Team 15
9/22/2014
Design Document

Cardglomerate

Santiago Abondano, Nate Cool, Kamal Khan, Jordan Mounts, Richard Phifer

Purpose:

To design and create a platform for users to play single- and multi-player card games online, with a framework that allows developers to easily add content. We intend to build a web-based application for one or more users to play a variety of card-based games with each other or computer AI, with a framework that allows developers to easily add content.. Within this application will also be metagame features such as statistics, achievements, and inter-user chat.

Design Outline:

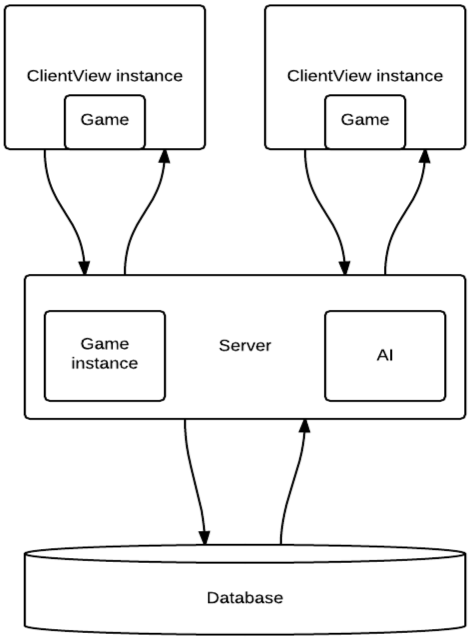
Client-Server Architecture

Client:

- Every user(player) of the system will be running a copy of the client application.
- Will provide a graphical user interface for the user.
- Will accept input from the user and validate it before sending it to the server.
- Because we have chosen to use a fat client to decrease server load, the client will do a majority of the game logic processing before sending the move made by the user(if the move is valid) to the server.

Server:

- Will facilitate all communications between clients.
- Will maintain the current game state for every game, updating a game instance when a client sends a valid change in that game's state.
    - Will update all clients connected to that game.
- Will have access and will maintain the database that stores player information.



Model-View-Controller Architecture in the Client Application

Model:

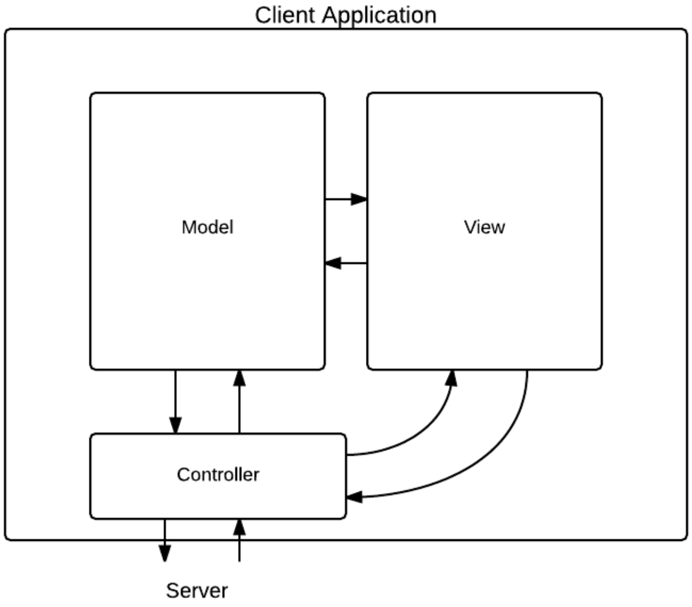
- Maintains the state of the application.
- Checks user input and communicates with the controller when valid input changes the client's application state.

View:

- Runs on a different thread of execution than the Model and Controller code.
- Draws the current state of the application to the screen(60 times or more per second).

Controller:

- Updates the Model and View.
- Communicates with the server to get updates to the application state made by other clients.
- Will run on the same thread of execution as the model.



Design Issues:

How clients communicate with each other.

Options: client-client, client-server-client.

Pros for client-client:

- Generally faster to talk directly to another client than through a server first.
- No dependance on a 3rd party.
    - With client-server-client communications, when the server goes down no clients can talk to each other.

Pros for client-server-client:

- Server can mediate and monitor communications.
    - Increased security for clients.
- As a conversation gets more clients(players), the processing for communicating between them stays roughly the same.
- If one client terminates the connection, the server can notify the other clients and allow them to continue.
- One client does not need to know anything about the other client’s location or other information to communicate.
    - Increased security.

Decision: Client-server-client communications

- Many games will have more than two users.
    - Client-client communication with more than two users becomes very complicated to implement.
- Clients should not have to store every client’s location in order to communicate with them.
- Also, locations are not static, and can change between sessions.

Long-term storage.

Options: Database or local files.

Pros for database:

- More secure - only accessible through a server.
- Easier to organize and maintain.
- The server connected to the database has knowledge of all the players.
- Clients(players) do not have access to other clients’ information.

Pros for local files:

- Less strain on the server in terms of storage.
    - Computing/storage is more distributed among clients.
- Could access information without talking to the server.
    - Faster.

Decision: Database

- Fits nicely with client-server communications.
- Much easier to manage all players and games.
    - Especially “meta-gaming” features.
- Makes player information much more secure.

Size of client application.
Options: fat client or thin client.
Pros for fat client:

- Much less server load.
    - Computing is more distributed.
    - Faster response time when server is under load.

Pros for thin client:

- The client application is lighter-weight.
    - Faster to download.
    - Uses less of the client's computing power -- less intrusive.
- Potentially more secure.
    - Less game logic and information on the client side.

Decision: Fat client

- Players will potentially use the application a lot, making a longer download less of a problem.
- The server needs to be able to scale up to accommodate a large number of players and games.
    - A fat client reduces the strain put on the server.
- Security is not a big issue for a card game engine.

Identification
Options: persistent identity or anonymity.
Pros for persistent identity:

- Ability to store information
    - achievements
    - statistics
    - chat
    - friends

Pros for anonymity:

- Quicker for a new player to begin playing.

Decision: Persistent Identity

- Many product requirements require user information to be stored
- We already ruled out using local files in previous design issue, which makes anonymity very difficult to implement and fulfill all requirements.

Chat history
Options: no chat history, full chat history, recent chat history.
Pros for no chat history:

- No database involvement.
- Simple chat implementation.

Pros for full chat history:

- Allows users to reference all chat messages at a later time.

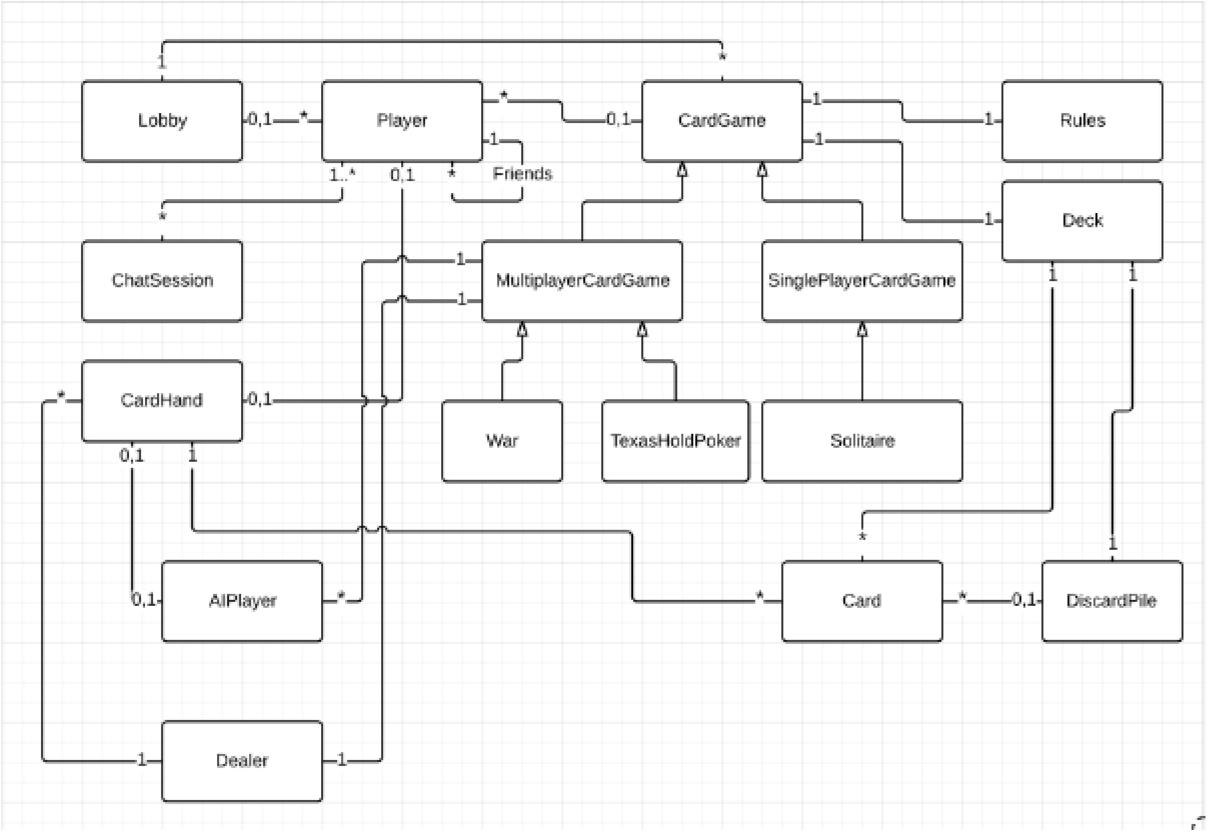
Pros for recent chat history:

- Allows users to reference recent chat messages from a different session.
- Less long-term storage needed to store messages.

Decision: No chat history

- Users of an online card game engine will likely use chat history very little.
- A simple chat service with no database footprint is preferred over the other options.

<u>Design Details:</u>



High-level class diagram

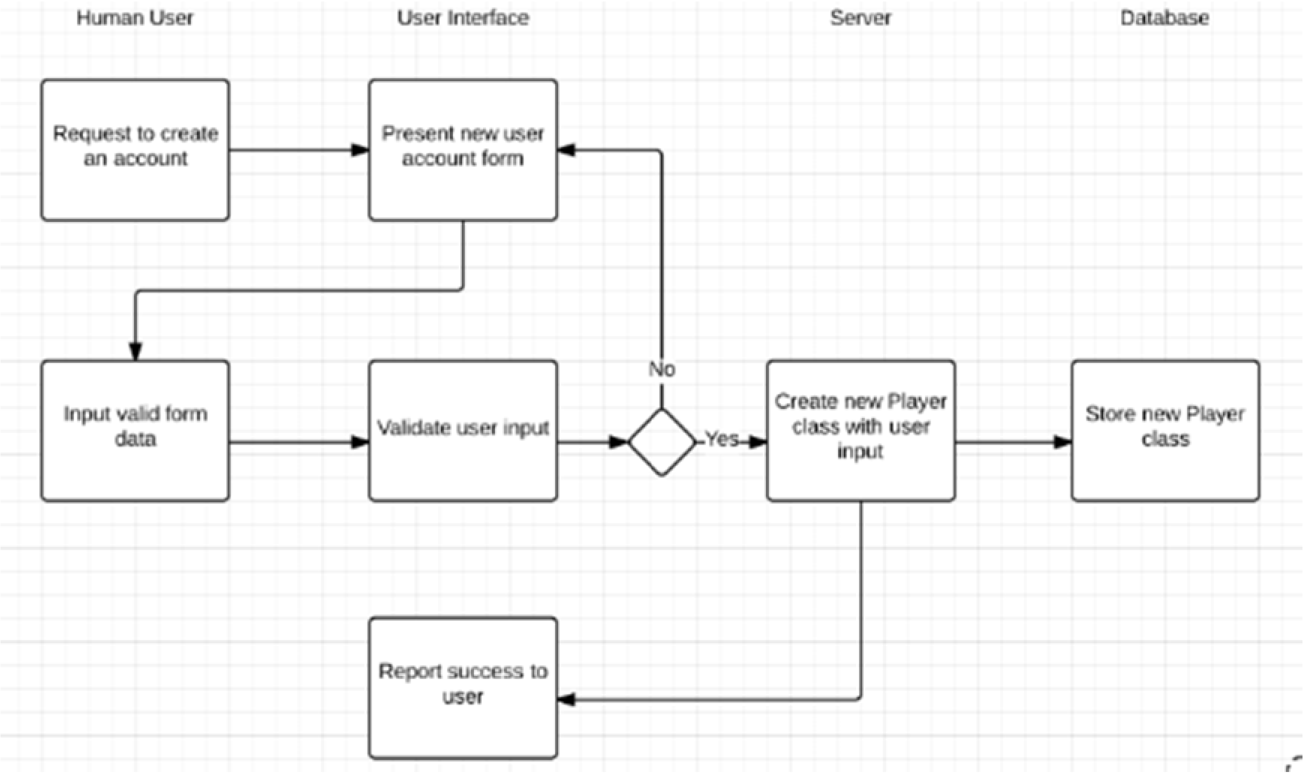
- There are two central classes in the system: Player and CardGame.
- Player class:
  - Models a single user of the system.
  - Can be associated with 0 or 1 card game.
  - Can be associated with 0 or more other Player instances, which represents that Player’s friend list.
  - Can be associated with 0 or 1 card hands.
  - Can be associated with 0 or more chat sessions.
  - Can be associated with 0 or 1 lobbies.
- CardGame class:
  - Models a single instance of a game in the system.
  - Can be associated with 0 or more players.
  - Is associated with a lobby.

  - Is associated with one deck of cards and one set of rules.
  - Is the root of the card game inheritance hierarchy.
  - Card game inheritance hierarchy:
    - Separates single and multi player card games.
    - Allows the different types of card games to be treated the same by the lobby and other game-management classes through polymorphism.
    - Each leaf class in the hierarchy represents a distinct card game and will provide its own set of rules.
    - Single player card games
      - Will have no AI players.
    - Multiplayer card games
      - Have AI players and a dealer.
- Lobby class
  - Can be associated with 0 or more card games.
  - Can be associated with 0 or more players.
  - Serves as a game-management service for users, allowing them to browse and create card games.
- ChatSession class
  - Can be associated with one or more players.
  - Provides communication capabilities between players.
  - All messages will first be sent to the server, which will then be sent to all other players in the chat session.
  - Because this system will have no chat history between sessions, this class will be relatively simple, relaying messages to the server and receiving updates from the server when other players update the same chat session.
- CardHand class
  - Models a player’s hand in a card game, including AI players.
  - Can be associated with 1 player or 1 AI player.
  - Is associated with a dealer.
- AIPlayer class
  - Models a computer player in the system.
  - Is only associated with multiplayer card games(single player card games have no need of AI players).
  - Can be associated with 0 or 1 card hands.
- Dealer class
  - Models the dealer in a card game.
  - Is responsible for shuffling, distributing card hands fairly, and checking victory conditions.
  - Can be associated with 0 or more card hands(the hands it deals).
  - Is associated with 1 card game.
- Rules class
  - Models the rules that make up a card game.
  - Decides what moves are valid and hand strength.
  - Provides victory conditions.

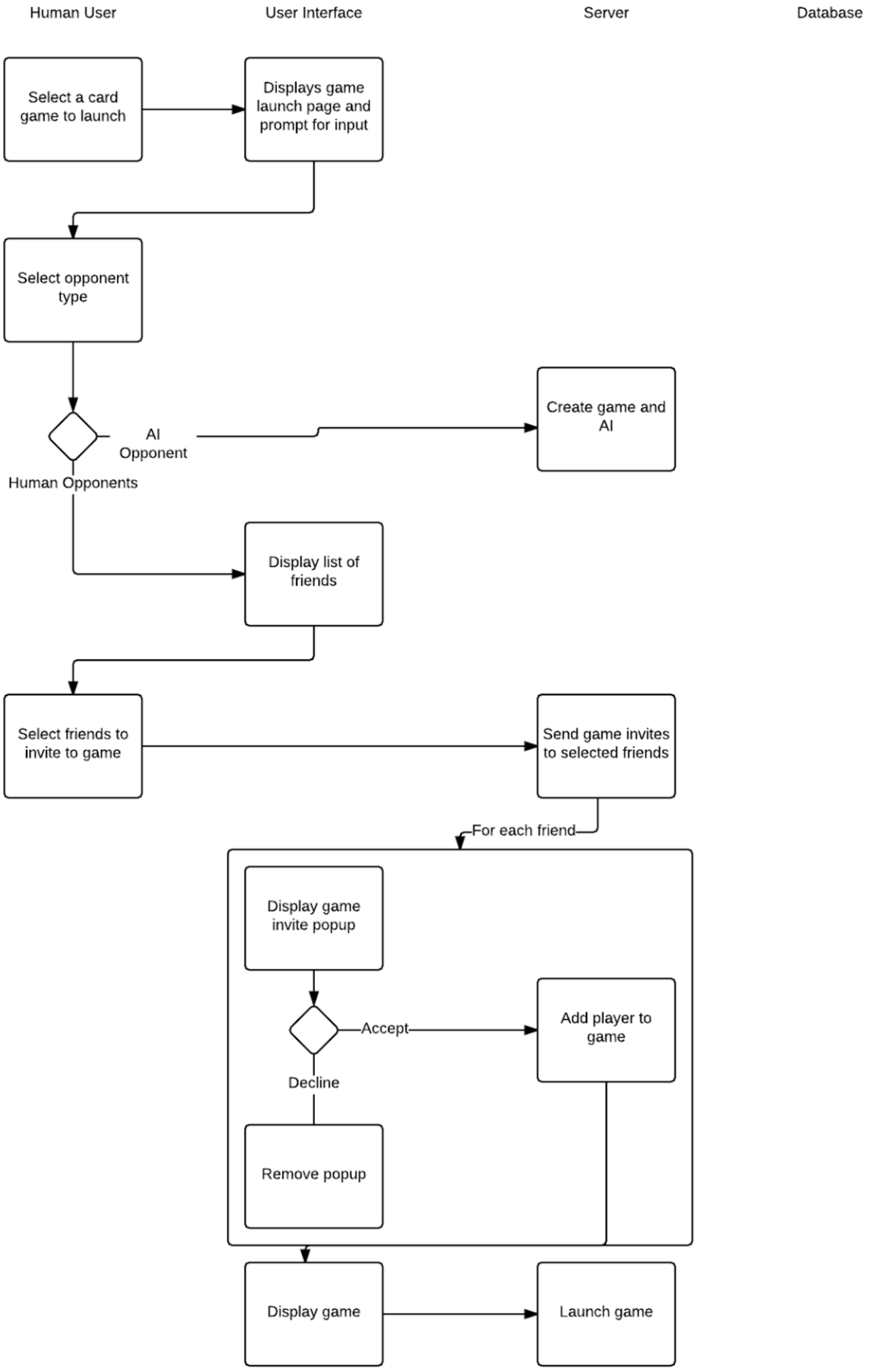
    - Is associated with 1 card game.
- Deck class
    - Models the deck of cards that a card game draws from.
    - In most cases, is a standard 52-card deck.
    - Provides a uniformly random shuffle service.
    - Is associated with 1 card game.

Sequence Diagrams

Create user account

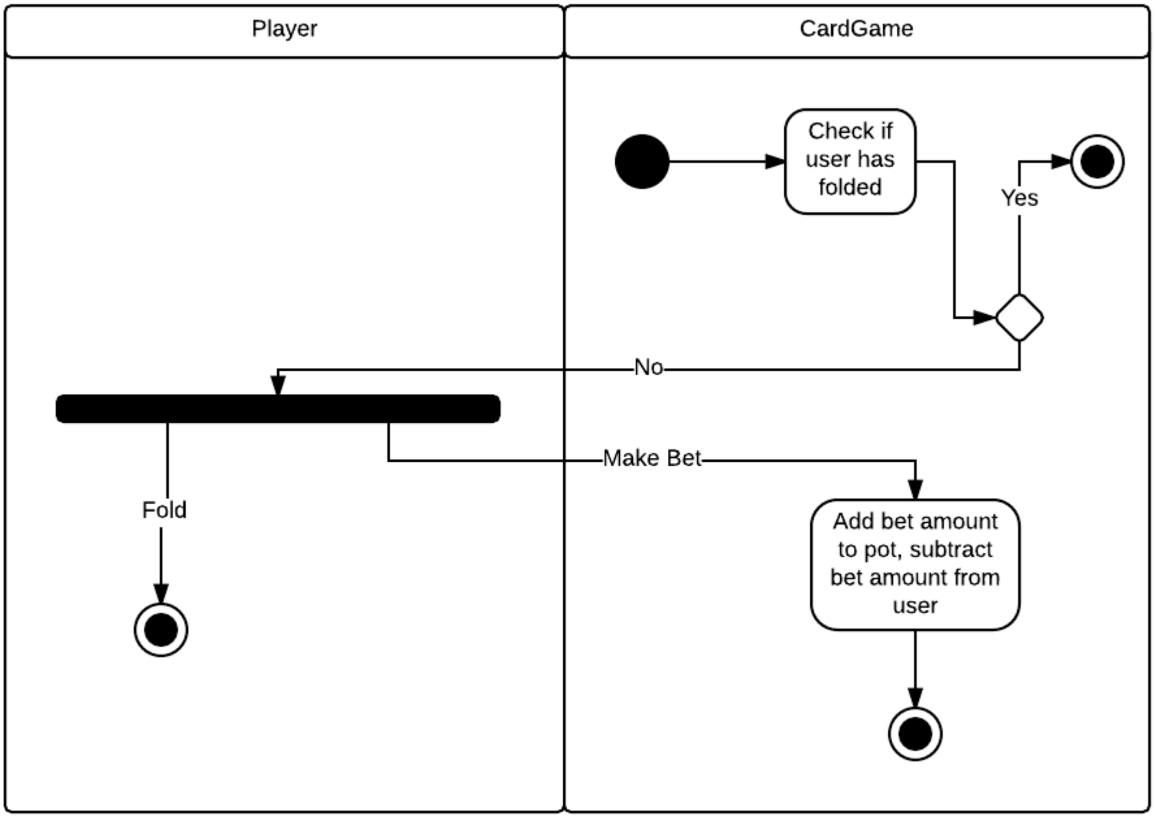


Launch a new game
(next page)

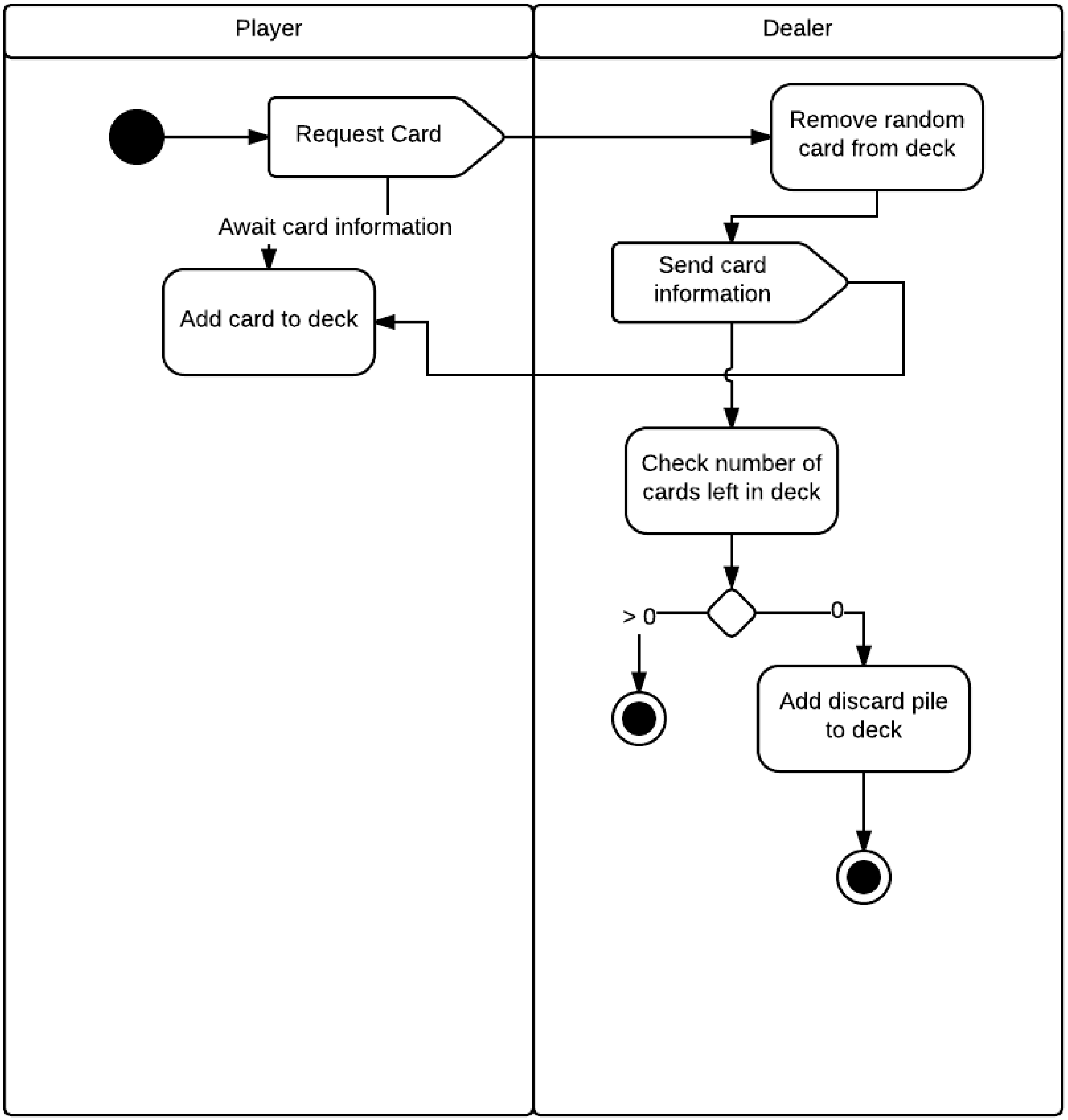

Human User
User Interface
Server
Database
Select a card game to launch
Displays game launch page and prompt for input
Select opponent type
AI Opponent
Create game and AI
Human Opponents
Display list of friends
Select friends to invite to game
Send game invites to selected friends
For each friend
Display game invite popup
Accept
Add player to game
Decline
Remove popup
Display game
Launch game

State Diagrams

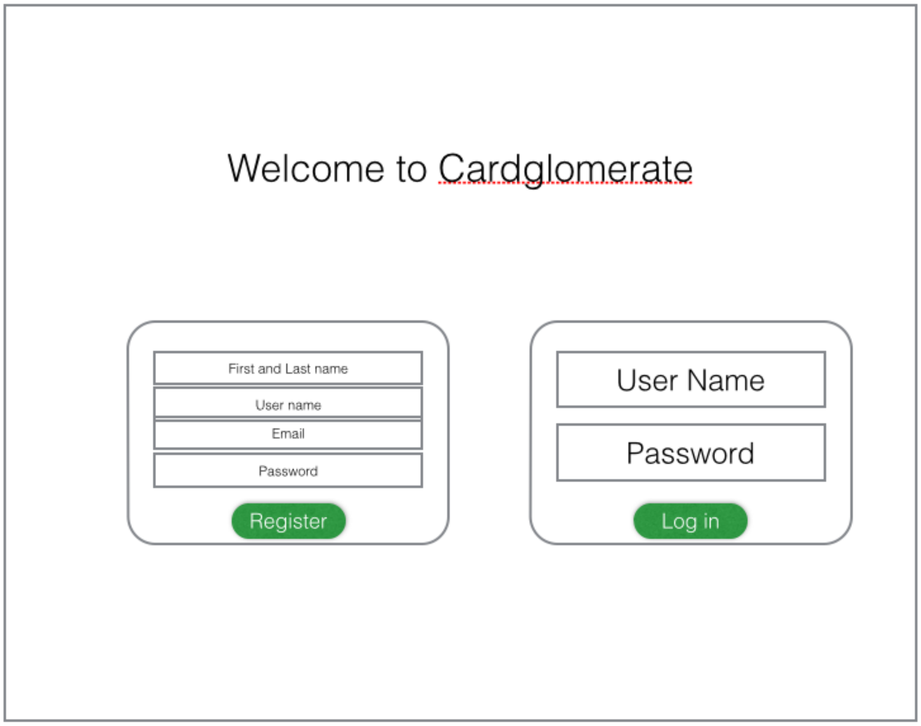
Sample Texas Hold’Em Turn

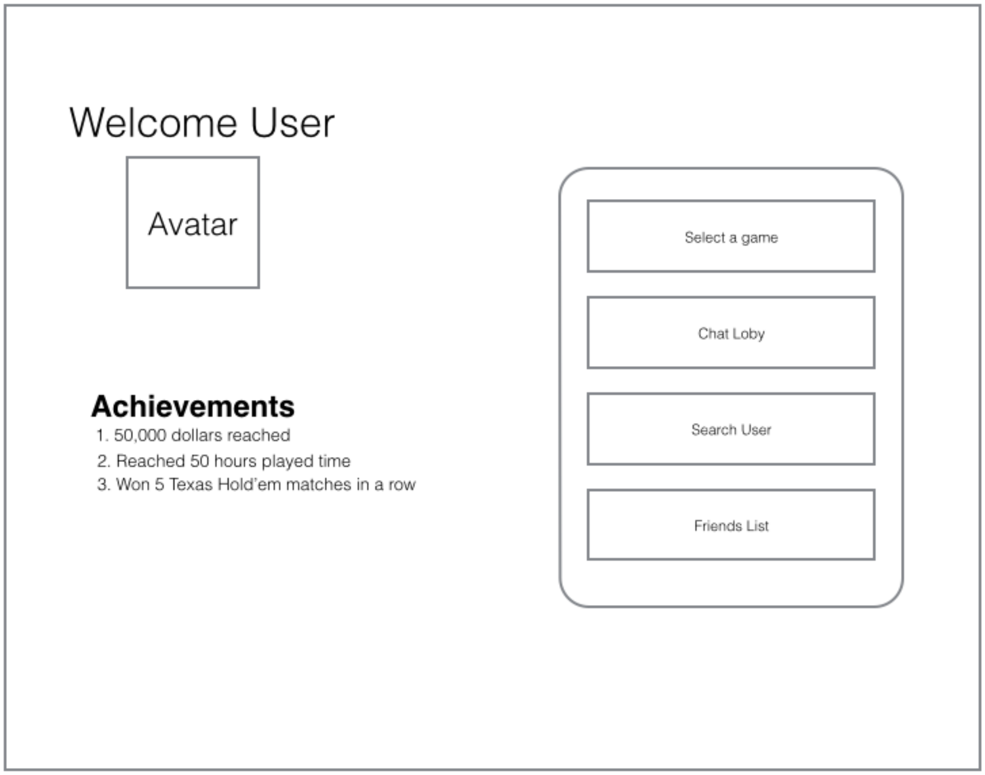


Sample Card Draw



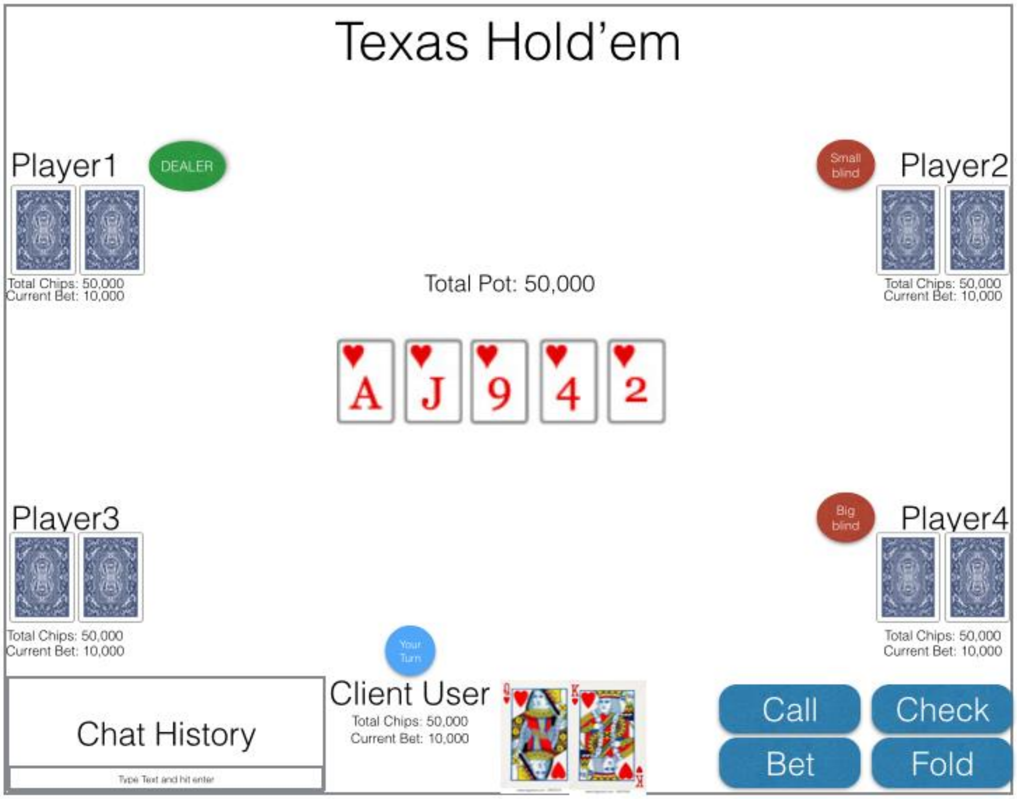
User interface mockups

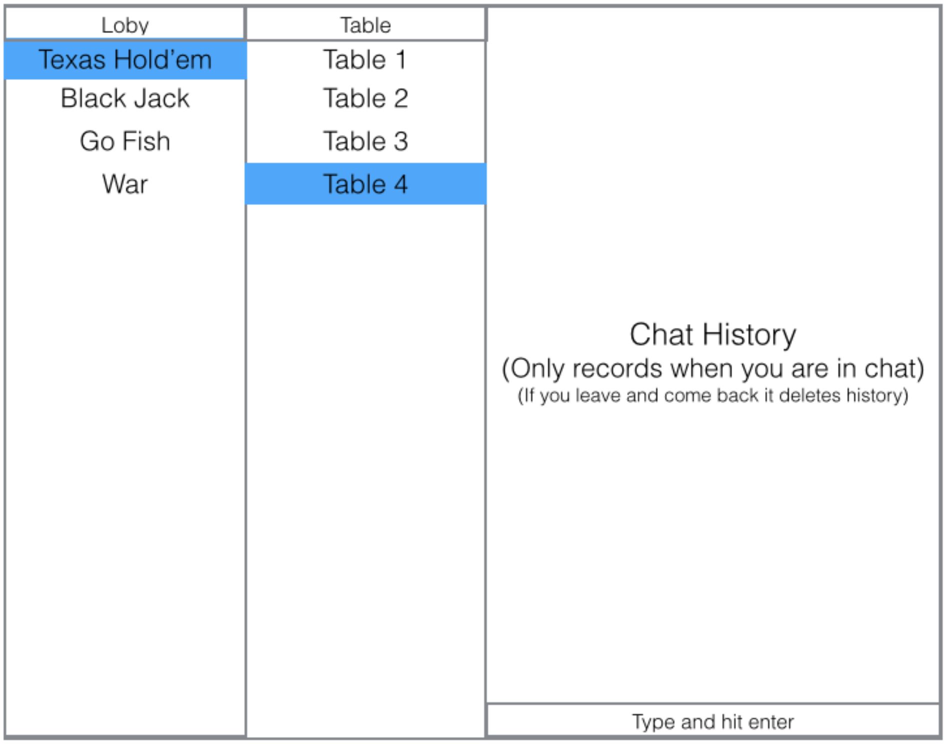




Achievements

1. 50,000 dollars reached
2. Reached 50 hours played time
3. Won 5 Texas Hold'em matches in a row





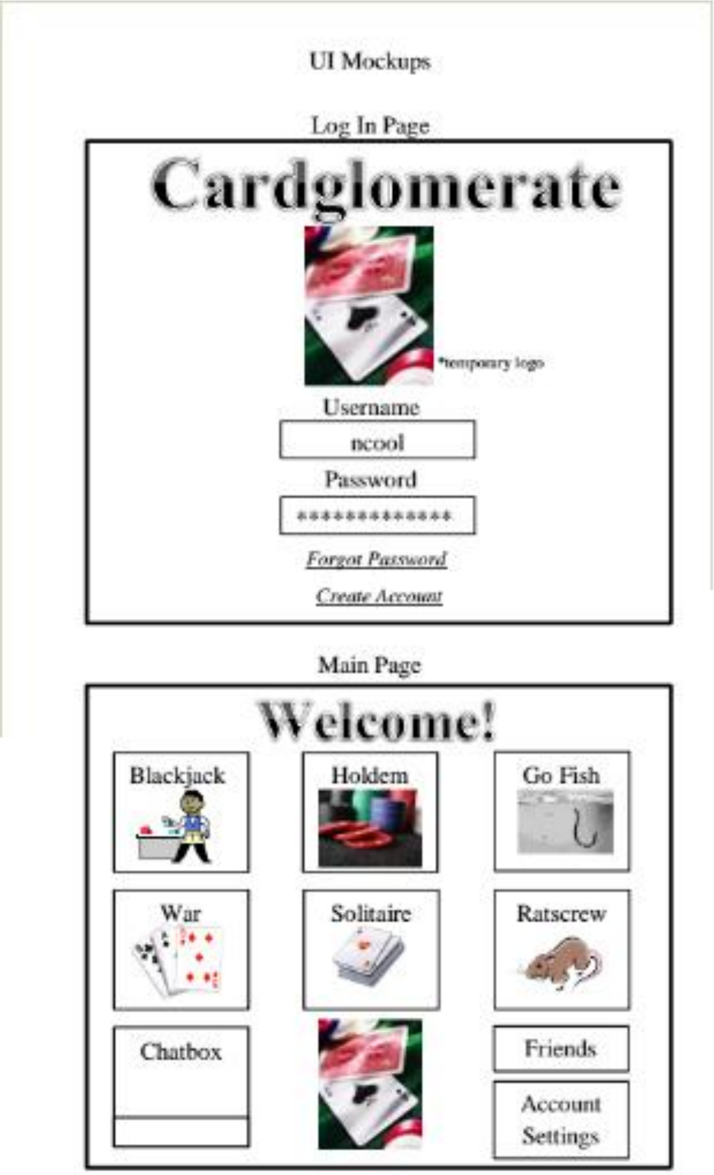
(Only records when you are in chat)

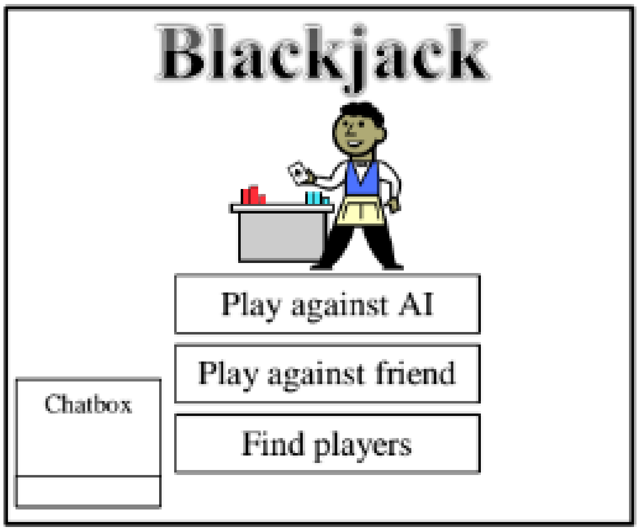
(If you leave and come back it deletes history)

Alternative UI Mockups

UI Mockups

Log In Page


Cardglomerate
*temporary logo
Username
ncool
Password
****
Forgot Password
Create Account
Main Page
Welcome!
Blackjack
Holdem
Go Fish
War
Solitaire
Ratscrew
Chatbox
Friends
Account Settings


Blackjack
Play against AI
Play against friend
Find players
Chatbox