Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

Product Design

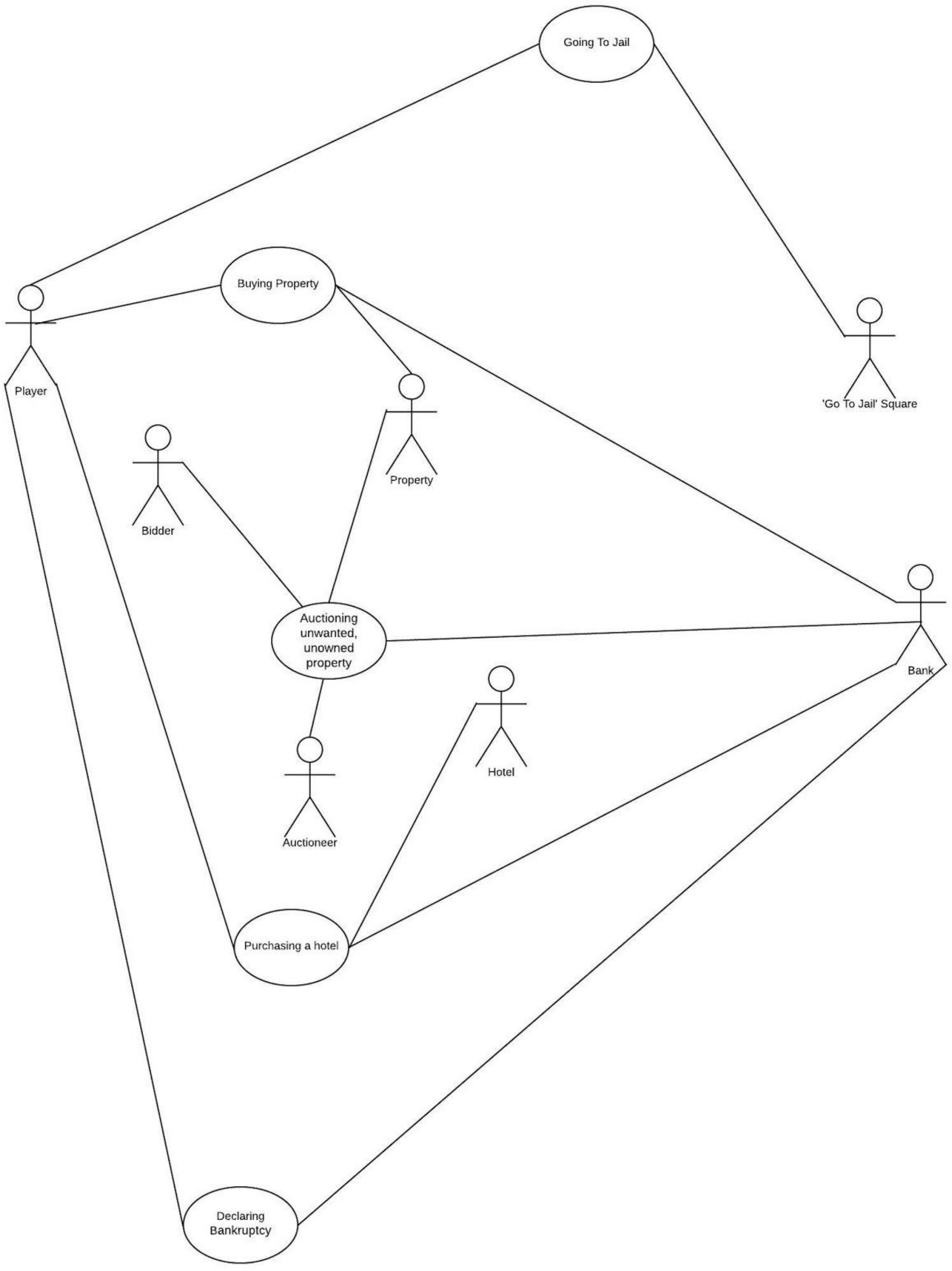
The second stage of the Project is the Product Design. This includes the UI designs, refinements on the classes, object and sequence diagrams. We will also be performing client server tests and providing class skeletons.

Revised Use Case Diagram

Previously, our use case diagram had crossing lines which unintentionally created connections that were not there. We have revised our diagram to remove these interconnecting lines.

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

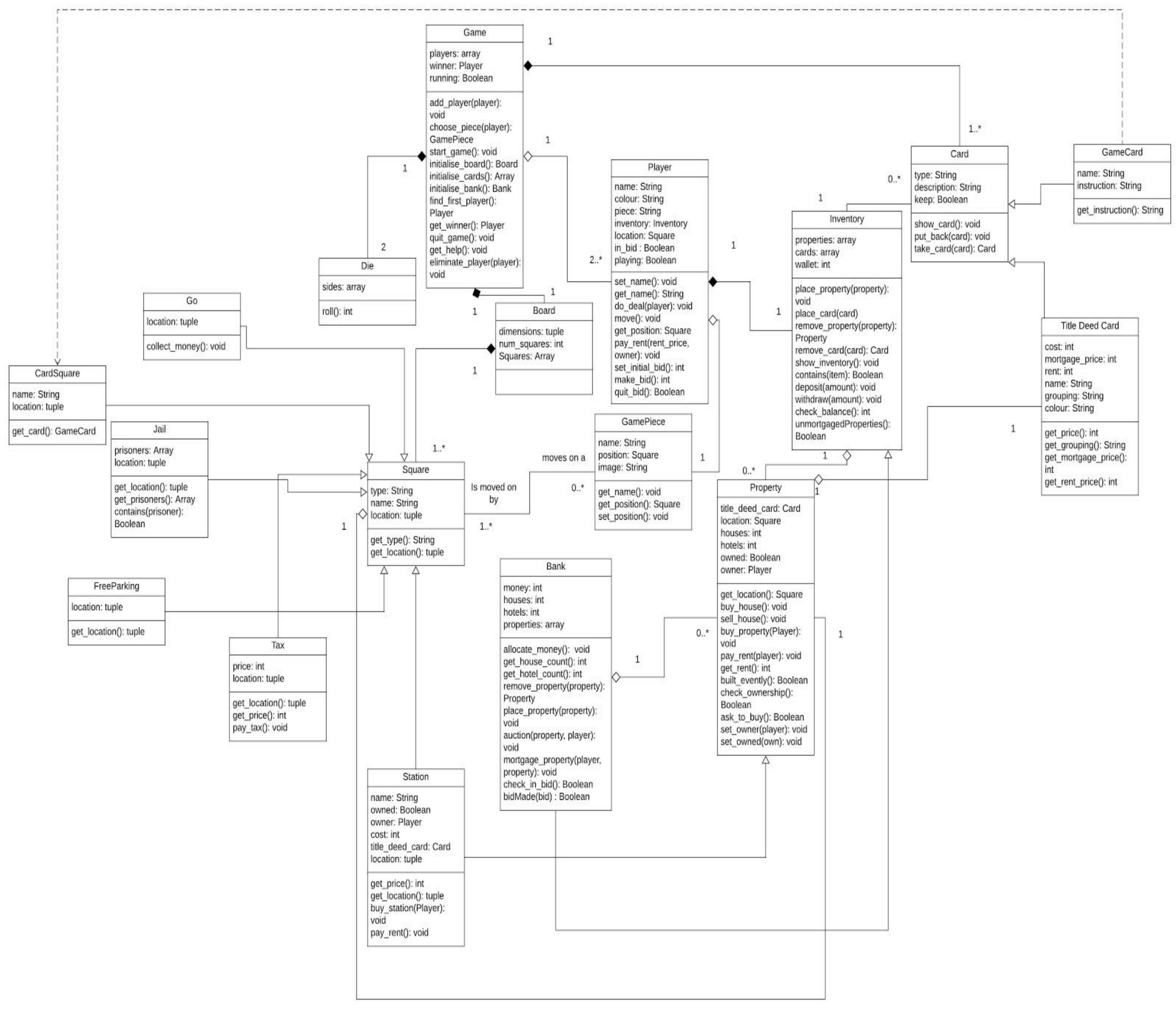


Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

Refined Class Diagram

The class diagram has been refined. We have split up some of the classes to make them more manageable and added some more functions to the classes as different flows and operations became more apparent.



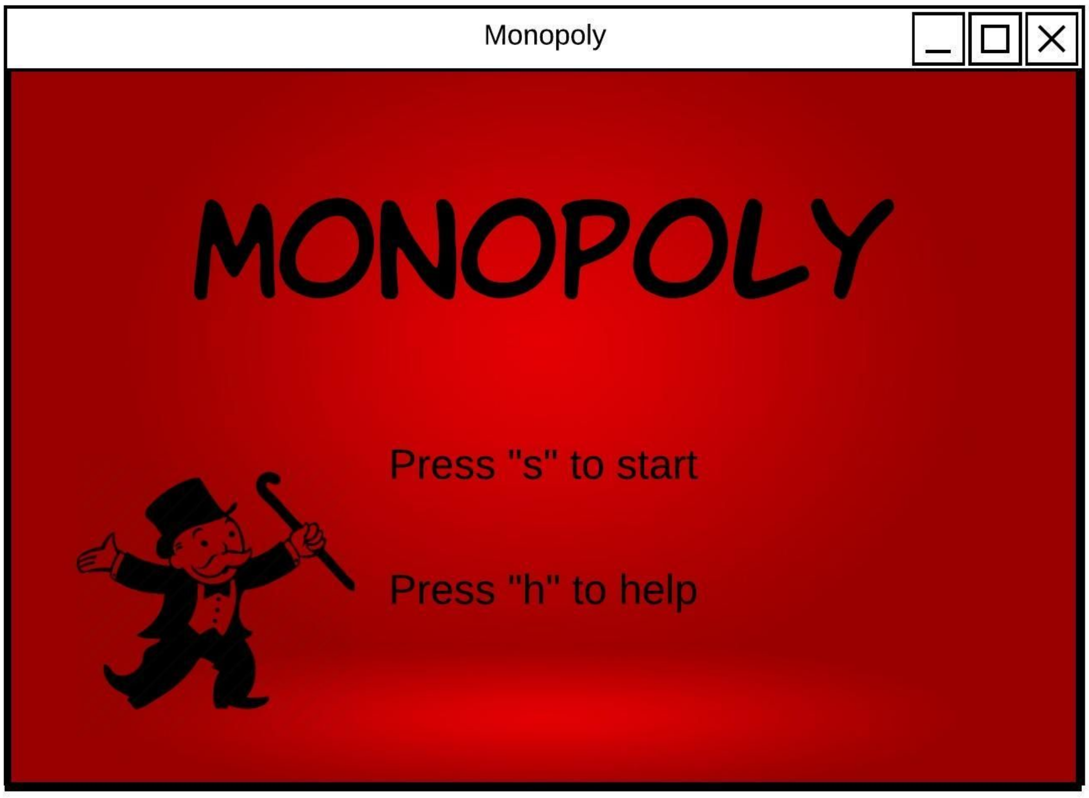
Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

User Interface MockUps

The Main Menu

The game will open on the below screen. This is the Home Menu. From here, the user can choose the start the game or learn how to play from the Help Menu.

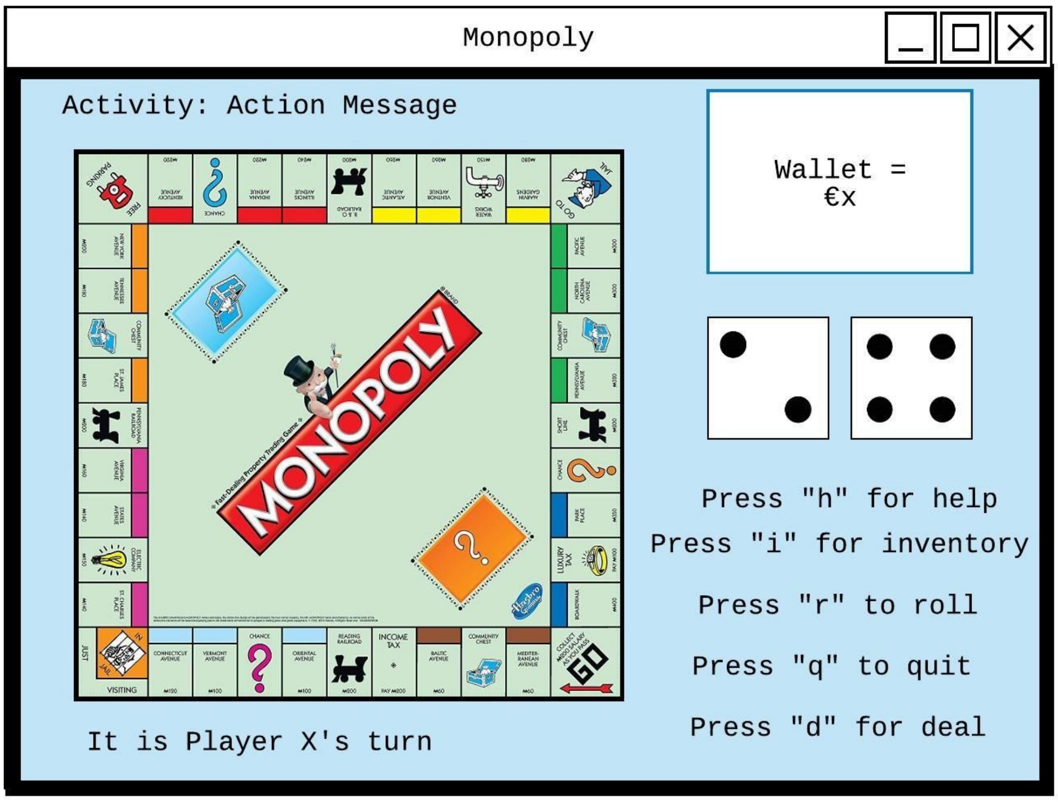


Game Screen

This is the main screen. It is where the main gameplay occurs. From this screen, the user clicks to roll, moves around the board and performs regular game options.

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white



Success Screen

When a player has won a game, a success screen will be shown before the screen switches back to the main screen again.

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

Client Server Experiments

Blah blah blah

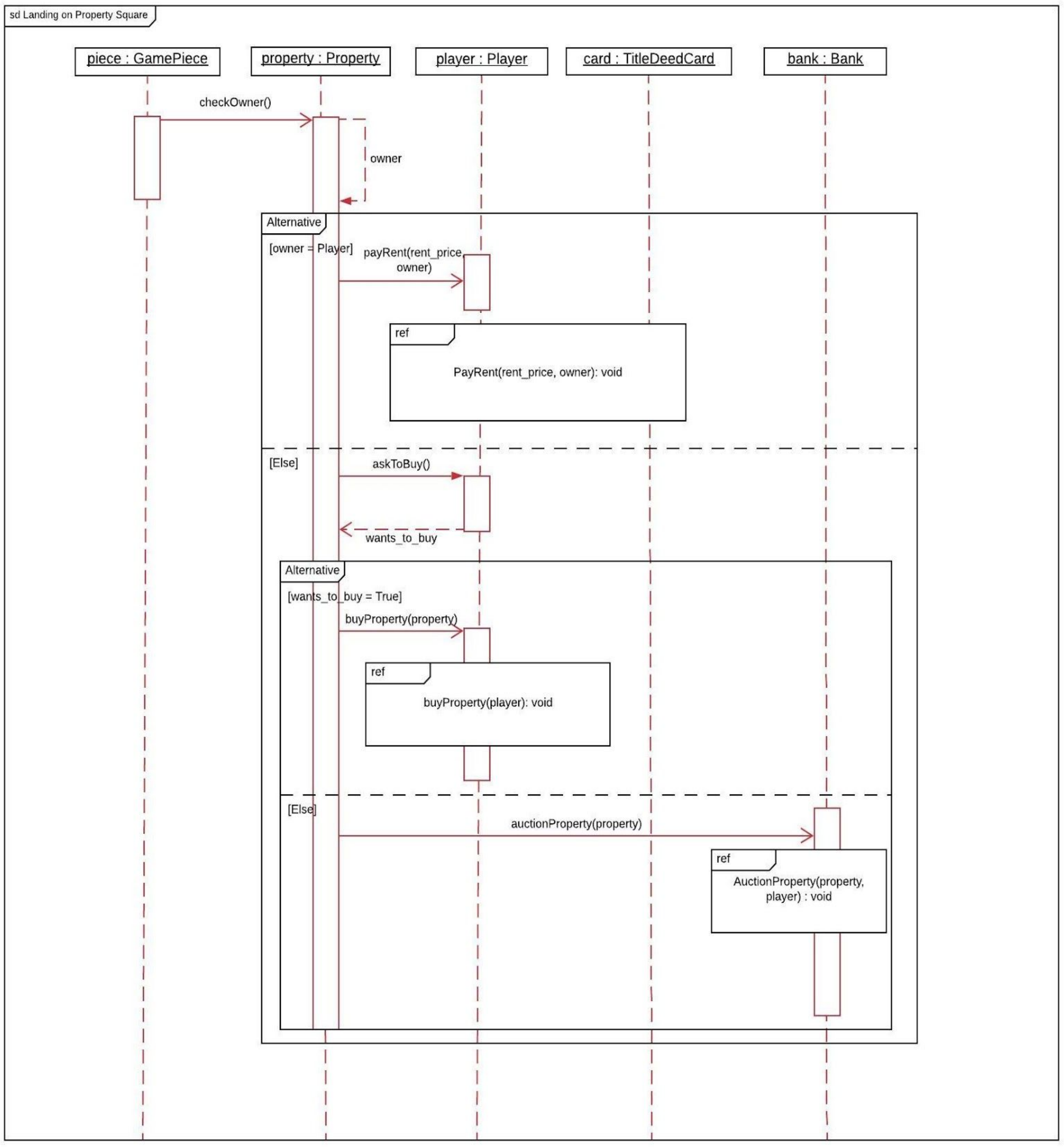
State Machines

Blah blah blah

Design and Analysis of a Game of Monopoly

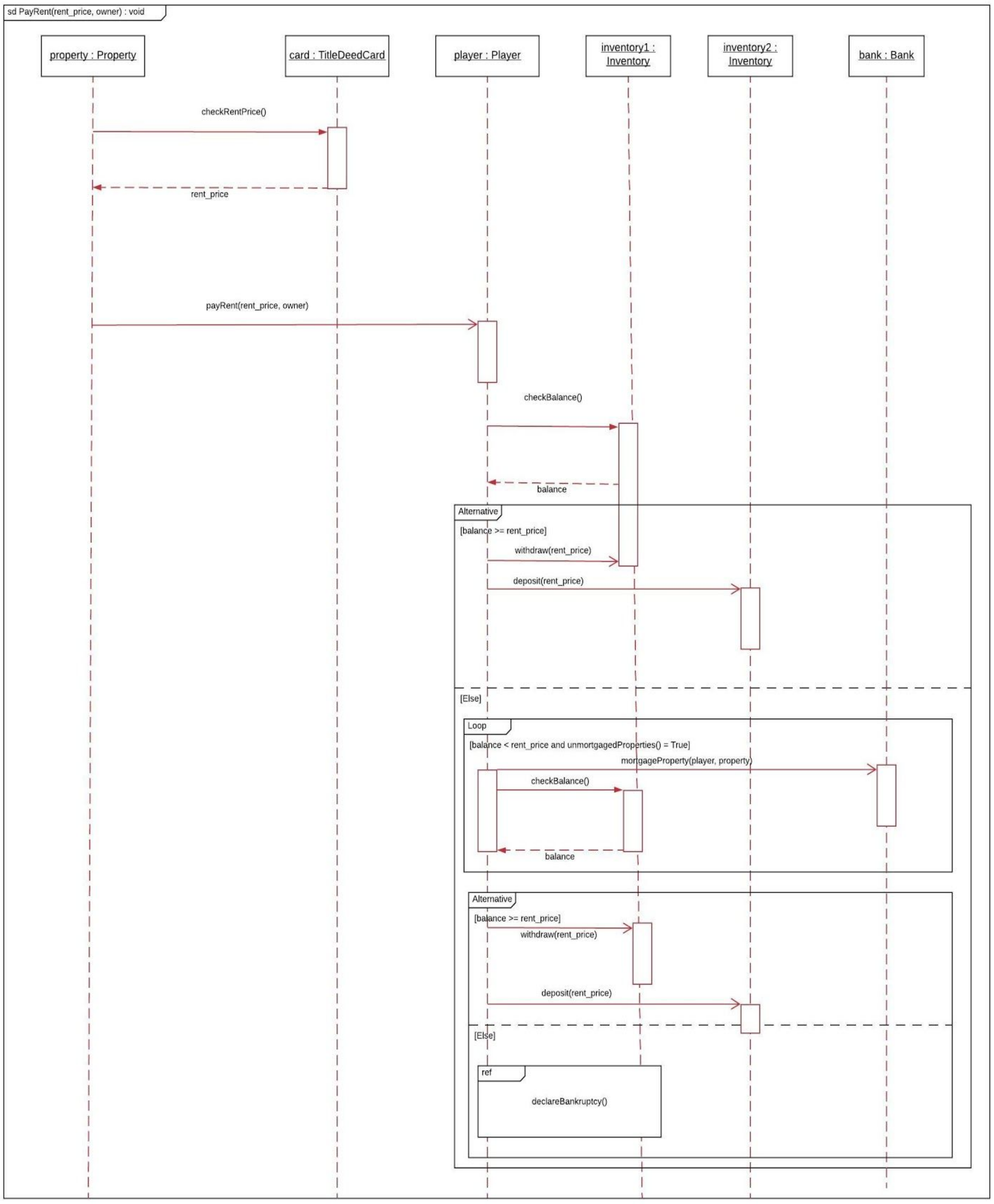
Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

Sequence Diagrams



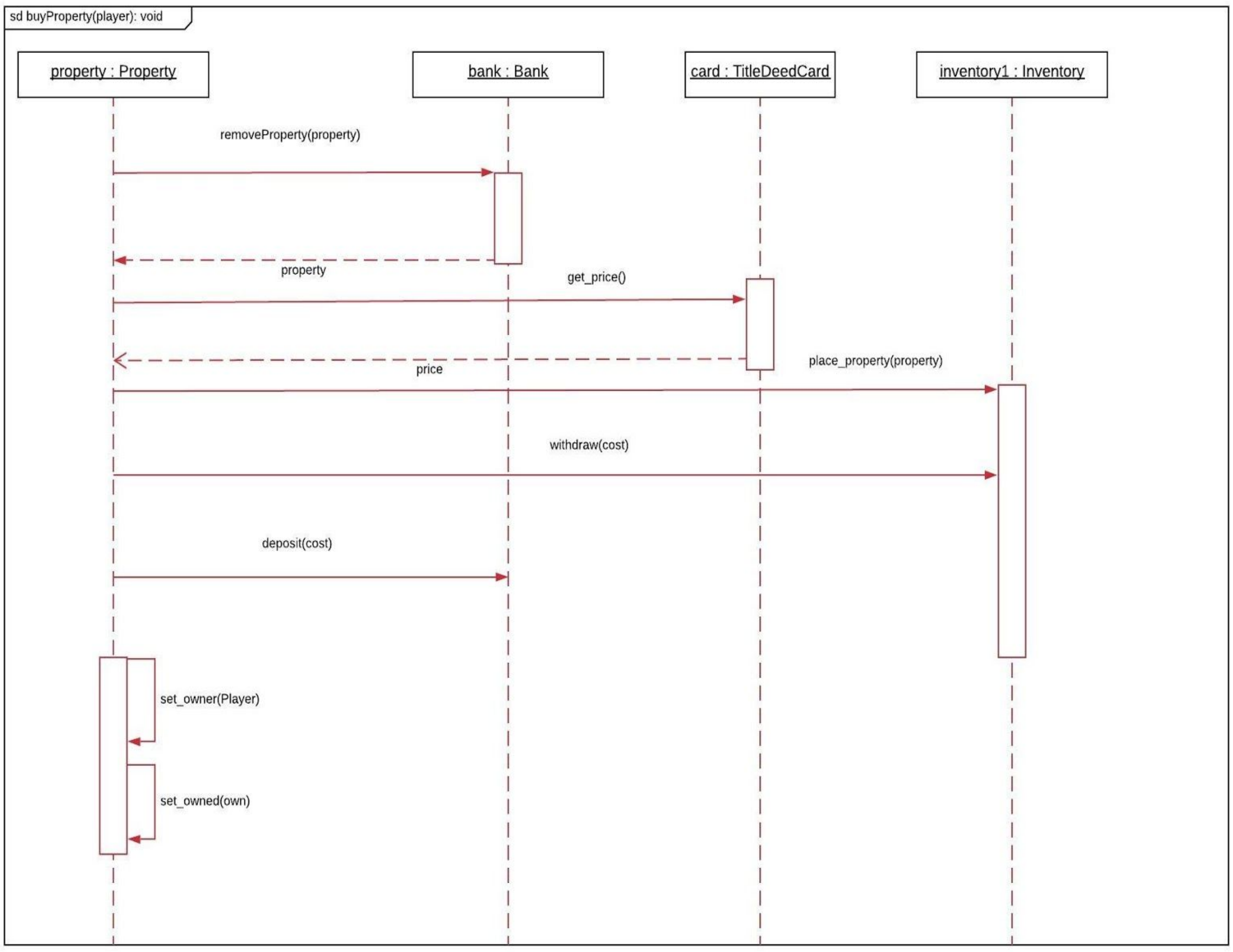
Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white



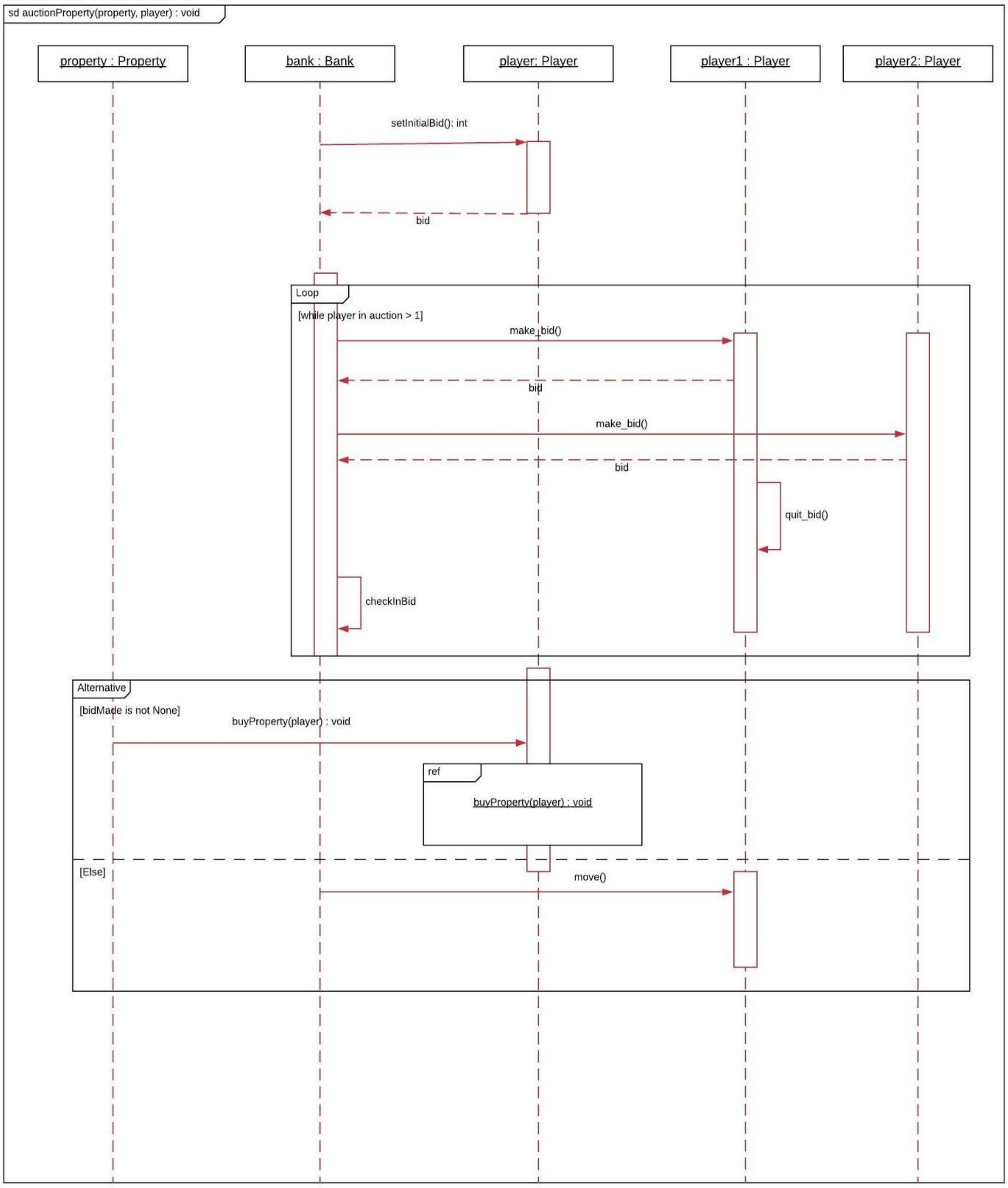
Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white



Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

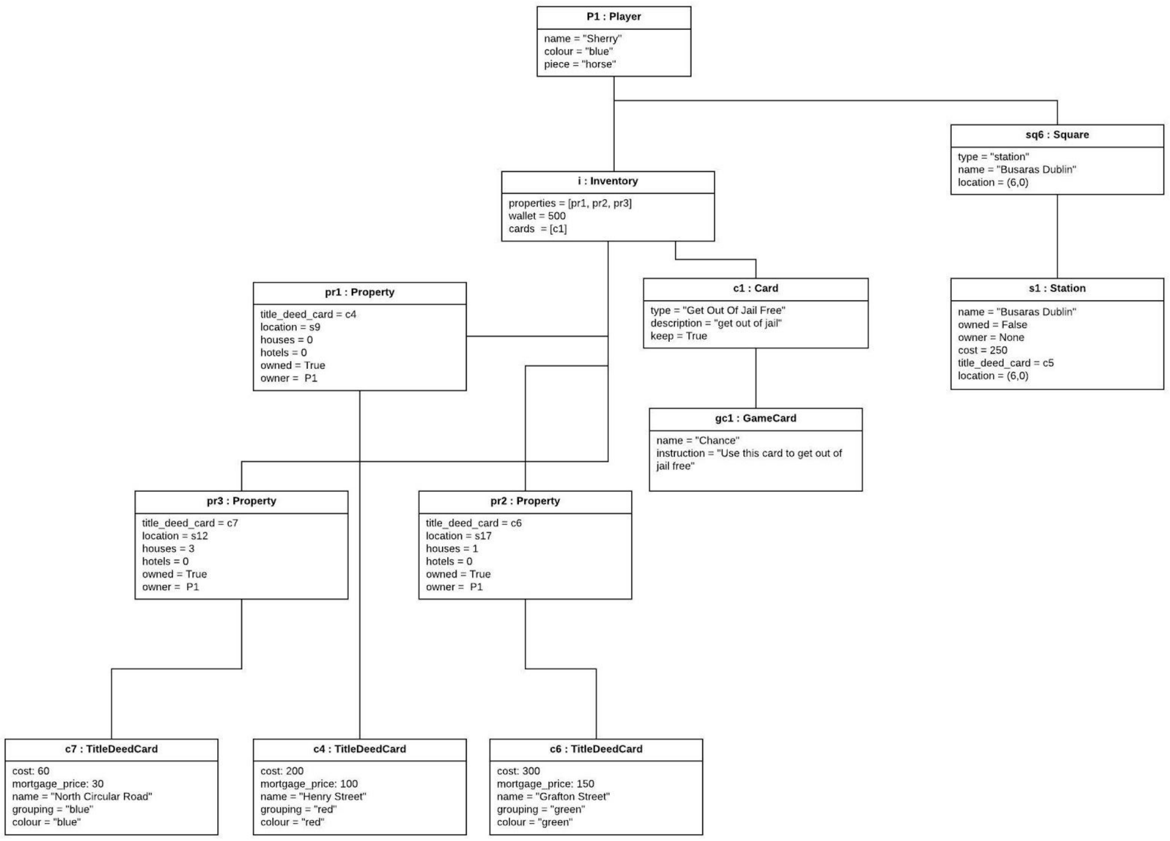


Design and Analysis of a Game of Monopoly

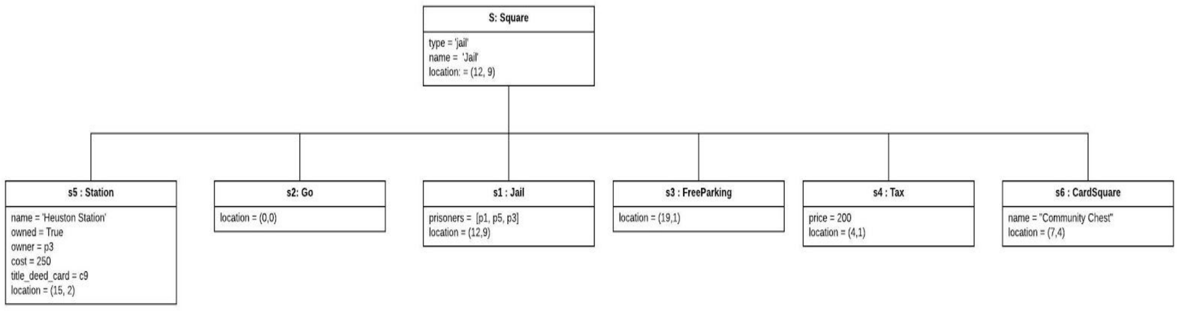
Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

Object Diagrams

Player



Square



Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

Communication Diagrams

Blah blah blah

Revised Object Diagrams

Blah blah blah

Class Skeletons

We have constructed class skeletons based on the refined class diagram. Each class skeleton is shown below,

Class Die

```
import random

class Die:

      def __init__(self):
            self.sides = [1,2,3,4,5,6]

      def roll(self):
            """Roll the dice and return the sum of the 2 dice"""
            index = random.randint(0,5)
            index2 = random.randint(0,5)
            return self.sides[index] + self.sides[index2]
```

Class Square

```
class Square:

      def __init__(self, type, name, location):
            """Initialises the square"""
```

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

```
		self.type = type #Is a string that represents the type of square
		self.name = name #Is a string that is the name of the square
		self.location = location #Is a tuple that represents the location on
the board

	def get_type(self):
		"""Returns the type of square"""
		return self.type

	def get_location(self):
		"""Returns the location of square"""
		return self.location

	def get_name(self):
		"""Returns the name of the square"""
		return self.name
```

Class Board

```
class Board:

	def __init__(self, squares):
		"""Initialises the board"""
		self.dimensions = (11,11) #The dimensions of the board
		self.num_squares = 40 #Number of squares on the board
		self.squares = squares #A list of squares on the board
		self.posx = posz #X Position of the board on the screen
		self.posy = posy #Y Position of the board on the screen
```

Class Inventory

```
class Inventory:
	"""A player's inventory, holds a player's item"""

	def __init__(self):
		"""Initiate the inventory"""
```

```
            self.properties = {} #Properties stored in a dictionary, keys are 
properties, values are mortgaged or unmortgaged
            self.cards = [] #List of cards
            self.wallet = 0 #Player's money

      def place_property(self, property):
            """Place the property in the inventory"""
            if property not in self.properties:
                  self.properties[property] = "unmortgaged"

      def place_card(self, card):
            """Place the card in the inventory"""
            self.cards.append(card)

      def remove_property(self, property):
            """Remove the property from the inventory"""
            self.properties.remove(property)

      def remove_card(self, card):
            """Remove the card from the inventory"""
            self.cards.remove(card)

      def show_inventory(self):
            """Show the inventory"""
            items = " ".join(self.properties) + " ".join(self.cards)
            return items

      def contains(self, item):
            """Check if the item is in the inventory"""
            if item in self.properties or self.cards:
                  return True
            else:
                  return False

      def deposit(self, amount):
            """Deposit money in the wallet"""
            self.wallet += amount

      def withdraw(self, amount):
            """Withdraw money from the wallet"""
            self.wallet -= amount
```

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

```
	def check_balance(self, amount):
		"""Check the balance of the balance"""
		return self.wallet

	def unmortgagedProperties(self):
		"""Check if there are any unmortgaged properties in the inventory"""
		for property in self.properties:
			if self.properties[property] == "unmortgaged":
				return True
		return False


	def check_balance(self, amount):
		return self.wallet

	def unmortgagedProperties(self):
		for property in self.properties:
			if self.properties[property] == "unmortgaged":
				return True
		return False
```

Class Player

```
from Inventory import Inventory
from Die import Die
from Bank import Bank

class Player:

	def __init__(self, name, colour, piece, location):
		"""Initialises Player"""
		self.name = name #name of player
		self.colour = colour #Player's colour
		self.piece = piece #Player's piece
		self.inventory = Inventory() #Player's inventory
		self.location = location #Location of Player's GamePiece. This is a tuple
		self.dice = Die() #Player's Dice
		self.square = square #This is the square the player is on
		self.in_bid = False #Is the player bidding in an auction
```

```
            self.playing = True #Is the player still playing?

      def get_name(self):
            """return the name of the player"""
            return self.name

      def do_deal(self, player, player_items, own_items):
            """Do a deal with a player. Takes a player that they are dealing with and a list
of items to trade"""
            #If the other player agrees to the deal
            if player.deal_agreed():
                  #Loop through array
                  for item in own_items:
                        #If the item is a property then place it in the other's inventory
                        if is_instance(item, Property):
                              player.inventory.place_property(item)
                        #If the item is a card, then place it in the other's inventory
                        elif is_instance(item, Card):
                              player.inventory.place_card(card)
                        else:
                              #Otherwise, it is money deposit it
                              player.inventory.deposit(item)
                  #Iterate through the other players items
                  for item in player_items:
                        #Is the item a Property, if so remove it from their inventory
                        if is_instance(player_items, Property):
                              self.inventory.remove_property(item)
                        #If the item is a Card, then remove it from their inventory
                        elif is_instance(item, Card):
                              self.inventory.remove_card(card)
                        #Otherwise it is money, so we withdraw it from their inventory
                        else:
                              self.inventory.withdraw(item)
            #The player didn't agree, see if they want to counter or reject
            else:
                  if player.deal_countered():
                        player.counter(self, player.items, self.items)
                  elif player.reject_deal():
                        return False
      def move(self, moves=None):
            """Move the player's game piece"""
            if move is None:
                  moves = self.dice.roll()
                  self.location += moves
            else:
                  self.location += moves
            pos_on_board = self.location.get_location() #Tuple position for square on board
            self.piece.update_position(pos_on_board[0], pos_on_board[1]) #Update game piece


      def get_location(self):
            """Get the location of the player"""
            return self.location

      def get_square(self):
            """Return the square the user is on"""
            return self.square

      def set_initial_bid(self, bid):
            return bid

      def make_bid(self, bid):
            self.in_bid = True

      def quit_bid(self):
            self.in_bid = False

      def deal_agreed(self, agree):
            """Takes a boolean 'agree' which if true, then the deal is agreed otherwise it
is not"""
            return agree

      def deal_countered(self, counter):
            """Takes a boolean, If the deal has been countered, then get them to choose the
deal"""
            return counter

      def reject_deal(self, reject):
            return reject
```

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

```
	def counter(self, player, player_items, own_items):
		do_deal(player, player_items, own_items)

	def want_to_buy(self, answer):
		if answer is False:
			return False
		else:
			return True

	def reject_deal(self, reject):
		return reject

	def counter(self, player, player_items, own_items):
		do_deal(player, player_items, own_items)

	def want_to_buy(self, answer):
		if answer is False:
			return False
		else:
			return True
```

Class Game

```
from Board import Board
from Card import Card
from Bank import Bank
from GameCard import GameCard
from Help import Help

class Game:
	"""Class Game manages the running of the game"""

	def __init__(self):
		"""Initialise all variables for game"""
		self.running = True
		self.players = []
		self.winner = None
		self.board = None
		self.size = self.weight, self.height = 800, 800
		self.cards = []
		self.help = None
```

```
      def add_player(self, player):
            """Add a player to a game"""
            self.players.append(player)

      def start_game(self):
            """start the game and enable the window"""
            while self.running is True:
                  self.find_first_player()

      def initialise_help(self):
            """Initialises help menu"""
            self.help = Help("rule.txt")


      def initialise_board(self):
            """Initialise the board"""
            self.board = Board()

      def initialise_cards(self):
            """Initialise the cards"""
            gc1 = GameCard("Chance", "It's your birthday! Collect €10
from each player", False, "Chance", 10, self.players, 0)
            self.cards.append(gc1)


      def initialise_bank(self):
            """Initialise bank"""
            bank = Bank()

      def find_first_player(self):
            """Find player to make first move"""
            highest = 0
            first = None
            for player in self.players:
                  roll = player.die.roll()
                  if roll > highest:
                        highest = roll
                        first = player
            return first
      def get_winner(self):
            """Get the winner"""
            return winner

      def quit_game(self):
            """quit the game"""
            self.running = False

      def get_help():
            """Get help"""
            self.help.show_rules()

      def eliminate_player(self, player):
            """Remove player from game"""
            self.players.remove(player)
            player.stop_playing()
```

Class Card

```
from Bank import Bank

class Card:

      def __init__(self, type, name, description, keep):
            """Initialises the card"""
            self.type = type
            self.name = name
            self.description = description
            self.keep = keep

      def show_card(self):
            return self.name + " " + self.description

      def put_back(self):
            Bank.place_card(self)

      def take_card(self, player):
            self.player.inventory.place_card(self)
```

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

Class CardSquare

```
from Game import Game
from Square import Square
import random

class CardSquare(Square):

      def __init__(self, name, location):
            super().__init__(name, location)

      def get_card(self):
            index = random.randint(0,109)
            card = Game.cards[index]
            card.show_card()
```

Class FreeParking

```
from Square import Square

class FreeParking(Square):


      def __init__(self):
            super().__init__(name, location)

      def get_location(self):
            return self.location
```

Class GameCard

```
from Card import Card

def GameCard(Card):

      def __init__(self, name, description, keep, type, cost, players, moves):
            """Initialises the GameCard"""
            super().__init__(name, description, keep) #Takes attributes from parent class
```

```python
        self.type = type #Classification of card, can be (gain money, lose money, move player, jail card)
        self.cost = cost
        self.other_players = self.players
        self.moves = self.moves

    def get_instruction(self):
        """Gets the card instruction"""
        return self.description

    def get_type(self):
        """Gets the type of card, chance / community chest"""
        return self.type

    def get_cost(self):
        """Gets the cost of the card if it has one"""
        return self.cost

    def get_moves(self):
        """Gets the number of moves on the if it has them"""
        return self.moves

    def follow_instruction(self, player):
        """Finds the instruction of the card"""
        if get_type() == "Gain":
            self.gain_money(player)
        elif get_type() == "Lose":
            self.lose_money(player)
        elif get_type() == "Move":
            self.move_player(player)
        elif get_type() == "GetOutOfJail":
            self.get_jailfree_card(player)

    def get_players(self):
        """Gets any other players involved with the card"""
        return self.other_players

    def players_is_empty(self):
        """Checks if there are any other players involved"""
        return len(self.other_players) == 0
```

```
      def gain_money(self, card_player):
            """Gives the player the money specified on the card"""
            #If other players are involved
            if not self.players_is_empty():
                  #For each player
                  for player in self.get_players():
                        #Take money out of their inventory
                        player.inventory.withdraw(self.get_cost())
                        card_player.inventory.deposit(self.get_cost()) #Give it to the
player with the card
            #Otherwise just give the player the money
            else:
                  Bank.withdraw(self.get_cost())
                  card_player.inventory.deposit(self.get_cost())

      def lose_money(self, card_player):
            """Takes away the money specified on the card"""
            #If other players are involved
            if not self.players_is_empty():
                  #For each player
                  for player in self.get_players():
                        #Withdraw the money from the player with the card
                        card_player.inventory..withdraw(self.get_cost())
                        player.inventory.deposit(self.get_cost()) #Deposit the money in
the players wallet
            #Otherwise, just take put the money
            else:
                  card_player.inventory.withdraw(self.get_cost())

      def move_player(self, card_player):
            """Move the player a certain amount of spaces"""
            card_player.move(self.get_moves())

      def get_jailfree_card(self, card_player):
            """Get a get out of jail free card"""
            card_player.inventory.place_card(self)
```

Class GamePiece

```
class GamePiece:
```

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

```
      """Represents a game piece"""

      def __init__(self, xpos, ypos):
            """Initialises game piece"""
            self.xpos = xpos #x position on screen
            self.ypos = ypos #y position on screen
            self.sprite = "" #sprite image

      def update_positions(self, x, y):
            """Update position on screen"""
            self.xpos = x
            self.ypos = y
```

Class Go

```
from Square import Square

class Go(Square):

      def __init__(self, location):
            """Initialise Go Square"""
            self.location = location

      def collect_money(self, player):
            player.inventory.deposit(200)
```

Class Help

```
class Help():
      """Represents a help class"""

      def __init__(self, filename):
            """Initiates the help class, takes in a file name of game rules"""
            self.filename = filename
            self.rules = []

      def show_rules():
            """returns the formatted rules"""
            return "\n".join(self.rules)

      def read_rules(self):
            """Reads the rules from the file and inserts them in a list"""
            with open(self.filename) as f:
                  self.rules = f.readlines()
```

Class Jail

```
from Square import Square


class Jail(Square):
    """Represents a jail square"""

    def __init__(self, location):
        """Initialises a jail square"""
        self.prisoners = [] #List of prisoners in jail
        self.location = location #Location of jail

    def get_location(self):
        """Get the location of the jail"""
        return self.location

    def get_prisoners(self):
        """Return the prisoners in the jail"""
        return "-".join(self.prisoners)

    def contains(self, prisoner):
        """Check if a prisoner is in jail"""
        if prisoner in self.prisoners:
            return True

        else:
            return False

    def remove(self, prisoner):
        """Free a prisoner from jail"""
```

```
            if self.contains(prisoner):
                  self.prisoners.remove(prisoner)
            else:
                  return None

      def add_prisoner(self, prisoner):
            """Lock another prisoner up"""
            if not self.contains(prisoner):
                  self.prisoners.append(prisoner)
            else:
                  return None
```

Class Property

```
class Property:

      def __init__(self, title_deed_card, location, owned, owner):
            """Initiate Property"""
            self.title_deed_card = title_deed_card #Title Deed Card
            self.location = location #Location of property
            self.houses = 0 #Count of houses on property
            self.hotels = 0 #Count of hotels on property
            self.owned = owned #Boolean Value representing if the property if owned
            self.owner = owner #Player who owns the property or None

      def get_location(self):
            """Get the location of the property"""
            return self.location

      def get_houses(self):
            """Get the houses count"""
            return self.houses

      def get_hotels(self):
            """Get the hotels count"""
            return self.hotels

      def buy_house(self, player):
            """Buy a house for the property"""
            #If the hotel count is less than 4, then the player can purchase
            if self.get_houses() < 4:
                  player.inventory.withdraw(self.title_deed_card.get_house_price())
#Charge the player
                  Bank.deposit(self.title_deed_card.get_house_price()) #Deposit the
money in the bank
                  self.house += 1 #Increment the house count
            #Otherwise, the player cannot purchase
            else:
                  return False

      def sell_house(self, player):
            """Sell the house"""
            #If the houses count is 0, then they can purchase
            if self.get_houses() == 0:
                  Bank.withdraw(self.title_deed_card.get_house_price()) #Take the
money out of the bank
                  player.inventory.deposit(self.title_deed_card.get_house_price())
#Deposit the money in the players wallet
                  self.house -= 1 #Decrement the house count
            #Otherwise, the user can't buy
            else:
                  return False

      def buy_hotel(self, player):
            """Function to buy a hotel"""
            #If there are no hotels on the property, the player can buy
            if self.hotel == 0:
                  player.inventory.withdraw(self.title_deed_card.get_hotel_price())
#Charge the player
                  Bank.deposit(self.title_deed_card.get_hotel_price()) #Deposit the
money from the bank
                  self.hotel += 1 #Increment the hotel count

      def sell_hotel(self, player):
            """Sell a hotel"""
            Bank.withdraw(self.title_deed_card.get_hotel_price()) #Withdraw the
money from the bank
            player.inventory.deposit(self.title_deed_card.get_hotel_price())
#Deposit the money in the players wallet
            self.hotel -= 1 #Decrement the hotel count
      def buy_property(self, player):
            """Buy a property for a player"""
            #If the property is unowned
            if self.owned is False:
                  player.inventory.withdraw(self.title_deed_card.get_price())
#Charge the player
                  Bank.deposit(self.title_deed_card.get_price()) #Deposit the money
                  self.set_owner(player) #Set the player as the owner
                  self.set_owned(True) #Set the owned value to True
                  self.player.inventory.place_property(self.title_deed_card) #Place
the card in the player's inventory
            else:
                  self.pay_rent()
                  return False

      def pay_rent(self, player, owner):
            """Make the player pay rent"""
            self.player.inventory.withdraw(get_rent()) #Withdraw rent from the
players wallet
            self.owner.inventory.deposit(get_rent()) #Deposit the rent in the
owners wallet

      def get_rent(self):
            return self.title_deed_card.get_rent_price()

      def built_evenly(self):
            pass

      def check_ownership(self):
            return self.owned

      def ask_to_buy(self, player):
            if player.wants_to_buy(answer):
                  return True
            else:
                  return False

      def set_owner(self, player):
            self.owner = player
      def set_owned(self, own):
            self.owned = own
```

Class Station

```
from TitleDeedCard import TitleDeedCard
from Square import Square
from Bank import Bank

class Station:

      def __init__(self, name, owned, owner, cost, title_deed_card,
location):
            self.name = name
            self.owned = owned
            self.owner = owner
            self.cost = cost
            self.title_deed_card = title_deed_card
            self.location = location

      def get_price(self):
            return self.cost

      def get_location(self):
            return self.location

      def buy_station(self, player):
            if check_ownership() is False:
                  player.inventory.place_card(self.title_deed_card)
                  player.inventory.withdraw(self.cost)
                  Bank.deposit(self.cost)
                  set_owned(True)
                  set_owner(player)
            else:
                  pay_rent(player)

      def pay_rent(self, player):
            player.withdraw(self.title_deed_card.rent)
            Bank.deposit(self.title_deed_card.rent)

      def check_ownership(self):
            return self.owned

      def set_owner(self, player):
            self.owner = player

      def set_owned(self, own):
            self.owned = own
```

Class Tax

```
from Square import Square

class Tax(Square):

      def __init__(self, type, name, location, price):
            """Initialise tax square"""
            super().__init__(type, name, location)
            self.price = price

      def get_location(self):
            """Get the location of the square"""
            return self.location

      def get_price(self):
            """Get the tax cost of the square"""
            return self.price

      def pay_tax(self, player):
            """Play the tax of the square"""
            player.inventory.withdraw(self.get_price()) #Withdraw tax
from the player's wallet
            Bank.deposit(self.get_price()) #Deposit the tax in the bank
```

Class TitleDeedCard

```
from Card import Card
```

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

```python
class TitleDeedCard(Card):

      def __init__(self, name, description, keep, cost, mortgage_price, rent, 
grouping, colour):
            super().__init__(self, name, description, keep)
            self.cost = cost
            self.mortgage_price = mortgage_price
            self.rent = rent
            self.grouping = grouping
            self.colour = colour

      def get_price(self):
            return self.cost

      def get_grouping(self):
            return self.grouping

      def get_mortgage_price(self):
            return self.mortgage_price

      def get_rent_price(self):
            return self.rent
```

Appendix

Team Meetings

Meeting 1

31st October
Attendees: Aifric, Comfort, Rachel, Aine
Topic: Sequence diagrams
Minute Taker: Rachel
Leader: Aine
Deputy: Aifric

- For today's meeting, we wanted to get all the hard bits done first such as the object diagram, sequence and its sub-diagram and the state diagram.

● We looked and discussed the object diagrams that Afric had made and then decided it was very good, we discussed how the sequence diagram was going to be implemented using the object diagrams.

● We revised our previous use case and discussed on how the design should look like, at first we were all confused on how it should be designed but after looking at various examples and the notes on loop we finally decide on what use case should be used for the sequence diagram and the sub-diagram.

● We drew out the sequence diagram on paper to understand the flow better and make functions that can connect classes accurately.

Meeting 2
8th November
Attendees: Aifric, Comfort, Rachel, Aine, Mahjabeen
Topic: UI Mockups
Minute Taker: Aifric
Leader: Rachel
Deputy: Aine

- We focused on the UI mockups for this meeting. We designed every screen on paper first and then started to translate it to the screen.
- It was difficult to design because we decided to use pygame which imposes a lot of limitations on the UI and user interactions. We couldn't include any buttons which was difficult.
- Jabeen emailed Renaat to enquire if we should include all our mockups in the assignment or just a few and we were told 3 would do.
- We created the start screen and the general game screen with Lucid Chart.
- We decided Aifric will design the end screen and perhaps give a few 'You Win' endings so the team can choose the best one.
- We are going to decide the last mockup and start the communication diagrams on Tuesday.

Sprint Burndown Charts

Legend

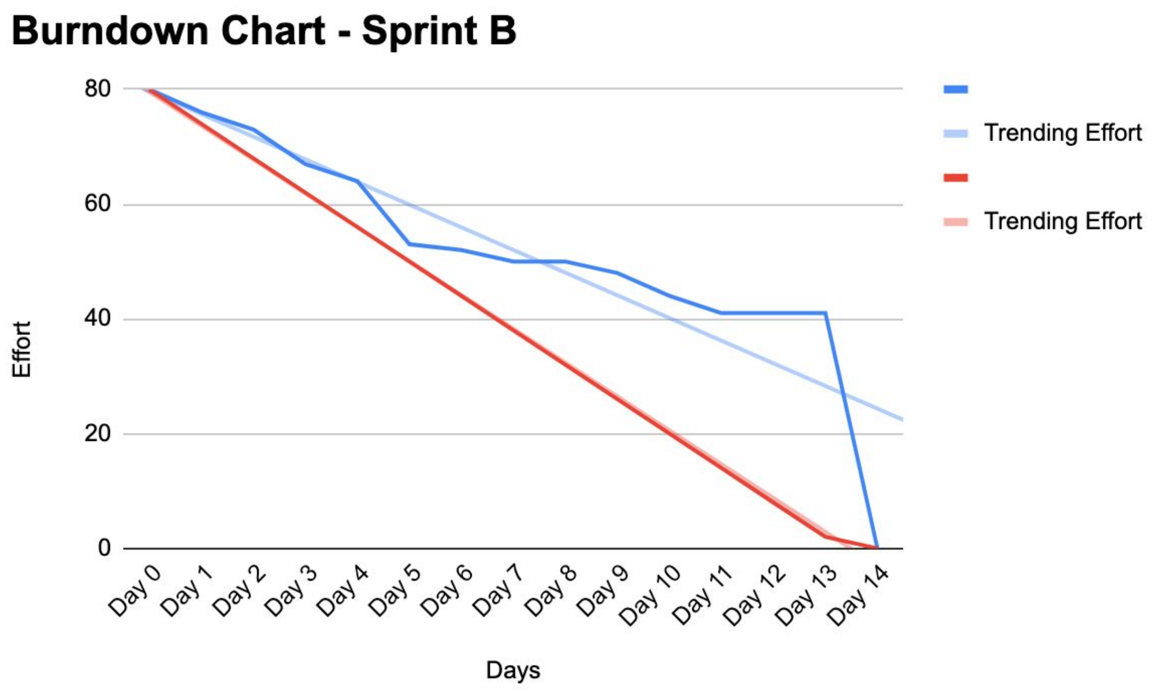
Colour	Meaning

Design and Analysis of a Game of Monopoly

Aifric Nolan, Aine McKeon, Comfort Dopamu, Mahjabeen Soomro, Rachel white

Red	Expected Burndown
Blue	Actual Burndown
Light Blue	Trending Burndown

Sprint B


Burndown Chart - Sprint B
Effort
Day 0
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Day 8
Day 9
Day 10
Day 11
Day 12
Day 13
Day 14
Days
Trending Effort
Trending Effort