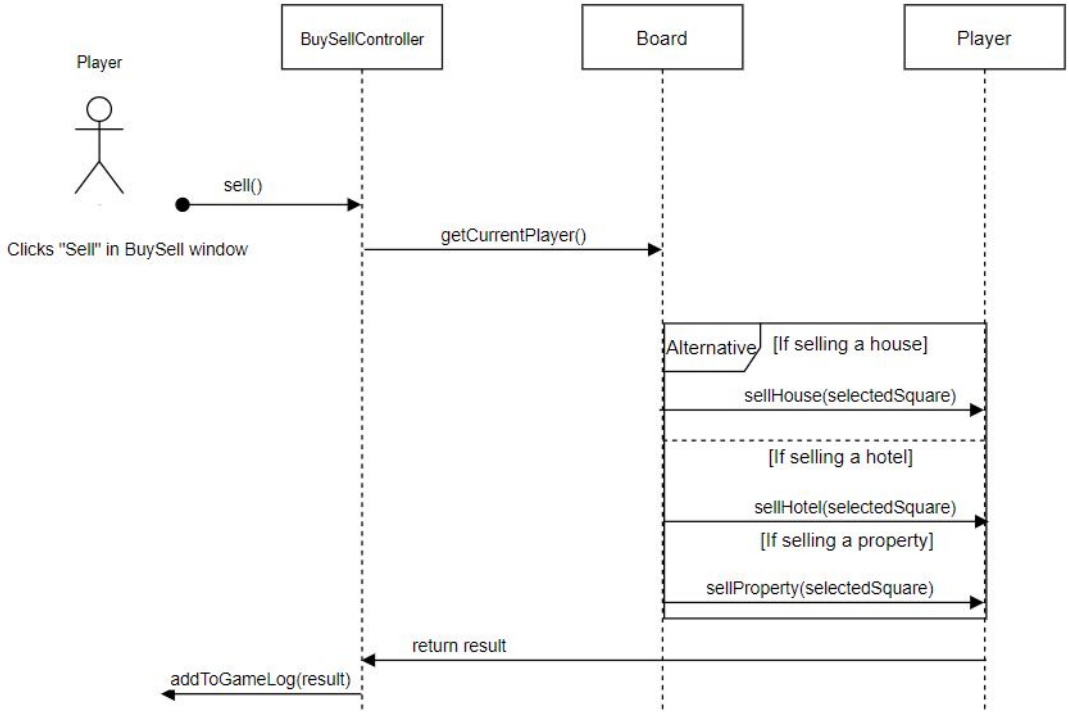
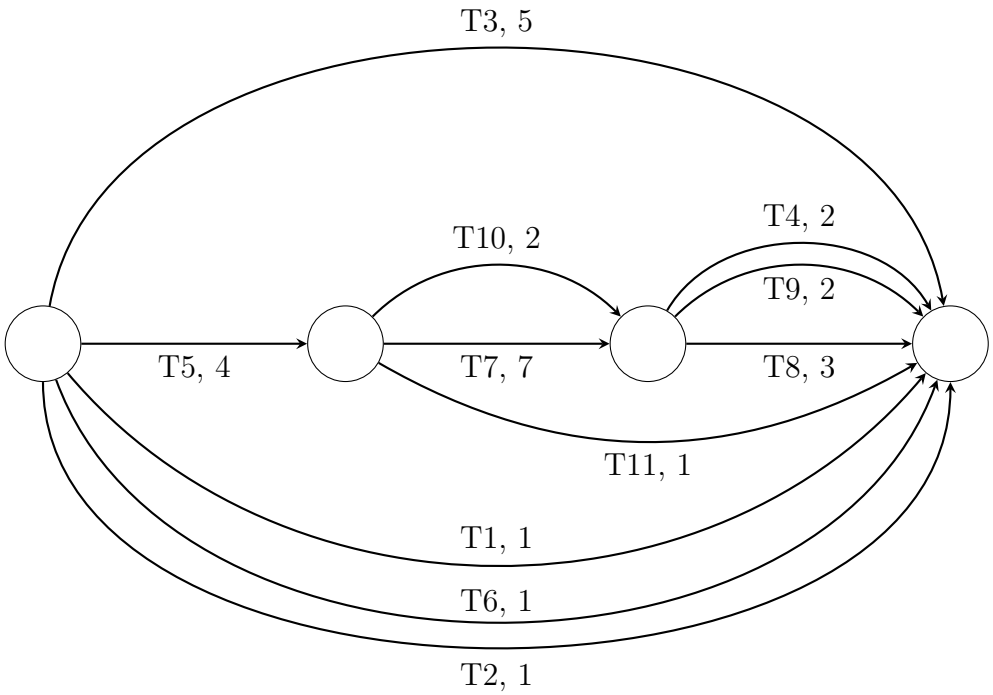


Figure 6.11: Sequence of actions when a player wants to sell a house/hotel/property

6.8 PERT Chart



Task Card	Days				Critical Path
	Earliest Start	Earliest Finish	Latest Start	Latest Finish	
1	0	1	13	14	No
2	0	1	13	14	No
3	0	5	9	14	No
4	11	13	12	14	No
5	0	4	0	4	Yes
6	0	1	13	14	No
7	4	11	4	11	Yes
8	11	14	11	14	Yes
9	11	13	12	14	No
10	4	6	9	11	No
11	4	5	13	14	No

Figure 6.12: Third Prototype: PERT Chart

Critical Path: T5, 57, T8

6.9 Risk Management

Risk Identification	Causes and Likelihood	Mitigation	Monitoring
We will be writing code in Java. Some members may not be comfortable on Java and prefer a different programming language.	Likelihood: 2 Some members may be used to writing in a different programming language.	People who are working on the code base have a greater say on the programming language. If these people are not comfortable working with Java, then we have agreed to switch to Python.	Discuss how the code is going during every team meeting. Technical Director should oversee how Technical Support members are doing.
We will be using JavaFX to create GUI elements of the game. Some members may not be comfortable with using JavaFX.	Likelihood: 2 Members may not have any experience using JavaFX. Some members prefer using Swing over JavaFX.	If not enough progress is being made with the code, swap to using Swing.	Review progress during every team meeting. Technical Supports should update the Technical Director about their progress with using JavaFX.
Software Development may be delayed.	Likelihood: 1 Deadlines for other assessments coming up. As a result, focus of members may be elsewhere near these assessments.	Plan work to take into account future deadlines. Project Manager organises tasks with deadlines in mind.	Project manager discusses future tasks and assigns tasks to each member during team meetings.
Members may have trouble completing a coding task, which can slow progress.	Likelihood: 2 Technical Supports having difficulty with completing a task.	Technical Support should notify the Technical Director for assistance.	Technical Supports will ask for support, if needed, from the Technical Director, during team meetings.

Likelihood scores are between 1-5, with 1 being highly unlikely and 5 being highly likely.

6.10 Implementation of Task Cards

Referencing JavaFX GUI components in the second prototype, having 40 stackpanes and labels, was inefficient and impractical. The customer recommended us to find a more concise way.

Our solution involves naming only the root of the window. From this root, loop through all of its children. If a children is a StackPane, then add it to squareCoorindates. Then loop through the children of every StackPane in squareCoorindates. If the child is a Label, add it to allNames, if the child is an ImageView, add it to allGroups.

```
for (Node n : theGrid.getChildren()) {
    if (n instanceof StackPane) {
        squareCoordinates.add((StackPane) n);
    }
}

for (StackPane s : squareCoordinates) {
    for (Node n : s.getChildren()) {
        if (n instanceof Label) {
            allNames.add((Label) n);
        }
        if (n instanceof ImageView) {
            allGroups.add((ImageView) n);
        }
    }
}
```

Figure 6.13: Refactoring Controller.java

The turnAction method, called only after rolling the dice, in Controller.java determines the events that occur during the turn based on the position the current player is on. If they land on a property that can be bought then a new window is opened. This window lets the player choose whether they want to buy said property.

If the player lands on a property that is owned by a different player, then payRent. The rent will automatically be paid if they have enough money in their balance to do so. When there is not enough money in but the player has enough to pay the rent, the player is instructed to sell assets and pay the player who owns the square. The player unable to pay the rent is removed from the game by allowing all properties they own back onto the market. Their token and balance label are removed from the board. Lastly, the player going bankrupt pays the owed player their total net worth.

```java
public void turnAction() {
    Player currentPlayer = board.getCurrentPLayer();
    Square currentSquare = board.getAllSquares().get(currentPlayer.getPosition());
    if (currentSquare.getCanBeBought()) {
        createBuyPropertyWindow();
    } else if (!Objects.isNull(currentSquare.getOwnedBy()) && !currentSquare.getOwnedBy().equals(currentPlayer)) {
        payRent();
    }
    updateBalanceLabels();
}
```

Figure 6.14: turnAction method in Controller.java

A new window is opened by loading the chosen .fxml file. We can obtain the controller to this file and use the methods of that controller. To allow communication between windows, set the controller field of the new window to the current controller.

```java
public void createBuyPropertyWindow() {
    FXMLLoader loader = new FXMLLoader();
    loader.setLocation(getClass().getResource( name: "BuyProperty.fxml"));
    Parent buyPropertyView = null;
    try {
        buyPropertyView = loader.load();
    } catch (IOException e) {
        e.printStackTrace();
    }
    BuyPropertyController bpc = loader.getController();
    bpc.setController(this);
    bpc.initialize(this.board);
```

Figure 6.15: Communicating between different windows

6.11 Testing

6.11.1 Unit Level

Tests for: T8, F14
Player one buys both properties belonging to the brown group. After purchase, check that both properties are not buyable and are in the list of owned

squares of player one. Test whether the balance of the player is correct by subtracting the costs of buying those properties.

Player one needs to buy atleast one house so we can test resetting a square after a player is removed from the game. After buying a house, check that is balance is correct by further subtracting the cost of buying a house.

When the player is declared bankrupt, the number of players should decrease by one. Check that player one is not in the list of all players. Test that both properties are back on the market, ready to be purchased. Finally, on the property that had a house, test that it no longer has any improvements on it.

```
@Test
public void declareBankruptcy() {
    Player one = board.getAllPlayers().get(0);
    Square crapperStreet = board.getAllSquares().get(1);
    Square gangstersParadise = board.getAllSquares().get(3);
    one.buyProperty(crapperStreet);
    one.buyProperty(gangstersParadise);

    assertFalse(crapperStreet.getCanBeBought());
    assertFalse(gangstersParadise.getCanBeBought());
    assertEquals(one.getOwnedSquares().size(), actual: 2);
    assertTrue(one.getOwnedSquares().contains(crapperStreet));
    assertTrue(one.getOwnedSquares().contains(gangstersParadise));
    assertEquals(one.getBalance(), actual: 1500 - crapperStreet.getCost() - gangstersParadise.getCost());

    one.buyHouse(crapperStreet);

    assertEquals(crapperStreet.getProperty().getRentPointer(), actual: 1);
    assertEquals(one.getBalance(), actual: 1500 - crapperStreet.getCost()
            - gangstersParadise.getCost() - crapperStreet.getProperty().getImprovementCosts().get(housePointer));

    board.declareBankruptcy(one);

    assertEquals(board.getAllPlayers().size(), actual: 5);
    assertFalse(board.getAllPlayers().contains(one));
    assertTrue(crapperStreet.getCanBeBought());
    assertTrue(gangstersParadise.getCanBeBought());
    assertEquals(crapperStreet.getProperty().getRentPointer(), actual: 0);
}
```

Figure 6.16: Unit Test: Declare Bankruptcy

Tests for: F9
The BoardCSVConverter has an added functionality that keeps track of the number of times each group has been seen, stored in the HashMap called

numOfGroups. This is needed for determining whether a player owns all properties of a group so they can buy houses and hotels on the square. Using the modified initial data file, with the inserted values under the group column, check if the BoardCSVConverter has counted the number of occurrences of each group correctly.

```
    public void numOfGroups() {
        assertEquals(numOfGroups.get("red").intValue(), actual: 3);
        assertEquals(numOfGroups.get("blue").intValue(), actual: 3);
        assertEquals(numOfGroups.get("orange").intValue(), actual: 3);
        assertEquals(numOfGroups.get("purple").intValue(), actual: 3);
        assertEquals(numOfGroups.get("yellow").intValue(), actual: 3);
        assertEquals(numOfGroups.get("green").intValue(), actual: 3);
        assertEquals(numOfGroups.get("deep blue").intValue(), actual: 2);
        assertEquals(numOfGroups.get("brown").intValue(), actual: 2);
        assertEquals(numOfGroups.get("pot luck").intValue(), actual: 3);
        assertEquals(numOfGroups.get("opportunity knocks").intValue(), actual: 3);
        assertEquals(numOfGroups.get("station").intValue(), actual: 4);
        assertEquals(numOfGroups.get("utilities").intValue(), actual: 2);
        assertEquals(numOfGroups.get("income tax").intValue(), actual: 1);
        assertEquals(numOfGroups.get("super tax").intValue(), actual: 1);
        assertEquals(numOfGroups.get("jail").intValue(), actual: 1);
        assertEquals(numOfGroups.get("go to jail").intValue(), actual: 1);
        assertEquals(numOfGroups.get("free parking").intValue(), actual: 1);
        assertEquals(numOfGroups.get("go").intValue(), actual: 1);
    }
}
```

Figure 6.17: Unit Test: Number of Groups

Tests for: F3
The worth of a square is needed to calculate the total net worth of a player. With the net worth of the player, we can calculate if a player is able to pay rent or not.

Player one's balance is set to a high number so they are able buy properties, houses and hotels. Player one buys both properties belonging to the brown group. Player one then buys 4 houses on Crapper Street and Gangster's Paradise, with an additional hotel on Crapper Street. Test the worth of a square by summing the cost of buying the square, the cost of buying 4 and, only on Crapper Street, the cost of buying a hotel on the square.

```
    @Test
    public void calculateSquareWorth() {
        int housePointer = 0;
        int hotelPointer = 1;
        Player one = board.getAllPlayers().get(0);
        one.setBalance(1000000);
        Square crapperStreet = board.getAllSquares().get(1);
        Square gangstersParadise = board.getAllSquares().get(3);
        one.buyProperty(crapperStreet);
        one.buyProperty(gangstersParadise);

        assertEquals(crapperStreet.calculateSquareWorth(), crapperStreet.getCost());
        assertEquals(gangstersParadise.calculateSquareWorth(), gangstersParadise.getCost());

        for (int i = 0; i < 4; i++) {
            one.buyHouse(crapperStreet);
            one.buyHouse(gangstersParadise);
        }
        one.buyHotel(crapperStreet);

        assertEquals(crapperStreet.calculateSquareWorth(), actual: crapperStreet.getCost()
                + crapperStreet.getProperty().getImprovementCosts().get(housePointer) * 4
                + crapperStreet.getProperty().getImprovementCosts().get(hotelPointer));
        assertEquals(gangstersParadise.calculateSquareWorth(), actual: gangstersParadise.getCost()
                + gangstersParadise.getProperty().getImprovementCosts().get(housePointer) * 4);
    }
}
```

Figure 6.18: Unit Test: Worth of Square

Tests for: F10, F11, F12
Test whether numOfHouses and numOfHotels are returning the correct value. If the rent pointer of the property is 4, then there is 4 houses on the property. However, if the rent pointer is 5, then there is 4 houses along with 1 hotel on the property.

```java
@Test
public void numOfHousesAndHotels() {
    Player one = board.getAllPlayers().get(0);
    Square crapperStreet = board.getAllSquares().get(1);
    Square gangstersParadise = board.getAllSquares().get(3);
    one.buyProperty(crapperStreet);
    one.buyProperty(gangstersParadise);

    assertEquals(crapperStreet.getProperty().numOfHouses(), actual: 0);
    assertEquals(gangstersParadise.getProperty().numOfHotels(), actual: 0);

    for (int i = 0; i < 4; i++) {
        one.buyHouse(crapperStreet);
        one.buyHouse(gangstersParadise);
    }
    one.buyHotel(crapperStreet);

    assertEquals(crapperStreet.getProperty().numOfHouses(), actual: 4);
    assertEquals(crapperStreet.getProperty().numOfHotels(), actual: 1);
    assertEquals(gangstersParadise.getProperty().numOfHouses(), actual: 4);
    assertEquals(gangstersParadise.getProperty().numOfHotels(), actual: 0);
}
```

Figure 6.19: Unit Test: Number of Houses and Hotels

Tests for: F3
Buy 4 houses on both Crapper Street and Gangster's Paradise, and buy 1 hotel on Crapper Street. Check that calculatePropertyWorth returns the sum of the costs of buying the number houses and hotels on the property.

```java
@Test
public void calculatePropertyWorth() {
    Player one = board.getAllPlayers().get(0);
    Square crapperStreet = board.getAllSquares().get(1);
    Square gangstersParadise = board.getAllSquares().get(3);
    one.buyProperty(crapperStreet);
    one.buyProperty(gangstersParadise);
    for (int i = 0; i < 4; i++) {
        one.buyHouse(crapperStreet);
        one.buyHouse(gangstersParadise);
    }
    one.buyHotel(crapperStreet);

    assertEquals(crapperStreet.getProperty().calculatePropertyWorth(),
            actual: crapperStreet.getProperty().getImprovementCosts().get(housePointer) * 4,
           +crapperStreet.getProperty().getImprovementCosts().get(hotelPointer));
    assertEquals(gangstersParadise.getProperty().calculatePropertyWorth(),
            actual: gangstersParadise.getProperty().getImprovementCosts().get(housePointer) * 4);
}
```

Figure 6.20: Unit Test: Worth of Property

Tests for: F3
The net worth of a player is the sum of the player's balance and the worth of all their properties. The player's net worth is checked after buying a Skywalker Drive, after buying 4 houses where the net worth is checked after every house, and after buying a hotel.

```
@Test
public void getNetWorthTest() {
    Player one = board.getAllPlayers().get(0);
    Square skywalkerDrive = board.getAllSquares().get(11);
    one.buyProperty(skywalkerDrive);

    assertEquals(one.getNetWorth(), actual: one.getBalance() + skywalkerDrive.getCost());

    for (int i = 0; i < 4; i++) {
        one.buyHouse(skywalkerDrive);
        assertEquals(one.getNetWorth(), actual: one.getBalance() + skywalkerDrive.getCost()
                + skywalkerDrive.getProperty().calculatePropertyWorth());
    }
    one.buyHotel(skywalkerDrive);

    assertEquals(one.getNetWorth(), actual: one.getBalance() + skywalkerDrive.getCost()
            + skywalkerDrive.getProperty().calculatePropertyWorth());
}
```

Figure 6.21: Unit Test: Net Worth

Tests for: T8, F14
After a player goes bankrupt, all properties they used to own should now be buyable and so owned by no one. These properties should have all houses and hotels removed. Lastly, the player's character should be set to null.

```
@Test
public void bankrupt() {
    Player one = board.getAllPlayers().get(0);
    Square skywalkerDrive = board.getAllSquares().get(11);
    one.buyProperty(skywalkerDrive);
    one.buyHouse(skywalkerDrive);

    assertEquals(skywalkerDrive.getProperty().getRentPointer(), actual: 1);
    assertEquals(skywalkerDrive.getOwnedBy(), one);
    assertFalse(skywalkerDrive.getCanBeBought());
    assertNotNull(one.getCharacter());

    one.bankrupt();

    assertEquals(skywalkerDrive.getProperty().getRentPointer(), actual: 0);
    assertNull(skywalkerDrive.getOwnedBy());
    assertTrue(skywalkerDrive.getCanBeBought());
    assertNull(one.getCharacter());
}
```

Figure 6.22: Unit Test: Bankrupt Player

Tests for: F9
Test whether number of groups returns the correct number squares owned by the player of the specified group.

```
@Test
public void numOfOwnedGroup() {
    Player one = board.getAllPlayers().get(0);
    Square purple1 = board.getAllSquares().get(11);
    Square purple2 = board.getAllSquares().get(14);
    Square blue = board.getAllSquares().get(9);

    assertEquals(one.numOfOwnedGroup("purple"), actual: 0);
    assertEquals(one.numOfOwnedGroup("blue"), actual: 0);

    one.buyProperty(purple1);
    one.buyProperty(purple2);
    one.buyProperty(blue);

    assertEquals(one.numOfOwnedGroup("purple"), actual: 2);
    assertEquals(one.numOfOwnedGroup("blue"), actual: 1);

}
```

Figure 6.23: Unit Test: Number of Owned Groups

Tests for: F14

Tests whether a player can pay an sum of money to the chosen player. Check whether the players' balances have been updated correctly.

```
@Test
public void payPlayer() {
    Player one = board.getAllPlayers().get(0);
    Player two = board.getAllPlayers().get(1);

    assertTrue(one.payPlayer(two,  amount: 1000));
    assertEquals(one.getBalance(),  actual: 500);
    assertEquals(two.getBalance(),  actual: 2500);

    assertFalse(one.payPlayer(two,  amount: 1000));
    assertEquals(one.getBalance(),  actual: 500);
    assertEquals(two.getBalance(),  actual: 2500);
}
```

Figure 6.24: Unit Test: Pay Player

Tests for: F5
A player should only be buy a property if they have enough money in their balance. Once a property is bought, their balance should be updated. The bought square should now be owned by the player who bought it and there should no houses or hotels on the property. The square should not be buyable after the purchase.

```
@Test
public void buyProperty() {
    Player one = board.getAllPlayers().get(0);
    Square skywalkerDrive = board.getAllSquares().get(11);
    Square reyLane = board.getAllSquares().get(14);

    assertTrue(one.buyProperty(skywalkerDrive));
    assertFalse(skywalkerDrive.getCanBeBought());
    assertEquals(skywalkerDrive.getOwnedBy(), one);
    assertEquals(skywalkerDrive.getProperty().getRentPointer(),  actual: 0);
    assertEquals(one.getBalance(), actual: 1500-skywalkerDrive.getCost());

    one.setBalance(0);
    assertFalse(one.buyProperty(reyLane));
}
```

Figure 6.25: Unit Test: Buy Property

Tests for: F10

A player may only buy houses or hotels on a property if there would not be a difference greater than 1 house between all properties of that group after buying a house or hotel. Improving by another house of all three properties should be possible. Buy two houses on Skywalker Drive, and one house on both Rey Lane and Wookie Hole. Now, it should not possible to buy a house on Skwalker Drive, but possible on Rey Lane and Wookie Hole.

```
@Test
public void canBuyHouseHotel() {
    Player one = board.getAllPlayers().get(0);
    Square skywalkerDrive = board.getAllSquares().get(11);
    Square reyLane = board.getAllSquares().get(14);
    Square wookieHole = board.getAllSquares().get(13);

    one.buyProperty(skywalkerDrive);
    one.buyProperty(reyLane);
    one.buyProperty(wookieHole);

    assertTrue(one.canBuyHouseHotel(skywalkerDrive));
    assertTrue(one.canBuyHouseHotel(reyLane));
    assertTrue(one.canBuyHouseHotel(wookieHole));

    one.buyHouse(skywalkerDrive);

    assertFalse(one.buyHotel(skywalkerDrive));
    assertTrue(one.buyHouse(wookieHole));
    assertTrue(one.buyHouse(reyLane));
}
```

Figure 6.26: Unit Test: Can Buy House or Hotel

Tests for: F10
A player may only sell houses or hotels on a property if there would not be a difference greater than 1 house between all properties of that group after selling a house or hotel. Buy two houses on Skywalker Drive, and one house on both Rey Lane and Wookie Hole. Selling a house of Skywalker Drive should be possible, but not on Rey Lane or Wookie Hole. Only after selling a house on Skywalker Drive should it be possible to sell a house on all of the squares.

```java
@Test
public void canSellHouseHotel() {
    Player one = board.getAllPlayers().get(0);
    one.setBalance(100000);
    Square skywalkderDrive = board.getAllSquares().get(11);
    Square wookieHole = board.getAllSquares().get(13);
    Square reyLane = board.getAllSquares().get(14);
    one.buyProperty(skywalkderDrive);
    one.buyProperty(wookieHole);
    one.buyProperty(reyLane);

    one.buyHouse(skywalkderDrive);
    one.buyHouse(reyLane);
    one.buyHouse(wookieHole);
    one.buyHouse(skywalkderDrive);

    assertTrue(one.canSellHouseHotel(skywalkderDrive));
    assertFalse(one.canSellHouseHotel(wookieHole));
    assertFalse(one.canSellHouseHotel(reyLane));

    one.sellHouse(skywalkderDrive);

    assertTrue(one.canSellHouseHotel(skywalkderDrive));
    assertTrue(one.canSellHouseHotel(wookieHole));
    assertTrue(one.canSellHouseHotel(reyLane));
}
```

Figure 6.27: Unit Test: Can Sell House or Hotel

Tests for: F11
A square can only be sold when all properties with the group of the square to sell have no houses or hotels on them. After buying a house on Crapper Street, it should not be possible to sell either Crapper Street or Gangster's Paradise.

```
@Test
public void canSellProperty() {
    Player one = board.getAllPlayers().get(0);
    Square crapperStreet = board.getAllSquares().get(1);
    Square gangstersParadise = board.getAllSquares().get(3);
    one.buyProperty(crapperStreet);
    one.buyProperty(gangstersParadise);

    assertTrue(one.canSellProperty(crapperStreet));
    assertTrue(one.canSellProperty(gangstersParadise));

    one.buyHouse(crapperStreet);

    assertFalse(one.canSellProperty(crapperStreet));
    assertFalse(one.canSellProperty(gangstersParadise));
}
```

Figure 6.28: Unit Test: Can Sell Property

Tests for: F11
A square cannot be sold if it has houses or hotels on it. Once a property is sold, the square should not be owned by anybody. The square should be buyable and the rent pointer should point to unimproved.

```
@Test
public void sellProperty() {
    Player one = board.getAllPlayers().get(0);
    Square crapperStreet = board.getAllSquares().get(1);
    one.buyProperty(crapperStreet);
    one.buyHouse(crapperStreet);

    assertFalse(one.sellProperty(crapperStreet));

    one.sellHouse(crapperStreet);
    one.sellProperty(crapperStreet);

    assertNull(crapperStreet.getOwnedBy());
    assertTrue(crapperStreet.getCanBeBought());
    assertEquals(crapperStreet.getProperty().getRentPointer(), actual: 0);
}
```

Figure 6.29: Unit Test: Sell Property

Tests for: F10, F12
A house on a property can only be bought if all of these points are true:

1. The player has enough money to buy the house.

2. If, after buying the house, there is not a difference of more than one 1 house between the properties of the group.

3. There is not already 4 houses on the property.

A hotel on a property can only be bought if all of these points are true:

1. There are 4 houses already on the property

2. If, after buying the hotel, there is not a difference of more than one 1 house between the properties of the group.

3. The player has enough money to buy the hotel.

The test below checks these three conditions in order.

```
public void buyHouseHotel() {
    Player one = board.getAllPlayers().get(0);
    one.setBalance(100000);
    Square crapperStreet = board.getAllSquares().get(1);
    Square gangstersParadise = board.getAllSquares().get(3);
    Square skywalkerDrive = board.getAllSquares().get(11);
    one.buyProperty(crapperStreet);
    one.buyProperty(gangstersParadise);

    assertTrue(one.buyHouse(crapperStreet));
    assertTrue(one.buyHouse(gangstersParadise));
    assertEquals(crapperStreet.getOwnedBy(), one);
    assertEquals(gangstersParadise.getOwnedBy(), one);
    assertFalse(crapperStreet.getCanBeBought());
    assertFalse(gangstersParadise.getCanBeBought());
    assertEquals(crapperStreet.getProperty().getRentPointer(),  actual: 1);
    assertEquals(gangstersParadise.getProperty().getRentPointer(),  actual: 1);

    assertTrue(one.buyHouse(crapperStreet));
    assertFalse(one.buyHouse(crapperStreet));
    assertTrue(one.buyHouse(gangstersParadise));
    assertFalse(one.buyHotel(crapperStreet));
    assertFalse(one.buyHotel(gangstersParadise));

    one.buyProperty(skywalkerDrive);
    for (int i = 0; i < 4; i++) {
        assertTrue(one.buyHouse(skywalkerDrive));
    }
    assertFalse(one.buyHouse(skywalkerDrive));
    assertTrue(one.buyHotel(skywalkerDrive));

    one.setBalance(0);
    assertFalse(one.buyHouse(gangstersParadise));
```

Figure 6.30: Unit Test: Buy House and Hotel

Tests for: F10
A house or hotel can only be sold if there are houses/hotel to sell and if, after selling, there is never more than 1 house difference between the properties in the set.

After buying Crapper Street and Gangster's Paradise, the player should not be able to sell houses or hotels on them since there are none to sell. Buying two houses on Crapper Street and one on Gangster's Paradise means the player should not be able to sell the house on Gangster's Paradise, otherwise there would be a difference of more than 1 house.

Buying 4 houses on Skywalker's Drive should now allow the player to buy a hotel. After this, the player should not be able to sell a house before selling the hotel.

```java
@Test
public void sellHouseHotel() {
    Player one = board.getAllPlayers().get(0);
    one.setBalance(10000);
    Square crapperStreet = board.getAllSquares().get(1);
    Square gangstersParadise = board.getAllSquares().get(3);
    Square skywalkerDrive = board.getAllSquares().get(11);
    one.buyProperty(crapperStreet);
    one.buyProperty(gangstersParadise);

    assertFalse(one.sellHouse(crapperStreet));
    assertFalse(one.sellHouse(gangstersParadise));
    assertFalse(one.sellHotel(crapperStreet));
    assertFalse(one.sellHotel(gangstersParadise));
    assertTrue(one.buyHouse(crapperStreet));
    assertTrue(one.buyHouse(gangstersParadise));
    assertTrue(one.buyHouse(crapperStreet));
    assertFalse(one.sellHouse(gangstersParadise));

    one.buyProperty(skywalkerDrive);
    for (int i = 0; i < 4; i++) {
        assertTrue(one.buyHouse(skywalkerDrive));
    }
    assertFalse(one.sellHotel(skywalkerDrive));
    assertTrue(one.buyHotel(skywalkerDrive));
    assertFalse(one.sellHouse(skywalkerDrive));
    assertTrue(one.sellHotel(skywalkerDrive));
    assertTrue(one.sellHouse(skywalkerDrive));
}
```

Figure 6.31: Unit Test: Sell House and Hotel

6.11.2 System Level

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
1	User starts the game.	Running main method in "Main" class.	User should be greeted with a window called "Main Menu". This window should have: a choice box called "Number of Players", a text field called "Path of Card Data", a text field called "Path of Board Data" and a button called "Create Game".	A window called "Main Menu". It has a choice box called "Number of Players", two text fields named "Path of Board Data" and "Path of Card Data" and a button called "Create Game".	PASS
2	User starts the game	Running main method "Main" class.	"2" should be the default option in "Number of Players". The default values in "Path of Board Data" and "Path of Card Data" should be the path to their data files respectively.	"2" is the default option selected in the choice box. Both "Path of Board Data" and "Path of Card Data" have the correct path to their data files.	PASS
3	User clicks the "Number of Players" choice box.	A choice box click event on "Number of Players".	There should be 5 options that lists the number from 2-6. The user can choose any one of the five options.	Clicking the choice box allows the user to choose one of the five options ranging from 2-6.	PASS
4	User clicks "Create Game" button.	A button click event on "Create Game".	Board should be loaded, correctly naming each square. The background of each square should be correct depending on the group of the square.	Board view correctly names each square. Background images correctly displayed only for squares with a group of a colour. The background images of all squares correct	PASS
5	User clicks "Create Game" button.	A button click event on "Create Game".	Board should be loaded with the specified number of players, board data and card data.	The correct number of players is loaded, with the correct board and card data.	PASS
6	User clicks "Roll Dice" button	A button click event on "Roll Dice", where the user rolls a double.	"Roll Dice" should be clickable, "End Turn" should not. The game log should have a message displaying the ID of the player who rolled the double.	"Roll Dice" is clickable, "End Turn" is not. The game log adds a message showing the player's ID who rolled the double.	PASS
7	User presses "Roll Dice" button.	A button click event on "Roll Dice", where player's token passes "GO".	After a player has passed "GO", £200 should be added to their balance.	£200 is added to the players' balance every time they pass "GO".	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
8	User presses "Roll Dice" button.	A button click event on "Roll Dice", whenever a player lands on a property that can be bought.	A new window, called "Buy Property?". Based on the square the player lands on, it should display the name, position, group, cost, rental prices, and cost of buying a house and hotel. There should be three buttons, called "Yes", "No" and "Confirm Choice".	A new window called "Buy Property?" is opened. It displays the correct name, position, group, rental prices and cost of buying a house and hotel on the square.	PASS
9	User presses "Yes" in "Buy Property?".	A button click event on "Yes" in the "Buy Property?" window.	"Yes" button should not be clickable. "No" button should be clickable.	"Yes" button is not clickable. "No" button is clickable.	PASS
10	User presses "No" in "Buy Property?".	A button click event on "No" in the "Buy Property?" window.	"No" button should not be clickable. "Yes" button should be clickable.	"No" button is not clickable. "Yes" button is clickable.	PASS
11	User presses "Yes" and then "Confirm Choice " button in "Buy Property?".	A button click event on "Confirm Choice".	The "Buy Property?" window should close. The player buys the property if they have enough money in their balance, otherwise they do not. A suitable message should be added to the game log on whether the player bought the property or not. The balance of the player should be updated.	The "Buy Property" window closes. The property is bought if player has enough money in their balance, is not bought otherwise. A suitable message is added to the game log depending on the outcome. Balance of the player is updated.	PASS
12	User presses "No" and then "Confirm Choice" button in "Buy Property?".	A button click event on "Confirm Choice".	The "Buy Property?" window should close. A suitable message should be added to the game log mentioning the property not bought.	"Buy Property?" window closes. A suitable message is added to the game log.	PASS
13	User presses "Status" button.	A button click event on "Status".	A new window called "State of all players" should be opened. Based on the number of players remaining in the game, the window should have the correct number of ListView objects. These objects should always contain the current properties each player owns.	A window called "State of all players" is opened. It has the correct number of ListView objects, with each ListView correctly showing the properties each player owns.	PASS
14	User presses "Roll Dice" button.	A button click event on "Roll Dice", where the player lands on a property owned by a different player and the player has enough money in their balance to pay the rent.	The current player should pay the correct rent of the property to the player who owns the square. The balances of the players should be updated.	The current player pays the correct rent of the property to the player who owns the square. The balances of the players is updated	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
15	User presses "Roll Dice" button.	A button click event on "Roll Dice", where the player lands on a property owned by a different player and does not have enough in their balance, but their net worth is enough to pay the rent.	A message instructing the player to pay the correct rent of the property to the player who owns the property should be added to the game log.	A message instructing the player to pay the correct rent of the property to the player who owns the property is added to the game log.	PASS
16	User presses "Roll Dice" button.	A button click event on "Roll Dice", where the player lands on a property owned by a different player and their net worth is not enough to pay the rent.	The player should be removed from the game. Their token, and all properties they own should buyable and reset. Their balance label should also be removed.	The player is removed from the game. Their token, and all properties they own is buyable and have no houses or hotels on them. Their balance label is removed.	PASS
17	User presses "Buy/ Sell" button.	A button click event on "Buy/Sell".	A window called "Improve and Sell Properties" should open. There should be three ListView objects. The furthest to the left should list all properties the current player owns. The other two should be empty.	A window called "Improve and Sell Properties" is opened. There are three ListView objects, with the left-most ListView showing all the properties the current player owns and the other two are empty.	PASS
18	User selects a property in "Improve and Sell Properties" window.	A ListView click event on list of properties.	Whenever the user selects a property, the centre ListView obect should display the selling price of that property, and cost/ selling price of buying/selling a house and hotel on the property.	The centre ListView object is updated to display the selling price of the property, and cost/ selling price of buying/selling a house and hotel of the selected property.	PASS
19	User selects a property belonging to a colour group in "Improve and Sell Properties" window.	A ListView click event on list of properties.	Below the list of properties, there should be information showing number of houses and hotels currently on the selected property.	Shows the correct number of houses and hotels on the property.	PASS
20	User selects a station or utility in "Improve and Sell Properties" window.	A ListView click event on list of properties.	The information about the number of houses and hotels on the property should be empty.	No information about number of houses and hotels.	PASS
21	User presses "Buy" button in "Improve and Sell Properties.	A button click event on "Buy" when the sell property option is selected.	The transaction should be declined with a message instructing the player to press "Sell", not "Buy", to sell the property.	Transaction is declined, with a message instructing the player to press "Sell", not "Buy", to sell the property.	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
22	User presses "Sell" button in "Improve and Sell Properties".	A button click event on "Sell" when the sell property option is selected. The property has no houses/hotels and all properties of the same group also have no houses/hotels.	The transaction should be accepted, with the selling price of the property added to the player's balance.	Transaction is accepted, with the selling price of the property added to the player's balance.	PASS
23	User presses "Sell" button in "Improve and Sell Properties".	A button click event on "Sell" when the sell property option is selected. The property has houses/ hotels.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS
24	User presses "Sell" button in "Improve and Sell Properties".	A button click event on "Sell" when the sell property option is selected. The property to be sold has no houses or hotels, but other properties of the same group have houses/hotels.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS
25	User presses "Sell" button in "Improve and Sell Properties".	A button click event on "Buy" when the house option is selected. The property to buy a house on does not have the maximum number of houses.	Transaction should be accepted. The number of houses and rent of the property should be updated. The balance of the player should be updated.	Transaction is accepted. The number of houses and rent of the property is updated. The balance of the player is updated.	PASS
26	User presses "Buy" button in "Improve and Sell Properties.	A button click event on "Buy" when the house option is selected. The property to buy a house on has the maximum number of houses.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS
27	User presses "Buy" button in "Improve and Sell Properties".	A button click event on "Buy" when the house option is selected. The player has enough money to buy a house on the property, owns all property of the colour group and there would not be a difference of more than 1 house between the properties.	Transaction should be accepted. The number of houses and rent of the property should be updated. The balance of the player should be updated.	Transaction is accepted. The number of houses and rent of the property is updated. The balance of the player is updated.	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
28	User presses "Buy" button in "Improve and Sell Properties".	A button click event on "Buy" when the house option is selected. The player does not have enough money to buy a house on the property.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS
29	User presses "Buy" button in "Improve and Sell Properties".	A button click event on "Buy" when the house option is selected. The property to buy a house on would have a difference of more than 1 house between other properties of the same group.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS
30	User presses "Buy" button in "Improve and Sell Properties".	A button click event on "Buy" when the hotel option is selected. There are 4 houses on the property and no hotels.	Transaction should be accepted. The number of hotels and rent of the property should be updated. The balance of the player should be updated.	Transaction is accepted. The number of hotels and rent of the property is updated. The balance of the player is updated.	PASS
31	User presses "Buy" button in "Improve and Sell Properties".	A button click event on "Buy" when the hotel option is selected. There are fewer than 4 houses on the property.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS
32	User presses "Buy" button in "Improve and Sell Properties".	A button click event on "Buy" when the hotel option is selected. There is a hotel on the property already.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS
33	User presses "Buy" button in "Improve and Sell Properties".	A button click event on "Buy" when the hotel option is selected. The property to buy a hotel on would make a difference of more than 1 house between other properties of the same group.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
34	User presses "Sell" button in "Improve and Sell Properties".	A button click event on "Sell" when the house option is selected. The property to sell a house on has a house and selling a house would not make a difference of more than 1 house between other properties of the same group.	Transaction should be accepted. The number of houses and rent of the property should be updated. The balance of the player should be updated.	Transaction is accepted. The number of houses and rent of the property is updated. The balance of the player is updated.	PASS
35	User presses "Sell" button in "Improve and Sell Properties".	A button click event on "Sell" when the house option is selected. The property has no houses to sell.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS
36	User presses "Sell" button in "Improve and Sell Properties".	A button click event on "Sell" when the house option is selected. The property to sell a house on would make a difference of more than 1 house between other properties of the same group.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS
37	User presses "Sell" button in "Improve and Sell Properties".	A button click event on "Sell" when the hotel option is selected. The property to sell a hotel on has 1 hotel and selling the hotel would not make a difference of more than 1 house between other properties of the same group.	Transaction should be accepted. The number of hotels and rent of the property should be updated. The balance of the player should be updated.	Transaction is accepted. The number of houses and rent of the property is updated. The balance of the player is updated.	PASS
38	User presses "Sell" button in "Improve and Sell Properties".	A button click event on "Sell" when the hotel option is selected. The property has no hotel to sell.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS
39	User presses "Sell" button in "Improve and Sell Properties".	A button click event on "Sell" when the hotel option is selected. The property to sell a hotel on would make a difference of more than 1 house between other properties of the same group.	Transaction should be declined, listing all the possible reasons for the failure in the right-most ListView object.	Transaction is declined and a list of all possible reasons of failure is displayed in the right-most ListView object.	PASS

Test Number	Testing For
1	T5, NF1
2	F13
3	F14, NF4
4	T10
5	T5
6	T2, T6, NF2
7	T1, F1, F2
8	T7, F6
11	F6, F7
12	F5, F6
13	T4, F4
14	T8, F14
16	T8, F14
17	T9, F8
18	F8
19	F8
20	F8
21	F8
22	F8, F10, F11
23	F8, F11
24	F8, F10
25	F8, F9, F10
26	F8, F12
27	F8, F9, F10
28	F8
29	F8, F10
30	F8, F10
31	F8, F10
32	F8, F12
33	F8, F10
34	F8, F10, F11
35	F8
36	F8, F10
37	F8, F10, F11
38	F8
39	F8, F10

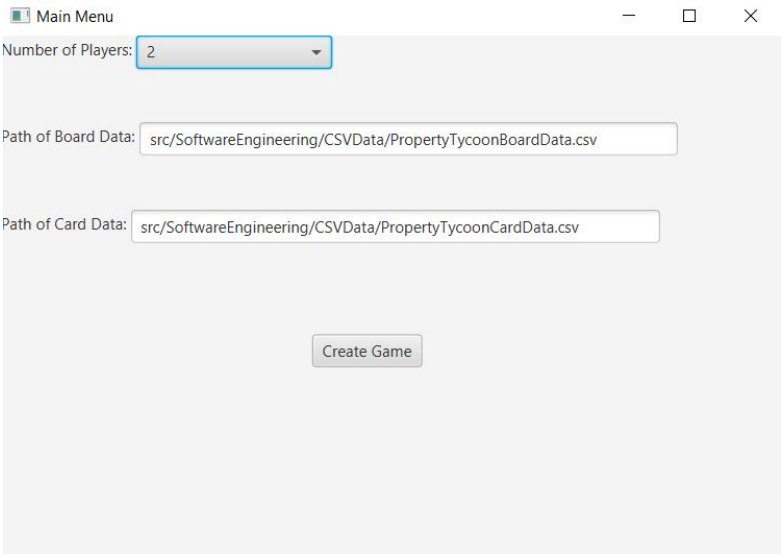


Figure 6.32: Menu after running "Main.java"

Figure 6.33: Board View

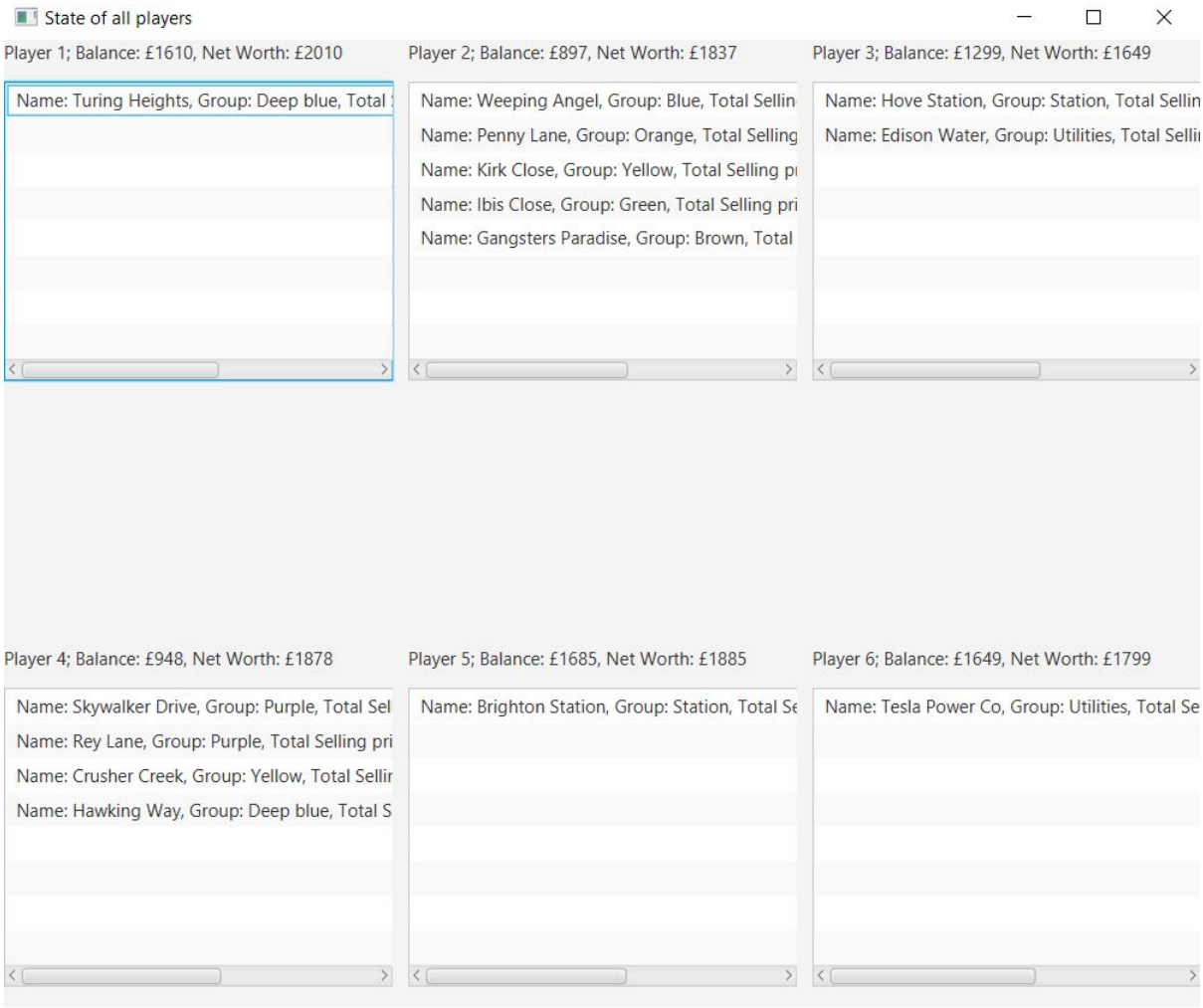


Figure 6.34: Status View

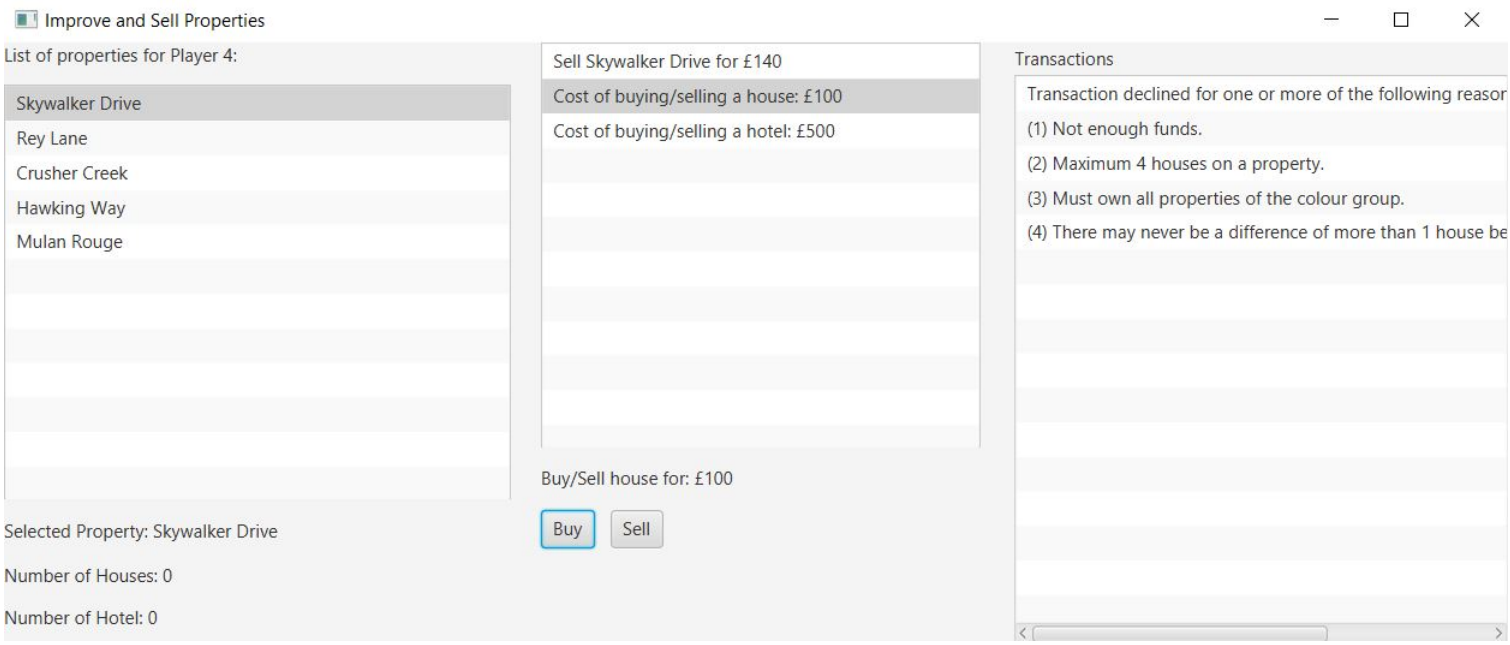


Figure 6.35: Buy and Sell View

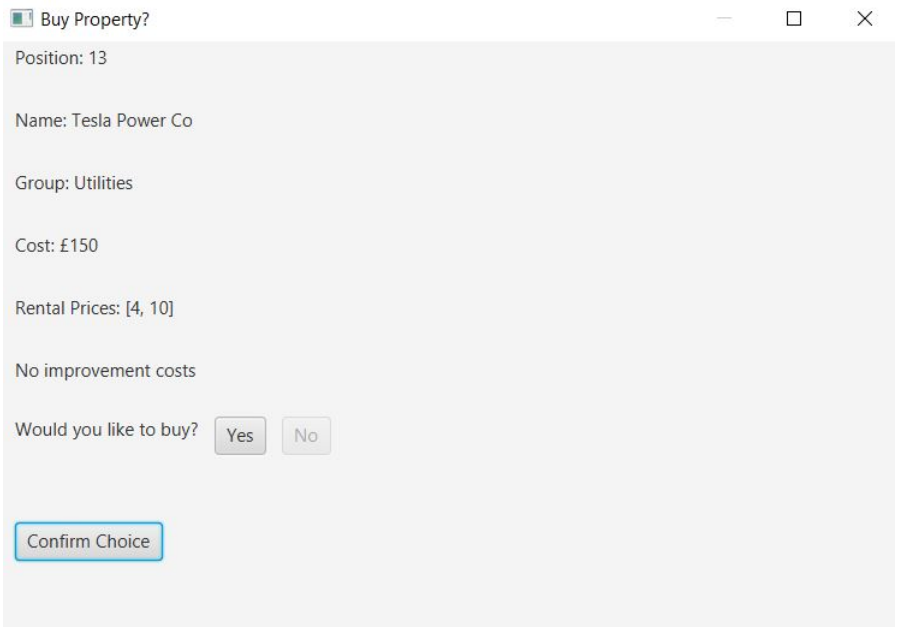


Figure 6.36: Buy Property View

6.12 Reflection

Overall, this sprint was hugely successful especially when considering the Covid-19 situation. Multiple key functionalities of Property Tycoon were implemented; the most important and complex part of the game being the buying and selling of houses, along with the ability to see the state of the board.

We also amended a key problem from the end of sprint 2, which was the background images of non-colour grouped squares not appearing. We were able to solve this by inserting our own data, with the permission of the customer.

Even though this sprint was more complex than both sprint 1 and 2, it went smoother than both. We learnt our lesson from sprint 1 and designed the code more cohesively, such as completely separating GUI elements from game logic, and GUI windows from one another by using separate controllers.

Next sprint, we will focus on implementing the final rules of Property Tycoon, such as what happens when a player lands on "Pot Luck" or "Opportunity Knocks", namely the actions of cards. We will add functionality to the following squares:

- Go to Jail square
- Income Tax
- Super Tax
- Free Parking
- Pot Luck
- Opportunity Knocks

Feedback from the customer was positive, and they were fond of the buying/selling system and the status view. The customer has recommended adding a list of all available properties that have not been purchased in the status view.

The code for the first prototype can be found in the submission folder, called "ThirdPrototype".

6.13 Individual Key Contributions

Test Member	Key Contribution(s)
Jun	T2, T4, T9, System Testing
Jihye	T4, T9, T11, System Testing
Ye-Rang	T6, T7, T8, System Testing
Ibi	T1, T3, T5, T9, T10, Unit Testing, System Testing

Sprint 4

Team Number: 32
Sprint Technical Lead: Ibi
Sprint start date: 19th March 2020
Sprint end date: 10th April 2020

7.1 User Stories

This sprint will produce the fourth prototype. The main focus will be implementing the action of card in Pot Luck and Opportunity Knocks. When the user lands on either Pot Luck or Opportunity Knocks, they will draw a card from the respective pile and automatically carry out the instructions of the card. We inquired to the customer about how best to go about making the card actions automatic, since the original column of actions were not useful in coding terms.

I quote, "the actions form a small set, basically moving (e.g. go back 3 spaces) paying money or receiving money from other players. This means that the card action can be made automatic" - Customer. Hence, the description of the card should be customizable, but the action should not change. For example, a card with description "Drunk in charge of a skateboard, pay a fine of £20", could also be "Caught speeding on the motorway, pay a fine of £20". The key word in both descriptions is "fine", which makes up a fundamental action of the game.

When a player lands on "Go to Jail", or draws the "Go to Jail", they are sent to jail. The player should be offered to pay bail. If they accept, then they can take their next turn as normal. Otherwise, they miss their next two turns. A player should be sent to jail if they roll three doubles consecutively.

If a player does not have enough money to pay the rent, but would have enough if they sold properties, then a window should open that forces the player to sell assets until they have paid the rent.

Along with options of accepting/declining to purchase a square, there should also be an option to mortgage the square for half the original price. A mortgaged property cannot collect rents until that property has been fully bought. A player should be able to pay off the mortgage of a property.

If time allows, whenever a player declines or is unable to buy/mortgage a property, an auction is opened where players can make private bids. Only players who have passed "GO" can participate in the auction.

Finally, players should be able to view a list of all buyable squares.

7.2 Task Cards

Scoring scale: 1 to 10, with 10 having the highest priority and 1 the lowest. A task card with priority 10 means that it must be completed this sprint, otherwise the prototype will not work or make enough progress from the previous prototype. Alternatively, a 10 can also be a feature that the customer has recommended. A card with priority 1 means that it does not have to be completed this sprint, and without it, it will not impact the current prototype.

1. Priority: 10, Complexity: 5

   Create a list of all fundamental card actions. The actions should be summarised by a single word.

2. Priority: 10, Complexity: 7

   When a player lands on "Pot Luck" or "Opportunity Knocks", they shall draw a card from the respective pile. The action of the card shall be automatic and shall be coherent with the description of the card. For example, if the description is "Fell down the stairs, pay £100 in hospital bills", then the player should pay £100, and not a different number like "£20".

3. Priority: 10, Complexity: 6

   Whenever a player is sent to jail their token is moved to the jail and a window shall open offering the player to pay a bail of £50. If they accept, then the player can take their next turn as normal. If they decline, they give up their next two turns.

4. Priority: 5, Complexity: 3

   When a player rolls three consecutive doubles, they shall be sent to jail.

5. Priority: 7, Complexity: 2

   If a player owns all of the properties in a colour coded group, but the properties are otherwise not developed further with houses and hotels, then the rent due on that colour group is double.

6. Priority: 5, Complexity: 2

   A player in jail cannot collect rent from any of their properties whilst in jail.

7. Priority: 8, Complexity: 1

   Whenever a player lands on "Go to Jail", they are sent to jail.

8. Priority: 4, Complexity: 4

   When a player lands on a square that is buyable, they shall have the option to mortgage the square. A player can mortgage a square for half the original price.

9. Priority: 4, Complexity: 2

   When a player lands on a property that is mortgaged, they do not pay the rent of the property.

10. Priority: 8, Complexity: 7

    Players shall have the option of paying off mortgages on their properties. The cost of paying off the mortgage is half the original cost of the property.

11. Priority: 4, Complexity: 1

    Players selling mortgaged properties shall sell for half the original price of the property.

12. Priority: 2, Complexity: 2

    Whenever a fine is paid, proceeds accumulate in "free parking". When a player lands on "free parking", they collect all the funds from it.

13. Priority: 5, Complexity: 1

    Whenever a player lands on either "Income Tax" or "Super Tax", they pay the tax respectively. If they do not have enough money in their balance, but their net worth is enough, then a window opens forcing the player to sell assets until the tax is paid.

14. Priority: 1, Complexity: 2

    If a player has a "Get of out jail free card" and is sent to jail, they use their "Get out of jail free card" to be released from prison.

15. Priority: 1, Complexity: 8

    If a player declines or is unable to buy a property, an auction is started. Only players who have passed "GO" can participate in the auction. Bids are only valid if the player making the bid has enough money in their balance.

16. Priority: 7, Complexity: 7

    Whenever a player is unable to pay a required amount of money from their balance, such as rent or tax, a window opens forcing the player to sell properties until the amount required can be paid off. This window should only open if the net worth of the player is enough to pay off the required amount.

17. Priority: 10, Complexity: 4

    In the "Status" window, there shall be a list of all buyable squares remaining on the board.

7.3 Requirements Analysis

7.3.1 Functional Requirements

F1 - Mandatory

The set of action shall be derived from the descriptions of the card. The action starts with one keyword, followed by an integer/name of square if needed. For example, a possible action for a card with description "Drunk in charge of a skateboard, pay a fine of £20" could be "Pay £20".

F2 - Mandatory

Based upon the action keyword, automatically carry out the action of the card. Note that the amount to pay/name of square could change so the action must take this into account.

F3 - Mandatory

In the rolledTheDice() method in "Controller.java", add a check to see whether the player has a double three consecutive times. If they have, send them to jail.

F4 - Mandatory

Whenever a player is sent to jail, a window opens with two buttons, named "Yes" and "No". Regardless of the choice they choose, their token is moved to "Jail/Just Visting". They do not collect £200 if they pass "Go". The window asks the player if they would like pay the £50 bail to be released from jail. If "Yes" is clicked, and they have enough money, they are released from jail. If the player clicks "No", they are sent to jail.

F5 - Mandatory

Whenever a player is unable to pay a certain amount of money because of rent, tax, card action, etc, a window opens that includes lists all the properties the player owns, along with buttons "Buy" and "Sell" that allow the player to buy/sell houses, hotels and properties. The player is instructed to sell assets until the debt is paid off. The window automatically closes once the debt has been paid off.

F6 - Mandatory

Refactor "BuyProperty.fxml" and "BuyPropertyController.java" to add a button called "Mortgage" that allows the player to mortgage the property for half the price.

F7 - Mandatory

There shall be a button called "Pay off Mortgage" that, when clicked, opens a new window showing a list of all properties currently under mortgage for the player. There shall be a button called "Buy" in this window that will pay off the mortgage of the selected property. There shall also be a label telling the player if their transaction was successful or not, and if not, why not.

F8 - Desirable

Whenever a player declines or is unable to buy a property, an auction is started. In this case, a new window is opened with information about the square to be bought. There should be labels for every player participating in the auction. Under every label there should be a method to enter a sum of money as a bid. Only players who have passed "Go" can participate in the auction.

If no bids are made, then the auction closes and the property remains unsold. If there is a tie, then the auction is restarted as wanted by the customer. The winner of the auctions buys the property for the amount they bid for. Players who make bids that they cannot afford have their bids deemed illegitimate and discarded.

F9 - Desirable

If a player does not have enough money to pay the rent, but would have enough if they sold properties, then a window should open that forces the player to sell assets until they have paid the rent. Refactor "BuySellController.java" to facilitate this need. This means that there should be a label informing the amount of money that needs to be raised. The window should close once enough assets have been sold to pay off the required amount.

F10 - Mandatory

In the "Buy Property?" window, add another button called "Mortgage" that lets the player mortgage the property for half the original cost. The mortgaged price shall also be visible.

7.3.2 Non-Functional Requirements

NF1 - Mandatory

The player shall enter bids in a JavaFX PasswordField[16]. Any character entered in the PasswordField object is hidden, which makes the bids private.

NF2 - Mandatory

The window to pay off mortgages shall have a ListView object containing all mortgaged properties the player owns. There shall be a transaction label informing the player the result of the transaction. There shall be a single button called "Buy" below the ListView object.

NF3 - Mandatory

The auction window shall contain Label objects that shows the player's id. Below this label will be a PasswordField to enter bids for the corresponding player.

NF4 - Desirable

There shall be a JavaFX Label that displays the current amount of money accumulated in "free parking" in the board view.

NF5 - Desirable

The PasswordField used to enter bids should only permit numeric values.

7.3.3 Domain Requirements

D1 - Desirable

The auction system shall be simple and easy to use. Using intuition alone, players should be able to understand how to use the auction. See NF3 and NF5.

7.4 Use Case Diagram

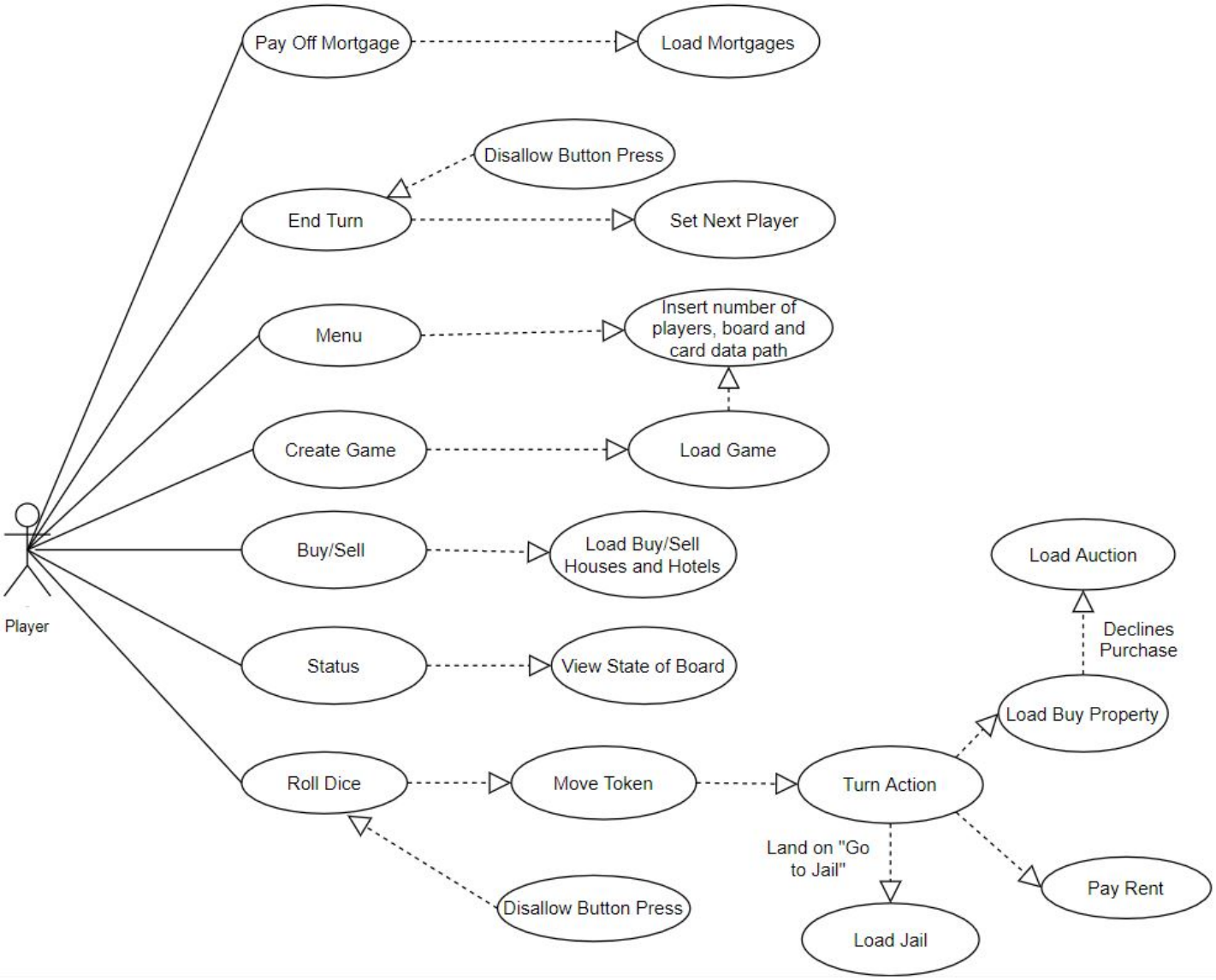


Figure 7.1: Fourth Prototype: Use Case Diagram

7.5 High Level Diagram

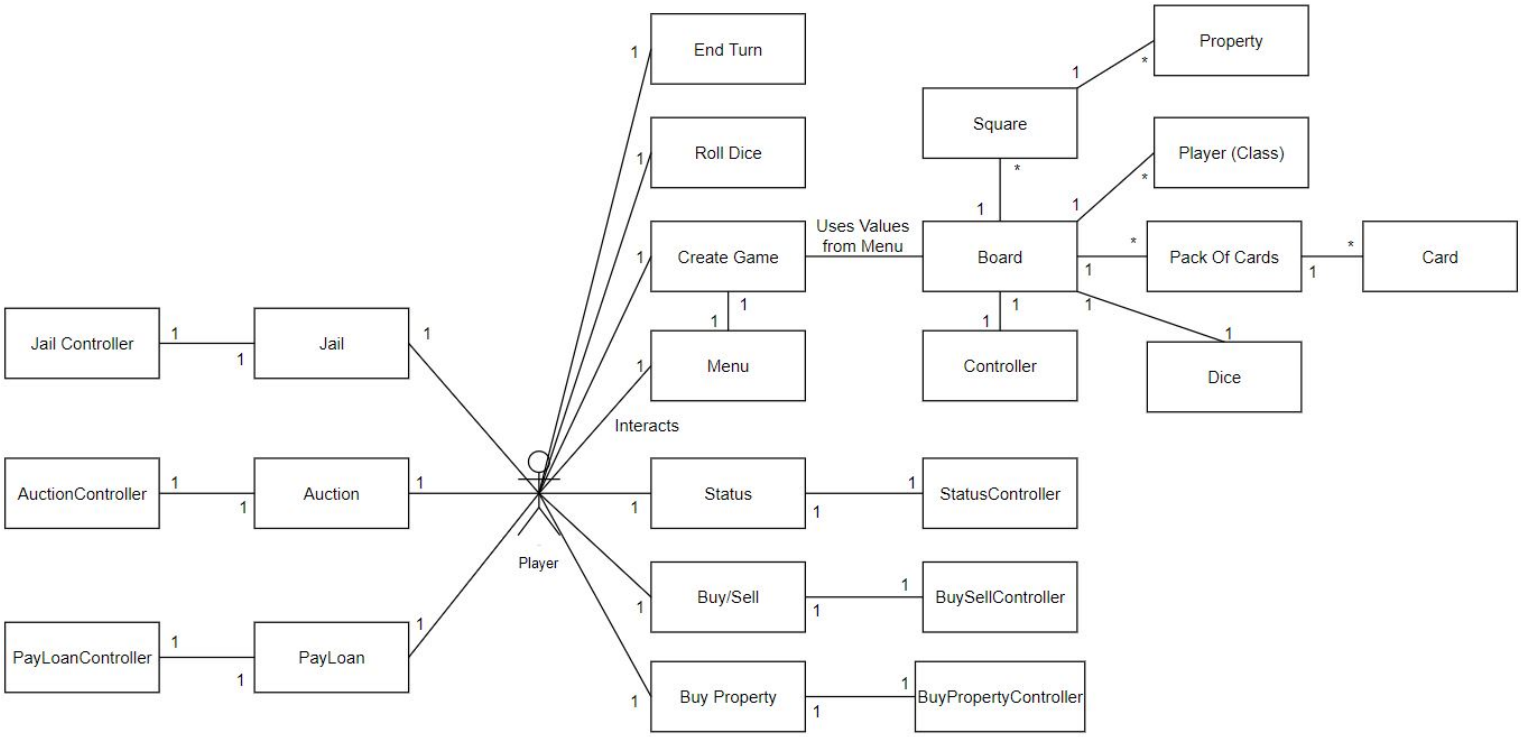


Figure 7.2: Fourth Prototype: High Level Diagram

7.6 Class Diagram

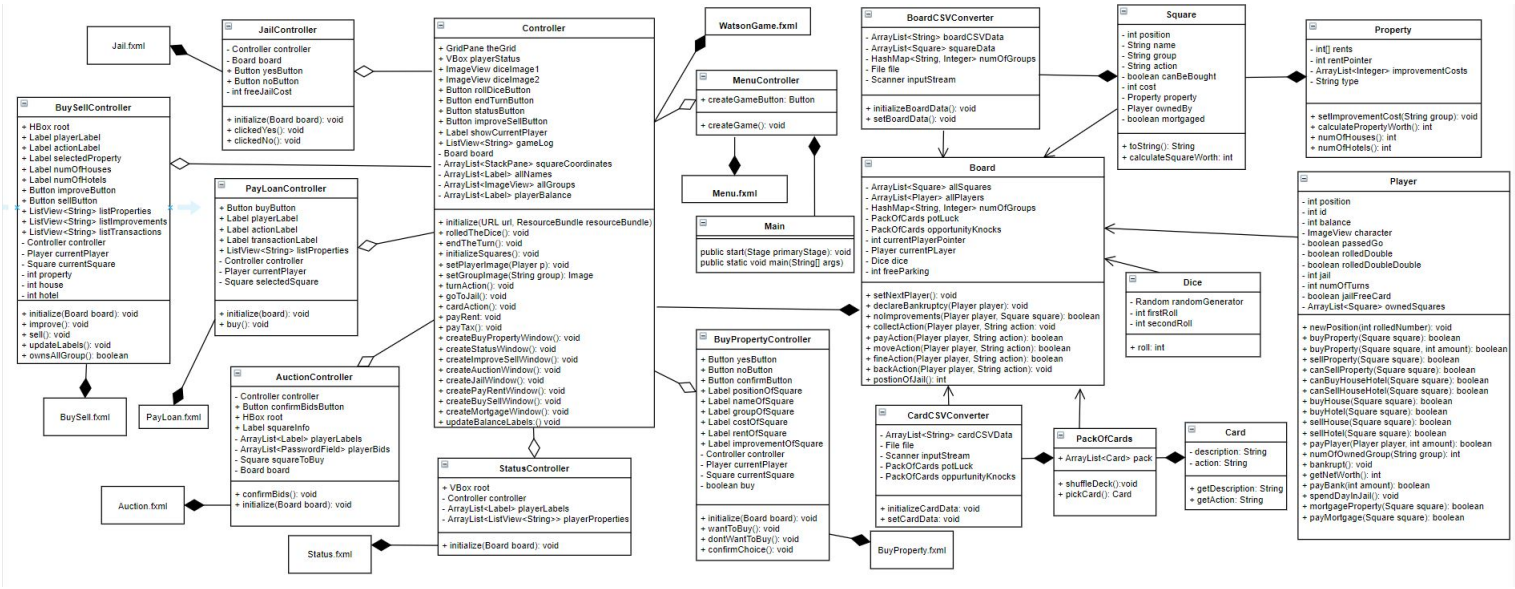


Figure 7.3: Fourth Prototype: Class Diagram

In order to keep the class diagram concise and presentable, getter and setter methods have been omitted with permission from the customer. Note that they are included in the code and used in sequence diagrams even though they do may not appear in the class diagram.

The class diagram can be viewed clearly in Draw.io. To open our class diagrams, please read chapter 1. The first class diagram is located in the "Class Diagrams" folder, called "FourthClassDiagram.io".

7.7 Sequence Diagrams

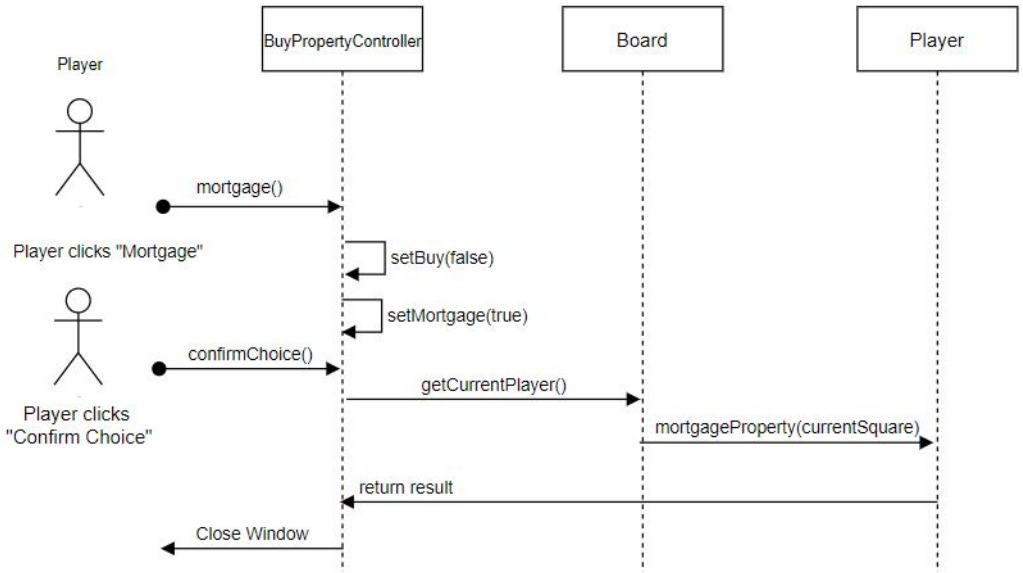


Figure 7.4: Sequence of actions when a player wants to mortgage a square

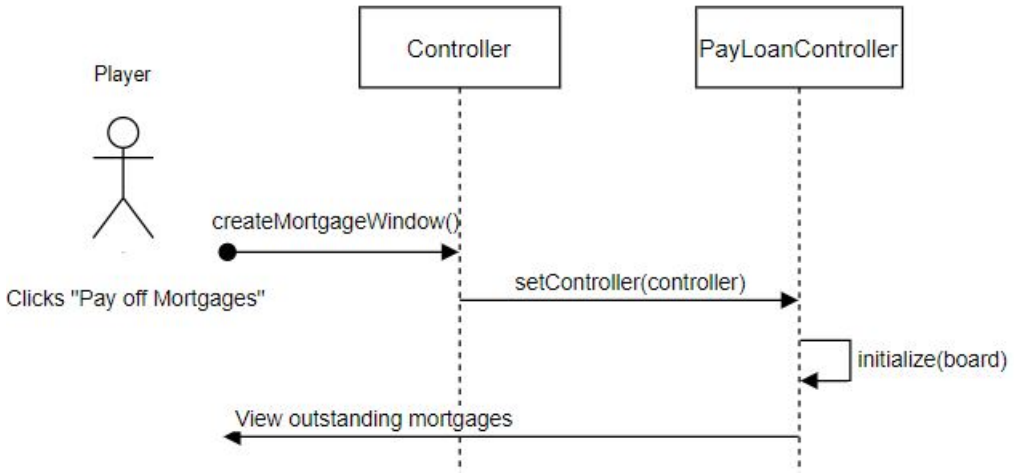


Figure 7.5: Sequence of actions when a player clicks "Pay off Mortgages"

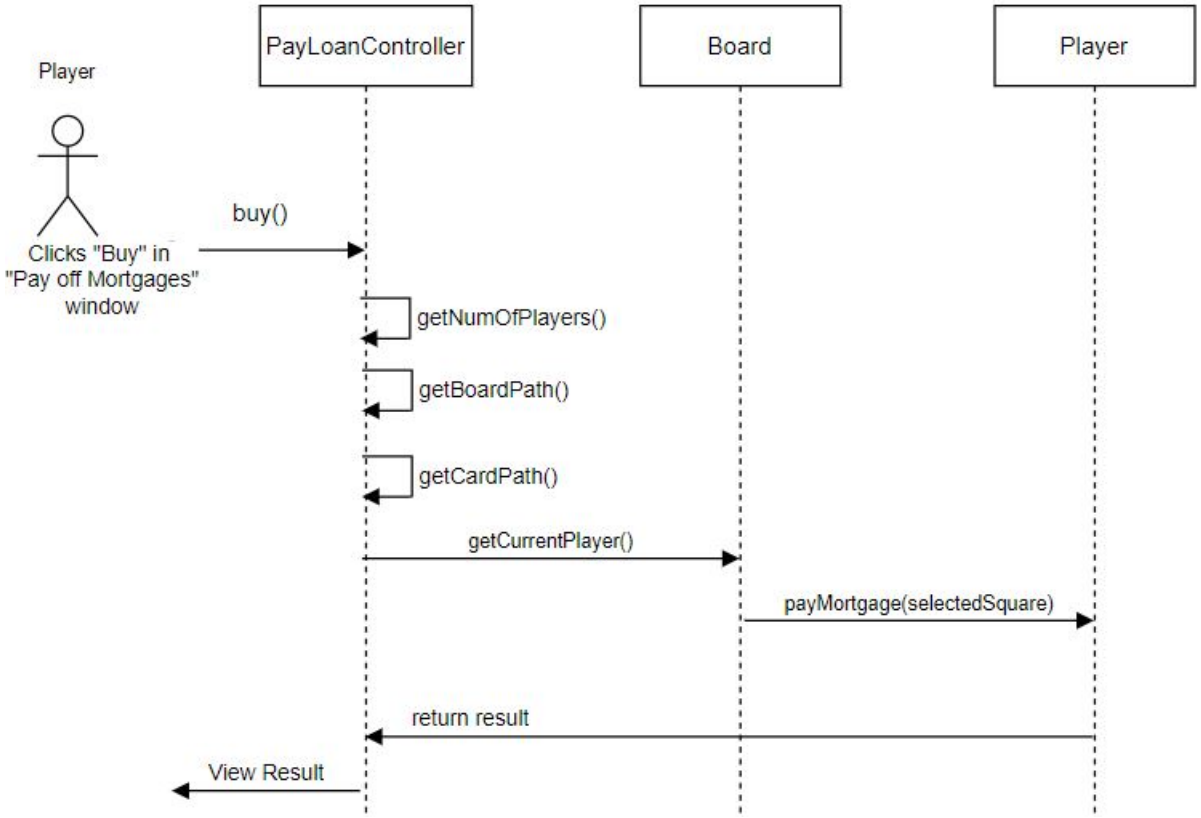


Figure 7.6: Sequence of actions when a player clicks "Buy" in "PayLoan.fxml"

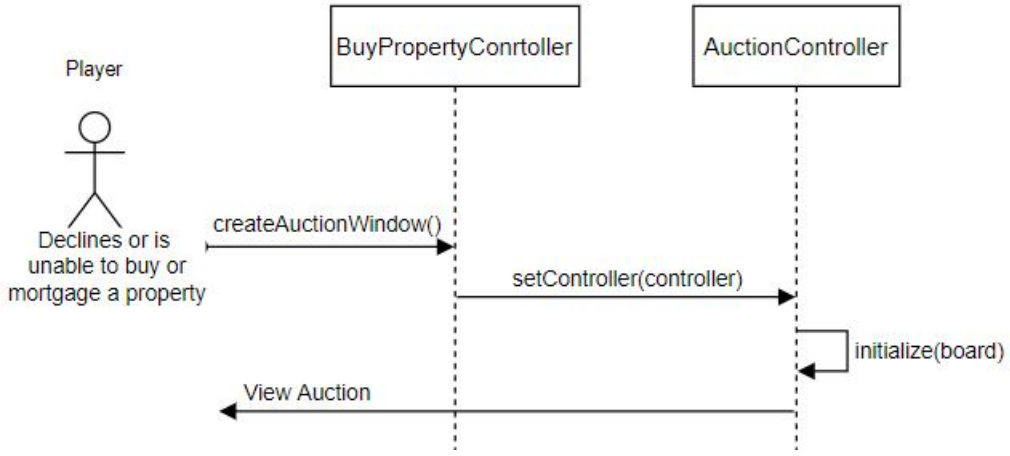


Figure 7.7: Sequence of actions when an auction is started

7.8 PERT Chart

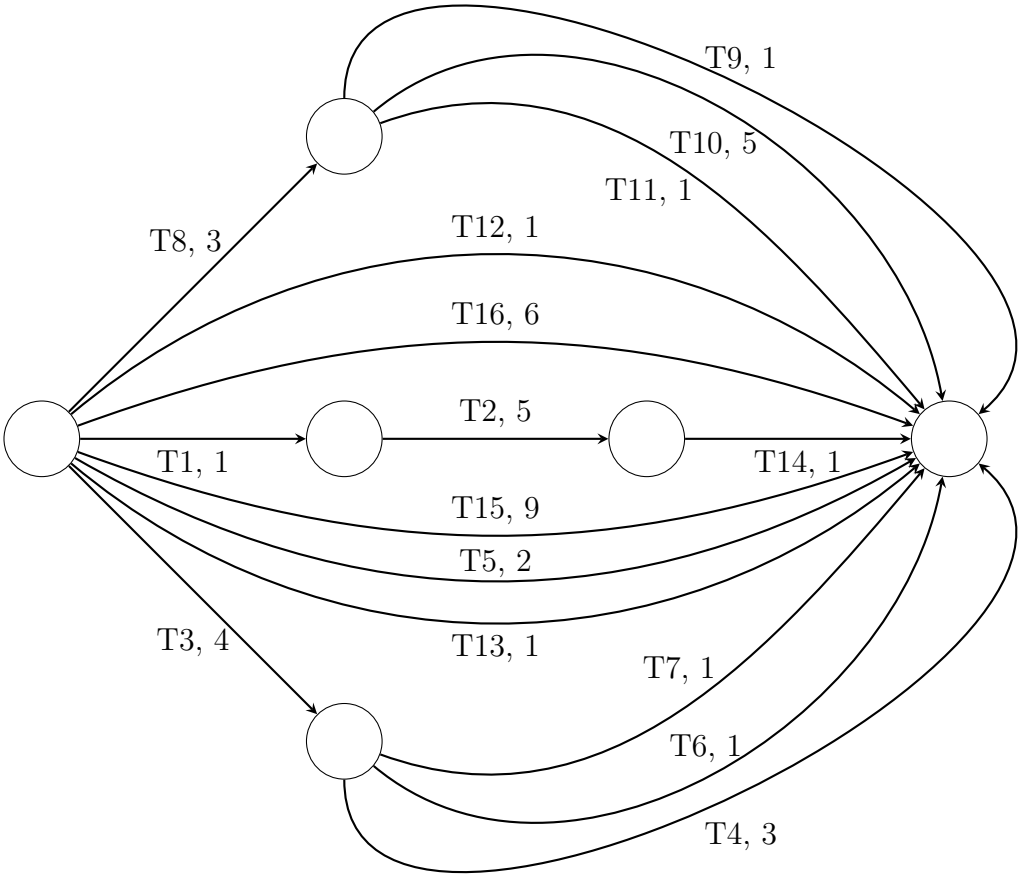


Figure 7.8: Fourth Prototype: PERT Chart

Task Card	Days				Critical Path
	Earliest Start	Earliest Finish	Latest Start	Latest Finish	
1	0	1	2	3	No
2	1	6	3	8	No
3	0	4	2	6	No
4	4	7	6	9	No
5	0	2	7	9	No
6	4	5	8	9	No
7	4	5	8	9	No
8	0	3	1	4	No
9	3	4	8	9	No
10	3	8	4	9	No
11	3	4	8	9	No
12	0	1	8	9	No
13	0	1	8	9	No
14	6	7	8	9	No
15	0	9	0	9	Yes
16	0	6	3	9	No

Critical Path: T15

7.9 Risk Management

Risk Identification	Causes and Likelihood	Mitigation	Monitoring
We will be using JavaFX to create GUI elements of the game. Some members may not be comfortable with using JavaFX.	Likelihood: 1 Members may not have any experience using JavaFX. Some members prefer using Swing over JavaFX.	If not enough progress is being made with the code, swap to using Swing.	Review progress during every team meeting. Technical Supports should update the Technical Director about their progress with using JavaFX.
Software Development may be delayed.	Likelihood: 3 Deadlines for other assessments coming up. As a result, focus of members may be elsewhere near these assessments.	Plan work to take into account future deadlines. Project Manager organises tasks with deadlines in mind.	Project manager discusses future tasks and assigns tasks to each member during team meetings.
Members may have trouble completing a coding task, which can slow progress.	Likelihood: 2 Technical Supports having difficulty with completing a task.	Technical Support should notify the Technical Director for assistance.	Technical Supports will ask for support, if needed, from the Technical Director, during team meetings.
Covid-19 is becoming an increasingly likely problem to cause disturbance.	Likelihood: 2 A member may travel back to their home country, or they may have family members affected by Covid-19.	Members should tell the Project Manager about any travel arrangements or problems in advance, so he can create a better plan and offload work to members not affected.	Simply send a message to the Project Manager should such problems arise. The team is more than willing to help affected members.
Members are travelling back to their home country due to Covid-19. As such, progress may be delayed.	Likelihood: 5 Members are travelling back to their home country. Understandably, no work should be done by these members for at least a week to help them reset.	Work will be off loaded to the Technical Director who has not been affected by Covid-19. After a week break for affected members, they will continue working on the project.	Technical Director will let members know about progress during the week off. As of now, we are ahead of schedule, so progress is looking good.

Likelihood scores are between 1-5, with 1 being highly unlikely and 5 being highly likely.

7.10 Implementation of Task Cards

From the original descriptions of the cards, we have created a list of key words that sum up the action of a card.

- Collect: The player collects the sum of money shown in the description.
- Pay: The player pays the sum of money shown in the description to the bank, i.e. nobody.
- Fine: The player pays the sum of money shown in the description. This sum is added to "free parking".
- Move: The player moves to the square named in the description. If the square named is not the jail the player collects £200 if they pass "GO".
- Jail: The player is sent to jail.
- Free: The player receives a "Get out of Jail card".
- Back: The player moves backwards by a number of squares specified in the description.
- Repair: The player pays money to the bank for the cost of repairing every house and hotel the player owns. The rates are specified in the description.
- Receive: The player receives the sum of money specified in the description from every other player.

Our system allows not only the description of cards to be customizable but also the value of numbers and name of squares. For example, both descriptions "Bought a train ticket, pay £20" and "Bought a concert ticket, pay £50" will function appropriately. We believe this customizability of the cards separates our product from others.

The actions of the cards should take a set form. The form that we have decided on consists of two parts. First, the action begins with one of the aforementioned keywords. Based on the keyword, an appropriate input follows it. Our game is extremely customizable, which is good for the user but it does not come with out its limitations - these are discussed later on.

Whilst coding, we noticed an inconsistency in the rules of Property Tycoon. Rules 7 and 27 contradict each other:

Rule 7 - If a player lands on a “pot luck” or “opportunity knocks” space, they take a card for the top of the corresponding pile and carry out the instructions on the card. When this is complete, the card is replaced at the bottom of the corresponding pile.

Rule 27 - If a player has a “get out of jail free” card, then they place the card at the bottom of the “pot luck” or “opportunity knocks” pile as appropriate, the player token is moved to “just visiting” and the players turn ends. The player takes a normal turn in the next round.

Rule 7 states that ”the card is replaced at the bottom of the corresponding pile”. However, in rule 27, if a player has a ”get out of jail free” card then they place the card at the bottom of the pile they obtained it from. This implies that when a ”get out of jail free” card is drawn, it is not replaced at the bottom of the pile which contradicts rule 7.

We have decided that a ”get out of jail free” card will behave like all other cards in the sense that they are replaced at the bottom of the pile after drawing, instead of the player ”taking” the card.

Description			Action
"Bank pays you divided of £50"			Collect £50
"You have won a lip sync battle. Collect £100"			Collect £100
"Advance to Turing Heights"			Move to Turing Heights
"Advance to Han Xin Gardens. If you pass GO, collect £200"			Move to Han Xin Gardens
"Fined £15 for speeding"			Fine £15
"Pay university fees of £150"			Pay £150
"Take a trip to Hove station. If you pass GO collect £200"			Move to Hove Station
"Loan matures, collect £150"			Collect £150
"You are assessed for repairs, £40/house, £115/hotel"			Repair £40/house £115/hotel
"Advance to GO"			Move to Go
"You are assessed for repairs, £25/house, £100/hotel"			Repair £25/house £100/hotel
"Go back 3 spaces"			Back 3 spaces
"Advance to Skywalker Drive. If you pass GO collect £200"			Move to Skywalker Drive
"Go to jail. Do not pass GO, do not collect £200"			Jail
"Drunk in charge of a skateboard. Fine £20"			Fine £20
"Get out of jail free"			Free jail card

Figure 7.9: An example of a correct set of actions

7.11 Testing

7.11.1 Unit Level

Tests for: F2
Test the player's balance after drawing a card with the action "collect". The player should receive the amount provided by the action.

```
@Test
public void collectAction() {
    Player one = board.getAllPlayers().get(0);
    int balance = one.getBalance();
    board.collectAction(one,  action: "Collect £100");
    assertEquals( expected: balance + 100, one.getBalance());
    board.collectAction(one,  action: "Collect £500");
    assertEquals( expected: balance + 600, one.getBalance());
}
```

Figure 7.10: Unit Test: Collect Action Card

Tests for: F2
When a player draws a card with action "pay", money should be taken from their balance by the amount specified by the action. There is a further test to make sure the player complete the action if they do not have the funds to do so.

```
@Test
public void payAction() {
    Player one = board.getAllPlayers().get(0);
    int balance = one.getBalance();

    assertTrue(board.payAction(one,  action: "Pay £1000"));
    assertEquals( expected: balance - 1000, one.getBalance());
    assertFalse(board.payAction(one,  action: "Pay £1000"));
    assertEquals( expected: balance - 1000, one.getBalance());
}
```

Figure 7.11: Unit Test: Move Action Card

Tests for: F2

When a player picks a card with action "move", their player should move to square named in the action. Furthermore, if they pass "GO" whilst carrying out the action, they player should receive £200. The player's position is set to 39, which is the last square before "GO" (Turing Heights). The action names "Crapper Street", so the player should move to the position of Crapper Street and as they do so, they pass "GO". Hence, the player should also receive £200.

The reason for "crapperStreet.getPosition()-1" is because the index of an array, which holds all the square of the board, starts at 0. The position of a square object is board data file, where the position of squares start from 1. The "-1" is just an offset.

```
@Test
public void moveAction() {
    Player one = board.getAllPlayers().get(0);
    Square crapperStreet = board.getAllSquares().get(1);
    one.setPosition(39);
    int balance = one.getBalance();

    assertTrue(board.moveAction(one,  action: "Move to Crapper Street"));
    assertEquals( expected: balance + 200, one.getBalance());
    assertEquals( expected: crapperStreet.getPosition() - 1, one.getPosition());
}
```

Figure 7.12: Unit Test: Move Action Card

Tests for: F2
When a player draws a card with action "fine", money should be taken from their balance by the amount specified by the action. The amount of money on "Free Parking" should increase by the amount paid.

```
@Test
public void fineAction() {
    Player one = board.getAllPlayers().get(0);
    int balance = one.getBalance();
    int freeParking = board.getFreeParking();

    assertTrue(board.fineAction(one,  action: "Fine £400"));
    assertEquals( expected: balance - 400, one.getBalance());
    assertEquals( expected: freeParking + 400, board.getFreeParking());
}
```

Figure 7.13: Unit Test: Fine Action Card

Tests for: F2
When a player draws a card with action "back", the player should move backwards (anti-clockwise) by the number of squares specified in the action. The edge case is when, whilst moving backwards, the player passed "GO" (position 0), in which case the position should be recalibrated from 40. The first part tests the edge case.

```
@Test
public void backAction() {
    Player one = board.getAllPlayers().get(0);
    int start = one.getPosition();
    board.backAction(one,  action: "back 3");

    assertEquals( expected: 37, one.getPosition());
    one.setPosition(4);
    board.backAction(one,  action: "back 2");
    assertEquals( expected: 2, one.getPosition());
}
```

Figure 7.14: Unit Test: Back Action Card

Tests for: T8
A player can only mortgage a property if they have enough money in their balance. The cost of mortgaging a property should be half the original price of the square. After mortgaging a square, the "mortgaged" field of the square should return true. The mortgaged property should set the "mortgaged" field to false and sell for half the original price of the square.

```java
@Test
public void mortgageProperty() {
    Player one = board.getAllPlayers().get(0);
    Square crapperStreet = board.getAllSquares().get(1);
    Square gangstersParadise = board.getAllSquares().get(3);
    int balance = one.getBalance();

    assertTrue(one.mortgageProperty(crapperStreet));
    assertTrue(crapperStreet.getMortgaged());
    assertEquals( expected: balance - crapperStreet.getCost() / 2, one.getBalance());
    assertEquals(crapperStreet.getOwnedBy(), one);

    assertTrue(one.sellProperty(crapperStreet));
    assertFalse(crapperStreet.getMortgaged());
    assertEquals(balance, one.getBalance());

    one.setBalance(0);
    assertFalse(one.mortgageProperty(gangstersParadise));
}
```

Figure 7.15: Unit Test: Mortgage Property

Tests for: T10
A player can pay off the mortgage of a property for the remaining half of the original price of the square. After paying off the mortgage, the "mortgaged" field of the square should be false.

```java
@Test
public void payMortgage() {
    Player one = board.getAllPlayers().get(0);
    Square crapperStreet = board.getAllSquares().get(1);
    int balance = one.getBalance();

    assertTrue(one.mortgageProperty(crapperStreet));
    assertTrue(crapperStreet.getMortgaged());
    assertEquals( expected: balance - crapperStreet.getCost() / 2, one.getBalance());
    assertTrue(one.payMortgage(crapperStreet));
    assertFalse(crapperStreet.getMortgaged());
    assertEquals(crapperStreet.getOwnedBy(), one);
    assertEquals( expected: balance - crapperStreet.getCost(), one.getBalance());
}
```

Figure 7.16: Unit Test: Pay Mortgage

Tests for: T5
There are two squares belonging to the "Brown" group, Crapper Street and Gangster's Paradise. After buying Crapper Street, "noImprovements" should

return false as the player does not own all squares belonging to the "Brown" group (Gangster's Paradise). "noImprovements" checks whether the player owns all properties with the colour group of the square in the game and that there are no houses/hotels on those properties. After buying a house on Crapper Street, it should return false since there are now houses on properties belonging to the "Brown" group. Selling the house of Crapper Street should result in true since both Crapper Street and Gangster's Paradise have no houses/hotels on them.

```
@Test
public void noImprovements() {
    Player one = board.getAllPlayers().get(0);
    Square crapperStreet = board.getAllSquares().get(1);
    Square gangstersParadise = board.getAllSquares().get(3);

    assertTrue(one.buyProperty(crapperStreet));
    assertFalse(board.noImprovements(one, crapperStreet));
    assertTrue(one.buyProperty(gangstersParadise));
    assertTrue(board.noImprovements(one, crapperStreet));
    assertTrue(board.noImprovements(one, gangstersParadise));
    assertTrue(one.buyHouse(crapperStreet));
    assertFalse(board.noImprovements(one, crapperStreet));
    assertFalse(board.noImprovements(one, gangstersParadise));
    assertTrue(one.sellHouse(crapperStreet));
    assertTrue(board.noImprovements(one, crapperStreet));
    assertTrue(board.noImprovements(one, gangstersParadise));
}
```

Figure 7.17: Unit Test: No Improvements

7.11.2 System Level

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
1	User presses "Status" button.	A button click event on "Status".	Along with a list of properties each player owns, there should be a ListView object that displays every buyable square remaining on the board.	Shows the list of properties each player owns and there is a ListView object containing every buyable square remaining in the square.	PASS
2	User presses "Pay off Mortgage".	A button click event on "Pay off Mortgage".	There should be a ListView object containing all the properties the player owns that are under mortgage. There should be a single button called "Buy".	There is a ListView object containing all the properties the player owns that are under mortgage. There is a single button called "Buy".	PASS
3	User selects a property from the list of mortgaged properties.	A button click event on the ListView object containing mortgaged properties.	There should be a label that names the selected property's name and cost of paying off the mortgage.	There is a label that names the selected property's name and cost of paying off the mortgage.	PASS
4	User presses "Buy" in "Pay off Mortgages" window	A button click event on "Buy" with no property selected.	There should be a label instructing the player to select a property before clicking "Buy".	There is a label instructing the player to select a property before clicking "Buy".	PASS
5	User presses "Buy" in "Pay off Mortgages" window	A button click event on "Buy" with a property selected and the player has enough to pay the mortgage.	Transaction should be accepted and the ListView object should be updated to remove the recently paid off property. There should be a label informing the success of the transaction.	Transaction is accepted and the recently paid off property is removed from the ListView object. There is a label telling the player about the success of the transaction.	PASS
6	User presses "Buy" in "Pay off Mortgages" window	A button click event on "Buy" with a property selected but the player does not have enough money to pay the mortgage.	Transaction should be declined, citing insufficient funds as the cause in a label.	Transaction is declined, citing insufficient funds as the cause in a label.	PASS
7	User is sent to jail.	An event, such as landing on "Go to Jail", occurs and the user is sent to jail. The user has enough money to pay the bail.	A window should open offering the player to pay a bail of £50. This window should have two buttons called "Yes" and "No". The player's token should move to the jail.	A window opens offering the player to pay a bail of £50. The window has two buttons called "Yes" and "No". The player's token moves to the jail.	PASS
8	User is sent to jail.	An event, such as landing on "Go to Jail", occurs but the user does not have enough money to pay bail.	The player should be jailed for the next two turns.	The player is jailed for the next two turns.	PASS
9	User presses "Yes" in "Pay Bail?" window.	A button click event on "Yes" and the user has enough to pay bail.	The "Pay Bail?" window should close and £50 should be taken from the player. The player should be released from prison.	"Pay Bail?" window closes and £50 is taken from the player's balance. The player is released from prison.	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
10	User presses "No" in "Pay Bail?" window.	A button click event on "No".	The "Pay Bail?" window should close and the user remains in jail for the next two rounds.	The "Pay Bail?" window closes and the user remains in jail for the next two rounds.	PASS
11	User presses "Roll Dice".	A button click event on "Roll Dice" and landing on a property that is owned by a player that is currently jailed.	The player should not pay the rent of the square.	The player does not pay the rent of the square.	PASS
12	User presses "Roll Dice".	A button click event on "Roll Dice" and landing on a property that is under mortgage.	The player should not pay the rent of the square.	The player does not pay the rent of the square.	PASS
13	User presses "Roll Dice".	A button click event on "Roll Dice" and the player is offered to buy the property they landed on.	A window offering the player to buy the property should open. This window should include a button called "Mortgage".	A window offering the player to buy the property opens. The window includes a button called "Mortgage".	PASS
14	User presses "Mortgage" and then "Confirm Choice" in "Buy Property?" window.	A button click event on "Mortgage" and the player has enough to pay the mortgage.	The player should mortgage the property for half the original price. The "Buy Property?" window should then close.	The player mortgages the property for half the original price. The "Buy Property?" closes.	PASS
15	User presses "Mortgage" and then "Confirm "Choice" in "Buy Property?" window.	A button click event on "Mortgage" and the player does not have enough money	The player should not be allowed to mortgage the property and the window should close.	The player did not mortgage the property and the window closes.	PASS
16	A property is not bought or mortgaged when given the chance.	A button click event on "Confirm Choice" where the property is not bought or mortgaged.	A new window called "Auction" should open. All players who have passed "GO" should have a label with their ID. Under this label, there should be a PasswordField to enter bids.	A new window called "Auction" opens. Every player who has passed "GO" has a label with their ID. Under these labels, there is a PasswordField for players to enter bids.	PASS
17	User presses "Confirm Bids" in "Auction" window.	A button click event on "Confirm Bids" where no bids are entered.	The "Auction" window should close. The property up for sale should remain unsold.	The "Auction" window closes and the property up for sale remains unsold.	PASS
18	User presses "Confirm Bids" in "Auction" window.	A button click event on "Confirm Bids" where there is a single highest bid, and the player making that bid has enough money in their balance to pay the bid.	The "Auction" window should close. The property should be sold to the player who made the highest bid. The property is bought for the amount placed on the bid.	The "Auction" window closes. The property is sold to the player who made the highest bid. The property is bought for the amount placed in the bid.	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
19	User presses "Confirm Bids" in "Auction" window.	A button click event on "Confirm Bids" where players have placed bids that they cannot afford.	The "Auction" window should close. Player's who make bids that they cannot afford to pay have their bids discarded. The property is sold to the highest legitimate bid, if any.	The "Auction" window closes. The property is sold to the highest legitimate bid. In cases where no legitimate bids are made, the property remains unsold.	PASS
20	User presses "Confirm Bids" in "Auction" window.	A button click event on "Confirm Bids" where multiple players have placed the highest legitimate bid.	The auction should be restarted by clearing entries in PasswordField objects.	The auction is restarted and all entries in PasswordField objects are cleared.	PASS
21	User entering bids.	A PasswordField listener action where non-digit characters are pressed.	The PasswordField should not allow any non-digit characters that are being entered. Only digits are allowed to be entered.	PasswordField does not allow any non-digit characters. Only digits are allowed to be entered.	PASS
22	User presses "Roll Dice".	Player lands on either "Income Tax" or "Super Tax".	The player should pay the tax specified under the "Action" column of the board data file.	The player pays the correct amount of money with respect to the number given in under the "Action" column of the board data file.	PASS
23	User presses "Roll Dice".	Player does not have enough money in their balance to pay off the required amount but their net worth is enough to pay the required amount.	A window called "Pay Funds" should open. This window should have two buttons called "Buy" and "Sell". There should be a list of all the properties the player owns. There should be a label showing the amount of money that needs to be raised by selling assets.	A window called "Pay Funds" opens. The window contains two buttons called "Buy" and "Sell". There is a list of all the properties the player owns. There is a label showing the amount of money that needs to be raised by selling assets.	PASS
24	User presses "Sell" in "Pay Funds" window.	A button click event on "Sell".	The amount of money to be raised should be updated. If the player has raised enough funds, the "Pay Funds" window should close and the player should pay the amount due.	The amount of money to be raised is updated. When the player has raised enough funds, the "Pay Funds" window closes. The player pays the amount due.	PASS
25	User presses "Buy" in "Pay Funds" window.	A button click event on "Buy".	The amount of money to be raised should be updated.	The amount of money to be raised is updated.	PASS
26	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on "Go to Jail" and has a "Get out of jail free" card.	The player should move to the jail. The player should then use up their "Get out of jail free" card and be released from jail.	The player is moved to the jail. The player uses up their "Get out of jail free" card and are released from jail.	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
27	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on a property owned by a different player. The player who owns this property owns all other properties of the same colour group and has not bought house/hotels on these properties.	The player should pay double the unimproved rent rate of the property.	The player pays double the unimproved rent rate of the property.	PASS
28	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on either Pot Luck or Opportunity Knocks and draws a card with action "Collect".	The player should collect the amount of money specified in the card.	The player collects the amount of money specified in the card.	PASS
29	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on either Pot Luck or Opportunity Knocks and draws a card with action "Pay".	The player should pay the amount specified in the card if they have enough in the balance. Otherwise, if their net worth is enough, then "Pay Funds" window should open. Otherwise, the player should be declared bankrupt.	The player pays the amount specified in the card to the bank if they have enough in the balance. Otherwise, if their net worth is enough, the "Pay Funds" window opens. Otherwise the player is declared bankrupt.	PASS
30	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on either Pot Luck or Opportunity Knocks and draws a card with action "Fine".	The player should pay the amount specified in the card if they have enough in the balance. The amount paid should be added to free parking. Otherwise, if their net worth is enough, then "Pay Funds" window should open. Otherwise, the player should be declared bankrupt.	The player pays the amount specified in the card to the bank if they have enough in the balance. The amount paid is added to free parking. Otherwise, if their net worth is enough, then the "Pay Funds" window opens. Otherwise the player is declared bankrupt.	PASS
31	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on either Pot Luck or Opportunity Knocks and draws a card with action "Move".	The player should move to the position of the named square specified in the card. If they pass "GO", the player should collect £200.	The player moves to the position of the named square specified in the card. If they pass "GO", the player collects £200.	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
32	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on either Pot Luck or Opportunity Knocks and draws a card with action "Jail".	The player should be sent to jail and the "Pay Bail?" window should open if they have enough money in their balance to pay the bail. Otherwise, they should remain in jail.	The player is sent to jail and the "Pay Bail?" window opens if they have enough money in their balance to pay the bail. Otherwise, they remain in jail.	PASS
33	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on either Pot Luck or Opportunity Knocks and draws a card with action "Free".	The player should obtain a "Get out of free" jail card.	The player obtains a "Get out of free" jail card.	PASS
34	User is sent to jail	An event, such as landing on "Go to Jail", occurs and the player has a "Get out of free" jail card.	The player should use up their "Get out of free" jail card and be freed from prison.	The player uses up their "Get out of free" jail card and is freed from prison.	PASS
35	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on either Pot Luck or Opportunity Knocks and draws a card with action "Back".	The player should move anti-clockwise by the number of squares specified in the card.	The player moves anti-clockwise by the number of squares specified in the card.	PASS
36	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on either Pot Luck or Opportunity Knocks and draws a card with action "Repair".	The player should pay the rate specified in the card if they have enough money in their balance. Otherwise, if their net worth is enough to pay rent then the "Pay Funds" window should open. Otherwise, the player should be declared bankrupt.	The player pays the rate specified in the card if they have enough money in their balance. Otherwise, if their net worth is enough to pay rent, then the "Pay Funds" window opens. Otherwise, the player is declared bankrupt.	PASS
37	User presses "Roll Dice".	A button click event on "Roll Dice" where the player lands on either Pot Luck or Opportunity Knocks and draws a card with action "Receive".	The player should receive the amount specified in the card from every other player in the game.	The player receives the amount specified in the card from every other player in the game.	PASS

Test Number	Testing For
1	T17
2	T10, F7, NF2
3	T10, F7
4	F7
5	T10, F7
6	F7
7	T3, T7, F4
8	T3, F4
9	T3, F4
10	T3, F4
11	T6
12	T9
13	T8, F10
14	T8, F10
15	T8, F10
16	T15, F8, NF1, NF3
17	F8
18	F8
19	F8
20	F8
21	NF5
22	T13
23	T16, F5
24	T16, F5
25	F5
26	T14
27	T5
28	T2, F1, F2
29	T2, F1, F2, F5, F9
30	T2, T12, F2, F5, F9
31	T2, F1, F2, F5, F9
32	T2, T3, F1, F2
33	T2, F2
34	T14
35	T2, F1, F2
36	T2, F1, F2, F5, F9
37	T2, F1, F2, F5, F9

Figure 7.18: Fourth Prototype: Board View

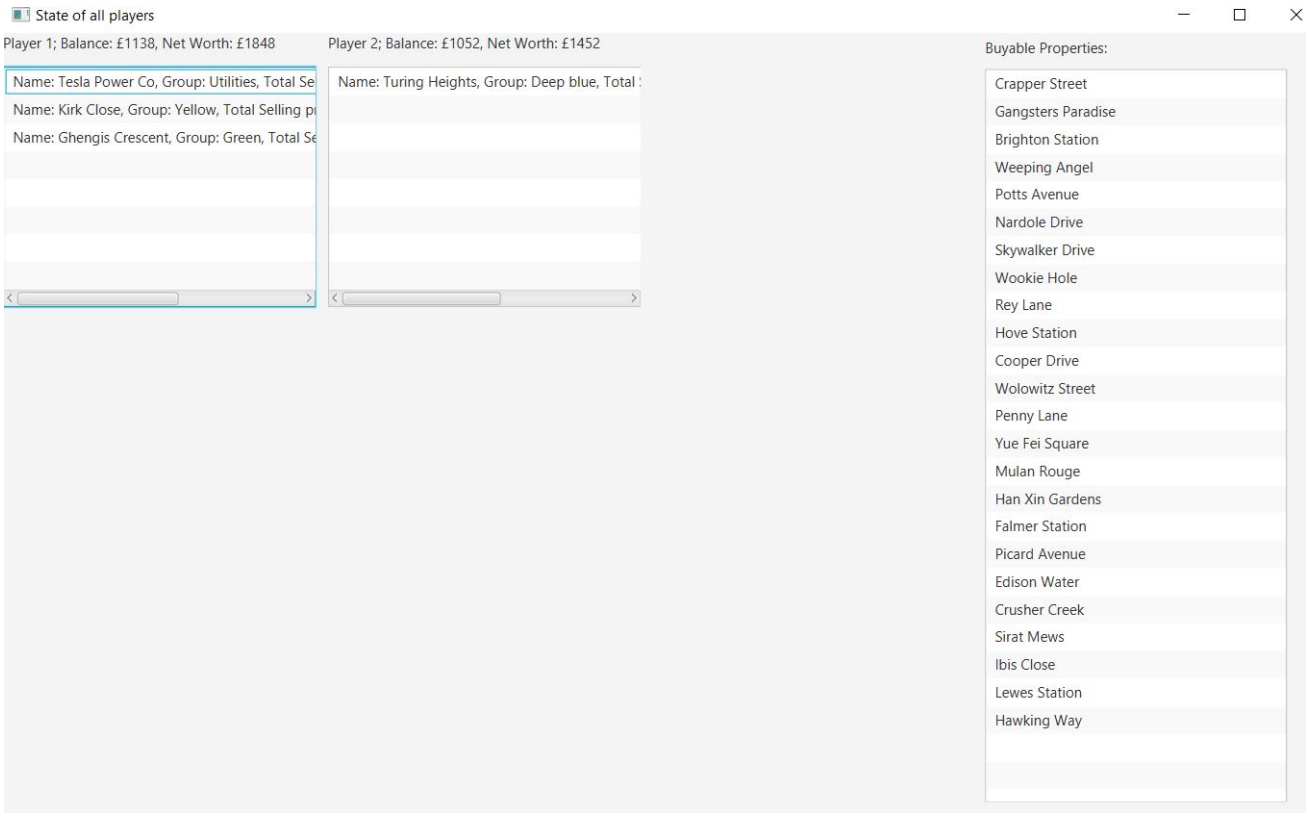


Figure 7.19: Fourth Prototype: Status View

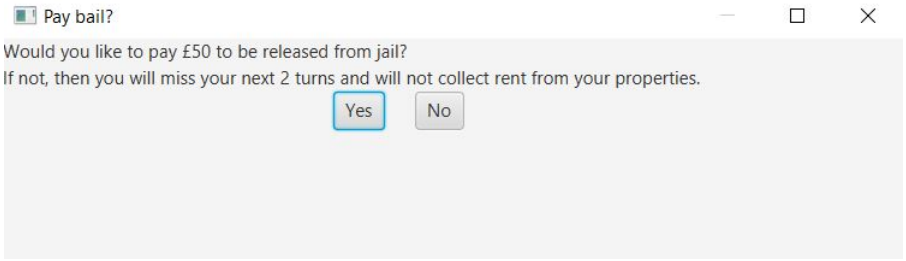


Figure 7.20: Fourth Prototype: Pay Bail View

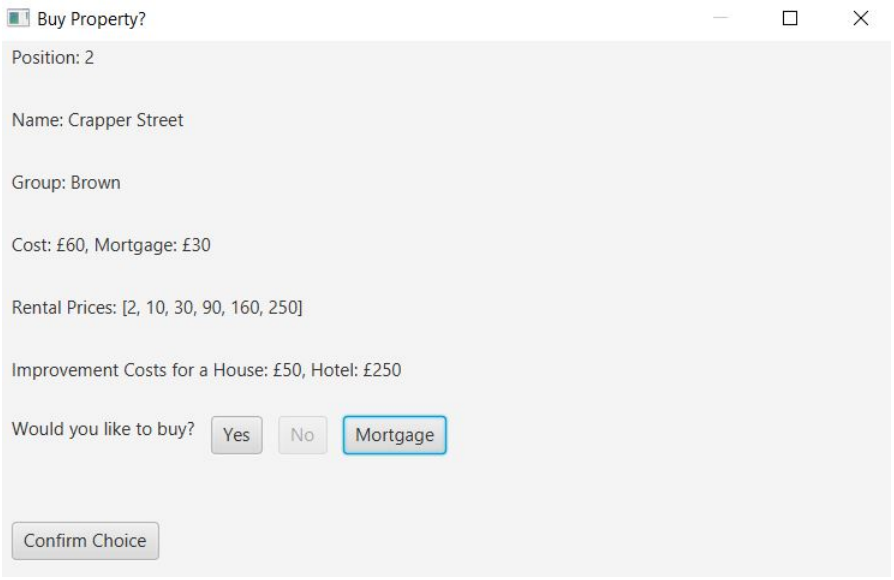


Figure 7.21: Fourth Prototype: Buy Property View

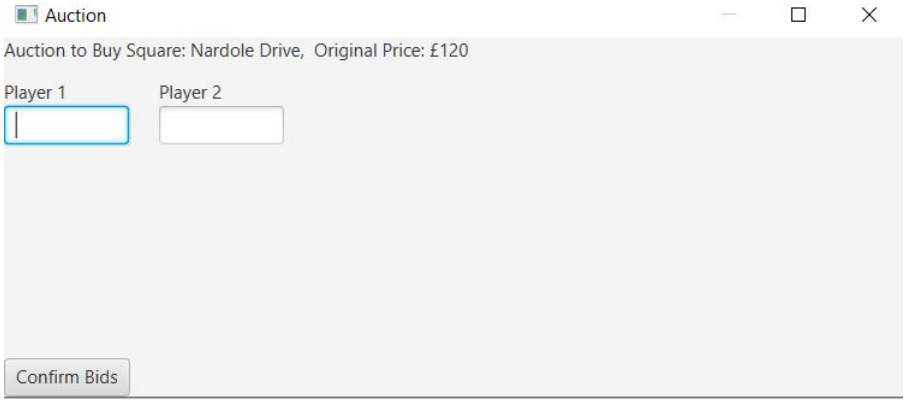


Figure 7.22: Fourth Prototype: Auction View

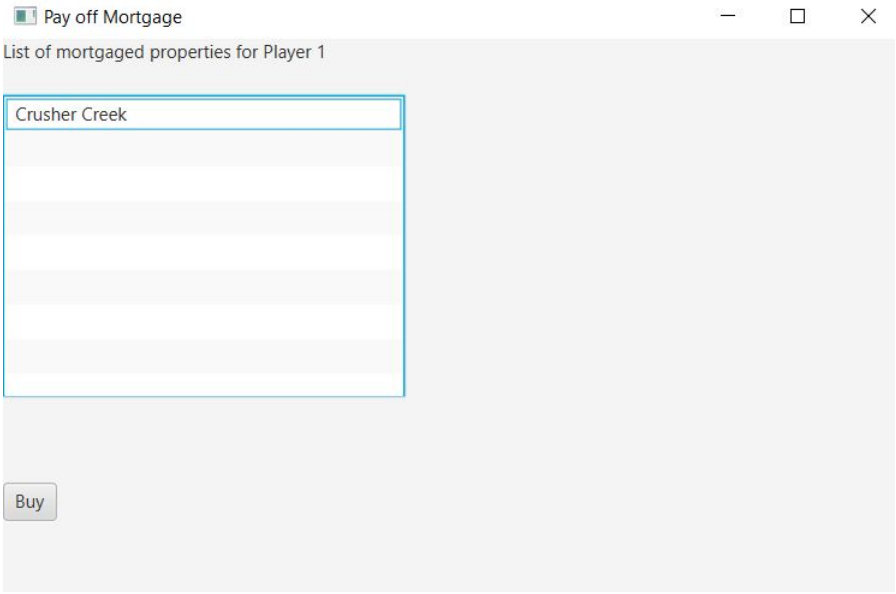


Figure 7.23: Fourth Prototype: Pay Mortgage View

7.12 Reflection

The main focus of this sprint was to implement the card actions, which have been accomplished by creating a set of actions that a card can have. We went further by designing the actions in a way that allows the user to customize the values of cards, such as changing the name of the square to move to, or the amount of money to pay. To add on, we implemented the option to mortgage properties and then pay off the mortgages. We also managed to complete features that were optional for this sprint, such as the auctioning system.

The main task for next sprint is to implement the two games modes, "Full Game" and "Abridged Game". The "Full Game" has no time limit and the game is over when one player remains. In the "Abridged" game mode, the game is over once the timer, specified by the user in the menu, hits 0 and all players have taken the same number of turns. The secondary task is to implement the Game Player Agent. This player agent will perform random decision making.

The feedback from the customer was positive and they advised us to make the game player agent be based upon random choices. How we implement these choices is up to us. Furthermore, whilst expanding the menu during next sprint, the customer has advised us to disallow the user from creating the game if the configurations are not allowed, such as having less than 2

players, etc.

Note that the team took a week break during 3rd April and 10th April. This is because some members are travelling back to their home country. However, as we are ahead of schedule, we took a week break from the project to let everyone get settled down before we continue onwards to the final sprint.

The code for the first prototype can be found in the submission folder, called "FourthPrototype".

7.13 Individual Key Contributions

Test Member	Key Contribution(s)
Jun	T1, T7, T9, System Testing
Jihye	T1, T5, T11, System Testing
Ye-Rang	T3, T8, T12, System Testing
Ibi	T2, T3, T4, T10, T13, T15, T15, T16, T17, Unit Testing, System Testing

Sprint 5

Team Number: 32
Sprint Technical Lead: Jun Baek
Sprint start date: 10th April 2020
Sprint end date: 24th April 2020

8.1 User Stories

This final sprint will produce the fifth prototype. The main feature of this prototype will be the addition of two game modes, "Full Game" and "Abridged Game", and the game player agent.

When first starting up the game, the menu should allow the user to choose the number of game player agents to include in the game. There should also be an option to select one of the two aforementioned game modes. If the game mode is "Abridged Game", then the user should be able to set a time limit for the game. When the time limit is reached and all players have taken the same number of turns, the game ends. The winner of the game is decided by their total net worth, which includes the player's balance, value of properties and value of houses/hotels on those properties.

The user should not be able to create a game if the options selected in the menu are not allowed. These include having at least 2 users playing the game and a maximum of 6. If the game mode is "Abridged Game", then the user should specify a valid time configuration.

The game player agent should make a random choice whenever it is faced with a decision. The game player agent should function automatically with no decision needed to be made by the user.

Finally, a player should have the option to leave the game. The game player agent should not be able to leave the game. Whenever a player voluntarily leaves the game, all their assets are sold back to the bank.

8.2 Task Cards

Scoring scale: 1 to 10, with 10 having the highest priority and 1 the lowest. A task card with priority 10 means that it must be completed this sprint, otherwise the prototype will not work or make enough progress from the previous prototype. Alternatively, a 10 can also be a feature that the customer has recommended. A card with priority 1 means that it does not have to be completed this sprint, and without it, it will not impact the current prototype.

1. Priority: 10, Complexity: 5

   Add the option to select one of the two game modes, "Full Game" or "Abridged Game", in the main menu.

2. Priority: 10, Complexity: 5

   Add the option to select the number of game agents to participate in the game.

3. Priority: 6, Complexity: 3

   If the user tries to create a game with incorrect menu configurations, then let them know what's wrong with the configurations and do not create the game.

4. Priority: 10, Complexity: 4

   When the game ends, the user shall be taken to a screen congratulating the winner(s) of the game, along with a button that takes the user back to the main menu.

5. Priority: 10, Complexity: 5

   If the game mode is "Full Game", then the game is over when there is 1 player remaining.

6. Priority: 8, Complexity: 8

   If the game mode is "Abridged Game", then the game is over when the timer, set by the user, hits 0 and all players have taken the same number of turns.

7. Priority: 8, Complexity: 1

   The winner(s) of the game in the "Abridged Game" mode is decided by their net worth. The net worth includes the sum of the player's

balance, value of properties and the value of houses/hotels on those properties.

8. Priority: 3, Complexity: 1

   The time limit of the game shall be entered in minutes and then converted into seconds for the board.

9. Priority: 7, Complexity: 9

   If the game mode is "Abridged Game" then the amount of time remaining shall be visible in minutes and seconds.

10. Priority: 5, Complexity: 3

    The player shall have a button that allows them to leave the game. Game Agents cannot use this button. When the player leaves the game, all their assets are sold to the bank, which includes all of the properties they own.

11. Priority: 9, Complexity: 4

    Whenever the Game Agent is faced with a decision, such as buying a property, a random choice is made on whether they want to go ahead with the action.

12. Priority: 6, Complexity: 5

    Each turn, the Game Agent will look through every property it owns, making a choice on whether it wants to buy a house/hotel on that property.

13. Priority: 7, Complexity: 7

    Whenever the Game Agent is unable to pay a required sum of money, but their net worth is enough is enough to pay the sum, then the game agent sells assets until their balance can pay the required sum.

14. Priority: 5, Complexity: 7

    Whenever the Game Agent participates in an auction, they make a choice to enter a bid or not. The game agent uses a random percentage of their balance to place a bid.

15. Priority: 10, Complexity: 3

    The user should not have to make any decisions on behalf of the Game Agent during its turn.

8.3 Requirements Analysis

8.3.1 Functional Requirements

F1 - Mandatory

A valid game configuration shall follow every rule listed below:

- The total number of players cannot be below 2.
- The total number of players cannot be above 6.
- If the game mode is "Abridged Game", the minutes specified cannot be 0.
- If the game mode is "Abridged Game", the minutes specified cannot be empty.
- If the game mode is "Full Game", the minutes specified must be empty.

F2 - Desirable

The user can only enter digits when specifying how the time limit of the game.

F3 - Mandatory

In the "Abridge Game" mode, when the timer hits 0 and all players have already taken the same number of turns, then the game shall end automatically. If when the timer hits 0 and players have not all taken the same number of turns, then the game shall end once the last player has ended their turn.

F4 - Desirable

When the game ends, the board view should close and a new window congratulating the winner(s) should open. This window should list the winning player(s) along with the net worth of the player(s). There should be a single button called "Go to Main Menu" that when clicked opens the "Main Menu" window.

F5 - Mandatory

Whenever a Game Agent is participating in the auction, their PasswordField objects shall be disabled to prevent players entering bids for them. If the Game Agent decides to place a bid, then it is automatically entered in its corresponding PasswordField object. The PasswordField

object of the Game Agent remains empty if it decides not to place a bid. The same process applies if the auction is restarted.

8.3.2 Non-Functional Requirements

NF1 - Mandatory

The user shall select the number of Game Agents to be in the game from a JavaFX ChoiceBox[17] object. This object shall hold integers from 0-6.

NF2 - Mandatory

The user shall enter the time limit of the game in a JavaFX TextField object.

NF3 - Mandatory

The user shall select the game mode of the game from a JavaFX ChoiceBox. This object shall hold the strings "Full Game" and "Abridged Game".

NF4 - Mandatory

The time limit of the game, if any, shall be displayed using a JavaFX Label object.

8.3.3 Domain Requirements

D1 - Desirable

The game should not crash during the creation of the game from the main menu. The fewer ways the user can break the game, the better. Hence, see F1, F2 and F5.

8.4 Use Case Diagram

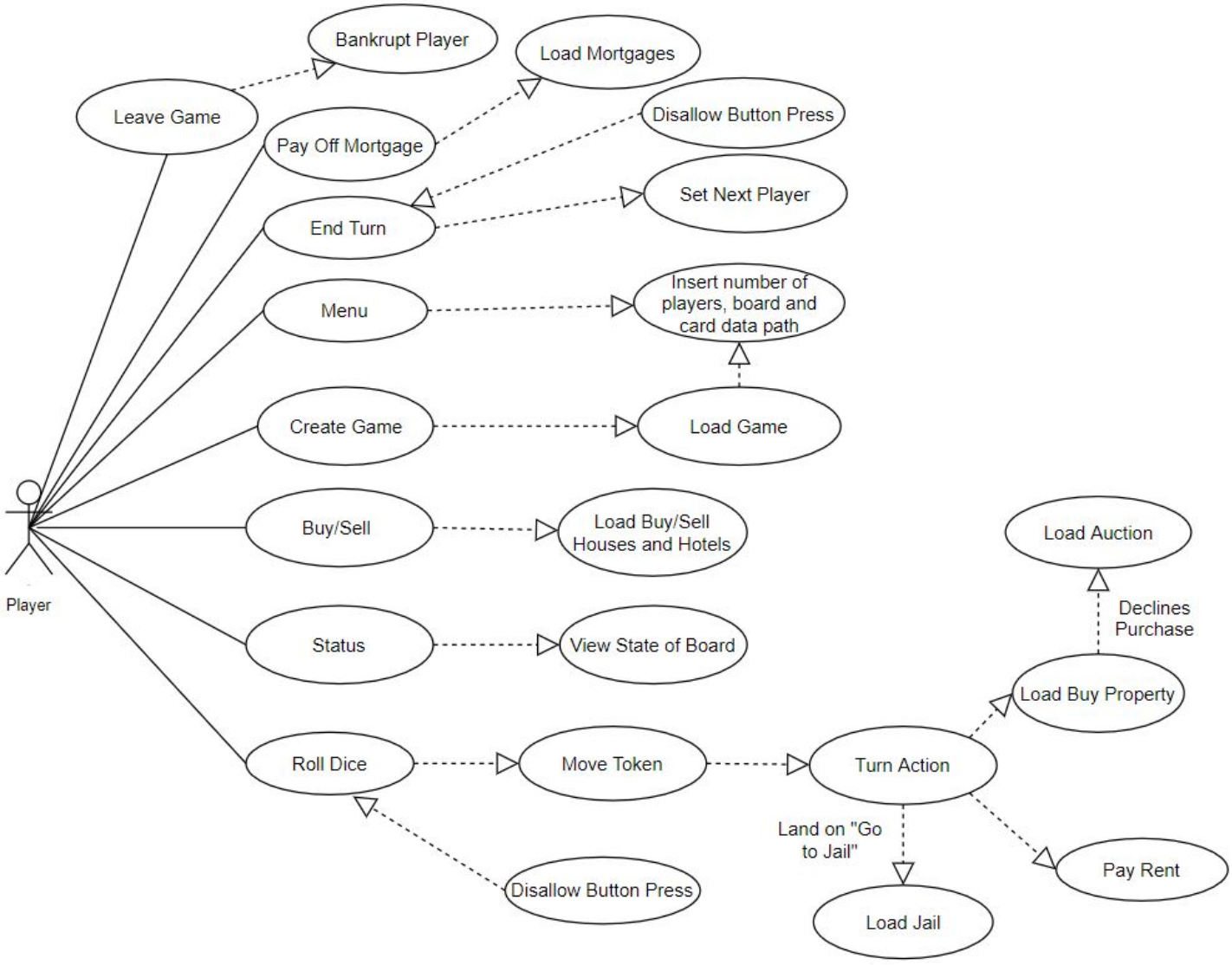


Figure 8.1: Fifth Prototype: Use Case Diagram

8.5 High Level Diagram

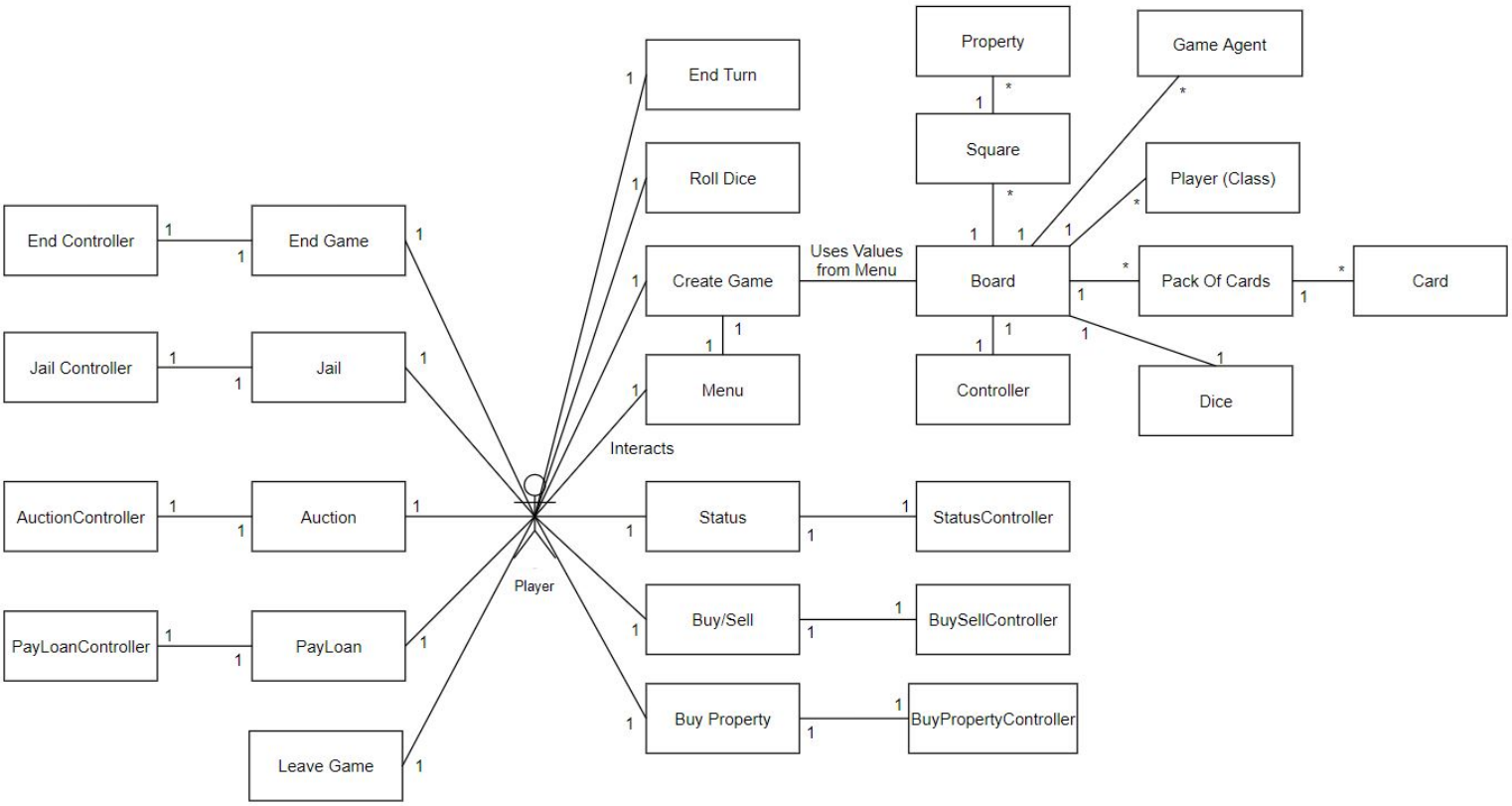


Figure 8.2: Fifth Prototype: High Level Diagram

8.6 Class Diagram

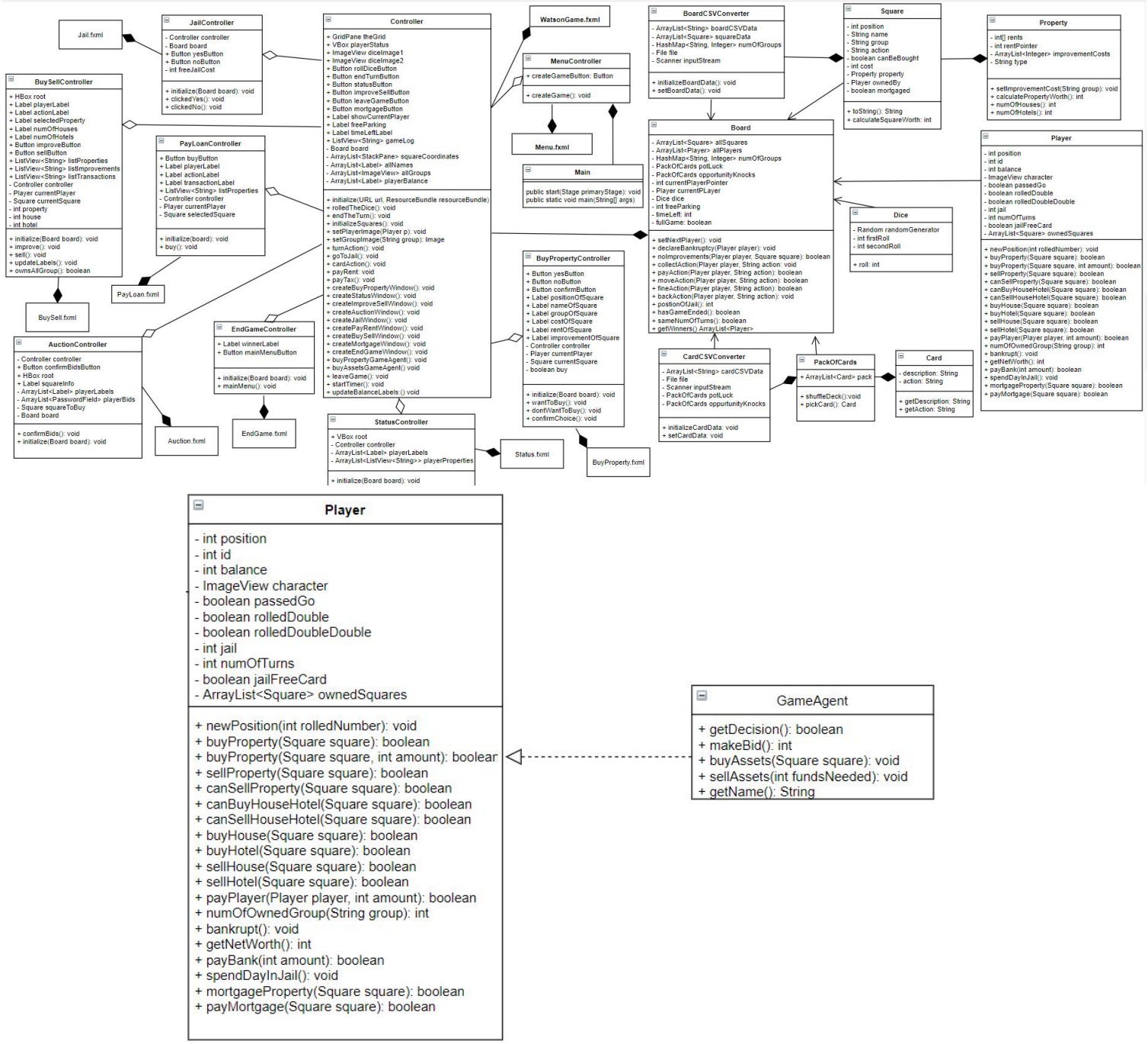


Figure 8.3: Fifth Prototype: Class Diagram

In order to keep the class diagram concise and presentable, getter and setter methods have been omitted with permission from the customer. Note that they are included in the code and used in sequence diagrams even though they do may not appear in the class diagram.

The class diagram can be viewed clearly in Draw.io. To open our class diagrams, please read chapter 1. The first class diagram is located in the "Class Diagrams" folder, called "FifthClassDiagram.io".

8.7 Sequence Diagrams

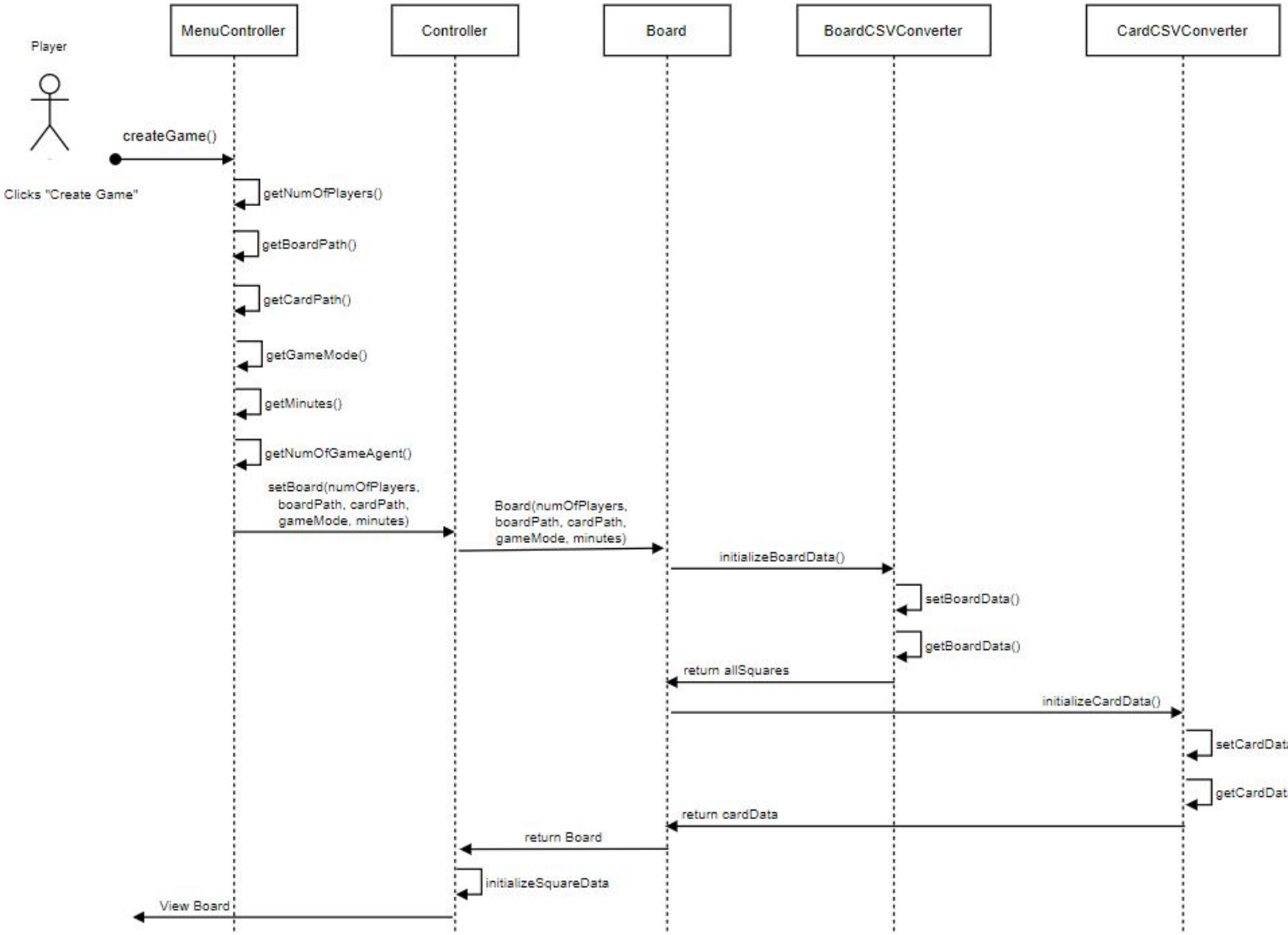


Figure 8.4: Sequence of actions for when "Create Game" button is clicked.

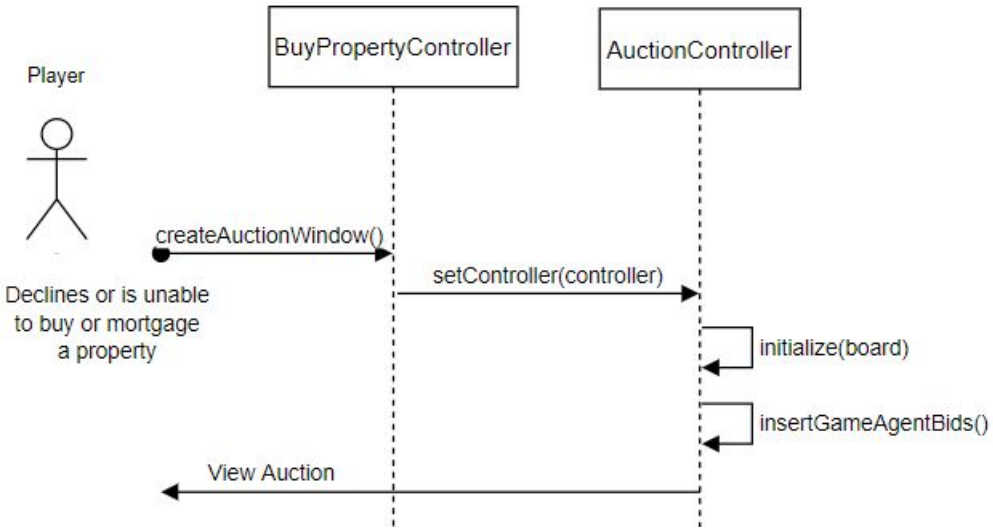


Figure 8.5: Sequence of actions for when an auction is started

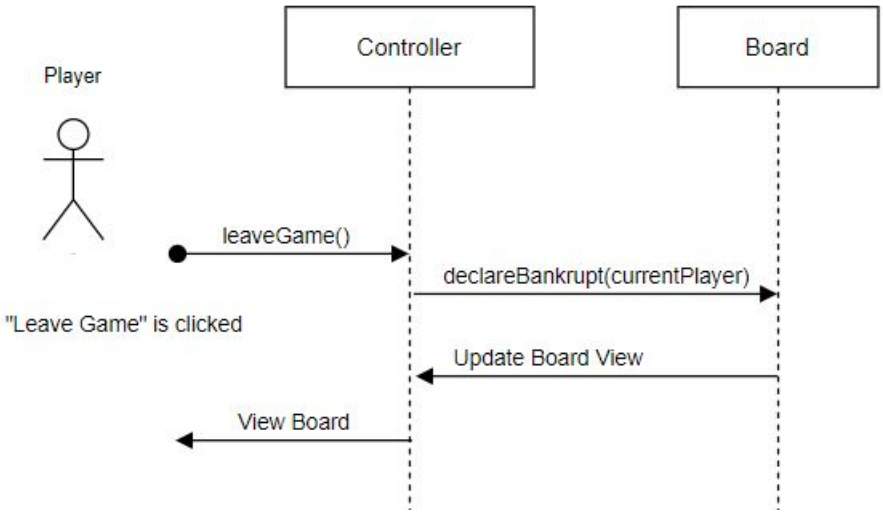


Figure 8.6: Sequence of actions for when "Leave Game" is clicked

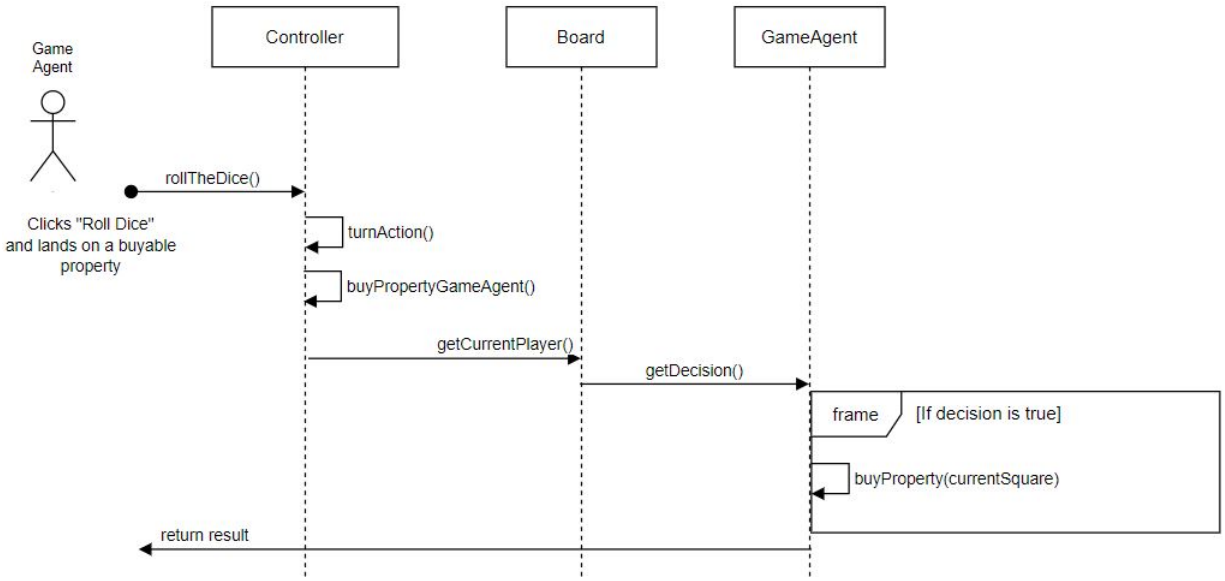


Figure 8.7: Sequence of actions for a game agent lands on a buyable property

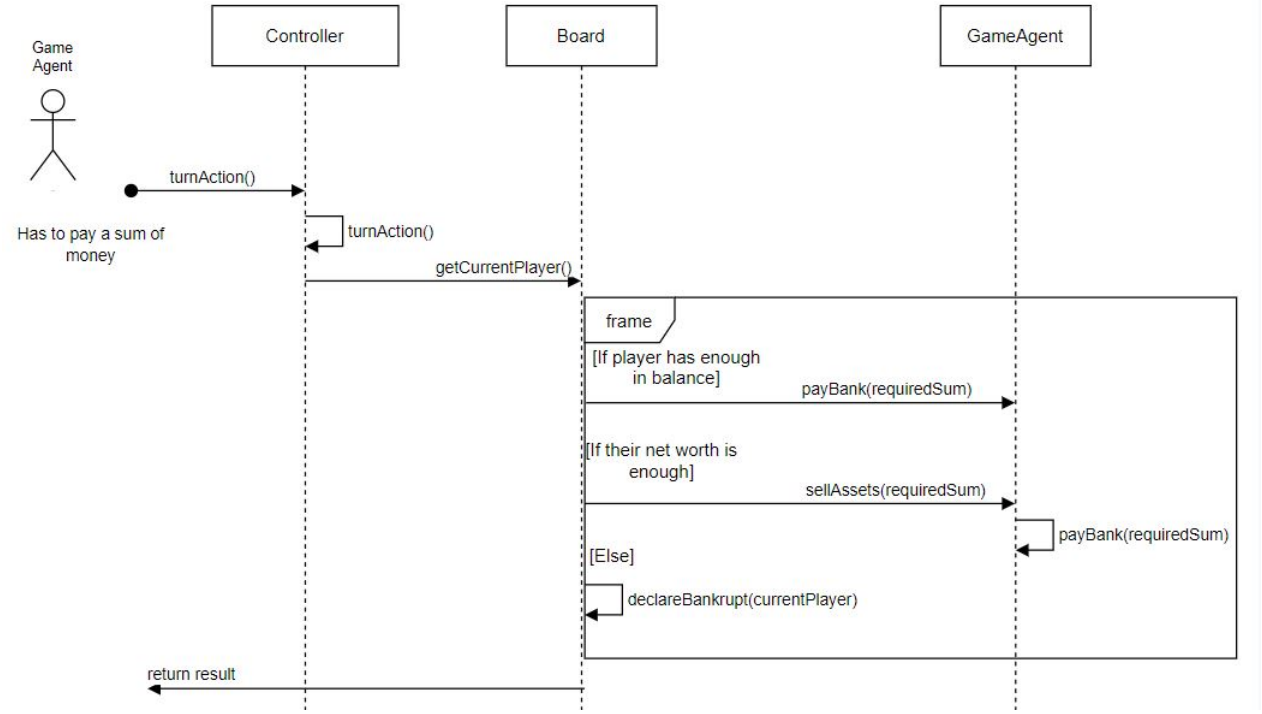
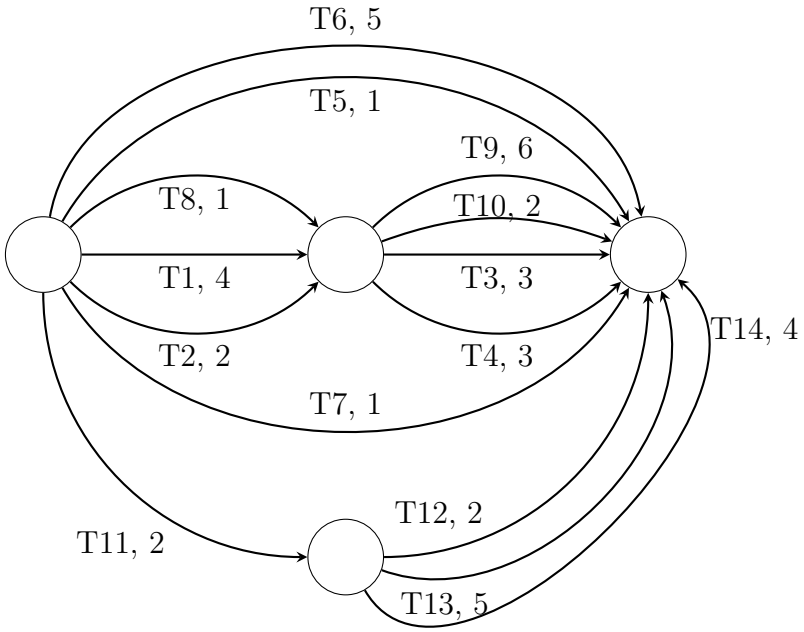


Figure 8.8: Sequence of actions for a game agent has to pay a required sum of money

8.8 PERT Chart



Task Card	Days				Critical Path
	Earliest Start	Earliest Finish	Latest Start	Latest Finish	
1	0	4	0	4	Yes
2	0	2	2	4	No
3	4	7	7	10	No
4	4	7	7	10	No
5	0	1	9	10	No
6	0	5	5	10	No
7	0	1	9	10	No
8	0	1	3	4	No
9	4	10	4	10	Yes
10	4	6	8	10	No
11	0	2	3	5	No
12	2	4	8	10	No
13	2	7	5	10	No
14	2	6	6	10	No

Figure 8.9: Fifth Prototype: PERT Chart

8.9 Risk Management

Risk Identification	Causes and Likelihood	Mitigation	Monitoring
Software Development may be delayed.	Likelihood: 1 Deadlines for other assessments coming up. As a result, focus of members may be elsewhere near these assessments.	Plan work to take into account future deadlines. Project Manager organises tasks with deadlines in mind.	Project manager discusses future tasks and assigns tasks to each member during team meetings.
Members may have trouble completing a coding task, which can slow progress.	Likelihood: 3 Technical Supports having difficulty with completing a task.	Technical Support should notify the Technical Director for assistance.	Technical Supports will ask for support, if needed, from the Technical Director, during team meetings.
Covid-19 is becoming an increasingly likely problem to cause disturbance.	Likelihood: 2 A member may travel back to their home country, or they may have family members affected by Covid-19.	Members should tell the Project Manager about any travel arrangements or problems in advance, so he can create a better plan and offload work to members not affected.	Simply send a message to the Project Manager should such problems arise. The team is more than willing to help affected members.
Members are travelling back to their home country due to Covid-19. As such, progress may be delayed.	Likelihood: 1 Members are travelling back to their home country. Understandably, no work should be done by these members for at least a week to help them reset.	Work will be off loaded to the Technical Director who has not been affected by Covid-19. After a week break for affected members, they will continue working on the project.	Technical Director will let members know about progress during the week off. As of now, we are ahead of schedule, so progress is looking good.
Members are in different countries and so team meetings are difficult to organize. This could affect organizing members to tasks and assessing progress.	Likelihood: 4 Members in different countries will be in different time zones. Technicality issues as we need a new method of communicating for team meetings. Messages alone do not suffice.	Use Discord to set up online communication and hold team meetings. Agreed on a meeting time of 12:30PM KST.	See the effectiveness of using Discord as a means of holding a meeting during the first session.

Likelihood scores are between 1-5, with 1 being highly unlikely and 5 being highly likely.

8.10 Implementation of Task Cards

The Game Agent functions like a probability tree where the root of the tree is a 50/50 split between true and false. If the result is false, then the tree ends. On the other hand, if the result is true, then the Game Agent carries on with the task. Below is an example of a simple probability tree when the Game Agent is offered to buy a property.

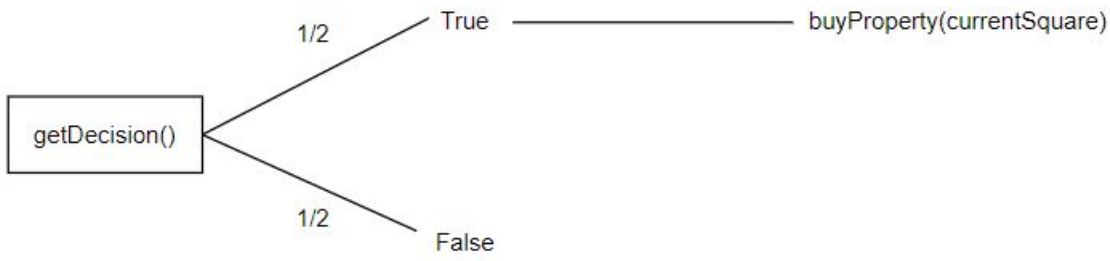


Figure 8.10: Game Agent Probability

This is only simple example. The probability trees expand when the task is to decide whether to buy a house/hotel on a property, for example. We decided to use a probability tree so there is a greater variety of outcomes the Game Agent can produce.

Whilst the concept of a timer is simple, displaying the timer proved to be difficult. This is because JavaFX GUI components run on different Threads, namely a JavaFX Application Thread. Therefore, we used the runLater method provided by the Platform[18] to update the change.

```
public void startTimer() {
    if (!board.getFullGame()) {
        Timer timer = new Timer();
        timer.scheduleAtFixedRate(new TimerTask() {
            int seconds = board.getTimeLeft();

            public void run() {
                if (seconds != 0) {
                    Platform.runLater(() -> timeLeftLabel.setText("Time Left: " + seconds / 60 + " minute " + seconds % 60 + " seconds"));
                    seconds--;
                    board.decrementTime();
                } else {
                    Platform.runLater(() -> timeLeftLabel.setText("Time is up! Waiting for all players to take equal number of turns"));
                    timer.cancel();
                    Platform.runLater(Controller.this::updateBalanceLabels);
                }
            }
        }, delay: 1000, period: 1000);
    }
}
```

Figure 8.11: Start Timer Method

8.11 Testing

8.11.1 Unit Level

Tests for: T13

When a Game Agent does not have enough money in their balance to pay a sum, but their net worth is enough, the Game Agent sell assets until they can pay off the fund. In the first instance, the Game Agent is asked to pay a sum of £1489. The Game Agent can only afford this amount if they sell Crapper Street.

After buying Skywalker Drive and building a house on it, the Game Agent's balance should be £1260. In the second instance, the Game Agent is asked to pay £1360. The Game Agent should only sell the house on Skywalker Drice, and not the property itself to afford the required funds.

```java
@Test
public void sellAssets() {
    GameAgent one = (GameAgent) board.getAllPlayers().get(0);
    Square crapperStreet = board.getAllSquares().get(1);
    Square skywalkerDrive = board.getAllSquares().get(11);
    int balance = one.getBalance();

    assertTrue(one.buyProperty(crapperStreet));
    assertEquals( expected: balance - crapperStreet.getCost(), one.getBalance());
    assertEquals(crapperStreet.getOwnedBy(), one);

    one.sellAssets( fundsNeeded: 1489);

    assertEquals(balance, one.getBalance());
    assertNull(crapperStreet.getOwnedBy());

    assertTrue(one.buyProperty(skywalkerDrive));
    assertEquals( expected: balance - skywalkerDrive.getCost(), one.getBalance());
    assertTrue(one.buyHouse(skywalkerDrive));
    assertEquals(skywalkerDrive.getProperty().numOfHouses(),  actual: 1);
    assertEquals( expected: 1260, one.getBalance());

    one.sellAssets( fundsNeeded: 1360);

    assertEquals(skywalkerDrive.getOwnedBy(), one);
    assertEquals(skywalkerDrive.getProperty().numOfHouses(),  actual: 0);
}
```

Figure 8.12: Unit Test: Sell Assets

Tests for: T14
The Game Agent should have enough money in their balance to afford the bid it places. Lower the balance of the Game Agent and check that the bid placed remains less than or equal to the its balance.

```java
@Test
public void makeBid() {
    GameAgent one = (GameAgent) board.getAllPlayers().get(0);
    assertTrue( condition: one.makeBid() <= one.getBalance());

    one.setBalance(1);
    assertTrue( condition: one.makeBid() <= one.getBalance());

    one.setBalance(0);
    assertTrue( condition: one.makeBid() <= one.getBalance());
}
```

Figure 8.13: Unit Test: Make Bid

8.11.2 System Level

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
1	User starts up the game.	Running main method in "Main.java".	There should be a ChoiceBox to select the number of players, the number of Game Agents and the game mode. There should be three TextField objects to enter the board and card data file, and the time limit of the game.	There is a ChoiceBox object to select the number of players, the number of Game Agents and the game mode. There is a TextField object to enter the board and card data file, and the time limit of the game.	PASS
2	User presses "Create Game" button.	A button click event on "Create Game" where the total number of players is less than 2.	Creating the game should be disallowed and there should be a label telling the user that too few players were selected.	Disallowed creation of game and there is a label telling the user that too few players were selected.	PASS
3	User presses "Create Game" button.	A button click event on "Create Game" where the total number of players is greater than 6.	Creating the game should be disallowed and there should be a label telling the user that too many players were selected.	Disallowed creation of game and there is a label telling the user that too many players were selected.	PASS
4	User presses "Create Game" button.	A button click event on "Create Game" where the game mode is "Abridge Game" and no time limit was entered.	Creating the game should be disallowed and there should be a label telling the user to enter a time limit.	Disallowed creation of game and there is a label telling the user to enter a time limit	PASS
5	User presses "Create Game" button.	A button click event on "Create Game" where the game mode is "Abridged Game" and the time limit equates to 0.	Creating the game should be disallowed and there should be a label telling the user to enter a non-zero time limit.	Disallowed creation of game and there is a label telling the user to enter a non-zero time limit.	PASS
6	Auction is started.	A property is not bought or mortgaged.	The auction window should open. All PasswordField for participating Game Agents should be disabled. Bids, if any, made by Game Agents should be entered automatically.	The auction window is opened. All PasswordField objects for participating Game Agents are disabled. Bids made by the Game Agents are entered automatically.	PASS
7	User clicks "Roll Dice" button.	A button click event on "Roll Dice" where the current player is a Game Agent.	The user should not have to make any decisions for the Game Agent. Game Agents should carry out the action of the turn automatically.	The user does not make any decisions for the Game Agent. Game Agents carries out the action of the turn automatically.	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
8	Game Ends	There is one player left where the game mode is "Full Game".	The board view should close and a new window called "Congratulations" should open. There should be label showing the player who won the game and the net worth of the player. There should be a button called "Go to Main Menu".	The board view closes and a new window called "Congratulations" opens. There is a label showing the player who won the game and the net worth of the player. There is a button called "Go to Main Menu".	PASS
9	Game Ends	There are multiple players left with joint highest net worth and all players have taken the same number of turns. The game mode is "Abridged Game" and the timer has hit 0.	The board view should close and a new window called "Congratulations" should open. There should be label showing the players who won the game and the net worth of the players. There should be a button called "Go to Main Menu".	The board view closes and a new window called "Congratulations" opens. There is a label showing the players who won the game and the net worth of the players. There is a button called "Go to Main Menu".	PASS
10	Game Ends	The game mode is "Abridged Game" and the timer has hit 0. Only one player holds the highest net worth.	The board view should close and a new window called "Congratulations" should open. There should be label showing the player who won the game and the net worth of the player. There should be a button called "Go to Main Menu".	The board view closes and a new window called "Congratulations" opens. There is a label showing the player who won the game and the net worth of the player. There is a button called "Go to Main Menu".	PASS
11	Game Ends	The game mode is "Abridged Game" and there is one player left and the time limit is not 0.	The board view should close and a new window called "Congratulations" should open. There should be label showing the player who won the game and the net worth of the player. There should be a button called "Go to Main Menu".	The board view closes and a new window called "Congratulations" opens. There is a label showing the player who won the game and the net worth of the player. There is a button called "Go to Main Menu".	PASS
12	User entering the length of the game	Enters non-digit characters.	All non-digit characters should not be inserted into the TextField object.	All non-digit characters were not inserted into the TextField object	PASS
13	The game mode is "Abridged"	User creates a game with the "Abridged Game" mode, specifying the number of minutes the game will last for.	At the top of the board view, the number of minutes and seconds left in the game should be visible. The time left should be updated every passing second.	At the top of the board view, the number of minutes and seconds left in the game is visible. The time left is updated every passing second.	PASS

Test Number	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
14	User clicks "Leave Game" button	Button click event on "Leave Game".	The current player's token should be removed from the game. All properties the player owned should now be owned by the bank.	The current player's token is removed from the game. All properties the player owned is now owned by the bank.	PASS

Test Number	Testing For
1	T1, T2, T8, NF1, NF2, NF3
2	T3, F1
3	T3, F1
4	T3, F1
5	T3, F1
6	T11, F14, F5
7	T12, T13, T15
8	T4, T5, F4
9	T6, T7, F3
10	T6, T7, F3
11	T6, T7, F3
12	F2
13	T9
14	T10

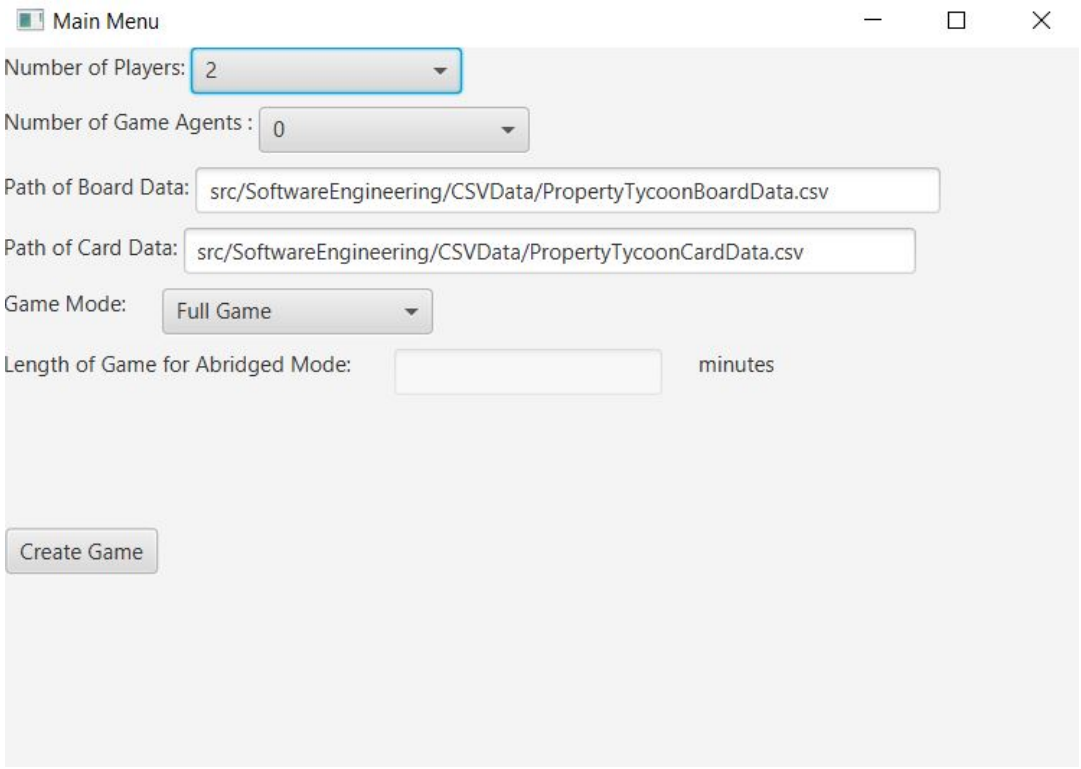


Figure 8.14: Fifth Prototype: Menu View

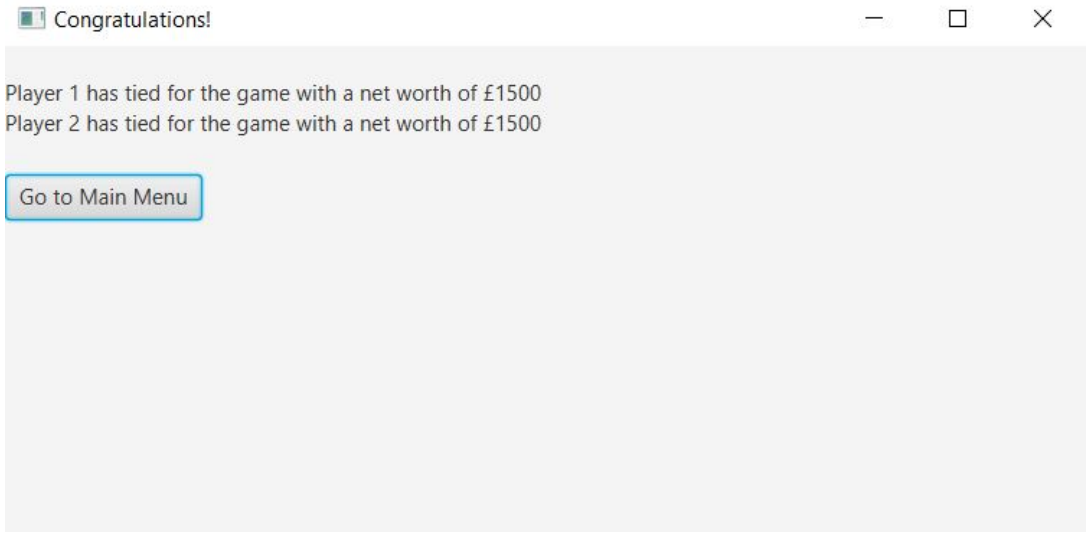


Figure 8.15: Fifth Prototype: End Game View

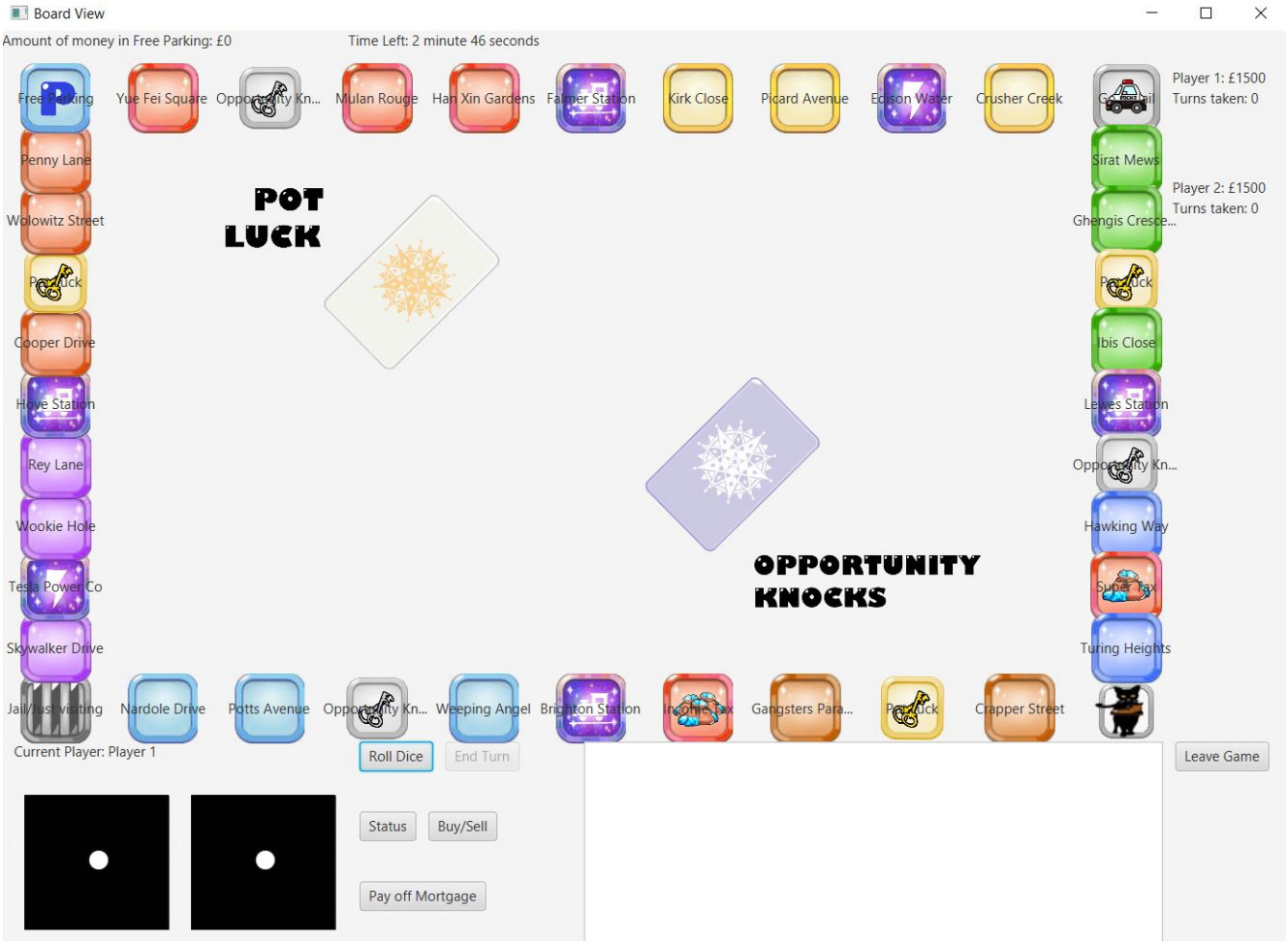


Figure 8.16: Fifth Prototype: Board View

8.12 Reflection

The final sprint was again successful. All tasks were completed on time even whilst some members were travelling back to their home country. We implemented features recommended by the customer, such as dealing with incorrect game configurations. As this is the final sprint, there is no need to talk about what to do for next sprint. Instead, the limitations of our final product will be discussed in the report.

The code for the first prototype can be found in the submission folder, called "FifthPrototype".

8.13 Individual Key Contributions

Test Member	Key Contribution(s)
Jun	T4, T8, T9, T10, System Testing
Jihye	T4, T8, T9, T10, System Testing
Ye-Rang	T1, T2, T5, T6, System Testing
Ibi	T1, T2, T3, T5, T6, T7, T11, T12, T13, T14, T15, Unit Testing, System Testing

Team Meetings

This is where we will document everything that was accomplished and discussed in every group meeting.

9.1 6th February 2020

Duration of meeting: 1 hour

Introduced ourselves to each other for the first time. Then, we had to choose a sensible time where all members are free for a group meeting. We chose a day by sharing our timetables, and decided there will group meetings every Thursday. Some members were confused on some aspects of the assignment. As a group, we thoroughly read through the assignment, explaining parts that were not understood. We found some aspects of "the rules of Property Tycoon" to be ambiguous, especially rules 22 and 23 which are concerned with mortgages. Consequently, we sent an email to the customer's representatives for clarification. Lastly, we plan to assign roles, identify risks and finish planning next week.

9.2 13th February 2020

Duration of meeting: 2 hours

This week we focused on the planning aspect of the project, namely creating a PERT chart and conducting risk management. We will review and update the likelihood of a risk at every team meeting, so we can be prepared if something were to go wrong.

We also decided team roles. Here are the following roles assigned to each member:

- Jun Baek: Project Manager
- Ye-rang Lee: Technical Support
- Jihye Ahn: Design Coordinator
- Ibi Khan: Technical Lead
- Georgios Ladikos: Technical Support

Next week's focus will be designing and coding the first prototype.

9.3 20th February 2020

Duration of meeting: 3 hours

Decided on the programming language to use, namely Java. The main reason being everyone is familiar and has experience with the programming language. We also discussed how we were going to organise and manage our code and decided to use GitHub. The reason is because GitHub is an industry standard and some members have previous experience using it. We set up GitHub accounts and exchanged information. Started to code the first prototype and chose JavaFX over Swing. Next week we will focus on creating a method to link external data files to game logic.

9.4 27th February 2020

Duration of meeting: 3 hours

Discussed many methods of converting the excel data into a suitable form. The two main ones were JSON and CSV. We decided to go with CSV as we are more familiar with this format. Due to issues IDE issues from last week, we concluded that our projects would conform to the Intellij project structure. Decided to use JUnit 4 for testing as this is the most widely used testing suite. Started designing characters. Next week, we will still focus on converting the excel data to CSV data into an ArrayList.

9.5 5th March 2020

Duration of meeting: 3 hours

To prepare for future prototypes, we decided to create a make-shift menu with a single button that loads the board view. This would be a good coding exercise where communications between different windows are needed. Using the CSV data in an ArrayList, we converted it into game logic. Decided that the rent system of a square is significant enough for it to warrant a new class, called "Property". This class will focus on tracking the current rent of the property. Problem with displaying images of squares with a non-colour group as the group column of the board data is empty for non-coloured squares. Asked customer if we can insert our own data. The focus for next week is to implement a buying/selling system where the player can purchase houses/hotels on properties they own according to the rules of Property Tycoon.

9.6 12th March 2020

Duration of meeting: 3 hours

Created the "Status" button where there it shows the state of the board by listing every property each player owns. Noticed that the actions of players were difficult to track. Decided to create a game log that would store information of key events. Though about the process of buying houses/hotels. The process of buying a house/hotel on a property should first be selecting the property to purchase houses/hotels on. The select the option of buying either house or hotel. Then list the outcome of the transaction. Next week, we will implement the actions of cards.

9.7 19th March 2020

Duration of meeting: 3 hours

Asked the customer the best way to go about implementing card actions. Response is that the cards form a small set of actions and so can be automated. Decided that there should be a keyword for every card action. We then decided that the integer/name of square of a card should also be customizable, as well the description. If possible, when the player needs to raise funds to pay a fee, a window should open that allows them to sell assets. Next week we will focus on implementing the jail system.

9.8 26th March 2020

Duration of meeting: 2 hours

Discussed methods of mortgaging and paying off the mortgage of a square. Settled on an extra option in the "Buy Property" window to allow the mortgaging of the square. To pay off the mortgage, there will be a new button that lists all properties under mortgage. In this window, there will be a button to pay off the mortgage. Finished designing the auction window. Decided to use PasswordField objects as bids made will be private. Next week we will focus on adding the two game modes, "Full Game" and "Abridged Game".

9.9 3rd April 2020

Duration of meeting: 0 hours

Some members are travelling back to their home country due to the Covid-19 situation. Set up online communication links using Dicord. We are ahead of schedule and so have cancelled team meetings for this week.

9.10 10th April 2020

Implemented both game modes, "Full Game" and "Abridged Game". Time of "Abridged Game" mode is displayed in the board view. We decided the user will choose the game mode from a ChoiceBox. This way, the user cannot enter an invalid game mode. As recommended by the customer, the Game Agent will be based upon random decisions. Decided that there should be a button to allow the player to leave the game. Created the congratulation screen when the game ends. Decided that there should be a button in the end game screen to take the user back to the main menu. Next week we will focus on implementing the Game Agent.

9.11 17th April 2020

Decided that the Game Agent will have a 50/50 chance to go through with an action. Whenever a Game Agent participates in an auction, it should automatically place its bid, if any, in its corresponding PasswordField object. We also decided that players should not be able to enter bids for the Game

Agent, and so the PasswordField object for Game Agents should be disabled. We also discussed the rules for creating the game. If any of these rules are broken, then the game will not be created.

Weekly Log

10.1 Week 1

- Got to know each other
- Decided that we would meet every Thursday
- Everyone understands the assignment
- Plans for next week: assign roles and complete risk management

10.2 Week 2

- Created a PERT chart
- Assigned roles to each member
- Identified risks and how to manage them
- Risk Management will be carried out at every team meeting
- Plans for next week: start coding and designing the prototype

10.3 Week 3

- Finished coding Dice.java class
- Finished Main.java class
- Updated Controller.java class
- Updated WatsonGames.FXML class

- Updated Player.java class
- Coded the first prototype; a token moves around the board based on what they rolled
- Discussed methods of linking external data files (Excel datasheets, CSV, etc.) to the game logic
- Made Unit tests for first prototype
- Re-evaluated PERT chart
- Monitored and updated risk assessment
- Set up GitHub accounts
- Plans for next week: Design board game characters and code a method to link external data files to game logic

10.4 Week 4

- Finished coding CardCSVConverter to game logic class
- Finished Card.java class
- Finished PackOfCards.java class
- Monitored and updated risk assessment
- Designed board game characters
- Updated Controller class
- Updated WatsonGames.FXML class
- Updated Board.java class
- Board displays multiple player tokens
- Players take turns to roll the dice
- Completed system level test for first prototype
- Plans for next week: finish coding BoardCSVConverter

10.5 Week 5

- Finished coding BoardCSVConverter to game logic class
- Updated Property.java class
- Multiple players can now move around the board
- Made unit tests for second prototype
- Finished Property.java class
- The turn of the current player is shown
- The current balance of all players' are displayed
- Plans for next week: start implementing the rules of Property Tycoon
- Completed system level test for second prototype

10.6 Week 6

- Users can see state of board by clicking "Status"
- Updated Player.java class
- Finished StatusController.java class
- Finished BuyProperty.java class
- Finished BuySellController.java class
- Implemented Buying and Selling rules of Property Tycoon
- Implemented the paying of rents
- Updated menu to allow user to specify number of players

10.7 Week 7

- Made unit tests for third prototype
- Completed system level testing for third prototype
- Added a game log
- Implemented declaring a player bankrupt
- Updated menu to allow user to specify board and card data file paths
- Player can now mortgage a square

10.8 Week 8

- Finished designing auction window
- Created set of action keywords
- Implemented jail functionality
- Amount of money on Free Parking is displayed
- Player can pay off mortgages
- Set up online communications using Discord
- Completed system level testing for third prototype

10.9 Week 9

Some members are travelling back to their home country and, as we are ahead of schedule, we decided to take a week break from the project to let everyone get settled down.

10.10 Week 10

- Added a timer
- Timer is visible in board view
- Finished designing congratulation window

- Finished EndGameController.java
- Made unit tests for fourth prototype
- Implemented both game modes, "Full Game" and "Abridged Game"
- Completed system level testing for fourth prototype

10.11 Week 11

- Finished GameAgent.java
- A game can only be created if the configurations are allowed
- GameAgent places automatic bid if they want to place a bid
- Made unit tests for fifth prototype
- Completed system testing for fifth prototype

Report

Overall, we are extremely pleased with the state of the final prototype. We implemented every criteria the customer has asked for, and more. We are even more pleased when taking in external factors, such as the Covid-19 situation and having a ghost member. When asked, we were able to implement all features/changes the customer asked for. We are fond of the buying/selling of houses/hotels system as it is intuitive and clear. Where deemed appropriate, we have also added extra features that the customer did not ask for. Some of these include the game log, which turned out to be very useful, and also giving the player an opportunity to sell assets in order to pay a required amount of money.

From our point of view, the key highlight of our prototype is the customizability. The name and cost of squares, prices of buying a house/hotel on each square, tax rates, description of cards, actions of cards are all customizable. However, more choices to the user also brings up more potential problems. For example, what if, for the prices of buying houses/hotels, the user enters a non-digit character, such as "£50" instead of "50". The prototype will not be able to convert the former to an integer due to the pound sign. This is just one of many examples of the user entering corrupted data. Other example are missing data and even no data. Dealing with all the potential incorrect data is a whole task on its own, and, if we were to continue developing the prototype, we would implement error checking for the board and card data files. We have, to a certain extent, minimised the ways the user can break the game, such as disallowing game creation with incorrect menu options and only letting digits to be entered in an auction bid.

At the start of the project, we had initially planned to make the Game Agent an actual AI instead of random choices. In fact, we were ahead of schedule and it was looking like an AI Game Agent was very possible, however, having a ghost member, and the Covid-19 situation later on, we opted to not create an AI.

11.1 Peer Review Marks

- Jun - 20
- Jihye - 20
- Ye-Rang - 20
- Ibi - 39
- Georgios - 1

Bibliography

[1] *javax.swing (Java Platform SE 7)*, https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html, accessed on 22/02/2020.

[2] *Overview (JavaFX 8)* https://docs.oracle.com/javase/8/javafx/api/toc.htm, accessed on 22/02/2020.

[3] *Lucidchart*, www.lucidchart.com, accessed on 28/02/2020.

[4] Lucidchart, *UML Class Diagram Tutorial*, https://www.youtube.com/watch?v=UI6lqHOVHic, accessed on 28/02/2020.

[5] Lucidchart, *UML Use Case Diagram Tutorial*, https://www.youtube.com/watch?v=zid-MVo7M-E, accessed on 28/02/2020.

[6] Lucidchart. *How to Make a UML Sequence Diagram*, https://www.youtube.com/watch?v=pCK6prSq8aw, accessed on 28/02/2020.

[7] *Designing a Swing GUI in NetBeans IDE*, https://netbeans.org/kb/docs/java/quickstart-gui.html, accessed on 01/03/2020

[8] *Coupling (Computer Programming)*, https://en.wikipedia.org/wiki/Coupling_(computer_programming), accessed on 01/03/2020

[9] *JavaFX Scene Builder - A Visual Layout Tool for JavaFX Applications*, https://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html, accessed on 01/03/2020

[10] *Class StackPane*,
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/StackPane.html, accessed on 01/03/2020

[11] *Class ImageView*,
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/ImageView.html, accessed on 01/03/2020

[12] *Class Image*,
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/Image.html, accessed on 01/03/2020

[13] *Java String split method*,
https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#split(java.lang.String), accessed on 02/03/2020

[14] *Class TextField*,
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TextField.html, accessed on 02/04/2020

[15] *Class ListView*,
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ListView.html, accessed on 02/04/2020

[16] *Class PasswordField*,
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/PasswordField.html, accessed on 05/04/2020

[17] *Class ChoiceBox*,
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ChoiceBox.html, accessed on 05/04/2020

[18] *Class Platform*,
https://docs.oracle.com/javase/8/javafx/api/javafx/application/Platform.html, accessed on 05/04/2020

[19] *Draw.io*,
https://app.diagrams.net/, accessed on 02/05/2020