Sellswords and Spellcraft Design Document

# GROUP MEMBERS:

Luke Schnetlage

Mason Evans

Nicolas Towery

# REVISED REQUIREMENTS:

## Home Page

* 1. When first loading webpage, user will arrive at home page
  2. Home page should contain the following
     1. A header containing an image of the logo
     2. Subheader text containing team member names
     3. A brief description of how to navigate the website that states
        1. What you will find on rules page
        2. What you will find on game page
     4. A description of why the project was created
     5. A brief description of what the game is
     6. Navigation menu

## Navigation Menu

* 1. Navigation menu should be present on all web pages including
     1. Home
     2. Game
     3. Rules
  2. Navigation menu should include
     1. A link labeled “Home”
        1. On click, will redirect to home page
     2. A link labeled “Game”
        1. If user is logged in, on click, will redirect to game page
        2. If user is not logged in, redirect to login page
     3. A button labeled “Rules”
        1. On click, will redirect to home page
     4. If user is not logged in, display a button labeled “Log In”
        1. On click, will redirect to login page
     5. If user is logged in, display a button labeled “Log Off”
        1. On click, display the following
           1. A message asking user if they wish to commit
           2. A button labeled “Yes”

On click, do the following

Log user out

Redirect to home page

* + - * 1. A button labeled “No”

On click, wipe message from screen

## Login page

* 1. The login page must include the following
     1. A textbox to enter a username
     2. A label for the username textbox
     3. A textbox to enter a password
     4. A label for the password textbox
     5. A “login” button
        1. If user is already registered and credentials match, send user to home page
        2. If user is not already registered or credentials do not match, display a relevant message
     6. A link labeled “Register New User”
        1. On click, will redirect to register page

## Register Page

* 1. The register page should include the following
     1. A textbox to enter a username
     2. A label for the username
     3. A textbox to enter a confirm username
     4. A label for the confirm username
     5. A textbox to enter a password
     6. A label for the password
     7. A label for the confirm password
     8. A textbox for the confirm password
     9. A “register” button
        1. If the username is already stored, display a relevant message
        2. If the username is not already stored
           1. Login user with credentials entered
           2. Redirect to home page

## Rules page

* 1. Rules page should contain the following
     1. A header containing an image of the logo
     2. A text block containing the rules and how to play the game
     3. Navigation menu

## Game Page

* 1. The game page must contain the following
     1. A header containing an image of the logo
     2. A window that can be in three of the following states:
        1. Pre game is the starting window
        2. Current game follows the pre-game window
        3. Post game follows the current game window
           1. Can loop back to pre-game
     3. Navigation menu
  2. Alerts user asking for confirmation when they attempt to do the following
     1. Close browser
     2. Close tab
     3. Leave web page

## Pre game window

* 1. Default state of game window
  2. Once user loads into game page, they may select one of the following options
     1. A button titled “Join Session”
        1. Once clicked, screen will change to the following:
           1. A label for list titled “Open Sessions”
           2. A list of joinable sessions

List items include the following

A label for the player name hosting the session

A “Join” button

On click, user will attempt to join session

If successful, will enter “current game” state

If unsuccessful, display error message

Session list should be updated dynamically without refreshing

A button to create a session

On click, enter 5.6.2

* + 1. A button titled “Create session”
       1. Once clicked, screen will change to the following
          1. A header titled “Current Session”
          2. Display user’s display name back to them
          3. Text explaining the game is waiting for another player
          4. A “back” button

On click, bring user back to “Open Sessions” screen (5.6.1)

* + - * 1. Upon another player entering the session.

If successful, enter “current game” state

If unsuccessful, display error message

## Active game window

* 1. A cog icon will constantly be presented to the user
     1. Once clicked, the cog will open an options menu with the following options
        1. Surrender
           1. On click, the user will be presented with the following options

A button labeled “Yes”

On click, automatically lose the game and return to menu

A button labeled “No”

On click, return to options menu

* + - 1. Back to game
         1. On click, the options menu will close
  1. Upon entering “current game” state, both users will be presented with the option to select one of three possible decks.
     1. “Fire deck”
     2. “Water deck”
     3. “Wind deck”
  2. After deck selection, users will be presented with a “confirm” button
  3. After confirmation, users will be presented with all the cards in their “hand”
  4. User can click on a card in their hand
     1. On click, this will select a card
        1. User can click on an empty zone that shares a type with the selected card
           1. On click, the selected card will leave the hand and enter the selected zone
        2. User can deselect card by clicking away from card
  5. Once on the field, your minions cards can be clicked to select them
     1. Once selected, minions will display a context menu with the following options:
        1. “Attack”
           1. On click, the user will be prompted to pay the relevant energy cost to attack

The user will then be given the option to select which terrain they wish to expend

Once valid terrain has been selected to pay costs, the user will be presented with a button labeled “confirm attack”

* + - * 1. Once costs are paid, the user will be prompted to select which zone within its proximity to attack

Once a zone is attacked, the attack will resolve

An attack on a creature that does not kill said creature will indicate how much damage was dealt

An attack that kills the creature occupying the zone or an attack on an empty zone will result in the attacking creature occupying the zone

An attack on an opponent will indicate how much damage was dealt

* + - * 1. Alternatively, the user may press a button labeled cancel

On click, the attack will be canceled and the user will return to the default game state

* + - 1. minions with an activated effect will also display “Active effect”
         1. On click, the user will be prompted to pay the relevant energy cost to activate the designated ability

The user will then be given the option to select which terrain they wish to expend

Once valid terrain has been selected to pay costs, the user will be presented with a button labeled “confirm attack”

Alternatively, the user may press a button labeled cancel

On click, the activation will be canceled and the user will return to the default game state

* 1. Once on the field, your invocation cards can be clicked to select them
     1. Once selected, invocations will display a context menu with the following option:
        1. “Pay Ritual Cost”
           1. On click, the user will be prompted to expend terrain to pay for its indicated cost as outlined in 6.6.1.1.1.
           2. Once costs are paid, the invocation will resolve

Resolving invocations that require targets will prompt the player to select targets before resolution

* 1. An opponent’s hand will be represented by a number of card backs equal to the number of cards in an opponent’s hand
  2. Simultaneously as the user can see their hand, the user will be presented with the default game board containing the following zones:
     1. 2 friendly invocation zones
        1. Players will be allowed to place their invocations to their empty zones to give them their relative time needed to resolve
     2. 2 enemy invocation zones
     3. 3 friendly minion zones
        1. Players will be allowed to summon minions to their empty zones to give them a position to attack adjacent contested zones
     4. 3 enemy minion zones
     5. 1 friendly terrain zone
        1. Players will be allowed to play one terrain per turn to their turn to their terrain zone
     6. 1 enemy terrain zone
     7. 3 initially neutral contested zones
        1. Players will be able to attack contested zones with adjacent minions
  3. An “end turn” button will be superimposed over the board,
     1. When pressed, the user will end their turn
        1. After a user ends their turn, their opponents turn will begin
        2. During an opponent's turn, the “end turn” button will be grayed out
  4. The player’s life total will be superimposed over the board in green text
  5. The enemy life total will be superimposed over the board in red text
  6. A player’s deck will be represented by a card back on the player’s side of the field
     1. A number will be superimposed over the player’s deck to denote how many cards remain in their deck
  7. An opponent’s deck will be represented by a card back on their side of the field
     1. A number will be superimposed over the opponent’s deck to denote how many cards remain in their deck

## Post game window

* 1. Upon entering “post game” state, user will see one of the following based upon win condition
     1. Relevant message based upon win condition
     2. Button titled “rematch”
        1. Upon clicking, player is “readied up” for rematch
     3. Counter that updates upon players’ choice to rematch
     4. Button titled, “Main Menu”
        1. Once clicked, brings user to Pre-Game window state

## Error Message

* 1. Must be presented when exception occurs
  2. Text relevant to the exception handled
  3. “Close” button
     1. On click, exit error message window
  4. “Home Page” button
     1. On click, the user will be returned to the home page

# DESIGN DESCRIPTION:

## Client Facing Website

### General

The user will be able to interact directly with the following files: home.html, rules.html, game.html, sign\_in.php, register.php, and database\_error.php. All of these files, with exception of database\_error.php, will have another file, nav\_bar.php, that will be included to display a navigation panel to the user. Upon landing on the site, the user will be greeted with the home.html. This page and rules.html are strictly html and have no scripting functionality within it. Along with these two pages, upon loading into the browser as a .html file, the database\_error.php page is strictly html.

### sign\_in.php

The sign\_in.php page has two variables, $username and $password. The user will enter these values and will press a corresponding button to commit. Upon clicking this button, a function named signIn, taking the values as parameters, will run. It will attempt to log the client into the server. If the database returns that the combination is invalid, it will echo a relevant message to the client. If the combination is valid, the user will be successfully signed in and return to the homepage.

### register.php

The register.php page is similar to the sign\_in.php page in many ways. It has four variables, a $username, $password, $confirmUsername and $confirmPassword. The user will enter these values and will press a corresponding button to commit. Upon clicking this button, a function named registerUser, taking the values as parameters, will run. It will first check the database to see if the username is already in its records. If so, it will echo a relevant message to the client.. If the username is not already in the database it will check the $password and $confirmPassword as well as the $username and $confirmUsername variables to make sure that they are equal to one another. If they are not equal to one another, it will echo a relevant message to the client. If they are equal to each other, it will add a new record to the database, sign in the user, and return the user to the homepage

### game.html

#### General

The game.html page will contain a div element which will act as the game window. The game.html page will use a script tag to perform simple scripts as needed. The game window div will be in one of three possible phases, pre-game, current-game, and post-game. These phases are not “hard-coded” into the system but rather understood at a conceptual level. In the game.html page, there are no actual computations happening. The game.html page simply serves as a way for the user to input and output data to the backend. To streamline this process, game.html will import the socket.io library and will connect to the server via creating an io object.

#### Pre-Game

Upon loading the game.html file, the game window div will be in the pre-game phase. The pre-game phase has the sessionList JavaScript Object that is specific to it.It also has the joinSession string variable that is specific to it. It has the socket.on(“sessionList”), socket.emit(“joinSession”) and socket.emit(“createSession”) functions that are specific to it. Once loaded into the pre-game phase, the user will see a list of available sessions to join that is dynamically updated. This is done through the event listener socket.on(“sessionList”). When a user clicks a session in the list, they will pull the session ID related to the session they wish to join and upload it to the server using the socket.emit(“joinSession”) function. Once successful, the game window will enter the current-game phase of the game window. Optionally, the user can create a session. Doing this will trigger the socket.emit(“createSession”) function to run and will put the user in their own session that will be added to the sessionList JavaScript object, becoming viewable to all other clients. Once a user joins their session, the game window will enter the current-game phase. Once the window leaves the pre-game phase, it will set any variables used (sessionList and joinSession) to null.

#### Current-Game

Upon joining a session or having a user join your’s, the game window div will swap to the current-game phase and will ask the user which deck they want to use. The current-game window will utilize the following variables unique to itself: myTurn, myHealth, opponentHealth, and opponentHand. The current-game window will also utilize the following JavaScript objects unique to itself: gameState, myBoard, myHand. The current-game window will also have the following unique functions: cardClicked, getBoard, getOpponentHand, getMyHand, endTurn, socket.on, and socket.emit.The two most critical functions are socket.on and socket.emit. Socket.on will serve as an event listener for the client to the server. It will have the gameState JavaScript object, a master JavaScript object containing all other relevant JavaScript objects, returned to the client. Anytime the server makes changes to its “copy” of the gameState JSON file, a copy will be pushed to the client via the socket.on function. This will allow for the client to be updated dynamically without the need for the user to manually request for an update. Upon acquiring the new instance of the gameState JavaScript object, other relative functions such as getBoard, getOpponentHand, and getMyHand will be called. These will serve to break up the gameBoard object and store the board state, health pools, client’s hand, and opponent hand in their respective variables. These variables will be used in order to correctly display the relevant information to the client. It is worth noting that myHand is a JavaScript Object as relative information will need to be pulled from it while opponentHand is just an integer. This is because we do not need to pull any information from the opponent’s hand itself. We just need a count of the opponent’s hand in order to display to the client how many cards are in their hand. When the opponent makes a play, this will be translated into an updated gameState passed through our socket which will in turn update myBoard, and any other variables / JS objects as needed. When a user clicks a card, it will call the cardClicked function. This will iterate through all possible zones the card can be placed and display it to the user. Once the user places or moves a card, it will update the gameState JavaScript Object. This will trigger the io.emit function to push changes to the server. The user will be presented with an “End Turn” button. Once this is clicked, it will trigger the endTurn function which will flip the myTurn boolean making it where the users interaction is restricted and will reflect this in the gameState.json file passed to the backend. Once the backend sends over a value in the winnerID field, all variables related to the current-game phase within the game.html file will be set to null and the game window will enter the post-game phase.

#### Post-Game

The game.html file has variables specific to the post-game.phase: didWin, playAgain, winMessage, and playAgainMessage. Upon leaving the post-game game window phase, these variables will be set to null, except for playAgainMessage as this is a constant. The value didWin gains a value of true or false depending on the client’s win condition, and will see a message displayed relevant to this, utilizing the winMessage variable. The user will be prompted if they wish to play their opponent again, if so this will send a message to the server saying so. If the opponent also wishes to play again, the game window will return to the current-game phase. If the opponent is the first to send to the server that they wish to play again, the client will see the playAgainMessage displayed. The playAgainMessage will tell the user that their opponent wishes for a rematch. The user will also be presented with a leave match button. If either user clicks this button, both will have their game window set to the pre-game state.

## Back-end Event Logic

### General logic flow

In the broad scope, game actions will be collected on the front end, asynchronously sent to the back end. Once there, the information provided will be used to invoke the relevant function in game\_action.php. game\_action.php will in turn invoke various support methods(see Figure.2) to pull already known information on the game from the database, calculate gamestate changes, and modify the database as needed. Once database changes have been made, game action will call back to the relevant socket to notify the client the board needs to be updated to the new information stored in the database.

### game\_action.php

game\_action.php will serve as the transitional gateway between events collected on the front end and the logic and storage implemented on the back end. Once game actions have been collected and packaged to clarify the type of event and surrounding context variables, socket.io will be used to send said package in JSON(see figure 4) format to the backend. game\_action.php will then be loaded with the relevant event string (attack, invocation effect, minion effect,place card,end turn) and information associated with it. Depending on the event string, relevant scripts will be loaded with the context variables. Attack events will utilize the attack() function found in minion\_effects.php while non-attack minion effects will use minionEffect() found in the same file. Invocation effects will use the invoke() function found in invocation\_effects.php. Placing cards from a player’s hand to the summoning circle will utilize placeCard() found in board\_builder.php. Finally, a player ending their turn will utilize endTurn() found in turn\_master.php. After the relevant logic has been performed, the modificationsDone() method will be used to relay all changes have been made and the client ‘s gamestate representation must be updated via socket.io emit() method.

### minion\_effects.php

When loaded, minion\_effects first determines if the effect is an attack or a non-attack effect based on the information provided. Attacks will first be evaluated to see if they can be paid for using the evalCosts() method found in cost\_payer.php. If an attack can be paid for we will consider the minion’s attack effects using a switch statement evaluating itsminionID. While most minion’s attack will utilize the statement's default “basic” attack where a defending minon’s power is reduced by the attacker’s, specific minions with effects associated with an attack will load their own custom attack code to provide the relevant effect. Non-attack effects will also utilize a switch statement indexed by minionID. Minion effects that draw/discard cards will utilize the draw() and discard() methods provided by card\_draw.php.

### contested\_zone\_effects.php

Contested zone effects will primarily trigger at the beginning of turn, therefore zoneEffect() will be invoked most frequently by startTurn() in turn\_master.php. Once called, zoneEffect() will use a switch statement indexed by zoneID to perform the relevant effect.

### turn\_master.php

turn\_master.php will be invoked during an end turn event. It will primarily serve to change the active player column in the game table of the database as well as trigger any relevant “beginning of turn” effects in contested zones.

### turn\_timer.php

turn\_timer.php will count the elapsed time between turn changes and automatically end the turn if the elapsed time exceeds a predefined time limit. If a turn is automatically ended because of the time limit, the AFKwarning column in the playerboard table will be incremented. If a player is issued 3 AFKwarnings in a game, they will be issued an automatic game loss for slow play.

### board\_builder.php

In the event of a card being placed from a player’s hand to their summoning circle, board\_builder.php will serve to remove that card from the player’s hand table and add it to the playerboard table. Before the process is completed, cost\_payer.php will be used to ensure the action is legal and costs can be paid.

### Game creation process

Games will begin when a session is created by a single user. Once a second player attempts to join the session, a game will be created to store both players and the session id using create\_game.php. Once a player selects their deck, select\_deck.php will be used to copy over the selected deck from the predefined\_deck table to the active\_deck table.

### Winning and Losing

After each attack and damage dealing invocation, player health will be checked to see if it is less than or equal to zero. If it is, winnerID in the game table will be updated to the player with health greater than zero. Before each draw() in card\_draw.php, a player’s active deck size will be checked to ensure it has more than 0 cards remaining. If it does not, the player will draw nothing and similarly, winnerID will be updated.

## Game Database Information

### Player Table

Within the Player table exists two fields, one of which is the primary key. The primary key in this table is the username, which is a unique ID that each user will create upon initial visitation to the site. This is being used as a VARCHAR of size 32, as usernames do not need to be excessive in size. After creating their username, the user must create a password to allow them to access their username on every visit to the website. This is being used as a VARCHAR of the maximum size of 255 since passwords should always be secured and longer passwords are always more secure.

### Session Table

The Session table contains two fields, the primary key and a foreign key. The primary key in this table is the sessionID, which is a SMALLINT type and it is set to NOT NULL as the session list cannot exist if there is a null value. The foreign key in this table is the username, which is taken from the Player table, and therefore it has matching values.

### Game Table

The Game table consists of six fields, of which are the primary key and three foreign keys. The primary key is the gameID, which is a SMALLINT set to be NOT NULL, same as the sessionID. The foreign keys are the sessionID from the Session table, first\_player and second\_player, which are composite keys for the username from the Session table. The first of the last two fields is the winnerID which is equal to null as this needs to be null for the duration of the game until there is a winner, and winnerID will then be set equal to either first\_player or second\_player, whichever of the two won the game. The final field in this table is the active\_player which is equal to the first\_player, which means that the active\_player name is assigned to the first player to be put into the game. This is a way of labeling a host of a session.

### Minions Table

There are three card types in the game, one of which is the minion card type, and their information is contained within the Minions table. There are seven fields in this table, and of those we have the primary key and two foreign keys. The primary key is the minionID set to NOT NULL as all the other IDs are. The two foreign keys are the card\_type, which denotes the type of card, and the energy\_type, which denotes the type of energy it uses. Energy is a separate mechanic and has its own respective table. The remaining fields are the minion\_name, which stores the name of each minion in the game, the summon\_cost, which stores the cost to summon each minion, the start\_power, which stores the power that each minion has, and the attCost, which stores the cost it takes to make an attack move on the game board.

### Invocations Table

The second card type is the invocation card type, which has seven fields, of which are the primary key and two foreign keys. The primary key is the invocationID set to NOT NULL. The two foreign keys are the card\_type and the energy\_type, which operate in the same way as they do in the Minions table. The remaining fields are the invocation\_name, which stores the name for each invocation, fastCost, which stores the cost to speed up the turns needed to use the invocation, and slowTime, which is used to slow the time of the invocation.

### Terrain Table

The final card type is the terrain card type, which has three fields, of which are the primary key and two foreign keys. The primary key is the terrainID which is functionally the same as the other card types’ IDs, but stores its respective ID. The remaining two fields are the foreign keys, which are the card\_type and the energy\_type, which store the same relative information as these fields do in the other two card type tables.

### Cards Table

The Cards table is the central hub for all three previously mentioned card types. There are five fields, of which are the primary key and four foreign keys. The primary key is the cardID which is used to translate the information from the three card type tables into the PlayerBoard table. The remaining fields, the card\_type, invocationID, terrainID, and minionID are all transitive information going to the PlayerBoard from the respective card type tables.

### ActiveDeck Table

The ActiveDeck table is used to store information about the deck visible to both players on the Player Board. There are four fields, of which are the primary key and three foreign keys. The primary key is the active\_deckID, used to transfer information from the ActiveDeck table to the PlayerBoard table. The foreign keys are the username, the gameID, and the cardID, all taken from the Hand table.

### PredefinedDeck Table

The PredefinedDeck table is used to store the decks that have been created and defined for the purpose of the game. There are two fields, which are the primary key, which is the set\_deckID, and the foreign key, which is the cardID that is pulled from the Cards table..

### Hand Table

The Hand table is used to store the information about each user or player’s hand within the game. There are four fields, of which are the primary key and three foreign keys. The primary key is the handID set to NOT NULL. The foreign keys are the gameID, pulled from the PlayerBoard table, the username, pulled from the Player table, and the cardID, pulled from the Cards table.

### Energy Table

The Energy table is used to store the information about the energy mechanic within the game. There are two fields, of which are the primary key and a candidate key. The primary key is the energyID set to NOT NULL. The remaining field is energy\_type, which is used to store the type of energy for a given card.

### zoneTraits Table

The zoneTraits table is used to store the traits that a specific zone on the game board can have. There are two fields, of which are the primary key, which is zoneID set to NOT NULL, and the zone\_name, which is used to store the name for each type of zone.

### ContestedZone Table

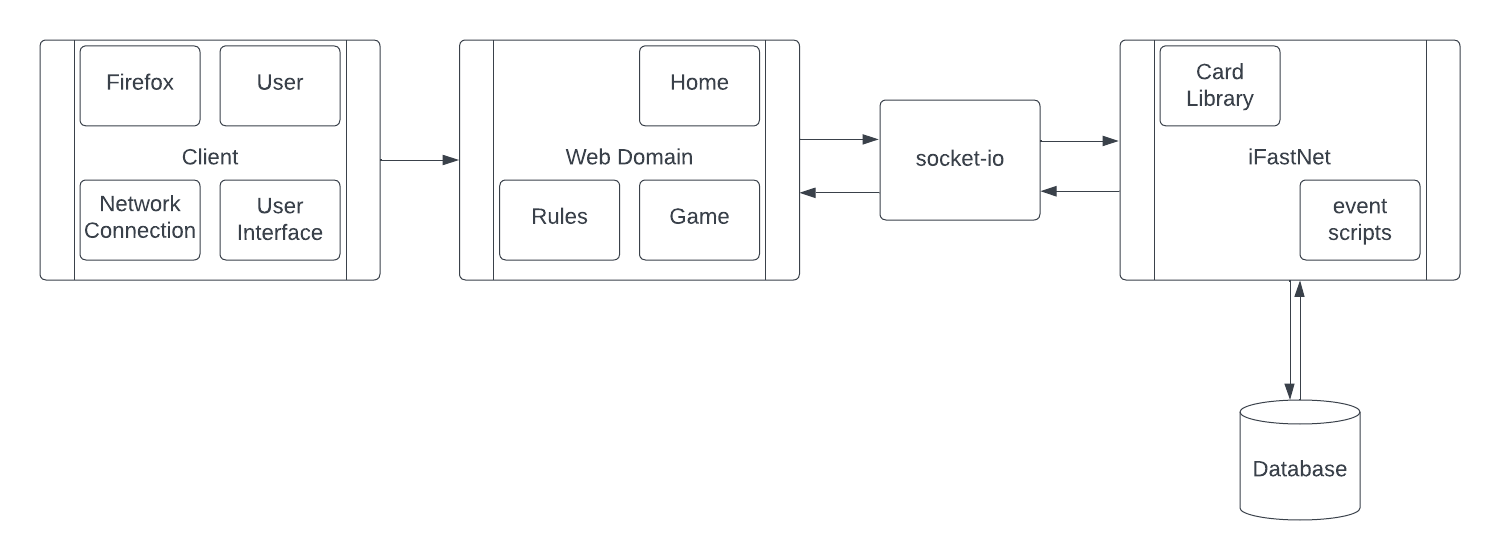
The ContestedZone table is used to store information about the contested zone on the active game board. There are thirteen fields, of which are the gameID from the Game table, the zone\_effect1-3 fields, which store the effect the zones can have, the isRevealed1-3 fields, which store boolean values for whether or not they are revealed on the game board, the zone\_slot1-3 fields, which are composite keys for zoneID from the zoneTraits table, and the minion\_pow1-3 fields, which are composite keys for the start\_power fields from the Minions table.

### PlayerBoard Table

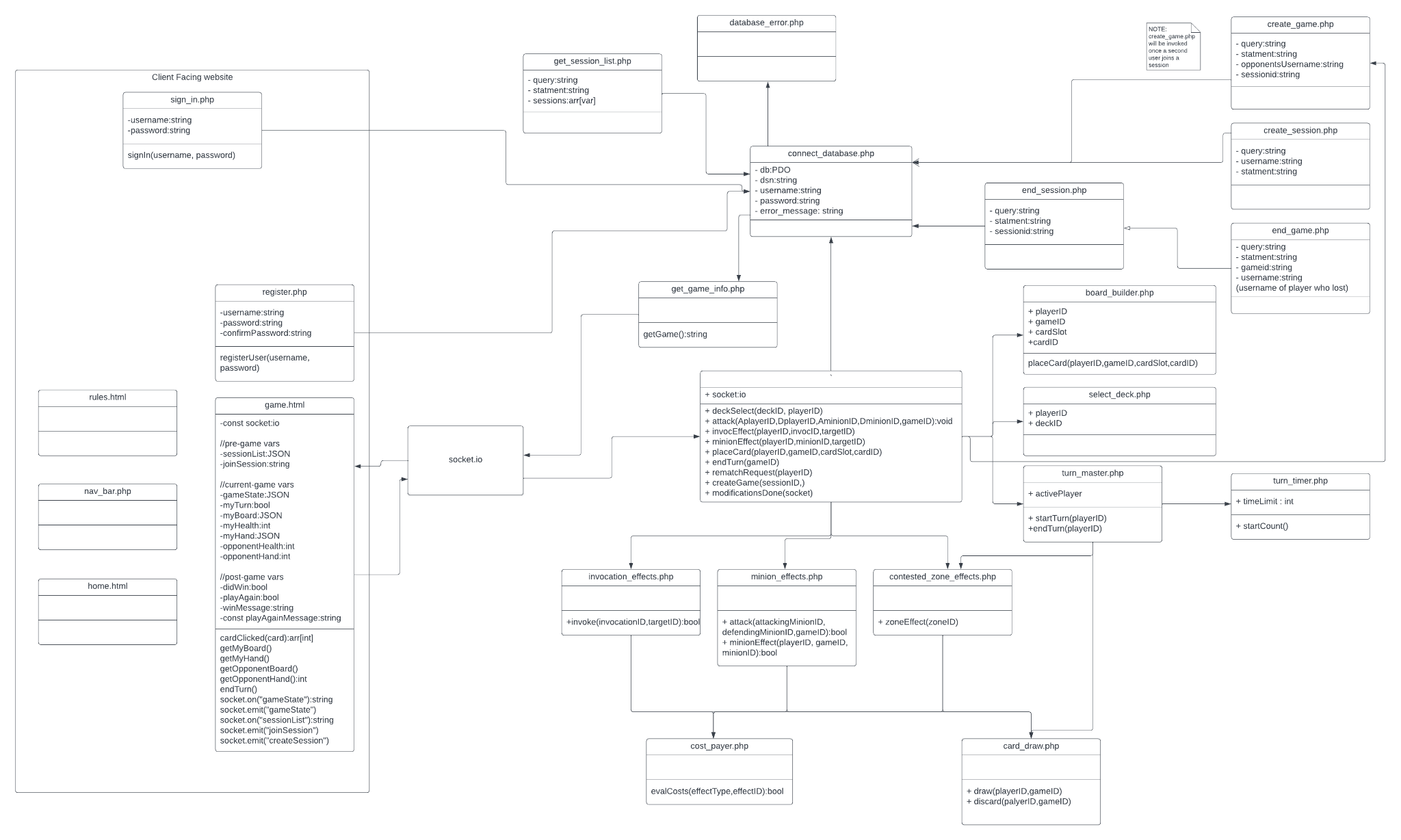
The final table in the database is the PlayerBoard table, which stores all information that will be made available and visible to the player. There are twenty fields in this table.The username is pulled from the Game table. The gameID is pulled from the Game table. The handID is pulled from the Hand table. The cardID table is pulled from the Cards table. The active\_deckID is pulled from the ActiveDeck table. The minion\_slot1-3 and minion\_pow1-3 fields, the invocation\_slot1-3 and invocation\_time1-2 fields, and the terrainID field are all taken from the Cards table. The active\_terrain\_count and the spent\_terrain\_count fields store the information regarding terrain cards that have or have not been used by the player. The player\_health field stores the information regarding the player’s health to make it available for the Player Board. The final field is the AFKwarnings field which is initially equal to zero, and it stores the information about whether the opponent player is AFK, meaning Away From Keyboard.

# Appendix:

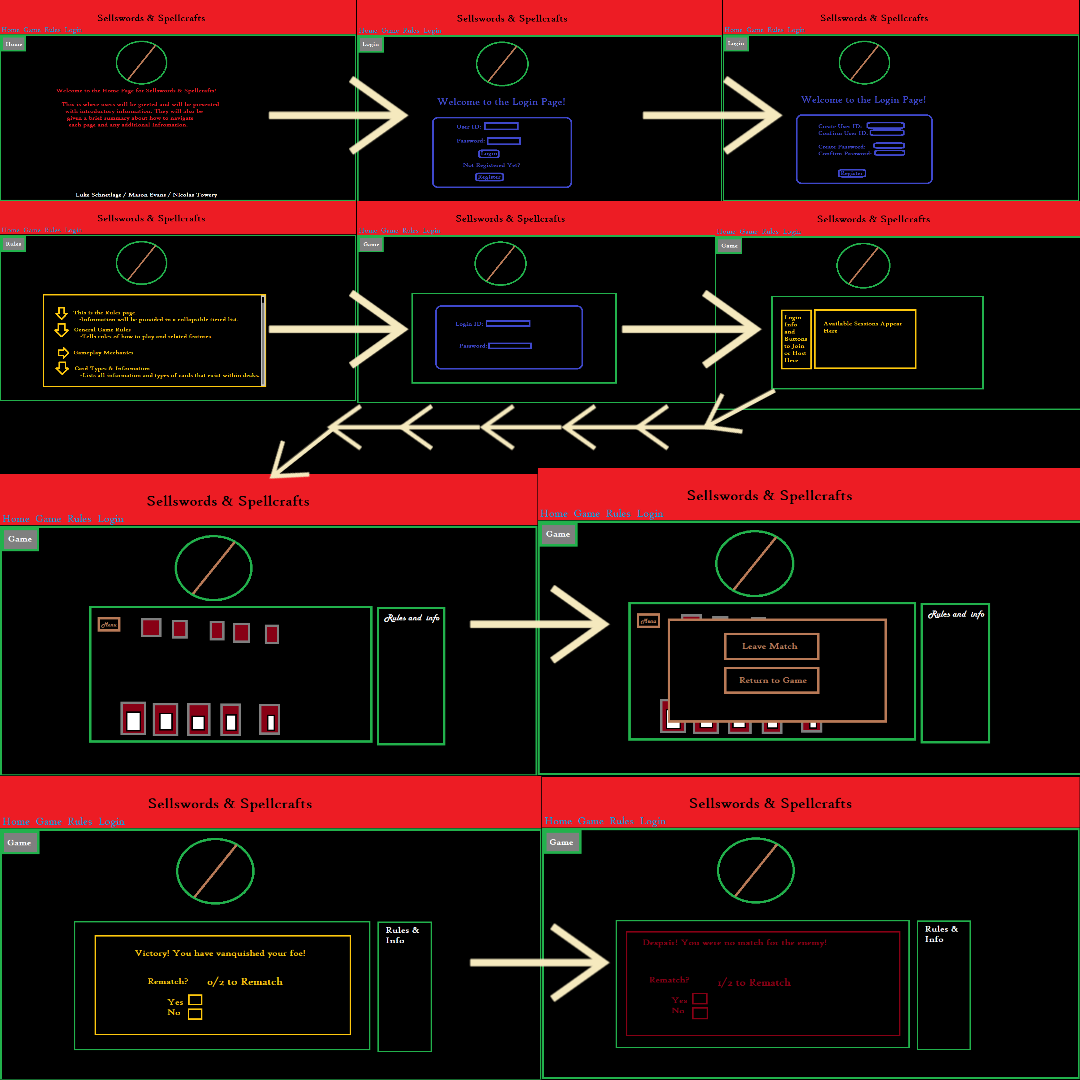
## FIGURE 1 - BLOCK DIAGRAM



## FIGURE 2 - COMPONENT DIAGRAM



## FIGURE 3 - USER INTERFACE STORYBOARD



[[1]](#footnote-0)

## FIGURE 4 - MESSAGE DOCUMENTATION

sessionList.json

{

“session”:[

{“sessionID”:”value”, “username”:”value”}

]

}

chooseDeck.json

{

“deck”:[

{“deck”:”value”}

]

}

gameState.json

{

“game”:[ {“gameID”:”value”,”first\_player”:”value”,”second\_player”:”value”,”winnerID”:”value”,”active\_player”:”value”}

]

“contestedZone”:[

{“zone\_effect1”:”value”,“isRevealed1”:”value”,“zone\_slot1”:”value”,”minion\_pow1”:”value”, “zone\_effect2”:”value”,“isRevealed2”:”value”,“zone\_slot2”:”value”,”minion\_pow2”:”value”, “zone\_effect3”:”value”,“isRevealed3”:”value”,“zone\_slot3”:”value”,”minion\_pow3”:”value”}

]

“playerBoard”:[

{“username”:“value”,“minion\_slot1”:”value”,“minion\_pow1”:”value”,“minion\_slot2”:”value”,“minion\_pow2”:”value”,“minion\_slot3”:”value”,“minion\_pow3”:”value”,“invocation\_slot1”:”value”,“invocation\_time1”:”value”,“invocation\_slot2”:”value”,“invocation\_time2”:”value”,“active\_terrain\_count”:”value”,“spent\_terrain\_count”:”value”,“terrainID”:”value”,“player\_health”:”value”}

]

“hand”:[

{“username”:”value”,“card1”:”value”,“card2”:”value”,“card3”:”value”,...}

]

}

## FIGURE 5 - Entity Relation Diagram

Mason

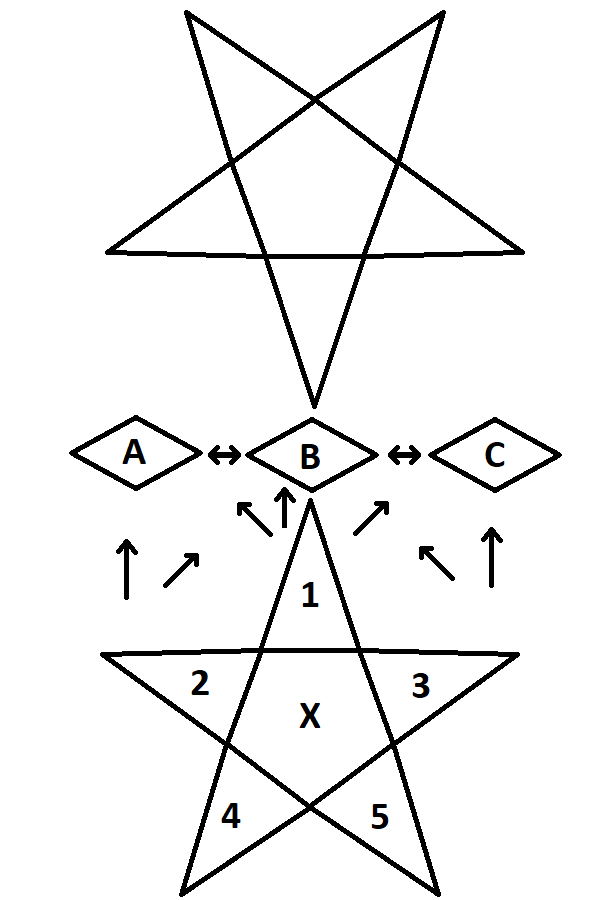
## FIGURE 6 - Game Rules Document

### 6.1. Anatomy of the board

The “board” will be divided into 3 broad zones: Your summoning circle, three contested zones, and the enemy summoning circle. Your summoning circle consists of three minion zones (Figure 6.1.1: 1,2,3), two invocation zones (Figure 1: 4,5), and a terrain zone (Figure 6.1.1: X). As the names imply, minions may only be summoned to minion zones, invocations may only be placed in invocation zones, and terrain may only be placed in the terrain zone. An opponent’s summoning circle is an exact copy of your own for your opponent.

Each contested zone will contain a contested zone card randomly selected from a pool of possible zones. Once occupied, a contested zone card will confer its specific benefit or drawback to the creature occupying its zone.

**Figure 6.1.1. Simplified rendition of the board**



### 6.2. Beginning of game

Games begin with both players drawing 5 cards from their randomized decks. A player will randomly be chosen to play first. The first player to play will not be allowed to draw a card at the beginning of their first turn. For all other turns, the turn player will draw a card at the beginning of their turn. All zones on both players' summoning circles start empty and all effects on contested zones start hidden.

### 6.3. Card types and their abilities

Player’s decks consist of 3 card types: minion cards, invocation cards, and terrain cards.

Minion cards have 5 pieces of information: summon cost, attack cost, power,energy alignment, and an optional effect. A minion's summon cost indicates the amount of life a player must pay in order to be summoned to an empty zone in a player’s summoning circle. An attack cost indicates the amount of terrain energy required for a minion to perform an attack. A minion’s power indicates the damage it deals to enemy players and minions. Energy alignment indicates the specific type of terrain a player must expend in order to pay for its attack or effect costs. Some, but not all, minions have additional effects that may require energy, life, or passive wait time to take effect.

Invocation cards have 4 pieces of information: casting time(slow), ritual cost(fast), an energy alignment, and an effect. Invocation cards have no cost associated with putting them from your hand to your summoning circle, however they have no inherent effect once placed. Casting time represents the number of turns you must pass through with a given invocation on your field (decremented at beginning of turn) before you may use an invocation’s effect. Alternatively, once an invocation is in a player’s summoning circle, at any point during a player’s turn, a player may pay energy of the type equivalent to its energy alignment in a quantity equivalent to its ritual cost to activate its effect immediately and forgo waiting for casting time. Each invocation’s effect is unique and includes, but is not limited to, gaining life, losing life, dealing damage, and drawing cards. Once an invocation’s effect is activated, it is removed from the zone it occupies.

Terrain only has 1 piece of information: energy alignment. Energy alignment determines what energy type a terrain card produces. Terrain may only produce 1 energy of its given type per turn.

### 6.4. Attacking and controlling zones

Once per turn, a minion may attack an empty, adjacent contested zone, a creature occupying an adjacent contested zone (illustrated by arrows in Figure 1), or an opponent. A minion may only attack after its respective attack cost has been paid.

Should a minion attack an unoccupied contested zone, that minion will automatically leave its current zone and occupy that zone. If that zone’s effect has not been revealed, the effect will be revealed as the minion occupies the zone.

Should a minion attack another minion, the attacking minion will reduce the defending minion’s power by the attacker’s power. If a minion’s power is reduced to zero, it is automatically destroyed. When a minion occupying a contested zone is destroyed by an attack, the attacker automatically takes the defending minion’s place in the contested zone.

A minion may only attack your opponent directly once it occupies a contested zone. Once a minion makes a direct attack on a player, the defending player loses life equal to the attacking minion’s power.

### 6.5. Win Conditions

A player loses the game once that player’s life reaches zero or below or they attempt to draw a card while no cards remain in their library.

1. See Sellswords & Spellcrafts Stuff.zip for more detailed images of webpages. [↑](#footnote-ref-0)