SFSU CSC 868/668 Advanced Object Oriented Software Design and Development

Final Project for CSC 668-868 Spring 2015

Enigma Monopoly

April 21, 2015

Google Code Project URL:

https://[code.google.com/p/sfsu-csc-868-enigma-group](https://code.google.com/p/sfsu-csc-868-enigma-group/svn)

Group 3: Enigma

Gurpartap Gill

Derek Ma

Robert Moon

Cheryl Nielsen (Team Lead)

([cnielsen@mail.sfsu.edu](mailto:cnielsen@mail.sfsu.edu))

Sylvain Ribstein

Kenneth Robertson

John Santos

Table of Contents (to be updated after conversion to word doc)

[Milestone Five: Project Description](#h.30j0zll)

[Technologies Chosen](#h.1fob9te)

[Platform and Software Used](#h.ozxtwt47xkcp)

[System Functionality](#h.3znysh7)

[Expected time of completion](#h.tyjcwt)

[Contributions of Group Members](#h.61cjed3vrkp6)

[Challenges](#h.96hro7cfyi8w)

[Milestone One](#h.1t3h5sf)

[Multithreading](#h.4d34og8)

[Glossary of Terms](#h.aylvujejzozz)

[Use Cases](#h.26in1rg)

[Actor User](#h.lnxbz9)

[Actor Player: Chance and Community Chest Cards](#h.35nkun2)

[Actor Player](#h.1ksv4uv)

[Use Case Diagrams (revision 3)](#h.eodu8zta21ac)

[Actor User](#h.nnn501er7tsg)

[Actor Player](#h.38vr57pg04b9)

[Actor Player: Chance and Community Chest Cards](#h.93m97m8035ty)

[Use Case Descriptions (revision 2) (All Updated - format errors will be corrected after conversion to Word doc.)](#h.z337ya)

[Conceptual Model Diagram (revision 4)](#h.j7jkhzy0i2bg)

[Sequence Diagrams](#h.mef9spycv64h)

[Use Cases 1, 2, 3](#h.aapm5fh27fzd)

[Use Cases 4, 5](#h.2te7t96d9s56)

[Use Cases 6, 7](#h.ttux5l2rwl0)

[Use Cases 8, 9](#h.cx6phk47p47e)

[Use Case 10](#h.7c8nz0nk5x1g)

[Use Case 12](#h.7di86jn1b2qy)

[Use Case 13](#h.l7ofx1on8hm7)

[Use Case 14](#h.sxdy18e0xwun)

[Use Case 15](#h.2mu2b7m14pqw)

[Use Cases 16, 17](#h.xvkz1tjzk80b)

[Use Cases 18, 19](#h.3dbsmkwk4jt)

[Use Cases 20, 21](#h.ct59k0wx7rp8)

[Use Cases 22, 23](#h.olvgp0tufwbu)

[Use Cases 24, 25](#h.2idlhghvg12l)

[Use Cases 26, 27](#h.nxppzw6fetu1)

[Use Cases 28, 29](#h.bfey294orepe)

[Use Case 30](#h.dbu67bbremdq)

[Use Cases 31, 32](#h.txcshp8y1mu9)

[Use Cases 33, 34a](#h.sc89tpxty49h)

[Use Cases 34b, 35](#h.flnmxkop8mt3)

[Milestone Two](#h.1y810tw)

[Packages (revision 2)](#h.2xcytpi)

[Game Package](#h.vccgh4bcd7i)

[User Package](#h.1o4mo9rjksq)

[Property Package](#h.x0sdvavbwacx)

[Database Package](#h.wib0ddaerl18)

[Card Package](#h.dwd6z8ytnzey)

[Web Pages Package](#h.al0rnsv87yza)

[Class Descriptions and Diagrams (revision 3)](#h.4i7ojhp)

[User Class](#h.1ci93xb)

[Player Class](#h.3whwml4)

[Bank Class](#h.7oqoe9dyxy57)

[StartGame Servlet Class](#h.qsh70q)

[Bank Account Class](#h.2rptqilbgmuf)

[The Database Controller Classes](#h.2p2csry)

[Login and Register Servlet classes](#h.3o7alnk)

[Register Class](#h.23ckvvd)

[Sell Property Class](#h.32hioqz)

[GameServlet Class](#h.1hmsyys)

[Dice Class](#h.41mghml)

[Space Class](#h.2grqrue)

[Property Class](#h.vx1227)

[Real Estate Class](#h.3fwokq0)

[Railroad Class](#h.4f1mdlm)

[Utility Class](#h.lc4w7us9xkoc)

[Card and Chance and Community Chest Classes](#h.19c6y18)

[GetoutOfJailFreeCard](#h.bnhmzehuo1gk)

[MoveToLocationCard](#h.s6kd7oz0cb01)

[MoveToSpaceCard](#h.awik311spkb6)

[RealEstateRepairCard](#h.uzxd9x7s8bqs)

[StaticCashTransferToPlayerCard](#h.b0bl0obhdpv1)

[Design Overview](#h.2h4kqj3cy02g)

[Dynamic Web Content](#h.hdgk213fqr2n)

[Flow Chart Diagram](#h.44sinio)

[Design Patterns](#h.1mrcu09)

[1. Model-View-Controller Introduction](#h.46r0co2)

[MVC in Enigma Monopoly](#h.2lwamvv)

[2. Singleton Design Pattern](#h.bv6pgzx6beju)

[Milestone Three](#h.111kx3o)

[Web Pages](#h.3l18frh)

[Use Case](#h.206ipza)

[Home Page](#h.4k668n3)

[About Monopoly](#h.2zbgiuw)

[How To Play](#h.1egqt2p)

[Register Page](#h.3ygebqi)

[Login Page](#h.2dlolyb)

[Lobby Page](#h.sqyw64)

[Game Page](#h.3cqmetx)

[Table of Contents](#h.rr4xsvlf2qr7)

[Introduction](#h.c6d8eqmjyaiz)

[Home Page](#h.6sitl0y3enhf)

[About Monopoly](#h.vug02xlbovh5)

[How To Play](#h.tbtk6lj404r)

[New User Registration](#h.tkk1c8jghc59)

[Login](#h.oa8soz5h9vk1)

[Lobby Page](#h.r9bwr6rof2ot)

[Game Page](#h.amvs14f1k2k8)

[Game Page Actions](#h.nv8jkruuored)

[Community or Chance Card Space](#h.j4619w6guxor)

[Free Parking](#h.ejfhk8vbp00g)

[Property Space Pop Up Message](#h.g4mj0ixovpnt)

[Landing on “Going to Jail” Space](#h.8uvk0u5lfx9f)

[User is In Jail](#h.jawlllg7l6jv)

[Get Out of Jail Pop Up Message](#h.rplq9ct49ery)

[Passing “Go” Space](#h.l2m331cqnln3)

[Pay Tax Pop Up Message](#h.qrm88vfsrb5h)

[User Lands on “Visiting Jail” Space](#h.4eps4skkl4po)

# Milestone Five: Project Description

For our project we have chosen to do an object oriented web based version of the classic board game of Monopoly. This is the multiplayer game of real estate tycoons where players buy and sell properties, utilities, and railroads.

## **Technologies Chosen**

For the backend we will be using Java, for the game logic and objects, with MySQL for the relational database. For the frontend we will be using the HTML5 technologies including JavaScript, jQuery, CSS, and HTML Canvas, or SVG, and Bootstrap. For dynamic content we will be using a combination of HTML5, JSP, and Servlets. Game states will be saved to the server database and displayed to each player via the Web GUI interface.

The game will use object oriented design for several aspects of the game including the game board, game cards, and game actions. The game board will have various type of game spaces that have some common functions and members they can all inherit like location and price. The subspace types will also have individual features like railroads, real estate properties, utilities, and others. The cards will also have a similar inheritance structure for railroads, real estate properties, utilities, community chest, and chance cards. We will also use objects for the game states of the players, the game board, and the bank.

## **Platform and Software Used**

* The Classes were written in Java 8 and Java EE 7 Servlets.
* Web Languages:
  + JavaScript 1.8.5, jQuery 1.11.2, HTML 5 with HTML Canvas, CSS 3
  + Java Server Pages (Java EE 7 JSP)
* Relational Database MySQL 5.6
* Adobe Photoshop Elements 13
* Development tools:
  + Netbeans IDE 8.0.2 and MySQL Workbench 6.3
* Collaboration Tools:
  + Google Drive, Google Code SVN, and Skype
* Deployed on:
  + Web Server Apache TomCat 8.0.15
  + Operating Systems: Window 7 and 8, Apple OS X 10.10
  + Web Browsers: Firefox 37 and Google Chrome 42 and Internet Explorer 11

## **System Functionality**

A typical user would go to the game website where he/she could look at the game rules and decide to play the game. To play a game the user would need to log in. They would then be taken to the game lobby to join up with other players to start a game with a total of 2 to 4 players. Once a game has started players are randomly assigned a game piece, a turn order, and given starter money. When a player’s turn comes they can roll the dice and play the game according to the rules of classic Monopoly. When a player lands on a property they can pay rent to the property owner, buy property, or buy houses and hotels to put on property that they own. The bank will not be assigned to a player; instead it will be played by the server computer. The bank will sell houses, hotels, and unowned properties. The bank is also in charge of auctions when they are needed. A player can win the game in two ways: a player can win by having the most money in cash and properties at the end of a set time of play, or a player can win by being the last player left after all other players have gone bankrupt.

## **Expected time of completion**

All timeline goals were met. However this required redefining the project scope in order for the game to be completed on time. This required prioritizing which features were the most important to focus our remaining time on in order to have a minimum usable project ready for delivery on the final due date. This meant that several classic features of Monopoly were eliminated or assigned low priority and then later they were not implemented. This included the following items:

* Bank Auctions of property and houses were eliminated, because it required implementing a chat feature that we did not have time to look into.
* Hotels were eliminated and 5 houses used instead to simplify coding.
* Properties can not be sold to other Players. So properties and can only be purchased from the Bank. Again this is related to the lack of a chat feature.
* Get out of Jail by paying the fine of $50, instead of tracking dice rolls, etc.
* Community Cards and Chance Cards were coded, but we ran out of time to connect them to the game code. So landing on one of the Card spaces on the game board currently causes no action to occur.
* Currently a game only has one Player.

## **Contributions of Group Members**

**Cheryl Nielsen**:

Team Lead and Tech Lead

Helped design the project, assigned responsibilities, arranged meetings, updated scrum documents, and was available for questions.

Wrote and edited documentation for the Milestones with the help of others in the group.

Made sure that all assignments were turned in on time.

Help with software installation and setup for the team.

MySQL Database design and implementation for the project.

Wrote code for the Database Package of six database controllers that were used as wrappers for the SQL queries and database connections for all database access.

Wrote Java Code: GameDatabaseController.java, UserPlayerDatabaseController.java, BankDatabaseController.java, RealEstateDatabaseController.java, UtilityDatabaseController.java, RailroadDatabaseController.java,

Reviewed and Edited: Property Package, User.java, Player.java, Bank.java, BankAccount.java

**John Santos**:

Front End Expert: The Web Pages Package HTML and JSP

Wrote and edited documentation for the Milestones with the help of others in the group.

Wrote Java Code: Bank.java, BankAccount.java

Wrote GUI Code: about.html, how-to-play.html, lobby.jsp, index.jsp, login.html, monopoly.css, navbar.html, noSuccess.html, register.html, registrationSuccess.html, edit\_user\_info.html

Wrote User Guide

Created older version of Use Case diagrams

**Gurpartap Gill**:

Tech Lead 2: Servlets, JSP, JavaScript, jQuery and AJAX.

Wrote and Developed: GameServlet.java, Register.java (Servlet), Login.java (Servlet), gamePage.jsp, monopoly.js (JavaScript)

Reviewed and Edited: index.jsp, login.html, register.html, lobby.jsp.

Wrote and edited documentation for the Milestones with the help of others in the group.

Description of Work Done: Developed gamePage.jsp using HTML canvas element for the game board to move player’s token around and perform actions regards to player’s token position on the game board. Developed JavaScript file monopoly.js to update game page content. Used jQuery and AJAX to make Server calls sending HTTP requests from gamePage.jsp to GameServlet and handling responses from there.

**Robert Moon:**

Wrote Code: startGame.java (Servlet)

Reviewed and Edited: Register page, Login page, Lobby page

Other Work: I developed background knowledge concerning servlets and JSP by reading a book on the topic. Wrote and edited documentation for the Milestones with the help of others in the group.

Descriptions of Work Done: I create session/application scoped variables to hold user and player information between stateless pages in our Monopoly game. I modified the Register page of our game to save user information to the database. I modified the Login page to check if the username and password the user enters in the login page exist in the database. I handled the Lobby page.

**Derek Ma:**

Wrote and Developed: Property.java, RealEstate.java, Utility.java, Railroad.java

Edited: Dice.java, GameServlet.java, gamePage.jsp

Other Work Done: Wrote class diagrams for CommunityChest, Chance, and Card. Wrote Use Case diagrams for all the Player actor interactions

Wrote and edited documentation for the Milestones with the help of others in the group.

**Kenneth Robertson:**

Code Written: Space.java, Dice.java, Property.java, RealEstate.java, Utility.java, Railroad.java, Card.java, CashTransferToPlayerCard.java, GetOutofJailFreeCard.java, MoveToSpaceCard.java,

RealEstateRepairCard.java, StaticCashTransferCard.java, CardDeck.java classes

Code Reviewed and edited: User.java, Player.java, Token.java, GameServlet.java, gamePage.jsp

Other Work Done: Developed class model for property classes and card classes. Developed game logic for GameServlet and gamePage interactions. Did class diagrams for property and card classes. Created wireframes for monopoly player to player trading, game page, lobby page,

and initial page. Wrote and edited documentation for the Milestones with the help of others in the group.

## **Challenges**

* Had some difficulty getting good communication with the rest of the team about what work should be prioritized.
* The meetings were not held on the same day of week and time of day, and were usually not when everybody was on campus.
* Difficulty remembering meeting times and days because of lack of fixed day and time.
* Other classes projects interfered with the time we had to do work on project on a couple of occasions.
* Lack of knowledge about internet programming made it difficult to figure out how to program the javascript and how to get it to interact with the servlets.
* Confusion over the structure of the classes Game, GameBoard, Player. Bank and BankAccount weren't written as quickly as we would have liked.
* Not all members attended every meeting leading to conflicts on work assignment, and members being out of touch with what was happening.
* It was challenging to use JSP or servlets with Java Script. Java Script is handled on the client side, and JSP/servlets are handled on the server side. Using scriptlets, it was difficult to figure out how to update information beyond when the page loads.

# Milestone One

## **Multithreading**

Threading was not used in the final version. The client side code was not as complex as expected. This is mainly because more work was done by calls to the server side than expected. Also time delays were not an issue. Threading may still be used for game updates, such as when a player moves a token on the board, and the result needs to be displayed on all other player’s screens. However this was not implemented in the final version due to change of scope.

## **Glossary of Terms**

* User: Someone just browsing the website, or it is someone who has logged in but not yet joined into a game, or it is someone who has left a game, but not yet logged out.
* Player: A player is a user who has joined a game.
* Bank: The automated Monopoly game bank and auctioneer.
* Double: When a player rolls the die with the same numerical value (e.g. rolling two die with the value of 3).
* Opponent: An opposing player.
* Monopoly: Owning all properties of a particular color or type.
* Space: A game square on the monopoly board.
* Token: A player’s game piece in a match that determines the position of the player on the game board.
* Home Page: A website page that is viewable without logging in.
* Net Worth: A Player’s total value of all assets including only the property/houses/hotels/Utilities owned by Player, Not on Mortgage.
* Chance and Community Chest cards: contain instructions that players need to follow when they land on one of these spaces.

## **Use Cases**

In the Use Cases we have the actors User, Player, and Banker.

### Actor User

1. Browsing the website
2. Register
3. Login
4. Logout
5. Get forgotten login username and password
6. Change password
7. Change token
8. Start a game for 2 to 4 players

### Actor Player: Chance and Community Chest Cards

1. Card: Bank pays Player
2. Card: Player pays Bank
3. Card: Player pays all other Players
4. Card: Player is paid by all other Players
5. Card: Move token to a specific Game Board space
6. Card: Advance token to nearest Utility, with 10 \* dice rent paid.
7. Card: Advance token to the nearest Utility or Railroad, with double the rent amount paid.
8. Draw Go directly to Jail Card
9. Draw Get out of Jail Free Card
10. Use Get out of Jail Free Card

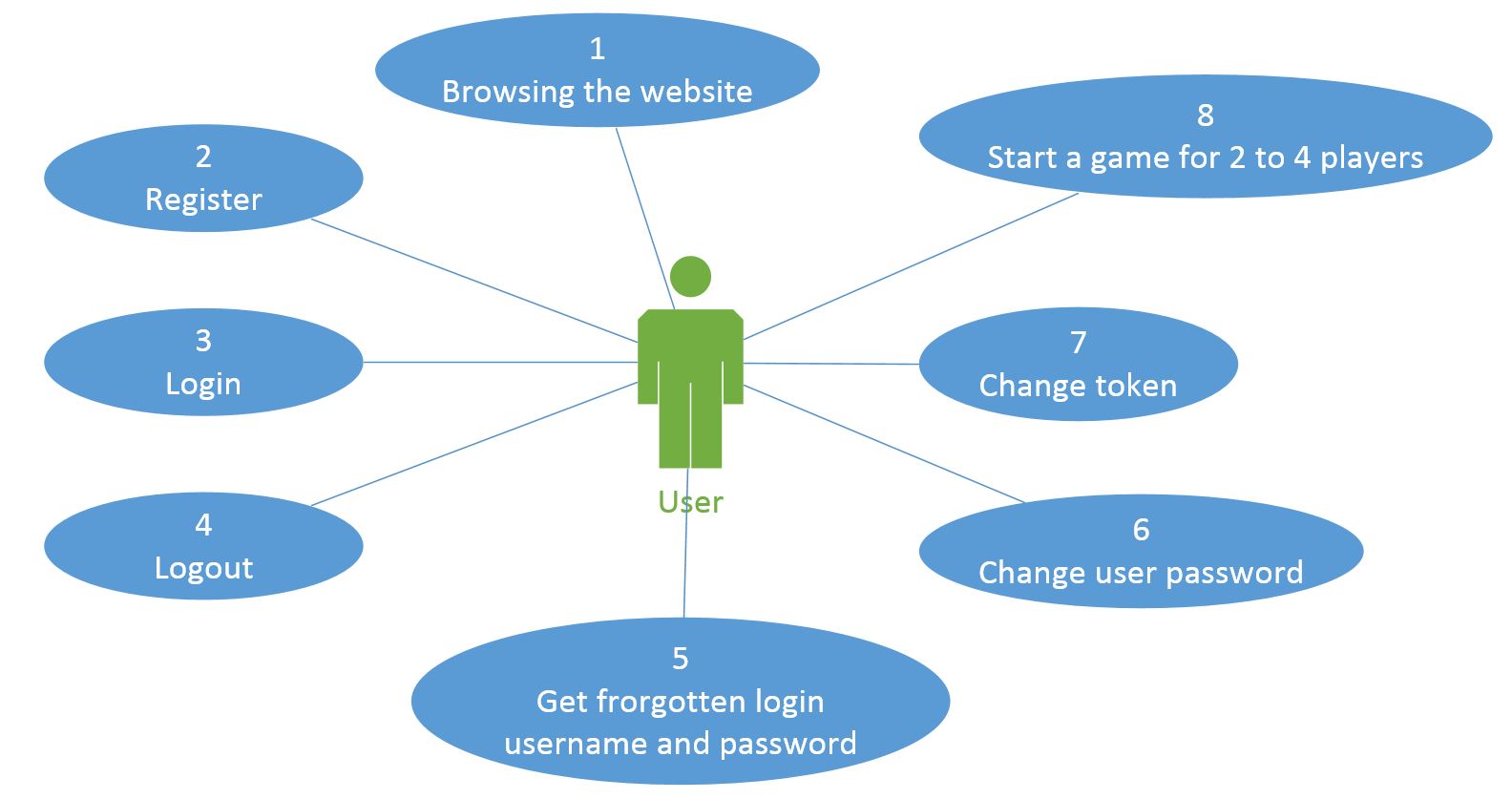
### Actor Player

1. Game Starter Money
2. Manage mortgages
3. Collect fees and taxes
4. Pay Salary
5. Bank auctions property/houses/hotels (low priority item)
6. Player assigned a game turn order
7. Take a turn
8. Buy houses/hotels from the Bank
9. Buy properties from the Bank
10. Sell houses/hotels to Bank
11. Buy/sell/trade properties to other Players
12. Go bankrupt
13. Pay Rent
14. Go to Jail
15. Get out of Jail
16. Player leaves the game
17. Join a game in progress

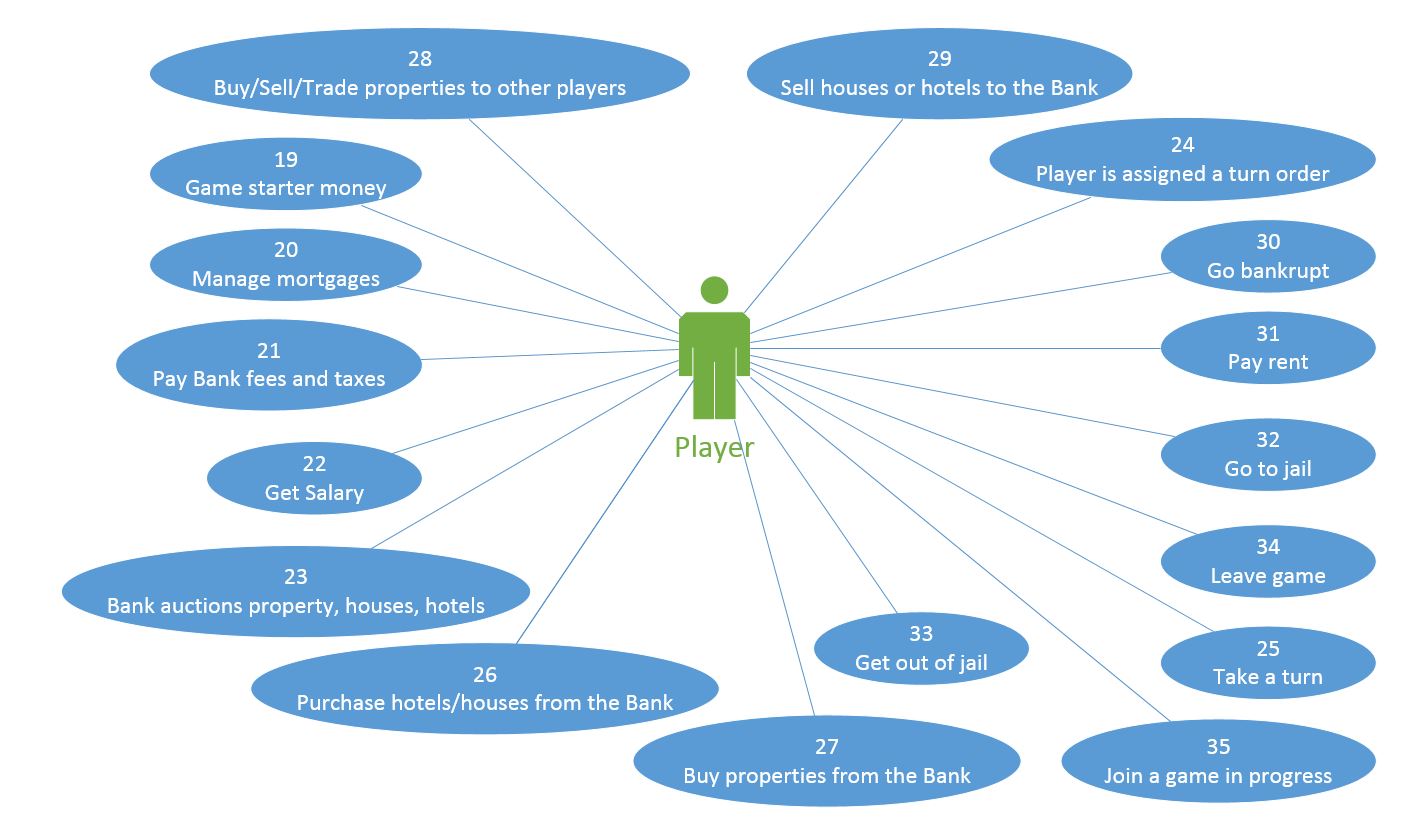
## 

## **Use Case Diagrams (revision 3)**

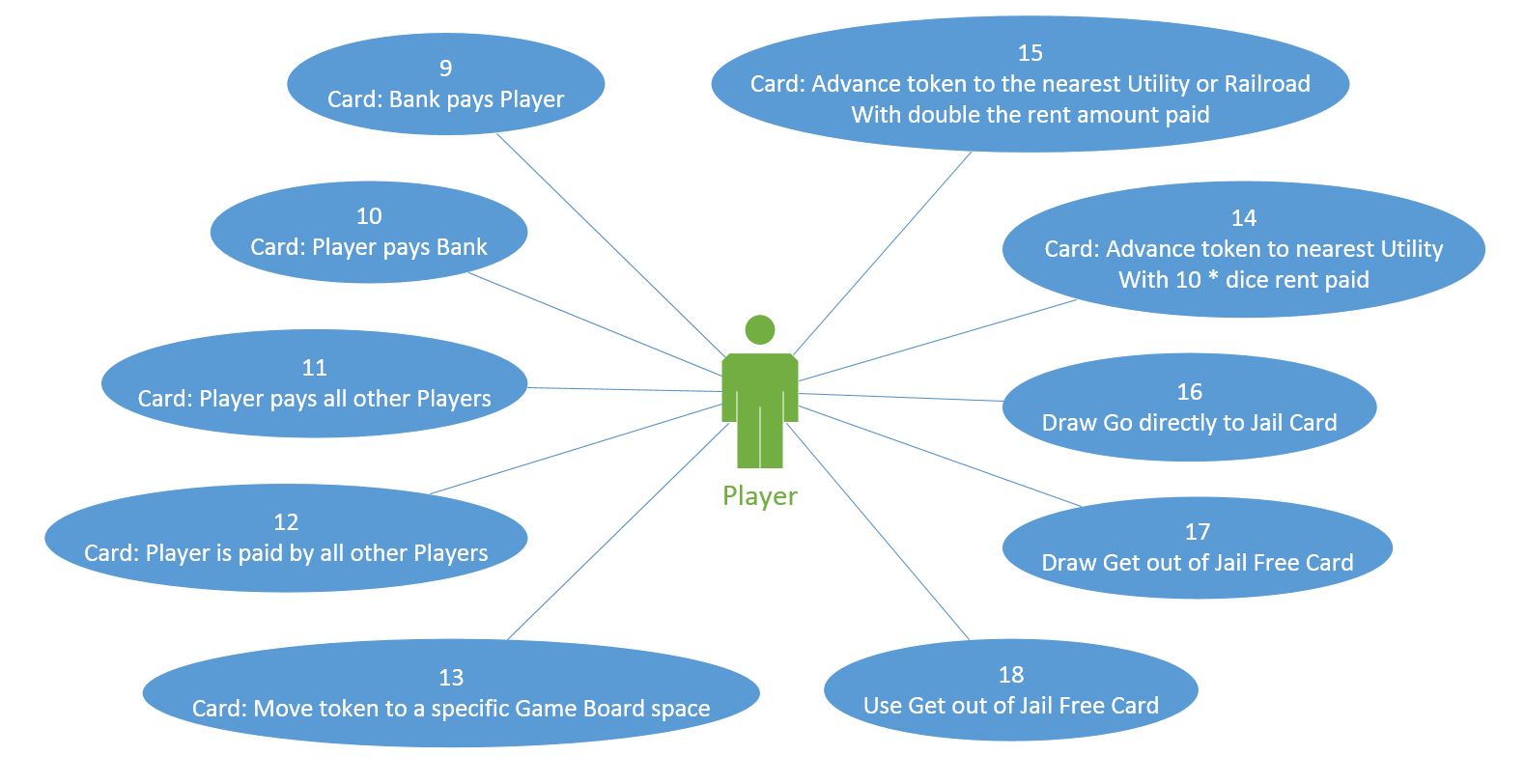
### Actor User



### Actor Player



### Actor Player: Chance and Community Chest Cards



## 

## 

## 

## **Use Case Descriptions (revision 2) (All Updated - format errors will be corrected after conversion to Word doc.)**

|  |  |
| --- | --- |
| **Use Case #** | 1 |
| **Use Case Name** | Browsing the website |
| **Summary** | A user looks at the website to find out about the game, and to decide if they want to become a registered user. |
| **Dependency** | none |
| **Actor** | User |
| **Precondition** | none |
| **Description** | * A user goes to the website and browses the public pages to find out how the game works, and to decide if they are interested in becoming a registered user. |
| **Alternative** | none |
| **Postcondition** | none |

|  |  |
| --- | --- |
| **Use Case #** | 2 |
| **Use Case Name** | Register |
| **Summary** | A user decides to register so they can play Monopoly. |
| **Dependency** | none |
| **Actor** | User |
| **Precondition** | none |
| **Description** | * A user chooses the option to register so they can play Monopoly. * The user enters a first name, last name, email address, login name, and password. * The user can the choose from a set of default Avatars or choose to upload an image to use as an Avatar. |

|  |  |
| --- | --- |
|  | * The user then gets a confirmation that they have registered successfully, or that they have already registered, or that they need to re-enter their information. |
| **Alternative** | * If previously registered, the user is given an option to retrieve their forgotten user information as described in Use Case 5. |
| **Postcondition** | The user registration information has been saved to the database. |

|  |  |
| --- | --- |
| **Use Case #** | 3 |
| **Use Case Name** | Login |
| **Summary** | A user logs in to play Monopoly. |
| **Dependency** | none |
| **Actor** | User |
| **Precondition** | The user has already registered. |
| **Description** | * A user chooses the option to login from the home page so they can play Monopoly. * The user enters their login name, and password. * If successful, the user is then taken to the game lobby webpage. * If not successful, the user remains on the homepage and is given a message to try again or register. |
| **Alternative** | none |
| **Postcondition** | The user is logged in and taken to the game lobby webpage. |

|  |  |
| --- | --- |
| **Use Case #** | 4 |
| **Use Case Name** | logout |
| **Summary** | A user logs out of the game website. |
| **Dependency** | none |
| **Actor** | User |

|  |  |
| --- | --- |
| **Precondition** | The user is currently logged in, and not participating in a game. |
| **Description** | * A user is in decides to leave the website. * The user chooses the option to logout. * The user is then taken to the homepage. |
| **Postcondition** | The user is logged out of the website. |

|  |  |
| --- | --- |
| **Use Case #** | 5 |
| **Use Case Name** | Get forgotten login username and password. |
| **Summary** | A user needs to get a forgotten login username and password. |
| **Dependency** | none |
| **Actor** | User |
| **Precondition** | none |
| **Description** | * A user cannot login because they have forgotten their username or password. * The user selects the option for forgotten login. * The user is taken to a page to enter their name and email address. * If the name and email address are in the database of registered users, then a confirmation message is displayed with the login and password. * If the name and email address are not in the database of registered users, then a message is displayed that says they are not registered with links to the registration page and the home page. |
| **Alternative** | * (low priority item) * If the name and email address are in the database of registered users, then a confirmation message is displayed, and the login and password are emailed to the user with a reminder to change the password. |
| **Postcondition** | The user has the login information they need to login. |

|  |  |
| --- | --- |
| **Use Case #** | 6 |
| **Use Case Name** | Change user password. |
| **Summary** | A user changes their password. |
| **Dependency** | none |
| **Actor** | User |
| **Precondition** | A user has logged in, and is in the lobby. |
| **Description** | * The user selects the option to change their password. * The user is taken to the password change page and enters their login name, old password, and new password. * If the information matches what is in the database, then the user gets a confirmation that the password has been changed, and   the database is updated.   * If the change of password fails, the user gets a failure message and is asked to try again. |
| **Alternative** | * If the password change is successful, an email message is sent to the user that their password for the website has been changed. |
| **Postcondition** | The user registration has been updated in the database. |

|  |  |
| --- | --- |
| **Use Case #** | 7 |
| **Use Case Name** | Change token |
| **Summary** | A user gets a new image assigned or uploads an image to use as an avatar. |
| **Dependency** | none |
| **Actor** | User |
| **Precondition** | A user has logged in, and is in the lobby. |
| **Description** | * A user chooses the option to change their avatar from the lobby. * The user chooses a new avatar from a list of random images, or the user selects to upload an image as their avatar. * If successful the user’s new avatar is displayed and saved to the database. |

|  |  |
| --- | --- |
|  | * If not successful, the user is given an error message and asked to try again. |
| **Alternative** | none |
| **Postcondition** | The user has a new avatar saved to the database, and displayed on the screen. |

|  |  |
| --- | --- |
| **Use Case #** | 8 |
| **Use Case Name** | Start a game for 2 to 4 players |
| **Summary** | A User starts a new game, and becomes a new Player. |
| **Dependency** | none |
| **Actor** | User, Player |
| **Precondition** | User must be logged in. |
| **Description** | * User starts a match from the lobby. * A match is created * The User becomes a Player, and waits for opponents to join the match. * The database is updated to associate that User with the Player, and the Player with the new Game. |
| **Alternative** | Match start   * A match can be started by the Player once enough other Players have joined. |
| **Postcondition** | * A game is created from the User’s actions. * A User is now associated with that Player and Game in the database. * The player then waits for others to join the game. |

|  |  |
| --- | --- |
| **Use Case #** | 9 |
| **Use Case Name** | Card: Bank pays Player |

|  |  |
| --- | --- |
| **Summary** | A Player draws a Chance Card or Community Chest Card, which gives them money, but not from another Player. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * The Player during their turn has landed on a Chance or Community Chest Card Space on the Board. |
| **Description** | * The Player draws the card from the card queue. * The Player’s Bank account is credited for the amount listed on the card. * The card is returned to the card queue. |
| **Alternative** | none |
| **Postcondition** | The card in back in the card queue, and the Player’s Bank account has been credited. |

|  |  |
| --- | --- |
| **Use Case #** | 10 |
| **Use Case Name** | Card: Player pays Bank |
| **Summary** | A Player draws a Chance Card or Community Chest Card, which requires them to pay money, but not to another Player. |
| **Dependency** | none |
| **Actor** | Player, Bank |
| **Precondition** | * The Player during their turn has landed on a Chance or Community Chest Card Space on the Board. |
| **Description** | * The Player draws the card from the card queue. * The Player’s Bank account is debited for the amount listed on the card. * The card is returned to the card queue. |
| **Alternative** | * If the Player does not have sufficient funds in their Bank account to pay the tax, then the appropriate Use Cases for selling, mortgages, and bankruptcy may be applied. |
| **Postcondition** | The card in back in the card queue, and the Player’s Bank account has been debited. |

|  |  |
| --- | --- |
| **Use Case #** | 11 |
| **Use Case Name** | Card: Player pays all other Players |
| **Summary** | A Player draws a Chance Card or Community Chest Card, which requires them to pay all other Players. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * The Player during their turn has landed on a Chance or Community Chest Card Space on the Board. |
| **Description** | * The Player draws the card from the card queue. * The Player’s Bank account is debited for the amount listed on the card \* the number of other Players. * All other Players’ Bank accounts are credited for the amount listed on the card. * The card is returned to the card queue. |
| **Alternative** | * If the Player does not have sufficient funds in their Bank account to pay the tax, then the appropriate Use Cases for selling, mortgages, and bankruptcy may be applied. |
| **Postcondition** | The card in back in the card queue, and the Players’ Bank accounts have been credited or debited. |

|  |  |
| --- | --- |
| **Use Case #** | 12 |
| **Use Case Name** | Card: Player is paid by all other Players |
| **Summary** | A Player draws a Chance Card or Community Chest Card, which gives them money from all other Players. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * The Player during their turn has landed on a Chance or Community Chest Card Space on the Board. |
| **Description** | * The Player draws the card from the card queue. |

|  |  |
| --- | --- |
|  | * The Player’s Bank account is credited for the amount listed on the card \* the number of other Players. * All other Players’ Bank accounts are debited for the amount listed on the card. * The card is returned to the card queue. |
| **Alternative** | none |
| **Postcondition** | The card in back in the card queue, and the Players’ Bank accounts have been debited or credited. |

|  |  |
| --- | --- |
| **Use Case #** | 13 |
| **Use Case Name** | Card: Move token to a specific Game Board space |
| **Summary** | A Player draws a Chance Card or Community Chest Card, which requires them move their token to a specific space on the Game Board. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * The Player during their turn has landed on a Chance or Community Chest Card Space on the Board. |
| **Description** | * The Player draws the card from the card queue. * The Player’s token is moved on the Game Board to the named location or by the number of spaces listed on the card. * The card is returned to the card queue. |
| **Alternative** | * If rent needs to be paid to the owner of the property on the space that is landed on by the Player, then see the appropriate Use Cases for payment of rent. * If the player lands on unowned property they can buy it, see the appropriate Use Case for purchase of property. |
| **Postcondition** | The card in back in the card queue, and the Player’s token has been moved. |

|  |  |
| --- | --- |
| **Use Case #** | 14 |
| **Use Case Name** | Card: Advance token to nearest Utility, with 10 \* dice rent paid. |

|  |  |
| --- | --- |
| **Summary** | A Player draws a Chance Card or Community Chest Card, which requires them move to a Utility and pay an altered rent amount. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * The Player during their turn has landed on a Chance or Community Chest Card Space on the Board. |
| **Description** | * The Player draws the card from the card queue. * The Player’s token is moved forward to the nearest Utility space. * If the Utility is owned   + The Player throws the dice   + The Player’s Bank account is debited for 10 times the amount thrown on the dice. * The card is returned to the card queue. |
| **Alternative** | * If the player lands on unowned property they can buy it, see the appropriate Use Case for purchase of property. * If the player does not have sufficient Bank account funds to pay rent, see the appropriate Use Cases. |
| **Postcondition** | The card in back in the card queue, and the Player’s Bank account has been debited. |

|  |  |
| --- | --- |
| **Use Case #** | 15 |
| **Use Case Name** | Card: Advance token to the nearest Utility or Railroad, with double the rent amount paid. |
| **Summary** | A Player draws a Chance Card or Community Chest Card, which requires them move to a Utility or Railroad and pay twice the rent amount. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * The Player during their turn has landed on a Chance or Community Chest Card Space on the Board. |
| **Description** | * The Player draws the card from the card queue. * The Player’s token is moved forward to the nearest Utility or Railroad space on the Game Board. * If the Utility or Railroad is owned |

|  |  |
| --- | --- |
|  | * The Player’s Bank account is debited for twice the rent that the owner is currently entitled to be paid. * The Railroad or Utility owner’s Bank account is credited for that same rent amount. * The card is returned to the card queue. |
| **Alternative** | * If the player lands on unowned property they can buy it, see the appropriate Use Case for purchase of property. * If the player does not have sufficient Bank account funds to pay rent, see the appropriate Use Cases. |
| **Postcondition** | The card in back in the card queue. |

|  |  |
| --- | --- |
| **Use Case #** | 16 |
| **Use Case Name** | Draw Go directly to Jail Card |
| **Summary** | A Player draws the “Go directly to Jail” Chance Card or Community Chest Card. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * The Player during their turn has landed on a Chance Space or Community Chest Space on the Board. |
| **Description** | * The Player draws the “Go directly to Jail” Chance Card or Community Chest Card from the card stack. * The Player’s token is moved to the Jail space. * If the Player passes GO on the way to Jail, the Player does not get paid $200 salary. * The card is returned to the card queue. |
| **Alternative** | none |
| **Postcondition** | The card in back in the card queue, and the Player is in Jail. |

|  |  |
| --- | --- |
| **Use Case #** | 17 |
| **Use Case Name** | Draw Get out of Jail Free Card |
| **Summary** | A Player draws the a Get out of Jail Free Card. |

|  |  |
| --- | --- |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * The Player during their turn has landed on a Chance Space or Community Chest Space on the Board. |
| **Description** | * A Player draws the a Get out of Jail Free Card from the card queue. * The Player keeps the card until they need it or sell it. |
| **Alternative** | none |
| **Postcondition** |  |

|  |  |
| --- | --- |
| **Use Case #** | 18 |
| **Use Case Name** | Use Get out of Jail Free Card. |
| **Summary** | A Player uses the a Get out of Jail Free Card. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * The Player is in Jail. * The Player already has the Get out of Jail Free card. * The Player has rolled the dice, but it did not get them out of Jail. |
| **Description** | * The Player uses the card to get out of Jail for free. * The Player moves their token the number of spaces on the dice. * The card is returned to the card queue. |
| **Alternative** |  |
| **Postcondition** | The Player is out of Jail, and the card is back in the card queue. |

|  |  |
| --- | --- |
| **Use Case#** | 19 |
| **Use Case Name** | Game Starter Money |

|  |  |
| --- | --- |
| **Summary** | The Bank gives each player money when they first join the game. |
| **Dependency** | None |
| **Actor** | Player |
| **Precondition** | The Player has just joined the game. |
| **Description** | * The Bank credits each Player’s bank account for $1500 when they join the game. |
| **Alternative** | None |
| **Postcondition** | Player gets $1500 in their Bank Account. |

|  |  |
| --- | --- |
| **Use Case#** | 20 |
| **Use Case Name** | Manage Mortgages |
| **Summary** | Bank manages mortgage creation and payments. |
| **Dependency** | None |
| **Actor** | Player |
| **Precondition** | * A player has a property on which he/she has a current Mortgage. * Or the Payer wants a new Mortgage on a property. * No houses or hotels are on the property. |
| **Description** | * Player’s Bank account will be debited/credited for the Mortgage value. * Player is prohibited from collecting rent on Mortgaged property.   + Rent value of property is set to zero. |
| **Alternative** | none |



|  |  |
| --- | --- |
| **Use Case #** | 21 |
| **Use Case Name** | Pay Bank fees and taxes |
| **Summary** | Player pays a tax or fee to the bank. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * It is the Player’s turn. * The Player owes a fee or tax as a result of actions taken during their turn. |
| **Description** | * The Player’s Bank account is debited for the amount of the tax. |
| **Alternative** | * If the Player does not have sufficient funds in their Bank account to pay the tax, then the appropriate Use Cases for selling, mortgages, and bankruptcy may be applied. |
| **Postcondition** | Player’s account is debited for the amount of tax. |

|  |  |
| --- | --- |
| **Actor** | Player |
| **Use Case #** | 22 |
| **Use Case Name** | Get Salary |
| **Summary** | Player passes or lands on the GO space on the board, and collects their salary. |
| **Dependency** | none |
| **Precondition** | * It is the Player’s turn. * Player’s token passes or lands on the GO space. |
| **Description** | * The Bank credits the Player’s account with a salary of $200, unless the Player is on the way to Jail. |
| **Alternative** | none |
| **Postcondition** | Player’s account is credited with $200. |

|  |  |
| --- | --- |
| **Use Case #** | 23 |
| **Use Case Name** | Bank auctions property/houses/hotels (low priority due to chat technology) |
| **Summary** | Players purchase property/houses/hotels from the Bank during an auction sale. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * The property/houses/hotels are owned by the bank due to a bankruptcy, a Player leaving the game, or a Player choosing not to buy an unowned property that they landed on. |
| **Description** | * If a player lands on an unowned property and does not want to buy it, then the bank must auction it. * If a player goes bankrupt, causing the bank to get all their property/houses/hotels, then the bank must auction them. * bids can start at any price. * The highest bid wins the auction for a particular item. * House/hotel rules of property buildup still apply. * The Bank debits the Player’s account for the listed purchase price of that property/house/hotel. |

|  |  |
| --- | --- |
|  | * The Player receives ownership for that property/house/hotel. * The database is updated with the new ownership information. |
| **Alternative** | none |
| **Postcondition** | The auction winner’s account has been debited, and the auction winner now own that property/hotel/house. |

|  |  |
| --- | --- |
| **Use Case #** | 24 |
| **Use Case Name** | Player is assigned a turn order. |
| **Summary** | Each player in the match is assigned a turn order after the match has started. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | A match has been started by the host |
| **Description** | * A each Player rolls the dice, and the turn orders are assigned from highest to lowest roll of the dice. * Or a random number for turn order is assigned to each Player. |
| **Alternative** | * If a Player joins a game already in progress, they have the last turn order available. |
| **Postcondition** | Match starts beginning with the player who goes first |



|  |  |
| --- | --- |
| **Use Case Name** | Take a turn |
| **Summary** | Player takes an action when it is currently his/her turn |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * It is the player’s turn |
| **Description** | * Player rolls the dice. * Player’s token is moved the number of spaces shown on the dice. * If the Player has rolled doubles, then they take another turn immediately. * If the Player has rolled doubles a third time their turn is over, and they go to directly to Jail without passing GO. |
| **Alternative** | * If the player is in Jail, see the Use Cases for Get Out of Jail. |
| **Postcondition** | Player continues their turn by taking an appropriate action based on the space where they landed. |

|  |  |
| --- | --- |
| **Use Case #** | 26 |
| **Use Case Name** | Purchase hotels/houses from the Bank |
| **Summary** | A Player purchase houses/hotels from the Bank. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * It is the Player’s turn. * Player owns all the real estate properties of a particular color. |

|  |  |
| --- | --- |
| **Description** | * The Player can only buy houses/hotels from the Bank if the following rules are followed.   + max of 4 houses or 1 hotel per property   + properties must be built up evenly within a single color group   + houses/hotels can only be put on a property when all of that color group is owned by the same Player   + hotels can only be put on a property when all properties in that color group have 4 houses or a hotel on them   + It is the Player’s turn to play. * The price of the house/hotel is based on the price table for the property where the house/hotel will be located. * The Bank debits the Player’s account for the required price of a house or hotel for that property. * The house/hotel is placed on the property that it was purchased for, and the database is updated with the information. |
| **Alternative** | none |
| **Postcondition** | The Player’s Bank account has been debited for the correct amount, and the house or hotel has been added to the correct property. |

|  |  |
| --- | --- |
| **Use Case #** | 27 |
| **Use Case Name** | Buy properties from the bank |
| **Summary** | Players purchase unowned properties from the bank. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * It is the Player’s turn. * Player’s token lands on an unowned property. * Player has decided to buy that property. |
| **Description** | * The Bank debits the Player’s account for the listed purchase price of that property. |
|  | * The Player receives ownership for that property. * The database is updated with the new property ownership information. |
| **Alternative** | none |
| **Postcondition** | The Player’s account has been debited, and the Player now owns that property. |

|  |  |
| --- | --- |
| **Use Case #** | 28 |
| **Use Case Name** | Sell houses or hotels to the Bank |
| **Summary** | Players sell houses or hotels from a given property to the bank at half price. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * It is the Player’s turn. * Player’s property contains houses or a hotel. |
| **Description** | * Player must sell houses/hotels in an even distribution from the properties within a given color group.   + A Player cannot sell multiple houses from only one property within a color group. * A Player’s Bank account is credited for half the cost of a house/hotel for each house/hotel sold to the bank. * The house/hotel is removed from the Player’s property, and database listing. |
| **Alternative** | none |
| **Postcondition** | The Player’s Bank account has been credited, and the house/hotel information for the property has been updated. |

|  |  |
| --- | --- |
| **Use Case #** | 29 |
| **Use Case Name** | Buy/Sell/Trade properties to other Players |
| **Summary** | Player buys sells or trades properties to another Player. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * It is the Player’s turn. |
| **Description** | * Any houses/hotels on the property must be sold to the Bank at half price following the appropriate Use Case rules. * The Players involved in the transaction come to an agreement for the price or trade of the properties included in the transaction. * The Bank debits/credits the Players accounts for the agreed on amounts of cash if needed. * The database is updated with the new property ownership information. |
| **Alternative** | none |
| **Postcondition** | The Player’s accounts has been debited/credited, and the property ownership has been transferred. |

|  |  |
| --- | --- |
| **Use Case #** | 30 |
| **Use Case Name** | Go bankrupt |
| **Summary** | Player goes bankrupt taking him/her out of the match. |

|  |  |
| --- | --- |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * It is the Player’s turn. * Player has insufficient funds to pay the Bank or another Player. |
| **Description** | * Bankrupt Player owed money to the Bank:   + All hotels/houses are sold to the Bank.   + All money is debited from the Player’s Bank account.   + All properties are given to the Bank. * Bankrupt Player no longer receives any turns. * Bank auctions the properties.   + (low priority item, may not be included) |
| **Alternative** | * Bankrupt Player owed money to another Player:   + All hotels/houses are sold to the Bank.   + All money is debited from the Player’s Bank account, and credited to the other Player’s Bank account.   + All properties are given to the other Player. * Bankrupt Player no longer receives any turns. |
| **Postcondition** | Player may watch the rest of the match or exit to the game lobby. |

|  |  |
| --- | --- |
| **Use Case #** | 31 |
| **Use Case Name** | Pay Rent |
| **Summary** | Player pays rent to the owner when they land on an opponent’s property. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * Player has moved their token. |

|  |  |
| --- | --- |
|  | * Player’s token lands on an opponent’s property. |
| **Description** | * Player must pay a certain amount of money to the owner of the property. * If Player has enough funds, their Bank Account is debited for the amount of rent, and the property Owner’s account is credited for the rental amount. * If not enough funds, player may sell/mortgage property or sell houses/hotels - (see the corresponding use cases for descriptions). |
| **Alternative** | * If the Player does not have sufficient funds in their Bank account to pay the tax, then the appropriate Use Cases for selling, mortgages, and bankruptcy may be applied. |
| **Postcondition** | Player continues their turn by taking an appropriate action or end his/her turn. |

|  |  |
| --- | --- |
| **Use Case #** | 32 |
| **Use Case Name** | Go to jail |
| **Summary** | Player goes directly to jail when landing on the jail space. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * It is the Player’s turn. * Player lands on the Jail space, or throws double dice 3 times in a row. |
| **Description** | * Player goes directly to jail. * Player does not pass GO, and does not collect a salary. |
| **Alternative** | none |



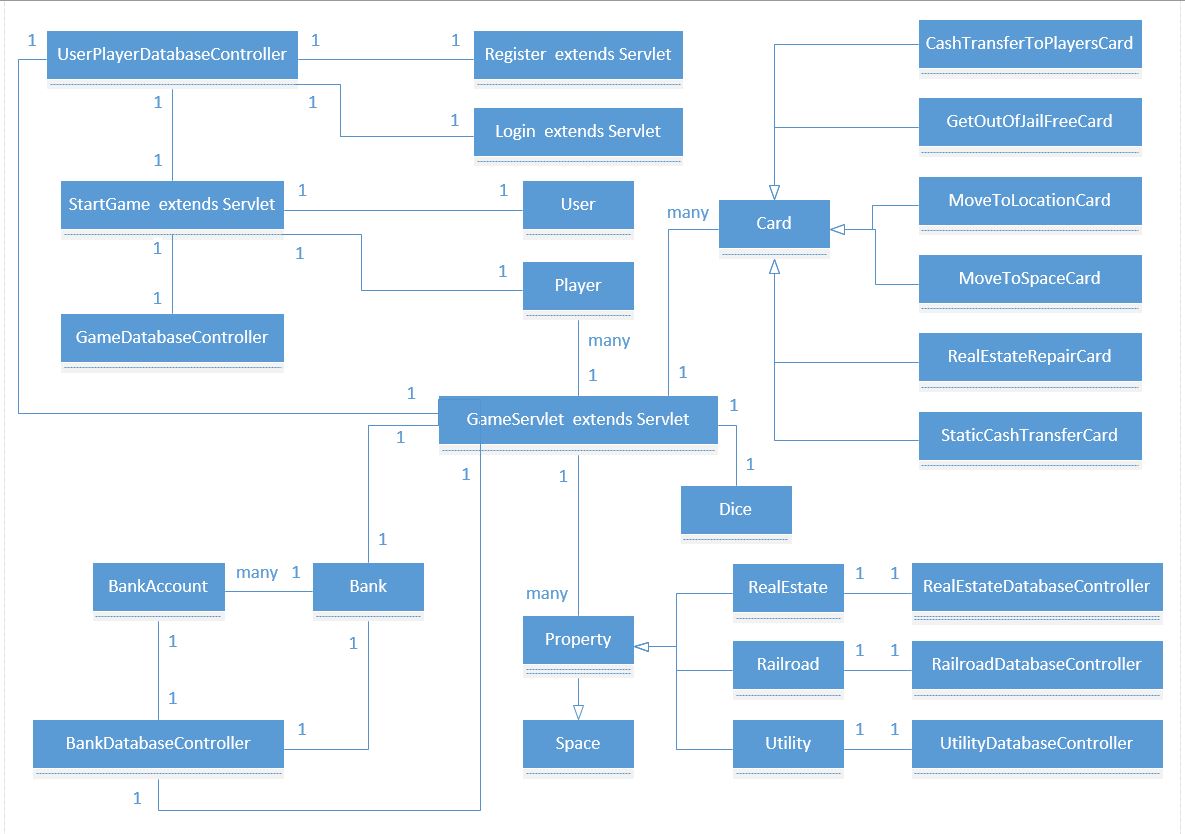
|  |  |
| --- | --- |
| **Use Case #** | 33 |
| **Use Case Name** | Get out of Jail |
| **Summary** | Player attempts to get out of jail. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | * It is the Player’s turn. * Player is in the Jail space, due to a previous turn. |
| **Description** | Player has 4 options to get out of Jail. The player can only try one of these in each of the next 3 turns after landing in Jail.   1. Player pays $50 on first or second turn in Jail. 2. Player succeeds in rolling doubles in any of the next 3 turns, after landing in Jail.    1. Player then moves to just visiting, and does not repeats his/her turn. 3. Player uses Get Out of Jail Free card.    1. Player may purchase the card from another Player if needed.    2. Player then moves to just visiting, and does not repeat his/her turn. 4. Player fails to roll doubles for a third turn, so they automatically pay the $50 fine to get out of jail. |
| **Alternative** | * If the Player does not have sufficient funds in their Bank account to pay the tax, then the appropriate Use Cases for selling, mortgages, and bankruptcy may be applied. |
| **Postcondition** | Player is out of Jail, and their turn is ended. |

|  |  |
| --- | --- |
| **Use Case** | 34 |
| **Use Case Name** | Leave game |
| **Summary** | Player leaves the current game. |
| **Dependency** | none |
| **Actor** | Player |
| **Precondition** | The player is currently logged in and participating in a game. |
| **Description** | * A player is participating in a game, and decides they need to leave both the game and the website. * The player chooses the option to logout. * All the player’s game assets are forfeited, and given over to the bank, as if the player has gone bankrupt. * The player is then taken to the homepage. |
| **Alternative** | * A set of players is participating in a game, and the players of the game decides to pause the game and leave the website. * The players chooses the option to pause and logout. * The state of the game is saved, so that it can be resumed later. * The user is then taken to the homepage. |
| **Postcondition** | The user is logged out of the website, and if needed their game assets have been repossessed by the bank, or the state of the game has been saved to be resumed latter(make this low priority). |

|  |  |
| --- | --- |
| **Use Case #** | 35 |
| **Use Case Name** | Join a game in progress. (Low Priority Item) |
| **Summary** | A User joins a match (game already in progress), and becomes a new Player. |
| **Dependency** | none |
| **Actor** | User, Player |

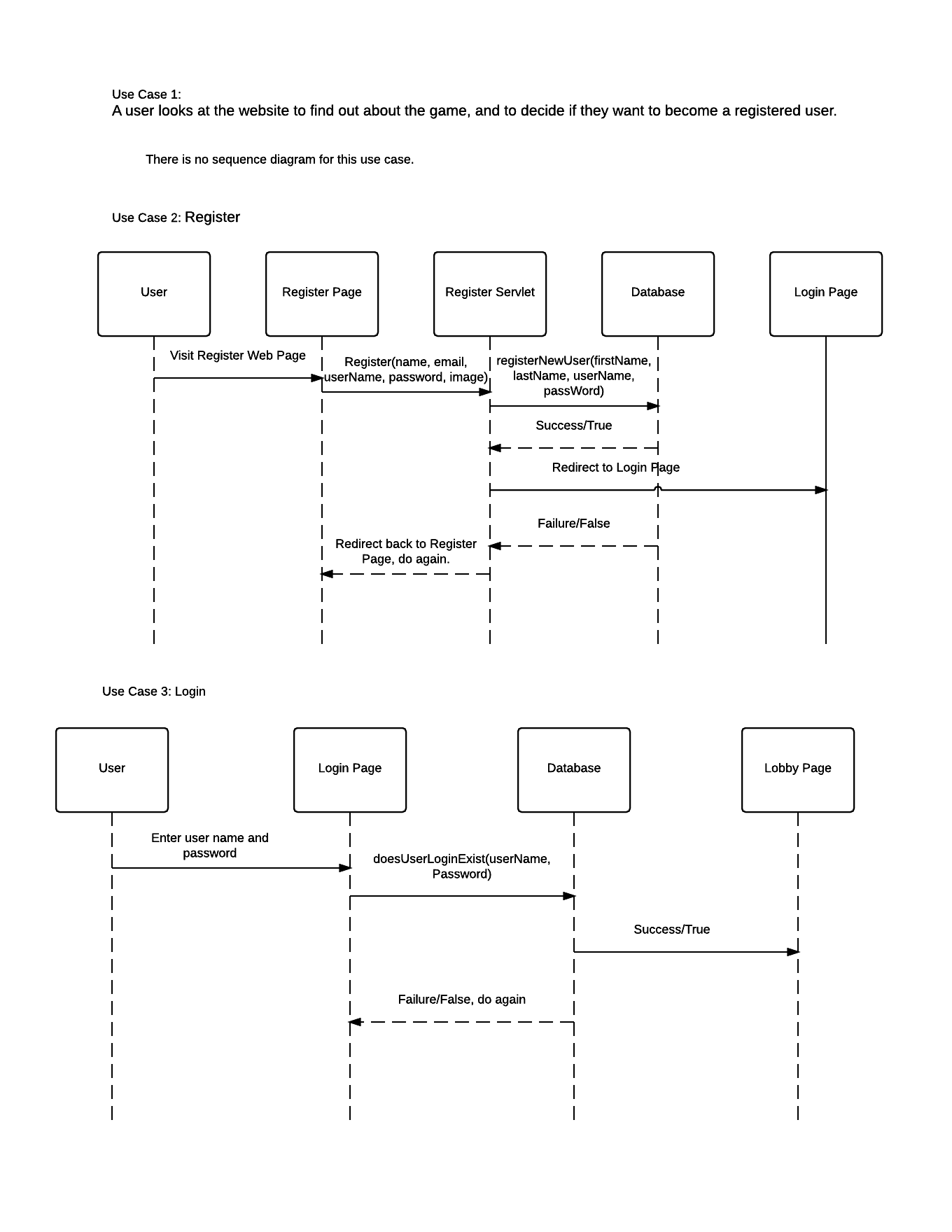
|  |  |
| --- | --- |
| **Precondition** | User must be logged in. |
| **Description** | * User choses a match from the lobby to join, that has already started previously. * If the Players already in the game agree to let the User join,   + The User becomes a Player, and join the match.   + The database is updated to associate that User with the Player, and the Player with the new Game. |
| **Alternative** | * Users are not allowed to join a Game in progress. |
| **Postcondition** | * The User is now associated with a Player and Game in the database. |

## **Conceptual Model Diagram (revision 4)**

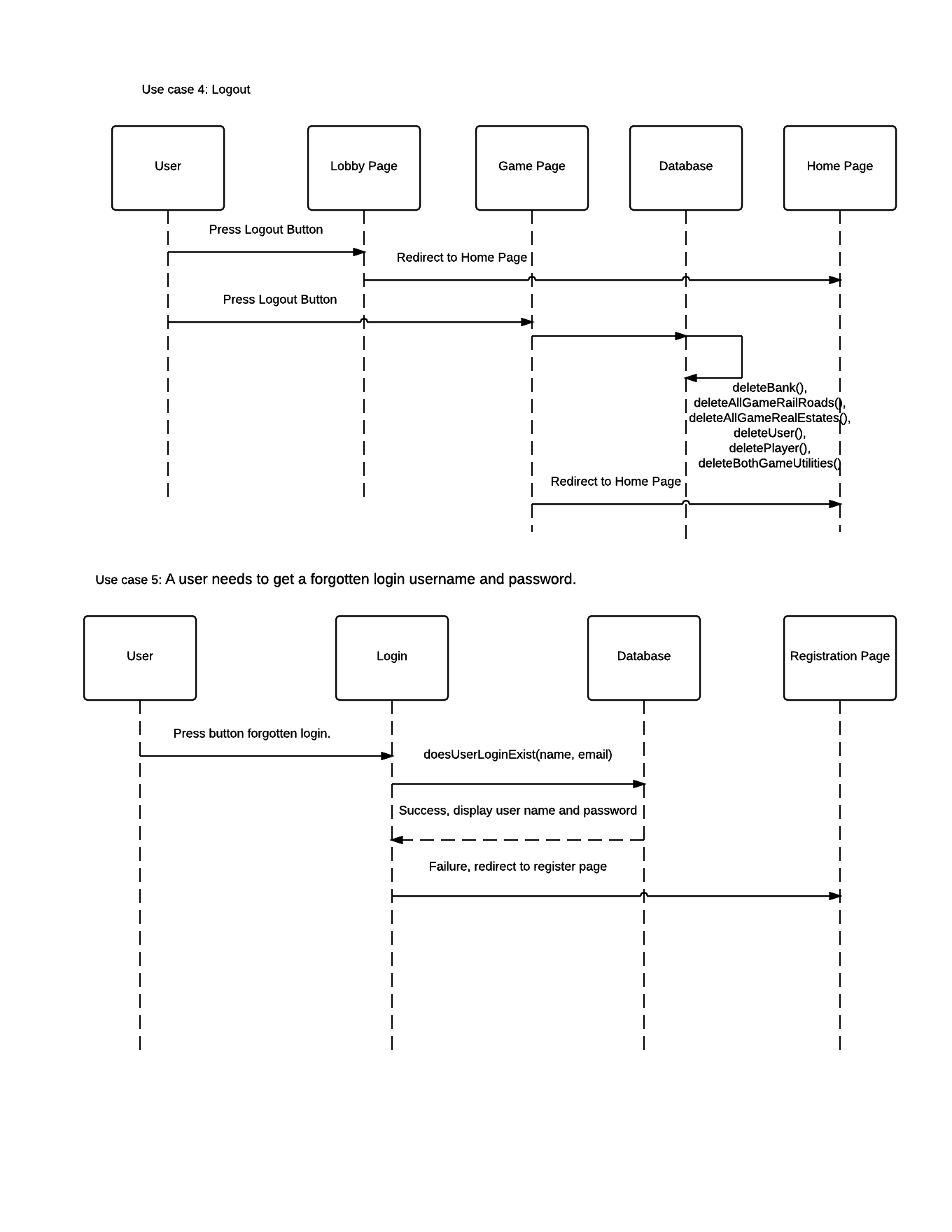


## **Sequence Diagrams**

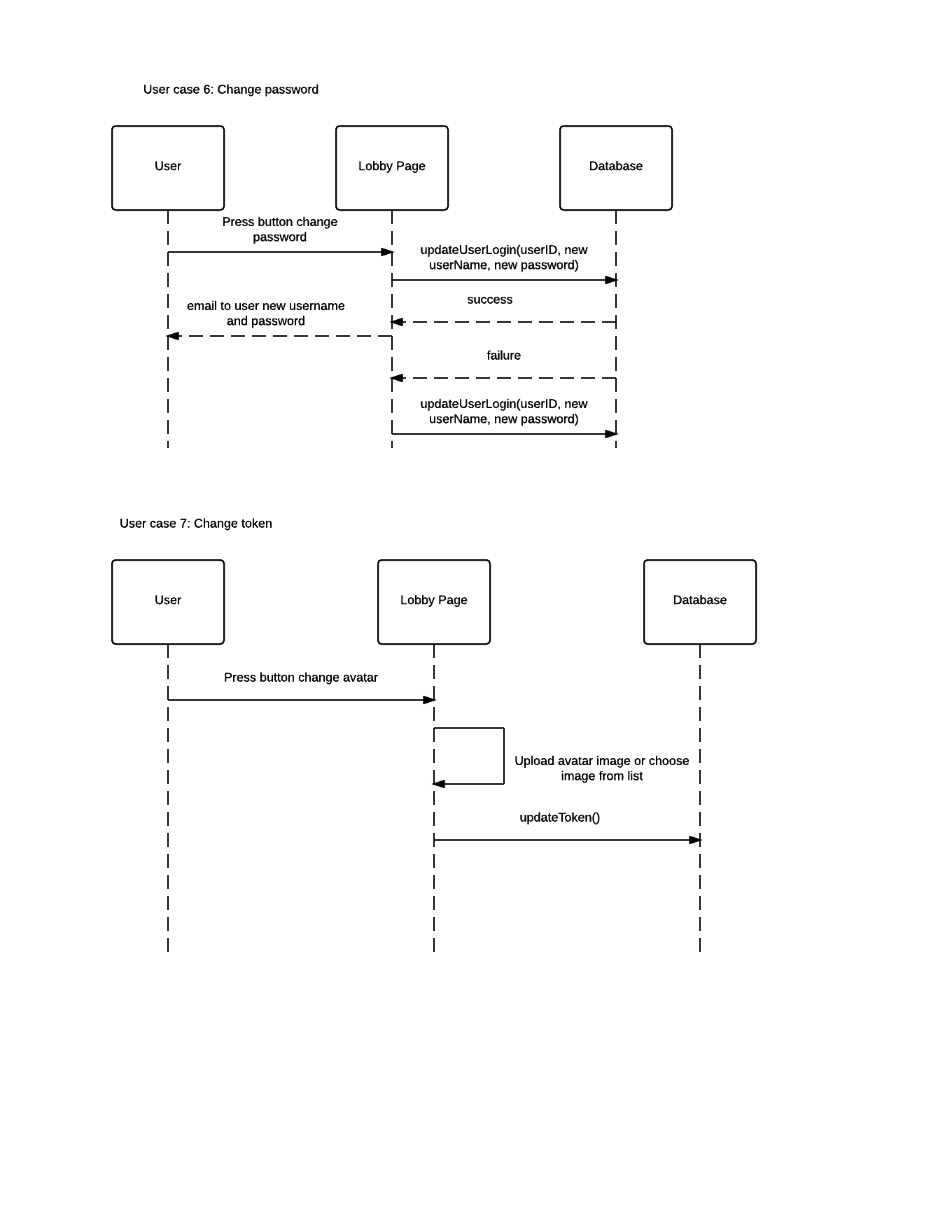
### Use Cases 1, 2, 3



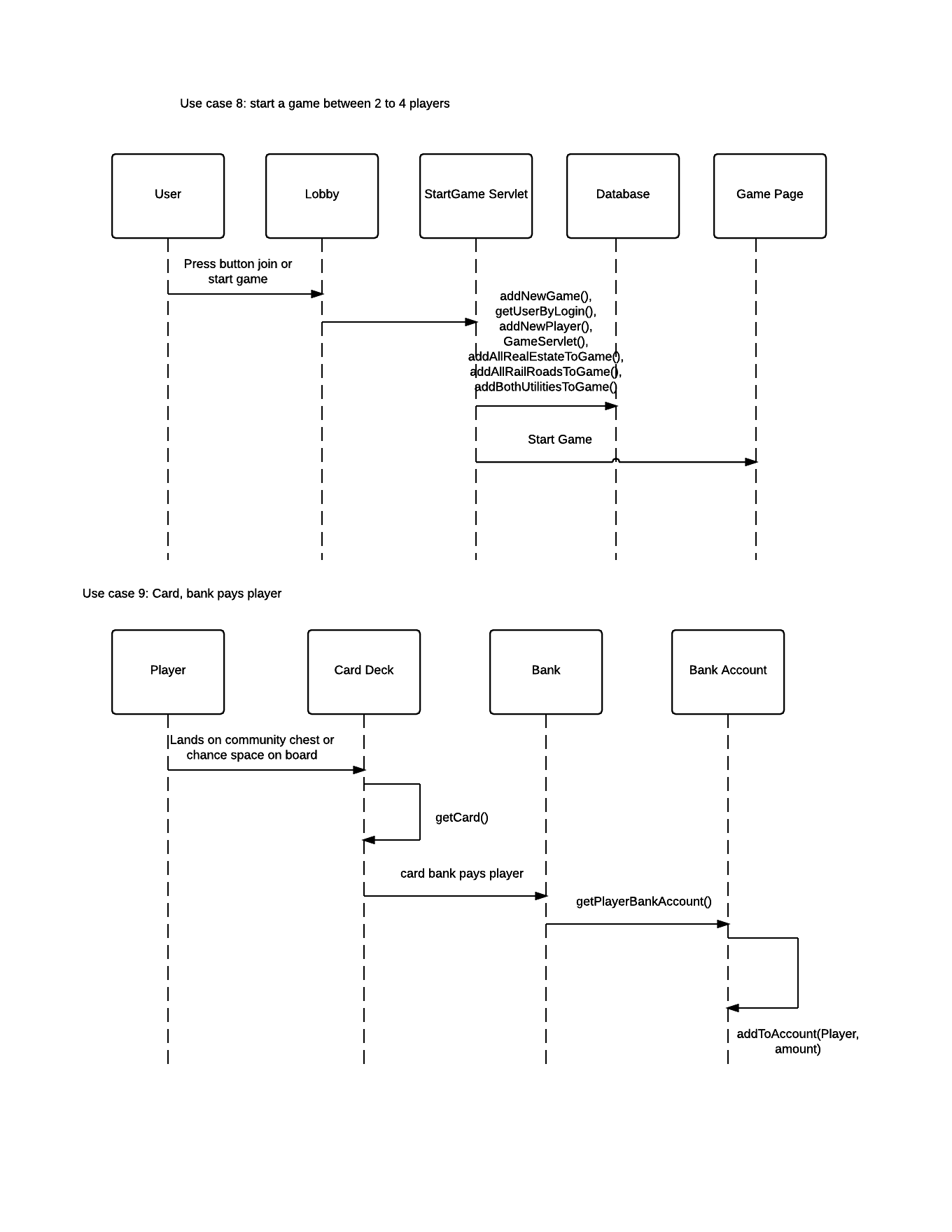
### Use Cases 4, 5



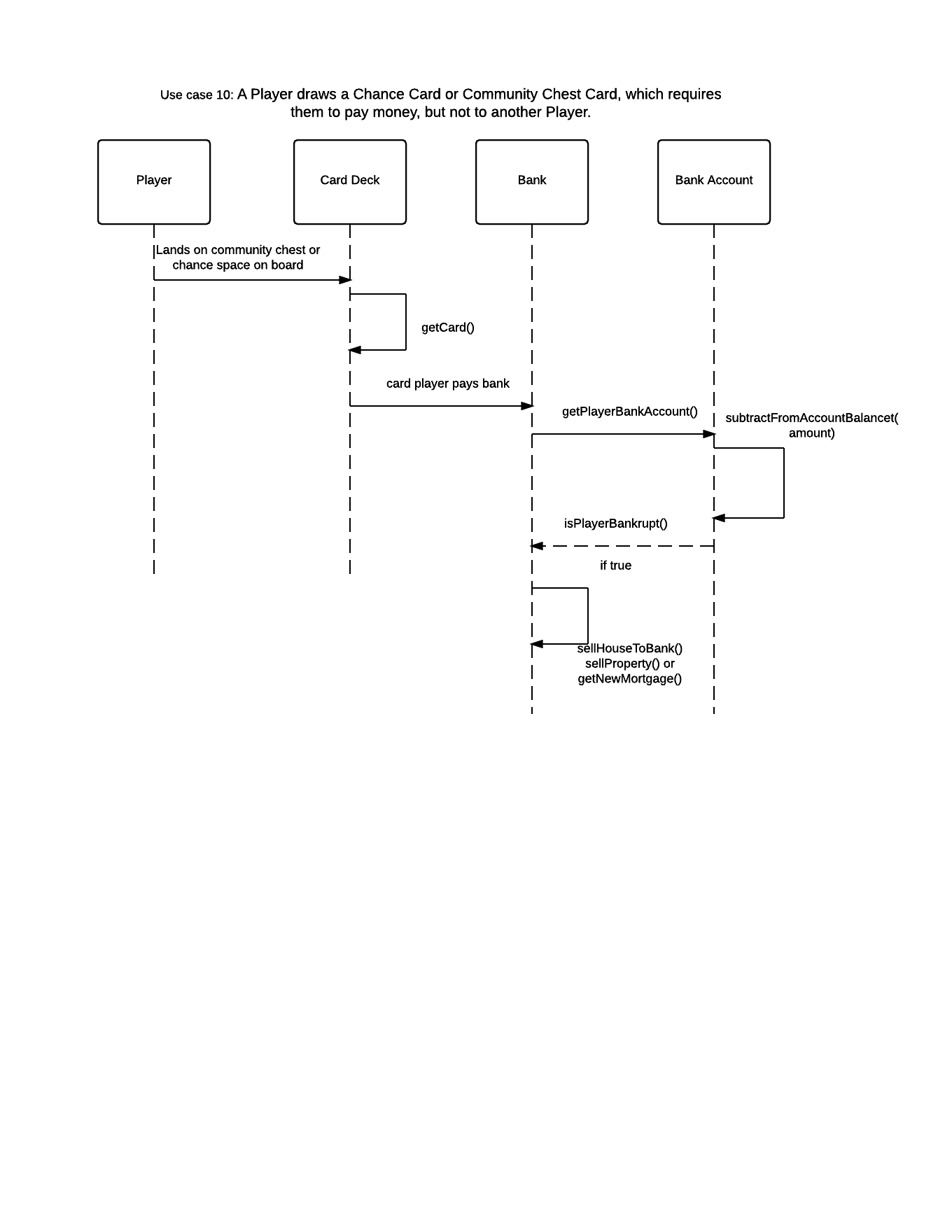
### Use Cases 6, 7



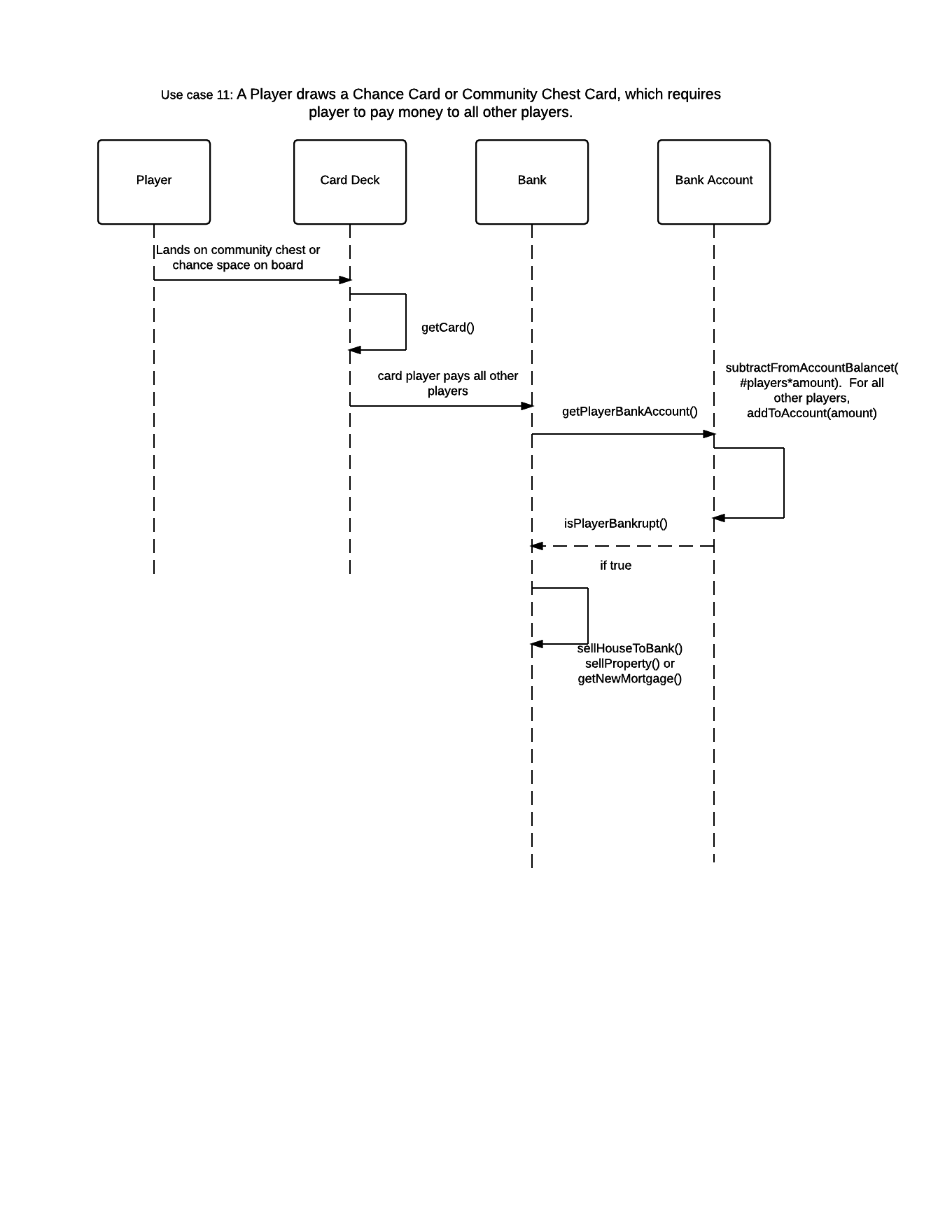
### Use Cases 8, 9



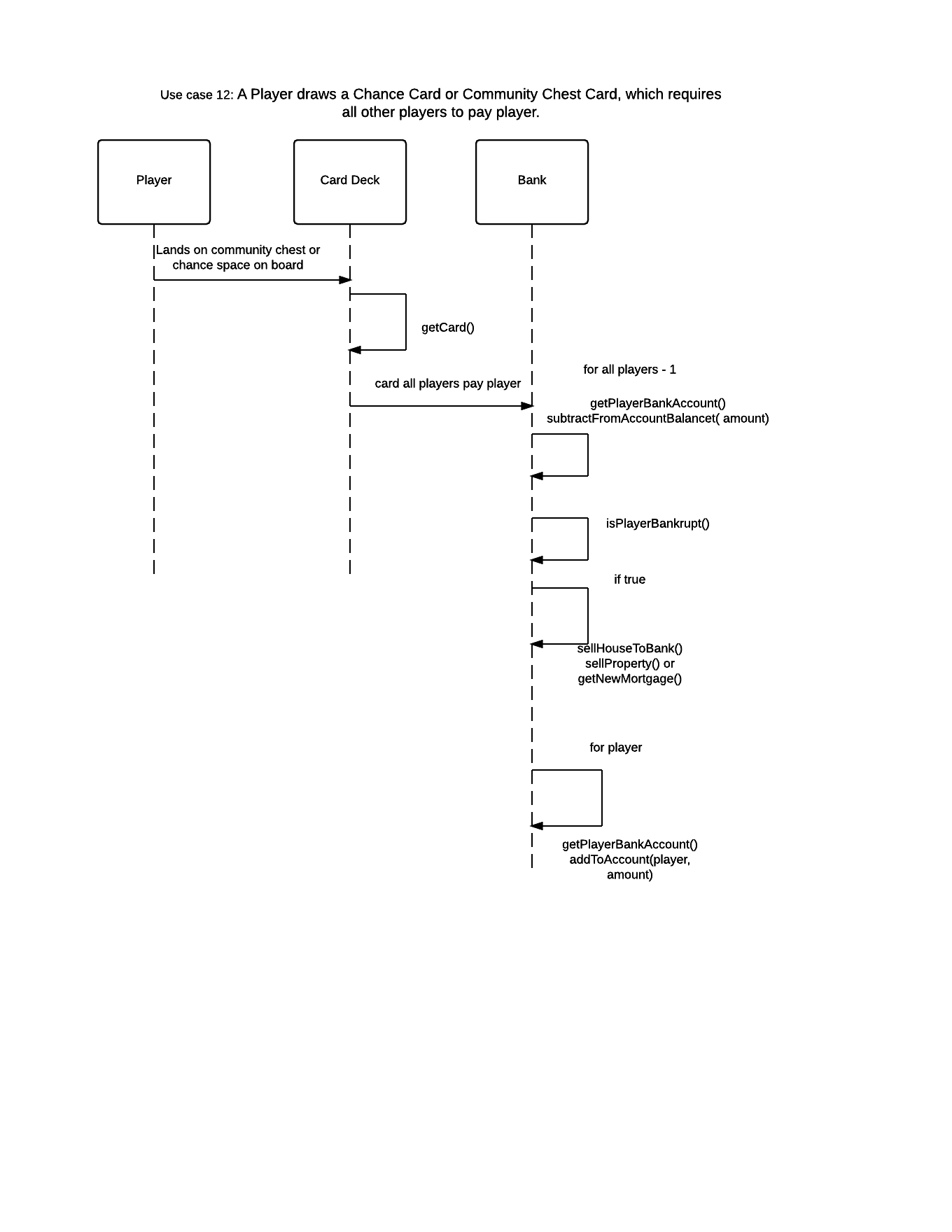
### Use Case 10



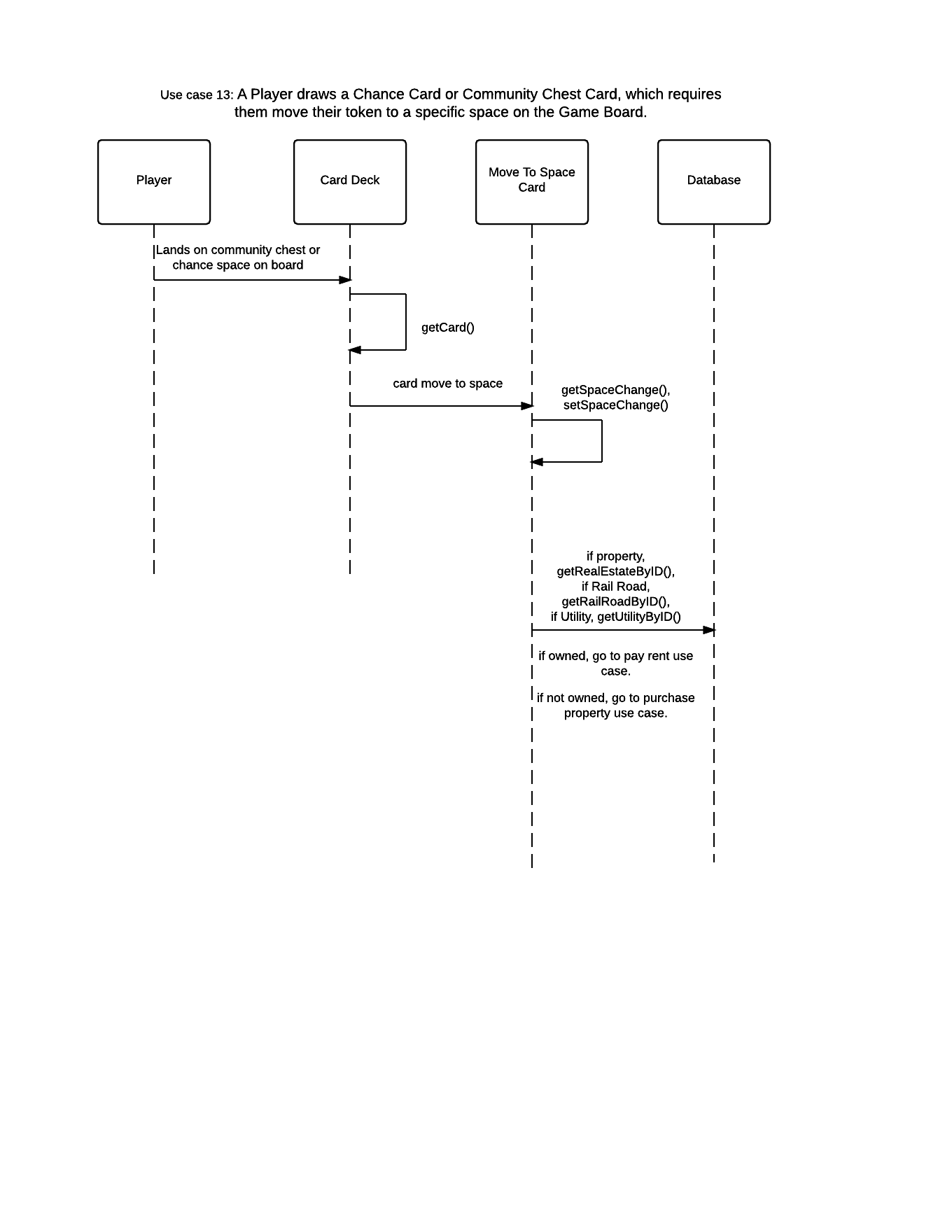
### Use Case 11



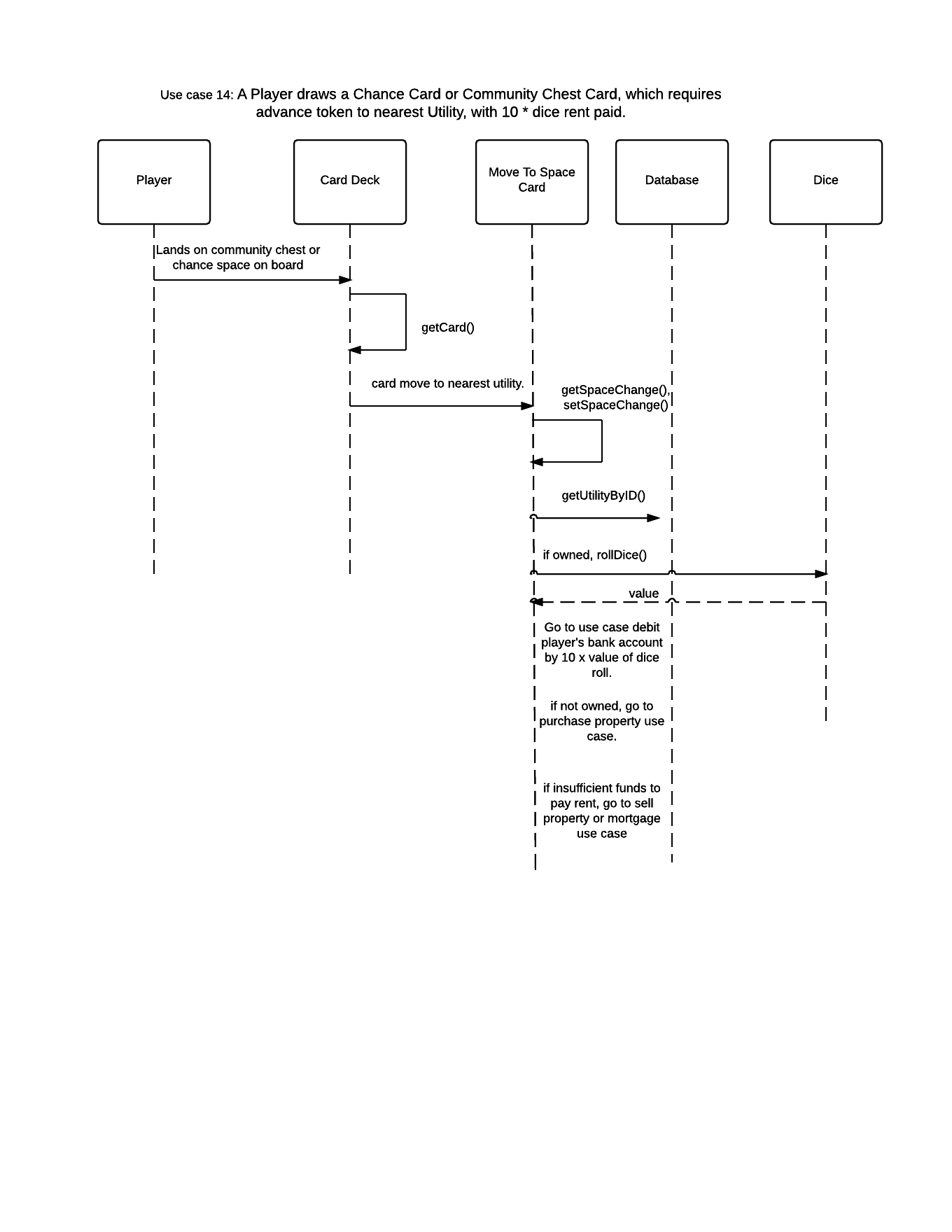
### Use Case 12



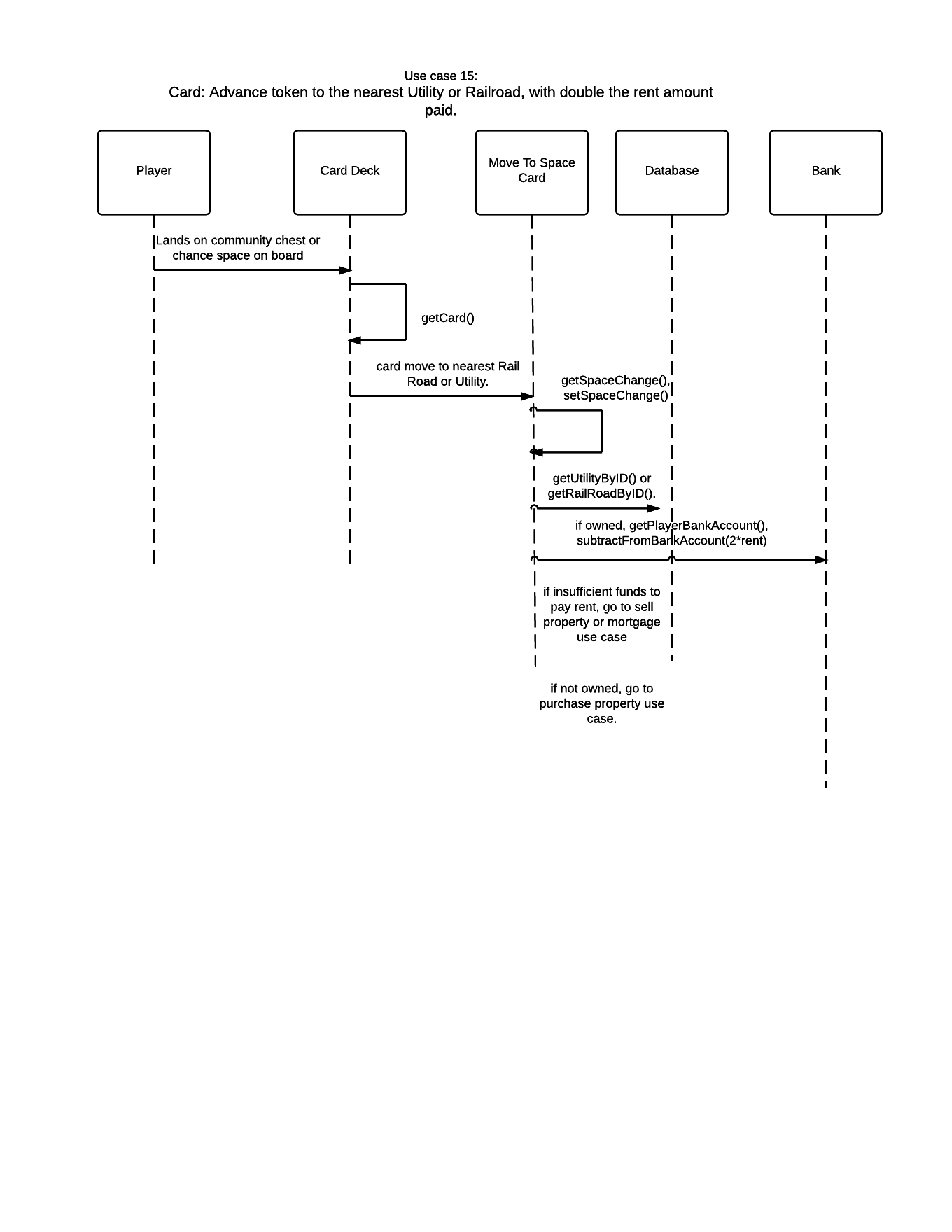
### Use Case 13



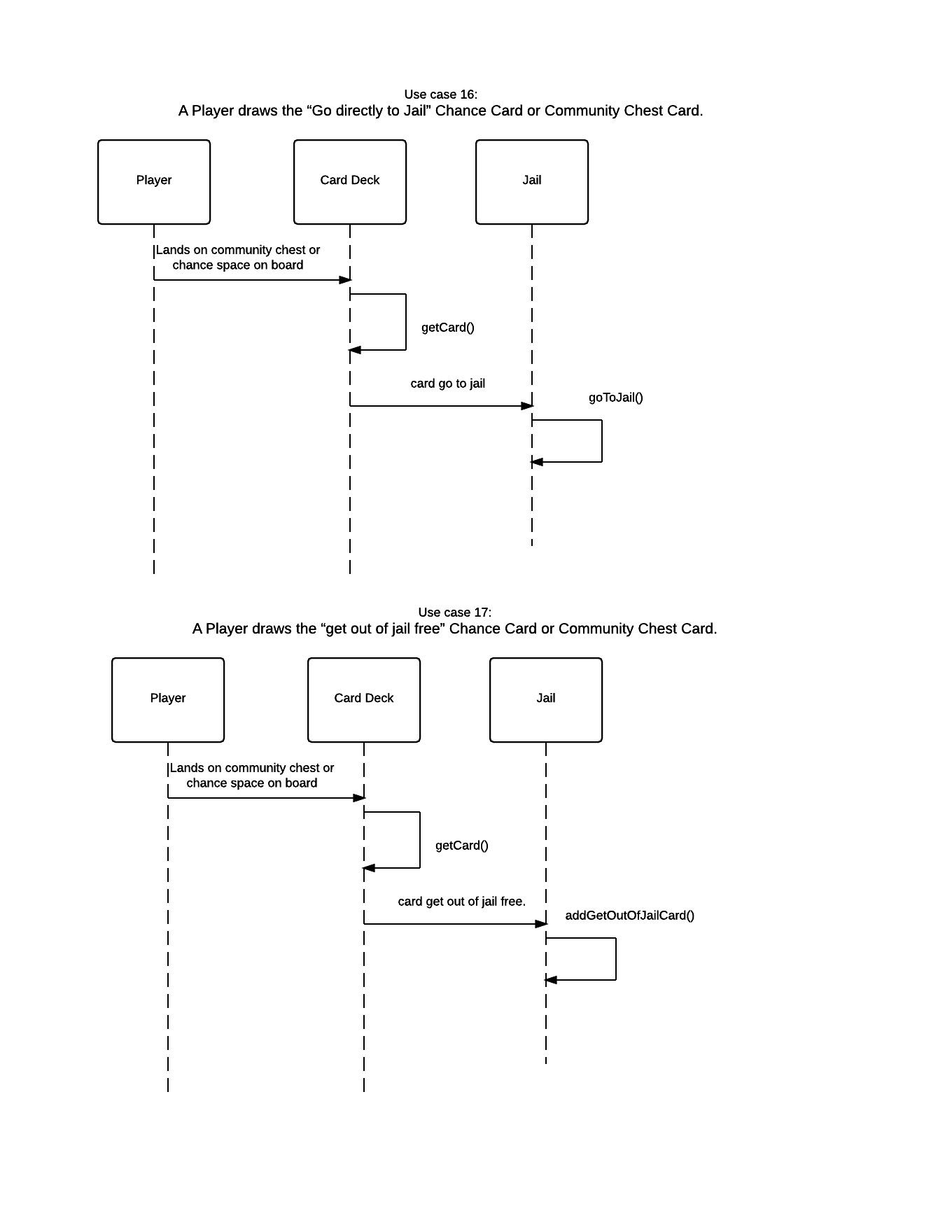
### Use Case 14



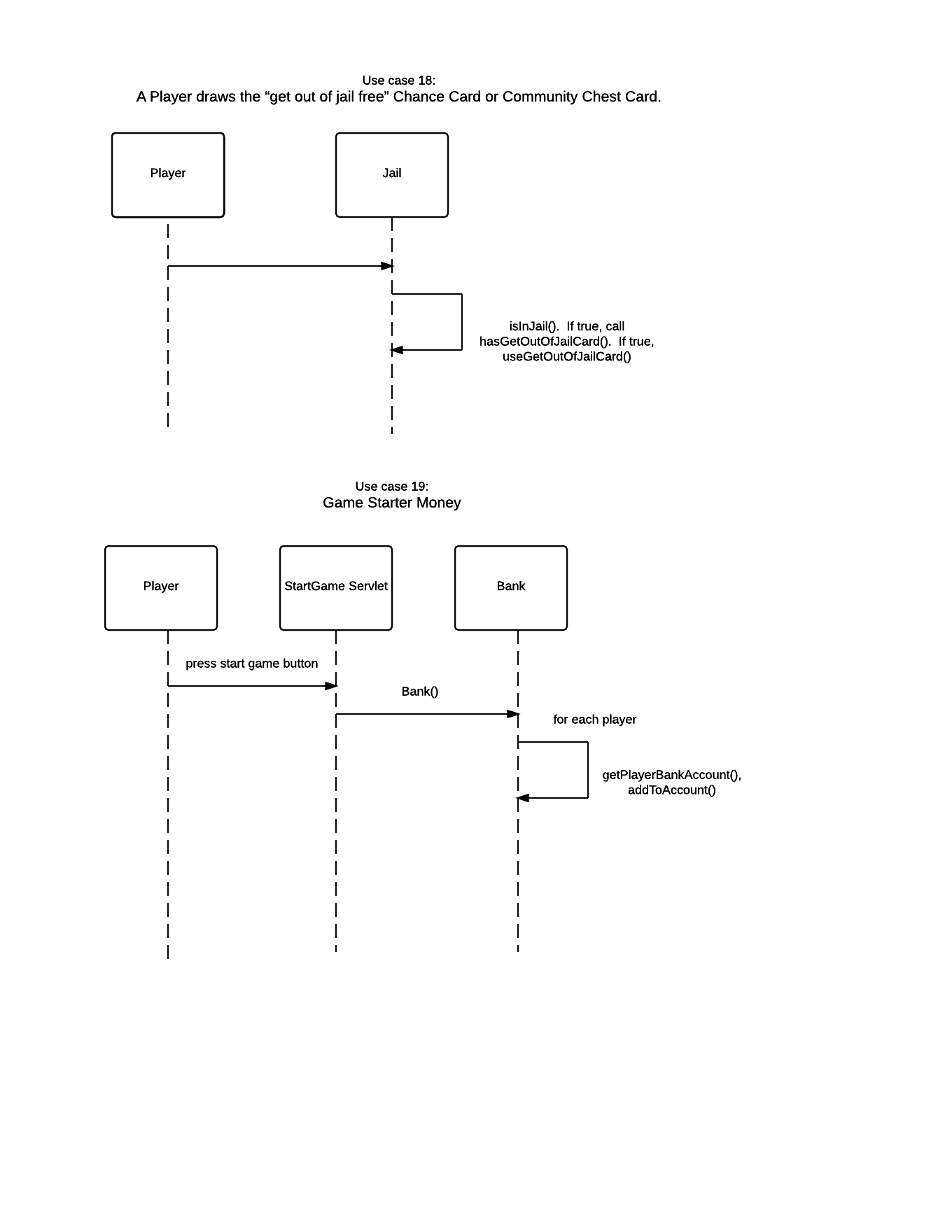
### Use Case 15



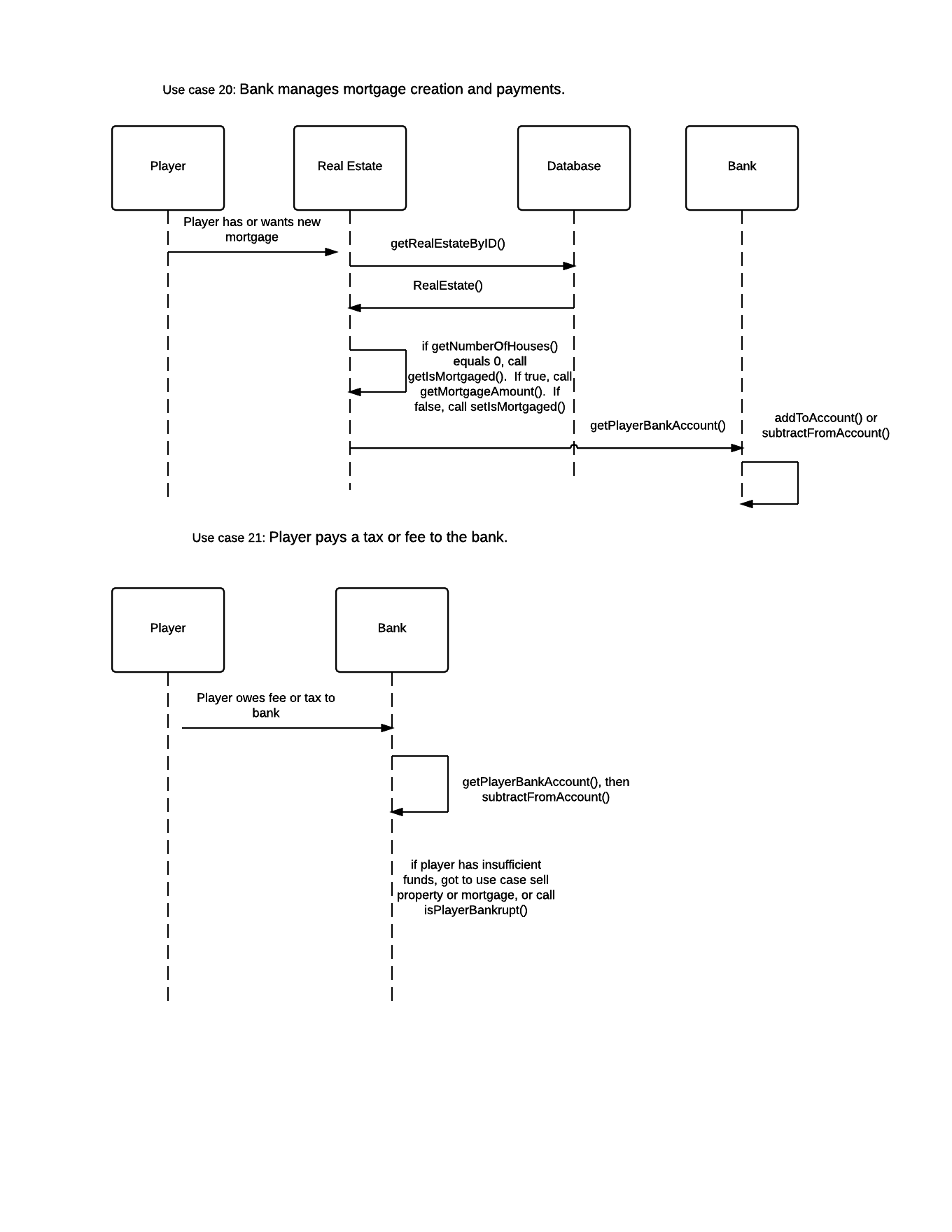
### Use Cases 16, 17



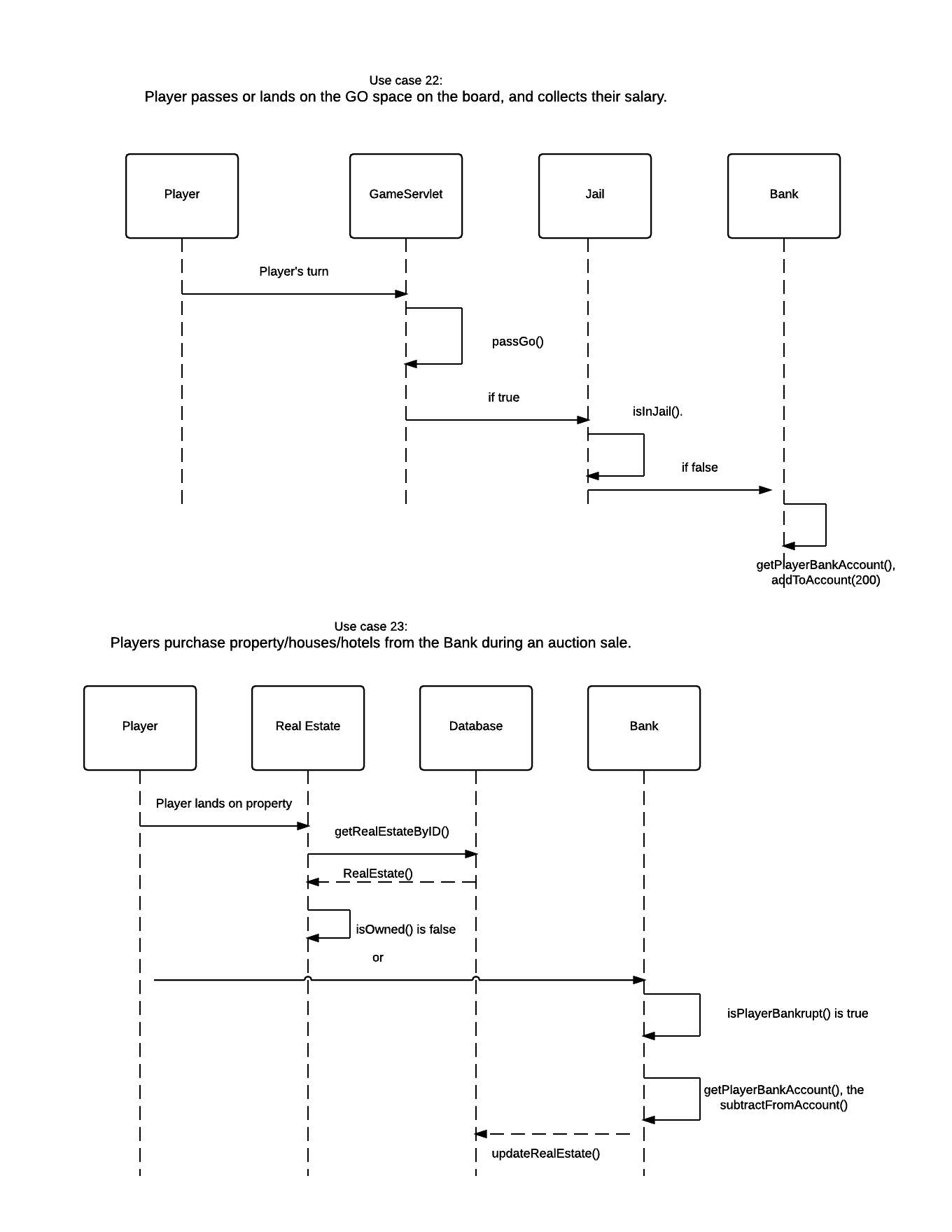
### Use Cases 18, 19



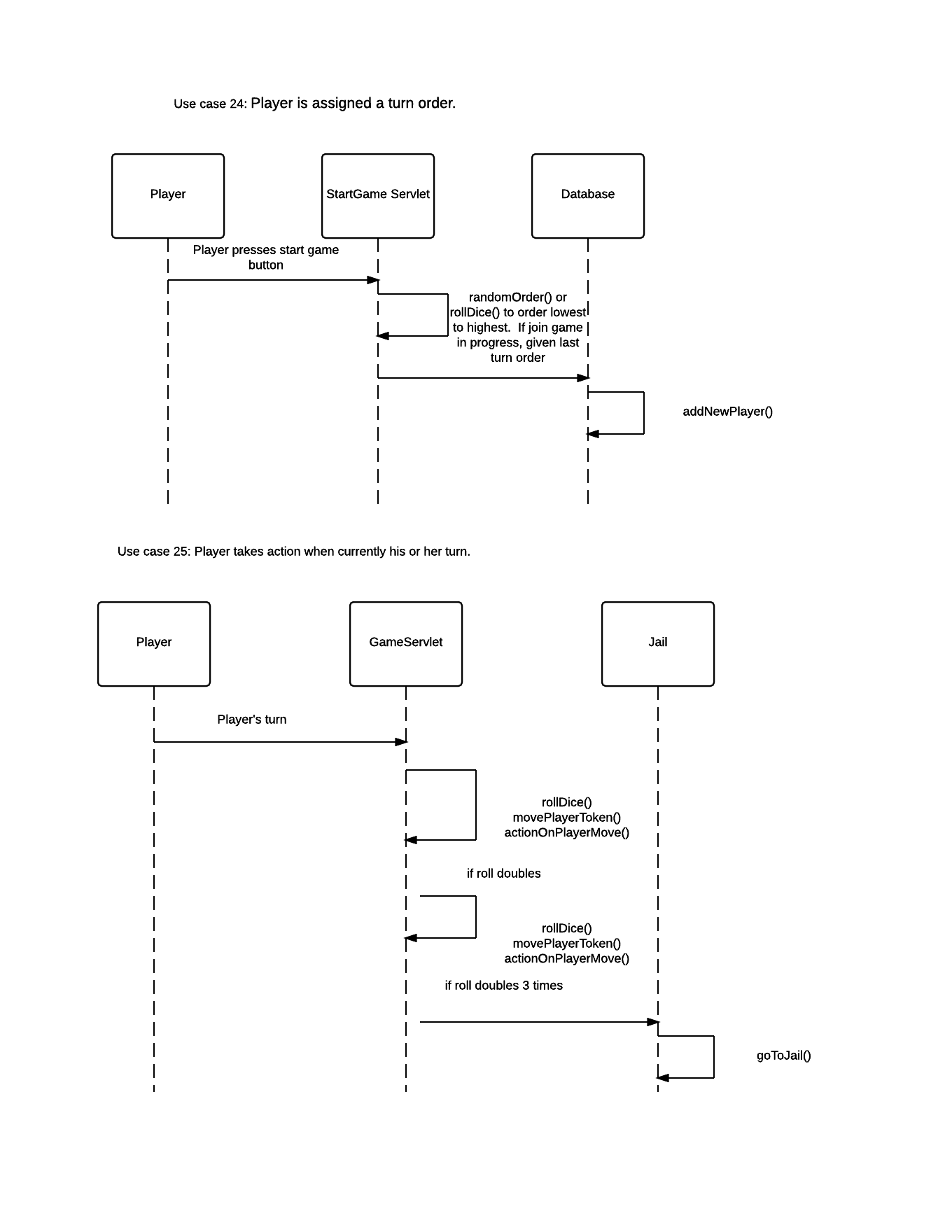
### Use Cases 20, 21



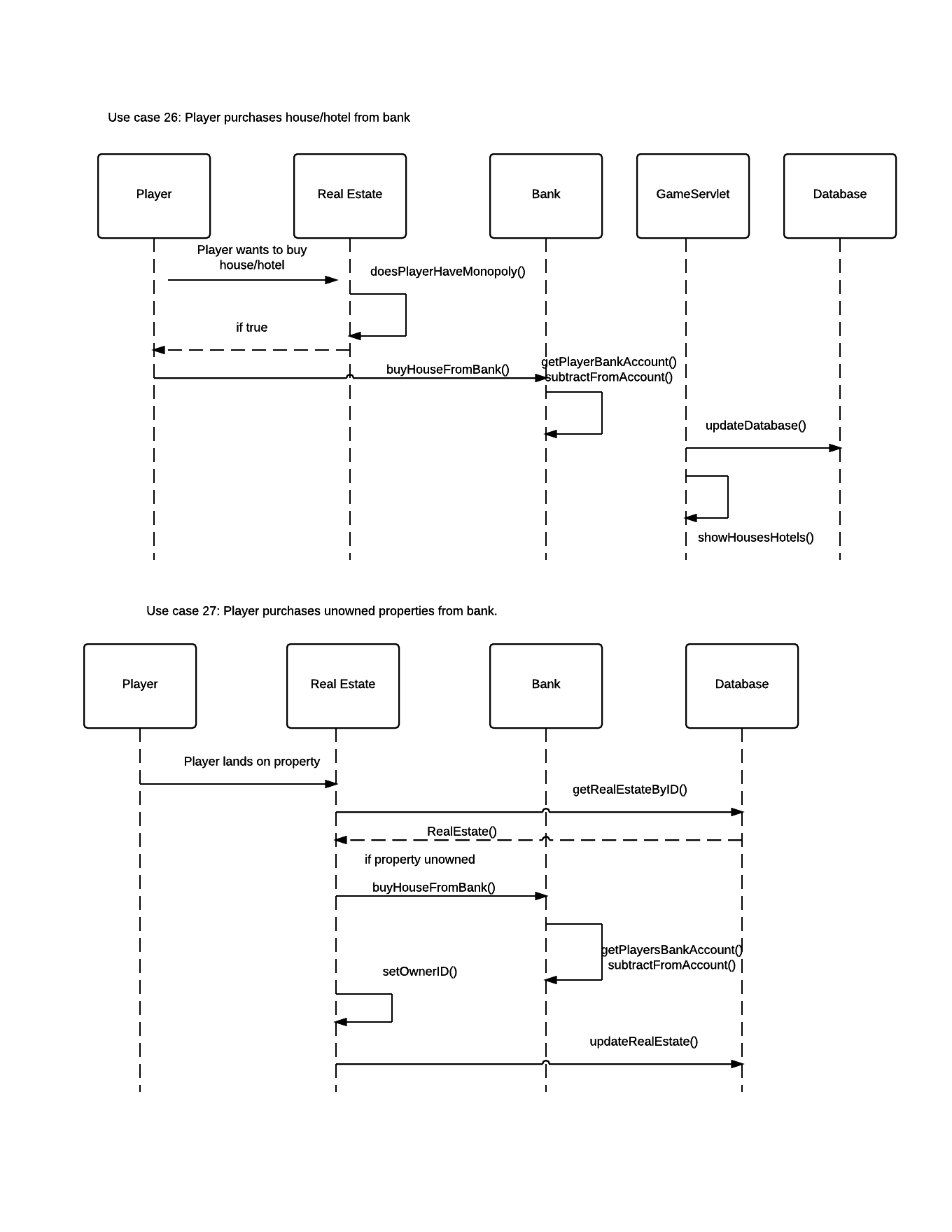
### Use Cases 22, 23



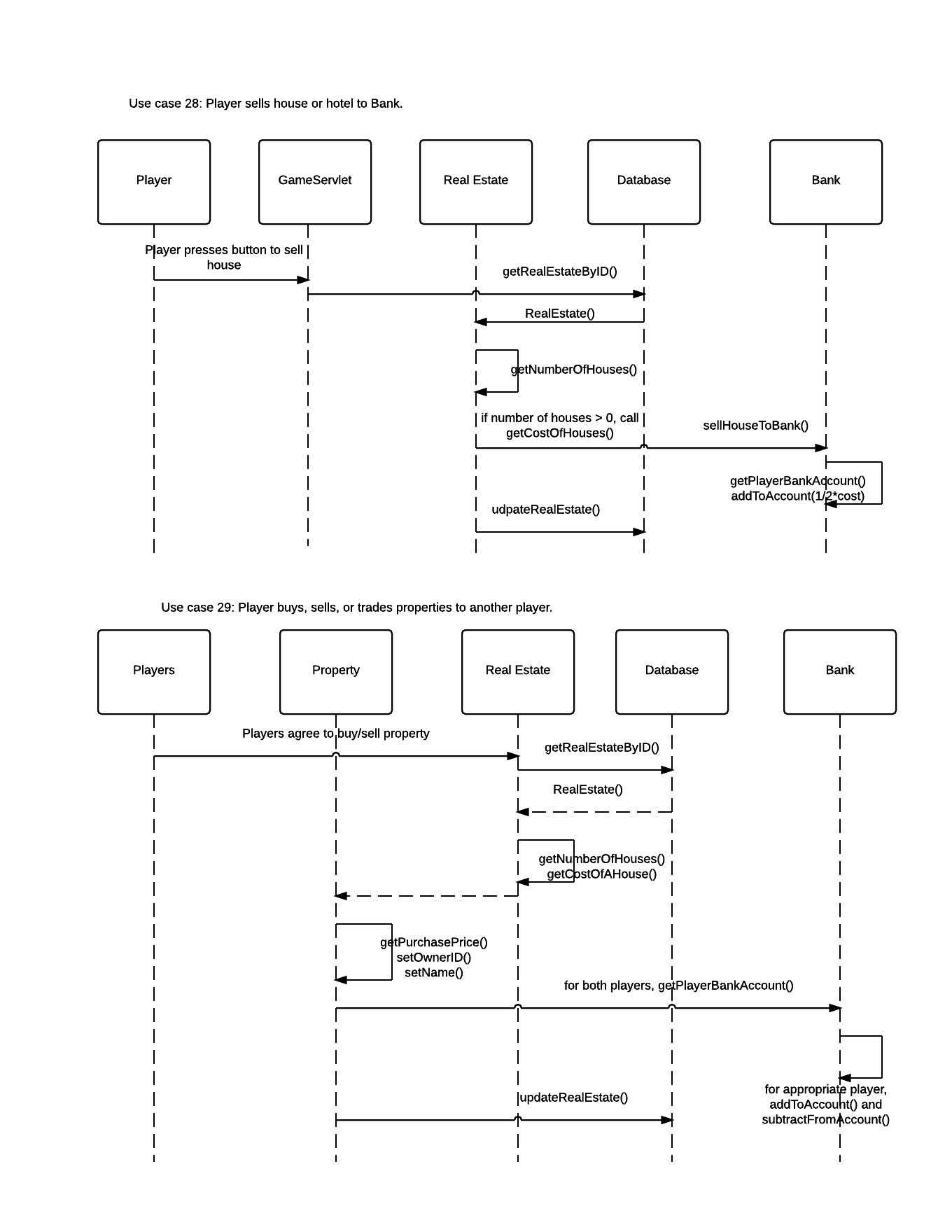
### Use Cases 24, 25



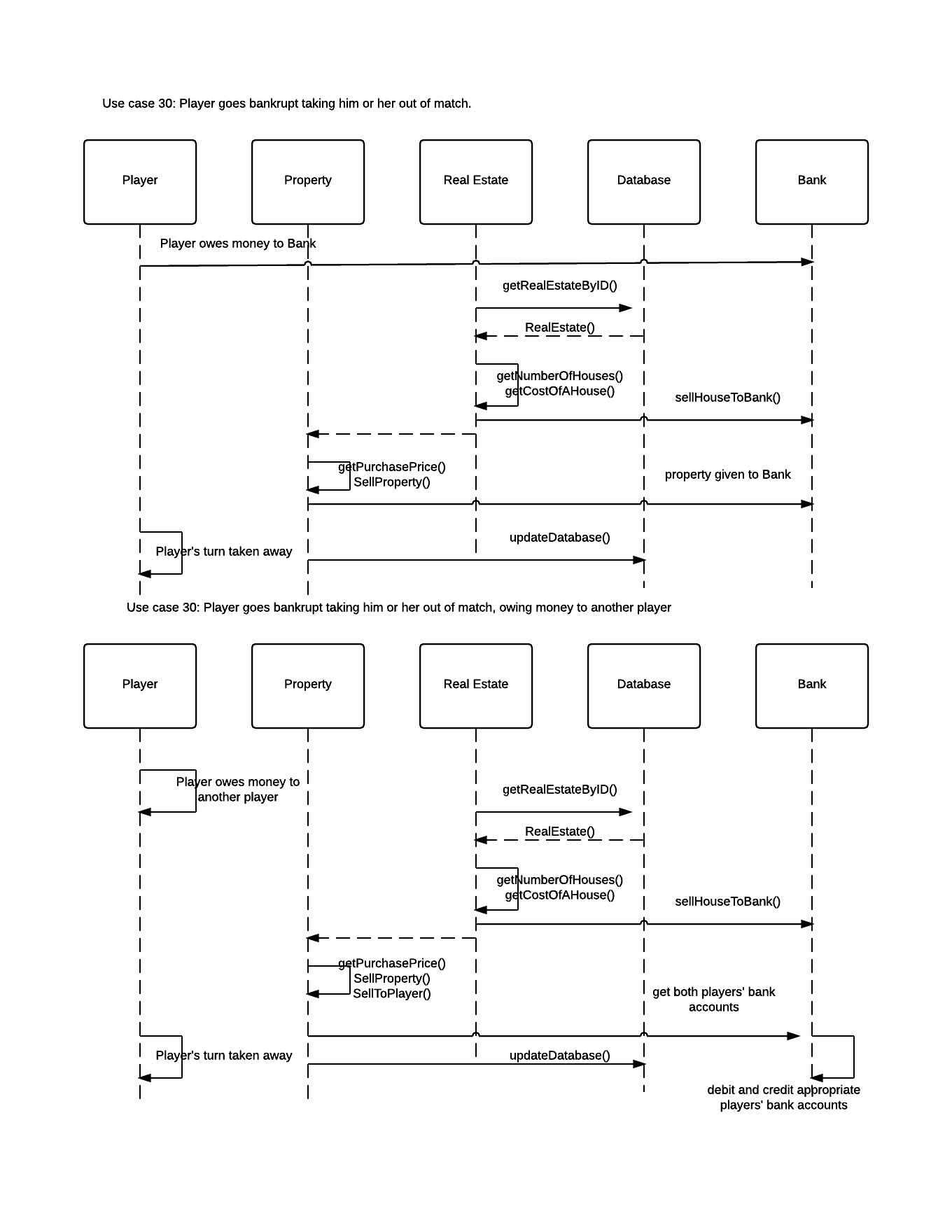
### Use Cases 26, 27



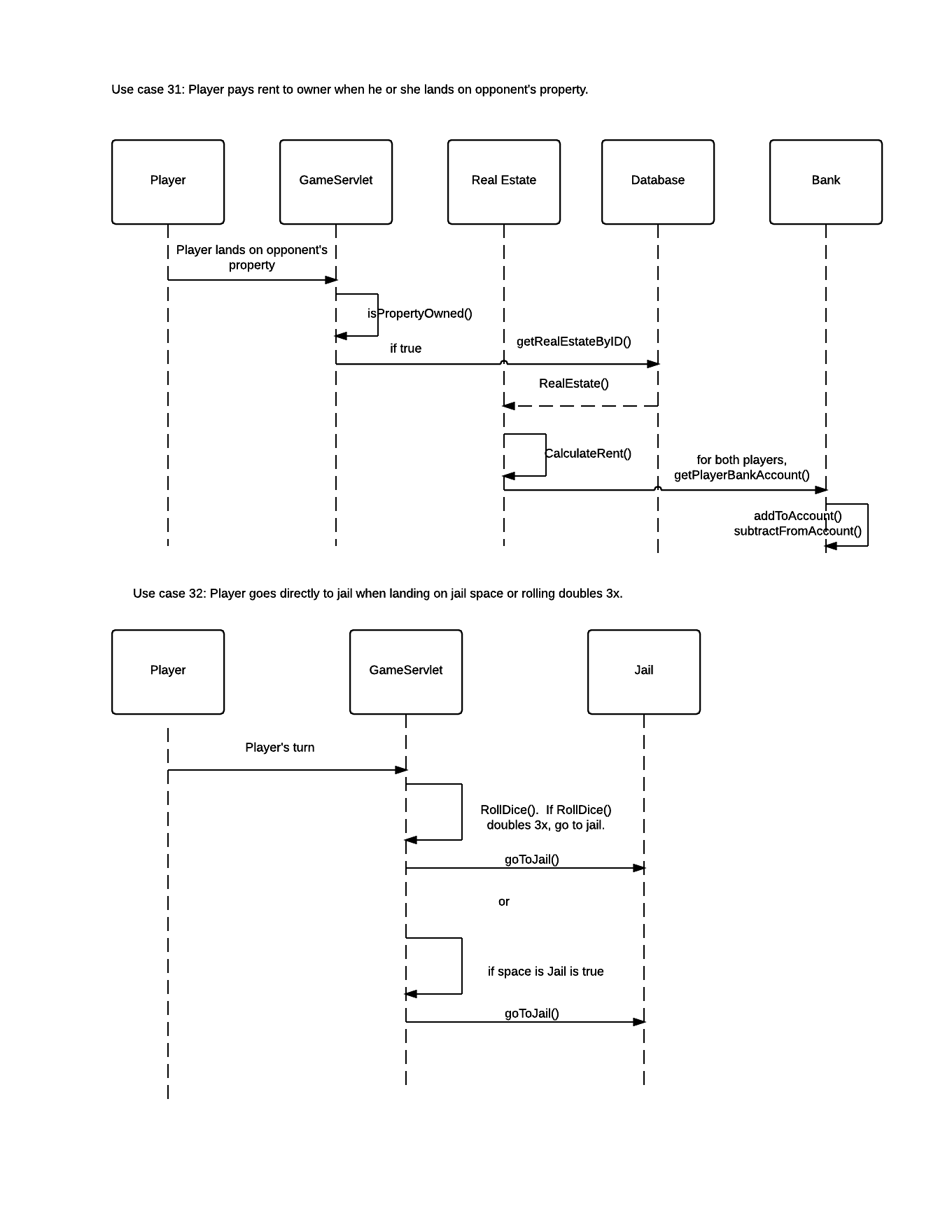
### Use Cases 28, 29



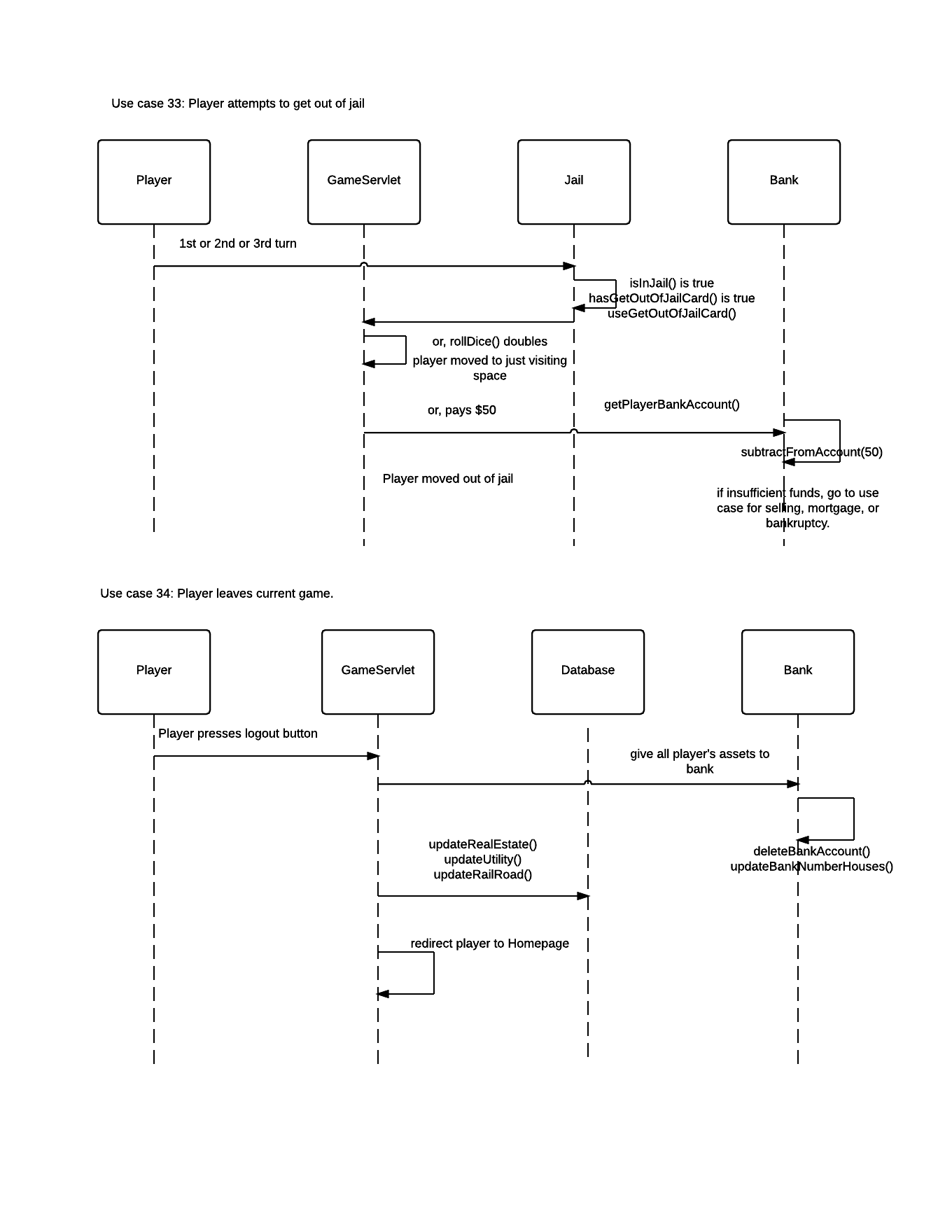
### Use Case 30



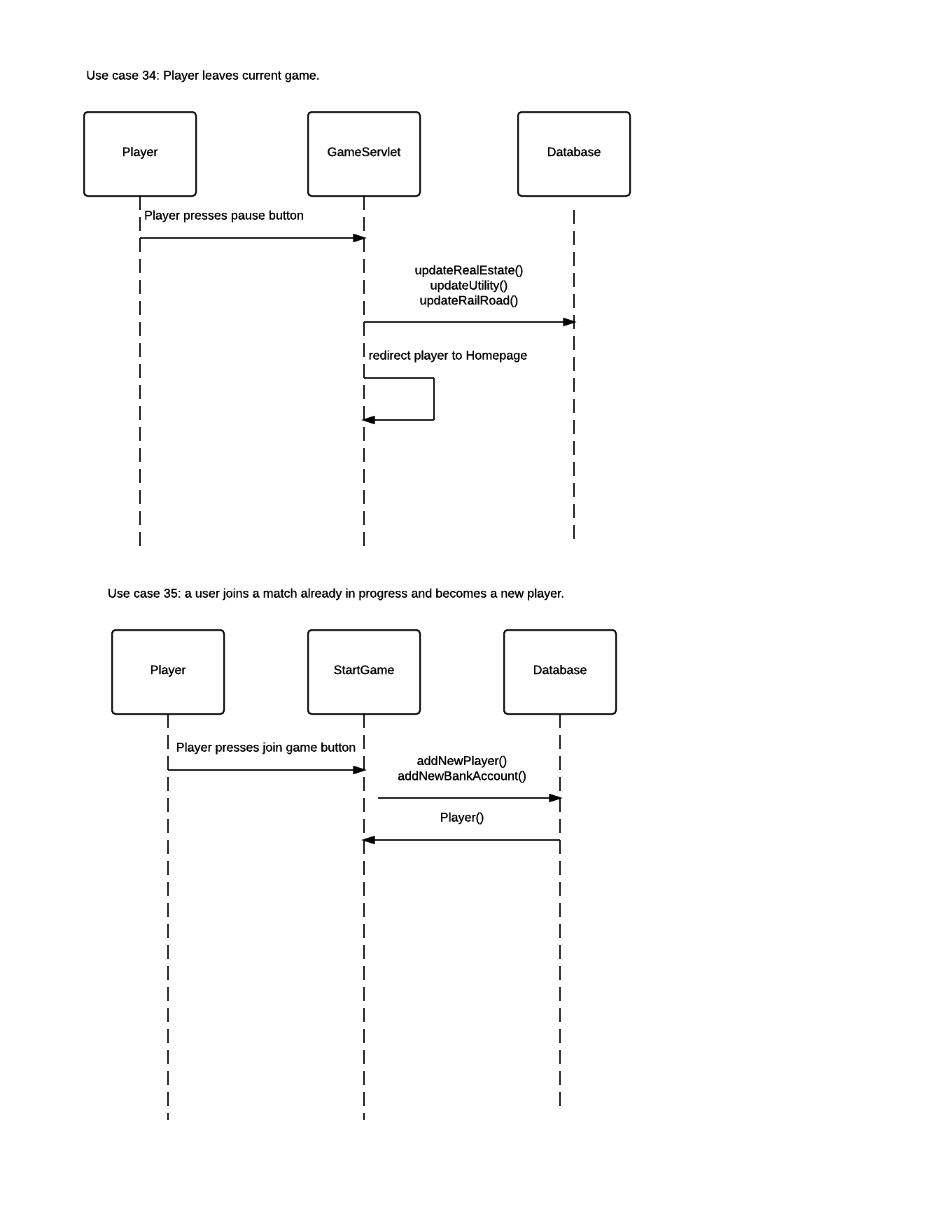
### Use Cases 31, 32



### Use Cases 33, 34a



### Use Cases 34b, 35



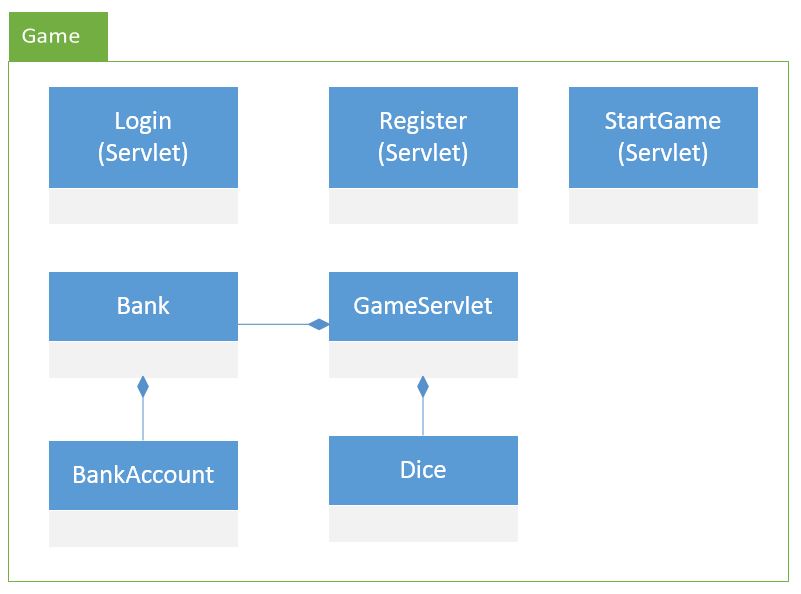
# Milestone Two

## **Packages (revision 2)**

Five packages will be used to organize the Java classes for the Monopoly game. An additional package is used for the web pages, CSS, and JavaScript/jQuery that form the GUI that the user interacts with.

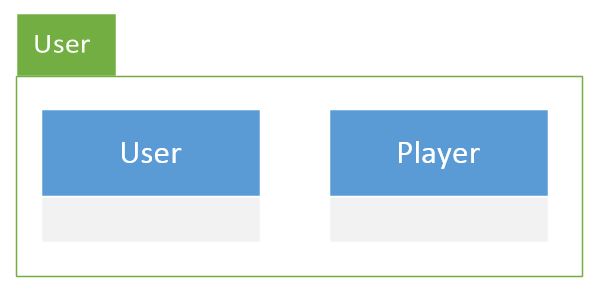
### Game Package

These are the classes that form the main game and website functions. They control the login, registration, starting a game, and the game model functions specific to playing a game of Monopoly.



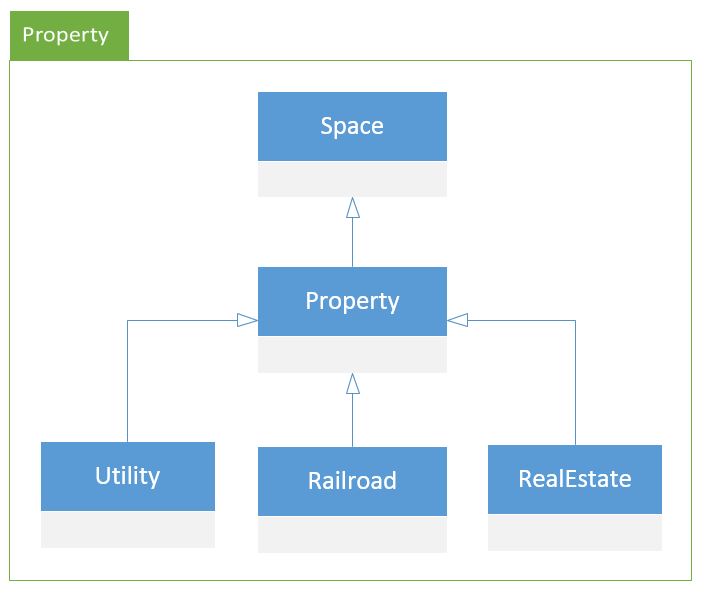
### User Package

These classes are for the User and Player information. This enables collecting registration and login information to object that are then stored in session to enable user specific features of the website. The Player is the representative of a User within a particular game. The Player is for the purposes of tracking game information specific to playing a game such as game board location and game assets.



### Property Package

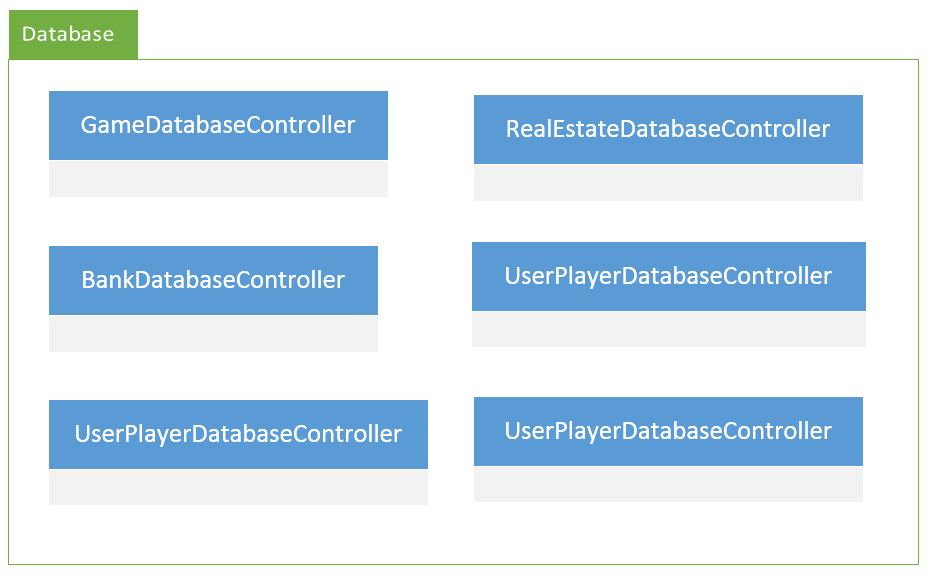
Theses classes represent the properties that Players can buy, sell, trade, and pay rent for when they land on their spaces during the game.



### Database Package

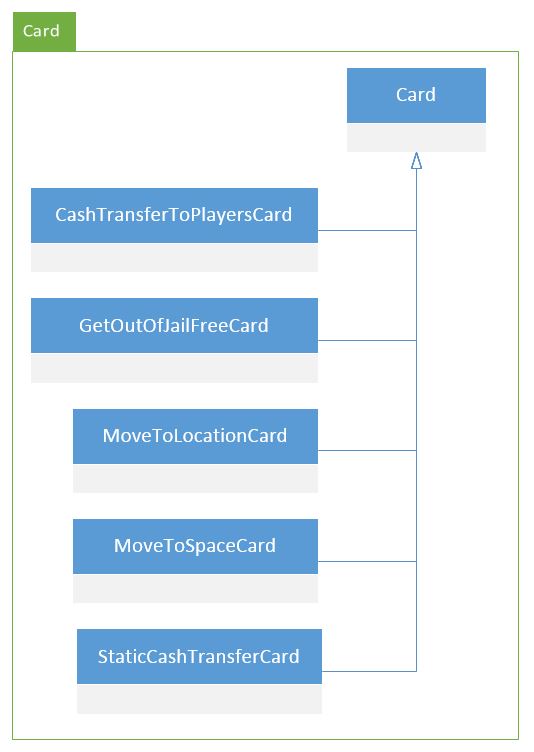
The Database Controller classes are a set of Facades for database access. Except for the GameDatabaseController, they are used to instantiate the new objects needed by a new Monopoly game based on the contents of the database. To prevent database corruption they also use the Singleton design pattern. The use of the Singleton pattern is why they do not all inherit from a common database class.

These 6 classes encapsulates all of the database interactions from the rest of the application so that no other classes will contain SQL or database connections. For brand new entries to the database the ids will be auto generated by the database. The new object being requested is returned by the Database Controller class when the new object is first created and saved to the database. Special functions specifically for the handling of login search and retrieval of users are also included for ease of use in the UserPlayerDatabaseController.



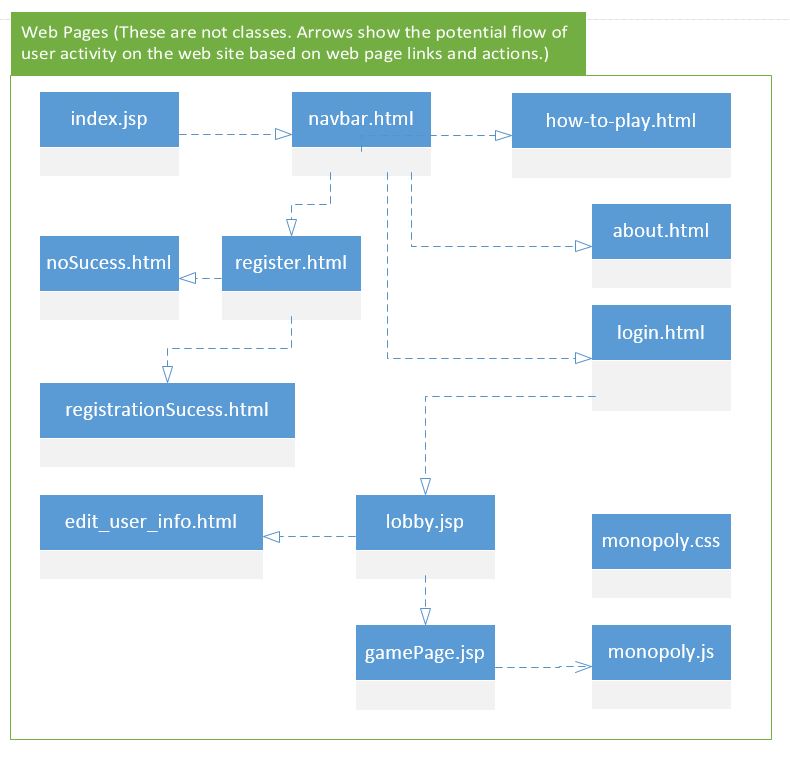
### Card Package

The Card classes represent actions that a Player must take when they draw a Card after landing on the Community Card space or the Chance Card space on the game board.



### Web Pages Package

These are the HTML and JSP files for the GUI. They control the look of the web site, and the human interaction with the application.



## **Class Descriptions and Diagrams (revision 3)**

### User Class

The User class allows different actions for a User. A User is someone who can browse the website, register, login, logout, get login username and password from the database, and change password.

|  |
| --- |
| **Class User** |
| -int userID  -int playerID  -String username  -String firstName  -String lastName  -String password |
| +User()  // initializers are used by the DatabaseController classes to instantiate objects from the database  +void initializeUser(int userID, String username, String password, String firstName, String lastName)  +void initializeUser(String username, String password, String firstName, String lastName)  // used for login and registration  +void setUsername(String username)  +String getUsername()  +String getPassword()  +void setPassword(String password)  +void setFirstName(String firstName)  +String getFirstName()  +public void setLastName(String lastName)  +String getLastName()  // the database gives the user a player id when the player is created at the start of a game  +void setPlayerId(int id)  +int getPlayerId()  // id is auto assigned by the database when the user is created the first time by registering  +void setUserId(int id)  +int getUserId() |

### Player Class

The Player class contains information about the player’s status in the game. Each player will have player\_id and token\_id. Each player will be assigned a turn order in the beginning of the game.

|  |
| --- |
| **Class Player** |
| -int userID  -int playerID  -int tokenID  -int spaceID  // This flag used when a player has gone bankrupt and cannot take a turn anymore, but chooses to // stay in the game as a spectator to watch how it ends.  -boolean spectator |
| +Player()  +void initializePlayer(int playerID, int userID, int tokenID, int spaceID)  +boolean getSpectator()  +void setSpectator(boolean spectator)  +int getTokenID()  +void setTokenID(int id)  +int getPlayerID()  +void setPlayerID(int id)  +int getUserID()  +void setUserID(int id)  +int getSpaceID()  +void setSpaceID(int id) |

### Bank Class

The bank owns all the properties, so whenever a player wants to buy or mortgage a property, he needs to call the bank class. The bank gives starter money to all the players in the beginning of the game. It stores all the bank accounts of all the players. It also owns the property cards that will be given to the player who wishes to buy a property.

|  |
| --- |
| **Class Bank** |
| -int bankId  -ArrayList <BankAccount> bankAccountList  -int numHouses  // the number of houses that a bank needs to have a the start of the game  -static final int starterNumberHouses  // for updating the database without a lot of SQL code in every class  -static BankDatabaseController database  +Bank()  +void initialize(int id, int numHouses, ArrayList<BankAccount> accountList)  +void getNewMortgage(Player player, Property property)  +void payOffMortgage(Player player, Property property)  +boolean isPlayerBankrupt(Player player, int amountOwed)  +BankAccount getPlayerBankAccount(Player player)  +ArrayList <Integer> getPropertyList ()  +ArrayList <Mortgage> getMortgageList()  +void sellHouseToBank(Player player) |
| +void buyHouseFromBank(Player player, int housePurchasePrice)  +void subtractFromAccount(Player player, int amount)  +void addToAccount(Player player, int amount)  +int getBankID()  +void setBankID(int bankID)  +int getNumHouses()  +void setNumHouses(int numHouses)  +static int getStarterNumberHouses()  +ArrayList<BankAccount> getBankAccountList()  +void setBankAccountList(ArrayList<BankAccount> bankAccountList) |

### 

### StartGame Servlet Class

The StartGame Servlet Class together with the GameServlet are the game managers for the Monopoly game. Directly or indirectly the Game Class owns all of the game assets. It also controls the game turns for the players.

|  |
| --- |
| **Class StartGame Servlet** |
| -User user  -Player player  -int gameId  -ArrayList<Player> playerList  -Bank bank  -UserPlayerDatabaseController database  -GameServlet gameservlet |
| -processRequest()  -doGet()  -doPost() |

### Bank Account Class

The BankAccount class keeps track of the account balance of a particular player. Each instance of this class stores the id of the particular bank account, the bank id associated with the account, the balance, and the player associated with the balance.

|  |
| --- |
| **Class BankAccount** |
| +int bankAccountID  +int bankID  +int playerID  +int cashBalance  // used at the start of the game to set the initial bank account balance  +static final int startingBalance  // used to update the database  +static BankDatabaseController database |
| //Constructors  +BankAccount()  +BankAccount(int bankAccountID, int bankID, int playerID)  // needed by the database controller to create a bank account from the database  +void initializer(int bankAccountID, int bankID, int playerID, int balance)  +int getBankAccountID()  +int getBankID()  +int getPlayerID()  +void addToAccountBalance(int income)  +void subtractFromAccountBalance(int amount)  +int getCashBalance()  +void setCashBalance(int cashBalance)  +void setBankAccountID(int bankAccountID)  +void setBankID(int bankID)  +void setPlayerID(int playerID)  +static int getStartingBalance() |

### The Database Controller Classes

The Database Controller classes are a set of Fasades for database access. Except for the GameDatabaseController, they are used to instantiate the new objects needed by a new Monopoly game based on the contents of the database. To prevent database corruption they also use the Singleton design pattern. The use of the Singleton pattern is why they do not all inherit from a common database class.

These 6 classes encapsulates all of the database interactions from the rest of the application so that no other classes will contain SQL or database connections. For brand new entries to the database the ids will be auto generated by the database. The new object being requested is returned by the Database Controller class when the new object is first created and saved to the database. Special functions specifically for the handling of login search and retrieval of users are also included for ease of use in the UserPlayerDatabaseController.

The class names are UserPlayerDatabaseController, GameDatabaseController, BankDatabaseController, RealEstateDatabaseController, UtilityDatabaseController, and RailroadDatabaseController.

**GameDatabaseController Class**

|  |
| --- |
| **Class GameDatabaseController** |
| * GameDatabaseController instance // static for the singleton design pattern   // database connection and query variables   * Connection connection * Statement statement * ResultSet resultSet * String url * String username * String password |
| // function for database connection   * void getDatabaseConnection()   // private constructor for the singleton design pattern   * GameDatabaseController()   // static for the singleton design pattern  +GameDatabaseController getInstance()  // returns auto-generated database id for the new game just saved in the database  + int addNewGame() |

**UserPlayerDatabaseController Class**

|  |
| --- |
| **Class UserPlayerDatabaseController** |
| * UserPlayerDatabaseController instance // static for the singleton design pattern   // database connection and query variables   * Connection connection * Statement statement * ResultSet resultSet * String url * String username * String password |
| // function for database connection   * void getDatabaseConnection()   // private constructor for the singleton design pattern   * UserPlayerDatabaseController()   // static for the singleton design pattern  +UserPlayerDatabaseController getInstance()  //checks the database to ensure a unique login or a correct login  +boolean doesUserLoginExist(String username, String password)  // adds a new user’s registration and login information to the database  +boolean registerNewUser(String firstname, String lastname, String username, String password)  // instantiates and returns a new object using information from the database  // where userID and playerID are the primary keys in the user and player database tables  +User getUserByID(int id)  +Player getPlayerByID(int id)  // creates a new player for a given user for a new game  +Player addNewPlayer(User user, int token\_id, int game\_id)  // creates and returns an array of new users for a new multiplayer game  ArrayList<Player> getPlayerListByGameID(int gameID)  // this lets a user change the login information when needed  +boolean updateUserLogin(int userID, String newUsername, String newPassword)  // the player has a token id that is used to lookup the correct image file for a given token  // primary key token id  String getTokenFileName(int token\_id)  // deletes the player when the game is done  boolean deletePlayer(int playerID)  // deletes the registration and login information for a user from the database if ever needed  boolean deleteUser(int userID) |

**BankDatabaseController Class**

|  |
| --- |
| **Class BankDatabaseController** |
| * BankDatabaseController instance // static for the singleton design pattern   // database connection and query variables   * Connection connection * Statement statement * ResultSet resultSet * String url * String username * String password |
| // function for database connection   * void getDatabaseConnection()   // private constructor for the singleton design pattern   * BankDatabaseController()   // static for the singleton design pattern  +BankDatabaseController getInstance()  // these functions make new objects for a new game or retrieve an existing object’s state from  // the database when needed  +Bank getBankByID(int bankID)  +BankAccount getAccountByID(int accountID)  +ArrayList<BankAccount> getAccountListByBankID(int bankID)  +Bank addNewBank(int[] playerIDs)  +BankAccount addNewBankAccount(int bankID, int playerID)  //updates the state of the bank or bank account in the database  +boolean updateBankNumberHouses(int bankID, int numberOfHouses)  +boolean updateCashBalance(BankAccount account)  // used to delete the bank and all of its accounts at the end of a game  +boolean deleteBank(int bankID)  +boolean deleteBankAccount(int accountID) |

**RealEstateDatabaseController Class**

|  |
| --- |
| **Class RealEstateDatabaseController** |
| * RealEstateDatabaseController instance // static for the singleton design pattern   // database connection and query variables   * Connection connection * Statement statement * ResultSet resultSet * String url * String username * String password |
| // function for database connection   * void getDatabaseConnection()   // private constructor for the singleton design pattern   * RealEstateDatabaseController()   // static for the singleton design pattern  +RealEstateDatabaseController getInstance()  / /these functions are used when getting or make a set of new objects for a new game  +RealEstate getRealEstateByID(int gameID, int spaceID)  +ArrayList<Integer> getRents(int spaceID)  ArrayList<RealEstate> addAllRealEstatesToGame(int gameID)  // updates the Real Estate information in the database to save game state  +boolean updateRealEstate(RealEstate realestate)  // used to find if a player has a color set real estate monopoly for various game functions  +boolean doesPlayerHaveMonopoly(int ownerID, int numberForMonopoly)  // deletes the real estate entries from the database at the end of the game  boolean deleteAllGameRealEstates(int gameID) |

**UtilityDatabaseController Class**

|  |
| --- |
| **Class UtilityDatabaseController** |
| * UtilityDatabaseController instance // static for the singleton design pattern   // database connection and query variables   * Connection connection * Statement statement * ResultSet resultSet * String url * String username * String password |
| // function for database connection   * void getDatabaseConnection()   // private constructor for the singleton design pattern   * UtilityDatabaseController()   // static for the singleton design pattern  +UtilityDatabaseController getInstance()  // these functions make a set of new Utility objects for a new game  +Utility getUtilityByID(int gameID, int spaceID)  +ArrayList<Property> addBothUtilitiesToGame(int gameID)  // updates the utility in the database to save game state  +boolean updateUtility(Utility utility)  // used to find if a player has a utility monopoly for various game functions  +boolean doesPlayerHaveMonopoly(int ownerID)  // deletes the utilities from the database at the end of the game  +boolean deleteBothGameUtilites(int gameID) |

**RailroadDatabaseController Class**

|  |
| --- |
| **Class RailroadDatabaseController** |
| * RailroadDatabaseController instance // static for the singleton design pattern   // database connection and query variables   * Connection connection * Statement statement * ResultSet resultSet * String url * String username * String password |
| // function for database connection   * void getDatabaseConnection()   // private constructor for the singleton design pattern   * RailroadDatabaseController()   // static for the singleton design pattern  +RailroadDatabaseController getInstance()  // these functions make a set of new Railroad objects for a new game  +Railroad getRailroadByID(int gameID, int spaceID)  +ArrayList<Railroad> addAllRailroadsToGame(int gameID)  // updates the railroad in the database to save game state  +boolean updateRailroad(Railroad railroad)  // used to find if a player has a railroad monopoly for various game functions  +int numberRailroadsOwned(int ownerID)  // deletes the railroads from the database at the end of the game  +boolean deleteAllGameRailroads(int gameID) |

### Login and Register Servlet classes

The Login class logs in the particular registered User on the Website and take the User to the Lobby Page. If the User tries to login but not registered, then it would ask the User to Register. The Register class then comes in effect and Registers a New User to the database.

|  |
| --- |
| **Class Login implements Servlet** |
| -String userName  -String password  -Database database |
| + Login(String username, String password)  +User getUser(String loginName, String password) // get the User from the database  //checks the database if there is a user associated with provided username  +boolean isRegistered()  /\*checks if the User is registered: if True logs in the User, take User to Lobby Page else asks to Register \*/  //logs out the User, and take User to Home Page  +void logOut(User user) |

### Register Class

Register class gets user’s name, email, username, password, and image then use these information to register. It then creates a new record in the database.

|  |
| --- |
| **Class Register implements Servlet** |
| -String name  -String email  -String userName  -String password  -String imageFile  -Database database |
| //creates a new User with required datafields  +Register(String name, String email, String userName, String password, String imageFile)  //changes/set the Avatar for User  +void setAvatar(String imageFile)  +updateDatabase(this) // add the new User to the database |

### Sell Property Class

The SellProperty class is a wrapper class for the property to be sold. Once a Player decides to sell a property either to the bank or other Player, the SellProperty class provide sale\_id for the property involved during the transaction.

|  |
| --- |
| **Class SellProperty** |
| -int saleId  -int sellPrice  // creates instance for the selling the property by setting sale\_id = property\_id  -SellProperty(int propertyId, int sellPrice) |
| //check price > 0  +void setSalePrice(int price)  +int getSellPrice()  +void setSaleId(int saleId) //sets the sale\_id for the property  +int getSaleId()// returns the sale\_id for the property  // sell the property to other player  +void sellToPlayer(Player otherPlayer) |

### GameServlet Class

The GameServlet class is mainly responsible for loading and displaying the players’ tokens, houses, hotels, cards, board, and properties, in the monopoly universe. The Game Board contains most of the controller actions for the game. Depending on which card a player draws or his/her dice roll, the Game Board will call the appropriate subclass of Space to do the actions for rent, salary, purchase of properties, etc. The game servlet controls the general progress, display, and logic of the game.

|  |
| --- |
| **Class GameServlet** |
| -int gameId  -Bank bank  -ArrayList<Players> players  -ArrayList<properties> properties  -boolean isInJail  -BankDatabaseController bankData  -UserPlayerDatabaseController playerData -Player activePlayer |
| -processRequest()  -doGet()  -doPost() |

### Dice Class

The Dice class must generate two random numbers between 1 and 6 save them separately so that the correct images of the dice can be drawn and return the results of the combined value of the two die. This class uses the Singleton design pattern.

|  |
| --- |
| **Class Dice** |
| * Random dieGenerator * int dieOne * int dieTwo * String dieOneImagePath * String dieTwoImagePath * static Dice instance // for the singleton pattern |
| + Dice()  +initialize(String dieOneImage, String dieTwoImage)  +void rollDice() // generates new values for dieOne and dieTwo  +int getDiceTotal() // returns sum of dieOne and dieTwo  //Getters  +static Dice getinstance() // for the singleton pattern  + int getDiceOneValue()  + int getDiceTwoValue()  + String getDieOneImage()  + String getDieTwoImage()  //Setters  +void setDieOne(int dieOne)  +void setDieTwo(int dieTwo)  +void setDieOneImagePath(String dieOneImagePath)  +void setDieTwoImagePath(String dieTwoImagePath) |

### Space Class

The set of locations on the Game Board where a Player can land during a turn.

|  |
| --- |
| **Class Space** |
| - int spaceID |
| + Space(int location) +Space()  +void initialize(int location)  +int getSpaceID()  + void setSpaceID(int location) |

### Property Class

The Property class is the root for all other property classes and contains all the basic data that all properties will share.

|  |
| --- |
| **Abstract Property extends Space** |
| * int ownerID * int gameID * String name * int purchasePrice * boolean isMortgaged * int mortgageAmount |
| +Property(int owner, int location, String name, int price, int game)  +Property()  +void initialize(int owner, int location, String name, int price, boolean mortgage, int gameID)  +abstract int calculateRent (int numOwned)  //Getters  +int getOwnerID()  +int getGameID()  +String getName()  +int getPurchasePrice()  +boolean getIsMortgaged()  +int getMortgageAmount()  //Setters  +void setOwnerID(int ownerID)  +void setGameID(int propertyID)  +void setName(String name)  +void setPurchasePrice(int purchasePrice)  +void setIsMortgaged(boolean isMortgaged)  void setMortgageAmount(int mortgageAmount) |

### Real Estate Class

The RealEstate class is what is used for all colored properties and contains extra information to keep track of houses and a static table to look up the cost of houses for a given real estate property color.

|  |
| --- |
| **Class RealEstate extends Property** |
| * String color * int numberOfHouses * int costOfAHouse * int numberForMonopoly * ArrayList<Integer> rents   +static RealEstateDatabaseController database |
| +RealEstate(int owner, int location, String name, int price, int costOfAHouse, boolean hasMortgage, String color, int numberOfHouses, int numberForMonopoly, ArrayList<Integer> rents, int gameID)    +RealEstate()  +void initialize(int owner, int location, String name, int price, int costOfAHouse, boolean hasMortgage, String color, int numberOfHouses, int numberForMonopoly, ArrayList<Integer> rents, int gameID)  + int CalculateRent(int numOwned) // override abstract parent function  //Getters  + String getColor()  + int getNumberOfHouses()  + int getNumberForMonopoly()  + ArrayList<Integer> getRents()  +int getCostOfAHouse()  //Setters  + void setColor(String color)  + void setNumberOfHouses(int numHouses) // override to update database  + void setNumberForMonopoly(int numberForMonopoly)  + void setRents(ArrayList<Integer> rents)  +void setCostOfAHouse(int costOfAHouse)  +void setOwnerID(int ownerID) // override to update database |

### Railroad Class

The Railroad class doesn't require any additional information, but must calculate rent based on the number of railroads owned by the owning player.

|  |
| --- |
| **Class Railroad extends Property** |
| -static final int baseRent = 25;  +static RailroadDatabaseController database |
| + Railroad(int owner, int location, String name, int price, int game)  +Railroad()  +void initialize(int owner, int location, String name, int price, boolean mortgaged, int game)  +int calculateRent(int numOwned) // override abstract parent function  //Getters  + static int getBaseRent()  //Setters  + void setOwnerID(int ownerID) // override to update database  + void setIsMortgaged(boolean isMortgaged) // override to update database |

### Utility Class

The Utility class like the Railroad class doesn't require any additional information, but must calculate rent based on the number of utilities owned by the owning player and the dice roll of the player who landed on the utility space.

|  |
| --- |
| **Class Utility extends Property** |
| +static UtilityDatabaseController database |
| +Utility(int owner, int location, String name, int price, int game)  +Utility()  +void initialize (int owner, int location, String name, int price, boolean mortgage, int game)  + int calculateRent(int numOwned) // override abstract parent function  // does the real rent calculation which is a multiple of the dice roll for utilities  +int calculateRent(int numOwned, int diceRoll)  //Setters  +void setOwnerID(int ownerID) // override to update database  +void setIsMortgaged(boolean isMortgaged) // override to update database |

### Card and Chance and Community Chest Classes

This is the base class for all the other card classes. It contains the information needed by all card objects including: text description, cardID, player who drew the card, and if it is a chance or community chest card

|  |
| --- |
| **Abstract Card** |
| -int cardID  -String cardDescription  -Player cardDrawer  -String cardStackType |
| //Constructor  + Card()  + Card(int cardID, String cardDescription, Player cardDrawer, String cardStackType)  + void initialize(int cardID, String cardDescription, Player cardDrawer, String cardStackType)  + abstract void playCard() throws Exception;  // Shows description of drawn card  + String getCardDescription()  + void setCardDescription(String description)  + void setCardId(int id)  + int getCardId()  + Player getCardDrawer()  + setCardDrawer(Player cardDrawer)  + String getCardStackType()  + void setCardStackType(String cardStackType) |

### GetoutOfJailFreeCard

This is a special card that the player who draws it gets to keep and use to get out of jail for free.

|  |
| --- |
| GetOutOfJailFreeCard extends Card |
|  |
| //Constructor  + GetOutOfJailFreeCard()  + GetOutOfJailFreeCard(int cardID, String cardDescription, Player cardDrawer, String cardStackType)  + void initialize(int cardID, String cardDescription, Player cardDrawer, String cardStackType)  + void playCard() throws Exception |

### MoveToLocationCard

This class is for cards that force the player to move to a specific location. Some of these cards have special modifers like changes in the rate owed and if the player gets money for passing go.

|  |
| --- |
| MoveToLocationCard extends Card |
| -String spaceTypeToMoveTo  -boolean hasRentModifier  -boolean doNotPassGo |
| //Constructor  + MoveToLocationCard()  + MoveToLocationCard(int cardID, String cardDescription, Player cardDrawer, String cardStackType, String spaceTypeToMoveTo, boolean hasRentModifier, boolean doNotPassGo)  + void initialize(int cardID, String cardDescription, Player cardDrawer, String cardStackType, String spaceTypeToMoveTo, boolean hasRentModifier, boolean doNotPassGo)  + void playCard() throws Exception  + String getSpaceTypeToMoveTo()  + void setSpaceTypeToMoveTo(String spaceTypeToMoveTo)  + boolean isHasRentModifier()  + void setHasRentModifier(boolean hasRentModifier)  + boolean isDoNotPassGo()  + void setDoNotPassGo(boolean doNotPassGo) |

### MoveToSpaceCard

This class is for the card that tells the player to move back 3 spaces, it could be used for other kinds of movements based on number of spaces.

|  |
| --- |
| MoveToSpaceCard extends Card |
| -int spaceChange |
| //Constructor  + MoveToSpaceCard()  + MoveToSpaceCard(String cardDescription, Player cardDrawer, String cardStackType,int spaceChange, int cardID)  + void initialize(String cardDescription, Player cardDrawer, String cardStackType,int spaceChange, int cardID)  + void playCard() throws Exception  + int getSpaceChange()  + void setSpaceChange(int spaceChange) |

### RealEstateRepairCard

This class is for the cards that require the player who receives it to pay a variable amount based on the number of houses and hotels that player owns.

|  |
| --- |
| RealEstateRepairCard extends Card |
| -int costPerHouse  -int costPerHotel |
| //Constructor  + RealEstateRepairCard()  + RealEstateRepairCard(int cardID, String cardDescription, Player cardDrawer, String cardStackType, int costPerHouse, int costPerHotel)  + void initialize(int cardID, String cardDescription, Player cardDrawer, String cardStackType, int costPerHouse, int costPerHotel)  + void playCard() throws Exception  + int getCostPerHouse()  + void setCostPerHouse(int costPerHouse)  + int getCostPerHotel()  + void setCostPerHotel(int costPerHotel) |

### StaticCashTransferToPlayerCard

This class is for cards that transfer sums of money that are known at the start of the game.

|  |
| --- |
| CashTransferToPlayersCard extends Card |
| -int amountTransfered |
| //Constructor  + StaticCashTransferCard()  + StaticCashTransferCard(int cardID, String cardDescription, Player cardDrawer, String cardStackType, int amountTransfered)  + void initialize(int cardID, String cardDescription, Player cardDrawer, String cardStackType, int amountTransfered)  + void playCard() throws Exception  + int getAmountTransfered(  + void setAmountTransfered(int amountTransfered) |

## **Design Overview**

### Dynamic Web Content

* Loading the Game Board and changing the Game Page content per Player actions with responses from the Server code.
* Technology Used:
  + AJAX
  + jQuery
  + JSP
  + Java Servlets
* What is AJAX ?
  + AJAX stands for Asynchronous JavaScript and XML. In short, AJAX is about loading data in the background and display it on webpage without reloading the whole page.
* jQuery along with AJAX ?
  + Yes, you heard it right, jQuery is a web framework provides a robust and enormous JavaScript library. This enhances client-side scripting and of course is the carrier of communication between Server-Client.

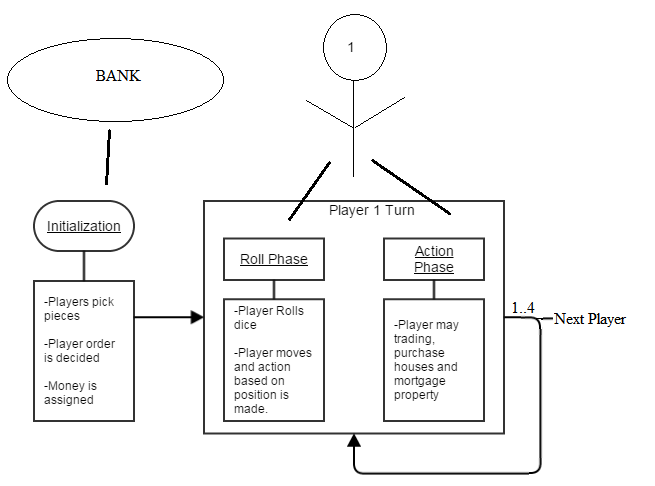
jQuery AJAX carrying on the information between Client-Server:

* Here is an example of a HTTP Get request to retrieve some data from server to display on the Web Page.
  + $.get(‘URL to servlet’, {optional parameters}, function(resp){});
  + The above request can be sent if User triggered any event such as button click etc.
  + The servlet will look up the parameter of the request and responds accordingly thereby sending the response in the ‘resp’, which is an HTTP Response object.
* The following is an example of HTTP Post request to send any data from Web Page to Server.
  + $.post((‘URL to servlet’, {data}, function(resp){});
  + Post is usually prioritized while sending any data to Server due to Security Concerns because sending data via HTTP Get request often shows up the data as parameters in the Web URL, which makes them visible to anyone out there.
  + The rest functionality is same as the HTTP response is again stored in resp.

Server Side Technology:

* Servlets
  + Java is big on Web and Servlets are one major reason behind it.
  + Java Servlet library is stored inside package ‘javax’, and handles Hypertext Transfer protocols.
  + A Servlet class is a classic Java class and becomes child class extending HttpServlet class. HTTP requests sent from the Web are received by Servlets
  + Here is a doGet() method that is called from the Web:
    - doGet(HttpServletRequest request, HttpServletResponse response){
      * //check request parameters and act
      * //send the response to the Web Page

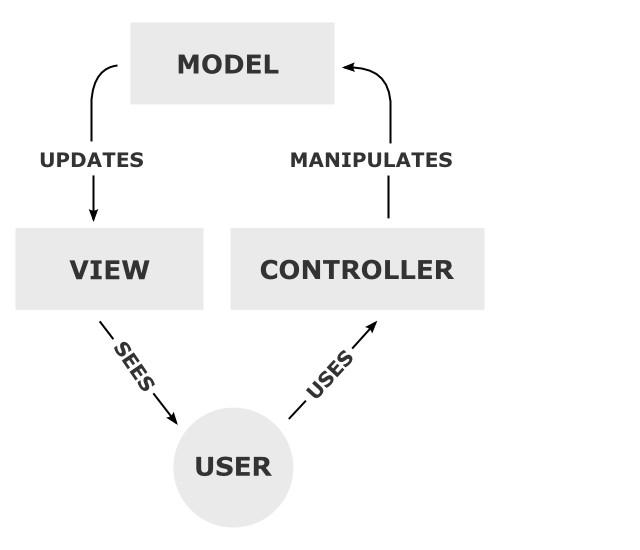
### Flow Chart Diagram



## **Design Patterns**

### 1. Model-View-Controller Introduction

Model View Controller design pattern also go by its short name as MVC. It is an architecture pattern that divides the software into three interconnected parts as Model, View and Controller. MVC pattern is great for object-oriented applications and have significant usage and advantage for Web applications. Enigma Monopoly has extensive Web environment as it is being hosted on a Web server for the Users to play the game. Enigma Monopoly requires a Presentation for the Users on the Web to perform interactions with it. Enigma Monopoly class structure serves the User requests by updating the screen presentation for the particular user and thus the functionality carries on. Here is a simple overview of MVC architecture.



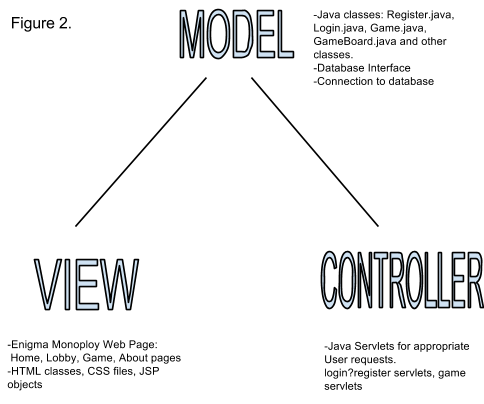
### MVC in Enigma Monopoly

Figure 2 overviews some of the Enigma Monopoly MVC architecture showing what will be in each part. Following is more detailed description of Enigma Monopoly’s MVC architecture

**Model.** This consists of all the Java classes described in Class Diagrams, thereby all the functionality and rules of Enigma Monopoly lies here. Also, the database to hold game state, users and any other data associated with Enigma Monopoly. Model responds to servlet requests sent by Controller and generate JSP to update the User view.

**View.** Enigma Monopoly’s web pages makes up the view for the User. Enigma Monopoly has four web pages as Home Page, Lobby Page, Game Page and About Page. Each page is build using Web Front End technologies like HTML, CSS and JSP. Once User requests for the Home page for Enigma Monopoly, the HTML file for Home will be pulled on the Web. The User can browse around Enigma Monopoly pages, and can Login/Register to view Lobby and Game pages. JSP generates content for User requests, so whenever User is involved in an Active, would like to start a game, Login/Register etc, JSP updates the View or User’s screen.

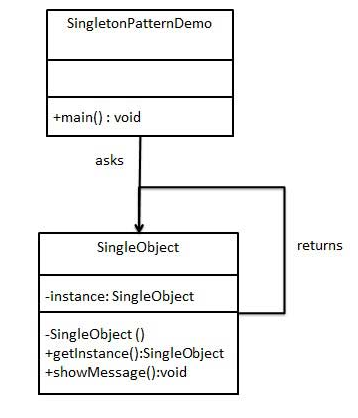
**Controller.** This holds what the User actually asks for, it is the main interface between the View and the Model. Controller is comprised of servlets that acts as servers to User requests by carrying out User requests to the Model. Each servlet in the Controller is associated with an appropriate Java class in Model. For example, in Enigma Monopoly when User starts a new game, the Game servlet in the Controller calls the Game class in Java to create a new game. Likewise, when a User requests to Register, the Controller sends Register servlet with the information from User to Register class to create new User in the database.



# 

### 2. Singleton Design Pattern

This pattern accounts for having a single instance of a class. The class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class. This is accomplished by declaring a static instance of a class and initializing it once the class is instantiated. Classes that follows this pattern are: Dice class and all DatabaseController classes in the Database package.



Since the Dice gets rolled by players, and all it really needs to return is the total value of rolled die. This can be accomplished by just having one instance of the Dice class, instead of creating a new object every time a Dice being rolled. Similarly the Database Controllers should return their instance after setting up the connection. The queries can then be submitted using the DatabaseController class instance. Following Singleton pattern definitely makes life easier for Garbage collector as well save lot of memory.

**3. Database Facade Design Pattern**

The Database package also follows the Facade Design pattern as all the Database Controller classes provides common functionality of setting up the database connection, submitting queries and returning result sets from the database. Using the singleton pattern, class instance for the database controller class is used and the interface provided by Database Controller class is used to interact with the database, for example returning list of properties, player’s account balance, list of bank accounts etc. to the Game servlet class or the Business logic in a whole.

**Monopoly Relational Database (MySQL)**



# Milestone Three

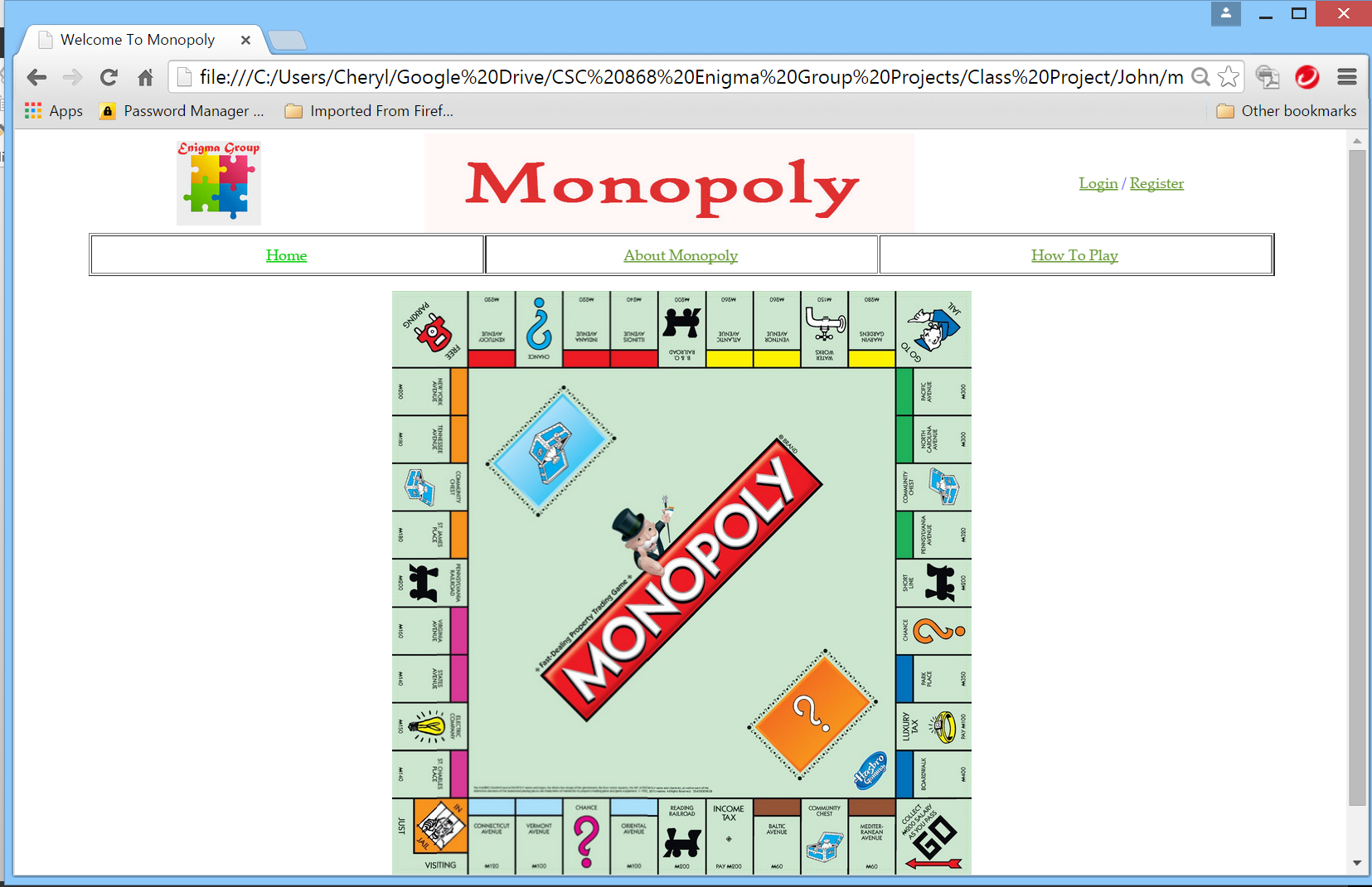
## **Web Pages**

### Use Case

Eric is a 14 year old student wanting to play the classic Monopoly game with his friends. He goes to the home page. It has been a while since he last played it so he reads the rules in the How To Play page. Excited the play the game, he goes ahead and register a new account where he enters his username, password, first name and last name. Upon registering he then logs in with his newly created username and password.He is then redirected to the Lobby page where he can start a new game or join an existing game with other users. In this page he sees his account information and an option to edit that info. He joins the game that his friend created and is now redirected to the Game page. He sees the name of his friends in the game he joined and some links of game actions for playing the game. After playing the game, he leaves the game and then logs out.

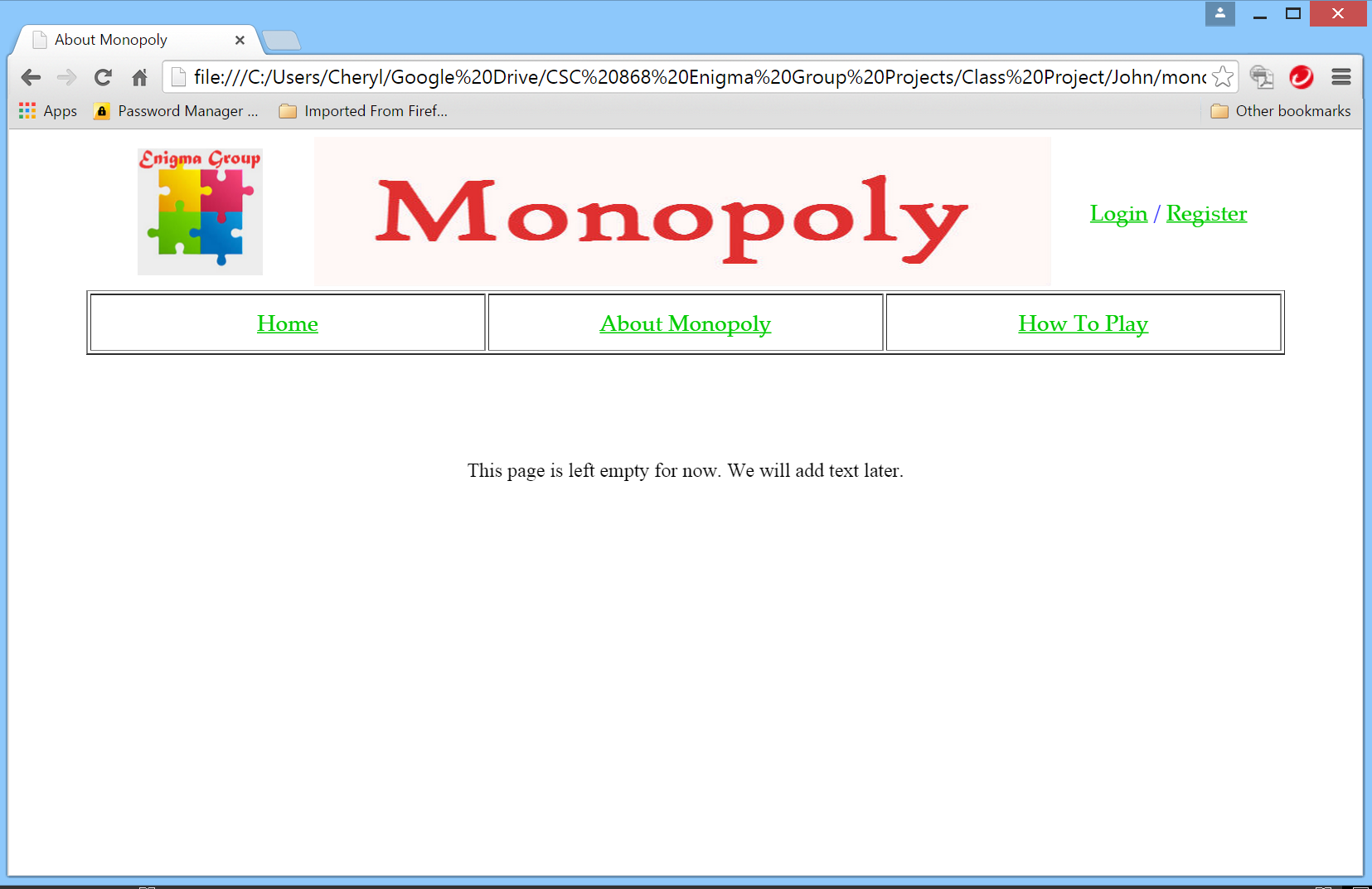
### Home Page

This is the page where a user will first arrive at the Monopoly web site. It provides links to the public pages that are accessible without being logged in. From here a user can browse information about the game and decide to register and login to play the game.



### About Monopoly

This is the page where a user will be able to read some information about the game of Monopoly web site. It will give some information specifically about this class project. It will also have the legal disclaimers for the web site.



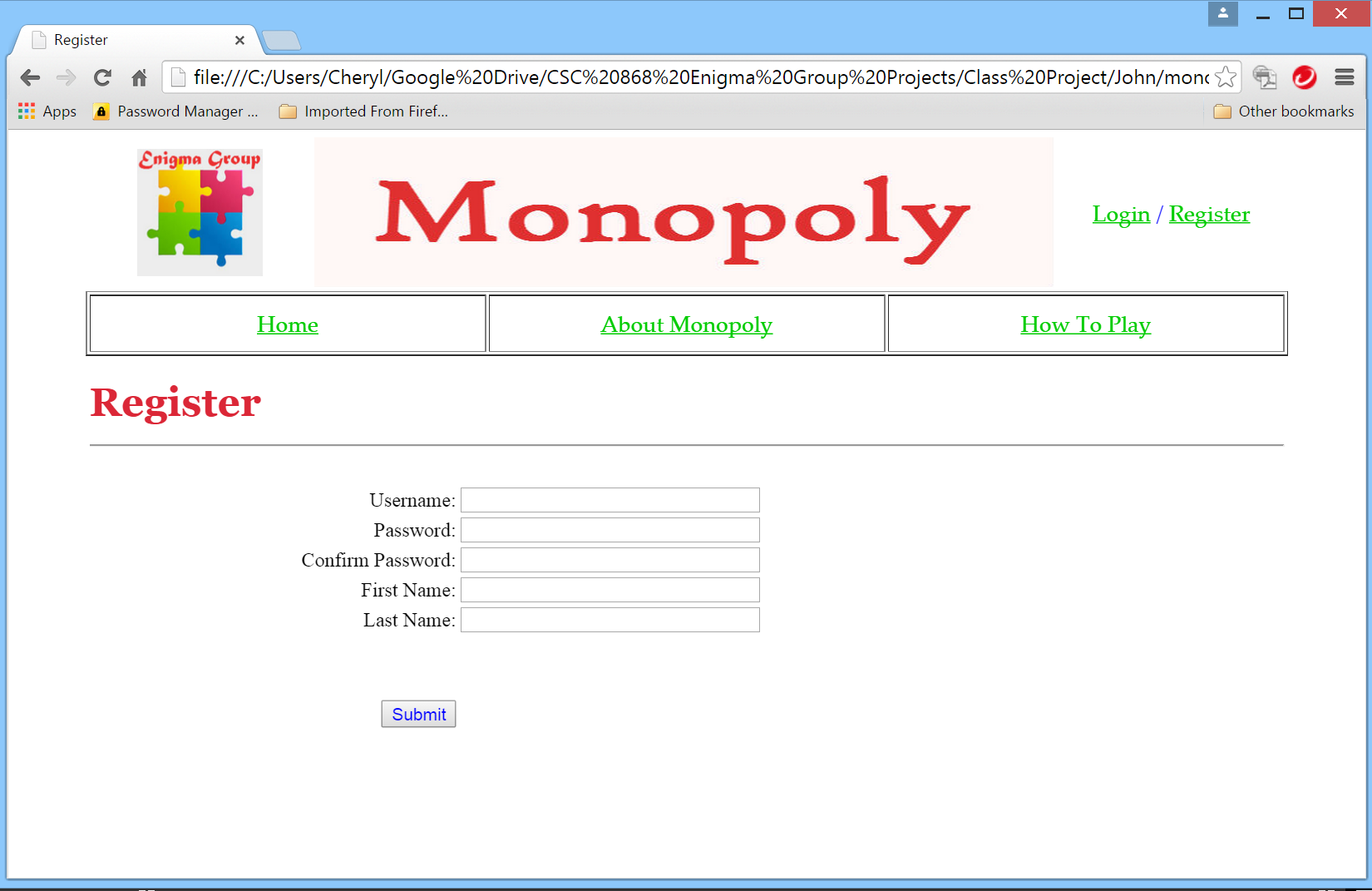
### How To Play

This page will help the user to understand how to play the game play of Monopoly. It gives a modified set of rules of the game as it will be implemented for our online version. Hopefully it will encourage the user to be interested in playing the game.



### Register Page

This is the page where the user will enter their information to become a registered user of the web site. Submitting the form will save their chosen a username and password to the database so they can login to play the game. Once registered the user will be taken to the Home page where they can choose to continue browsing or login to start playing Monopoly.



### Login Page

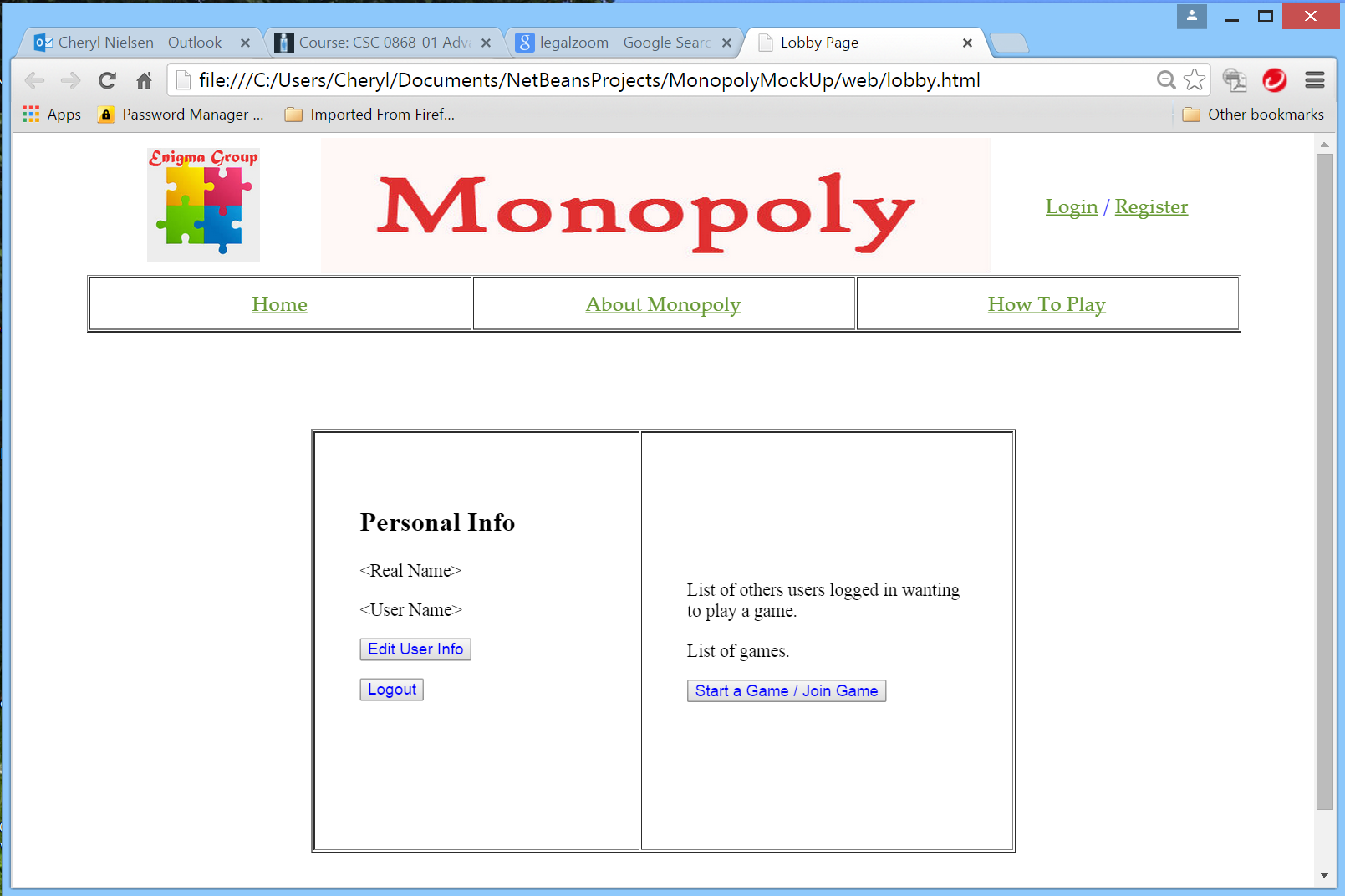
This is the page where a registered user can login to play a game of Monopoly. A successful login will then take them to the Lobby page where they can start a game.



### Lobby Page

The Lobby page is where users are taken after a successful login. This page is not accessible to users who are not logged in. The main purpose of the Lobby is to let users join with other users to start a new game or join an existing game. It is also where the user is taken when they leave a game, or the game has ended.

This page provides an area that lets the user change their registration information, such as changing their password. It is also where the user can logout.

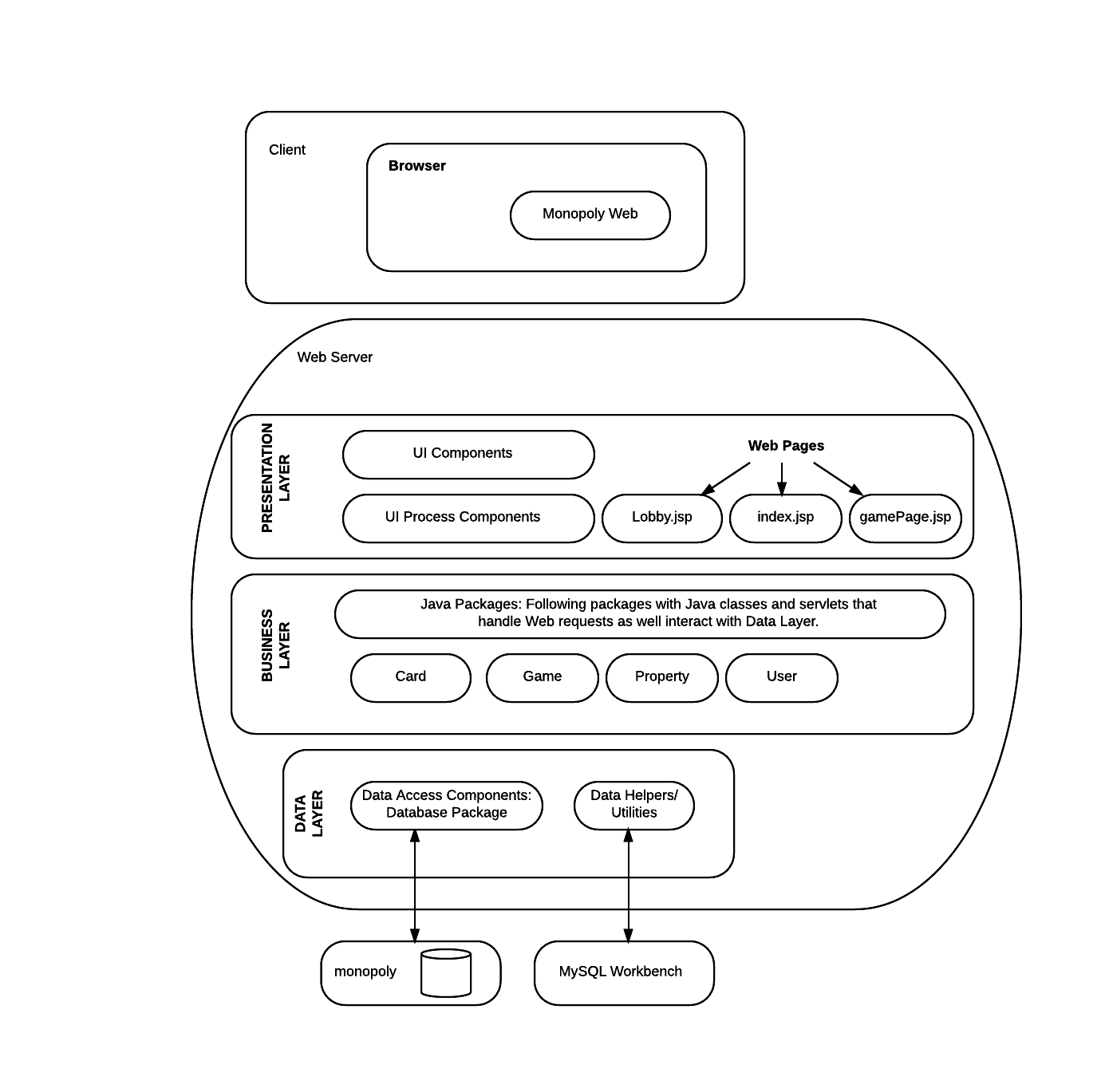


### Game Page

The Game page is where users are taken from the lobby to play the game. This page is not accessible to users who are not logged in. It provides the game board, information about the status of players, and various game action buttons.



**Design Overview Of System**



The application being deployed on Web follows general design and key attributes of a Web Application. Thereby, the system follows the typical layered structure. This also follows with MVC design pattern as the overview design of the system is described in the following three layer architecture as:

**Presentation Layer**

The presentation layer displays the UI and facilitates user interaction with the system via Web. This layer makes up the View of MVC pattern. The login/register pages provide interface for User to perform these operations. The jsp pages lobby.jsp and gamePage.jsp handles game session making user to player, and taking player to game mode. The gamePage.jsp then provides UI to play game as the User Guide describes the rest of the Front-End functionality.

This layer also consists of JavaScript files, jQuery and AJAX code to make Server calls sending user HTTP requests and as well handling responses from the server to update gamePage.jsp content accordingly. Here is an example of using jQuery & AJAX to get data from Business Layer:

$.get('Game.GameServlet', {"param": "rollDice"}, function (resp) {}

Here we sending the doGet() request to GameServlet class in Game package seding parameter ‘rollDice’, which is a way of telling servlet to call on Dice class to roll the Dice.

**Business Layer**

This layer holds the logic and the code for the system. The logic includes the servlet responsive to the Web pages and the rest of java classes making up the logic of the system. Four packages makes up the system logic: Card, Game, Property, User. The Game package holds the servlets login, Register, StartGame and GameServlet. The servlets handle HTTP Web requests through HTTPServlet objects sent via HTML form validation or AJAX mechanisms.

The rest of packages and classes are regular Java classes that holds respective data for the Player or other related elements of Game.

**Data Layer**

The Database package holds the Controller classes to interact with the system database named ‘monopoly’. The Database package classes following Singleton and Facade design pattern interact with the database.

The respective classes in the Database package acts as Database controllers to interact with the database thereby minimizing the number of open connections and using batch operations to reduce round trips to the database. The data layer uses MySQL workbench as GUI for database interactions for the team members.

## enigma.jpg

Monopoly Web Application User Guide

## **Table of Contents**

[Table of Contents](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.otugb4w9vpgq)

[Introduction](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.a6ps7r9crr16)

[Home Page](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.cnue309qcd1d)

[How To Play](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.uo2xbnoi8z0)

[New User Registration](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.dmgw6subikbg)

[Login](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.3nwy44mje2oz)

[Lobby Page](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.rvkegix9khdb)

[Game Page](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.o9zga1nu5cir)

[Game Page Actions](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.jbvr3x87s1wl)

[Community or Chance Card Space](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.7fmicw89b096)

[Free Parking](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.p03m6haqz9b)

[Property Space Pop Up Message](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.7nupsmk9as3y)

[Landing on “Going to Jail” Space](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.dwazlpo8zuq3)

[User is In Jail](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.r2zq7t5gxt61)

[Get Out of Jail Pop Up Message](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.waevqzy9dx1a)

[Passing “Go” Space](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.6v616r5hgbk0)

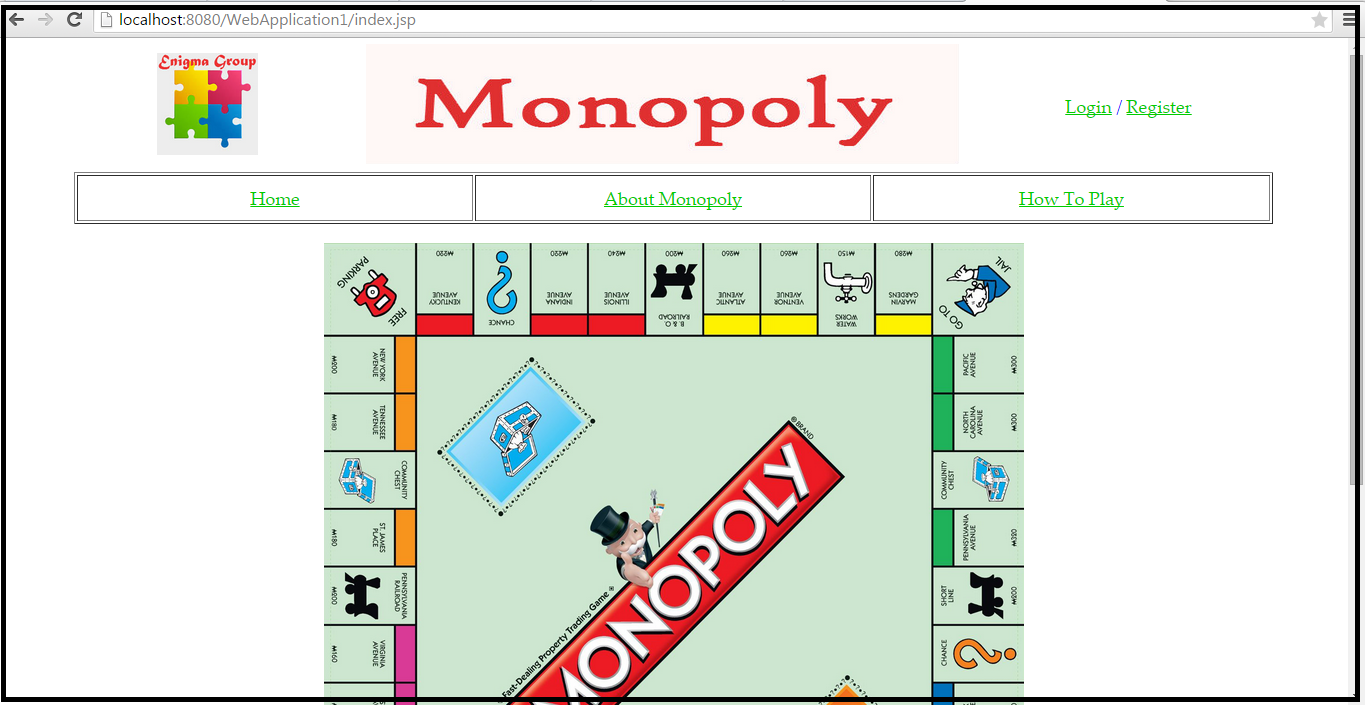
[Pay Tax Pop Up Message](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.jn66e8jt23uy)

[User Lands on “Visiting Jail” Space](https://docs.google.com/document/d/1PSgjrI5YlQl5JefOFcGq_V9T6Eb-AzWN5qOVI_aPT9s/edit#heading=h.gol6zbgkbzz)

## **Introduction**

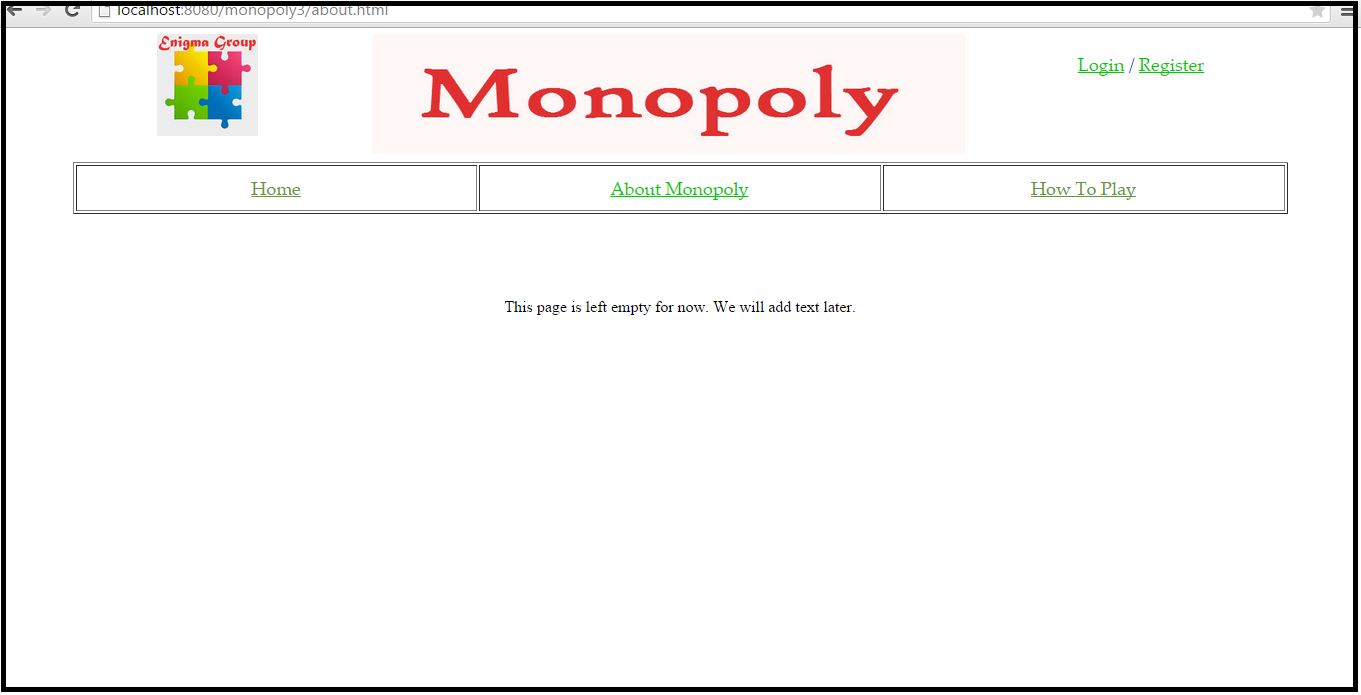
Enigma group’s Monopoly Web Application is a project for the CSC 668, Advanced Object Oriented Software Design and Development class for Spring 2015. A group of six students: Cheryl, Robert, John, Derek, Gill, and Kenneth decided to create this simple web game to showcase their skills in Object Oriented programming using different web technologies such as HTML, CSS, JavaScript, Java, etc. This web application is very intuitive and easy to use. It is free and available to anyone who has a computer and internet connection.

## **Home Page**



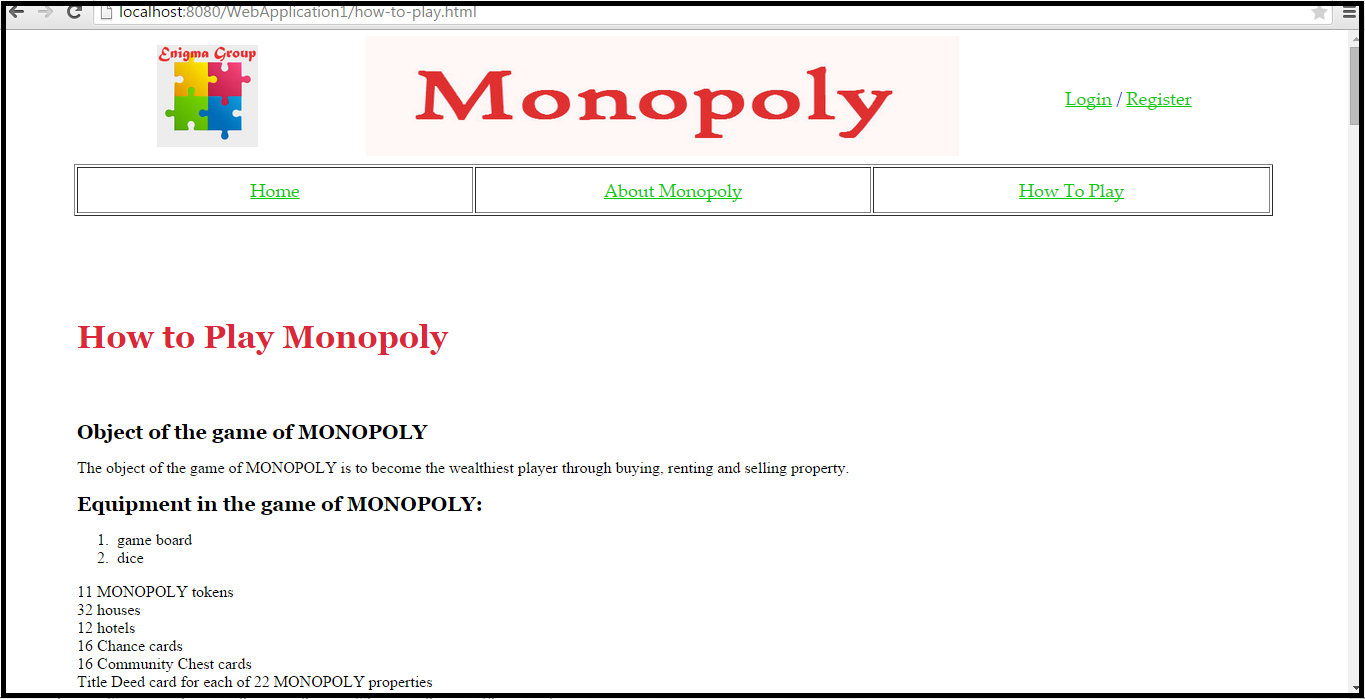
User lands on the homepage when they first visit the website. User can navigate to different pages from here.

## **About Monopoly**



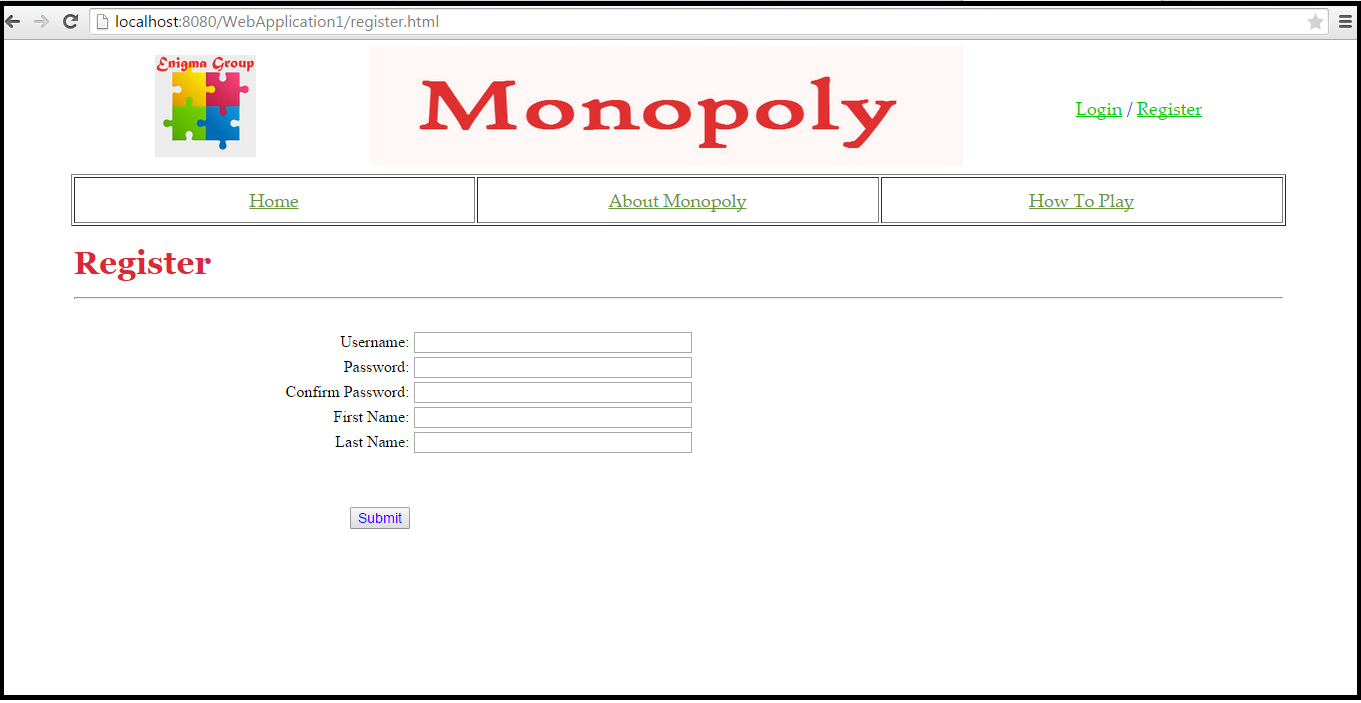
This page is empty for now but later on we plan to add more details about the site’s monopoly. We will also put legal disclaimers.

## **How To Play**



This is the page where the user can read the rules on how to play the classic game Monopoly.

## **New User Registration**



Users will be asked to enter the following information

**Username**

Enter the user’s username

**Password**

Enter the user’s password

**Confirm Password**

Enter the user’s password. It should match with the password entered in the above field.

**First Name**

Enter the user’s first name

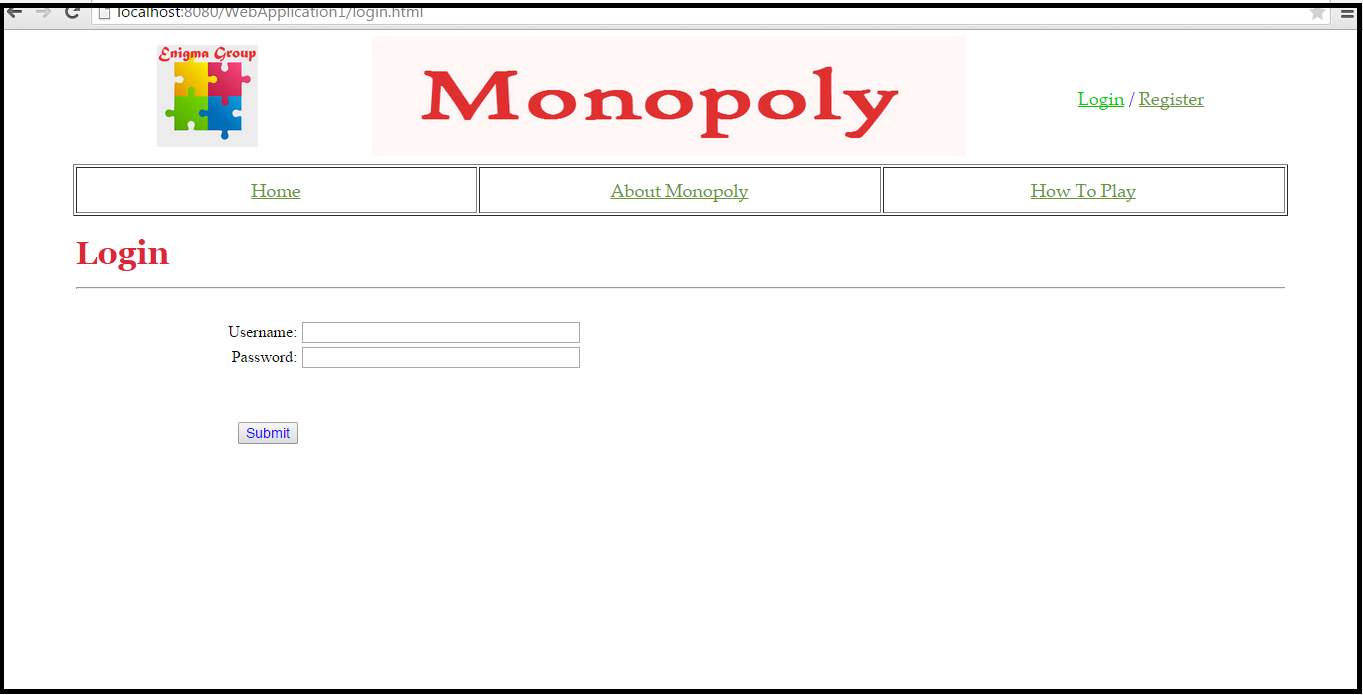
**Last Name**

Enter the user’s last name

**Submit**

Select "Submit" to submit the registration information and continue to the next screen.

## **Login**



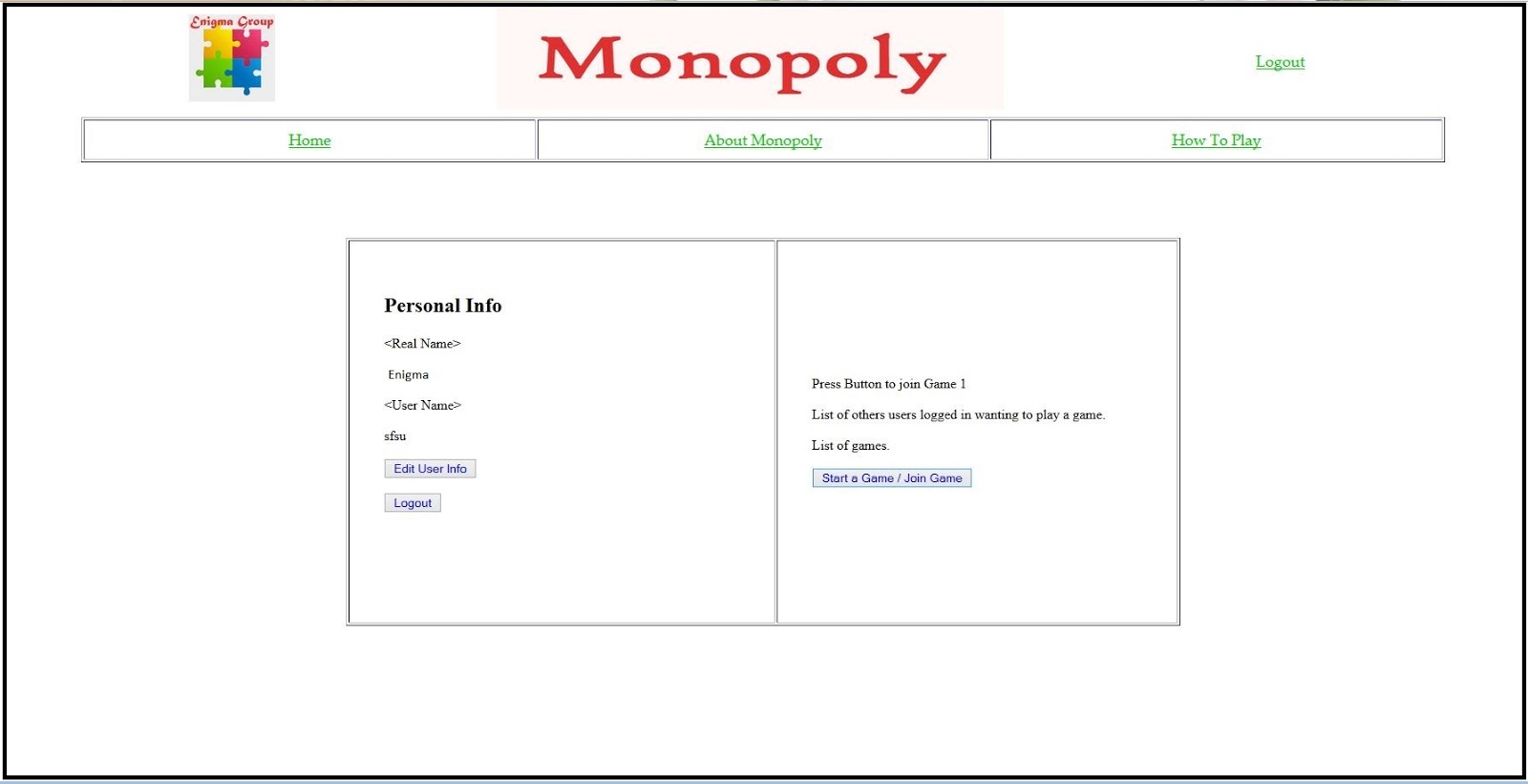
Users who have previously registered for the Monopoly Web Application must login by:

Entering their **Username**.

Entering their **Password**.

Selecting **Submit** to advance to the next screen and begin using the application.

## **Lobby Page**



Users who are able to login successfully will be redirected to the Lobby page.

Personal info is displayed on the left column.

**Real name**

**Username**

**Edit User info**

Clicking Edit User info redirects user to the Edit User Info page where they can change their user profile.

**Logout**

Clicking this button logs user out of the Lobby page and redirects to the home page.

**Press Button to Join Game 1**

Not yet implemented.

**List of other users logged in wanting to play a game**

Not yet implemented.

**List of games**

Not yet implemented.

**Start a Game / Join a Game**

Clicking “Start a Game / Join a Game” button redirects user to the game page where they can start playing the game.

## **Game Page**



**Player Info**

**Player**

Name of the player is displayed

**Game ID**

Game ID is displayed

**Player ID**

Player ID is displayed

**Game Actions**

**Roll Dice**

Clicking the “Roll Dice” button will move the user’s token depending on the random number (from 1 to 12) they receive.

**Pay Taxes**

When player lands in one of the two pay taxes spaces, this button will be enabled. When player clicks on this button, then the bank will be credited some amount of money.

**Get Out of Jail**

Disabled by default. Only enabled when the user is in jail.

**Player Stats**

**Player Drop Down Menu**

Not implemented yet. User can only view status for player 1.

**Property**

Not implemented yet.

**Account Balance**

Not implemented yet.

**In Jail?**

Will return “Yes” if player lands in “Go to Jail” space and is currently in jail, “No” otherwise.

**Game Stats**

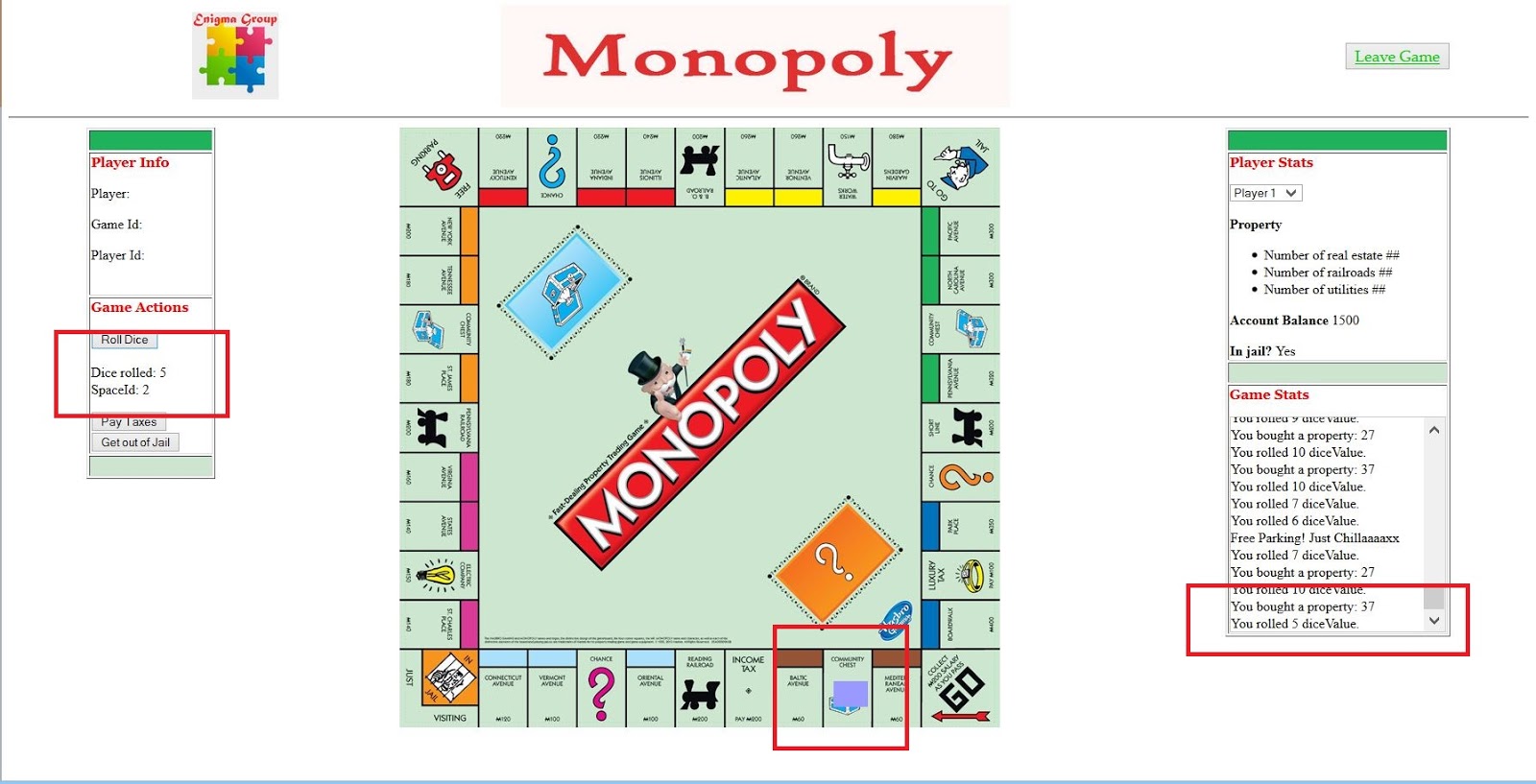
Every time the user rolls the dice, user actions are displayed on this page.

**Leave Game**

Clicking this button will redirect user to the home page.

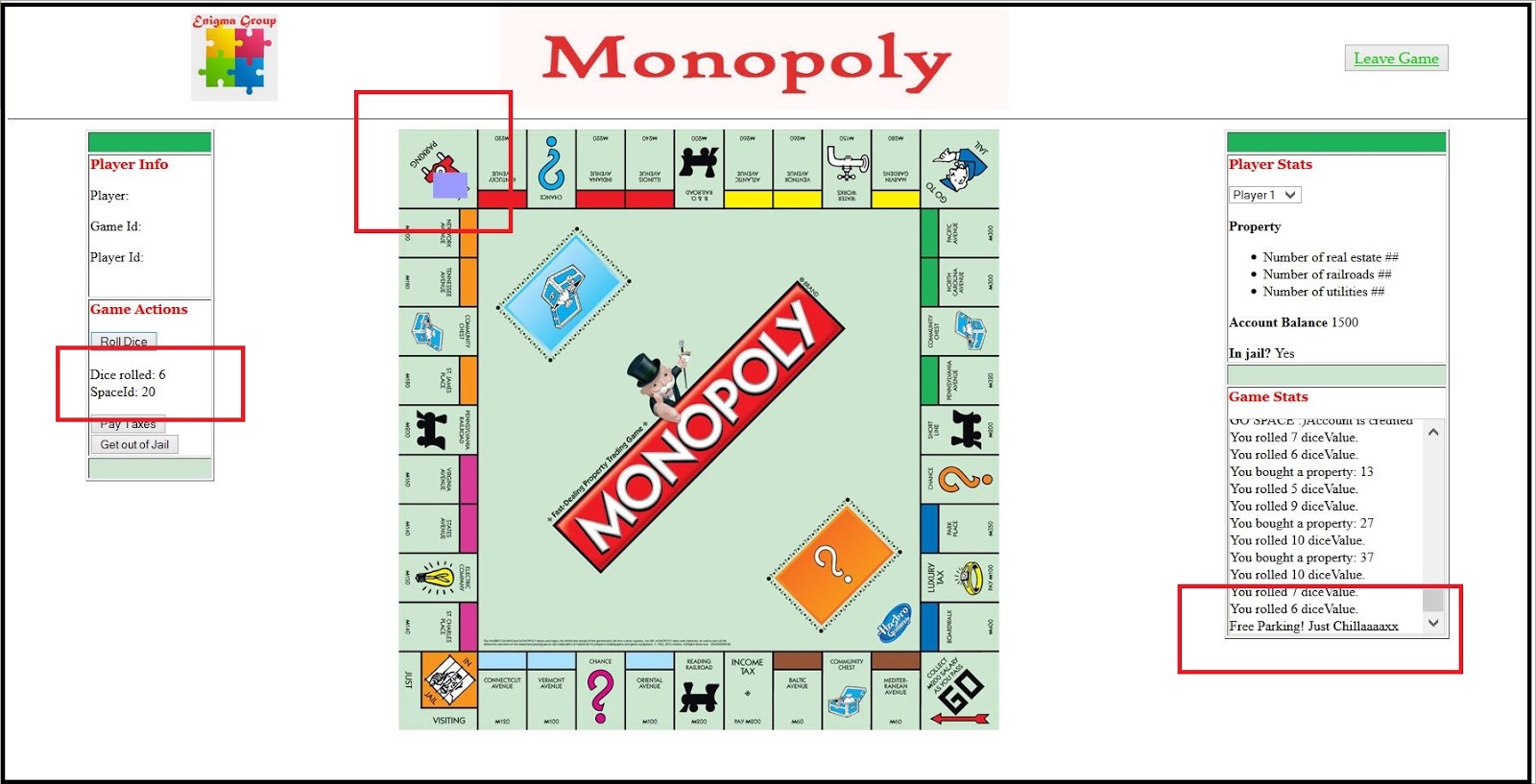
## **Game Page Actions**

## **Community or Chance Card Space**



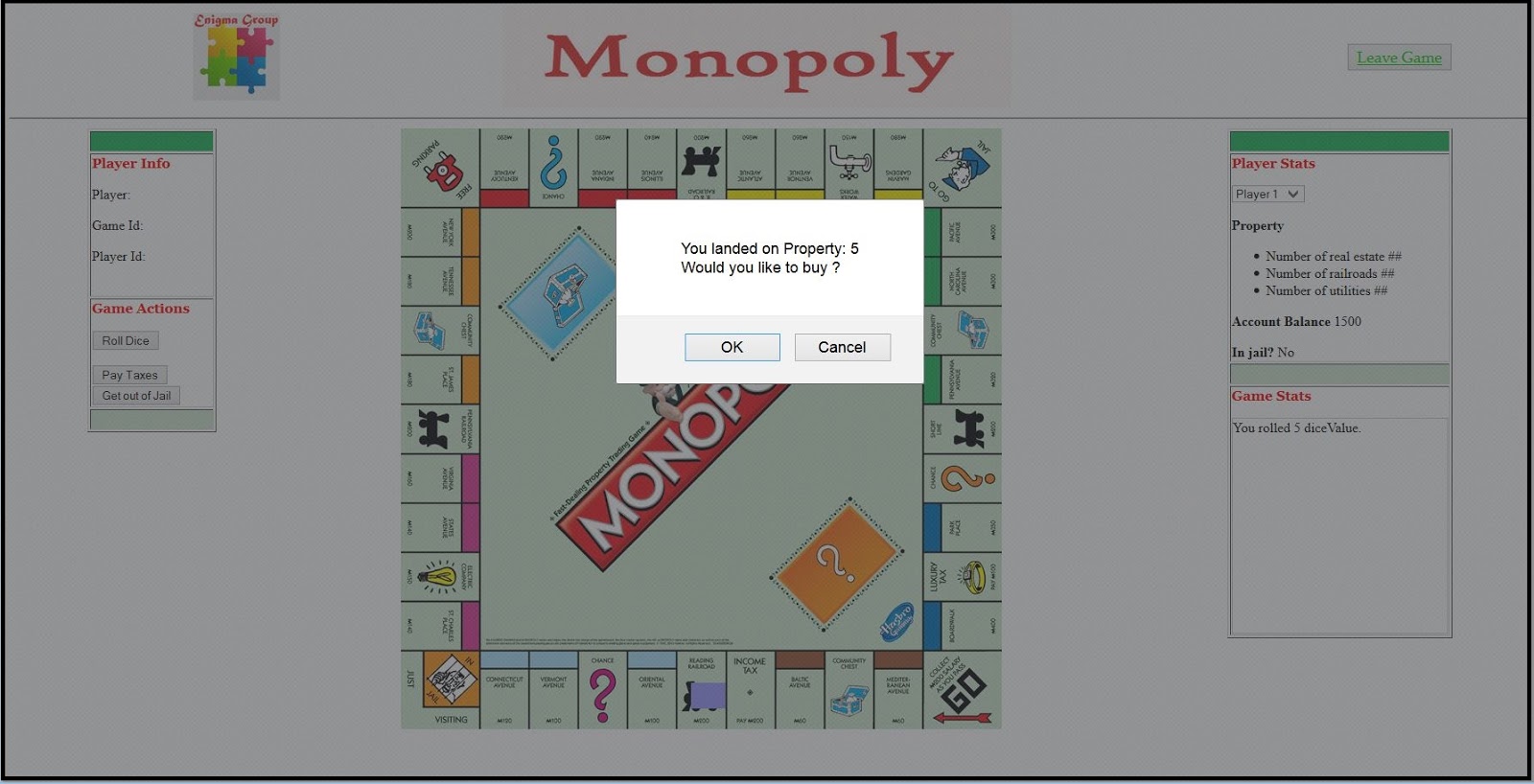
When user rolls the dice and lands in space id 2, which is Community or Chance Card Space, the token is moved to space 2, the dice rolled result and space id “2” is showed in Game Actions section and the action is printed in Game Stats section.

## **Free Parking**



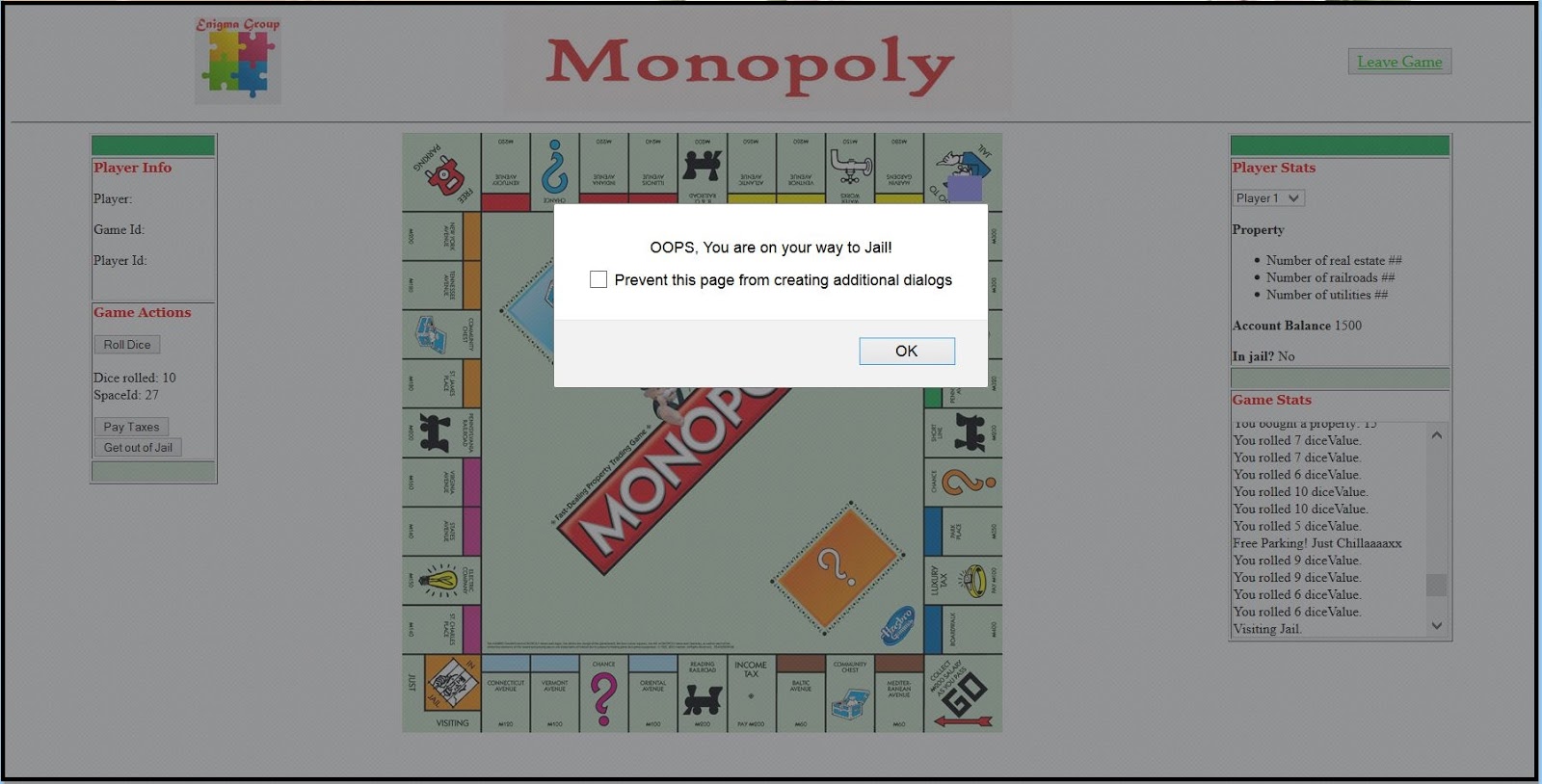
When user rolls the dice and lands in space id 20, which is Free Parking Space, the token is moved to space 20, the dice rolled number and space id “20” is showed in Game Actions section and the action is printed in Game Stats section.

## **Property Space Pop Up Message**



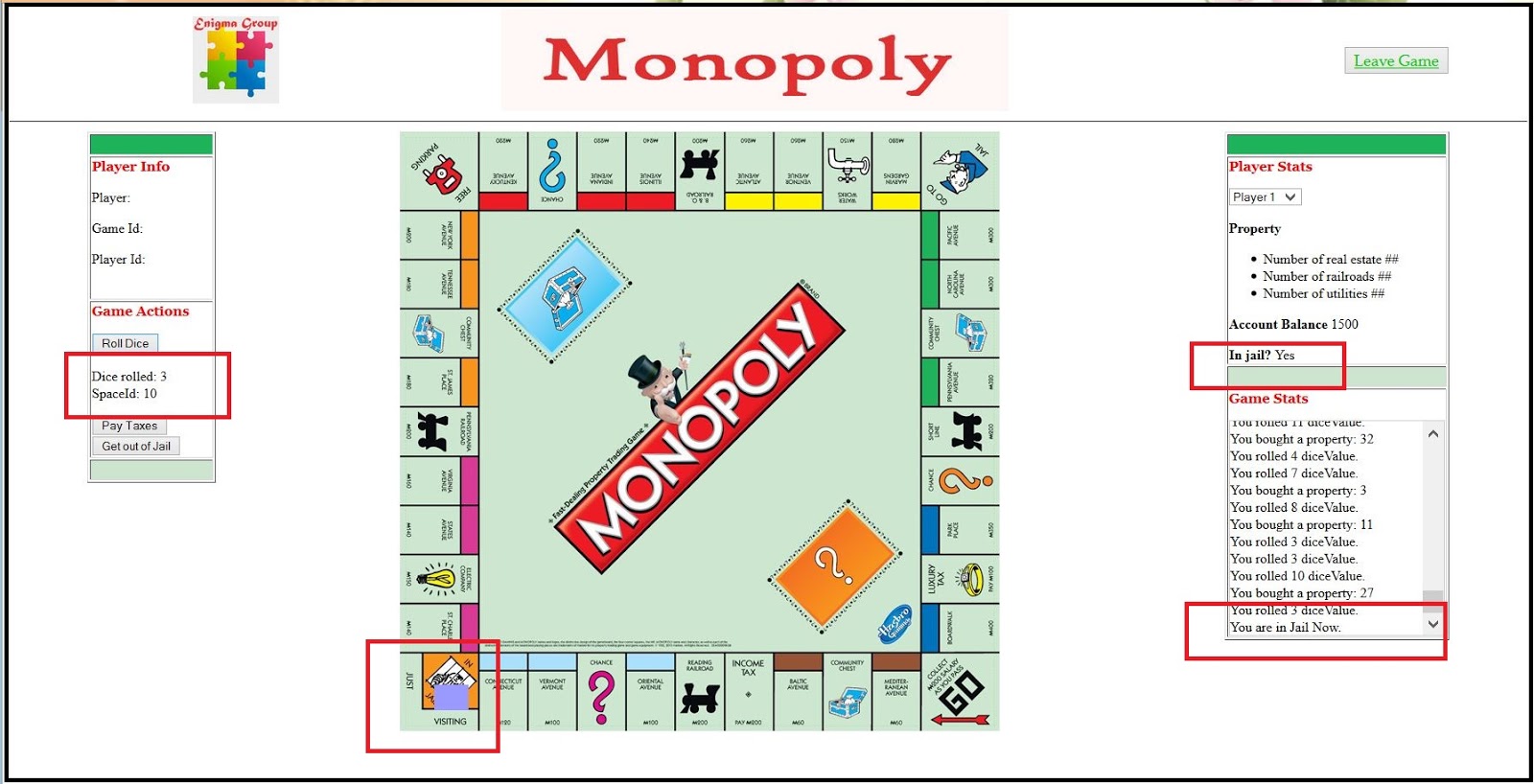
When user rolls the dice and lands in a property space, a pop up message shows asking if the user wants to buy that property on the space.

## **Landing on “Going to Jail” Space**



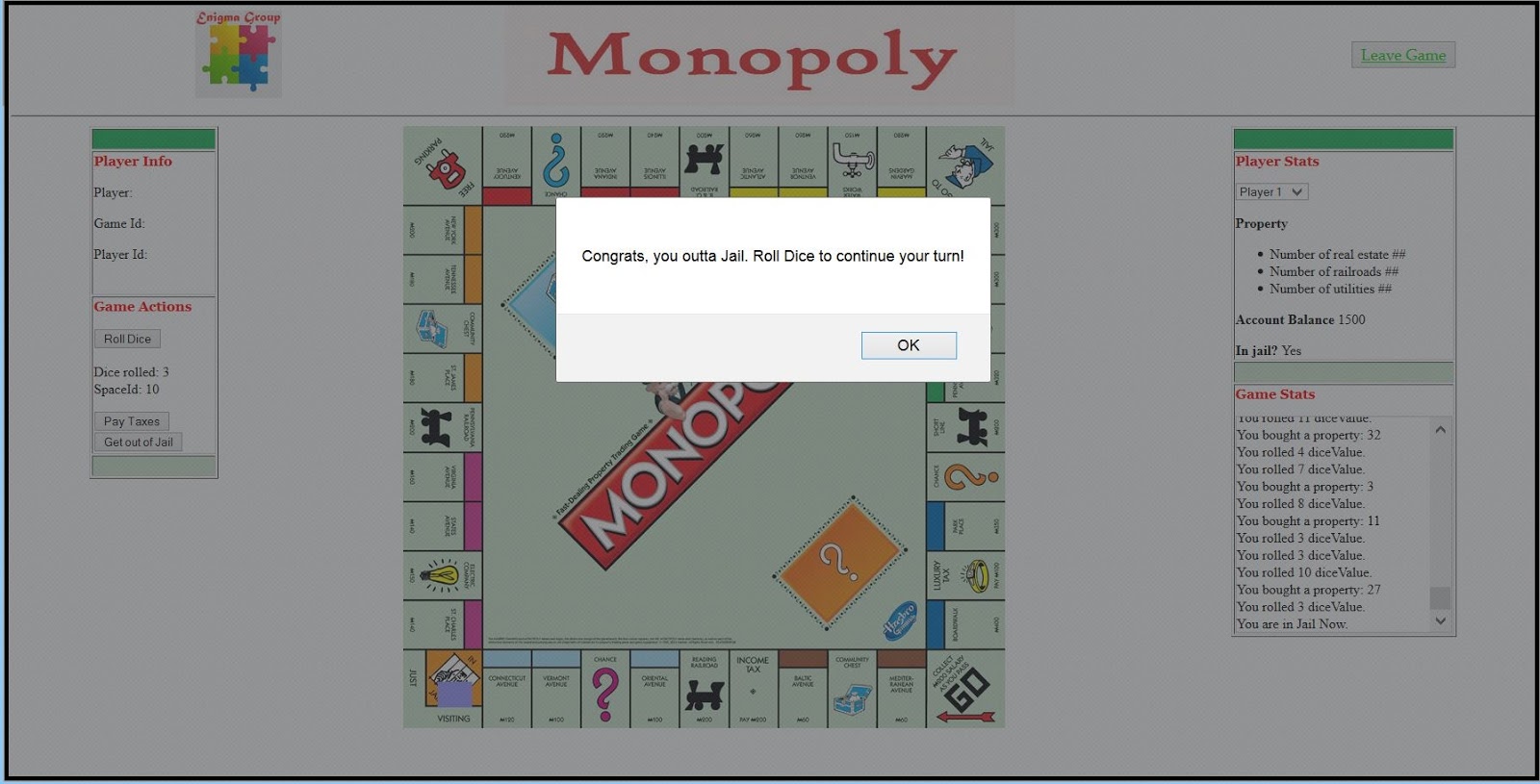
When the user rolls the dice and lands on “Go to Jail” space, a pop up message shows saying, “OOPS, You are on your way to Jail!”

## **User is In Jail**



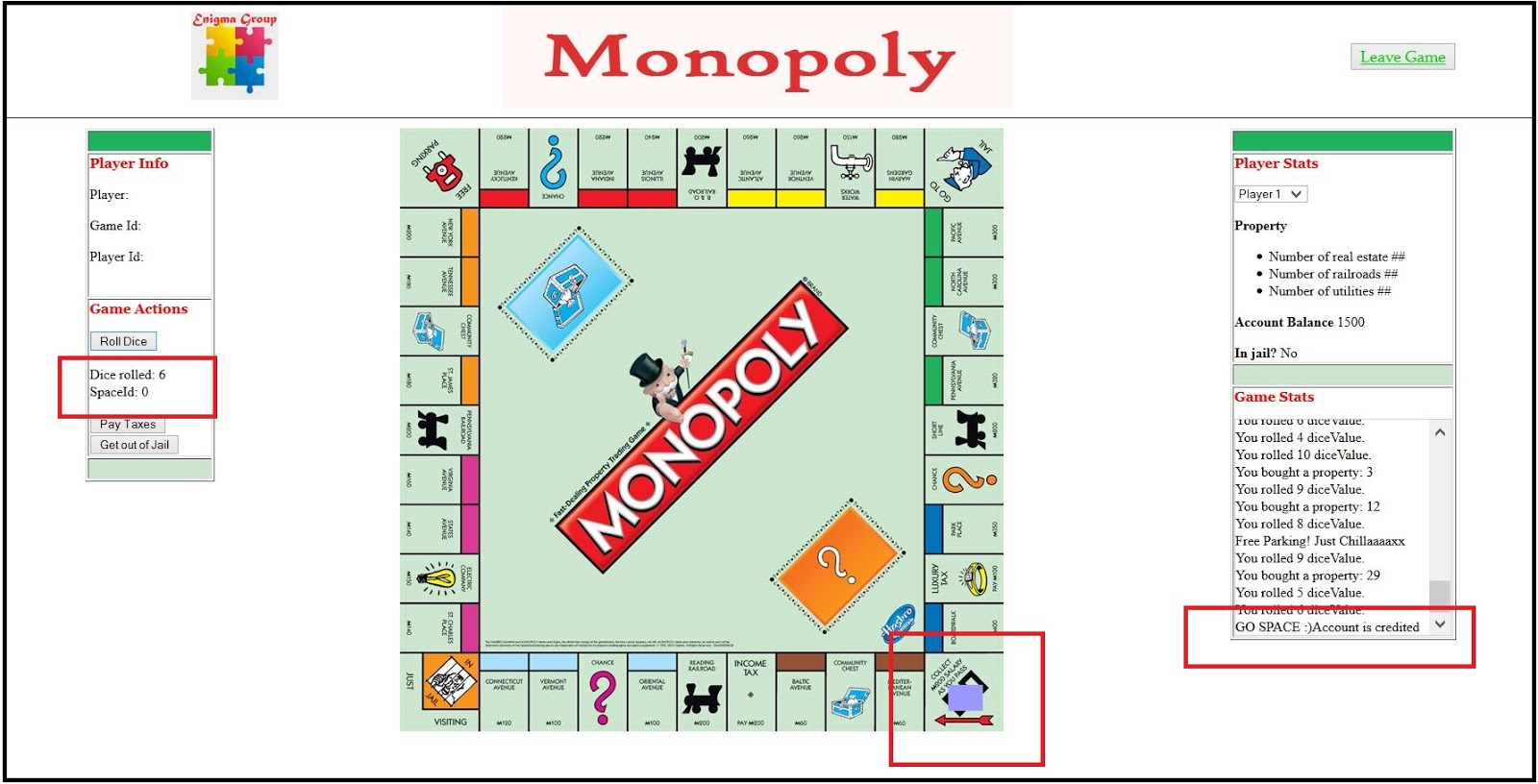
After the user lands in the “Go to Jail” space, the “In jail?” in Player Status section should be set to “Yes”. The token is in space id 10 and the Game Stats message says, “You are in Jail Now.”

## **Get Out of Jail Pop Up Message**



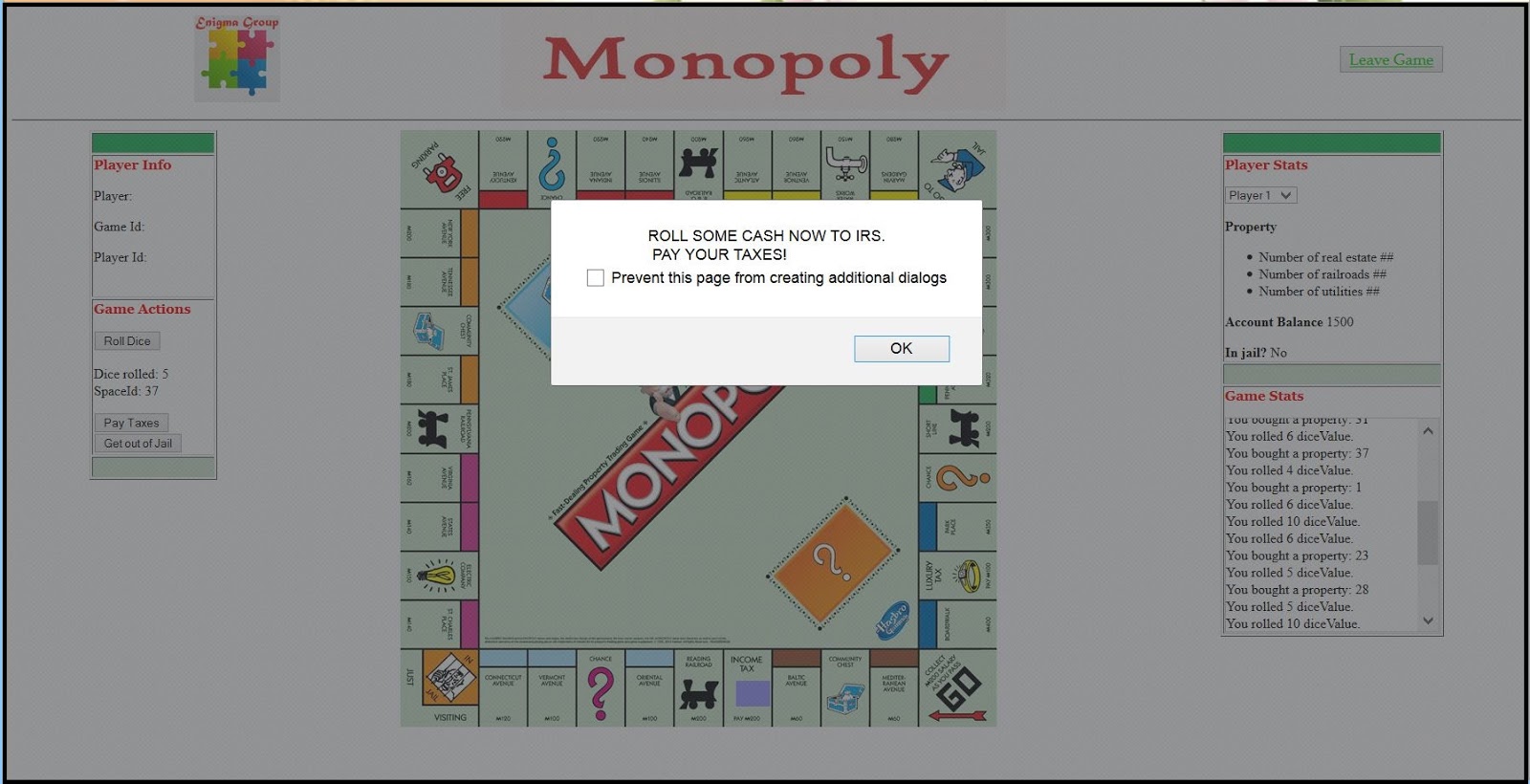
When user is currently in jail, the “In jail?” on the Player Status is set to “Yes”. When user clicks the “Get Out of Jail” button, a pop up message saying, “Congrats, you outta Jail. Roll Dice to continue your turn!” shows.

## **Passing “Go” Space**



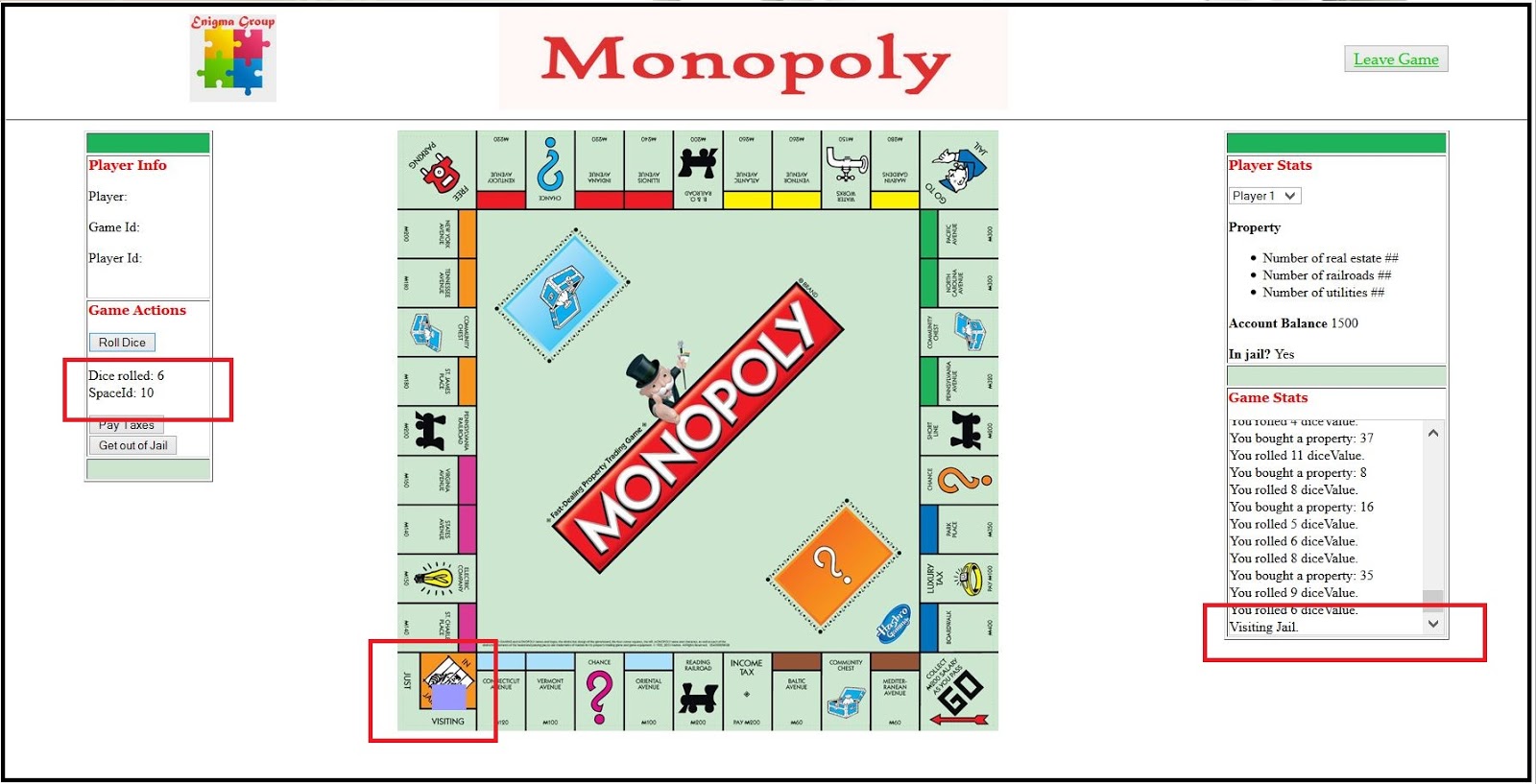
When user passes the “Go” space, user account is credited. User will see the message, “GO SPACE ;)Account is accredited” on the Game Stats section.

## **Pay Tax Pop Up Message**



When user lands in one of the pay tax spaces, user gets a pop up message saying, “ROLL SOME CASH NOW TO IRS. PAY YOUR TAXES!”. When the OK button is clicked the message disappears.

## **User Lands on “Visiting Jail” Space**



When user rolls the dice and lands on the “Visiting Jail” space, the token should be in that space, and the Game Stats says “Visiting Jail”.