KOÇ UNIVERSITY

COMP 302: Software Engineering

Team: Null Pointer

Final Report

Group Members:

Alihan Zorlu

Baran Berkay Hökelek

Berkay Barlas

Furkan Şahbaz

Tümay Özdemir

Table of Contents

Introduction and Vision

Introduction

We aim to create an online version of Ultimate Monopoly, a fan-produced version of the popular board game Monopoly.

Business Requirements

Our main driving cause was the inefficiency of physical board games and their limitations when it comes to long distances. Physical board games require constant care and maintenance, and their pieces may still get lost or damaged quite easily. In addition to that problem, playing a physical board game requires all players to be knowledgeable in the rules and internal workings of the game; which, most players are unfortunately not. Lastly, physical board games do not allow players from long distances to play together and require additional scheduling & meeting, which deters most people from playing most of the time. Our solution, which is creating a digital, online version of Monopoly with built-in rules not only solves all these problems, but also adds additional benefits to players.

Product Solution/Overview

In this version, everything is digital so no piece requires any maintenance or protection whatsoever. Players don't have to know all the rules of the game before playing, since every aspect is taken care of by the program. We will also add a network option to the game in order to allow people from different computers to play together, no matter how far they are. Our additional features to Ultimate Monopoly include the Message Board, a private chat room in which players can communicate during the game; and a bot player that makes random moves and posts random, mostly funny texts to the Message Board. We believe that this game will increase the amount of people who play Monopoly regularly, and draw new audiences from people who prefer playing games on the computer, and introduce them to this modern board game legend.

KOÇ UNIVERSITY

Teamwork Organization

Designing and programming in a team is very different than what we expected. It is a really big project, therefore, we divided project in to small tasks, everybody focused on programming different parts. To prevent misunderstandings we gathered and discussed project and our ideas. These discussions helped us to design better program. In addition to these, we used Git version control system extensively, we solved many conflicts and merged feature branches.

In the first week, all team members worked on reading and understanding the game rules, defining the use cases casually and detailing them a little bit. In the later weeks, generally team members always involved in the planning of the general tasks such as design patterns, discussing design issues.

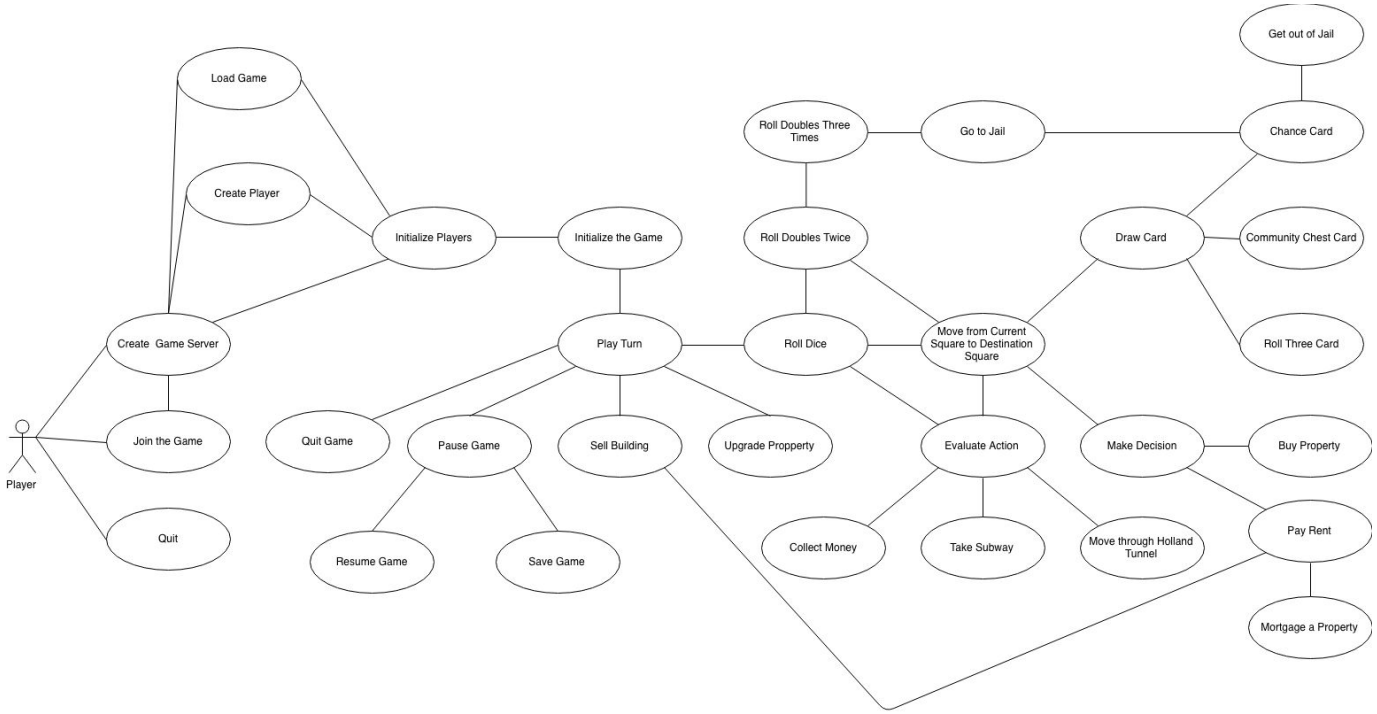
In week 4 and 5, Alihan, Tümay and Furkan worked on UML Class Diagram, Baran worked on UML Package Diagram, Berkay worked on Collaboration & Sequence Diagrams & Working on the code for network part of the game. Also, Furkan and Berkay worked on the code for player panel, Tümay worked on the code for Message Box and Animator Class and Alihan worked on the code for button panel, dice panel and adding the board image.

In week 6 and 7, Berkay worked on new use case narratives for Load/Save/Pause/Resume, Furkan worked on nonfunctional requirements for animation, Tümay worked on refining Domain models, Alihan worked on new System sequence diagrams for Load/Save/Pause/Resume and Baran worked on operation Contracts per use case.

In week 8, Berkay worked on connecting UI and domain part of generic animation, Furkan & Berkay worked on customized client display, square generation, client specific colors, some of the save/load methods, Object Stream, Tümay worked on generic animation, adding coordinates for squares and adding methods for cards, Baran worked on adding coordinates for squares, calculation of path as indexes of squares in the domain according to dice results and Alihan worked on pause/ resume/ save/ load implementation.

In week 9 and 10, Berkay worked on pre-requirements of Save-Load, Tümay worked on implementing the cards and the functionality of roll3 card square. Additionally, Tümay worked on JUnit testing of GameEngine class. Furkan and Baran worked on customized client display, square generation, client specific colors, some of the save/load methods, Object Stream, Baran worked on adding Javadoc and Alihan worked on JUnit testing of MoneyController class. Also, worked on SaveLoadController.

UML Use Case Diagram



Use Cases

Use Case UC1: Initialize the Game

Scope : Monopoly game initialization
Level : User-goal
Primary Actor : Founder of Server
Stakeholders and Interests :
Founder → Wants to find enough players to play the Monopoly Game
Player → Wants to play the Monopoly Game
Observer → Players who have joined the server should be displayed.
Preconditions : The number of players who want to join the game is greater than or equal to the maximum of players that will be declared (When,there are sufficient number of players).
Success Guarantee (or Postconditions) : The players have successfully connected to the game and the order that they will play is determined.
Special Requirements : Since the game has not started yet, the board is not displayed. In the monitor of the founder, UI for selecting the number of players and starting the game is presented. UI is visible from at least 0.5 meters. Since there may be problems with the network

connection during the initialization of the game, the system can have delays of up to 5 seconds for 90% of the time. Players are assumed to know English.
Frequency : Occurs exactly once during the game.
Main Success Scenario
1- The founder creates the network and declares the maximum number of players that will play in this server.
2- Other computers join the network and choose number of players.
3- Server initiates the game.
4- Starting with the first player joined, each player throws the dice. The player with the highest total starts the play.
Extensions
*a- A bot player takes the control of the player if the players quits the game:
    1- If all of the players or the founder quit the game, the game is cancelled.
*b- If any problem related with the network occurs, the game is cancelled.
*c- At any time, the system fails:
    1- The game is cancelled.
    2- A message is displayed to each of the players.
3a- If any two or more players have equal roll sum, they compete with each other again, the procedure is repeated until a player rolls a higher value.

Use Case UC2: Joining the Game

Scope : Game Connection
Level : User goal
Primary Actor : Player
Stakeholders and Interests :
Player → Wants to establish a connection to the game so that he/she can join it.
Founder → Wants the game server to be set up and ready to be initialized.
Observer → Current situation of the connected players should be displayed to the observers.
Preconditions :
The founder has established the game server.
The game has not started yet and there's still room to join.
Success Guarantee (or Postconditions) :
Player has successfully connected to the game server.
Main Success Scenario:
1- Player provides the IP address of the game he/she wants to join.
2- Player's join request is delivered to the host.
3- The connection between player and server has been established.
4- Number players that will play on this client is set

5- Player waits until the game is initialized.

Extensions:

*a- If the connection breaks while connecting to the game, player will not be connected.

3-a- If the server is not available, or it has reached the maximum number of players declared by the founder, the player's request will be turned down.

Special Requirements :

The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.

Frequency: Once for each player

Use Case UC3: Communicating with Server

Scope : Server, Client

Level : Subfunction

Primary Actor : Client

Stakeholders and Interests :

Client → Wants to send request to server and take response.

Server → Wants to manage and control communication of all clients

Preconditions :

Game is initialized with a working server.

Client is connected to server.

Success Guarantee (or Postconditions) :

Client sent request to server. Server is received and evaluated request.

Main Success Scenario:

1- Client receives messages from game

2- Client makes a request to server according to player's decision.

3- Server takes the response and sends responses to all clients according to request.

Extensions:

1-a- Player waits too much or server does not take any response from client:

1. Server sends a timeout message.

Special Requirements :

Clients responses should be checked during the time limit.

Frequency : Every time a player takes action.

Use Case UC4: Server Plays on behalf of a Player when Time Limit is Exceeded

Scope : Current Player's turn.

Level : Subfunction

Primary Actor : Server

Stakeholders and Interests :

Client → Receives responses from the server.

Server → Wants to keep the game going without further delays.

Preconditions :
Current player has not taken any actions during the time limit provided by the server.
Success Guarantee (or Postconditions) :
Current player's turn has ended and the next player's turn starts.
Main Success Scenario:
1- Server waits for a response from the client.
2- If no responses have been received after a while, the server assigns a bot player for this turn and acts as if current player rolls the dice and acts accordingly.
3- After the turn is executed, it is passed.
Extensions:
1-a- If any response is received while waiting for the client, current player will be allowed to play.
1-b- If any responses are received after the time limit has passed, they will be ignored for this turn.
2-a- If the player will be required to make any decisions while executing the turn, the decisions will be made randomly by the bot.
Special Requirements :
Clients responses should be checked during the time limit.
Frequency : Every time a player exceeds the time limit to take an action.

Use Case UC5: Moving from Current Square to Destination Square

Scope : Current Player's Turn
Level : User goal
Primary Actor : Current Player
Stakeholders and Interests :
Observer → The current state of the game should be available to them. This includes the number of turns that have passed, the player who has the turn, the amount of money and the properties and the cards that each player has and the roll sum. Also, the movement of the player should be displayed as an animation.
Current Player → In addition to interests of the observer, the current player should also be able to roll dice.
Preconditions :
The current player has permission to roll the button and no other player has that permission.
Success Guarantee (or Postconditions) :
The position of the player has correctly changed according to the sum of the values of the dice.
Main Success Scenario
1- Current player's turn starts.
2- Dice are rolled for the current player.
3- Total result of three dice is calculated.
4- Current player moves to the destination square.
Extensions
*a- A bot player takes the control of the player if the players quits the game:
    1- If all of the players quit the game, the game is cancelled.
*b- If any problem related with the network occurs, the game is cancelled.

*c- At any time, the system fails:

1- The game is cancelled.

2- A message is displayed to each of the players.

3a- When player rolls doubles for the for the first time, player makes a move, performs the corresponding action and player rolls the dice again.

3b-When player rolls doubles for second time, player makes a move, performs the corresponding action and player rolls the dice again.

3c- Throwing Doubles for Three Times and Going to Jail (referring to the use case 6)

4a- If the player passes from a Bonus Square or Go Square, the corresponding money is added to the player's account.

4b- If the player throws a triple, the player can move to any square that the player wishes.

*a - If any time, system fails:

1 - System logs player info up to point of disconnection. When player is reconnected, his/her latest state is loaded back.

2- If player is unable to connect for 30 seconds, it is replaced by a bot player.

Special Requirements :

The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card. UI is visible from at least 0.5 meters. The system can have delays of up to 3 seconds for 90% of the time. Players are assumed to know English.

Frequency :

Occurs at least once for each turn and for each player. Occurs one more time for each rolling a double except when the double is rolled for a third consecutive time.

Use Case UC6: Throwing Doubles for Three Times and Going to Jail

Scope : Current Player's Turn

Level : Subfunction

Primary Actor : System

Stakeholders and Interests :

Observer → The state of the game should be available to them. This includes the number of turns that have passed, the player who has the turn, the amount of money and the properties and the cards that each player has and the roll sum. When the player goes to jail, all observers should be notified. Also, in the subsequent turns, when the player who is in the jail can not make a move because of being in a jail, an explanation should be displayed to all observers. This would make the game more user friendly.

Current Player → Has the same interests as the observer.

Preconditions :

Doubles have been thrown for two consecutive times already.

Success Guarantee (or Postconditions) :

The current player goes to jail.

The next player's turn starts.

Main Success Scenario:

1- Dice are rolled for current player for the third time.
2- The dice are evaluated. It is noted that values of the regular dice are the same for the third time.
3- Current player is moved to the jail.

Extensions:

3-a- If the current player has drawn a “Get out of jail free” chance card during any of its previous turns and has not used it, it may choose to use the card to get out of the jail immediately.

Special Requirements :

The board and the positions of the players are displayed. Each player’s monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.

Frequency : Depends on luck. Mathematically, it is (⅙) * (⅙) * (⅙) = 0.46 %. So it is relatively rare.

Use Case UC7: Buying a Property

Scope : Current Player’s Turn

Level : User-goal

Primary Actor : Current Player

Stakeholders and Interests :

Current Player Wants to buy the property.

Preconditions :

Player should have enough money for buying property.
The property should be unowned.

Success Guarantee (or Postconditions) :

Current player’s money is reduced as indicated by its Deed Card. The property that is bought now belongs to the current player and no other player can buy it.

Main Success Scenario

1- Current player lands on a property.
2- Whether or not the property is owned is checked.
3- If the property is not owned, current player decides on buying the property.
4- Player pays money to the bank.
5- Current player’s turn ends.

Extensions:

*a- A bot player takes the control of the player if the players quits the game:
    1- If all of the players quit the game, the game is cancelled.
*b- If any problem related with the network occurs, the game is cancelled.
*c- At any time, the system fails:
    1- The game is cancelled.
    2- A message is displayed to each of the players.
3-a-If player’s current money is less than the price of the property, player cannot buy the property, nothing happens.

Special Requirements :

The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.
Frequency : Can occur at the turn of each player. Can occur one more time for each rolling a double except when the double is rolled for a third consecutive time.

Use Case UC8: Paying Rent on a Property

Scope : Current Player's Turn
Level : Subfunction
Primary Actor : System, Current Player
Stakeholders and Interests :
System → takes the rent from current player and gives it to the owner player.
Current Player → has to pay the rent, if current player does not have enough money, current player chooses a property to sell or mortgage.
Owner player of Property → The money of the owner player is updated by being added the rent.
Preconditions :
Property should be owned by another player.
Property should not be mortgaged.
Success Guarantee (or Postconditions) :
The player who owns the property gains money equal to the amount that is indicated by its Deed Card. The same amount of money is taken from the player who lands on that property.
Main Success Scenario :
1- Current player lands on a property.
2- Whether or not the property is owned is checked.
3- If the property is owned, current player pays the rent.
4- Current player's turn ends.
Extensions:
3-a-If current player does not have enough money

1. If player owns any building (house, hotel or skyscraper), player sells some of its buildings until player has enough money to be able to pay the rent.
2. 2. Else if player owns some properties, railroads and transition stations or any utility, player sells some of its properties, railroads and transition stations or any utility until player has enough money to be able to pay the rent.
3. Else player goes bankrupt, and is forced to quit the game.

Special Requirements :
The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.
Frequency :

Rent is played when a player lands on a square that is owned by another player. Frequency rate is never higher than the number of turns. Frequency becomes greater when most of the properties are owned by the players.

Use Case UC9: Upgrading a Property

Scope : Monopoly game
Level : User goal
Primary Actor : Selected player
Stakeholders and Interests :
Selected player → building a house or hotel or skyscraper.
Preconditions :
The property which is wanted to be upgraded should be owned by the selected player and should not be mortgaged.
The selected player should satisfy majority ownership to be able to build house and hotel and satisfy monopoly to be able to build skyscraper.
Success Guarantee (or Postconditions) :
Money of the selected player decreases according to the price of the building and rent of the chosen property increases.
Main Success Scenario:
1- During any point in the game, a player decides to improve a property he/she owns.
2- It is checked whether player owns the majority of properties of the color of the property he/she wants to improve.
3- Player decides on which type of improvements he/she wants to make on the property.
4- Player pays the required amount of money to bank.
Extensions:
1-a- If the player is currently in jail, he/she cannot make any improvements.
2-a- If player does not own the majority of properties of the color of the property he/she wants to improve, he/she will not be allowed to make any improvements.
3-a- If the player has not built anything on any of the properties of the color group he/she has the majority ownership on, he/she can build a house on the property it chooses.
3-b- If the player has built 4 houses on each of the properties of the color group he/she has the majority ownership on, he/she can build a hotel to the property he/she chooses.

1. If player has recently purchased the final property of a the color group he/she has the majority ownership on, and has a Monopoly, he/she cannot make any improvements on that group until the new property has equal level of improvement as the other players.

3-c- If the player has a Monopoly and has built a hotel on each of the properties of the color group he/she has a Monopoly on, a skyscraper can be built on the property he/she chooses.

4-a- If the player does not have enough money to complete the purchase, he/she is warned that this improvement cannot be made.

Special Requirements :

The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.

Frequency : Whenever a player wants to improve a property.

Use Case UC10: Mortgaging a Property

Scope : Current Player's Turn

Level : User-goal

Primary Actor : Current Player

Stakeholders and Interests : Current Player → mortgages the property.

Preconditions : The player already owns a property that he/she can mortgage.

Success Guarantee (or Postconditions) :

The property is still owned but its status rather becomes mortgaged. Other players can still not buy that property directly but it can be foreclosed by using the relevant Action Cards.

Main Success Scenario:

1-During any point in the game, a player is allowed to mortgage one of or many of her/his properties, makes her/his property mortgaged.

2-If there is no building on the specified property, player mortgages her/his property and gets her/his money.

3-Since the property is mortgaged, player is not going to be able to collect rent over this property by anyone landing on this property.

4-Until the player removes the mortgage on the property, land will stay mortgaged.

Extensions:

2-a-If property has buildings over it, first buildings should be sold by selling a building use case and money is taken back, and now player can mortgage the property.

Special Requirements : The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card. Additionally, the mortgaged property is marked on the property display of the player.

Frequency : The number of times that mortgaging occurs is always less than the sum of the number of times buy occurs and dismortgage occurs.

Use Case UC11: Selling a Building

Scope : Current Player's Turn

Level : User-goal

Primary Actor : Current Player

Stakeholders and Interests : Current Player → sells a house, hotel or skyscraper.

Preconditions : The player already owns a property that contains the building he/she wants to sell.
Success Guarantee (or Postconditions) :
Current player's money increases.
Level of the property that current player sold a building from changes.
Main Success Scenario:
1- During any point in his/her turn, current player decides to sell a building he/she has built.
2- Current player sells the property of his/her choice to the bank.
3- Current player receives half of the amount of money he/she spent on making that building, and improvements that are equal to one level less than his/her previous level.
Extensions:
2-a- Current player must sell his/her buildings within a certain color to the bank evenly.
3-a- If the bank doesn't have any replacements left to perform a transition from a level to another, current player is served 2 options:

a. To not sell anything back.
b. To sell everything back until the bank is available to perform the transition.

Special Requirements :
The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.
Frequency : Whenever a player who has at least one building wants to sell a building.

Use Case UC12: Paying Utility Rent

Scope : Current Player's Turn
Level : User goal
Primary Actor : System, Current Player
Stakeholders and Interests :
System → takes the rent from current player and gives it to the owner player.
Current Player → pays the rent of the landed utility.
Owner of the property → takes the rent.
Preconditions :
The current player lands on the utility rent square.
Utility rent square is already owned by a player.
Success Guarantee (or Postconditions) :
The current player pays money to the owner of the utility rent by an amount that is calculated by using the number of utilities that are owned by the utility rent owner.
Main Success Scenario :
1- Player rolls all three dice.
2- The total of all three dice including speed dice is calculated and Mr. Monopoly and Bus Icon are counted as zero.
3- Total number of utilities owned by the renter of the square which landed by the player is calculated.

4- The rent paid by the player according to how the player landed on the square, the total number of utilities owned by the renter and the dice value.

Extensions :

4-a- 1- If the player landed on the square by dice value

1- If one utility is owned, rent is 4x amount shown on dice.
2- If two utilities are owned, rent is 10x amount shown on dice.
3- If three utilities are owned, rent is 20x amount shown on dice.
4- If four utilities are owned, rent is 40x amount shown on dice.
5- If five utilities are owned, rent is 80x amount shown on dice.
6- If six utilities are owned, rent is 100x amount shown on dice.
7- If seven utilities are owned, rent is 120x amount shown on dice.
8- If all eight utilities are owned, rent is 150x amount shown on dice.

2- If the player landed on the square by chance card or the like which takes a player to nearest utility, player rolls only two regular dice and pay only 10x amount shown on the dice if the card says so.

Special Requirements : The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.

Frequency : Occurs strictly less than the number of turns. Can only happen when a player lands on a utility rent square and that utility rent is owned by another player.

Use Case UC13: Landing on Railroads & Transit Stations

Scope : Current player's turn

Level : Subsystem and User goal

Primary Actor : Current player

Stakeholders and Interests :

System → decides whether or not the current player changes the track to a higher or lower one or staying on the current track.

Current player → buys the railroad and transit station if it is unowned.

Preconditions :

Current player landed on the railroad and transit station.

Success Guarantee (or Postconditions) :

The owner of the railroad and transit station is changed if current player decided to buy it.

Main Success Scenario:

1- Current player lands on a railroad and transit station.
2- Checks whether or not the railroad and transit station is owned.
3- If the railroad and transit station is not owned, player decides to buy the railroad and transit station.
4-Player pays the price of buying the railroad and transit station.
5- If total value of dice is odd, player continues on her/his track.

Extensions:

1-a-If current player does not land on a railroad and transition station however, if any chance action card directs player to go to another track of monopoly game and this railroad and transition station is the closest railroad allowing player to go to specified track, then player should also play this railroad and transition station.
3-a-If the railroad and transit station is owned
1-If player is the owner of it
1-If player wants to buy a train depot, player decides to buy a train depot.
2-Else if player does not want to buy a train depot, nothing happens.
2-If some other player owns the railroad and transit station, the current player pays its rent. Money of the current player and owner player are updated.
4-a-1- If player has enough money to be able to buy a train depot, player buys a depot and the rent of the train and transition station is doubled and updated.
2-If player does not have enough money, nothing happens.
5-a-If total value of dice is even, player goes to opposite track and gets ready to continue in the clockwise.
Special Requirements :
The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.
Frequency : Whenever a player lands on the railroad and transit station on the game board.

Use Case UC14: Landing on an Action Square

Scope : Current player's turn
Level : Subsystem
Primary Actor : Current player
Stakeholders and Interests :
Current player → plays whatever written on the action card.
Other players → if the action card drawn is related about some other players, some other players are also affected.
Preconditions :
Not being in jail and landing on an action square by dice value.
Success Guarantee (or Postconditions) :
According to whatever the action card changes about the current player or other player, current player's or other player's state changes.
Main Success Scenario:
1- Current player lands on an action square.
2-Player draws a card from action cards deck.
3-Play action card.
Extensions:
3-a-If "go to jail" card is drawn, player goes to jail directly without collecting any money.
Special Requirements :

The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards and other relevant information such as playing an action card.
Frequency : Whenever current player lands on the action square on the game board.

Use Case UC15: Getting Out of Jail

Scope : Current player's turn
Level : System
Primary Actor : Current player
Stakeholders and Interests :
Current Player -> Changes their state from "in jail" to "free".
Observer -> Current player's status, money and card info is distributed to other players.
Preconditions :
Current player is in jail.
Success Guarantee (or Postconditions) :
Current player gets out of jail.
Main Success Scenario:
1 - System gives the available options to player.
2 - Player pays the required money.
3 - System changes the current player's status from "in jail" to "free".
4 - Player rolls the dice.
5 - System returns the face value.
Extensions:
1-a- Player is in jail for 3 turns and failed to roll doubles. System doesn't show options, it directly deducts the required money from current player and moves to step 4.
1-b- If the player is in jail for less than 3 turns, he/she may choose to try rolling dice. In this case, skip to step 4. If it is doubles, execute step 3 and continue turn (Player won't get a second turn after doubles in this case).
2-a- Player uses "Get Out of Jail Free" card instead of money.
Special Requirements :
The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards.
Frequency : Whenever a player is in jail and can get out of jail.

Use Case UC16: Playing Mr Monopoly and Buying

Scope : Current player's turn
Level : User-goal
Primary Actor : Current player
Stakeholders and Interests :

Current player
Preconditions :
Mr monopoly should land on the speed die.
Success Guarantee (or Postconditions) :
Current player's money decreases and the property's owner changes to the current player.
Main Success Scenario:
1- Prevent all players' upgrading a property and trading.
2- If there are some unowned properties and player can reach that property square by railroads and transition stations if the property and the player are in different layers, player moves to the closest unowned property.
3- If player has enough money, player buys the unowned property.
4-Player's turn is over and all the other players can now upgrade their properties or trade if they wish to do so.
Extensions:
2-a-If there are some unowned properties in a different layer than that player is on
    1-If the player rolled totally an even number, player can change layers, player changes the layer and goes to the unowned property.
    2-If player rolled totally an odd number, player cannot change her/his track, so stays where she/he is and nothing happens.
3-a- If player does not have enough money, player cannot buy the property and nothing happens.
Special Requirements :
The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.
Frequency : Whenever the current player rolls Mr Monopoly on the speed die.

Use Case UC17: Playing Mr Monopoly and Paying Rent

Scope : Current Player's Turn
Level : Subfunction, Current Player
Primary Actor : Current Player
Stakeholders and Interests :
System → takes the rent from current player and gives it to the owner of the property.
Current Player → if she/he does not have enough money, she/he sells or mortgages.
Owner of the Property → takes the rent and so in the end, money of the owner of the property increased.
Preconditions :
Speed die value is rolled, and value of speed die is Mr. Monopoly.
Success Guarantee (or Postconditions) :
Money of current player and owner of the property changes.
Main Success Scenario:
1- Prevent all players' upgrading a property and trading.

2- If there is no unowned property, player moves to the closest owned property.
3- If player has enough money, player pays the rent of the owned property.
4-Player's turn is over and all the other players can now upgrade their properties or trade if they wish to do so.

Extensions:

2-a-If there are some unowned properties in a different layer than that player is on

1-If the player rolled totally an even number, player can change layers, player changes the layer and goes to the unowned property.

2-If player rolled totally an odd number, player cannot change her/his track, so stays where she/he is and nothing happens.

3-a- If player does not have enough money.

1- If player has buildings, player sells some of its buildings until the player is able to pay the rent.

2-Else if player has properties, player sells some of its buildings until the player is able to pay the rent.

3-Else player goes bankrupt.

Special Requirements :

The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.

Frequency : Whenever Mr Monopoly is rolled and there are no accessible unowned properties on the board.

Use Case UC18: Playing Hurricane Card

Scope : Current player's turn
Level : User level
Primary Actor : Player
Stakeholders and Interests :
Player → Wants to use the card
Observer → Display the effects of this action
Preconditions : Current player drew a Hurricane card.
Success Guarantee (or Postconditions) : One of the players (selected by current player) has either one of his/her houses removed from each property of a certain colour group; or has one of his depots/cab stands from each railroad/cab company he/she owns. Skyscrapers removed are converted to 4 houses.
Main Success Scenario:
1 - System presents available players to current player.
2 - Player selects one of the available players.
3 - System highlights the property groups & railroad/cab companies in which selected player owns a property with a house house or depot/cab stand on a railroad/cab company.
4 - Player selects one of the property groups or railroads/cab companies.

5 - System removes one of the houses(depots/cab stands) from the selected property group(railroad/cab company).
6 - Status of the selected player is updated.
Extensions:
1a - No player has any house or depot/cab stand anywhere. In this case, nothing happens. Current player's turn is continued as usual.
2a - Selected player uses a "Just Say No!" card. In this case, the action is reverted at the system level, and current player's turn is continued as usual.
Special Requirements :
The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.
Frequency : Whenever a hurricane action card is drawn and played.

Use Case UC19: Bot Player Playing its Turn

Scope : Monopoly Game
Level : Subfunction
Primary Actor : System
Stakeholders and Interests :
Bot Player → Wants to play it's turn.
Observer → All other players that can be affected by the bot player's decisions.
Preconditions :
-It is the bot player's turn to play and bot player is not in jail.
Success Guarantee (or Postconditions) :
-Bot player's state, property status, and money status may change depending on the decisions made during its turn.
-Other players can also be affected by the bot's actions and decisions during its turn.
Main Success Scenario:
1- Bot player's turn starts.
2- System rolls the dice for the bot player.
3- Bot player is moved according to the dice results.
4- Wherever the bot lands is evaluated, and if any decisions will be required to be made during this process, they will be made by the system.
5- Bot player's turn ends.
Extensions:
*a- If at any point in its turn, bot player goes bankrupt, it will be eliminated from the game.
Special Requirements :
Unlike the human players, bot players are not displayed anything. They play randomly. They buy/sell/mortgage properties randomly. They can have action cards but they are not displayed to them.
Frequency : Whenever it is bot player's turn.

Use Case UC20: Current Player Winning The Game

Scope : Monopoly Game
Level : User-goal
Primary Actor : Current player
Stakeholders and Interests :
Current Player → Wants other players to go bankrupt use case.
Other Player → Wants to continue playing game by not going bankrupt.
Preconditions :
There are only two players left in the game.
Success Guarantee (or Postconditions) :
The winner player is determined and a message is sent to every player that announces the winner of the player.
Special Requirements :
The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card.
Frequency : Occurs exactly once in each game.
Main Success Scenario
1-The opponent goes bankrupt, the last standing player (current player) wins the game.
Extensions
There are no extensions for this use case.

Use Case UC21: Pause the Game

Scope : Monopoly Game
Level : User-goal
Primary Actor : Current Player
Stakeholders and Interests :
Current Player → Wants to pause the game.
Other Players → Cannot play until game is resumed, since only the current player can resume the game.
System → Pauses the game.
Preconditions :
Game should be active.
Success Guarantee (or Postconditions) :
Game is paused.
Special Requirements :
All interactions are disabled.
Frequency : Occurs when any player wants to pause the game.
Main Success Scenario
1- A request to resume the game is received from the current player.
2- Game is paused for every player.
Extensions

2-a The players can still use the message box system. If
2-b If the player who is paused the game quits, game resumes.

Use Case UC22: Save the Game

Scope : Monopoly Game
Level : User Goal
Primary Actor : Current Player
Stakeholders and Interests :
Current Player → Wants to save the game.
System → Saves the game.
Preconditions :
Game is paused.
Current player is not a bot player.
Success Guarantee (or Postconditions) : The current state of the game is saved, which means the amount of money /properties/ cards that they have … etc. .
Special Requirements : The board and the positions of the players are displayed. Each player's monitor displays the same information about the turn numbers, money amounts, properties, action cards. However, the current player can also see buy/ sell/ upgrade options and other relevant information such as playing an action card. A message that shows game is saved should be visible by players.
Frequency : Occurs as frequently as one of the players clicks the save button.
Main Success Scenario
1- A request to save the game is received from the current player.
2- Game data is written to a file.
3- A message is sent to a player that notifies the player that the game is saved.
Extensions
*a- If the saving of the game is failed due to a system failure, the saving of the game is repeated until the game is saved.

Use Case UC23: Resume the Game

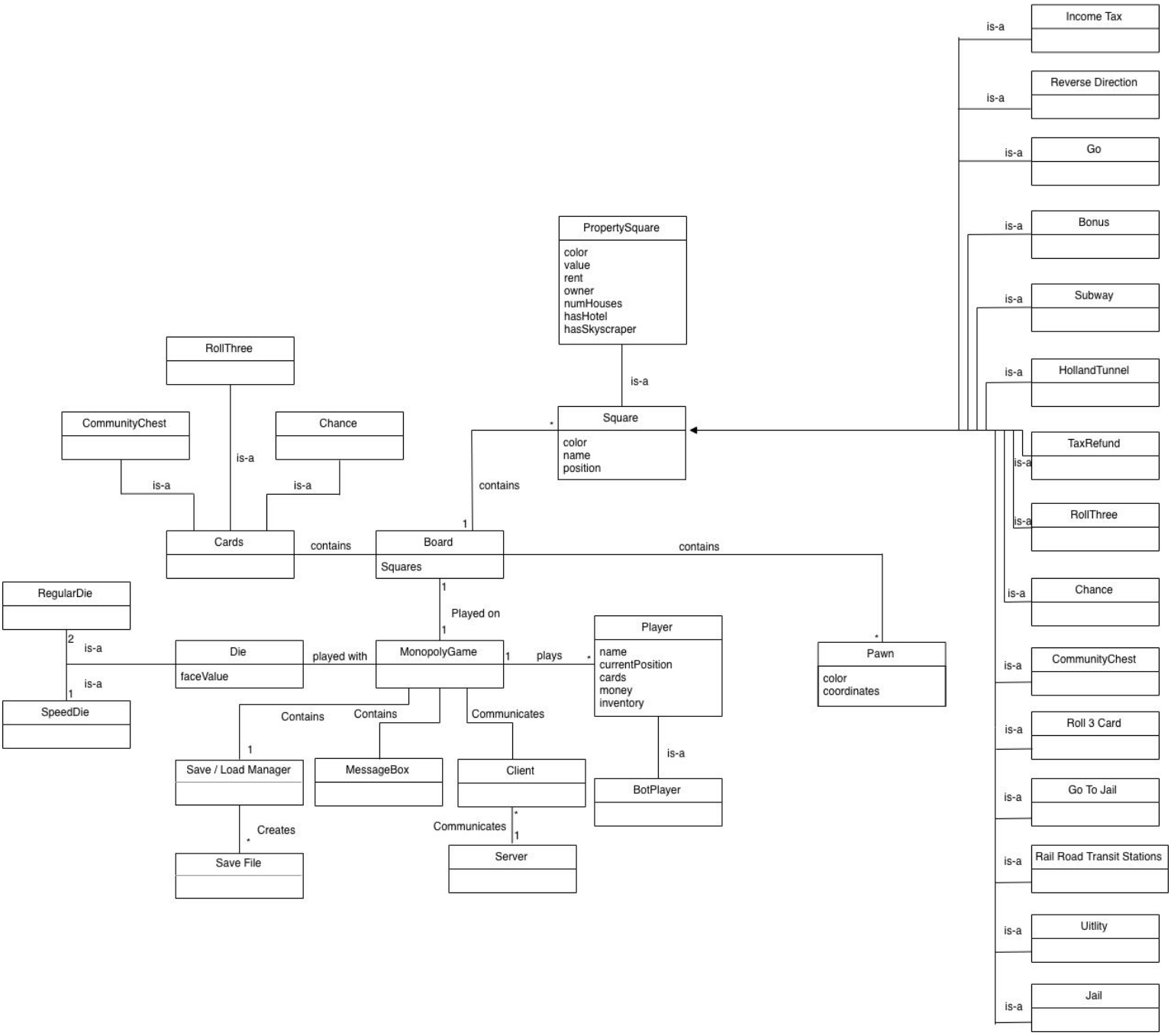
Scope : Monopoly Game
Level : User-goal
Primary Actor : Current Player
Stakeholders and Interests :
Current Player → Wants to resume the game.
Other Players → Wait to continue playing.
System → Resumes the game.
Preconditions:
Game was paused.
Current player is the player who paused the game.
Success Guarantee (or Postconditions) :

Game is resumed.
Special Requirements : All interactions are enabled.
Frequency : Occurs when any player resumes the game.
Main Success Scenario
1- A request to resume the game is received from the current player.
2- Systems resumes the game.

Use Case UC24: Load the Save File

Scope : Monopoly Game
Level : User-goal
Primary Actor : Current Player
Stakeholders and Interests :
Players→ Players want to load a game.
Preconditions :
There is a previously saved game file.
Previously saved game's players all joined and are ready to play.
Success Guarantee (or Postconditions) :
Game content is loaded and game is ready to start.
Special Requirements : The game starts from the game state that was previously saved. This includes the amount of money that each players has, their list of properties, action cards … etc and the positions of the pawns on the game board.
Frequency : Occurs when any player loads the game.
Main Success Scenario
1- Player chooses the saved game file.
2- A request to load a game is received from the current player.
3- System reads the selected game state file.
4- System loads previously saved game state on each player client.
Extensions
3-a System waits for other players to load to initialize the game.

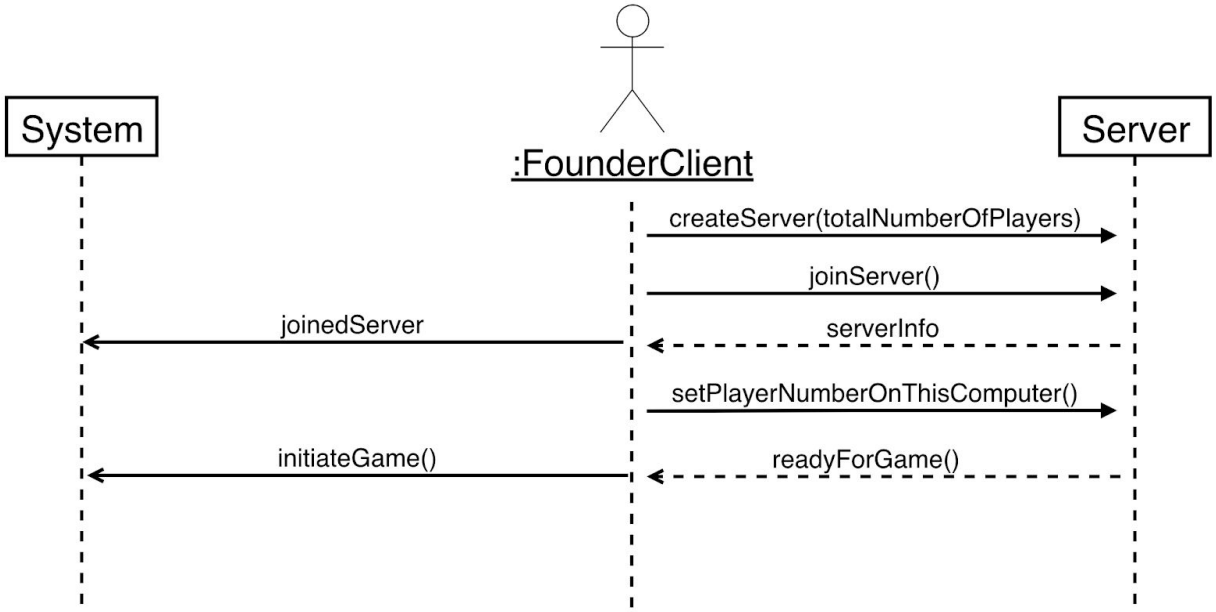
Domain Model



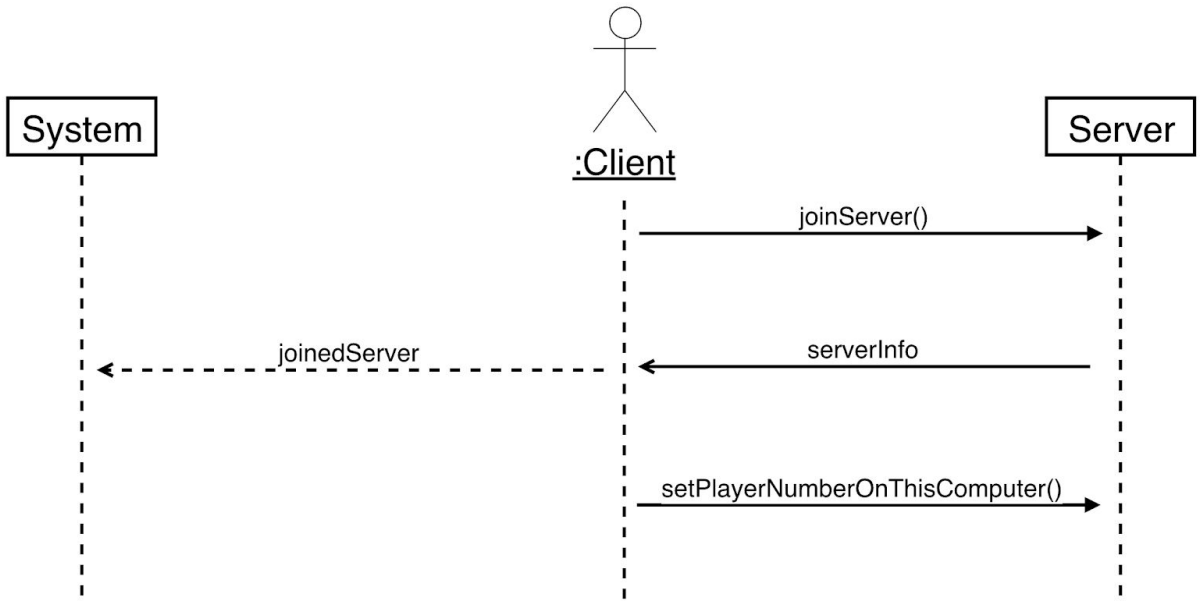
System Sequence Diagrams

For clarity, we defined System as a combination of server, client and Game System when they are not shown explicitly.

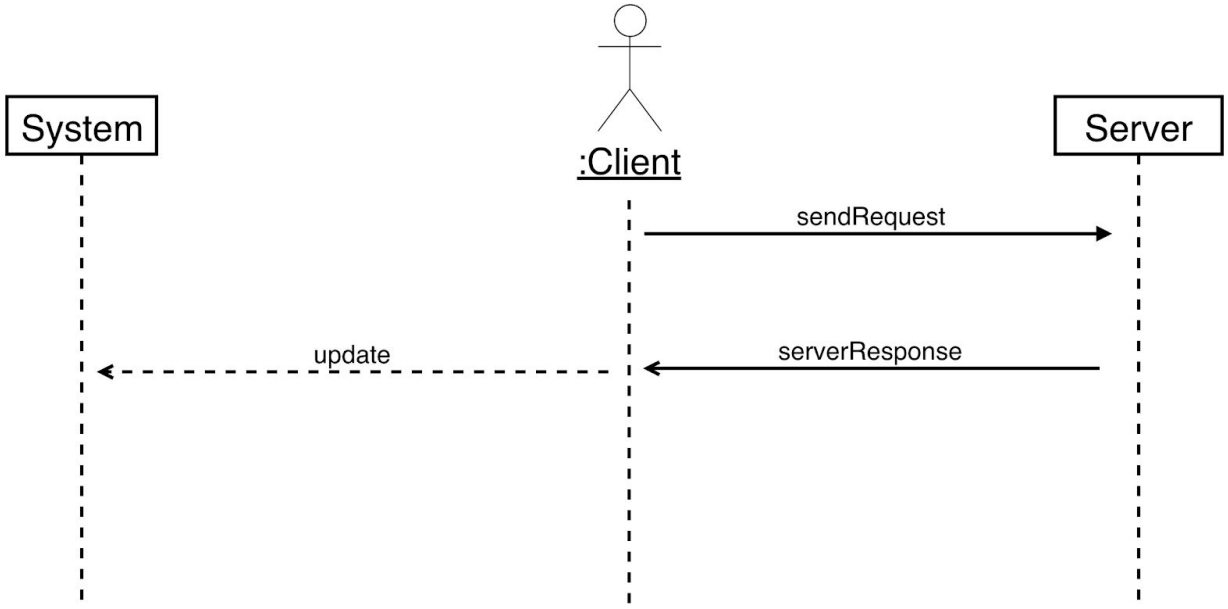
SSD1: Initialize the Game



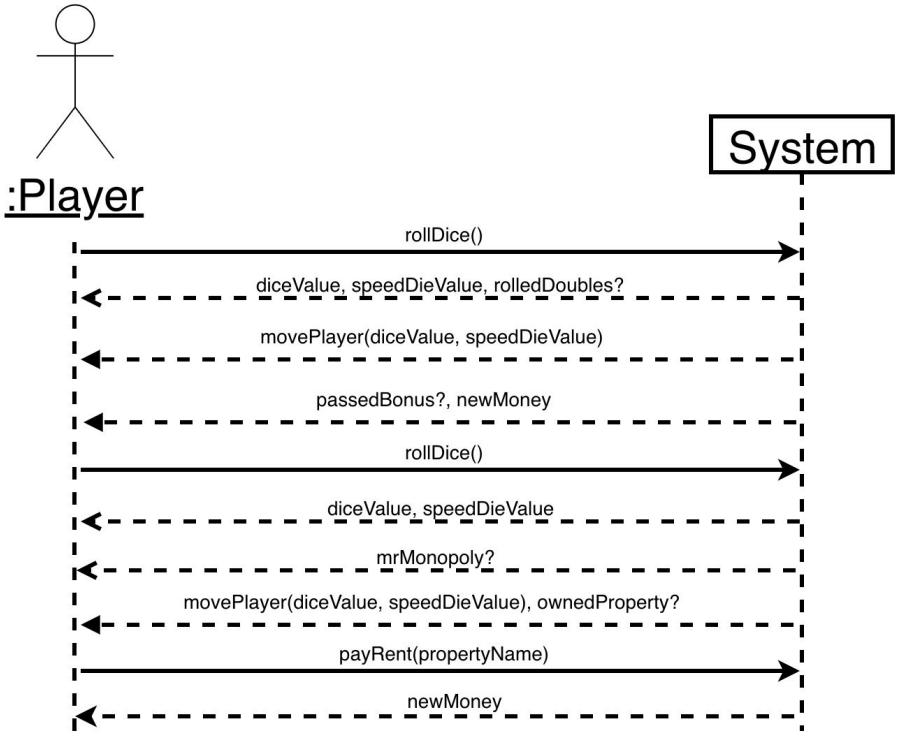
SSD2: Joining the Game



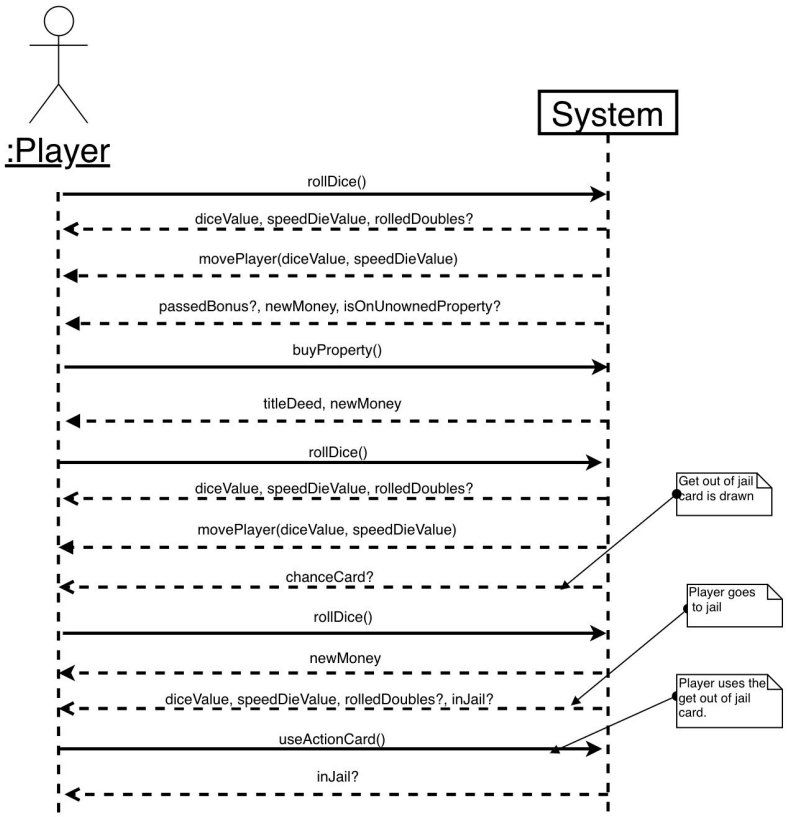
SSD3: Communicating with Server



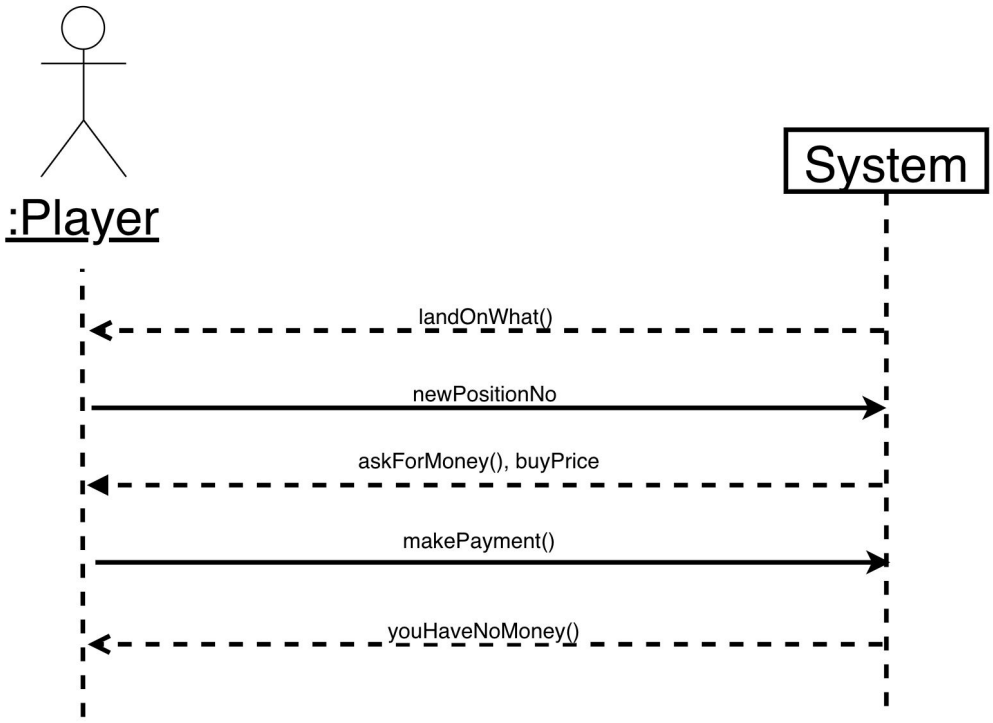
SSD4: Moving from Current Square to Destination Square



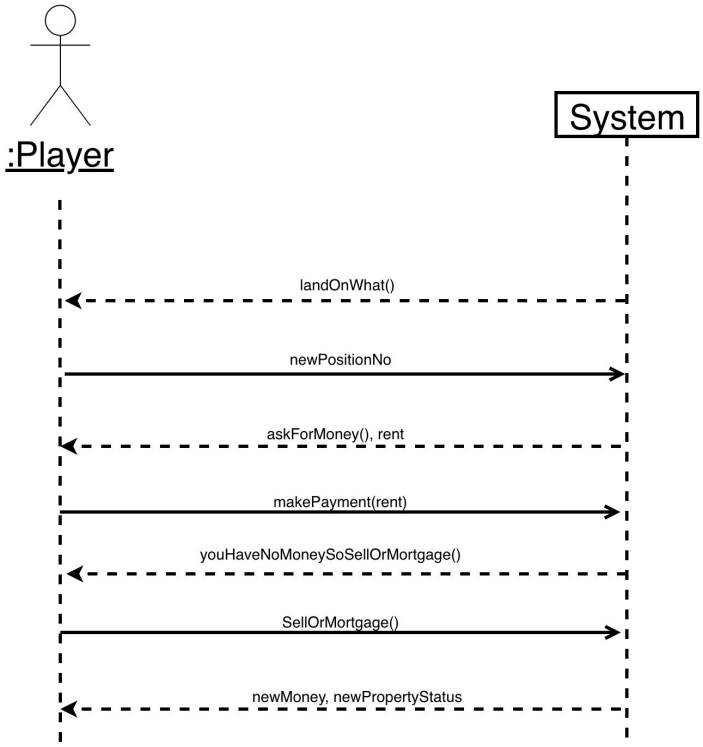
SSD5: Throwing Doubles for Three Times and Going to Jail



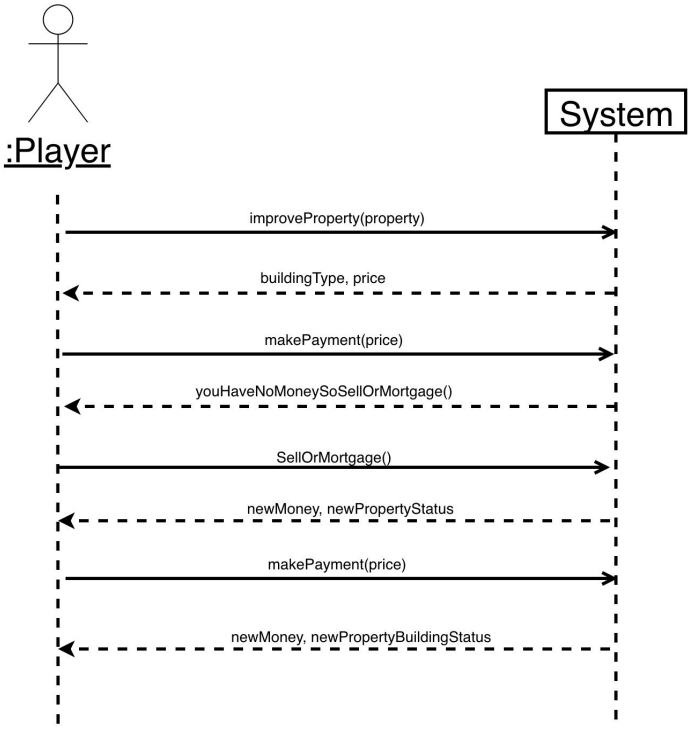
SSD6: Buying a Property



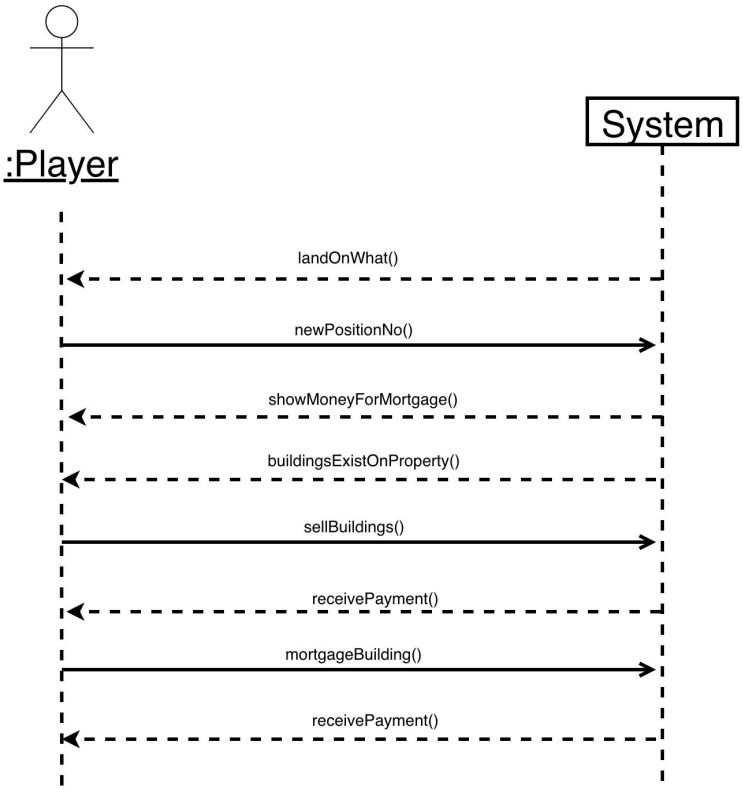
SSD7: Paying Rent on a Property



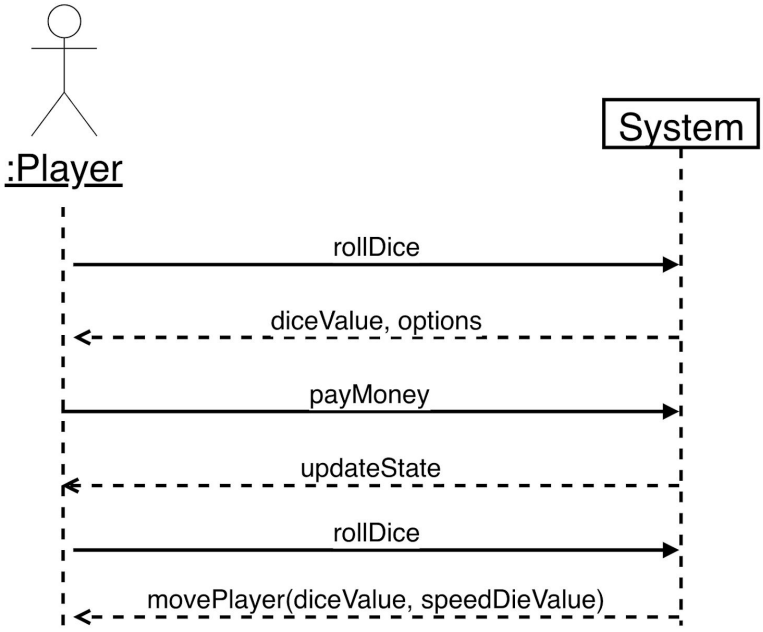
SSD8: Upgrading a Property



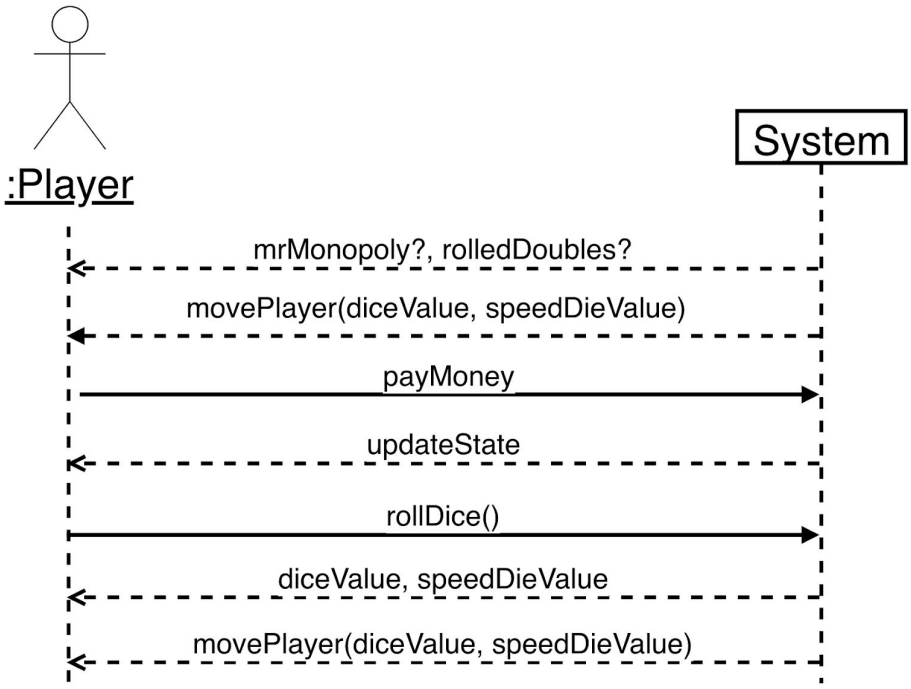
SSD9: Mortgaging a Property



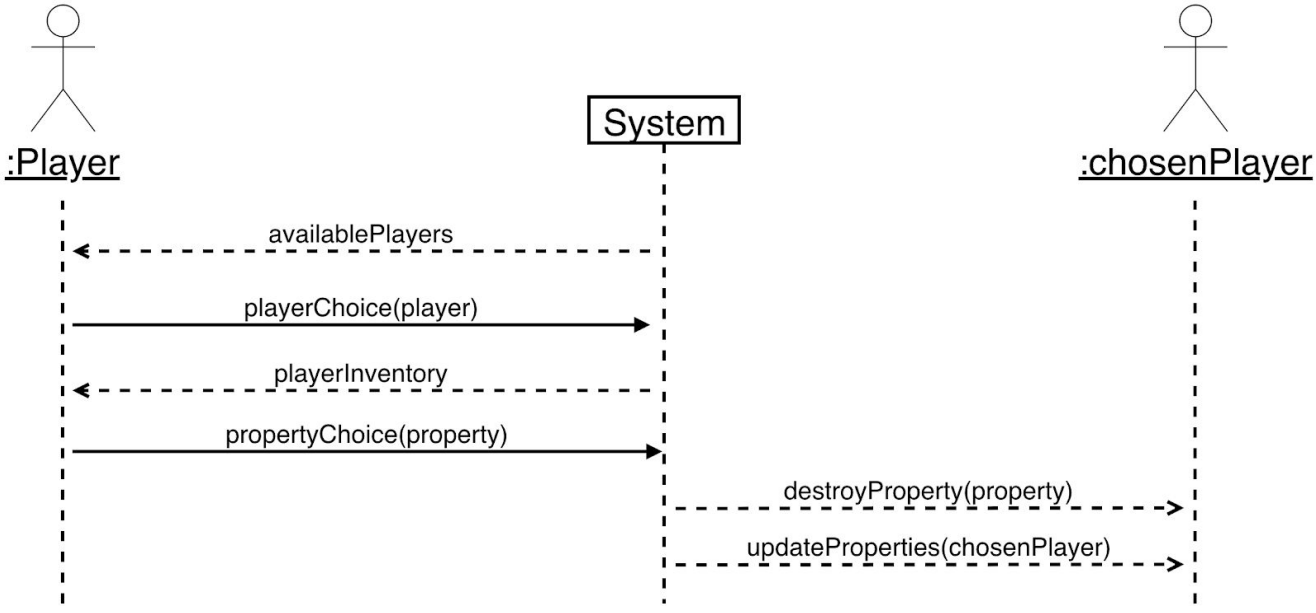
SSD10: Getting Out of Jail



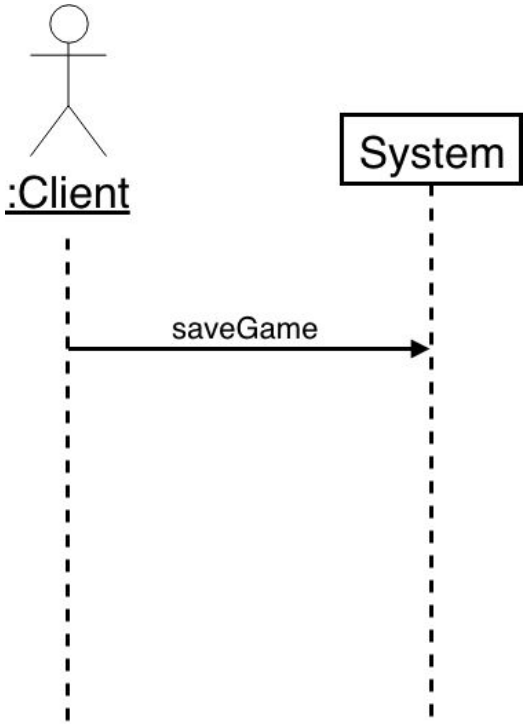
SSD11: Playing Mr Monopoly and Paying Rent



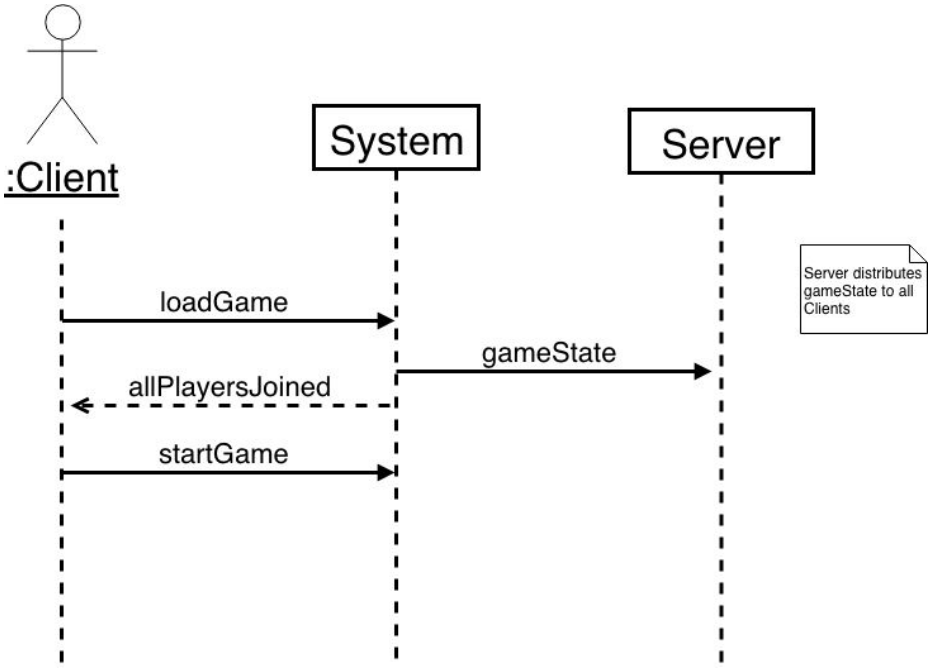
SSD12: Playing Hurricane Card



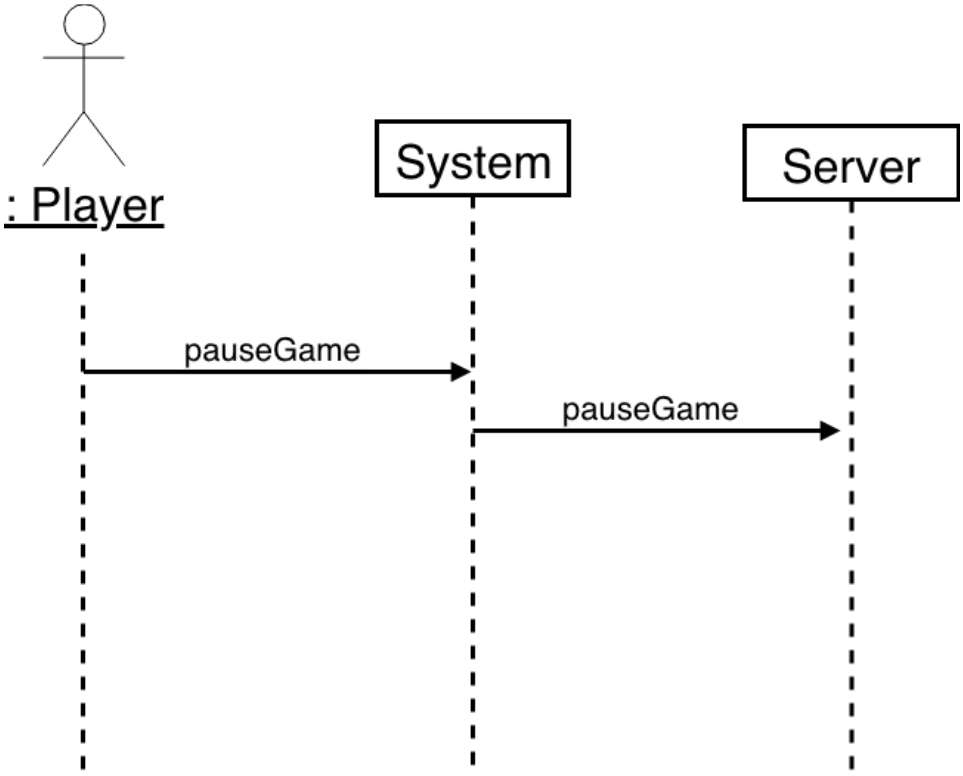
SSD13: Save Game



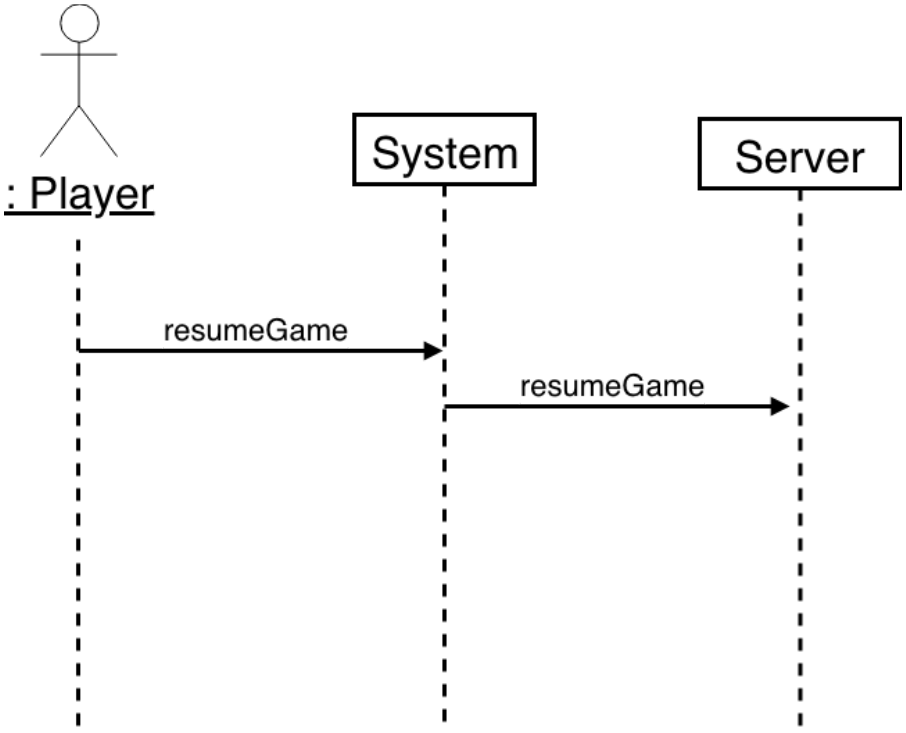
SSD14: Load Game



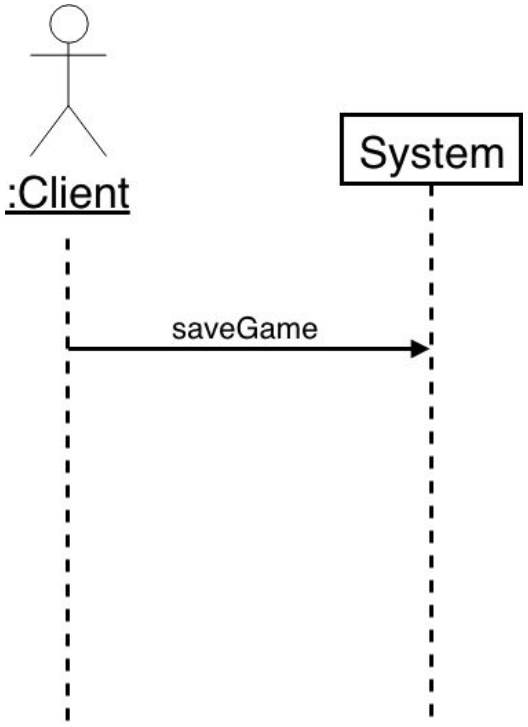
SSD15: Pause Game



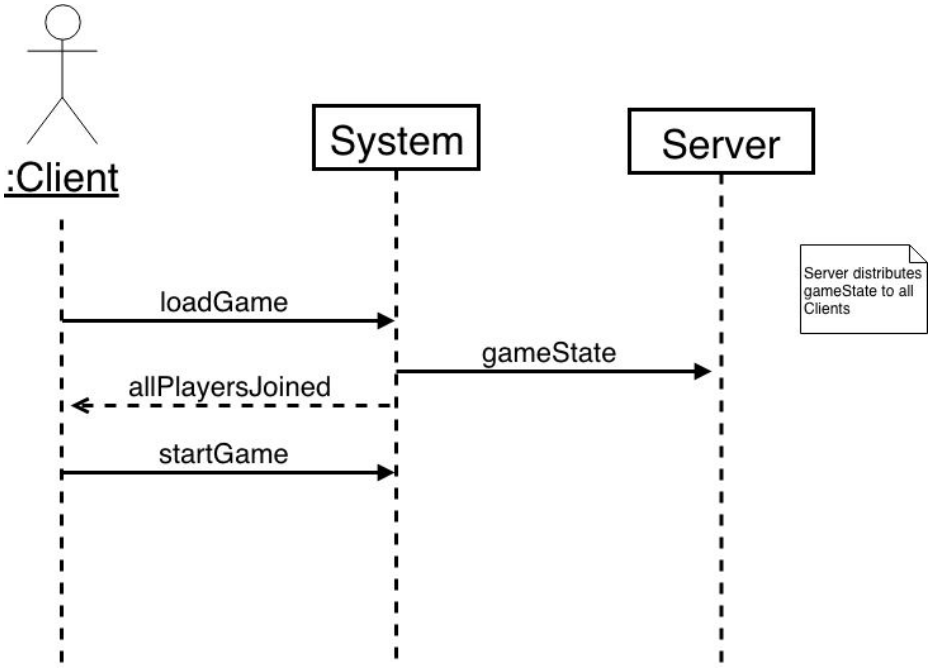
SSD16: Resume Game



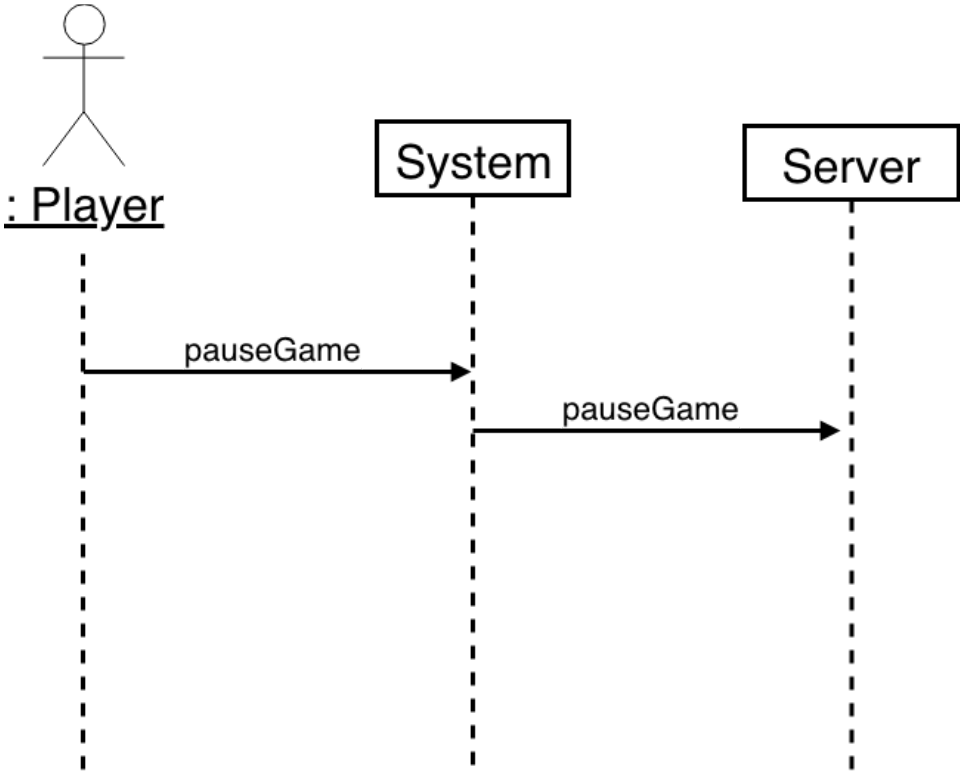
SSD17: Save Game



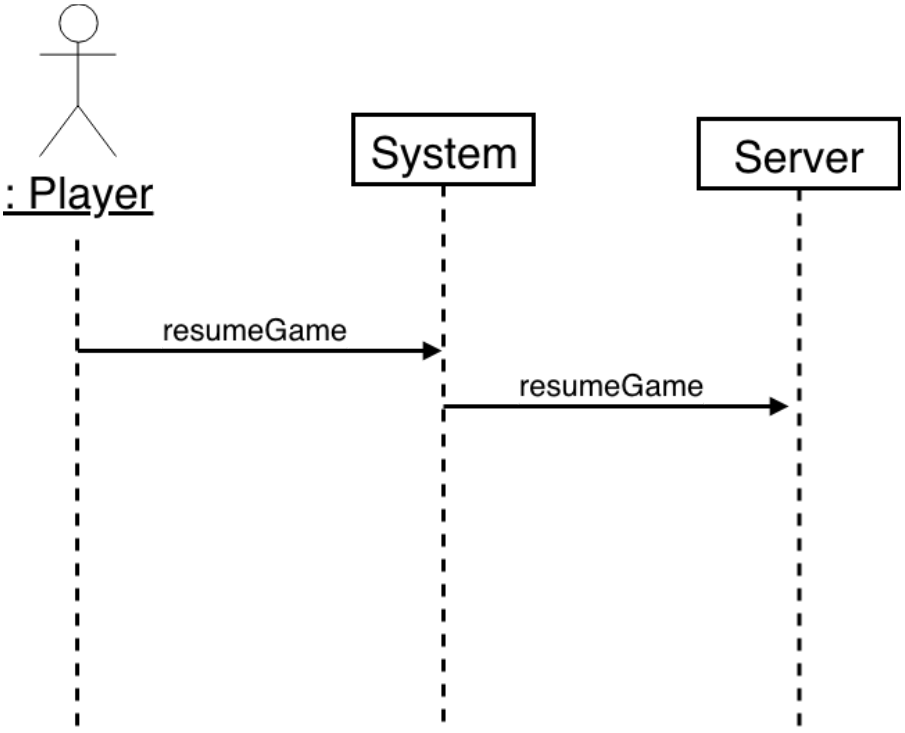
SSD18: Load Game



SSD19: Pause Game



SSD20: Resume Game



Operation Contracts

Contract CO1: improveProperty

Operation: improveProperty(Property)
Cross References: Use Cases: Upgrading a Property
Preconditions:
- Majority ownership is satisfied if player wants to build a house or hotel.
- Monopoly is satisfied if player wants to build a skyscraper
- Player has enough money to make the desired improvement

Postconditions:
- Added one more house, or hotel or skyscraper
- Player's money is updated
- Rent of property square is updated

Contract CO2: passPayDay

Operation: passPayDay(Player)
Cross References: Use Cases: Playing a Turn, Landing on an Action Square
Preconditions: The player rolled dice or the player directly moved to a pay day square by using an action card.
Postconditions:
- If the player rolled an odd number, player's money is increased by 300.
- If the player rolled an even number or if the player landed on a pay day square through an action card, player's money is increased by 400.

Contract CO3: Buy

Operation: buy(propertyName)
Cross References: Use Cases: Moving from Current Square to Destination Square**,** Buying a Property, Landing on Railroads & Transit Stations, Playing Mr. Monopoly and Buying
Preconditions: Player landed on a purchasable square that is unowned and has enough money to buy it.
Postconditions:
-Ownership status of property with the name propertyName is changed.
-Money of player is reduced by the amount of the price of that property.

Contract CO4: rollDice

Operation: rollDice()
Cross References: Use Cases: Throwing Doubles for Three Times, Landing on Railroads & Transit Stations, Landing on an Action Square, Server Plays on behalf of a Player when Time Limit is Exceeded, Moving from Current Square to Destination Square, Paying Utility Rent, Getting Out of Jail, Bot Player Playing its Turn
Preconditions: The player who rolls the dice is the current player and is not in jail.
Postconditions:
-The values of dice faces have been recorded.
-Whether the player will play again is determined by checking if doubles have been rolled.

Contract CO5: randomDecision

Operation: randomDecision()
Cross References: Use Cases: Bot Player Playing its Turn, Server Plays on behalf of a Player when Time Limit is Exceeded
Preconditions:
- The bot has rolled the dice and landed on a square, on which it will make a decision.
- A bot player has been assigned for a players turn, since no response has been received within three minutes.
Postconditions:
-The amount of money the bot possesses, the square it stands on and its current situation may change depending on the random decision it makes.
- The amount of money the other players possess, and their states may change depending on the random decision the bot player makes.

Contract CO6: sellHouse

Operation: sellHouse(propertyName)
Cross References: Use Cases: Selling a Building, Paying Utility Rent, Landing on Railroads & Transit Stations, Landing on an Action Card, Playing Mr. Monopoly and Paying Rent
Preconditions: Player owns the said property, has at least one house on it, and selling that house does not result in an uneven distribution of houses in that colour group.
Postconditions:
-Money of Player is increased.
-NumHouses of Player of the chosen property is decreased.
-Rent of said property is decreased.

Contract CO7: payRent

Operation: payRent()
Cross References: Use Cases: Paying Rent on a Property, Paying Utility Rent, Playing Mr. Monopoly and Paying Rent
Preconditions: Utility is owned by someone else, landing on a utility square
Postconditions: Rent is paid depending on the result of the 3 dice and number of utilities owned. (2 dice is considered when when the player arrives to a utility square through a CHANCE card.)

Contract CO8: switchTurn

Operation: switchTurn(Player current, List players)
Cross References: Use Cases: Throwing Doubles for Three Times, Landing on an Action square, Landing on Railroads & Transit Stations
Preconditions: System determines that the turn of current player is over.
Postconditions:

- isCurrentPlayer of current is set to zero.
- isCurrentPlayer of next element in players list is updated to 1.

Contract CO9: mortgage

Operation: mortgage(propertyName)
Cross References: Use Cases: Mortgaging Properties, Paying Utility Rent, Paying Rent on a Property, Playing Mr. Monopoly and Paying Rent
Preconditions: Player owns said property and doesn't have any houses on any property of that colour group.
Postconditions:
-State of property changes to "mortgaged"
-Player's Money increases.
-Rent of property becomes zero.

Contract CO10: drawCard

Operation: drawCard(List cards)
Cross References: Use Cases: Landing on an Action Square
Preconditions: Player has landed on a Community Chest or Chance square.
Postconditions:

- Information of the first element of cards is passed to system.
- First element of cards is pushed to the last.

Contract CO11: addNewPlayer

Operation: addNewPlayer(name : String)
Cross References: Use Cases: Initializing the game
Preconditions: Predefined max player and time limits are not reached yet.
Postconditions:

- A new Player instance called newPlayer is generated.
- A new Pawn instance called newPlayerPawn is generated.
- An association is formed between newPlayer and newPlayerPawn.
- numPlayers field is updated.

Contract CO12: playHurricaneCard

Operation: playHurricaneCard(ActionCard)
Cross References: Use Cases: Playing Hurricane Card, Landing on an Action Square
Preconditions: The player has already landed on an Action square and drew a Hurricane Card.
Postconditions: The player that is selected by the current player loses one of his/her houses from each property of a certain colour group or one of his depots/cab stands from each railroad/cab company he/she owns. In the case of skyscrapers, they are converted to 4 houses.

Contract CO13: player2bot

Operation: player2bot(Player)
Cross References: Bot Player Playing its Turn, Server Plays on behalf of a Player when Time Limit is Exceeded
Preconditions:
One of the players, say Player A, has left the game. (Because of disconnection or quitting the game.)
The current turn is Player A's turn. The current player has not interacted with the system for more than 3 minutes.
Postconditions:
Player A is replaced by a bot player who makes decisions randomly.
All the belongings of Player A (properties, money, Chance and Community Chest cards) are taken from him/her. A new bot player is initiated and all the belongings are transferred to it.

Contract CO14: movePlayer

Operation: movePlayer(regular dice: diceValue, speed die: speedDieValue)
Cross References: Moving from Current Square to Destination Square
Preconditions: 3 dice are rolled
Postconditions: Several postconditions are possible depending on the result of the dice:

- If the Speed die's face value is 1, 2 or 3 the player moves according to the roll sum of 3 of the dices.
- If Bus Icon is rolled, the player moves to according to the roll sum of the two dices. Then, the player moves to the next nearest Chance or Community Chest card.
- If Mr. Monopoly is rolled the player first moves according t the roll sum of the two dice. Then the player fulfills the requirements of the square that he/she lands on. After then:
    - If all the properties are not owned, the player moves to the next unowned property.
    - If all the properties are owned, the player moves to the next property and pays rent.
    - If all the properties are mortgaged, the player does not move.

  Other players can not trade or build during the Mr. Monopoly move.

Contract CO15: winTheGame

Operation: winTheGame(List players)
Cross References: Current Player Winning The Game
Preconditions: Only two players are left in the game. (Other players have gone bankrupt)
Postconditions: The winner player is announced by sending a message to each player.

Contract CO16: pauseGame

Operation: pauseGame()
Cross References: Use Cases: Pausing the Game, Communicating with the Server
Preconditions: Current player is not a bot.
Postconditions:

- The game was paused in every client
- Interactions were disabled. No one can buy, sell, or roll dice.
- All of the animations that were playing stopped.

Contract CO17: resumeGame

Operation: resumeGame()
Cross References: Use Cases: Resuming the Game, Communicating with the Server
Preconditions: The game has been paused.
Postconditions:

- The game continues from the last moment that it was paused.
- If there were an animation that was stopped, the animation continues.
- Interactions were enabled.

Contract CO18: saveGame

Operation: saveGame()
Cross References: Use cases: Save the Game, Communicating with Server
Preconditions: The game is paused.
Postconditions:

- A save file is created.
- Current game state is written to save file.

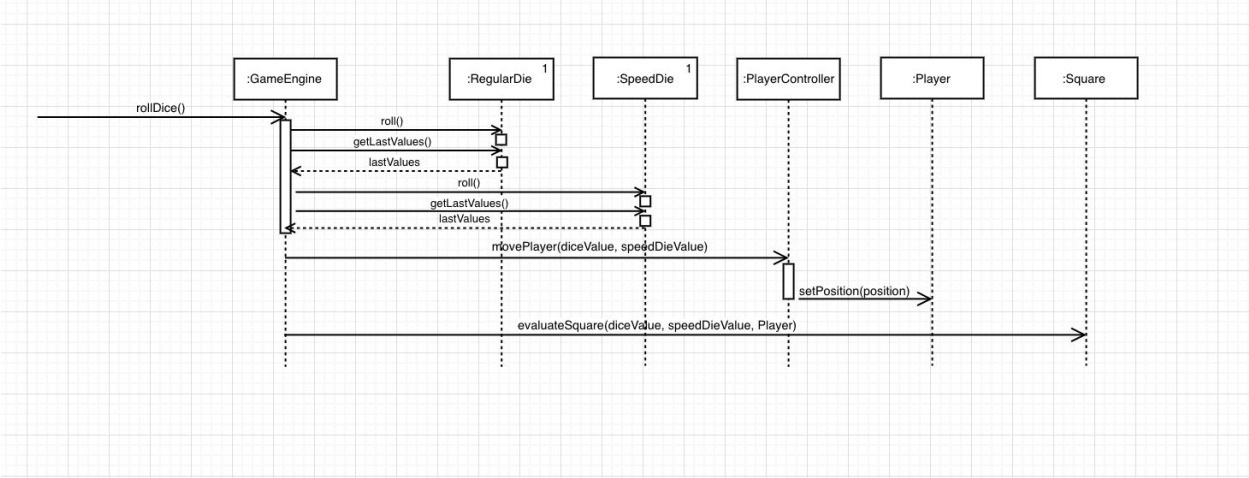
Contract CO19: loadGame

Operation: loadGame(fileName: String)
Cross References: Use Cases : Loading the Game, Initializing the Game, Joining the Game, Communicating with the Server
Preconditions: There is at least one save file.
Postconditions:

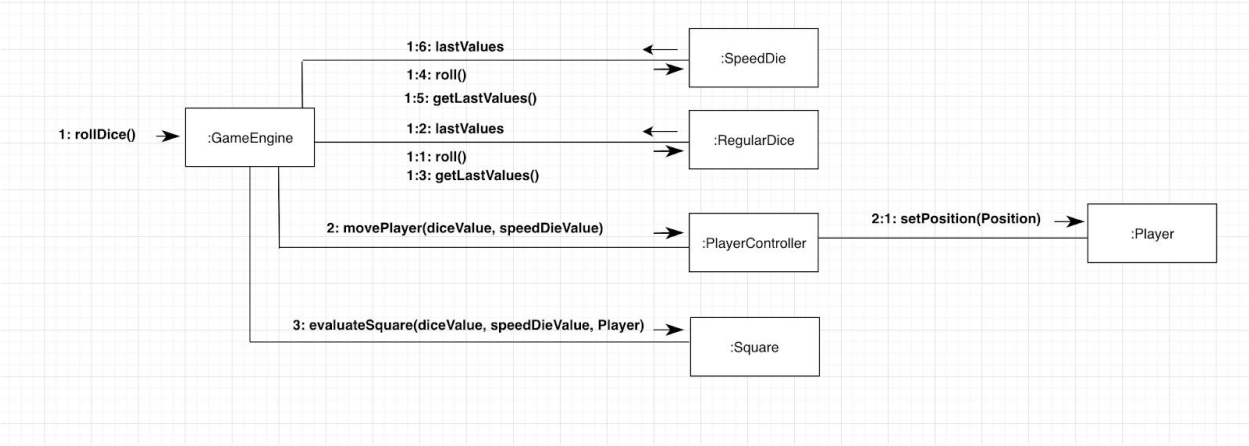
- The game state in the save file was loaded in all clients who joined the server.

Interaction Diagrams

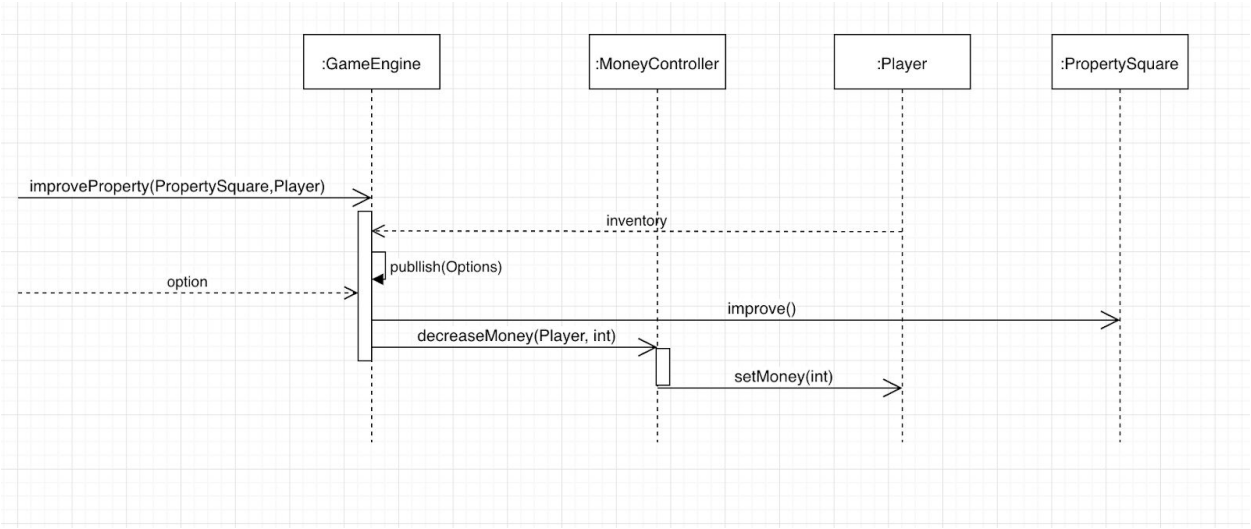
Sequence Diagram 1: Move from Current Square to Destination Square



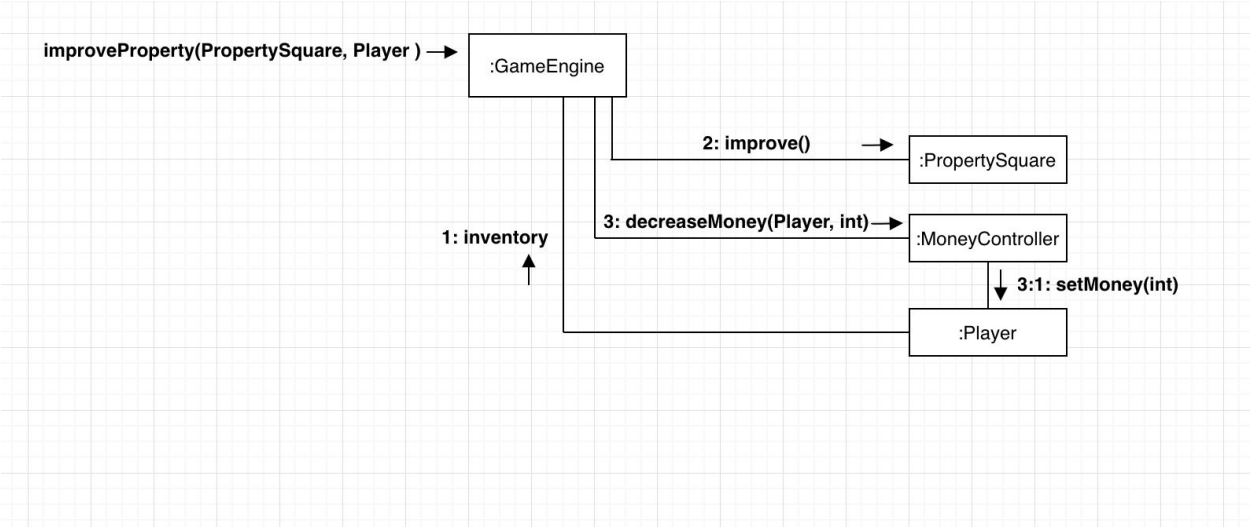
Communication Diagram 1: Move from Current Square to Destination Square



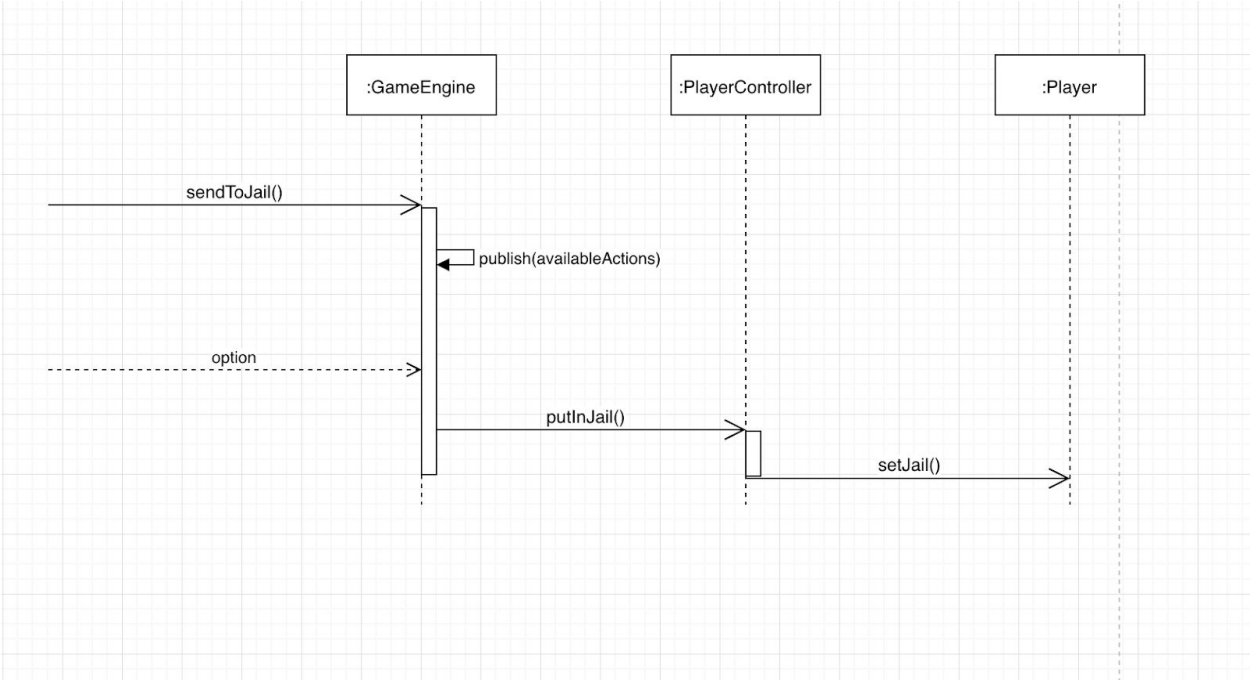
Sequence Diagram 2: Improve Property



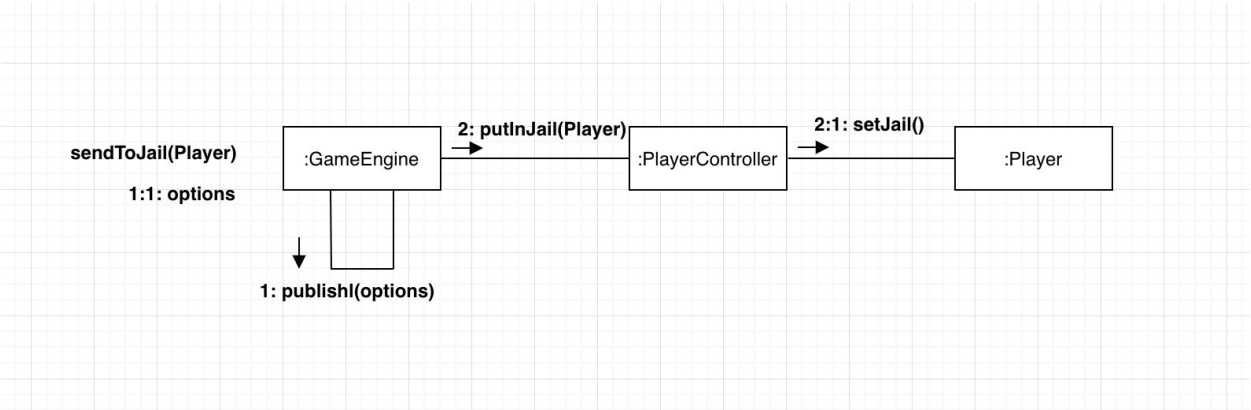
Communication Diagram 2: Improve Property



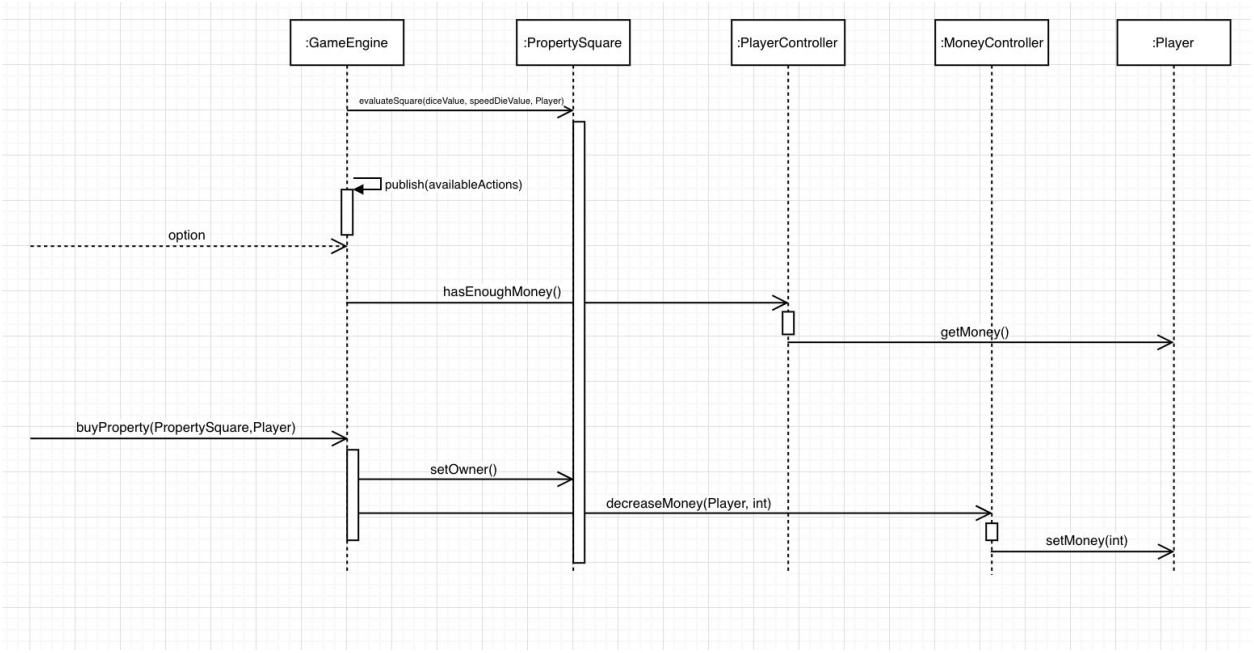
Sequence Diagram 3: Go to Jail



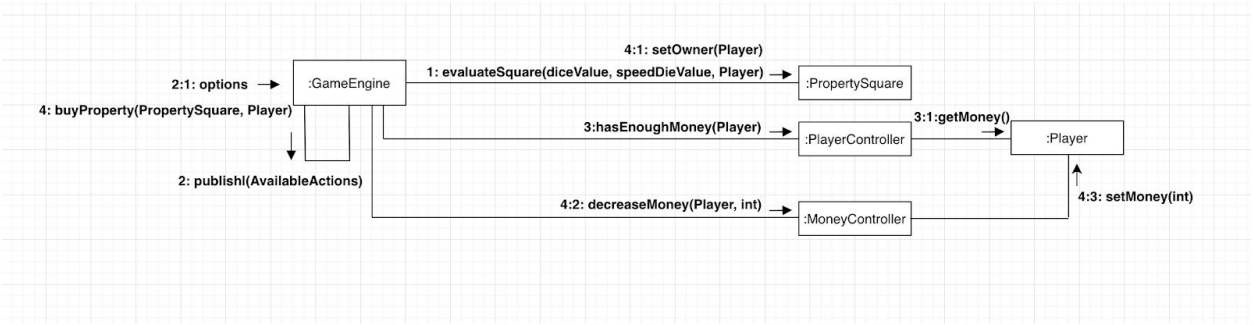
Communication Diagram 3: Go to Jail



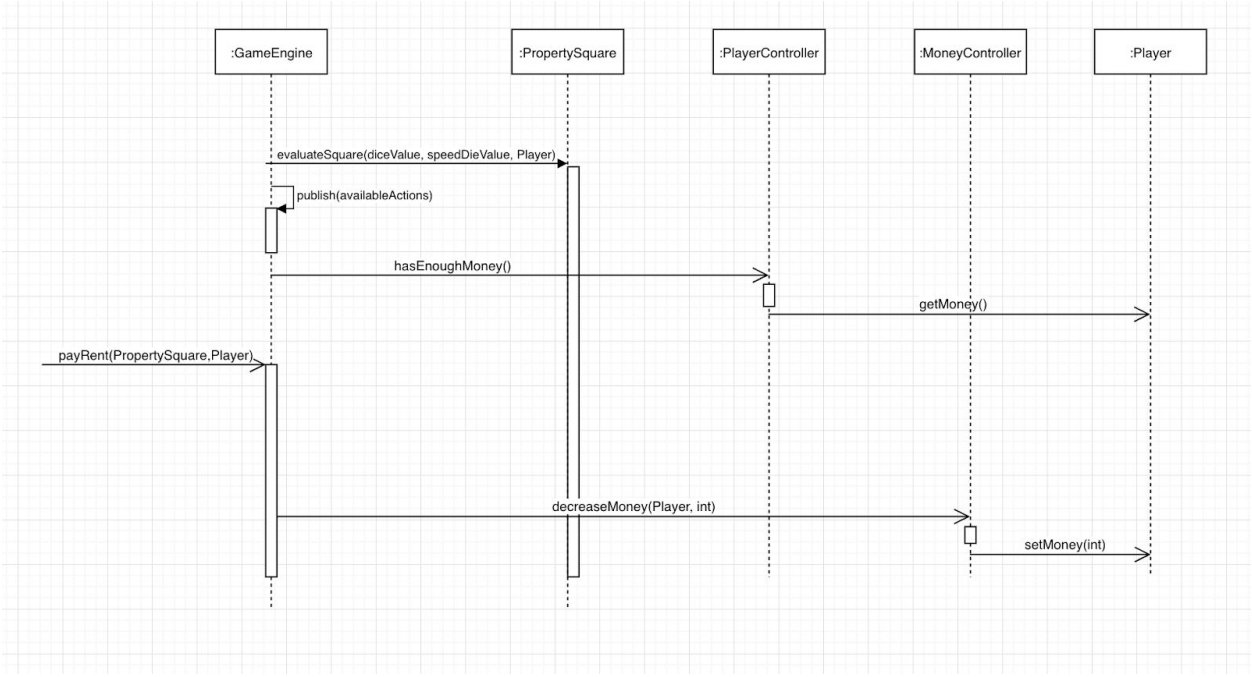
Sequence Diagram 4: Buy Property



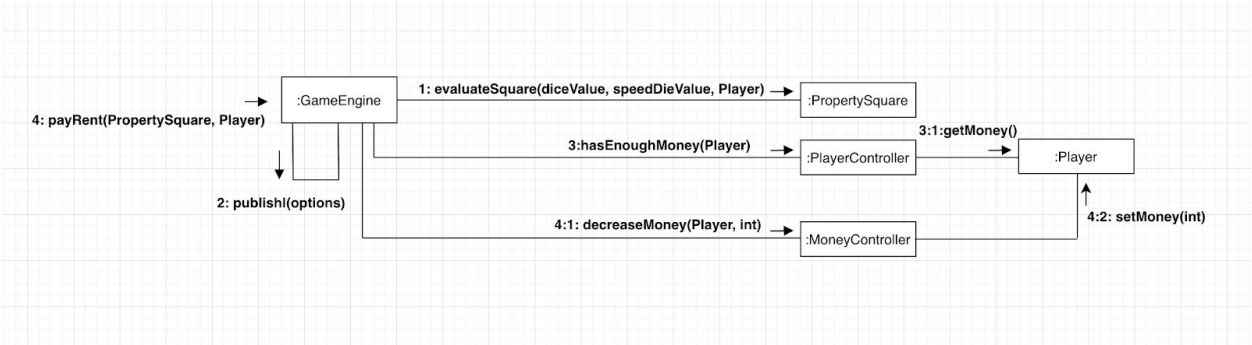
Communication Diagram 4: Buy Property



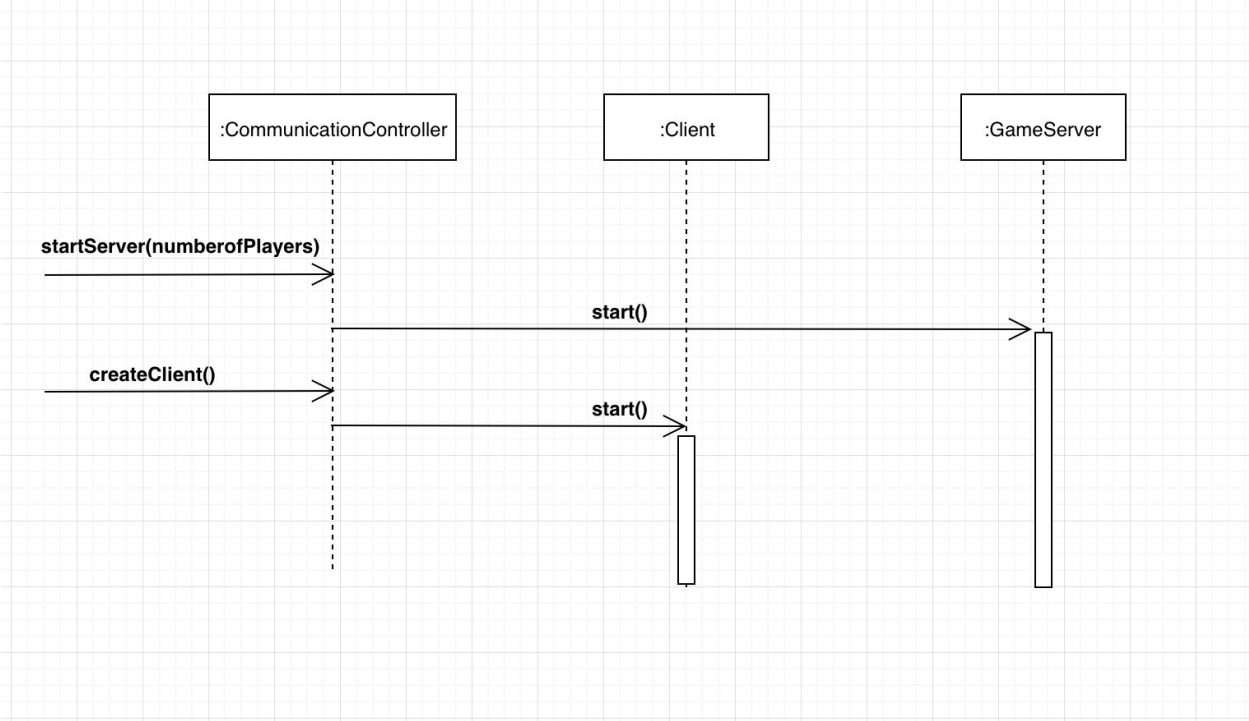
Sequence Diagram 5: Pay Rent



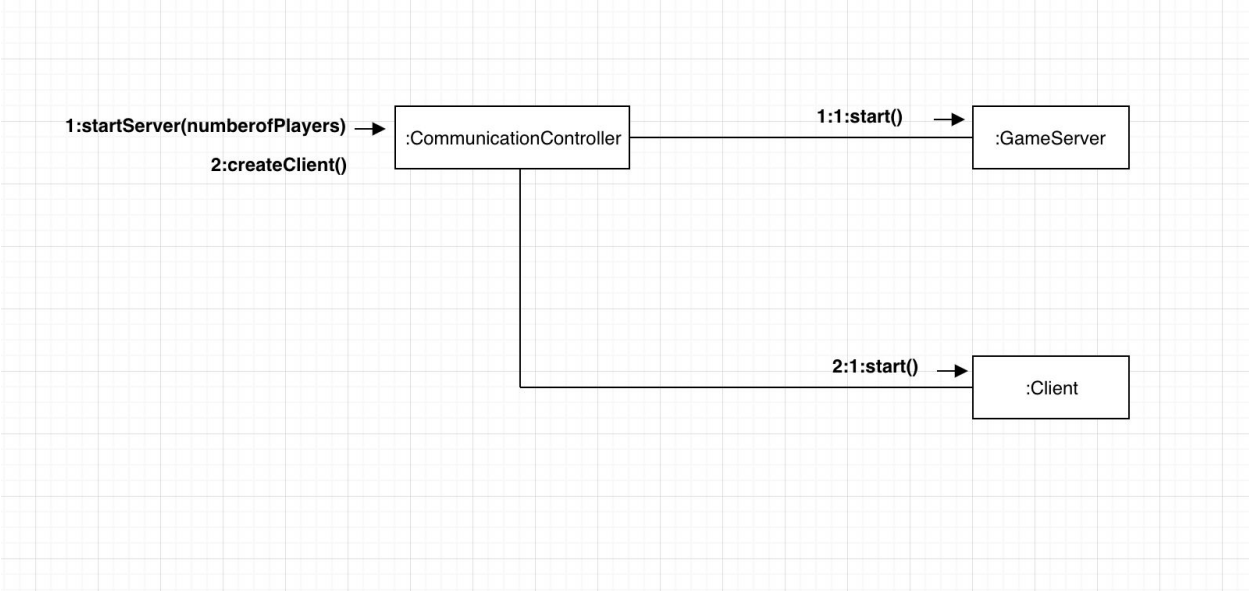
Communication Diagram 5: Pay Rent



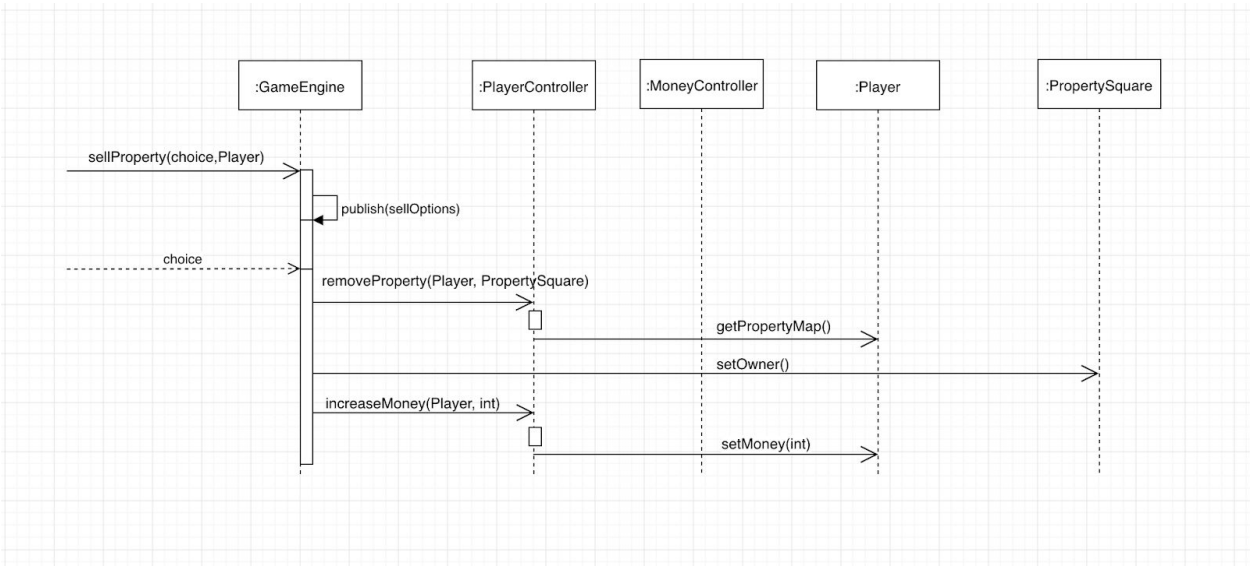
Sequence Diagram 6: Start Server



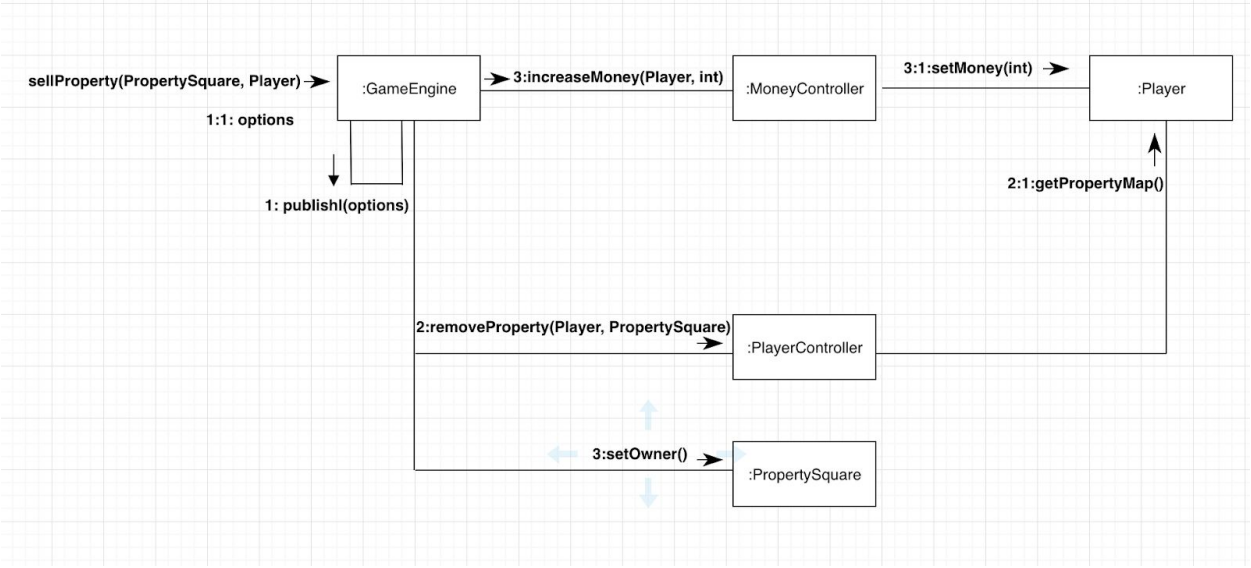
Communication Diagram 6: Start Server



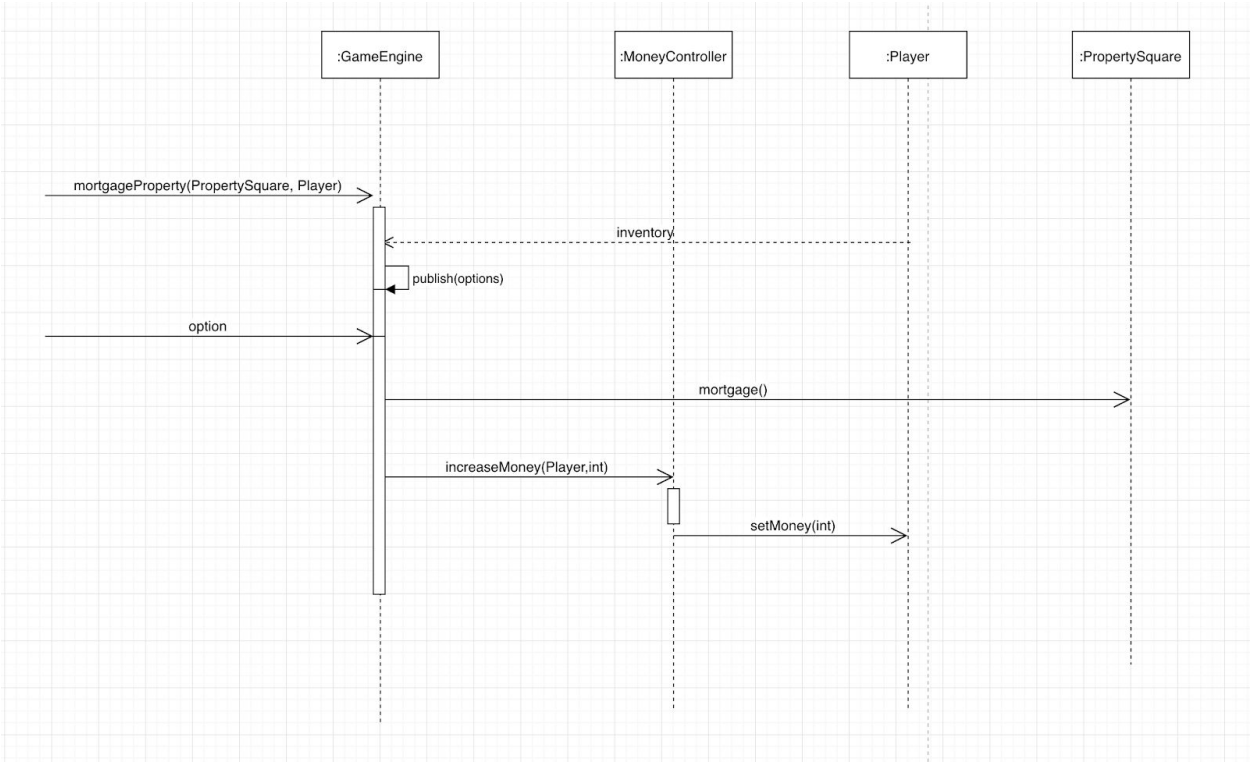
Sequence Diagram 7: Sell Property



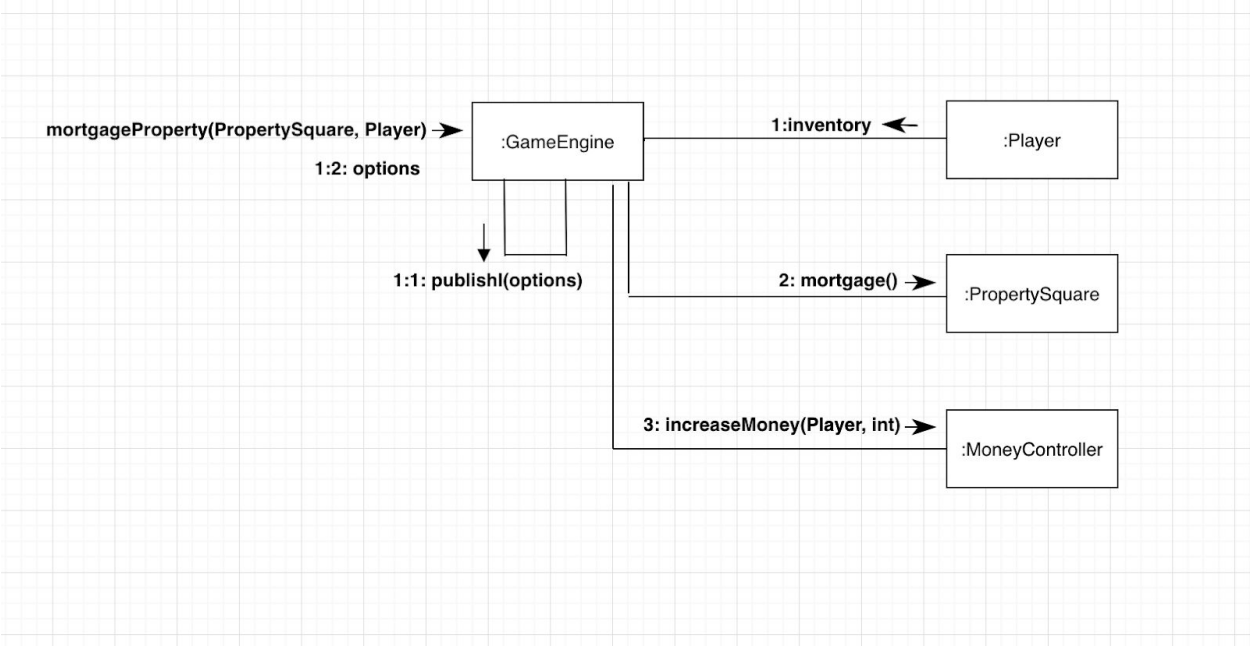
Communication Diagram 7: Sell Property



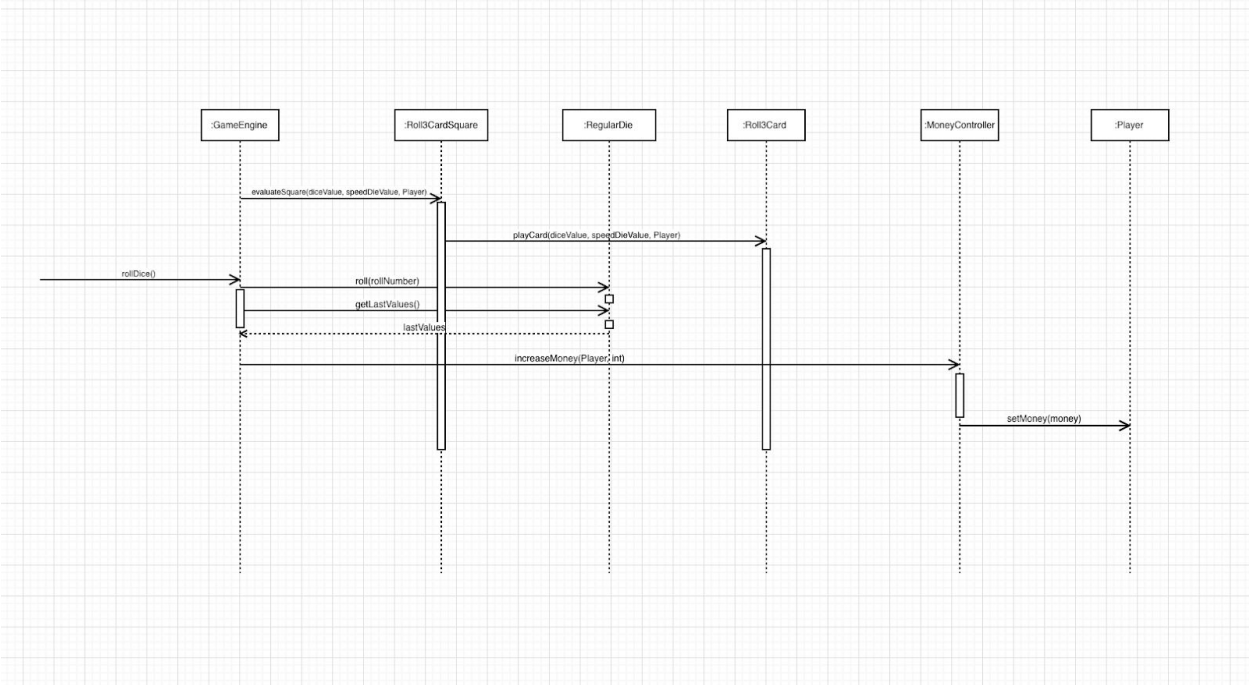
Sequence Diagram 8: Mortgage Property



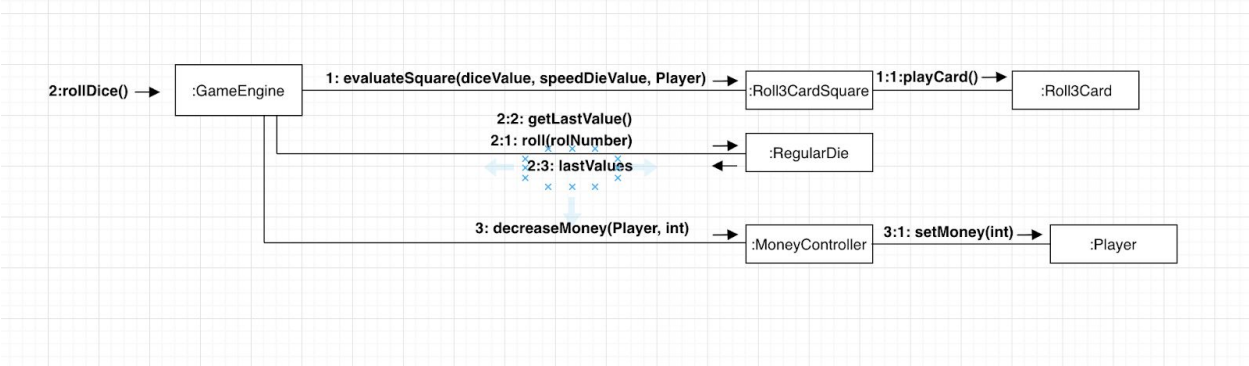
Communication Diagram 8: Mortgage Property



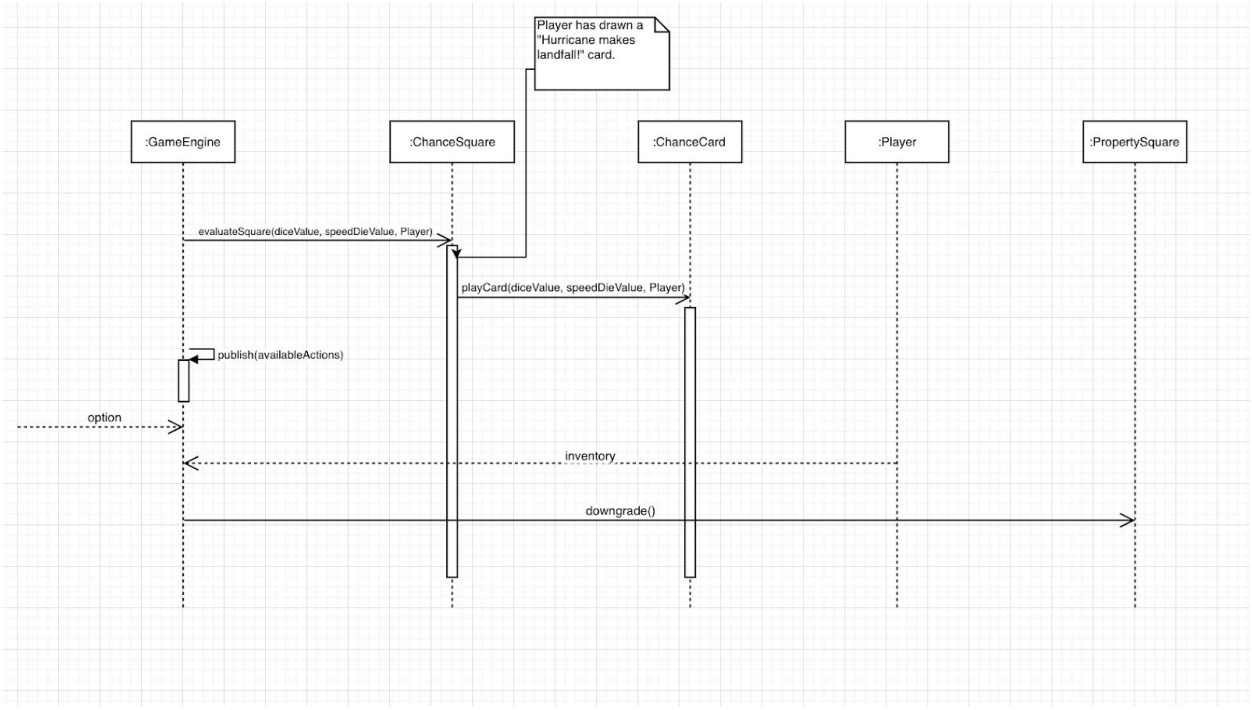
Sequence Diagram 9: Landing on Roll 3 Square



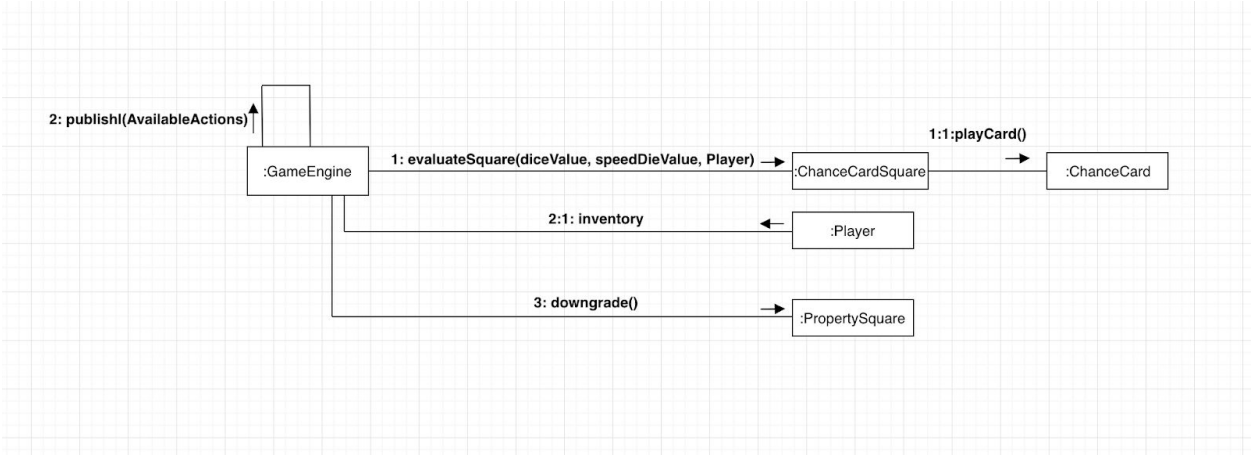
Communication Diagram 9: Landing on Roll 3 Square



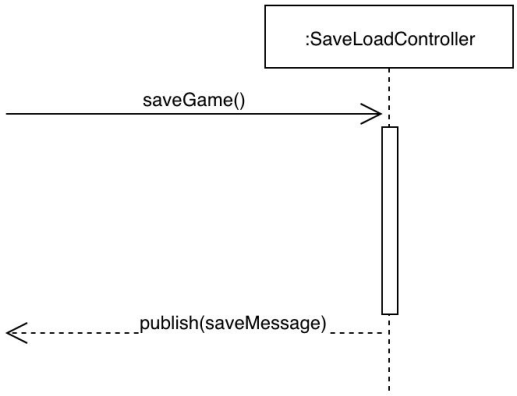
Sequence Diagram 10: Play Hurricane Card



Communication Diagram 10: Play Hurricane Card

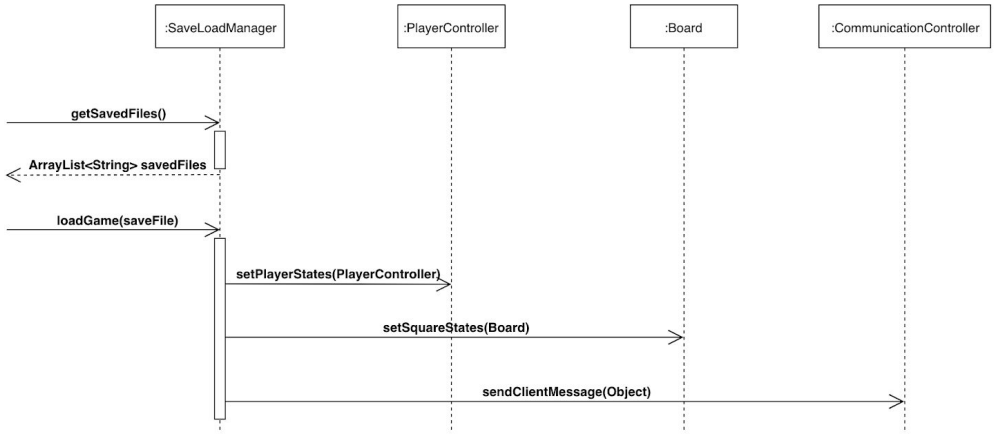


Sequence Diagram 11: Save Game

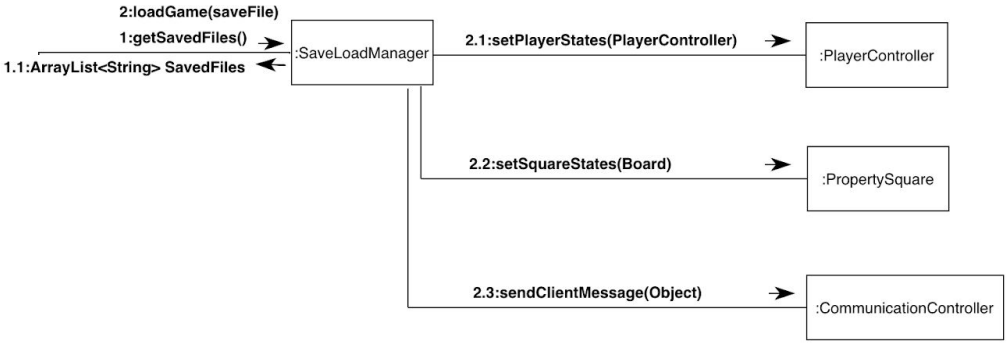


Communication Diagram 11: Save Game

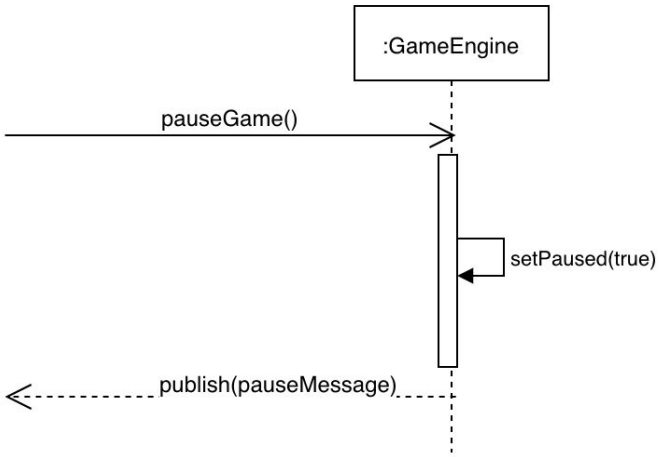
Sequence Diagram 12: Load Game



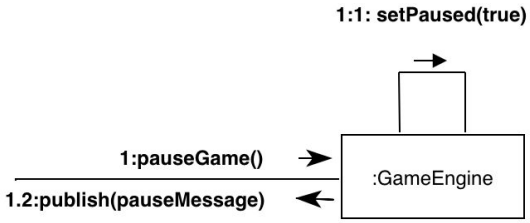
Communication Diagram 12: Load Game



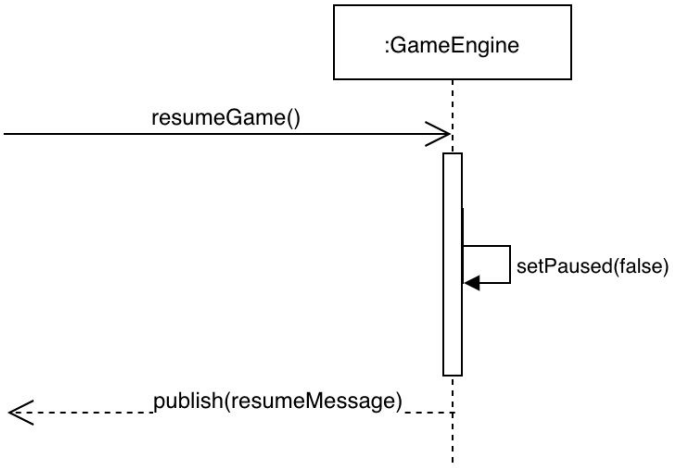
Sequence Diagram 13: Pause Game



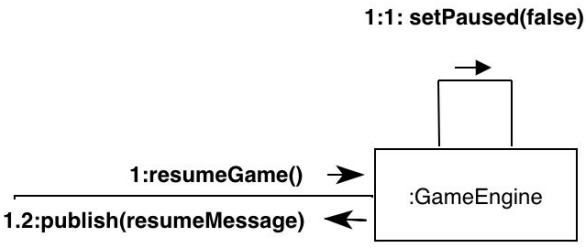
Communication Diagram 13: Pause Game



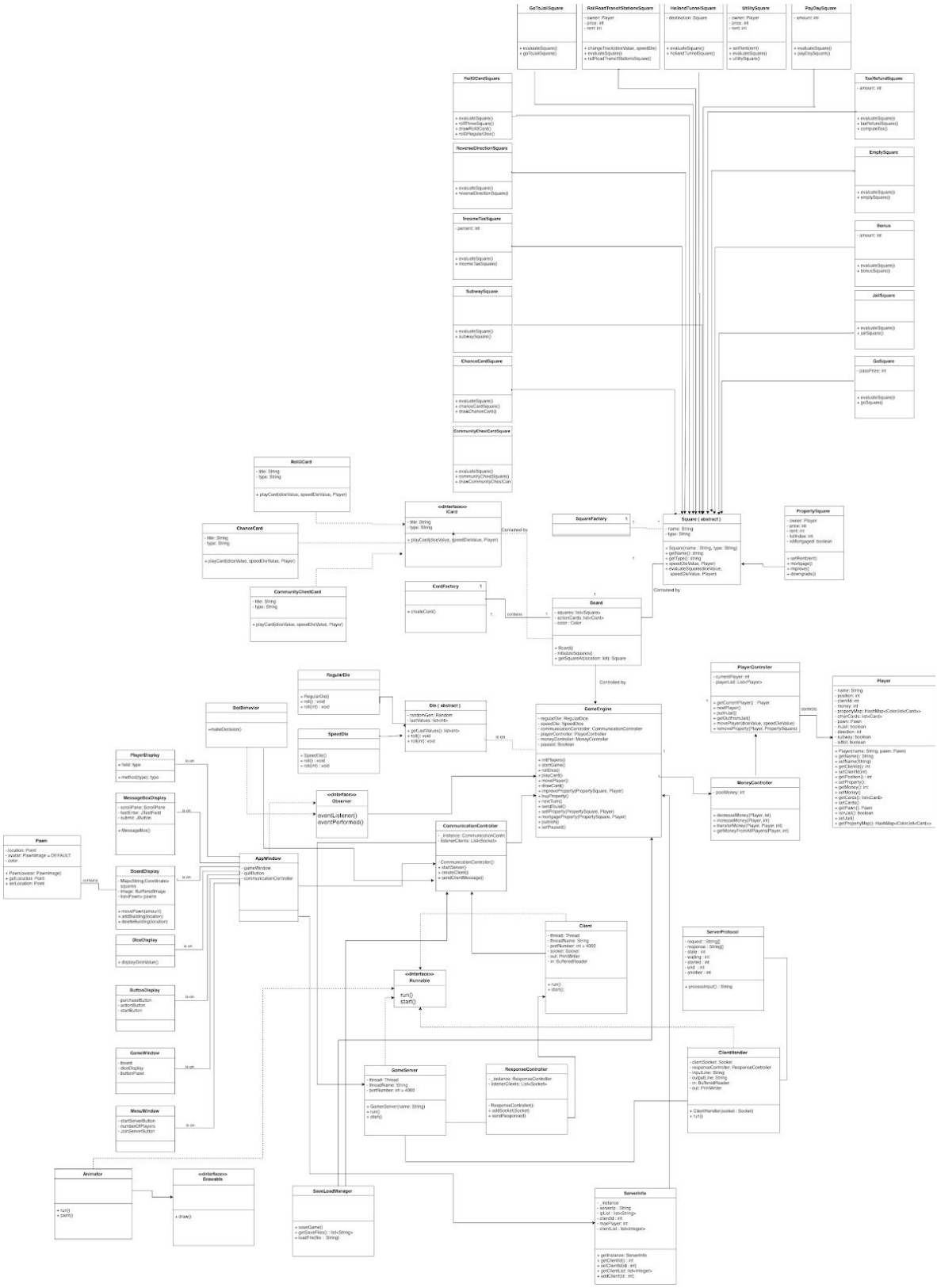
Sequence Diagram 14: Resume Game



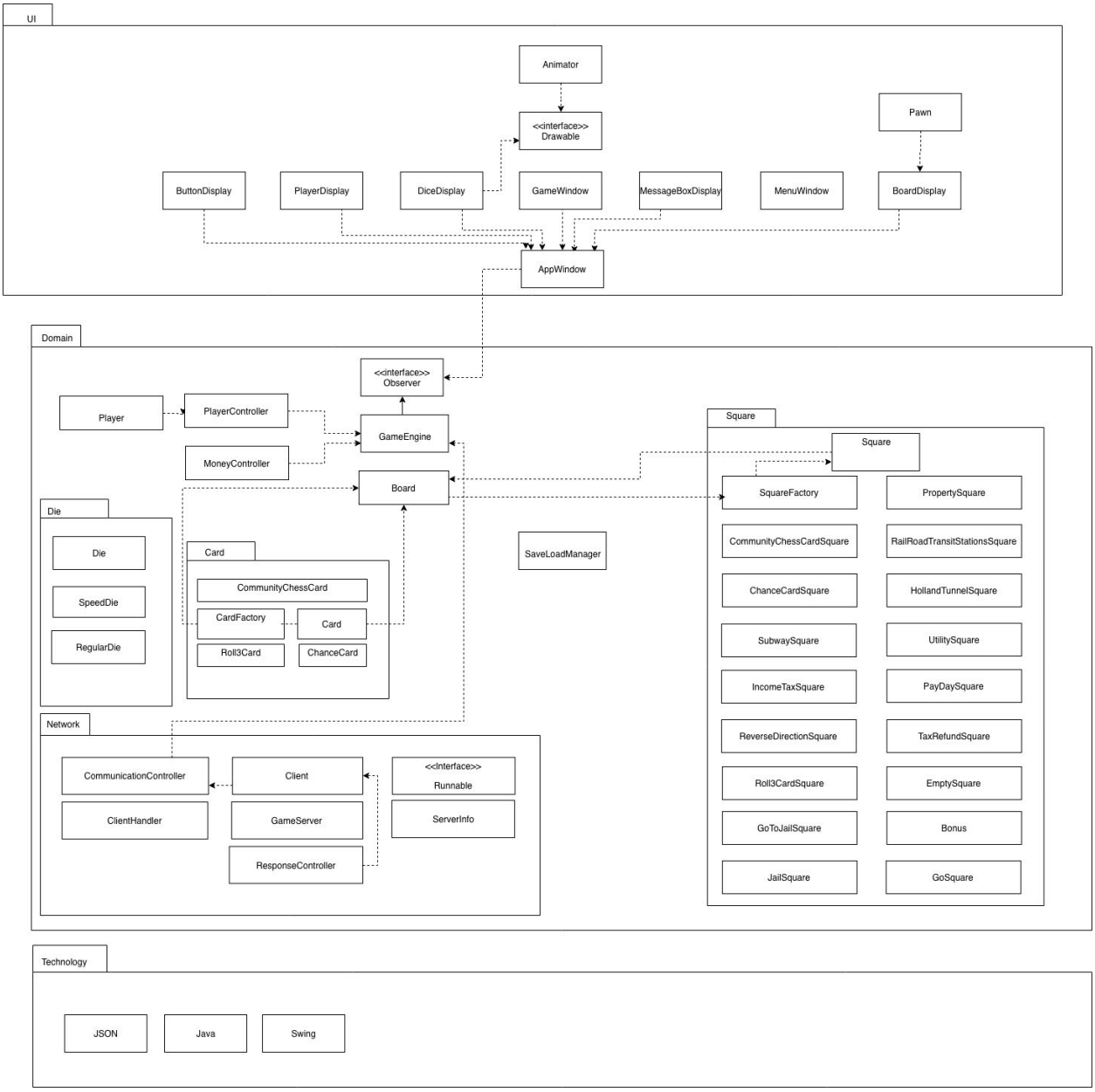
Communication Diagram 14: Resume Game



UML Class Diagram



Logical Architecture and Subsystems



Final Test Plan

Domain Classes to be Tested:

- GameEngine,
- PlayerController,
- MoneyController,
- PropertySquare,
- Square.

Planned Tests for the GameEngine Class:

- Planning to test this class, since the GameEngine acts as the main logic of the game. Most of the interactions are directed by it.
- Checking whether the dice rolling results in the correct intervals.
- Checking whether the amount the player will move after the resulting dice rolling is calculated correctly.
- Checking if drawing cards result in the correct card types.
- Checking the buying functionality in the case that a player lands on an unowned property.
- Checking whether or not paying rent is decreasing the money of the payer and increasing the money of the owner, otherwise the paying rent is signaling a bankrupt of a player if the payer does not have enough money to pay.

Planned Tests for the PlayerController Class:

- Planning to test this class, since the states of the players are mainly controlled by the PlayerController.
- Checking if the PlayerController switches to the next player correctly.
- Checking if a player that the PlayerController sends to the jail is indeed in jail.
- Checking if a player that the PlayerController releases from the jail is indeed out of the jail.
- Checking if the target position of a moved player is correct.
- Checking several instances of the class with the repOk() method.

Planned Tests for the MoneyController Class:

- Planning to test this class, since the game's winning logic depends on all players except one going bankrupt. In order for this to happen, the money transactions should be completed without errors.
- Checking if basic increasing and decreasing money logics work.
- Checking whether money transfers result in the correct way for both parties.
- Checking if collecting money from all players has the correct logic. (new players are created and they are added to the player controller)

- Checking the hasEnoughMoney() which is the method that checks whether the player has more money than a certain amount.

Planned Tests for the PropertySquare Class:

- Planning to test this class, since during most of the game, players in various states can land on a PropertySquares that can also have various states. Also, the majority of the squares on the board consist of PropertySquares.
- Tested various conditions of landing on a property square and evaluating it, where the square is either owned or unowned; and the players can either afford to pay the rent or cannot afford to pay the rent.
- Checking if an unowned square is indeed seen as unowned.
- Checking if an owned square is indeed seen as owned.
- Checking whether the rent of an improved property is greater than before.
- Checking whether the rent of a downgraded property is smaller than before.
- Checking several instances of the class with the repOk() method.

Planned Tests for the Square Class:

- Planning to polymorphically test various Square classes.
- Checked Square classes such as:
    - Go to Jail,
    - Go,
    - Bonus,
    - Reverse Direction,
    - Luxury Tax.
    - Roll 3 Square
- Mainly focused on the squares' evalueateSquare methods with various conditions.

Design Discussion

Model-View Separation

Model-view separation states that UI objects are not aware of the application logic, are not responsible from application logic, and also domain objects do not have any coupling, or dependency to UI objects. We applied model-view separation principle in our design. For example, in our project, domain objects do not have any coupling to UI objects and UI objects are not aware of game logic.

General Responsibility Assignment Software Patterns (GRASP)

Creator Principle

Creator Pattern defines which class should be assigned the responsibility to create some other class. Upon considering the advantages, the creator class is chosen to be the creator because this class has the knowledge about the created class. However to speak about disadvantage of creator pattern, when we allow the creator class to create another class just because it has some knowledge or relation to the other class, we may create extra coupling or low cohesion. For example, in our project, we have assigned the responsibility of creating the Board to the GameEngine, since it has the knowledge about the Board. However, the creation of the squares on the Board is done according to the Factory Pattern.

Controller Principle

Controller Pattern defines a controller as an object that receives and coordinates system operations. In our project, we used a communication controller for the network part of the monopoly game. Also, since there are a lot of operations related with the Player class, we also implemented a player controller class. One advantage of the controller is that the controller can be used again. In addition, it allows to control the sequence of the activities. However, there is a risk of making a bloated controller. If a class is overloaded with too many responsibilities the controller becomes bloated. Also, a controller has to delegate other classes to perform tasks.

Low Coupling Principle

Low Coupling Pattern states the importance of low coupling such that impact of change will be very low. Throughout our project, we aimed to connect the components in our system such that their dependency with each other is minimized. The main advantage of it is that the components in our system can easily be replaced since the dependency between them is restricted. However, this approach usually lead to data replication and it may cause problems in data synchronization.

Gang of Four (GoF)

Singleton Pattern

Singleton is a design pattern that restricts the instantiation of a class to one object. In our project Board, Die, Controllers, Factories, Animator and Navigator are implemented as singleton since throughout the game, we only instantiate them once. By using a singleton pattern, we do not need to instantiate a new object each time we need an object. The main disadvantage is that the singleton pattern makes the testing of the code more difficult. The most important advantage of Singleton is that it creates a unique class which is available to us throughout the application. However, its disadvantage is that it makes the testing more difficult, we would need to have a full-functional class that is used to test Singleton.

Simple Factory Pattern

The simple factory pattern allows the creation of objects without having to specify the exact class of the object that will be created. This is achieved by using factory methods. In our project we used the simple factory pattern for Square and Card classes since they have a lot of different subclasses and these two classes can not anticipate the type of object that they need to create in advance. Also, the different types of subclasses have an overall complex structure and we wanted to localize the logic. However, as a disadvantage, simple factory pattern makes the reading of the code more difficult to read since it is another abstraction layer.

Facade Pattern

GameEngine wraps PlayerController, SquareController, CommunicationController and MoneyController to simplify the originally complex methods, such as player operations(e.g. moving a player), money operations(e.g. selling a property of a player) and communicating with the server(publishing the player status). Facade pattern increases the loose coupling. The disadvantage of it is that the subsystem methods are connected to the Facade layer. So, when the subsystem structure is changed, there has to be additional changes to the Facade layer.

Observer Pattern

Observer pattern is used when a function of an object notifies other objects to call their functions. We used observer pattern in our project to communicate between UI and domain classes. The interaction of the UI with the domain is ensured by using the observer/publish model. For example, after the dice are rolled in GameEngine, domain, we need to display the dice value in the UI, while doing this, domain publishes the dice value and dice display is notified by domain and dice display displays the dice value. Observer pattern helps us to ensure model-view separation (objects in the domain model should not know of objects in the UI). It allows us to send data to many objects efficiently. The main disadvantage is that publisher do not know about each observer, so publishing a basic event cause a checking in all of the observers.

We also used Observer pattern in our generic animator class. The Animator class published draw method classes that implements Drawable interface. Therefore it makes it easier to add new animations in our project.

Supplementary Specifications

FURPS+

- **Functionality**
  When error occurs in the system, the errors are written to a text file and the game is ended.
  The founder player can choose the total number of players.
  The user needs authentication to connect to the server.

- **Usability**
  There will be visible pictures for the squares. Also, all the GUI interface, including the buttons and the texts, will be visible from 1 meter. Colors of the squares will be chosen according to the Ultimate Monopoly game so avoiding colors that are associated with common forms of color blindness is limited with the design of the Ultimate Monopoly.

- **Reliability**
  When a minor error occurs in the system, the system continues to work even if a minor incorrect result is observed. For example when a player lands on a square, if the square is evaluated incorrectly due to an exception, the game would still continue.

- **Performance**
  The performance of the system is crucial for our system since there is a frequent interaction between the system and the players. The player should not wait for the system for more than 3 seconds.

- **Supportability**
  The system should be adaptable to changing requirements. For example, if the bot player who plays randomly is changed into a more intelligible AI, the new kind of bot player should be easily adaptable to the current game. Similarly, the system should be flexible when new rules are added or when different kinds of properties, action cards, community chest cards … etc are introduced to the game.
  About the configurability of the system, the system should be able to handle players with different network configurations. Also, the system should still be able to work even after the network configurations are modified.
  Cross platform**

Nonfunctional Requirements for Animation

- **Usability**
  The animation should have a proper refreshing rate that (~30 fps) so that most people can observe the animation without distortion; but keeping the animation at an optimal speed is also important in order not to keep the players waiting too long or cause them to miss the movement. Also, the colors of the frame should not interfere with the movement and they should be commonly perceivable colors.

- **Reliability**
  If there is a failure during an animation, game state will not be distorted in order to prevent any conflicts. Any animated content that is not synchronized with the game state will be restored. Both the animation and the network system use threads. Since these threads do not try to access and change a shared data at the same time , race conditions do not occur. Therefore, thread synchronization is not necessary. The lack of thread synchronization also adds predictability and reliability for our system, because if we had used thread synchronization (for example by using mutexes, semaphores or monitors), there would have been a significant risk of overlooking a synchronization problem. This would have caused system failure. Even though, the frequency of occurrence of thread synchronization problems are generally quite low, detecting them are often very difficult.

- **Performance**
  All of the speed, time, and refreshing discussions above also apply critically to the performance of the animation. The response time of the animation should be low such that the players wouldn't realize the time that it takes between the clicking of the roll button and the start of the movement of the pawn. The number of actions that we have to do in a time unit is relatively low since only a single piece moves at a time. Therefore, the throughput requirement can easily be satisfied. As discussed before, the threads do not interfere with the threads of the network system. Also the threads in the animation system never involve in a race condition. This increases the accuracy of our system. Resource usage will be minimal and the animation system will be almost always available when it is necessary.

- **Supportability**
  The animation will be flexible in terms of which path to follow, which object will be following that path, and which rules to follow throughout that path. There may be different path requirements that will be required for different objects (such as cards, pawns, etc.) to follow, and each of them should be able to follow their required paths without distortion. Therefore, the animation will be performed in a generic fashion that is able to adapt to multiple paths and can be configured in order to handle new types of animations and overcome further obstacles.

Glossary

	Glossary	
Term	**Definition and Information**	**Format**
Player	A person who is interacts with the game and trying to win.	Object
Bot Player	A player that is controlled by the system. Makes random decisions.	
Pawn	The representation of the of the player on the game board.	Image
Turn	The maximum number of times that each player has played.	Integer
Dice	A random number generator that gives a number between 1-6 (inclusive)	
Rent	The money that a player pays when landing on a property that is owned by another player	
Bankrupt	A player who has 0 or less money.	
Title Deed	A card that gives information of a property such as rent cost, upgrade cost ... etc	
Rolling Double	Occurs when both of the regular dice have the same face value for two consecutive turns	
Rolling Triple	Occurs when both of the regular dice have the same face value for three consecutive turns	
Subway	A special square.	
Utility	Purchasable squares. Can be owned and sold.	
Holland Tunnel	A special square that enables the player to move to the other Holland Tunnel.	
Community Chest Card	A special square which requires player to draw a Community Chest card.	
Chance Card	A special square which requires player to draw a Chance card.	
Mr. Monopoly	A side of the speed die which, when rolled, leads to a special move.	
Jail	A square on the board. Players can go to jail, visit it or pass from it.	
Tax Refund	Pay $90 to the bank or pay 10% of your worth to the bank.	
MessageBox	A private chatroom in which players can send each other text messages.	
Monopoly	Having three of the same property. Enables to collect triple rent.Additionally, it enables the owner to build skyscrapers	
Majority Ownership	Having two of the same property. Enables to collect double rent. Additionally, it enables the owner to build houses and hotels.	