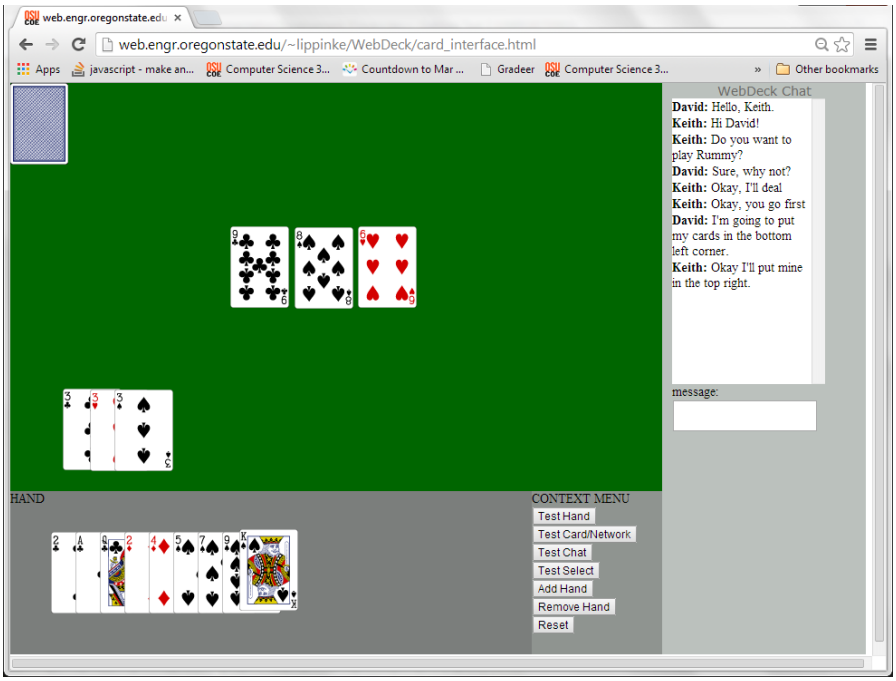
The Web Deck

A card playing interface

Oregon State University
CS 361: Software Engineering I
Prof. Danny Dig
Winter 2014

Contents

Figure 1: The game playing environment



1 Project Description

The Web Deck is a web-based application that connects players to a full virtual card playing environment. The application allows players to handle a virtual deck of cards as if it were a regular, physical, 52-card deck, giving them options to shuffle the deck, deal cards, place cards face up on the table, and more! Unlike currently available card playing applications, this system is not specialized to a single game, or to any set of games, but provides players the essential functionality necessary to setup and to play any card game they can think of. Along with the game playing environment there is a chat feature allowing players to discuss turns, game rules, or anything else that arises throughout the experience.

2 The Team

We are a group of undergraduate computer science students studying software engineering at Oregon State University. The members in our team are:

- Jackson Carter
- Keith Lippincott

- Jonathan Chang
- Tommy Ming Zhu
- Kamal Chaya
- David Rebhuhn
- Isacc Alltucker

3 The Software Developement Process

Our team followed the software development methodology known as Extreme Programming (XP). This approach allowed us to make significant progress in two week intervals. Among the core features of Extreme Programming that we involved in our software development are user stories, iterative development, testing, paired programming, code simplicity and quality, and frequent integration. Each of these components of extreme programming and the extent that we followed them is described below.

3.1 User Stories

At the outset of our project, we met with the two team members (Jackson and Keith) who proposed the initial idea for the project. We then gathered the functional requirements from them in the form of user stories (See section 4).

Initially, as we were new to the process, a few of the functional requirements for the project were proposed from a development perspective, rather than a client perspective. In most real world projects, the clients would not be on the development team, as Jackson and Keith were; thus, the distinction between client and developers were not as clear in the beginning of our project. After we had gathered these requirements, we asked Jackson and Keith which user stories were of the highest priority. Because this project had time constraints, we only had time to focus on the user stories proposed by these two members of our team. See section 6 for a description of the functionalities we hope to implement in the future.

3.2 Iterative Development

We released our project in iterations known as "milestones". Before we started development for each milestone, our group held a meeting known as the *planning game*. For each planning game, we asked our clients, Jackson and Keith, if they had ammendants or additions to the user stories, and which user stories they would like the team to implement that iteration. Once we identified the user stories we wanted to implement, we broke them down into tasks. We then assigned each task to a pair of programmers, and asked that pair to give a time estimate for how long it would take them to fulfill that task. We found that the nature of our project made it difficult to offer accurate time estimations because we were all learning a new set of programming languages, and performing tasks different than anything we had previously encountered.

3.3 Testing and Refactoring

One of the core features of extreme programming is test driven development (TDD). At the beginning of the project we did not fully understand the concept of TDD, and fell into the trap many developers fall into: we wrote a decent amount of code to get started, then worried about testing it near the end of each iteration. It was not until the third iteration after we saw a demonstration of test driven development that we began to start testing before, instead of after, writing code.

After this demonstration, we revised our testing approach. For each piece of functionality of our code, we first wrote a function that would act as a unit test for this bit of functionality (so initially of course, this unit test would be failing; this stage is usually referred to as the red stage). We would then write some code to accomplish this functionality that would hopefully pass the previously written unit test we just wrote earlier without breaking the functionaity of all preceding tests. This part of the test driven development cycle is known as the green stage. After accomplishing this rounds of tests, we would do what is commonly referred to as the refactor stage. In this stage, the programmer pair spends time mainly to clean up the unit tests and code.

Although this process seemed cumbersome at first, we found it to be a good way to ensure that all of our functions were working correctly. It gave us a systematic way to ensure that the requirements for each iteration were being met, and that most of the previous code remained operational upon adding new functionality.

3.4 Code Quality and Simplicity

Throughout the project we strived to keep our program design simple and high quality.To do this we maintained a coding standard that we required all team members to follow. Some example of the requirements that we imposed: long lines of code or string should be broken up, variables and functions must have descriptive names, class constructors should be at the bottom of a JavaScript class and the member variables at the top. For a full listing, see our wiki page.

In addition to keeping a coding standard, we also spent time, as mentioned in the previous section, to refactor our code to make it cleaner and more simple. One major refactoring we did was to transfer information stored in the JavaScript object portion of the `card class` to the html div portion. This made card handling simpler in the card methods.

3.5 Collaborative Coding

During each work session, we ensured that all portions of code being written for our application were written in pairs according to the *paired programming* model. In each pair, one person, known as the driver, is responsible for writing the code, and the other, the observer, continually reviews the code being produced, offering corrections for syntax errors and coding standard adherence.

We found that this helped us improve productivity a lot, mainly because two heads is

better than one. If the person writing the code made an error, often the error would be noticed quickly by the observer. We also ensured that, conforming to the rules of paired programming, the observer and driver in each pair switched roles periodically. We found that this helped team members keep their minds alert. As it is easy for people to get bored if they are doing the same type of work all the time, switching roles within a pair keeps our workflow dynamic. Preventing this type of boredome boosted our productivity.

Paired programming also gave us the benefit of matching up based upon experience. Team members with less knowledge or skill in a certain area could learn from a more skilled member, but offer new ideas. Numerous studies have shown that people who program in pairs will produce better code overall, so we felt this was a very useful practice for our team to adhere to.

Another part of collaborative coding is to do peer code reviews. Some teams have formal code reviews, but our team took a more casual approach to code review. We reviewed code produced by other pairs when questions arose about certain parts of the code and when we were performing integration. For the most part, our team felt like paired programming was enought to produce good quality code.

3.6 Frequent Integration

In extreme programming frequent integration is emphasized. By integrating frequently, the team can see how the pieces fit together and verify that all parts of the system are cohesive. At our biweekly meetings, we start off by having a group discussion to determine what each person would be accomplishing that meeting. Then after a few hours of coding, we would end the meeting by showing each other what we had finished. At the outset of our project, the tasks given to each pair were reasonably distinct, and it took a while for us to get to a point where everyone had a part of the system that functioned to the point where it could be integrated. For example, the chat feature could be developed independently from the card interfaces, thus for the first few weeks, the two were never integrated together into one web page. When both parts had been developed enough to do so, we integrated the two. Now that our project is up and running, we are able to integrate more frequently. Our code is broken into several different classes and each class has its own file. Early on we found that we had problems when multiple people worked on the same file and both tried to push changes later. To ensure that integrating is seamless at the end of each meeting we created a file checkout system. If a pair would like to work on a file they must possess a notecard with the name of the file written on it. Once a pair was done with the file they were working, they would push their changes then return the notecard to the end of the table.

4 Requirements and Specifications

4.1 User Stories

Below are the major user stories our clients (Jackson and Keith) requested be completed for this project. Dashes indicate sub-user (-) stories and numbered items indicate

important tasks needed to be completed for the user story.

- As a card game player, I want a vitual deck of cards I can use like a real deck of cards to play most card games using a standard 52 card deck (no pinochle decks or jokers).
  - I want to be able to drag cards anywhere on the table.
    1. Create a card class for card manipulation
    2. Create a function that detects mouse clicks.
    3. Create a function that turns on and off draggability for a card.
  - I want to be able to flip cards.
    1. Create a function that changes the card image displayed.
  - I want to be able to deal cards.
    1. Create a function that deals a specified amount of cards to each player.
  - I want to use cards without other users interfering.
    1. Make a class to handle selection and locking
    2. Create a function that deselects all cards.
  - I want to collect cards into piles.
    1. Create new deck class.
    2. Create database entry for decks.
    3. Integrate deck class with database through network class.
    4. Integrate deck class with interface through CSS.
    5. Create buttons in the interface that make new stacks, shuffle stacks
  - I want a hand of cards other users can't see.
    1. Create a player class to track player information.
    2. Create functions that create and destroy card divs on the table and hand.
    3. Create a button in the interface that moves a card to the players hand.
  - I want to be able to arrange cards in my hand.
    1. Create a function that will sort the cards in the players hand based on their relative values.
- As a card game player, I would like to have an account so I can login to WebDeck
  1. Modify the player class so that it stores a password for each player
  2. Store every user's username and password in the database.
  3. Create a form that appears upon first entering the WebDeck website, which prompts users to log in.
- As a card game player, I want to be able to add other users of WebDeck as friends.

1. Modify the player class so that it can store user ID's of the users that are considered friends
2. Create a function for the player class to add a user with a specific user ID as a friend.

- As a card game player, I would like to be able to rate other players.
  1. Modify the player class so that it can store a rating for each player.
  2. Create a function for the player class to rate a user with a given user ID with a rating (out of 10).
- As a game host, I want to create my own game session my friends can join.
  1. Make new game session class to handle game session information.
  2. Make new lobby webpage to handle username and session id joining.
  3. Integrate deck with game session initialization.
  4. Auto-generate user IDs for joining users.
- As a card game player, I want to be able to search for a game my friends have made.
  1. Create a function to search for a game based on session ID.
  2. Create a function to search for a game based on its description.
  3. Create a function to search for a game based on the user who created it.
- As a card game player, I want to be able to chat with the people I'm playing with.
  1. Enable session IDs to separate chats for different games.
  2. Make long lines of chat rap new onto new lines.

4.2 Use Cases

Game Play Use Cases

Actor	Task-Level Goal	Priority
Dealer	Shuffle	1
Player	Move Cards	1
Player	Chat	1
Player	Player hand	2

Lobby Use Cases

Actor	Task-Level Goal	Priority
Player	Join a Game	1
Player	Create a Game	1

Figure 2: Game use case diagram

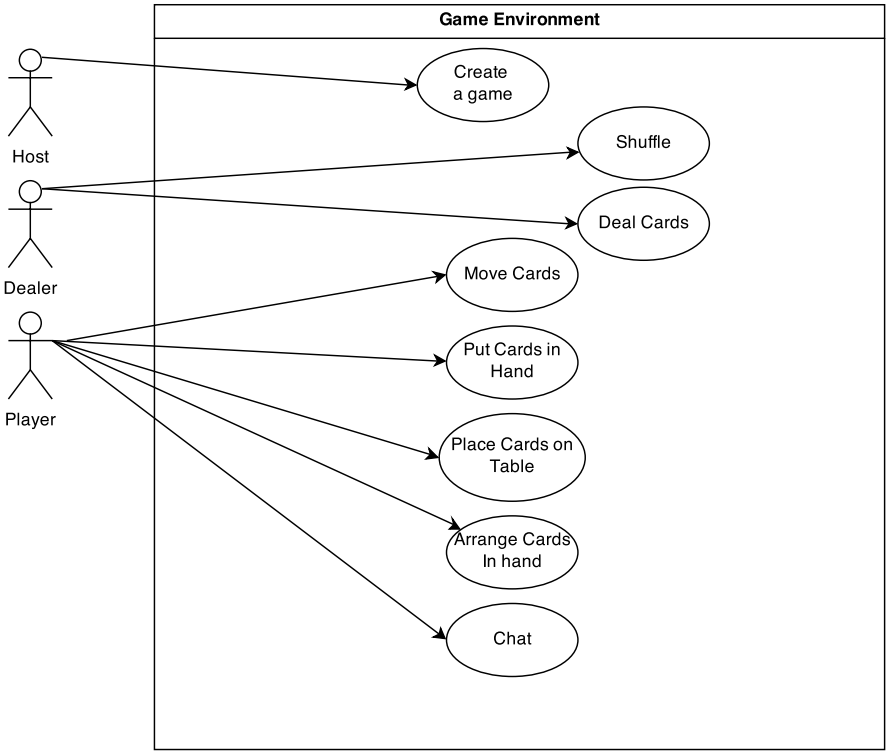


Figure 3: Lobby use case diagram

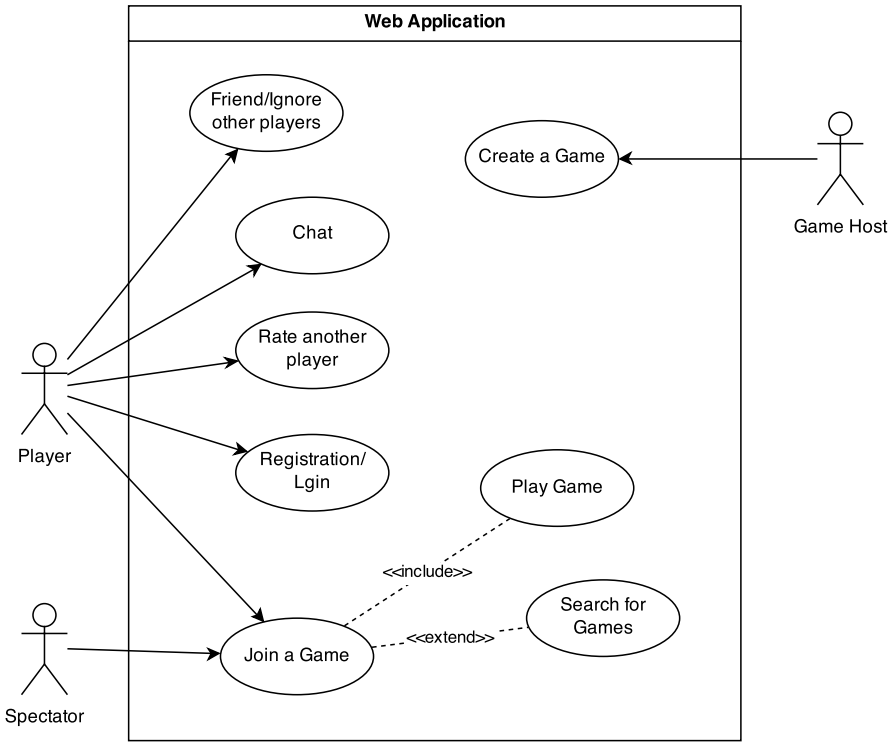


Figure 2 is a use case diagram that shows some of the main use cases in our application when users are playing a card game.

Figure 3 is a use case diagram showing the use cases involved with the lobby, such as joining games, and creating games.

The following use cases from the previous diagrams were not implemented because we felt that they were secondary to the core functionality required by our application:

- Use cases not implemented from Figure 2
  - Deal cards
  - Shuffle cards (currently you can reset the deck which shuffles it but you can not shuffle a given set of cards.)
- Use cases not implemented from Figure 3
  - Friending and Ignoring other players
  - Rating another player
  - Registration and logging in
  - Searching for games

Also, it is important to note that our application currently does not have a "host" or "spectator" implemented. Everyone using the application is treated as a player.

4.2.1 Create a Game

Primary Actor: User
Goal in Context: Create a card game session
Stakeholders and Interests:

- Hosts: Want to create a new game and invite fellow players in a timely fashion.
- Players: Want to be able to join a newly created game as a player.

Level: User (task)
Scope: System Level - User interacts with system to achieve goal
Trigger: The user wants to set up and create an instance of a new game.
Precondition: The user is on the WebDeck website.
Success End Condition: A new game session is created with a game id value that others can use to join the game.
Failure End Condition: The new game session is not created.
Minimal Guarantees: The interface will give the user some sort of feedback to the users inputs (confirming failure or success).
Performance Constraint: Game board load within 5 seconds without crashing the browser.
Main Success Scenario:

1. User enters a value for the game id that is not already used by a game session.
2. Site redirects user to a game session with the game id they entered.

4.2.2 Join a Game

Primary Actor: Player
Goal in Context: Play cards online
Stakeholders and their Interests:

- Player: Wants to join a card game.
- Host: Wants to ensure a set of good players joins card game (knows and follows rules).

Level: User level
Scope: Game Lobby System
Triggers: A player has found the game they want to join.
Preconditions: The player has logged in.
Guarantees:

- The player has a name to display.
- The game exists and is open.

Main Success Scenario:

1. A player requests to join a game.
2. The host for the game chooses to accept the player.
3. The player enters the game environment and added to the chat session.
4. The players name is put on an edge of the table.

Extensions:
2a. The host rejects the player.

1. The player is returned to the lobby to search for other games.

4.2.3 Shuffle

Primary Actor: Dealer
Goal in Context: Play a card game
Stakeholders and their Interests:

- Dealer: Wants to deal cards to players from a shuffled deck.
- Players: Want to receive unordered cards for playing.

Level: Sub-function
Scope: Game Environment
Triggers: The players are present and ready to play.
Preconditions: Cards have been selected for shuffling.
Guarantees: Selected cards are added to the current deck for shuffling.
Main Success Scenario:

1. Selected cards are returned to the deck.

2. The cards are arranged into a random permutation.

Extensions:
2a. If only one card is selected, no random permutation is performed.

4.2.4 Move Cards

Primary Actor: Player
Goal in Context: A player moves card(s)
Stakeholders and interests:

- Player: The player wants to move the cards correctly based on the rules of their card game.

Level: User level
Scope: Card game simulation
Trigger: The rules of the card game they are playing require that they move the cards
Precondition: There are cards available for the player to move.
Minimal Guarantees: The card is moved from its original location to the location that the player chooses.
Main success scenario:

1. The game reaches a point in time where the player is required by the rules to move card(s).

2. The player selects the cards to move and moves them to the location required by the game rules.

Failure:

1. The cards are not moved to the location that the player chose.

2. The cards are not moved at all.

3. The wrong card is moved (a card other than the one that the user chose to move).

4.3 Dealing cards

Primary Actor: Player
Goal in Context: To allow a player to deal cards to other players. Stakeholders and Interests:

- Player: Certain card games require that one player deal a certain amount of cards to other players. Implementing dealing functionality will ensure that players can play these card games.

Level: User level
Scope: Card Game Simulation
Trigger: The player clicks on a deck, then clicks "deal". Precondition: None Minimal Guarantees: the specified amount of cards are dealt from the selected deck to the specified players hands. Main Success Scenario:

1. After the player selects the deck, clicks deal, and enters in the number of cards to deal, and which player to deal it to (a dialog will pop up prompting the player for this information), and clicks "OK", the number of cards specified will be dealt to the players specified.

Failure:

- The cards are not dealt to the players in the manner that was specified when the deal button was clicked.

4.3.1 Chat

Primary Actor: Player
Goal in Context: To allow players to commnicate with each other using text messages
Stakeholders and Interests:

- Player: Wants to be able to chat with other players because communication is required to play a card game.
- Host: Needs to be able to communicate with the other players to ensure the card game is being played correctly, and needs to be able to give warnings to players if they are not following proper conduct.

Level: User level
Scope: Card game simulation
Trigger: The player enters a message in the chat box and presses enter
Precondition: None
Minimal Guarantees: The message that one player sends appears in the chat area, and all the other players see it
Main Success Scenario:

1. The message is seen by all the other players in the chat area after the player types the message in the chat box and presses enter.

Failure:

- The player types the message in the chat box and presses enter, but it cannot be seen by all the other players.

4.3.2 Player hand

Primary Actor: Player
Goal in Context: Continue game
Stakeholders and Interests:

- Player: Wants all selected cards to be placed in hand, and removed from the table
- Host: Wants game state to be successfully transmitted to all involved users.
- Other players: Want current game state to be transmitted.

Level: User
Scope: Card game simulation
Trigger: The player drags one or more cards into their hand area
Precondition: The cards are still on the table at the time of movement (the game state is synced)
Minimal Guarantees: The cards are removed from the table, the cards are added to the players hand.
Main Success Scenario:

1. Cards are added to the players hand.
2. Cards are removed from the table.

Failure:

- Cards are not removed from the table when they are added to the players hand.
- Cards are not added to the players hand.
- One or more cards are missing from the group added.

5 Architecture and Design

Our system allows users to perform various actions on cards and decks within a game table. Each user performs these actions by using the mouse and the keyboard. Actions performed by one player must be reflected in the browser window of all other players that are connected to the game without needing to manually refresh the system. In order to coordinate all of the players' actions, we use a backend database that stores information about the state of players, cards, and decks involved in each game instance. Each action performed by a player invokes a method which will update the database, as well as update the layout of the cards visible to all players.

5.1 The classes

Figure 4 shows our classes created within our code. The network class provides an interface for interacting with the database. The card class contains fields to uniquely identify each card, the image it uses, and so on. The deck class, not fully implemented at present, can collect a set of cards onscreen and convert it into a single deck object. Cards would be able to be added or removed as necessary from this object. The selection class handles the locking and selection mechanisms used in our system. Lastly the player class stores player information and has methods that allow players to add and remove cards to their hands.

5.1.1 The `card` class

Each card on the table contains two main parts: an html div and a Javascript object. The html div contains an SVG image of a card, sized appropriately to the playing board. This image's source is set to either the cards front image or the cards back image, depending on its flipped state (face up or face down). The card object stores the file names for the front and back images. When a user flips the card, the image that the card displays is toggled to its face-up or face-down image. Along with flippability, the card class contains functions that handle card dragging.

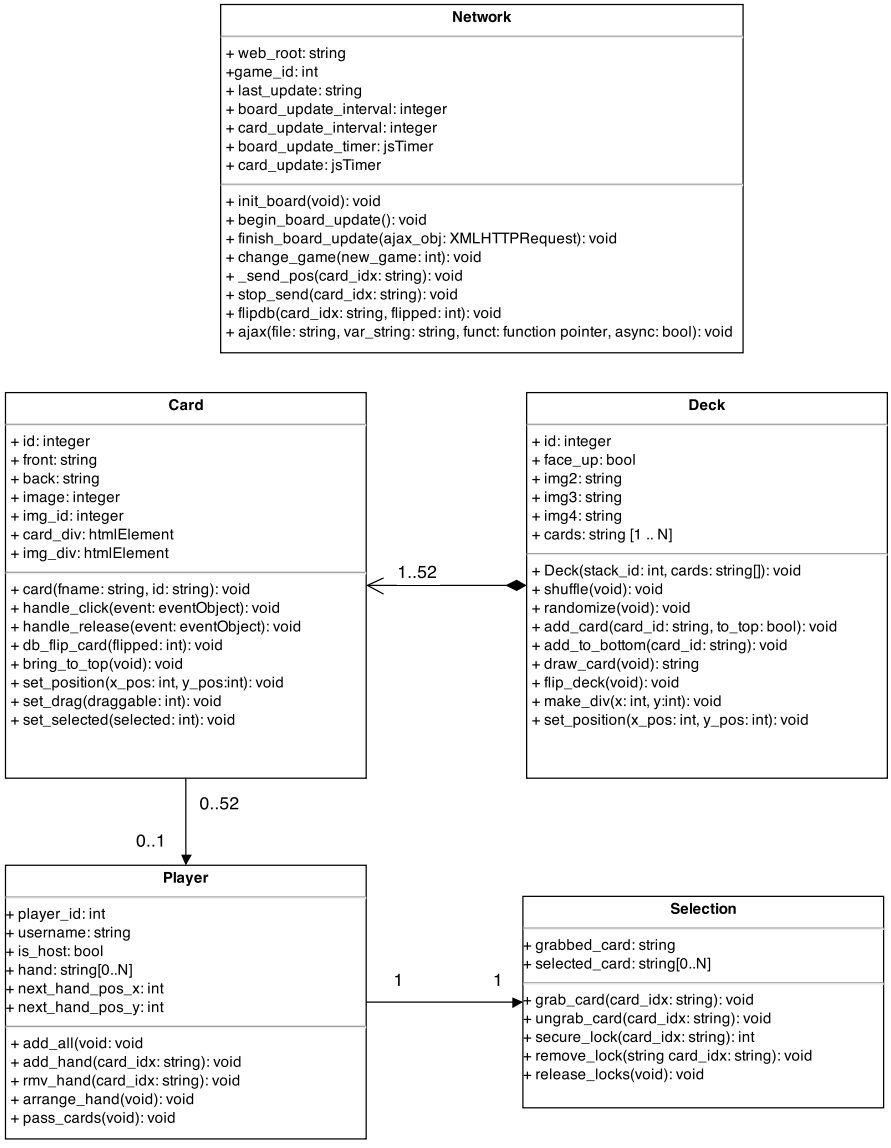
5.1.2 The `selection` class

In order to ensure that two players cannot simultaneously try to drag a card or deck, the system requires a locking mechanism. The `selection` class handles this locking mechanism. When a player clicks down on a card, it is defined to be "grabbed." When a card is "grabbed" the player that has it selected has ownership of the card. They can currently drag the card, and that cards' position is continuously updated in the database. As soon as the player lets go of the card, the card is now ungrabbed, but now "selected." A "selected" card is one which the player still has ownership over and no other player can move. A player can have multiple selected cards but only one grabbed card. The selection class handles all of the information about which cards are selected and grabbed and also contains the methods that are invoked when a player tries to grab another card.

5.1.3 The `network` class

This class is the interface to the database. Any query that needs to be made to the database goes through the `network` class. The `network` class also contains the very crucial update board method. This method asynchronously updates the game board on the clients browser at a specified time interval (on the order of 200ms). It is the update board method that coordinates each players game board so that all players see the same card layout.

Figure 4: The class diagram



5.1.4 The `player` class

The `player` class contains information about a users id, name, and what cards the player currently has in his or her hand. There are also methods in the `player` class to add and remove cards to the players hand.

5.1.5 The `deck` Class

The `deck` class was created to support deck functionality; i.e., the ability to assemble cards into a pile on the screen. This class implements several client-side methods for manipulating a deck: creating new deck objects, adding card objects to the system, and shuffling (randomizing the order) of card objects on the deck. However, we were unable to fully implement this class; network operations for synchronizing these client-side operations with all users proved too difficult to integrate with the rest of the system in the time we had left.

5.2 Framework

Our project did not utilize any existing frameworks. Rather, we built the application from the ground up. The one library we used was the jQuery JavaScript library to implement the draggability, but this library did not play a large role in our design decisions.

5.3 Sequence Diagrams

The following sequence diagrams illustrate some of the major use cases in our application. While most of the functionality in these sequence diagrams have been implemented, some features are still being developed. For example, a class to handle deck objects is still in the making and thus the sequence diagrams refering to these deck objects are more of plan for what we want our architecture to eventually look like.

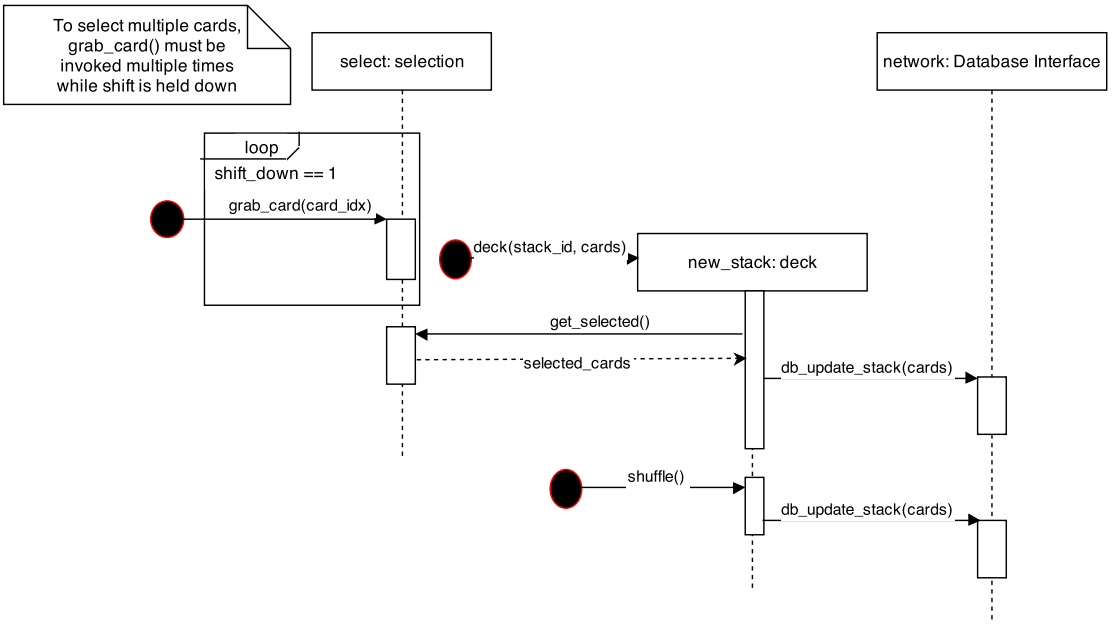
5.3.1 Creating and shuffling a deck of cards

Figure 5 shows the use case making a deck of cards and then shuffling it. This simple use case provides some details of the inner-workings of our architecture.

As you can see here, a user invokes the the `grab_card()` method in the `selection` class by holding down the shift key while clicking. Each grab will add another card to the list of the players selected cards. Then the user makes a deck out of the selected cards by pressing a button in the context menu. This button press invokes a method a to create a deck. When this happens, the cards on the screen that were selected will be replaced by an image of a deck of cards.

Finally, the network class invokes a method called `db_update_stack()`, which updates the attributes of the cards in the deck in the database. To shuffle cards, user can then invoke the `shuffle()` method by clicking another button in the context menu. The

Figure 5: Creating a deck of cards and shuffling them



new ordering of the deck must be reflected in the database as well, so the network class invokes `db_update_stack()`.

5.3.2 Manipulating cards

Figure 6 shows how cards are manipulated. First, a user invokes the `draw_card()` method when he or she right clicks on the deck, which draws a card from the main 52 card deck.

The `draw_card()` method removes the card from the stack (the stack has an internal list of cards), and creates a new card on the table that is visible to all of the players.

When a user left clicks a card then the `handle_click()` method is invoked from the `card` class. `handle_click()` detects that the click is a left click and calls the `grab_card()` method in the `selection` class. If no other player has this card selected the `grab_card()` calls `set_selected()` in the `card` class which puts a colored border around the card. Then `grab_card()` calls `_send_pos()` which sends the position of the card to the database at regular intervals. After `grab_card()` is called, `handle_click()` then calls `bring_to_top()` which will place the grabbed card on top of all other cards.

Also illustrated in figure 6 are the `handle_release` and `flip` functionalities. To deselect a card, the `handle_release()` method from the `card` class is invoked when the user releases the left mouse button. The card is removed from the array of currently selected

Figure 6: Manipulating cards

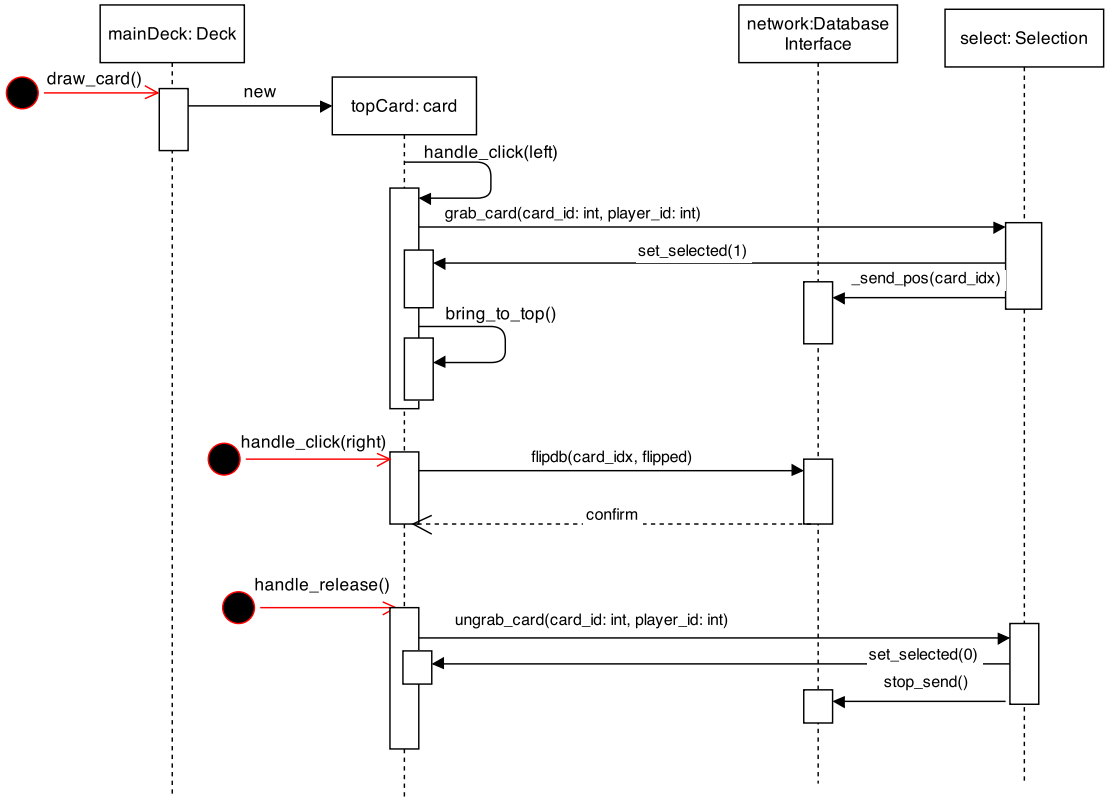
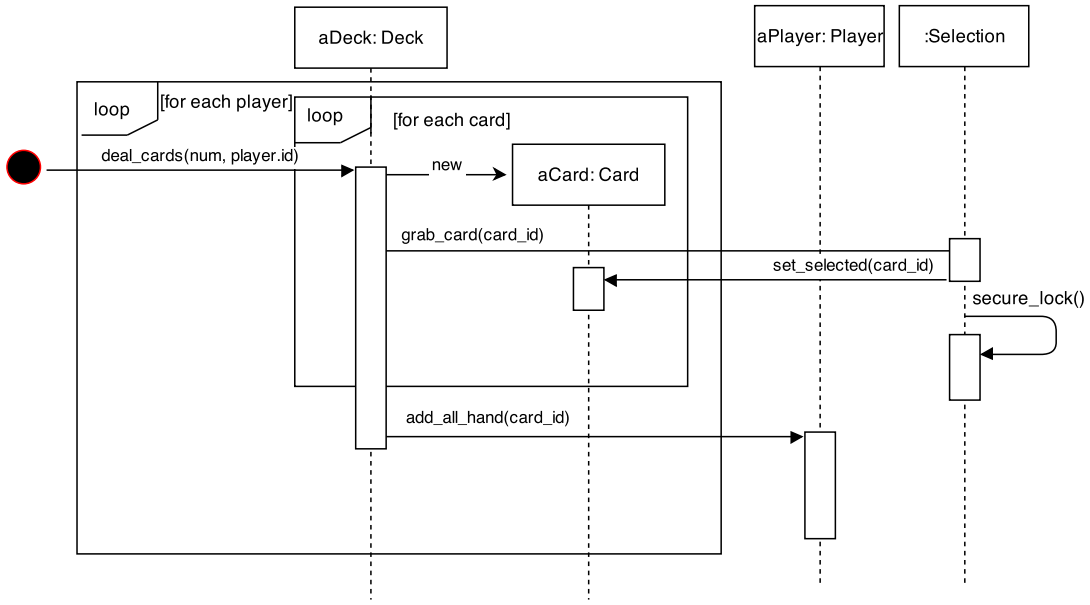


Figure 7: Dealing Cards



cards via the `ungrab_card()` method from the `selection` class, and `set_selected` is called to remove the border from the card. Then `ungrab_card()` calls `stop_send()` in the network class so that changes in the card's attributes are no longer sent to the database.

To flip a card, the user right clicks on a card, and this invokes the `handle_click()` method with the argument as "right". Then, the `flipdb()` method from the `network` class is invoked, to update the cards "flipped" attribute in the database (which tells us whether the card is face up or face down). After this, the `flipdb()` method returns a value stating whether or not the query to the database was successful or not. If it is, the card image source is toggled to be the back if it is currently the front or front if it is currently the back.

5.3.3 Dealing Cards

The sequence diagram displayed in figure 7 shows a functionality that has not yet been added to our architecture. A deal action will be invoked when a player clicks on a "deal cards" button in the context menu.

The `deal_cards()` method in turn will loop through each of the cards in the stack and call `grab_card()` for each player. This sets an even number of cards to be selected by each player. After it is done looping, the `deal_cards()` method will call `add_all_hand()` which adds all of the selected cards to the players hand.

6 Future Plans

As WebDeck as it stands is still a development release, we want to improve upon some of the technical issues which have yet to be resolved as well as implementing new features.

6.1 Current Technical Issues

- Latency: The most prominent issue present in WebDeck is high latency. When a user drags a card across their screen, this movement appears choppy and uneven to other players. This is likely because the server we are using, provided to us by Oregon State University, is not well-suited for a continuously updating a card-playing application due to limited bandwidth and data transfer speeds. As a consequence, we have needed to limit the refresh rate such that the server itself will not slow down the speed at which the page can load. We might address this issue in a few ways: by porting the application to a seperate server for more consistent loading speed, by optimizing PHP scripts and SQL queries to perform fewer unnecessary updates, or by simply managing the latency by smoothly transitioning the card from one place to the next rather than immediately render it at the new location.

- Card Draggability Quick-Click Problem: When a card is clicked or dragged too quickly, the card will get stuck to the mouse cursor and continue to be dragged even after the user has let go of the card. Currently, the issue may be fixed by clicking and holding the mouse for a few moments until the card is not draggable anymore; however, such workarounds are non-intuitive and detract from the overall experience.

- Deselect Problem: If you don't move a card before selecting another card, the lock on the first card is lost. This hinders the selectability of the cards.

6.2 Features to implement in the future

- Decks: In the future, we would like to fully implement the process of collecting a set of cards into a single deck object. While we have prototyped the client-side interactions of this functionality, additional work would need to take place to make deck creation consistent across all player views. This would involve modifying the network interactions that take place to ensure that the database gets updated, and that all players can see the changes that take place with cards on the board.

- Dumping the Hand: Players should be able to relinquish all cards from their hand at once, instead of one-at-a-time. Having a special function which automates the surrendering of all cards would improve the overall experience.

- Dealing cards directly to the hand of other players.

- Common Card-Playing Arrangements: In future versions, we would like to implement controls on the user interface that allow the user to specify a pre-existing

arrangement of cards for common card-games. For example, if a user wants to play Texas Hold'Em, a user of our current system would have to manually give 2 cards to each player and drag five cards into the middle. Automating this functionality would reduce the amount of time performing this operation and increase overall enjoyability of the system.

- Identifying the game session ID within the game.
- Identifying Users on the Game Table: Ideally, we should be able to see how many people are actually present in a given gaming session. While players are uniquely identified through the chat, the interface should include a visual representation of, at the very least, each player's username, and the number of cards present in his or her hand. This representation should appear when a user is present in the current session, and disappear when this user logs out of the system.
- Lobby Search: We would like to extend the lobby's current functionality by allowing users to search for specific games. This means that when someone creates a game, they should be able to specify certain tags or search descriptors which allow others to find a session and join it.
- Host-Only Abilities: Currently, the host - the user who created the game - has no special priviledges. Hosts cannot perform player moderation functions, such as accepting or rejecting users, or booting unruly players from a game session. In future iterations of our system, we would like to grant these priviledges to the game host, so that game hosts have greater control over the session and the game experience is as pleasant as possible.
- User Profiles: While our current system allows for uniquely-identified players, it does not possess the full functionality of a user-based system. User accounts are only used at this point to manage locks on card objects; there is no password validation and users are able to create and log in to any account that they wish, whether or not each user is actually logged in. In the future, we would like to add the ability to create an account beforehand, specifying details such a username, password, and a profile picture. This profile could be expanded in later iterations to display player statistics, such as number of games played, type of games played, total play time, and so on.
- Friends: As an extension of individual user account management, future additions to our system should allow users to add other players to a user-managed list of friends. Similar to other web apps such as YouTube or FaceBook, users should be able to send and recieve messages from other players, and invite them to gaming sessions.
- Facebook Integration: A more ambitious route may vastly improve the social aspect of WebDeck by integrating it with Facebook. If we merged WebDeck with Facebook's API, it will allow us to share the results of games and invite potential

players to join through Facebook. Creating a fan page of the application would also help boost the popularity of WebDeck in the long run.

6.3 Personal Reflections

6.3.1 Keith

All I have to say is that almost all new things are difficult and this project was no exception. This was the first time I used a software development process to develop code. This was the first time I did a project that lasted all term and this was the first time I did a project with more than four people. I found that it is difficult to divide work up effectively and to communicate precisely what each member is doing. In doing this project I felt like a learned why a development process is crucial to the success of a project. As we went on through the term I felt like we understood how to go about the process better and better. With that said, next time around, I think our team could do a much better job now that we know what we are doing, but given the fact that we were learning as we went, I think that we did quite well.

6.3.2 Kamal

Previously I did not have much experience with web based languages besides HTML, so this project was a huge learning opportunity for me. I really liked this project because it gave me practical experience using agile software development methodologies (namely extreme programming), as well as experience using test driven development. Gaining this kind of experience is a must in my opinion because it will ease the transition into the workforce when we become professional software engineers. Besides all this, I also liked this project because I felt that our project had the most innovative core idea (allowing people to play any card game they want online with each other).

6.3.3 David

This project has been the most difficult project I've ever had to work on in my career as a college student. Where most of the difficulty lied, however, was in working as part of a programming team, which I had never had experience with. Working out how to program effectively alongside other programmers and integrate the code into one project was a stressful and time-consuming experience, but in the end gave me valuable insights into the dynamics of team-oriented programming and how to collaborate with my team mates. I learned several methodologies and strategies during this term that I would not have learned as effectively had I not attempted to utilize them in a project setting, and I am greatful for the experience.

My other most valuable insight came from Test-Driven Development. I feel fortunate to have learned how to implement this process during the course of this term, and will attempt to follow it for many of my future programming assignments, such as those for my undergraduate research assistantship. I personally wish that this was one of the first

things we had learned during the term, as this would have helped us create a system from the bottom up and work out the kinks in our design from the start.

6.3.4 Jonathan

This project challenged the way I think about software development and how I approach group work. At the start, I had a lot of ideas I wanted to implement and bring a personal spin to the project. However, group dynamics and core features should have been first on my mind. We could have used much more cohesive planning in the beginning. Bad or ignorant design decisions had a domino effect for the entire course of the project. Particularly, we designed a multi-user application in totally the wrong direction. We started with the client user interface and worked our way to the database, when it should have been the other way around. This resulted in very lengthy re-design once it was time to integrate with the database.

It was also clear how important communication and documentation is for a group. There were many times we thought everyone was on the same page, but midway through the iteration, a multitude of miscommunications showed up. Even if you think everyone's on the same page, write everything down and double check. It's also super important to have a well defined leader who makes final calls and can direct everyone's workload. Ultimately, while it was a difficult term, I feel much better prepared for prjojects in the real world, and I think I've learned a lot about development.

6.3.5 Jackson

Boy, it's been a long haul. Bugs at every corner, managerial and linguistic challenges every meeting, github (enough said); lot of things could have gone a lot smoother through this project if we'd done our due diligence at the beginning of the term. I feel that Keith and I did not strongly ask the group to learn Javascript, GitHub or the XP programming model initially, and our team suffered for it. When we should have been doing numerous tutorials and readings, we divvied ourselves into groups and dived straight in, making costly mistakes regularly. Instead of planning out modules and interfaces, we focused on individual functionality, thinking it was less interconnected than it was. Integration became an immediate hassle, and it was just the start of our problems...

Hindsight being what it is, I feel now that it is of the utmost importance to thoroughly understand the language you will be using before you program in it (this means tricks and hacks, structure and layout, and best practices); it is of similar importance to know how to use a version control system; and it is key that the group develop a modularized, logical system architecture before even thinking about coding. Additionally, Test Driven Development is a wonderful practice that should be applied to all coding. Lastly, know your limits: a programmer simply isn't helpful when he/she is tired, sick, and/or burned out; this can do more harm than good.

I'm glad to have done this project, but I'm a little disappointed in the result. Progress was slow going given the general air of confusion and misunderstanding. Everybody needs to be alert and capable if the group is to be productive. Group work is challenging

and slow, but I believe in many cases we reached a better design as a group than I would have alone. Besides; it was more fun.

6.3.6 Isaac

I learned a lot of things throughout the project, but very few of them were specifically taught to me. The software development project was the first large, long-term group project I have been involved in. While some class homework assignments required significant time commitments, nothing compared to the hundreds of hours that went into the project. Through it all, I learned about code coordination, repository operation, proper use of commenting, and many other techniques and skills that could only be learned by actually programming in a group. I believe I can say that I am proud of the project. It is the physical product of many hours of programming, debugging, and research. Unlike a typical class project or homework assignment, the goals and design were left completely up to our group, and the amount of things we accomplished, even when compared against our ambitious goals that we did not accomplish, are significant. The design project was a valuable learning experience, the result of which I can proudly point to as something real that I accomplished.

6.3.7 Tommy

The development of WebDeck was an educational experience to say the least. My knowledge of GitHub, JavaScript and PHP is far greater than before. I also learned how to work with a team of seven people. Everyone had unique perspectives for approaching and working on certain tasks and it was interesting to see how seven different personalities meshed. We followed the Extreme Programming development process and I felt that it was a great guide for our team. A big part of extreme programming is to have a solid plan before doing anything. Our planning was a bit off at first and we suffered from it. At first it was hard to organize what needed to be done and who needs to work on which task, but as the project progressed so did our organization. I would say we handled our mistakes pretty well and had overall good team chemistry. Most team members were pretty flexible and I would say that had a major part to do with the success of our project.

7 Appendix

7.1 How to use WebDeck

First, navigate to the WebDeck website in your web browser (It is currently located at http://web.engr.oregonstate.edu/ cartejac/WebDeck/lobby.php), Enter in your desired username and the ID of the game you want to join in the form and press join.

If you enter the ID of a game which has not been created yet, a game with that ID will be created for you and you will join that game. If you enter a username that has not been used before with WebDeck, a notification will show, asking you if you want to

create an account. If you accept this, it will take you to the game session. If you click cancel, you will go back to the lobby screen from earlier.

Each game session has a unique game ID number which distinguishes it from other game sessions. If you have a particular game you would like to join, you must know its game ID.

Assuming enough players have followed the above instructions and are in the same game session as you, you can now begin to play a card game. WebDeck is not specialized towards any card game, it simply provides a virtual deck of cards that you can use as you wish. So at this point, you and the other users in the game session can begin to play a card game as you would in real life.

The virtual 52 card deck is in the upper left corner of the playing table (the green area). Left clicking on a card will select the card for you. You can left click, hold, and then drag the mouse to move the card wherever you want. The cards are all face down initially, but you can right click on the cards to flip them. When a card has a blue border around it, that means it is currently selected by you. You can left click anywhere on the screen that is not a card to deselect it. When another player has a card selected, the card will have a red border around it. You will be unable to select and move this card until the player who currently has the card deselects it.

You can add any currently selected cards to your "hand". The hand is the gray area of the screen below the central green playing area. When a card is in your hand, it cannot be seen by other players.

Inevitably you will need to communicate with other players while you are playing a card game. The chat area is to the right of the green playing area, titled "WebDeck Chat". There is a small box titled "message:", where you can type in a message, and press enter. When you do this, your chat message will appear in the chat area (the bigger box) above, and all of the other players will be able to see it.