|  |  |
| --- | --- |
| Property tycoon  Design Report | SOFTWARE  ENGINEERING |

# **Acknowledgements**

I would like to thank Professor Kingsley Sage for his guidance and his invaluable support throughout this project. As this project would not have been completed without their diligence, communication and hard work.

Contents

[**Acknowledgements** 1](#_Toc102074290)

[**Chapter One** 3](#_Toc102074291)

[**Project Layout** 3](#_Toc102074292)

[**1.1** **Introduction:** 3](#_Toc102074293)

[**1.2** **Aims:** 3](#_Toc102074294)

[**1.3** **Objectives:** 3](#_Toc102074295)

[**1.4** **Plan of Action:** 4](#_Toc102074296)

[**1.5** **Methodology:** 4](#_Toc102074297)

[**Chapter Two** 6](#_Toc102074298)

[**Property Tycoon** 6](#_Toc102074299)

[**3.1 Introduction** 6](#_Toc102074300)

[**3.2 Rules** 7](#_Toc102074301)

[**Chapter Three** 9](#_Toc102074302)

[**Game Implementation** 9](#_Toc102074303)

[**Project UML Class Diagram:** 9](#_Toc102074304)

[**Classes and Methods:** 9](#_Toc102074305)

[**BoardDefinition.py** 9](#_Toc102074306)

[**PlayerFunctions.py** 10](#_Toc102074307)

[**Button.py** 13](#_Toc102074308)

[**SpecialCardFunctions.py** 14](#_Toc102074309)

[**Back End** 14](#_Toc102074310)

[**SpecialCards.py Class:** 15](#_Toc102074311)

[**GameAgent.py Class:** 16](#_Toc102074312)

[**Front End** 16](#_Toc102074313)

[**Low Level Design:** 27](#_Toc102074314)

[**Chapter 4** 27](#_Toc102074315)

[**Understandability vs Functionality** 27](#_Toc102074316)

[**Implementation Time vs User Experience** 27](#_Toc102074317)

[**Packages & Libraries** 28](#_Toc102074318)

[**Chapter 5** 28](#_Toc102074319)

[**NumPy** 28](#_Toc102074320)

[**Random** 28](#_Toc102074321)

[**PyGame** 28](#_Toc102074322)

[**Sys** 28](#_Toc102074323)

[**References** 30](#_Toc102074324)

# **Chapter One**

# **Project Layout**

## **Introduction:**

Property Tycoon is a property trading game which is like the well-known game called Monopoly. My team and I have been appointed as a software contractor for Watson Games, the producers of this game, to produce the video game version of their infamous board game, Property Tycoon. Watson Games have over 50 years of experience developing and marketing board games.

However, they have never designed such a system before and so, they do not have a formal document stating how the software should operate. In addition, Watson Games have come up with a set of user requirements and instructions that, in their own words, describes what they want the system to do. It is likely that the requirements alter as my team and I progress with the software, thus, it is a good idea to design and plan in a flexible manner.

As my team and I are most comfortable with the Python language, we decided it would be optimal to use it in the process of planning, developing and implementing the software system.

## **Aims:**

The aim of this project is to work as a team and successfully create a software that replicates Watson Games’ most famous board game, ‘Property Tycoon’ by planning the design, implementing code and frequently test it and thus, deliver the software system.

## **Objectives:**

Since Watson Games have provided us with a set of requirements that they want to see in the software system, it makes sense to come up with some objectives which will help us stay on track throughout the system.

|  |
| --- |
| 1. Develop an appropriate software system doing what Watson Games require. |
| 1. Use the brief as a checklist in order to maximise efficiency and fulfil requirements. |
| 1. Develop a test plan for the software (unit testing level and system level) |
| 1. Demonstrate that we can work as a team and show understanding of principles of development within software engineering. |
| 1. Give a demonstration of the completed software system. |

## **Plan of Action:**

My team and I are given a total of 11 weeks to complete the system software. Thus, it is a good idea to come up with a plan that divides each member tasks to make the teamwork efficient and organised.

|  |
| --- |
| **Member 1 – Ammar**  Task: Implement main parts of code and keep track of own code.  Aim: To work on code 6-8 hours a day (when Htet is off duty).  **Member 2 – Htet**  Task: Implement main parts of code and keep track of own code.  Aim: To work on code for 6-8 hours a day, (when Ammar is off duty).  **Member 3 – Taha**  Task: To regularly test codes and update changes and document full project, with help of the rest.  Aim: To keep track of previous code vs updated code (Pre-Testing and Post Testing). To document project.  **Member 4 – Gary**  Task: Use Video Game Production experience and is responsible for developing and designing the GUI.  Aim: Ensures the project is user friendly.  **Member 5 – Oussama**  Task: Implementing GUI. Helping hand in implementing code & testing and creating group report.  Aim: ensures checklist and requirements are being met, as well as record data for group report. |

Week 1-6:

* Begin designing and implementing GUI (Graphical User-Interface).
* Begin implementing code by following brief.
* Regularly carry out testing.

Week 6-11:

* Ensure GUI is user friendly and consistent.
* Finish up code, brief checklist.
* Keep testing to find possible errors or to improve efficiency.
* Finish system software.

## **Methodology:**

It is very important to note that while doing this project, methodology is constantly changing. As mentioned previously, a sense of flexibility among all team members is needed prior to starting the project. An increase in testing means a potential increase in errors and way to improve efficiency, hence, testing at a unit testing level and at a system level is required.

Software’s such as Pycharm and Redo will have been used to implement the code and design the Graphical User Interface (GUI) respectively. Furthermore, the requirements provided to us will be used as a checklist or project brief to complete the system software. It is important to acknowledge that there will be constant changes to the software (these changes will be noted and documented), as the project develops.

# **Chapter Two**

# **Property Tycoon**

## **3.1 Introduction**

Property Tycoon is an excellent choice to do this project on. A game like property tycoon covers a lot of the crucial software engineering aspects in the technical as well as theoretical world. The relationship and interaction between the AI and different players have within the game makes it very interesting to develop. In addition, in a game of this calibre, testing plays a significant role. This will allow my group members and I to observe and discover the best solutions to different problems and bugs.

The game itself is has two versions, with the main contrast being time limit. In the first version of the game, also known as the full game, the game is not finished until there remains only one player, with the other players having retired due to bankruptcy or because the players have left the game for any reason, with the agreement of other players. Furthermore, in the second version of the game, which is known as the abridged game, a time limit is agreed beforehand, by all the players in the game. Once the timer runs out, every player counts their assets and simply, the player with the highest value of assets is declared the winner.



**Figure 1.0** The Game Board

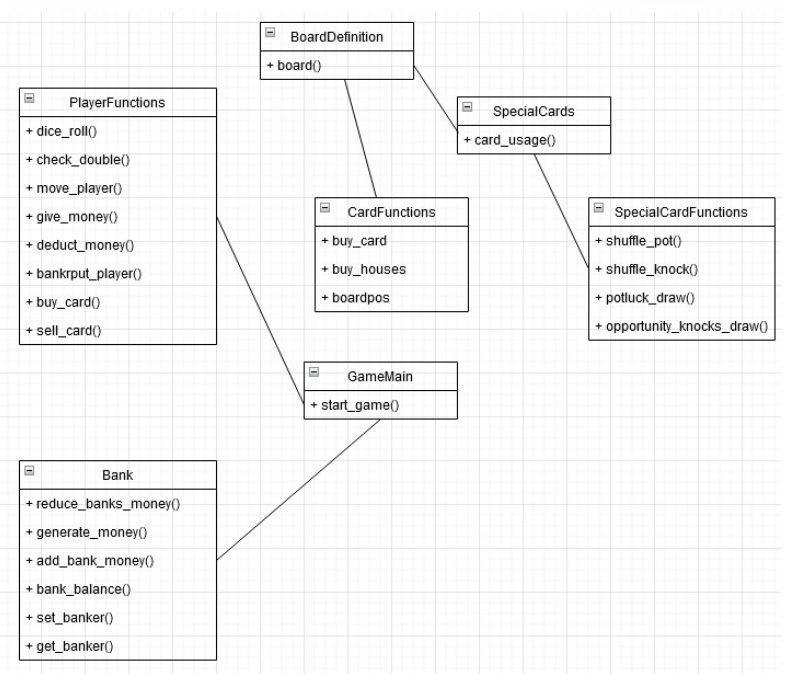
# **3.2 Rules**

|  |
| --- |
| * The game may be played by 1 to 5 human players. One of the game tokens is given to each player. Boot, smartphone, ship, hatstand, cat, and iron are the tokens. Each player takes a turn on the board, rolling two dice to decide their movement. All players begin on the Go board spot and work their way clockwise around the board. * The game will have a minimum of one autonomous computer player agent. * The bank has a total of 50,000 pounds, where each player is given 1,500 by a player who has been chosen as the banker. * Two packs of cards known as ‘pot-luck’ or ‘opportunity knocks’ are shuffled and placed on the board. * A player rolls two dice per turn and move the number of spaces on the board in respect to the number shown on the dice added together. The player must move clockwise around the board. * If a player was to roll the two dice and get the same number, the player must roll again. If this happens for the third consecutive time of rolling, the player must directly move to jail. * If a player falls on a "pot-luck" or "opportunity knocks" spot, they must choose a card from the top of the matching pile and follow the instructions. When you're done, place the card in the bottom of the correct pile. * As they go around the board, players gain progress in the game by purchasing property.  Players may not acquire property until they have passed the Go space and made one complete circuit of the board. When a player passes Go, the bank gives them £200. * As they go around the board, players advance in the game by purchasing property. Players can't buy property until they've passed the Go space and completed one circuit of the board. A player receives £200 from the bank if they pass Go. * If a player lands on a property that has not yet been acquired after making their move, they have the option to purchase it. If they opt not to purchase the property, the bank will auction it off. To the bank, each player makes a bid. The property is sold to the highest bidder by the bank. If no bids are received, the property will stay unsold. All bidding players must have completed one circuit of the board before proceeding to the next round. * If a player lands on another player's property, the player who owns the property must pay the amount of the rent given on the card to the person who owns the property. * The rent required is increased if a player owns all the properties in a color-coded group, but the properties are not otherwise developed with residences and hotels. * If a property is enhanced with dwellings or hotels, the rent due is as shown on the card. * Rent must be always paid in cash. If a player can't pay the rent on a property they've landed on, they'll have to sell game assets to make up the difference. They are bankrupt and must leave the game if they are unable to pay the rent after selling all of their gaming assets. After then, their game token is taken from the board. * Players are unable to borrow or lend money to one another, and they are unable to borrow money from the bank. * When a player has finished transferring their token and performed any property acquisition activities, they may upgrade their properties by purchasing houses and hotels. At any other moment, players are not allowed to upgrade their homes. * Houses and motels may only be acquired when a player owns all of the properties in a specific colour coded category. * The sum given on the game card is used to purchase houses and motels. * A player can sell a property back to the bank for its original value as displayed on the game card if they need money. Only when there are no residences or hotels on the site can it be sold. Houses and hotels can also be sold back to the bank for the original acquisition price. * When a player owns and develops a coloured group of properties, there can never be more than one house difference between the properties in that set. If a player wants to buy a hotel, it will cost them the same as five residences. On one set, a player may have four residences and a hotel on another. * One hotel is the maximum development allowed on any one land. * If a player needs money, they can borrow money from a bank by mortgaging a property. The bank will pay the player half of the property's worth as shown on the game card. While the property is under mortgage, no rentals may be collected. * When a mortgaged property is sold back to the bank, it is sold for one-half of the original purchase price. * The earnings from penalties are accumulated on the free parking spot in the centre of the board. When a player lands on a free parking spot, they collect all the cash that are currently on the spot. * If a player is sentenced to prison, they can pay £50 to be freed. The £50 is added to the parking fees that are already free. After that, the player token is changed to "simply visiting," and the player's turn is completed. In the next round, the player takes a regular turn. * If a player chooses to remain in prison for the next two rounds, they forfeit their turn. A player can't collect rent from other players while they're in jail. The player token is changed to "only visiting" after the following two rounds, and the player's turn comes to an end. In the next round, the player has a regular turn. |

# **Chapter Three**

# **Game Implementation**

## **Project UML Class Diagram:**



## **Classes and Methods:**

## **BoardDefinition.py**



**Figure 1.1**

This class is to do with the user-interface of the main game board. In this class, we have created a class named board\_images and game\_board. These classes will be explained below.

**def board\_images(index):Method**

This method is to do with all the images that are necessary for the game board. Refer to figure 1.0 of this report to know how this class and this method helps is creating the user interface of the game board. In this method we add all the images and create an array containing all the images. We then return the array.

**def game\_board():Method**

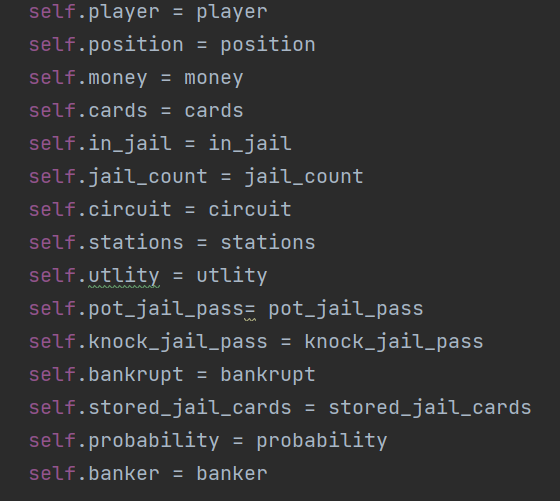
This method helps arrange all the properties and cards onto the board itself in the correct places and allocates the colours of the properties to the place it belongs to.

## **PlayerFunctions.py**



**Figure 1.2**

As the name suggests, this class is exactly to do with the functionality of the players. It is responsible for the player tasks in game such as moving the player, give money to players, deduct money to players and so on. We start the class of by creating a new class named ‘token’. In this class we create instances of objects that are needed to be called. These objects will be demonstrated in te picture below.



**Figure 1.3**

**def use\_jail\_pass(self):Method**

This method allows players (both active players and bots) to us the jail pass special card (if and only if they have obtained it from a deck of either special card.) We use if statements to append te card back into the appropriate deck. For instance, after a player has used their get out of jail pass, firstly a message will be printed saying “…. has used the get out of jail pass”. Next, the card will be popped back into the deck it belongs to. If the player had obtained the card through the potluck deck, the card will be placed back there. If the player obtained the card through opportunity knocks card, the card will be popped into that deck.

**def use\_jail\_pass\_name(self,card\_name):Method**

This function determines which deck the jail pass obtained by the user belongs to. This is needed in order to help the use\_jail\_pass function to pop the cards back into the deck. Hence, this function helps determine if the card belonged to knocks or pot, and will then place the card into its appropriate deck.

**def buy\_out\_of\_jail(self): Method**

A part of this game is that players and bots will be placed in jail, if they were to get unlucky. This method is to do with the functionality of getting out of jail if they player DOES NOT have a get out of jail pass. As the name suggests, this function allows the player to buy themselves out of jail or to skip their turn for the next three turns. We use if statements to check if the player has more than 50 pounds and if they do and the player chooses to leave jail, we have implemented a print statement tat will display a note in game saying “you paid to get out of jail.” The player will then leave jail and roll in the next turn.

**def currently\_in\_jail(self):Method**

This function detects that the player is in jail and asks the player for different options such as to buy their way out of jail, use jail pass or to skip turn.

**def turns\_spent\_jail(self):Method**

As mentioned before, if a player does not want to buy their way out of jail and does not want or does not have a get out of jail pass, the player can skip their turn three times and leave jail. This method is implemented for this specific scenario. We have used an if statement to determine that if it is the players third turn and he/she is still in jail, they will now leave jail and the status would change to “You are visiting jail.”

**def dice\_roll(self):Method**

For a player to move, the dice needs to be rolled. This function rolls the dice and determines how far a player’s token must move. It will then return the two dices’ value.

**def check\_double(self,plist, dice1, dice2):Method**

As mentioned in the rules section earlier, it was mentioned that if a player was to roll the two dice and get the same number, the player will have to roll again. If it is the players third time rolling and the same number is obtained for the third time, the player will have to go to jail.

In this function, if the player has got the same number, we increment the count by one and ask the player to roll again. With the use of if statements, the player will be sent to jail if the incrementation equals to three.

**def move\_player(self, dicesum): Method**

This method is to do with the functionality of moving a players’ token, after the dice has been rolled. After the dice is rolled, the player moves to the position the number on the dice says. If the player was to pass ‘GO’, they acquire 200 pounds.

**def player\_banker(self, banker):Method**

Chooses a banker for the game.

**def give\_money(self, moeny\_added):Method**

Gives money to a player when owed.

**def deduct\_money(self, money\_removed):Method**

Deducts money from a player that owes another.

**def check\_set(self):Method**

We create a dictionary and a count to keep count of the ‘set’ of colors owned by a player. If a player owns a certain number of properties from the same color, they will have a ‘set’ which can be used to build a house which is the next function. This function checks if players obtain a set or not.

**def check\_houses(self):Method**

This function checks if any player owns a house.

**def check\_hotels\_and\_houses(self):Method**

This function checks if a player has a certain number of houses, they can build a hotel.

**def check\_mortgage(self):Method**

We create a dictionary to check if any player has mortgaged their card.

**def bankrupt\_player(self):Method**

This function declares a player bankrupt if they do not have any assets left and have a debt or cannot pay for something.

**def total\_property\_cost(self):Method**

This function is to do with allocating the cost to the different properties in the game. The cost is allocated depending on the color of the property, using if / else statements.

**def card\_color\_owned(self):Method**

This method keeps track of the color of properties a player owns. This will help keep track of the ‘set’ owned too in case a player wants to buy a house on one of the properties.

**def total\_value(self):Method**

This method is for the AI to buy houses that are worth a good value. This function uses a for loop to properties instead of ‘railroad’ or ‘utilities’. Otherwise it will return the value of cost.

**def selected\_card\_group(self, color):Method**

This method allocates each color to its own group.

**def mortgage\_card(self, selected\_card): Method**

The mortgage\_card function checks if a certain player is eligible to mortgage a card or not. If they are, a print statement will allow them to know that they are and if not, then a print statement will display “you cannot mortgage” the card that the player selected. If the player tries to mortgage the same card again, a display message will inform them that they can not do so again.

**def sell(self, amount,bank\_player):Method**

This method allows players to sell properties if needed. We implemented a while loop that checks if a player has a property to sell when money is needed. If the player does not have any property available, he/she is declared bankrupt.

Once the property is sold, it becomes an asset of the bank.

**def sell\_house(self, select):Method**

Allows players to sell a house by pressing ‘y’ or to skip by pressing ‘n’. The house sold will become a property of the bank.

**def play\_list\_cards(self, player\_list):Method**

This method creates a count of the number of properties owned and increments the color of the property owned, which helps with keeping track in case a player owns a set.

**def get\_player\_banker(self): Method**

This will return the banker of the game.

## **Button.py**



**Figure 1.4**

This class is to do with the front-end programming. It is to do with the functionalities of the buttons, once a player clicks something. The functionalities of this class are dependent on what the player clicks.

We start the class off by creating a button class in which we implement the width and the height of the image. This helps us implement the button to an appropriate scale, which helps in being user friendly.

**def draw(self, surface):Method**

This method was used to create a button on screen.

**def click(self):Method**

This function is designed to get the position of the mouse in game. We have used ‘if statements’ tp check mouseover and clicked conditions. The action being done is dependent on what the user clicks.

**def hover(self): Method**

This function has to do with the ‘hovering’ of buttons, which will be talked about further in this report. If a player is eligible for something such as buying a house, the player can hover over the button, indicating that he/se is eligible.

## **SpecialCardFunctions.py**

### **Back End**



**Figure 1.5**

This class is the back-end development of this project. The SpecialCardFunctions.py class works directly proportional with GameAgent.py, which is shown on figure 1.1. This class represents what happens in the back end to implement the rules needed in-game.

**def shuffle\_pot(pot): Method**

In this method, we use the ‘random’ package to randomly shuffle the special cards, also known as the pot-luck cards which are needed in the main game, which will be explained later in this report. We use a parameter ‘pot’ in this method to refer to the pot-luck cards.

**def shuffle\_knocks(knocks):Method**

Similarly, this method uses the random package to randomise the shuffling of the special card known as knock opportunity. It uses the parameters ‘knocks’ to refer to the knock opportunity cards.

**def pot\_luck\_draw\_img(sentence):Method**

This class helps us with the user-interface of the game. In this class, we implement the pictures of all 17 pot-luck cards. These card pictures illustrate what it is that the player that has landed on it must do. For each pot, we have added instructions that will be demonstrated depending on what card was give. We created an array named img\_list and named all the pots alongside the what each card must represent. For instance: pot3, "You are up the creek with no paddle - go back to the Old Creek."

**def potluck\_draw(): Method**

This method chooses a card to pop out of the deck. In addition, if the card that has popped out of the deck is the ‘Free Jail Pass’ card, the card will stay with the player and not go back to the deck. Once it has been used, it will go back to the deck. However, if the card is another card, such as “Move to Go”, the player should move to go and the card goes at the bottom of the pot-luck special card deck

**def opportunity\_knocks\_draw\_img(sentence):Method**

Like the class regarding the pot-luck image class, this class also helps with the user-interface. It also implements pictures of special cards, but instead of pot-luck, the cards are known as opportunity knocks. In the class, we create the images for the 16 ‘opportunity knocks’ cards. We also use the array named img\_list and add the 16 cards alongside what they must represent. For example: knocks1, "Bank pays you dividend of £50." We then use a for loop and allocate each card to what it must represent.

**def opportunity\_knocks\_draw():Method**

This method is similar to the potluck\_draw(): method. However, this is for the opportunity knocks cards rather than the pot-luck cards this time. Again, the method pops out an opportunity knocks card from the deck and if that card is the ‘Free Jail Pass’ card, the special card does not go back to the deck, it remains with the player until they decide to use it. Once it has been used, it will go back to the deck. Furthermore, if the card obtained is not a free jail pass card, the card will go back to the deck.

**def pick\_pot(token, plist, total, choice):**

This function allows us to pick the special pot-luck card from the top of the deck. We use this as using index 0 referring to the first card in the deck.

**def pick\_knock(token, plist, total, choice):**

This function is exactly like pick\_pot: method however, the only difference is that this method is to do with the opportunity knocks cards rather than the pot-luck cards.

## **SpecialCards.py Class:**

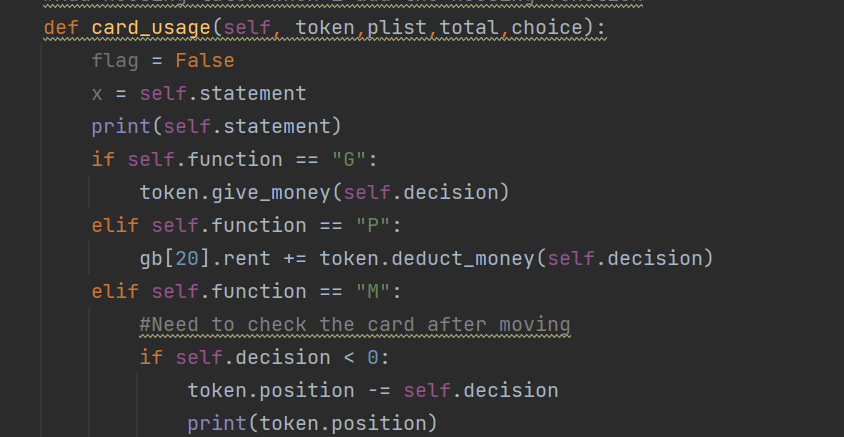


**Figure 1.6**

This class acts as the mother class of the SpecialCardFunctions.py class. This class represents how the functionalities of the cards in the SpecialCardsFunctions.py class work. Essentially it acts like a ‘behind the scenes’ to the SpecialCardsFuncitons.py class. In this class, we created a new class named ‘specialcards’ in this class. In this class, we have a method with parameters “self, card\_name, statement, function, decision): In the method, these parameter instances call themselves, to figure out which card is being picked.

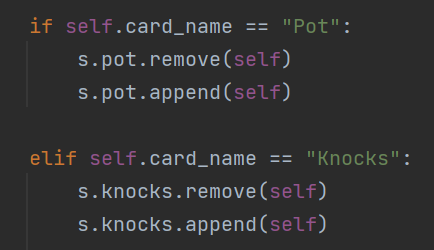
**def card\_usage(self, token,plist,total,choice):**

In this method, we allocate a letter for the functionality for each type of card. For example, in the image below extracted from our code, the arrows represent letters such a “G”, “P”, “M” that stand for Go, Pay, and Move respectively.



**Figure 1.7**

In the method, we use some if statements too, as shown below in the picture that has been extracted from our code.



The if statements allow us to pop the ‘pot-luck’ and ‘opportunity knocks’ cards out from the deck and back in, all the way at the bottom of the dek.

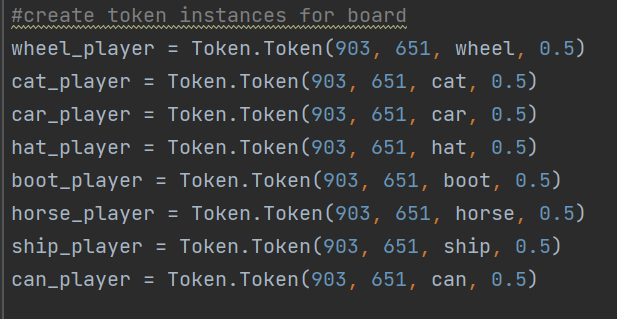
### **GameAgent.py Class:**

### **Front End**



**Figure 1.8 Main Class**

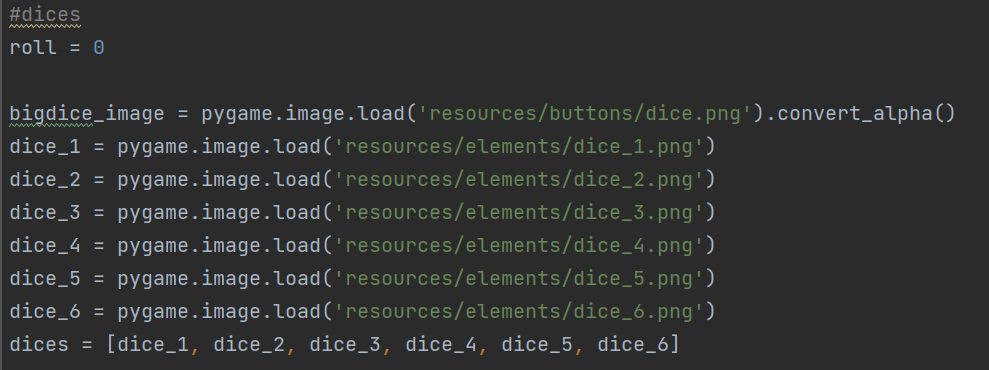
This class primarily focuses on the front-end development of the project as it focuses mainly of the user-interface and animations. Moreover, this is the main class of the game, containing all the main functionalities needed to implement the project. This class contains all the necessary background images, buttons, and tokens. Within this class, we created token instances for the board as well as create a banker, create dice, create all the game user interfaces to start the game and make it presentable. To play property tycoon, one of the main necessary things are the tokens that represent each player. This class creates all the objects and background details needed to make the game playable and creates all the tokens such as: *cat token, wheel token, car token, hat token, boot token, horse token, ship token and can token.*



**Figure 1.9 Creating token instances for the board.**



**Figure 2.0 Implementing and choosing a banker.**



**Figure 2.1 Creating the dice.**



**Figure 2.2 start\_screen(): Function**

**start\_screen(): Method**

The purpose of this method is to create a start screen, as the name suggests, to create a starting menu to welcome the user and give the user a choice of how they would like to proceed. Figure 1.6 shows the start screen implementation that has been worked on in the start\_screen(): method.



**Figure 2.3 Start Screen**

**credit\_menu(): Method**

The purpose of this method is to create a credits menu button and implement it. When the user clicks this button, he is directed to a ‘credits’ menu which will be illustrated in **Figure 1.8.** This credits menu will allow the user to know about the developers of the game and their role in developing Property Tycoon.



**Figure 2.4 Credit Menu Button**



**Figure 2.5 Credits Menu**

**def\_menu(): Method**

Once the user clicks the ‘start’ button on the start game screen, the user will be asked if they want to play with other players or with bots (AI). This is what the def\_menu(): method does. Below, Figure 1.9 is a picture of what the screen looks like.



**Figure 2.6 Menu Choice Screen**

**Def only\_players(): Method**

This method directs the user onto a screen where the user(s) are to pick a token to represent themselves in game. The token they choose will be their representation throughout the whole game. I.e. if a player chooses to be a car, the car will represent the player’s movement throughout the game.

**def ai\_and\_player: Method**

This method is dependent on which option the user picks in the def\_menu method, also known as the menu choice screen in the game. If the user picks ‘Play with Bots’, they will be directed to a screen where they choose the tokens for player and for the bot. Moreover, if the player chooses the ‘play with players’ screen, the users will have to choose a token for themselves. Each player will have to choose a different token.

**def pick\_banker(num\_of\_ai, names): Method**

This method contains the arguments ‘num\_of\_ai and names. The method itself, as the name suggests, allows the users to pick a banker necessary for the game, as the rules state. It is necessary to contain the arguments as we need to know the number of bots playing and the names of the players, depending on which game mode has been chosen. This then makes it simpler to choose a banker, after the tokens have been chosen.

If the user chose the player v player option and there are multiple players that want to be the banker, then a dice will be rolled to determine the outcome of who is the banker. For example, if we have two players that want to be a banker, only one out of the two will become the banker, hence there is a 0.5% probability of either becoming the banker. In this method, once the dice is rolled, if there are two players that want to be the banker, the dice will roll either a ‘one’ or a ‘two’. If player 1 said they want to be the banker first and the dice rolls a ‘one’, then player one is the banker. However, if player 1 said they want to be the banker and the dice rolls a ‘two’, then the second player becomes a banker. Furthermore, if there are more than two people that want to become the banker, lets say there are three people, then there is a 1/3 chance for each person to become the banker as the dice will roll any number from 1-3. Again, the players agree upon a number and whatever number the dice rolls, the player that chose that number becomes the banker.

**def start\_game(): Method**

This function is where the game starts. The user is now in an **UN-TIMED** game mode, playing the game. In this method, we have implemented the functionality of the animation and the game, such as rolling the dice, moving the tokens that have been chosen by the player(s).

In this method, we have a loop that checks that if there is not just one player left, i.e. if one or more other player(s) are not bankrupt, the game will keep running. However, if there is just one player left, the game will go to the winner screen.

In the game rules, a player is not allowed to buy a house unless they own a set of properties that belong to the same set (such as a set of red properties.) In this method, we made this possible by using if statements. If the player DOES own a set of properties, they will see a button ‘hover’, which gives the player(s) an option of buying the house. However, if they do not own a set of properties, the button will NOT hover, indicating that they do not own a set of properties and hence, they cannot buy a house.

Similarly, if a player becomes bankrupt, there will be a bankrupt button that will hover, to let the players know that the certain player is bankrupt and hence has lost. This was also done using if statements. In addition, there is always a mortgage button, this hovers only if a player has mortgaged. Figure 2.0 shows the mortgage button.



**Figure 2.7 Mortgage Button**

Within this function, there will be a button called ‘roll dice’ which will allow the player (who’s turn it is) to roll the dice, once clicked.

If the player rolls the dice and gets the same number on the dice, they will be moved on the board and will see the buy menu and roll again. If they get the same consecutive number when rolling the dice, the player will be moved to the jail menu and will be asked to either pay to leave or remain in jail. However, if they have a special card (will be talked about later) they may leave jail using that card.

Finally, there is a quit button, which exits the system, in case the players want to end the game.

**def get\_player(players, name): Method**

This method gets the index of the players, to display the position of each player on the board. This will display each player’s token on the board.

**def auction\_menu(players, card, idx): Method**

As the name suggests, this function allows the players as well as the AI (bots) to auction a property if needed to. However, we have made a loop to make auctioning only possible if the player has completed one full round around the board. i.e. (passes go). If only one person has passed go, that certain player will be the only one allowed to auction. If no one has auctioned the property, the asset stays with the bank.

This function has arguments of players, card, and index to know which player has passed go and can auction, and which card is being auctioned. The auctions can only be placed with 10, 50 or 100 of the game currency. Once someone has auctioned less than other players, there bet will not be accepted, and the game will display this. If a player places more money and other players decide not to place any more money, the one who has placed the highest bid wins.

**def ai\_menu(player): Method**

This function does the same thing as a regular player; however, the dice rolls automatically as it is AI. It checks for a double roll (of the same number). If the roll does not give the same number, it will move the player and display the buy scree., then move to the next players turn. However, if the player rolls the same number, it will move the player, open the buy menu and allow the player to roll again. Moreover, if the player rolls the dice and the same numbers appear for the third consecutive time, the AI to jail options.

**def buy\_ai\_menu(player, roll): Method**

As the name suggests this method is to do with the buy menu for the bots (AI). The function does something depending on where the AI lands. The AI can land in different places and assets such as on someone else’s property. If this is the case, then the AI will pay the owner of that property. Furthermore, the bot can also land on jail, super tax, income tax and in this case, the AI will go to jail or owe the bank tax money respectively.

If the AI lands on a mortgaged property, they do not have to pay as it is mortgaged and a display message pops up, using a print statement saying, “You do not need to pay because card is mortgaged.” In addition, if the AI lands on a card owned by someone else, and that someone else is in jail, the AI does not have to pay.

Special cards are given when landed on ‘potluck’ or ‘opportunity knocks. These are special cards that give the players and the AI an unknown outcome which can benefit the players, or they may backfire and play a role in leading the player into bankruptcy.

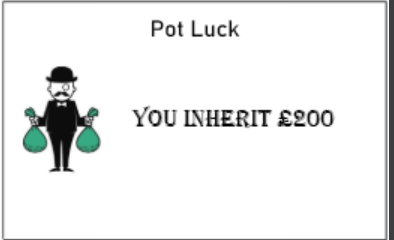
If the bot lands on ‘free parking’ and there is money in the middle of the board, the money goes to the bot and the count of money in the middle of the board is set to zero.

**def buy\_skip\_card\_menu(player,roll):**

This method is the same as the function explained above this for the AI. However, this is for te active players rather than the bots. Hence, these will contain buttons instead of the game automatically continuing as the players have to wait and think of their choices.

**def special\_cards(player, roll, type):**

This function is to do with the implementation of the special cards known as the potluck and opportunity knocks cards. These cards give an unexpected outcome to the player when landed on either of these. It can be good or bad either helping the player(s) get money or the opposite. Figure 2.1 shows an example of the ‘pot-luck’ card. In this case (figure 2.1) the luck that the player as received is good as the player inherits 200 pounds. Moreover, figure 2.2 shows an example of a player’s bad luck, where the player has to pay a late-night taxi bill of 50 pounds. This illustrates that as mentioned before, the pot-luck special cards can be either beneficial or detrimental.



**Figure 2.8 Example of player’s good luck with pot-luck special card**



**Figure 2.9 Example of player’s bad luck with pot-luck special card**

Similarly, Figure 2.3 shows the other special card named opportunity knocks helping the player financially in game.



**Figure 3.0 Example of player’s good luck with opportunity knocks**



**Figure 3.1 Example of player’s bad luck with opportunity knocks**

**def choice(player, roll):Method**

Since we do not know the order of cards for potluck, there is a chance the user may get ‘pay a 10 pound fine or take opportunity knocks card.’ The player has a choice to take the risk and take the opportunity knocks card rather than paying.

As for the AI, it works for a similar process, however the AI has a certain probability that makes them choose between picking an opportunity knocks card or pay.

**def birthday(player,allplayers):Method**

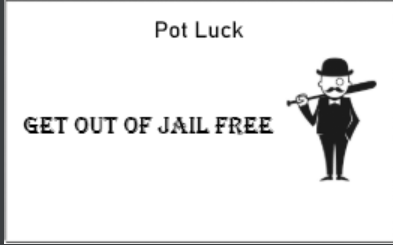
This function is for a specific special card which gives the player the following: "It's your birthday. Collect £10 from each player." Once the player gets this card, he/she will receive 10 pounds from each active player.

We have used ‘if’ statements to know who will have to pay in this situation. Each player with more than 10 pounds will have to pay the 10 pounds to the player who received the potluck birthday card. If the players money is less than 10 however, they will need to sell a card in order to pay the 10 pounds owed to the birthday player.

**def jail\_passes(player,roll,type):Method**

This method is to do with the jail pass card which is shown below in figure 2.5. This method allows the player to keep the pot-luck card if it is the "Get out of jail free" card. The card essentially acts as a free pass for the player to leave jail without paying a fine.

We use an if statement



**Figure 3.2 Get Out of Jail Card**

**def display\_player\_properties(player, cards):**

In this method, if a player bought a card, they can hover on top of that display card button ‘cards owned.’ However, if the player does not own a card, he/she is unable to click the button. Once a player has bought a certain card / asset, they are able to display the card they own.

**def jail(player):Method**

This method is when a player enters jail. Once the player enters jail, the player is shown five options. “Wait until release, Exit Jail, Use Jail Knocks Card, Use Jail Pot Card and Pay.” If the user clicks pay, he/she will be charged 50 pounds. Those 50 pounds go to the middle of the board. However, if the person decides to remain in jail, the player will remain in jail for the next three turns. If the player decides to use the free pass given from potluck or opportunity luck, the person does not have to pay the 50 pounds. The free pass card will then go to the special card deck that it belongs to and hence, the player will not have the free pass anymore.

If the player decides to remain in jail for the next three turns, after the three turns, the player will need to click exit jail to leave.



**Figure 3.3 Four Choices Player in Jail Gets**

**def buy\_houses(player, cards, color\_set):Method**

This method is a check function. The player is given 6 buttons of each property card. This function checks which color set the player is able to buy houses for. Once a specific color is picked, the function will go to the ‘two\_house\_set’ method or ‘three\_house\_set’ method depending on the color set that the player has.

**def two\_house\_set(player, cards):Method**

This method has been kept as we did not want to create a whole function to sell houses, this was done as it was much easier for us to read the code as it was more robust and efficient than making a new function. The method itself was implemented to allow a player to buy houses. This method first displays the specific card that a player owns, it then shows the number of properties that the card has, on top of the card. We use the indexmax function to know if a card has properties than the other cards. Thus, it picks the card with the lower number of cards.

If a player ends up having a hotel, the player has reached the maximum number of properties owned.

**def three\_house\_set(player, cards):Method**

This method is exactly as the ‘two\_house\_set’ method however, the difference is that this method is for three houses rather than two.

**def check\_sellable(player, amount, pay\_player):Method**

This function checks which cards a player can sell as well as if the card the player wants to sell includes a house within the set of properties. If that is the case, the user will be directed to the sell house menu.

**def sell\_card(player, cards, amount, color, pay\_player): Method**

This function allows the player to either mortgage the card or sell the card. Selling the card would allow the bank to pay the player the original amount that the card was bought for. The bank would then own the card. However, if the player mortgages the card, the player will still own the card and receive 50% of the value of the card. The player can also un-mortgage the card given that he/she as enough money in the future.

**def sell\_two\_house\_set(player, cards):Method**

This method is the opposite of the ‘buy\_two\_house\_set’ method.

**def sell\_three\_house\_set(player, cards):Method**

This function is the exact opposite of the ‘buy\_three\_house\_set’ method.

**def bankrupt\_person(player): Method**

This function allows us to eliminate any player that is bankrupt. We have used ‘it statements’ to do this. A player’s token that represents a balance of zero and is bankrupt, will be eliminated. We have also used Booleans to set the value of some functions as ‘false’ and ‘true’ to determine which token is eliminated.

**def bankrupt\_player(player):**

This method too, has to do with bankruptcy but more of to do with the animation of it instead. Once the player is eliminated, the bankrupt button hovers so the user can click it.

**def get\_best(players):**

If the players chose to play a ‘timed’ game or a player decides to end the game, to make sure that there is a winner, we use the get\_best(players): function as it gives us a winner based on the amount of money and accumulated assets that a player owns. The player with the most money and accumulated assets will be declared the winner.

**def winner(players):**

This method is to do with implementing the functionality of determining the winner and the winner’s token. The use of the while loop allows us to display the winner with a ‘winner background’, in addition to that, the winner’s token will also be displayed.

# **Low Level Design:**

## **Chapter 4**

## **Understandability vs Functionality**

Property Tycoon has very similar rules to the board game Monopoly. In fact, the rules are pretty much identical. In addition, Monopoly is one of the most played games in the world and everyone that is to buy Property Tycoon is likely to know the rules of it, prior to buying it. Us developers have followed Watson Games’ brief and hence have followed the game rules to implement it. No additional functions were added by the developers as we have followed exactly what was needed for the game to work as it always did. Hence, this will make it user-friendly and not confusing for the customers and players.

## **Implementation Time vs User Experience**

We have developed this game using Python hence, our coding environment will be PyCharm and PyGame. We do not have a database to store the money & assets of the players so there is no other environment application used. Instead, all the information is stored in the memory, which has its downsides as well as benefits and is something that could have been improved, given more time. The game can be played with other players given they play on the same device, thus despite the fact multiple players can play, the game is not an online multiplayer game. This will also allow to load the game much faster as no internet is required and so, as soon as the project is run, the game will load. A benefit of using Python is that it is quite efficient and easy to implement a GUI which ultimately helps in making the code more efficient. For instance, before we had implemented the GUI, we were testing & running the game on the console. During that time, the code was long and less efficient than it was compared to after we had implemented the GUI. This is as the GUI allowed us to cut out any unnecessary code and allowed us to shorten it while doing the same job, just more efficiently.

A game like Property Tycoon requires us to implement quite a few animations. Moving the tokens, rolling the dice, auctioning are a few examples of the animations that are included in this project. In addition, we did have more ideas and resources such as adding noise when the dice is rolled and choosing the difficulty when playing against the AI however, due to insufficient time, we focused on the core elements needed for the game. With more experience and practice, we believe that implementing these additional features will become easier.

As for the user experience we believe we have worked hard to allow players to experience a smooth and user-friendly game. Although we did make the game aesthetically pleasing to the eye, we knew that focusing on the accessibility is more important. The font of letters and sizes of the buttons are very well designed. In addition, the placement of these buttons is also well-thought of and designed to be user friendly. As mentioned beforehand in the report, we have also created an ‘**about’** button in the main menu to allow the user to briefly revise the rules if necessary.

# **Packages & Libraries**

## **Chapter 5**

## **NumPy**

NumPy is the most important Python module for scientific computing. It's a Python library that includes a multidimensional array object, derived objects (such as masked arrays and matrices), and a variety of routines for performing fast array operations, such as mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation, and more *(What is NumPy? — NumPy v1.22 Manual, 2022).* NumPy uses much less memory to store data which allows the code to be further optimised *(NumPy: the absolute basics for beginners — NumPy v1.23.dev0 Manual, 2022).*

## **Random**

Python's Random module generates random numbers. These are pseudo-random numbers, which does not imply they are genuinely random. This module may be used to generate random integers, print a random value for a list, or generate a random text *(Python Random Module - GeeksforGeeks, 2022).*  Property Tycoon requires the rolling of dices and hence, we need to randomise the numbers appearing on the dice to stop any cheating or un-fairness.

## **PyGame**

PyGame is a Python language library. It's a framework for creating 2-D games that allows you to build up Python modules to create a game. It's a user-friendly platform that makes it simple to create games *(How to Install Pygame on Windows ? - GeeksforGeeks, 2022).* Since we chose to code this project, PyGame was a very suitable library to use as it is a game we are developing.

## **Sys**

The sys module in Python contains several methods and variables for manipulating various aspects of the Python runtime environment. It helps you to work with the interpreter since it gives you access to variables and functions that work closely with it *(Python sys Module - GeeksforGeeks, 2022).*

# **References**

1. Numpy.org. 2022. *What is NumPy? — NumPy v1.22 Manual*. [online] Available at: <https://numpy.org/doc/stable/user/whatisnumpy.html#:~:text=NumPy%20is%20the%20fundamental%20package%20for%20scientific%20computing%20in%20Python.> [Accessed 27 April 2022].
2. GeeksforGeeks. 2022. *Python Random Module - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/python-random-module/#:~:text=Python%20Random%20module%20is%20an,a%20list%20or%20string%2C%20etc.> [Accessed 27 April 2022].
3. GeeksforGeeks. 2022. *How to Install Pygame on Windows ? - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/how-to-install-pygame-in-windows/#:~:text=PyGame%20is%20a%20library%20of,build%20games%20quickly%20and%20easily.> [Accessed 27 April 2022].
4. GeeksforGeeks. 2022. *Python sys Module - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/python-sys-module/#:~:text=The%20sys%20module%20in%20Python,interact%20strongly%20with%20the%20interpreter.> [Accessed 27 April 2022].