Software Engineering Project

Elevation

Project Description

Our large scale software development process will be the development of a software analogue of the board game Elevation. Elevation is a two player board game similar to chess, with the additional element of terrain serving as an active participant and resource. The board itself is a 6x6x3 grid of “terrain pieces”. These pieces can be “dug” and moved to another location, and player pieces can “climb” these to get a height advantage over their opponents, which is necessary to take another piece. Please refer to the appendix for a screen shot of the actual board game.

Rationale for choosing topic

The board game Elevation is similar to chess and checkers, which already have many software emulations, and is suitably different and complex. The requirements necessary for a true emulation include online and offline multiplayer, an artificially intelligent opponent, a complex user interface, and social media interactions. A real-life application of this software would be a popular game in demand of mass production, but due to budget limitations, the company has decided to publish a web-based version to reach out to a global audience instead.

Delegation of Tasks

Mito, Sophie and Matt were tasked to extract the use cases from the requirements specifications. Because Oliver had previous experience of game development, he was tasked to handle the duration and cost estimation part as well as the GUI design of the project. After the use cases have been formed, all members convened to extract the classes for the class and sequence diagrams.

Project Timeline

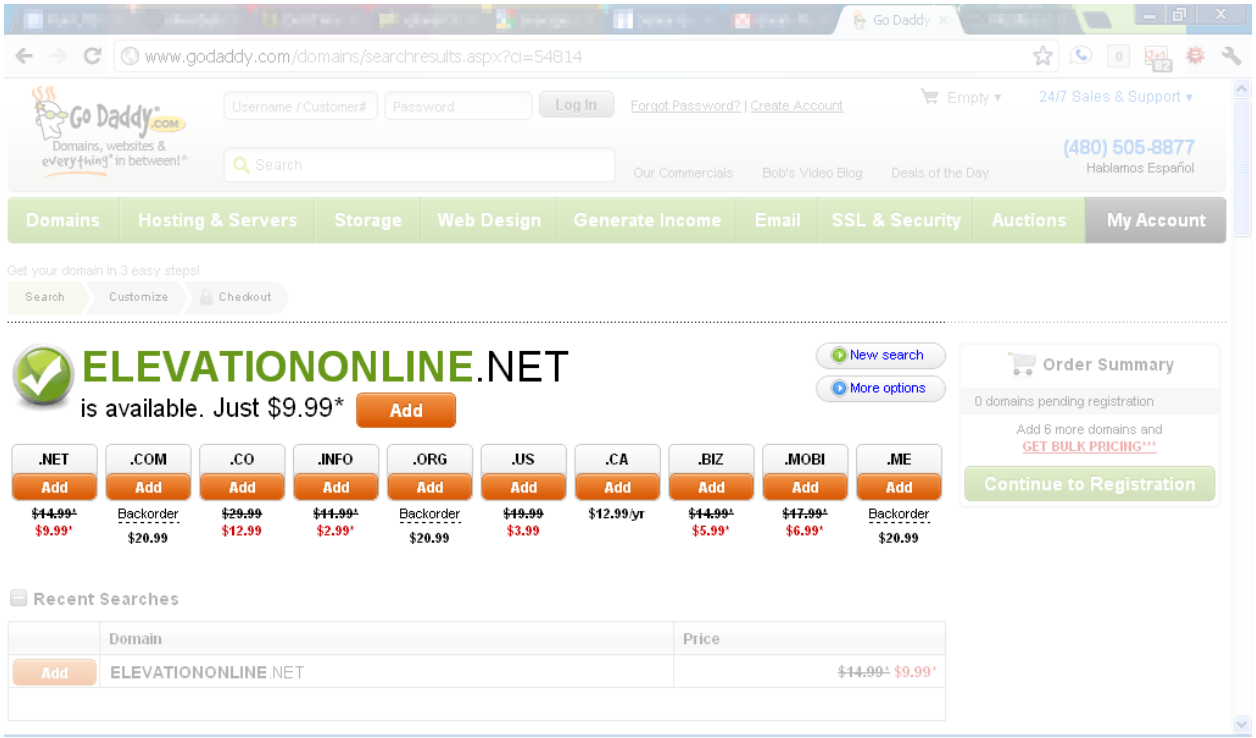
For this project, our team decided to employ the Synchronize-and-Stabilize software development strategy. At each of our meetings, our team looked at the individual progress of each member and performed the necessary refinements for the current parts that we’re working on before compiling everything together.

Duration and Cost Estimation

To create this web based video game, our company will need at least an artist versed in graphic design, designing for the web, GUI, and 3D modeling; and a programmer who is fluent in C++, PHP, JavaScript, HTML5, and CSS3. With only these two employees, we could deliver the product within approximately 6 months of project start [1].

Senior PHP Developers cost about $93,000 a year [2], which would amount to approximately $46,500. The artist would most likely be a Senior Graphics Artist or a Junior 3D Modeler, whose salary would approximate $58,000 per year [3] [4], amounting to approximately $29,000 for the project. Both of these combine to a total employee cost of $75,500.

Depending on the status of the domain name of the website (high demand or low demand) the price could range from a few dollars to thousands of dollars a month. For this project, we got a quote for "ElevationOnline.net" which is sold for $9.99 from GoDaddy.com (a prominent domain registration site).



This would be an ongoing price though. During development it would cost a mere $60, and to keep the site going for longer, say 5 years, would be $600. Which considering the total cost of the site, would only be .8%.

Looking at our example, the Barfight! game has approximately 20,000 lines of code for server and client code, which includes 3d rendering and sound engines, multiplayer support, and websockets interfaces [1].

Requirements Specifications

Web based

To appeal to a wider audience, the game should be made accessible through the internet. This means that multiple users will be able access the game at the same time. Therefore, a database of users' accounts is necessary to store their information, the status of their games and their game statistics. A bug reporting system can also be made available to gather data regarding the software's quality. Furthermore, because the game will be made web-based, it can be easily integrated into social media applications such as Facebook.

Graphical User Interface

The game will need graphical counterparts of its real life tangible components as well as an interface to interact with the game pieces to play the game. This includes a 6x6x3 3D game board with perspective and moving camera and graphical menus. The user interface should be an accurate representation of the original tangible board game both visually and mechanically (game rules). There should also be a way to change the graphics quality of the game to support low-end devices.

Gameplay

The game will be available for player vs. player or player vs. computer modes. This necessitates the creation of an AI system with varying degrees of difficulty. The user will also have a choice to customize some attributes of the game such as the terrain dimensions, number of player pieces and their positions.

Multiplatform

The user should be able to play the game regardless of their device's OS. This is possible because the game will be implemented as a web-based application. The user will only need to have a web browser to access the website and play the game.

Game Mechanics

Basic Movement

The six pieces that each player starts with are two koi, two dragons, and two samurai. All of these pieces have two basic moves that they may perform. Once one player moves their chosen piece, their turn ends and it becomes the next players turn, and so on.

The first of these moves is called *Dig*. A Dig involves picking up a terrain piece from an adjacent (vertically, diagonally, or horizontally) square and moving it to a different adjacent square. It is important to note that a piece may not remove terrain pieces from beneath enemy pieces, however, you may move terrain pieces out from your own pieces if you choose. The exception to this rule is that you may also spend a full turn to dig beneath an opponent, but you cannot do anything else that turn.

The second basic move every piece possesses is *Move*. In this move, the player may choose to move one of their pieces to any adjacent square, provided they do not make any movement violations. Each piece may only use this basic form of movement to move up one terrain level at a time, but a piece may drop down as many terrain levels as needed. If your piece is at least two elevation levels above your opponent's piece, then you may move on top of it and capture that piece. You may not move to the same terrain piece as your opponent unless you are capturing their piece.

Each piece can perform the movements *Dig* and *Move* in any order they would like. It is also allowed for a piece to perform only one of the two basic moves, but this results in the player forfeiting the remainder of their turn. If a player chooses not to move a piece, they may simply end their turn and let the opponent make their move.

Various Game Pieces

There are three different types of game pieces on each side of the board, and each one is unique. The three pieces are The Samurai, The Koi Fish, and The Dragon. Each of these pieces may perform the basic movements described above. In addition to the basic movements, The Koi Fish and The Dragon may swap out one of their basic moves (*Dig or Move)* for an alternate move. All of the normal movement rules still apply (terrain rules, choosing to only make one of the two moves, etc.)

As noted above, The Samurai may only perform the two basic movements whenever they are moved.

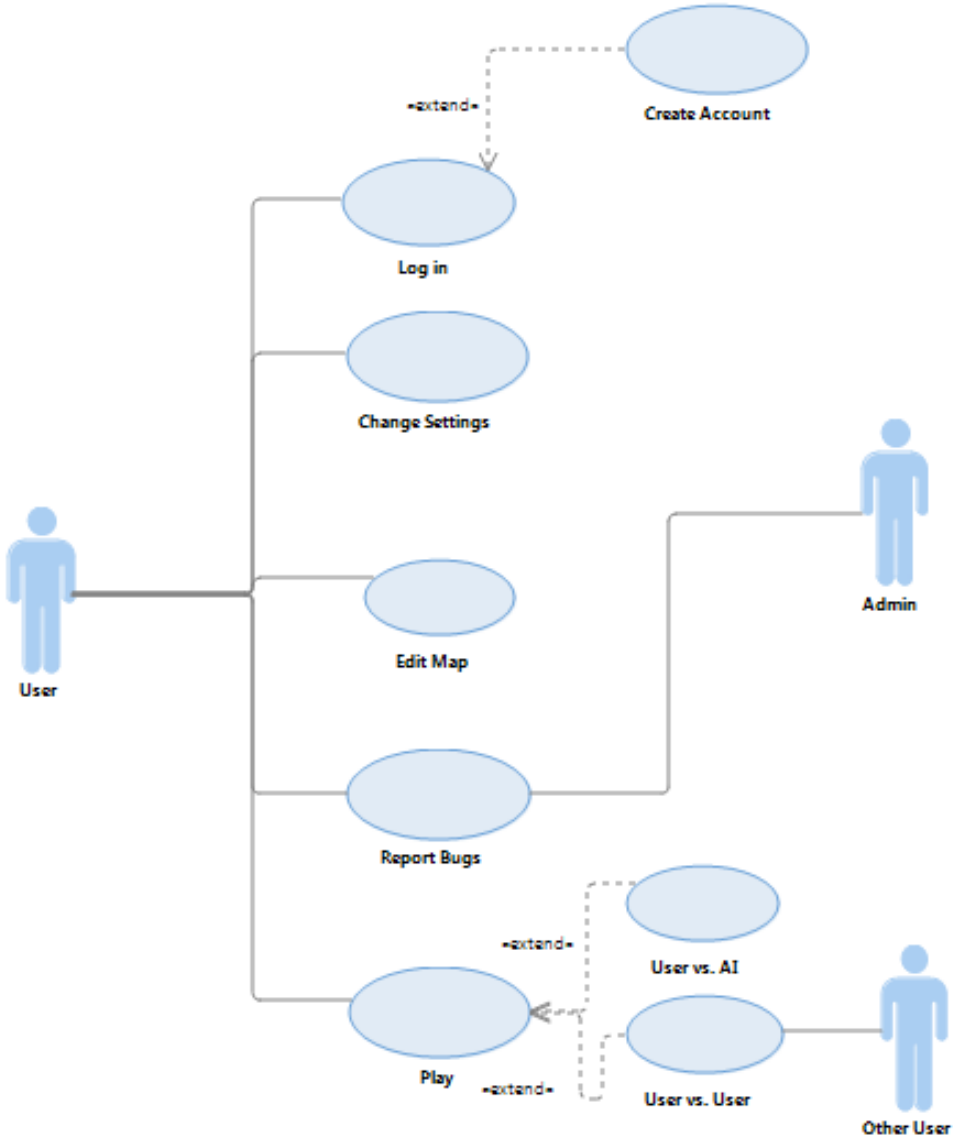
The Koi Fish can simply move according to the normal movement rules, but it may also choose to move two squares instead of one when moving diagonally. All of the normal movement rules still apply.

Like The Koi Fish, The Dragon can also move according to the normal movement rules, but it has the special ability to exchange either its *Dig* or *Move* operation for another of those operations. In other words, The Dragon can exchange its *Move* for an extra *Dig*, or exchange its *Dig* for an extra *Move*. There is also the special case where The Dragon can exchange its *Dig* for an extra step, meaning, it can move up two levels of elevation as opposed to the normal one level.

Ending The Game

The game normally ends when one of the players captures all of their opponent's pieces. However, a stalemate can occur when neither player wishes to move, or the players each "undo" the other's moves three times in a row. Needless to say, when a stalemate occurs, everyone loses.

Use Case Descriptions and Diagrams



Elevation Use Case Diagram

Use Case: Log In

Actors: User

Extends: Create Account

Includes: none

Brief Description: The Log In use case enables the user to access the game's main menu to play the game

Step-by-Step Description:

1. The user accesses the game's website through a web browser.
2. The game server loads the game's webpage onto the user's browser.

3. The user inputs his username and password into the respective fields then clicks the Log In button.
4. The game server authenticates the information inputted by the user.
    4.1. Search the database for the username
        i. if the username is in the database, proceed to Step 4.2
        ii. if the username is not in the database, reload the webpage with the message "This username is not taken yet. Are you trying to create a new account?" Do Step 3 again.
    4.2. Verify that the password typed corresponds to the password for the username in the database
        i. if the passwords match, proceed to Step 5
        ii. if the passwords do not match, reload the webpage with the message "Incorrect password." Do Step 3 again.
5. The game server loads the game's main menu onto the user's browser.

Use Case: Create Account

Actors: User

Extends: none

Includes: none

Brief Description: The Create Account use case enables the user to create a new account to be able to play the game or download a standalone version of the game.

Step-by-Step Description:

1. The user accesses the game's website through a web browser.
2. The game server loads the game's webpage onto the user's browser.
3. The user clicks the Create Account button.
4. The game server loads the Create Account webpage.
5. The user enters his chosen username, email and password into the respective fields.
    5.1. The user confirms his password by reentering it into the "confirm password" field.
6. The user clicks Create Account button.
7. The game server checks the validity of the users inputs (username and password).
    7.1. Check if username already exists in the database.
        i. if it does, reload the webpage with the user name field highlighted and the message "Username already taken."
        ii. if it is unique, proceed to Step 7.3.
    7.2. Check if Password field and Confirm Password fields match
        i. if it does, proceed to Step 8.
        ii. if it does not, reload the webpage with the Confirm Password field highlighted and the message "Passwords do not match."
8. Save the user's information in the game server database.
9. Send an email containing his new account information (username, password).
10. Game server loads the game's main menu.

Use Case: Change Settings

Actors: User

Extends: none

Includes: none

Brief Description: The Change Settings use case enables the user to change aspects of the game such as its sound and graphics quality.

Step-by-Step Description:

1. The user clicks on Change Settings button on the main menu.
2. The game server loads the Change Settings webpage (Initial Settings: Sound = ON; Graphics Quality = HIGH).
3. User changes Sound status (clicks on ON or OFF radio button).
4. User changes the Graphics Quality (clicks on LOW, MED, or HIGH radio button).
5. User clicks the Save button.
6. Game system applies new settings
    - 6.1. Apply new sound settings.
        - i. user clicked ON
            1. if the sound is already on, do nothing.
            2. if the sound is off, turn it on.
        - ii. user clicked OFF
            1. if the sound is already off, do nothing.
            2. if the sound is on, mute it.
    - 6.2. Apply new graphics quality settings
        - i. user clicked LOW
            1. if the quality is already low, do nothing.
            2. if not, set it to low.
        - ii. user clicked MED
            1. if the quality is already medium, do nothing.
            2. if not, set it to medium.
        - iii. user clicked HIGH
            1. if the quality is already high, do nothing.
            2. if not, set it to high.
7. Save new game settings.
8. Game server loads main menu.

Use Case: Map Editor

1. The Play clicks the “Map Editor” button from the game’s main menu page.
2. The game server loads the map editor initial menu onto the player’s browser.
    - 2.1. The player clicks the “New Map” button.
        - i. The game server sends the default settings to step 3. Proceed to step 3.
    - 2.2. The player clicks the “Load Map” button.
        - i. The game server accesses the map(s) in the player’s database entry and loads them to the player’s browser.

1. If there are no maps associated with the user, the game server displays an error message. Jump back to step 2.

ii. The player clicks their desired map.

iii. The game server accesses the corresponding data in the user's database entry.

iv. The Game Server sends this data to step 3. Proceed to step 3.

3. The game server loads the map editor main menu to the player's browser. The settings are set equal to the settings obtained in steps 2 or 3.

3.1. The player clicks the "New Map" button.

i. The game server sends the default settings to step 3. Proceed to step 3.

3.2. The player clicks the "Load Map" button.

i. The game server accesses the map(s) in the player's database entry and loads them to the player's browser.

1. If there are no maps associated with the user, the game server displays an error message. Jump back to step 2.

ii. The player clicks their desired map.

iii. The game server accesses the corresponding data in the user's database entry.

iv. The Game Server sends this data to step 3. Jump to step 3.

3.3. The player clicks the "Save Map" button.

i. The game server allows the user to input their desired map name.

ii. The game server uploads the map name and settings to the player's database entry.

iii. The game server displays a message confirming that the save was successful.

iv. Jump back to step 3.

3.4. The player clicks the "Exit" button.

i. The Game Server loads the game's main menu onto the player's browser.

3.5. The player clicks the "Modify Starting Units" button.

i. The game server loads the unit modification interface onto the player's web browser.

ii. The player inputs their desired settings into the corresponding input fields.

iii. The game server sends these settings to step 3. Jump to step 3.

3.6. The player clicks the "Modify Terrain" button.

i. The game server loads the terrain modification interface onto the player's browser.

ii. The player inputs their desired settings into the corresponding input fields.

iii. The game server sends these settings to step 3. Jump to step 3.

Use Case: Report Bugs

Actors: Player

Extends: none

Includes: none

Brief Description: The Report Bugs use case allows the player to report any errors they may come across while playing the game. The report they file is uploaded to a database along with the user's system properties.

Step-by-Step Description:

1. The Player clicks the “Report Bugs” button from the game’s main menu page.
2. The Game Server loads the bug reporting form to the player’s browser.
3. The Player inputs the description of the error they encountered and the browser they were using when the error occurred and clicks the “Submit” button.
    3.1. If the “description” field is blank, jump back to step 2.
    3.2. If the player failed to select their browser, jump back to step 2. 4.

The game server uploads the error report to the appropriate database. 5.

The game server loads the game’s main menu onto the player’s browser.

Use Case: Play

Actors: User

Extends: Player vs. Player, Player vs. AI

Includes: none

Brief description: The Play user case allows the player to select with mode s/he wishes to play in: Player vs. Player or Player vs. AI.

Step-by-Step description:

1. User selects “Play” from the title screen.
2. The screen changes to display three options: “Player vs. Player”, “Player vs. AI”, and “Back”.
    2.1. The user selects “Player vs. Player.”
        i. The game enters Player vs. Player mode.
    2.2. The user selects “Player vs. AI.”
        i. The game enters Player vs AI mode.
    2.3. The user selects “Back.”
        i. The game switches back to the title screen.

Use Case: User vs. User

Actors: User

Extends: none

Includes: none

Brief description: The Player vs. Player use case allows the player to play a game against another user that is currently online.

Step-by-Step description:

1. The server searches for online players.
    1.1. The server cannot find any other players currently online.
        i. The server will conduct another search every five seconds.
    1.2. Other online players are found. Proceed to Step 2.
2. A list of the usernames of currently online players is displayed on the screen, along with the buttons “Request match” and “Back”.

2.1. The list is updated by the server to reflect any changes in online or offline players every five seconds.
    i. The user selects “Back.”
        1. The screen changes back to the Play menu.
    ii. The user waits until another player requests a match, where a pop-up window appears.
        1. The user rejects in the request.
            a. The request is denied and the pop-up window disappears.
        2. The user accepts the request. Proceed to Step 3.
    iii. The user selects the name of another player in the list and clicks “Request match.”
        1. The other player rejects the request.
        2. The other player accepts the request. Proceed to Step 3.
        3. The user selects “cancel.”

3. The server loads the game start screen.
4. The two players are randomly assigned to each color, black or white.
5. The white player moves first.
6. The black player moves next.

(REFER TO GAMEPLAY DESCRIPTION)

7. Alternate until a condition to take a piece is met.
    7.1. The user takes an opponent player piece.
    7.2. The opponent player takes a user piece.
8. Repeat Step 5-7 until a win condition is met.
    8.1. The user wins.
        i. Player vs. Player statistics is changed.
    8.2. The opponent player wins.
        i. Player vs. Player statistics is changed.
9. The screen displays the options “Play again”, “Back to title screen”, and “Exit.”
    9.1. The user selects “Play again”.
    9.2. The user selects “Back to title screen.”
        i. The screen changes back to the title screen.
    9.3. The user selects “Exit.”
        i. The game exits.

Use Case: Player vs. AI

Actors: User

Extends: none

Includes: none

Brief description: The Player vs. AI use case allows the player to play a game against an AI, with varying degrees of difficulty to choose from.

Step-by-Step description:

1. The user is given the options “New Game”, “Continue Game”, and “Back”.

1.1. The user selects “Continue game”.
    i. The screen displays the user’s saved game files.
        1. The user selects a save file.
            a. Proceed to Step 2.
        2. The user selects “Back.”
            a. The screen switches back to the previous one.
1.2. The user selects “New Game.”
    i. The screen will show the difficulties “Easy,” “Normal,” “Hard,” and “Impossible,” along with the colors “Black” and “White” to choose from.
1.3. The user selects “Back.”
    i. The screen switches back to the title menu.
2. The game starts.
3. The white player moves first.
    3.1. If the game is loaded from a save file, the player moves first.
4. The black player moves next.

(REFER TO GAMEPLAY DESCRIPTION)

5. Alternate until a condition to take a piece is met.
    5.1. The user takes an opponent player piece.
    5.2. The opponent player takes a user piece.
6. The user can choose to save the game when it is his/her turn.
    6.1. If a save slot is available, the game will save to the first available save slot.
    6.2. If all save slots are full, the user can select which save slot to overwrite, or cancel.
7. Repeat Step 3-5 until a win condition is met.
    7.1. The user wins.
        i. Player vs. AI statistics is changed.
    7.2. The opponent player wins.
        i. Player vs. AI statistics is changed.
8. The screen displays the options “Play again”, “Back to title screen”, and “Exit.”
    8.1. The user selects “Play again”.
    8.2. The user selects “Back to title screen.”
        i. The screen changes back to the title screen.
    8.3. The user selects “Exit.”
        i. The game exits.

Gameplay Descriptions

User vs. User

1. Generate board
2. White player goes 1st
3. Choose and play piece
    3.1. Get board state
        i. locations of each piece and player pieces eliminated

3.2. SAMURAI:

i. Menu pops up

ii. CASE 1: (no adjacent enemy pieces present)

1. MOVE

a. Get board state b.

Pop up goes away

c. Highlight valid adj spaces (1 space around the piece)

d. Player clicks a space

i. If highlighted = move to that space

ii. If not = do nothing

e. Update board state

f. DIG

i. Same as CASE 1 DIG BELOW

ii. End player's turn (turn over control to other player)

g. END TURN

i. Pop up goes away

ii. End player's turn (turn over control to other player)

2. DIG

a. Get board state b.

Pop up goes away

c. Highlight valid adj spaces (1 space around the piece)

d. Player clicks a space

i. If highlighted = take 1 tile level from that space and transfer to adj space

1. If highlighted = place tile there, increase that space's elevation

2. If not = do nothing

ii. If not = do nothing

e. Update board state

f. MOVE

i. Same as CASE1 MOVE above

ii. End player's turn

g. END TURN

i. Same as CASE 1 END TURN ABOVE

3. BACK

a. Pop up goes away

b. Player can choose another piece to play

iii. CASE 2: (adjacent enemy pieces present)

1. MOVE

a. Same as CASE1

2. DIG

a. Same as CASE 1

b. CASE 1 + ATTACK
    i. See below
3. DIG FROM UNDER
    a. Pop up goes away
    b. Validate adjacent enemy pieces present (highlight)
    c. Click an enemy piece
        i. If highlighted = chosen Enemy piece's elevation is reduced by 1
        ii. If not = do nothing
    d. Highlight valid spaces to put tile taken
    e. Click a space
        i. If highlighted = Chosen space's elevation is increased by 1
        ii. If not = do nothing
    f. Update board state
    g. Turn is over
4. ATTACK
    a. Pop up goes away
    b. Validate adjacent enemy pieces present (highlight)
        i. If the adj enemy piece elevation is 2 or more levels lower than player piece = highlight
    c. Click and enemy piece
        i. If highlighted = enemy piece is eliminated from the board
        ii. If not = do nothing
    d. Update board state
    e. Pop up menu
        i. DIG
            1. Same as CASE 1 dig or CASE 2 dig
        ii. END TURN
            1. Same as CASE 1
5. BACK
    a. Pop up goes away
    b. Player can choose another piece to play

3.3. KOI:

i. Menu pops up
ii. CASE 1: (no adjacent enemy pieces present) - same as SAMURAI CASE 1 but 1.c. MOVE - “Highlight valid adj spaces (1 space around the piece + 2 spaces in diagonal direction from the piece)
iii. CASE 2: (adjacent enemy pieces present) - same as SAMURAI CASE 2

3.4. DRAGON:

i. Menu pops up

ii. CASE 1: (no adjacent enemy pieces present)
    1. MOVE
        a. Get board state b.
        Pop up goes away
        c. Highlight valid adj spaces (1 space around the piece)
        d. Player clicks a space
            i. If highlighted = move to that space
            ii. If not = do nothing
        e. Update board state
        f. DIG
            i. Same as CASE 1 DIG BELOW
            ii. End player's turn (turn over control to other player)
        g. MOVE
            i. Same as CASE1 MOVE above
            ii. End player's turn
        h. STEP
            i. get board state
            ii. pop up goes away
            iii. highlight valid adj spaces (spaces 2 levels higher than player piece)
            iv. Player clicks a space
                1. If highlighted = move to that space
                2. If not = do nothing
            v. Update board state
        i. END TURN
            i. Pop up goes away
            ii. End player's turn (turn over control to other player)
    2. DIG
        a. Get board state b.
        Pop up goes away
        c. Highlight valid adj spaces (1 space around the piece)
        d. Player clicks a space
            i. If highlighted = take 1 tile level from that space and transfer to adj space
                1. If highlighted = place tile there, increase that space's elevation
                2. If not = do nothing
            ii. If not = do nothing
        e. Update board state
        f. MOVE
            i. Same as CASE1 MOVE above
            ii. End player's turn

g. DIG
    i. Same as CASE 1 DIG ABOVE
    ii. End player's turn (turn over control to other player)
h. END TURN
    i. Same as CASE 1 END TURN ABOVE

3. BACK
    a. Pop up goes away
    b. Player can choose another piece to play

iii. CASE 2: (adjacent enemy pieces present)

1. MOVE
    a. Same as CASE1
2. DIG
    a. Same as CASE 1 b.

    CASE 1 + ATTACK
        i. See below
3. DIG FROM UNDER
    a. Pop up goes away
    b. Validate adjacent enemy pieces present (highlight)
    c. Click an enemy piece
        i. If highlighted = chosen Enemy piece's elevation is reduced by 1
        ii. If not = do nothing
    d. Highlight valid spaces to put tile taken
    e. Click a space
        i. If highlighted = Chosen space's elevation is increased by 1
        ii. If not = do nothing
    f. Update board state
    g. Turn is over
4. ATTACK
    a. Pop up goes away
    b. Validate adjacent enemy pieces present (highlight)
        i. If the adj enemy piece elevation is 2 or more levels lower than player piece = highlight
    c. Click and enemy piece
        i. If highlighted = enemy piece is eliminated from the board
        ii. If not = do nothing
    d. Update board state
    e. Pop up menu
        i. DIG
            1. Same as CASE 1 dig or CASE 2 dig

ii. END TURN

1. Same as CASE 1

5. BACK

a. Pop up goes away

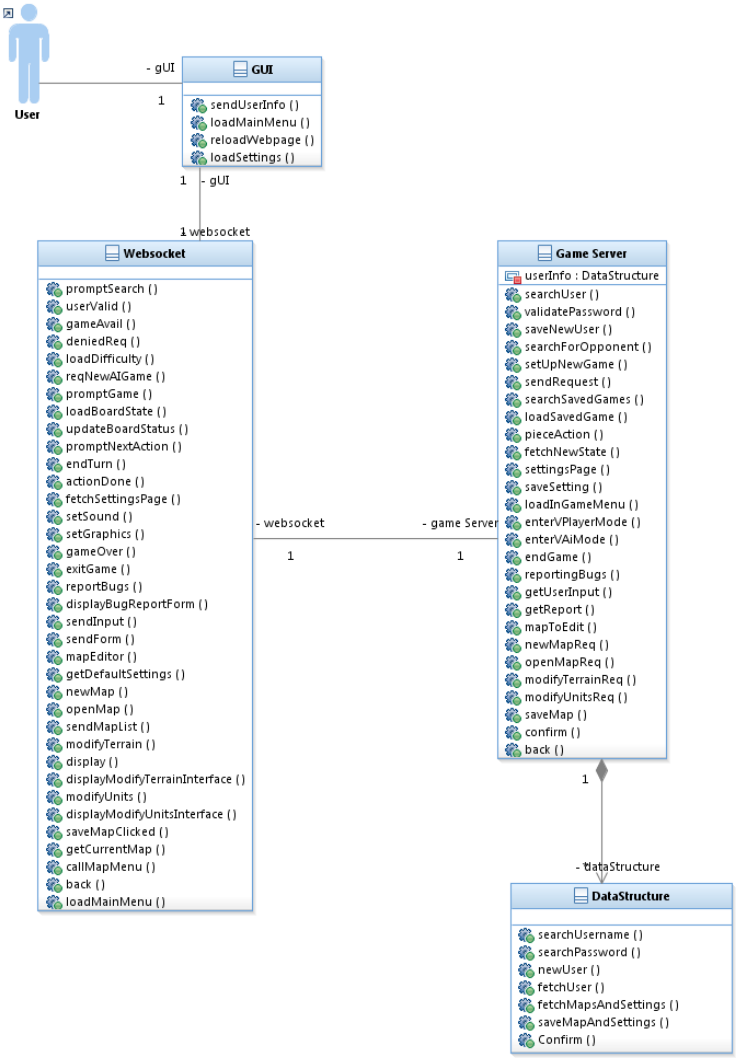
b. Player can choose another piece to play

4. Repeat process until all player pieces of a user is eliminated.

User vs. AI

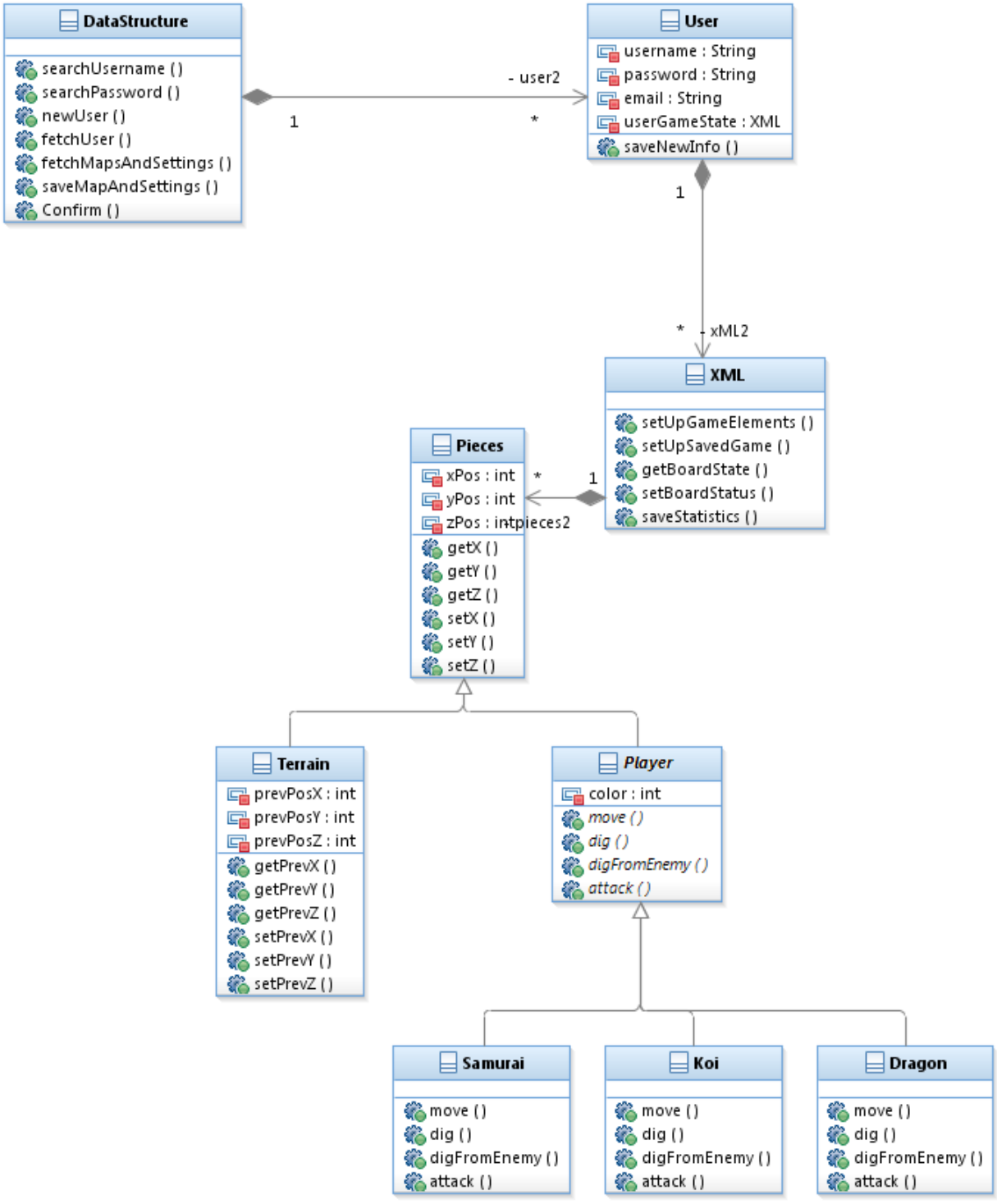
The process described above for User vs. User gameplay is parallel to the User vs. AI gameplay. The only difference is the AI class will implement the actions while doing an analysis over the current state of the game and the possible movements it should make to try to defeat the user.

Class Diagrams



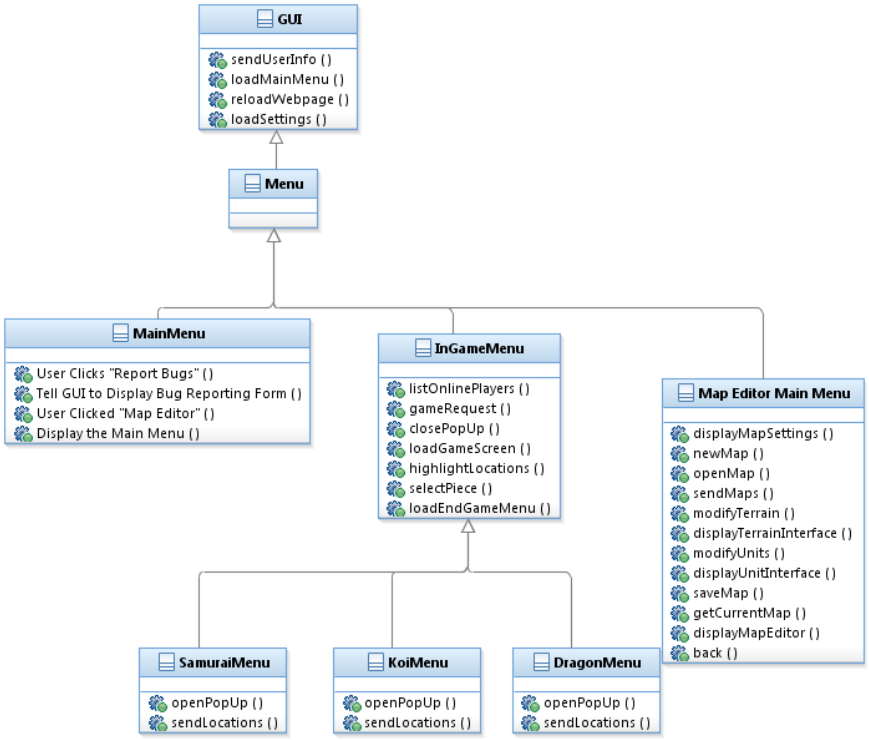
Top level Class Diagram

The GUI class interacts directly with the user. The Websocket class acts as a controller class that relays information between the user's web browser and the game server. The game server is composed of a database of users and this database is itself a type of data structure.



DataStructure Class Diagram

The DataStructure class will contain a set of User objects. Each User object will contain the user's account information (username, password, email address, game statistics) as well as save points and the game status information. The game status will be recorded in an XML file that every user object will have. The game status pertains to the location of each of the games 120 pieces (108 terrain pieces and 12 player pieces). The pieces themselves can either be a terrain piece or a player piece. There are 3 types of player pieces in the game: Samurai, Koi, Dragon.



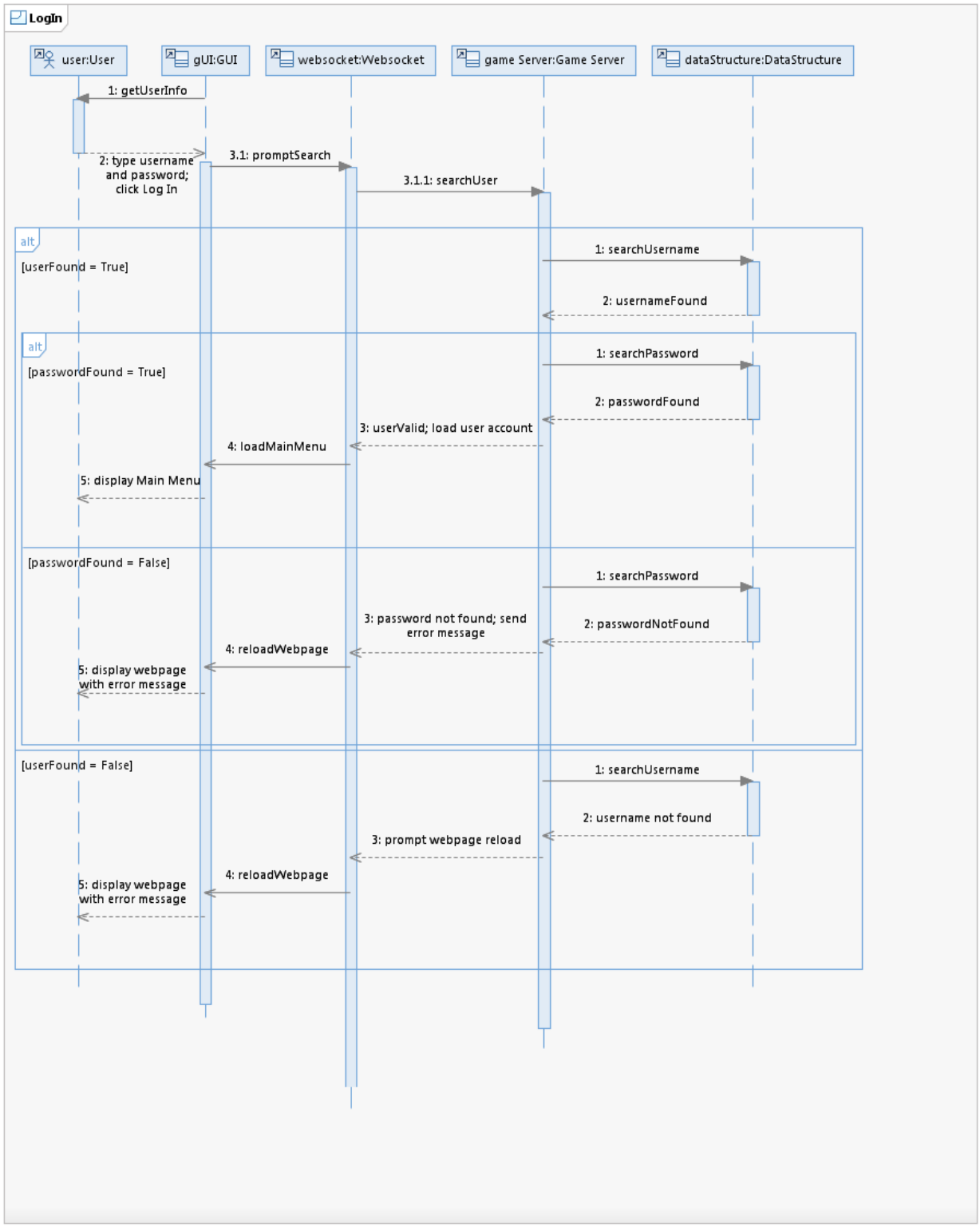
GUI Class Diagram

The GUI class extends all the user interface classes. From the main menu to the actual game play, the application will be highly dependent on the GUI. The InGameMenu class extends the menus for the samurai, koi and dragon pieces. During the game, each whenever the player plays a piece, only the valid spaces will be highlighted and a menu pops up with the different actions the player can choose from. This design was chosen to avoid excessive error checking (see details in the Sequence Diagram or the Use Case section).

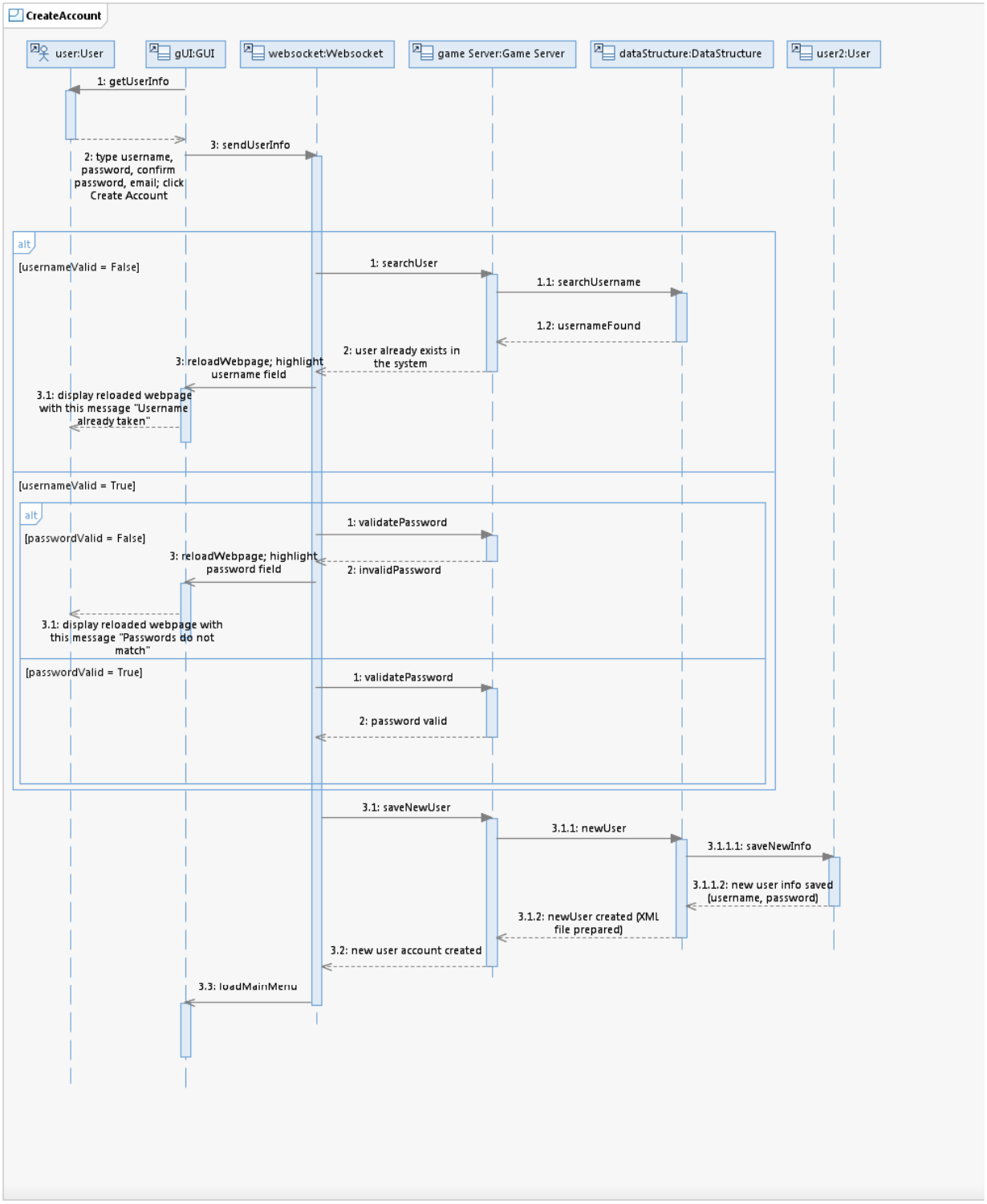
Sequence Diagrams

The sequence diagrams below correspond to the process of each scenario as described in the use case descriptions provided earlier. (To view the diagrams in better quality, please refer to the published UML design in the ElevationWeb folder.)

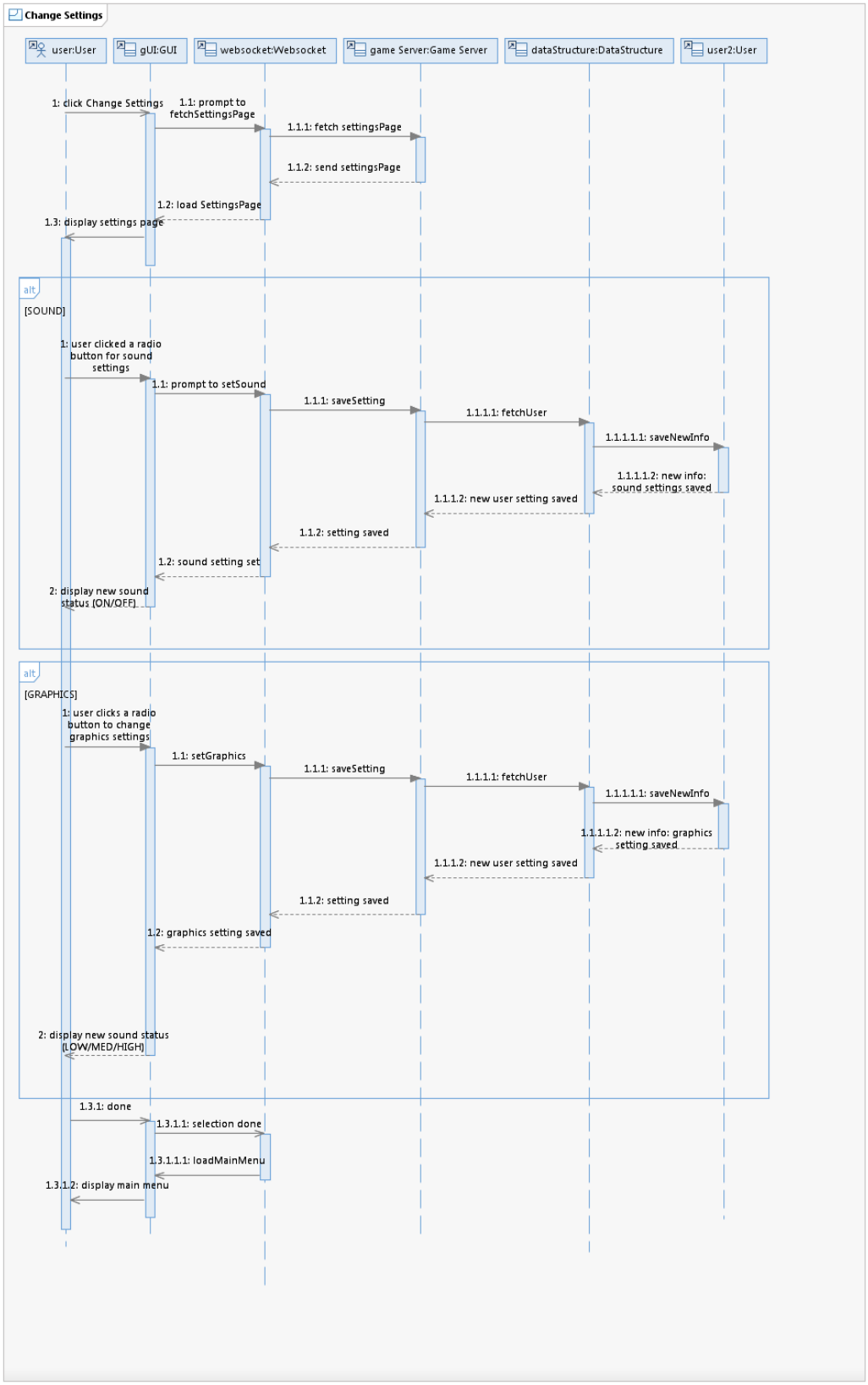
Use Case: Log In



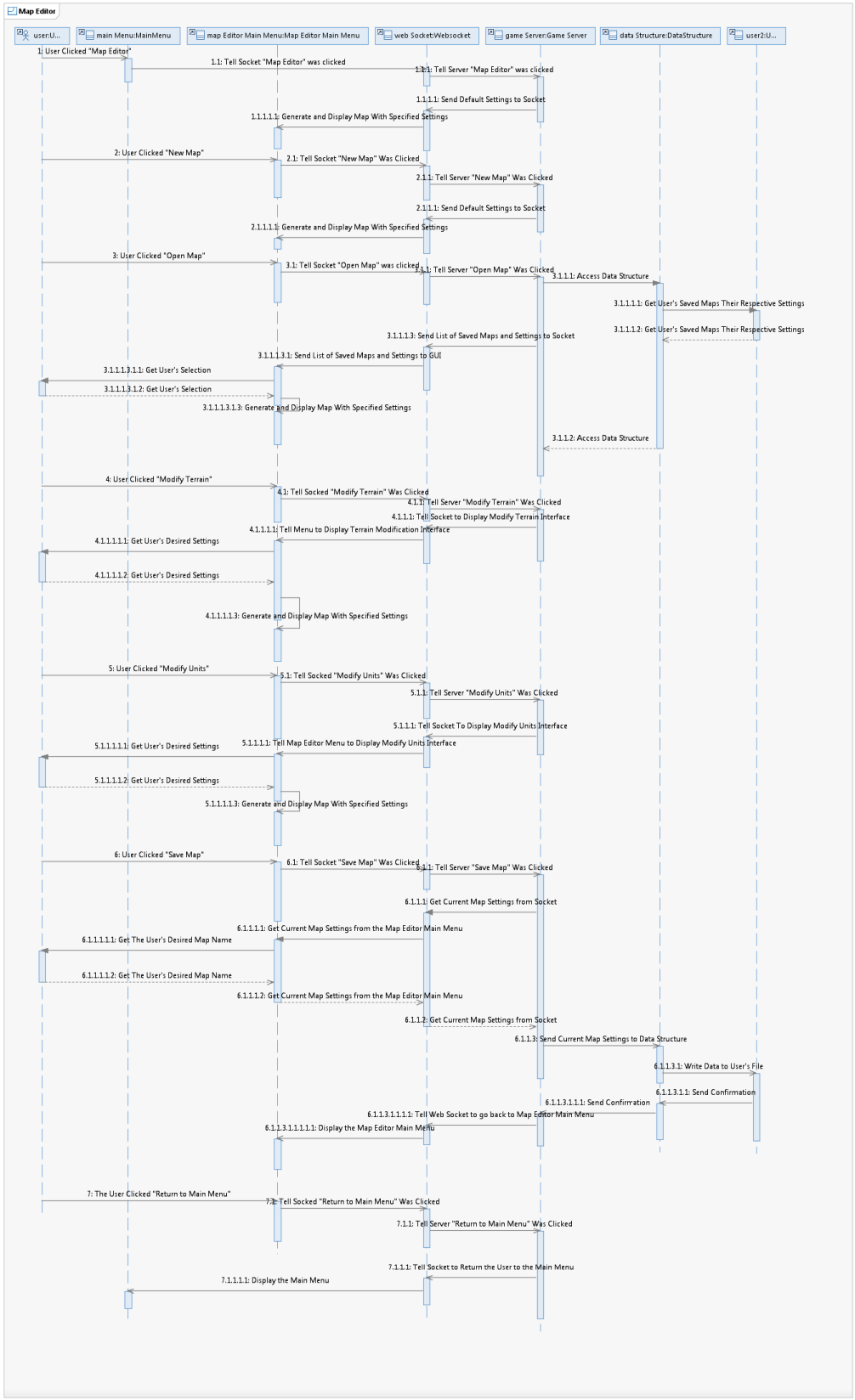
Use Case: Create Account



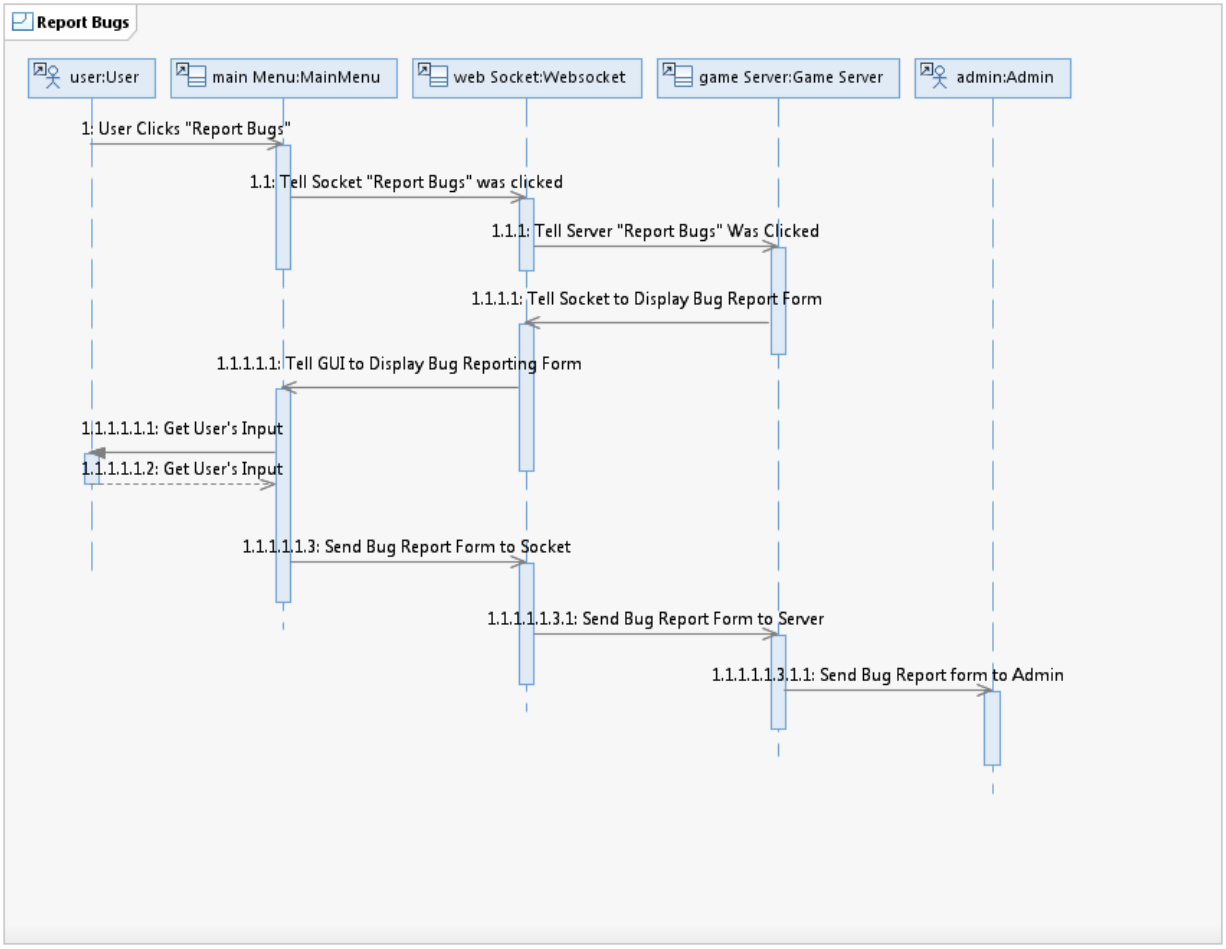
Use Case: Change Settings



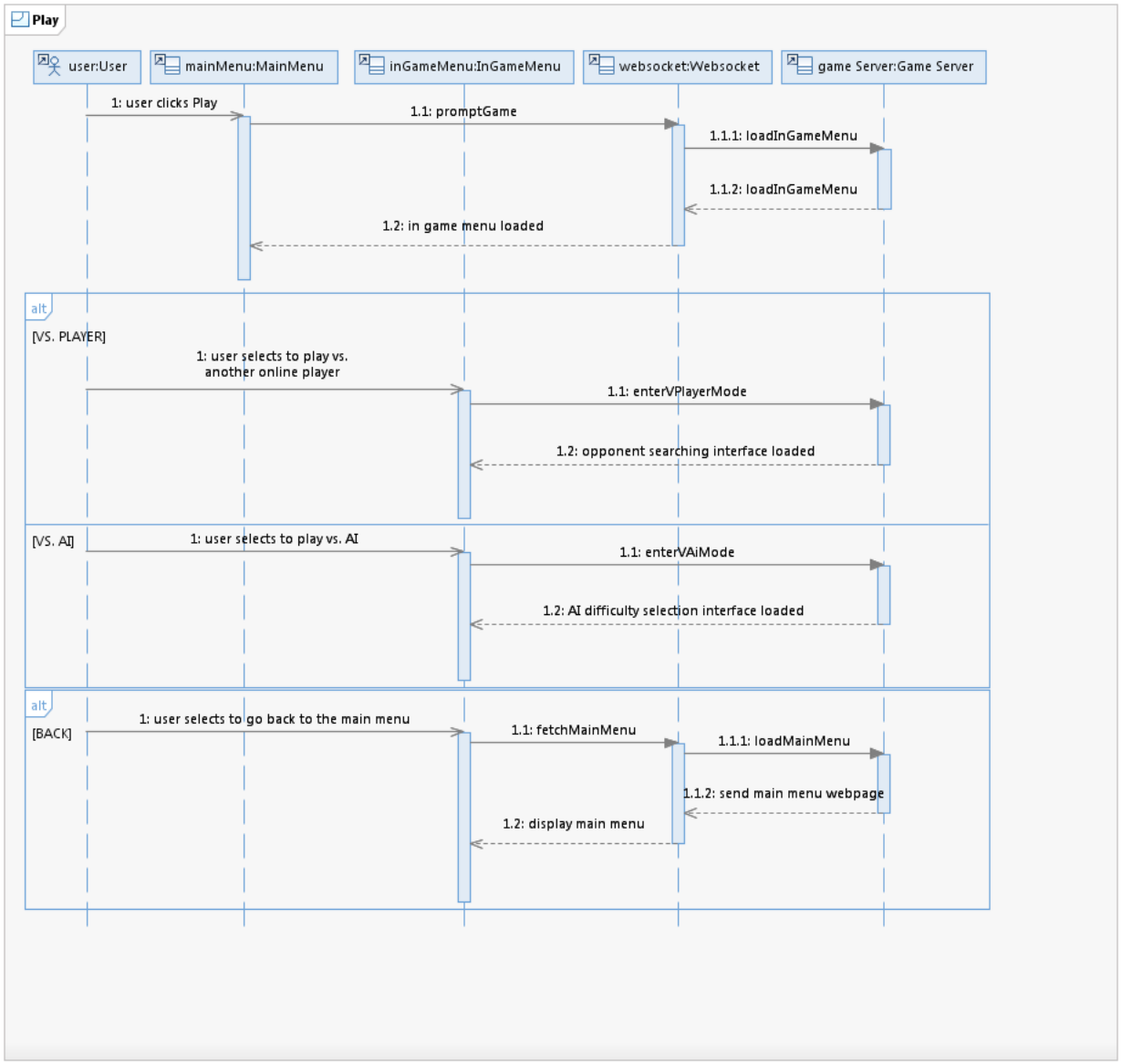
Use Case: Map Editor



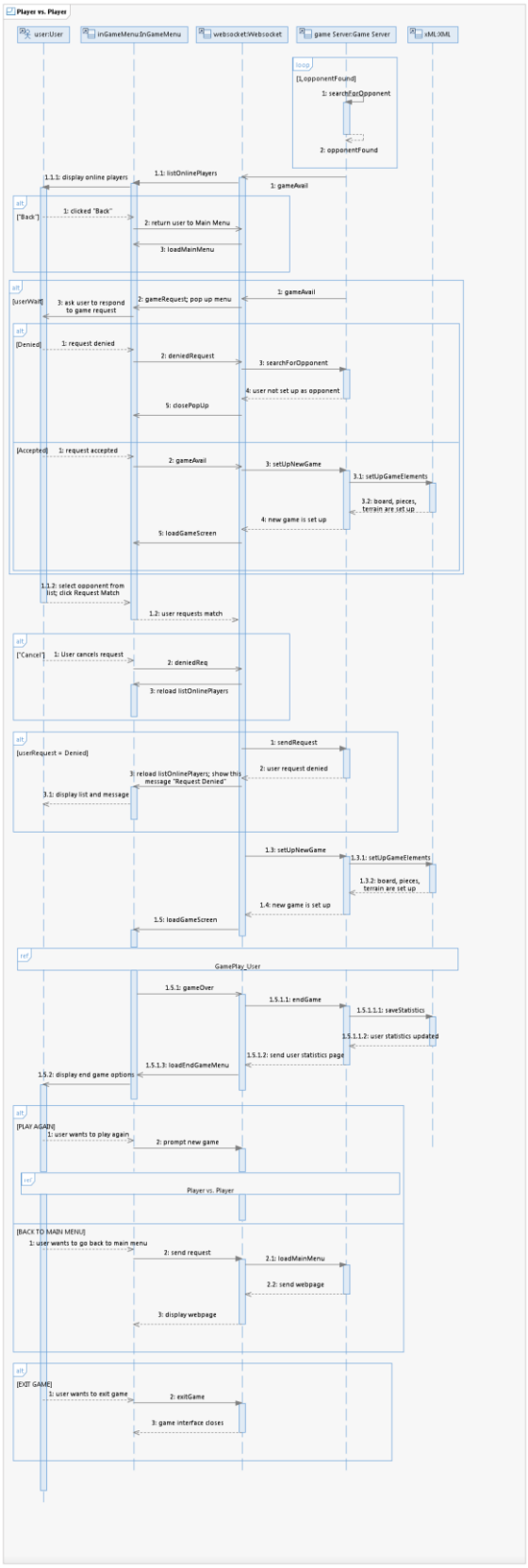
Use Case: Report Bugs



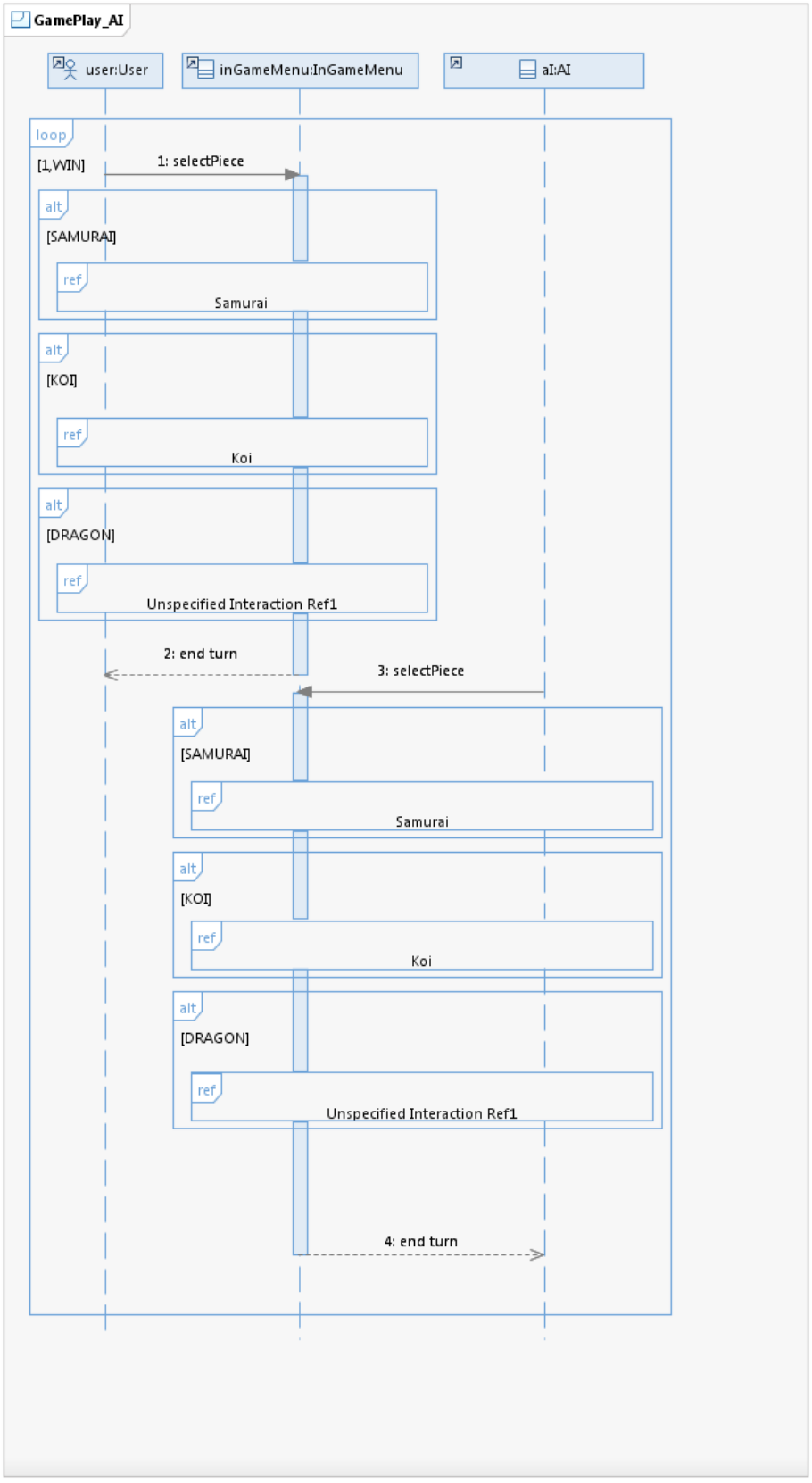
Use Case: Play



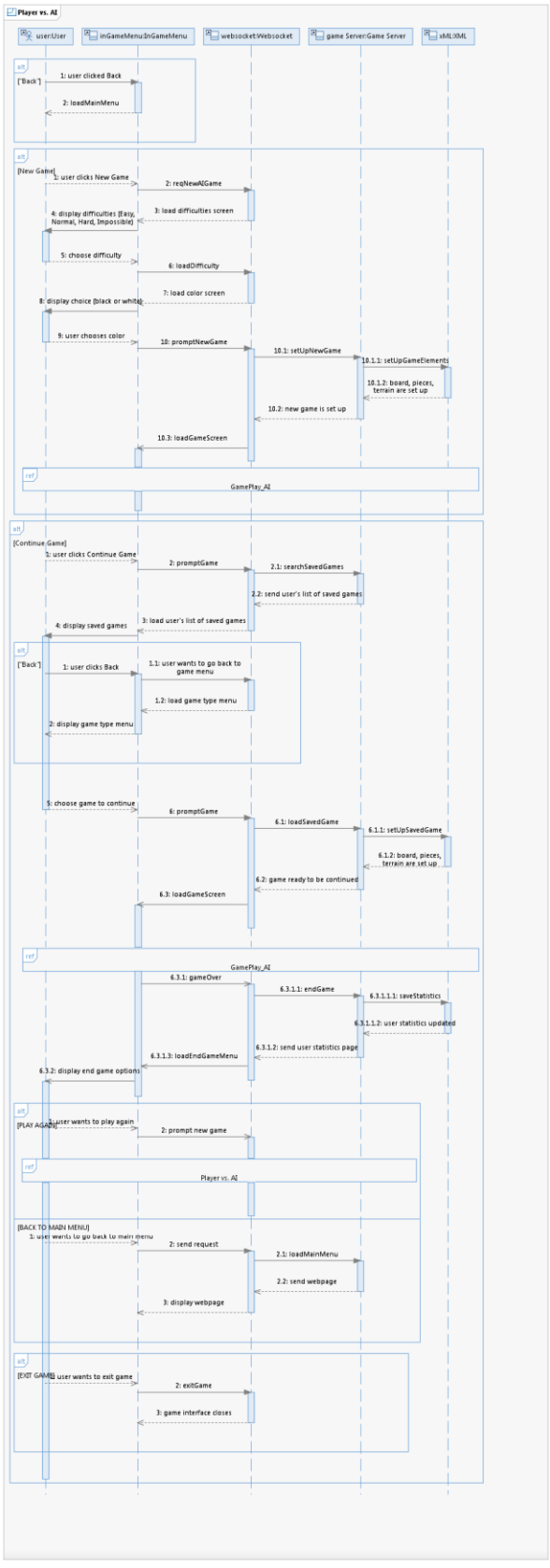
Use Case: User vs. User



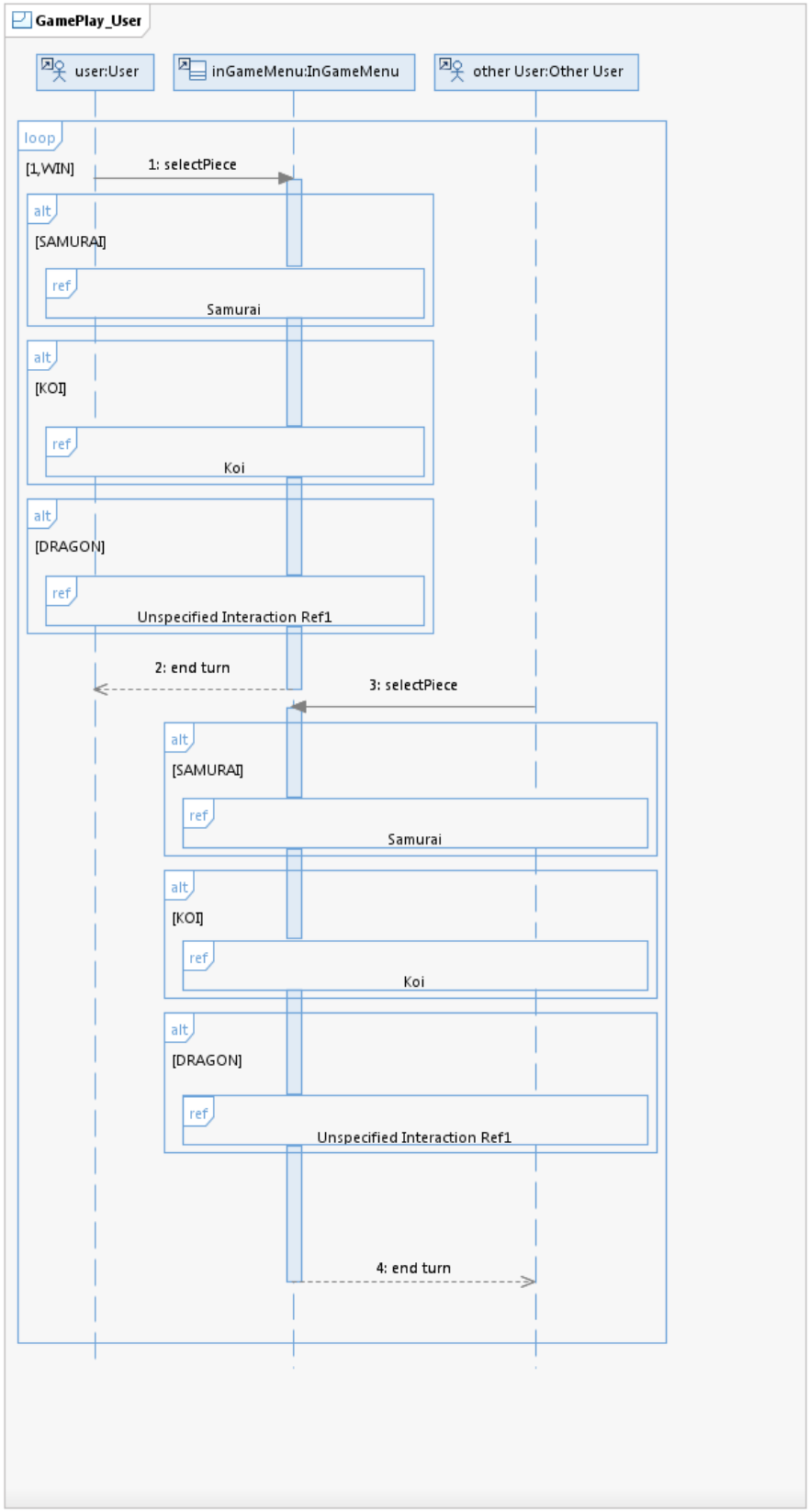
Gameplay_AI diagram



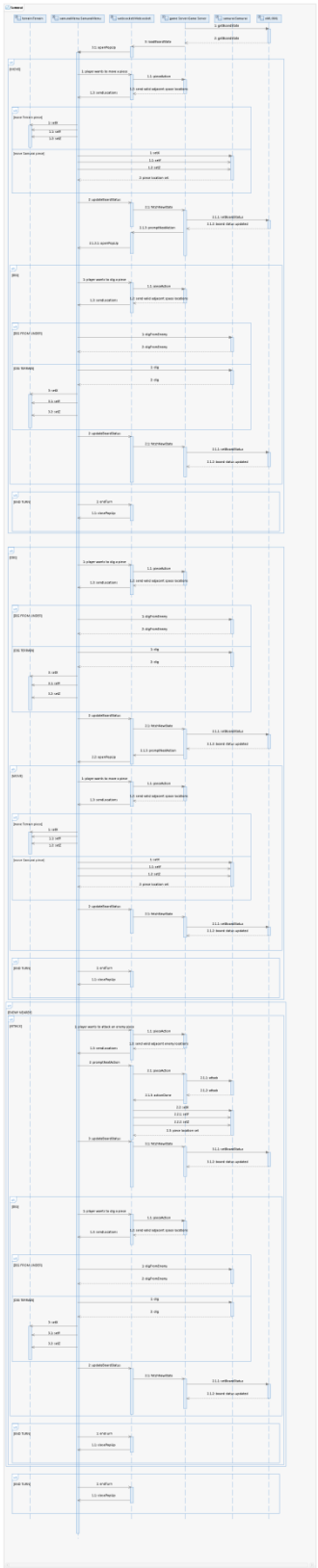
Use Case: User vs. AI



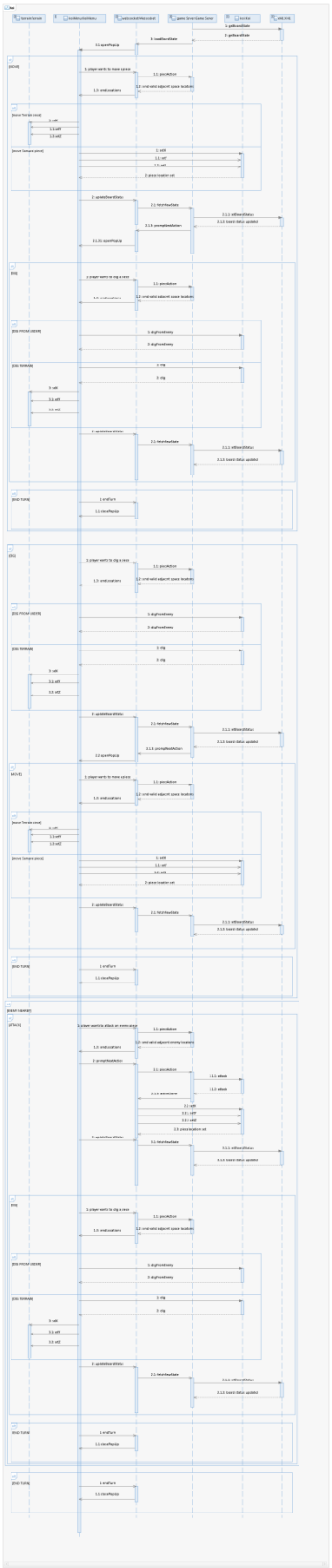
Gameplay_User diagram



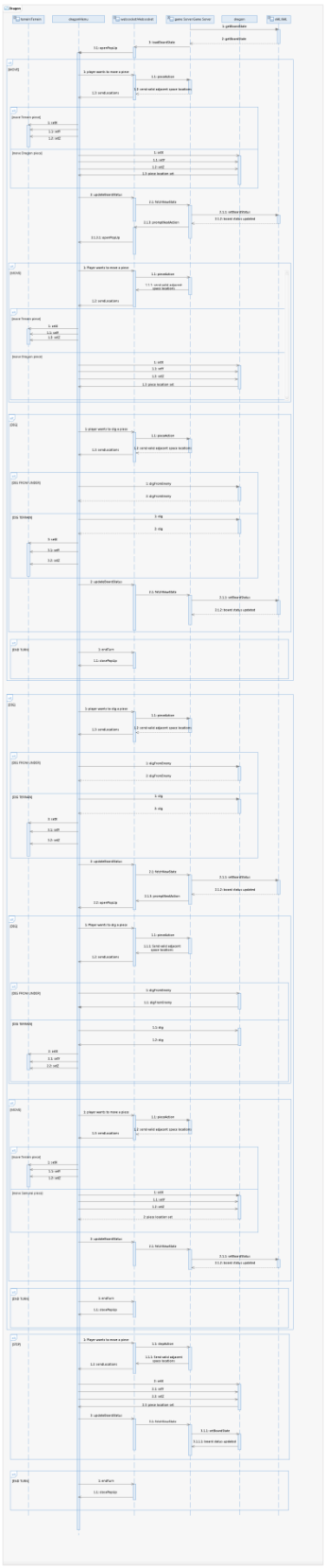
Samurai diagram



Koi diagram



Dragon diagram



Published Design

As per the project specifications, the design for the software emulation of the board game Elevation has been made available as web documentation for professional reference. Please refer to ElevationWeb¥index.html to access the web documentation.

Java Skeleton Code

The basic Java code has been generated to enable the programmers for this project to start the actual implementation of the design. Please refer to the ElevationJava folder to access the java files for this project.

GUI Design

This is the most recent design for the game's graphical user interface. The screen layouts for each of the use cases will generally be the same. We are basing our design closely to the theme of the actual board game.

Conclusion

Our initial goal was to complete a project which required the necessary steps to analyze and design a sufficiently complex software scheme. Our team achieved this goal, but our specification changed several times throughout the project. We had initially decided that the software should be playable as a standalone application apart from being accessible through a web browser. The web-based version and

the standalone version were to be identical, except the standalone version was to include an option to allow the user to design and play their own custom board configuration. At our following meeting, we decided that it was much more feasible to simply make the application playable through a web browser. We still wanted to allow the user to create their own board configuration, so we had to make the required changes to our specification to allow for the user to do this through the web-based version.

Appendix

Elevation Board Game

References

[1] Julie Choi, Steve Baker, Oliver Baker. “Game On Spotlight: Bar Fight” Internet: http://mozillalabs.com/gaming/2011/02/07/bar-fight/, Feb. 7th 2011 [April 4th 2012].

[2] Indeed Search Engine. “Senior PHP Developer Salary” Internet: http://www.indeed.com/salary?q1=Senior+PHP+Developer&l1=, April 6th 2012 [April 6th 2012].

[3] Indeed Search Engine. “Senior Graphic Artist Salary” Internet: http://www.indeed.com/salary?q1=Senior+Graphic+Artist&l1=, April 6th 2012 [April 6th 2012].

[4] Indeed Search Engine. “Senior Graphic Artist Salary” Internet: http://www.indeed.com/salary?q1=3D+Modeler&l1=, April 6th 2012 [April 6th 2012].