Cards Against Humanity Mobile Game

Design Document

Team Members

Christina Atallah, Kalena Galarnyk, James Miller, Eric Templin

Table of Contents

Purpose

Our design document will describe our project and specify our design choices. We want to give an adequate view of the system by providing class diagrams, details of system choices, and more. Our software will allow the users to play “Cards Against Humanity” on their mobile phones with people nearby without the physical deck. It will emulate the experience of the actual game as closely as possible.

Game Rules

Our project models a real life card game, and all card games have rules. Before reading any further into this design document, it is imperative that you read the official “Cards Against Humanity” rules which can be found here: http://s3.amazonaws.com/cah/CAH_Rules.pdf. Understanding the rules of the card game will help convey our choices in wording such as the term “Czar” for current card reader. This document assumes that the reader is intimately familiar with the game already.

Summary of Functional Requirements

- Design a client server architecture to be able to build multiple clients (fat-server as it's a long test cycle on Android clients)
- Scalability in the sense that the interface is resizeable so that the application can appear properly on any Android screen.
- Enable the application to handle action events: scroll over cards, select card, submit card.
- Have a variety of white and black cards necessary for game play.
- Display the current black question card, current players, and current "Czar", and the player's own deck of white cards.
- User should be able to create a new game/table.
- User should be able to join an existing table.
- User should be able to exit a game without causing a disruption.
- Current "Czar" must be able to view card submissions and select a winning white card.
- Must keep track of how many cards are in each deck, and how many black cards each player has won.
- Ability to handle exceptions and edge cases to avoid crashing (This is a functional requirement b/c highly valued and necessary for mobile applications to succeed).
- Ability to play next to friends so that the game is true to its original form.
- Be able to choose as an option at the beginning of the game how many black cards must be collected to be considered the victor.
- Ability to custom-make cards.

General Priorities

- Usability

  The interface designed in a fashion that is user friendly and easy to access all functionality of the game is a straightforward fashion. Users will have minimal barriers to understand how to play on the device.

- Accessibility

  We have designed our system for Android versions 2.2 and up so that our application is still accessible to older phones. Multiple clients should be able to play a game at the same time.

- Performance

  Clients should be able to interact with server and receive information in a short amount of time.

- Stability

  Our application should be robust and handle exceptions and edge cases so that the application does not crash during use.

Design Issues

- **Fat Client vs. Fat Server**
    - When thinking of how to approach the design we considered both a “fat client” or a “fat server”. The primary benefit of a “fat client” is turning the game into a distributed system capable of operating without a master server. Although due to the primary drawback of maintaining and synchronizing state changes between an unknown number clients prevented us from going with the “fat client” pattern.
    - Instead we choose the “fat server” pattern with a majority of the applications logic being processed on the server. The primary benefit of the “fat server” pattern is the ease of keeping state consistent between actions. With only one authority responsible for updating the game state the game logic is greatly simplified.

- **Client-Server Protocol**
    - We chose JSON as our client-server protocol format. JSON is human readable and available in a majority of programming languages on most platforms.
    - Our application sends very little data and the ease of using JSON on both the client and the server outweighs the bandwidth cost of sending ASCII text.

- **Client Platform**
    - When developing applications for Android, it is important to consider which version of Android you would like to target. Picking a newer version of Android, such as Jelly Bean, offers more API features but less people will be able to use the application because half of Android devices are still running Gingerbread or earlier versions of Android. We chose to target Android 2.2 (Froyo) because 97% of Android devices are running a version equal to or newer than that version.

Design Outline

- Architecture:

  We are using a Client-Server architecture where the server handles as many clients as it has resources for.

- Server:

  The server is written in Go and makes use of the languages coroutines.

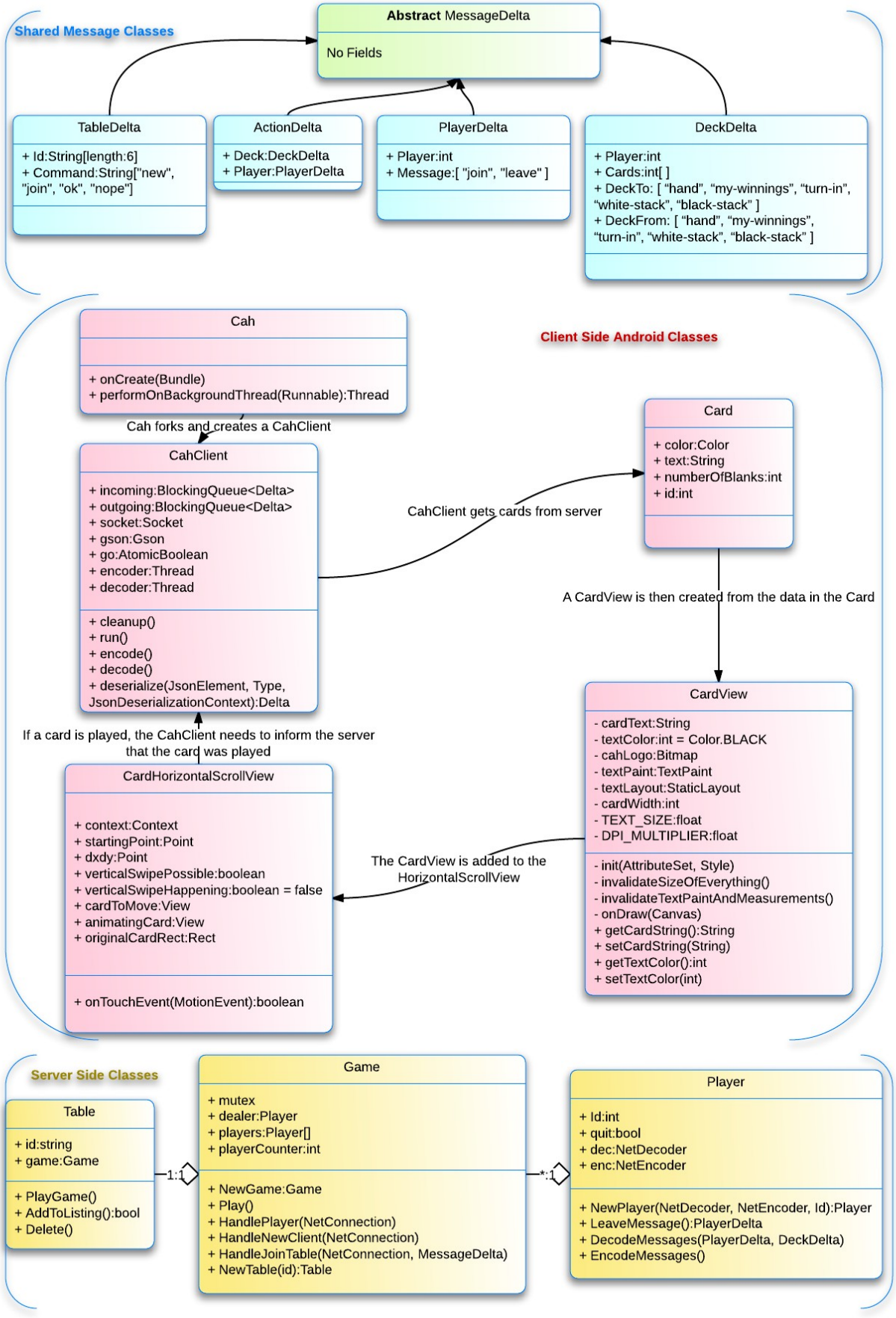
- Client:

  The first client application will be written in Java for Android.

- JSON:

  To transmit information between the client and the server we will use JSON (JavaScript Object Notation). JSON is designed to be a human-readable means of exchanging data. Both the client and server will encode and decode JSON messages that contain the game's state changes.

Client-Server Class Diagram



Client-Server Design

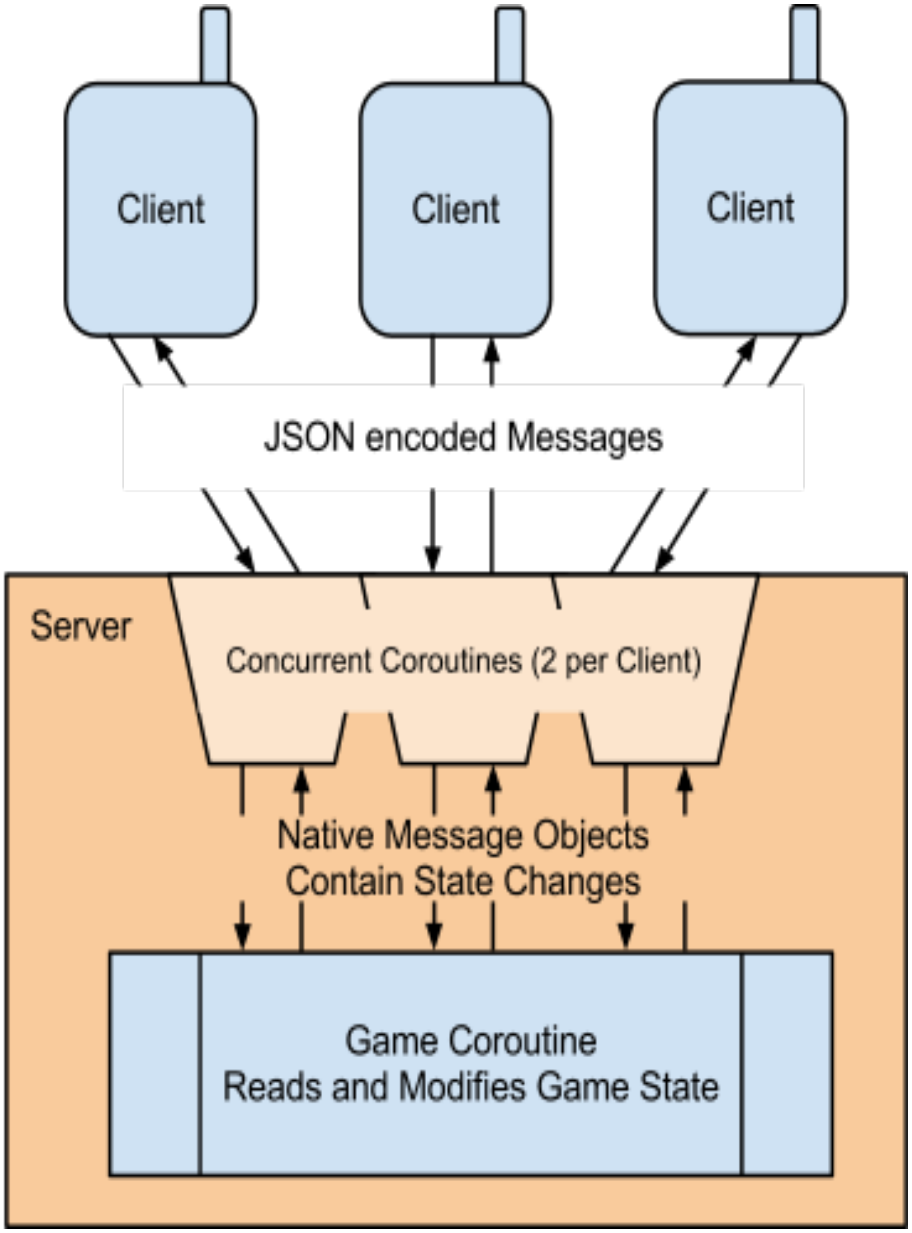


Figure 1: The server process hosts concurrent coroutines that include 2 per connected client plus 1 coroutine receiving messages. The client coroutines encode and decode JSON messages both receiving messages from the game coroutine and sending the game coroutine messages from the client.

Server's Logic of Game Flow

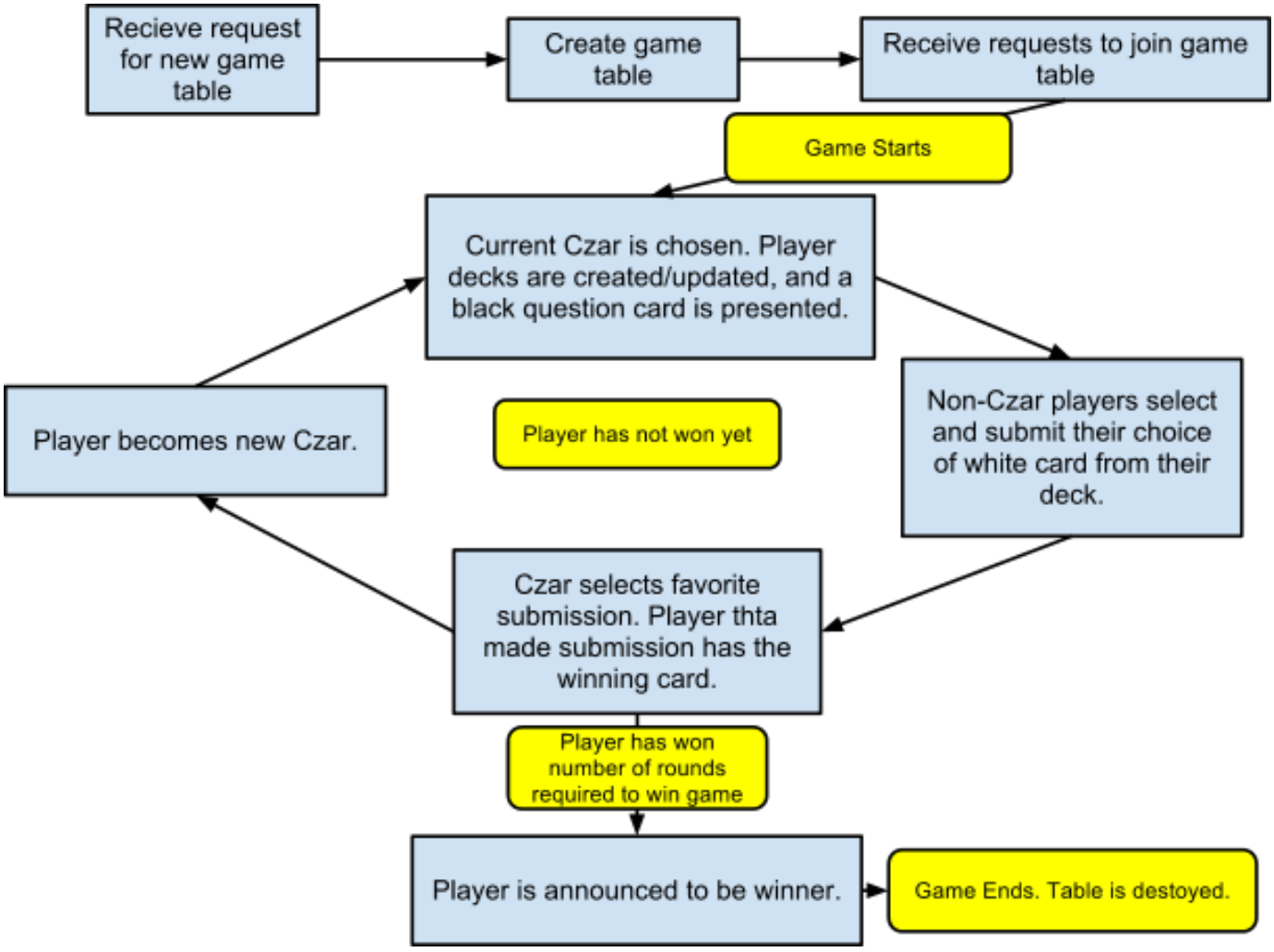


Figure 2: Activity Diagram of Game

The multi-player game involves a question card being chosen from a black deck, to which the players choose what they feel is the most amusing response from their hand of white cards. The player that chose the black question card will judge these white response cards and pick his or her favorite one. The player that submitted that winning white card becomes the game's "Czar", and collects that black question card to signify their victory in that round. At the end of the game, the individual that has collected the most black cards is the winner.

Message Design - Passes state between clients and server

Deltas are broadcast to all players and allows the client GUI to render self and other player actions accurately. These classes are shared between client and server code bases.

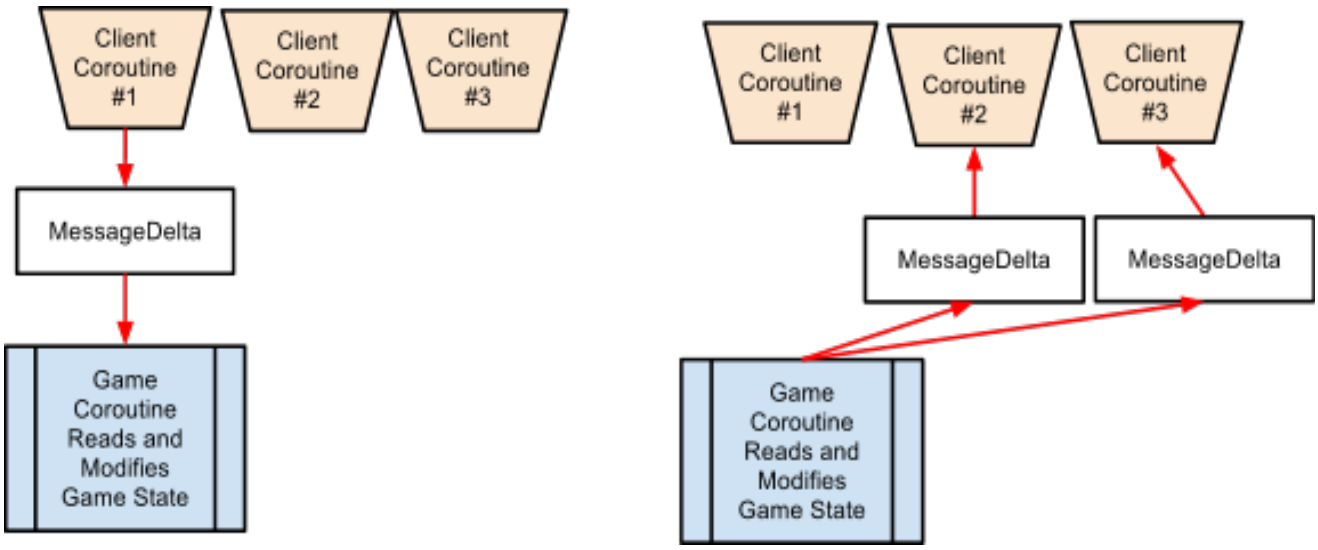


Figure 3: MessageDelta's are broadcast to all clients notifying them of state changes to be rendered on the client.

Shared Classes, Available to both Client and Server

Both client and server expect a single:TableDelta followed by infinite:{PlayerDelta or DeckDelta}'s until the game ends.

TableDelta
Id: String[length:6]
Command: String
Legal Values for Command:
"new", "join", "ok", "nope"

Table deltas are used to ask the server to create or join a specific table with the associated ID. If the "new" command is used, the server creates a random ID of six characters for clients to use.

PlayerDelta
Id: int
Message: String
Legal Values of Message:
"join", "leave"

Player deltas are sent to all players when another player joins or leaves the table. The ID indicates which player has left, joined or rejoined the table.

DeckDelta
Player: int
DeckTo: String
DeckFrom: String
Legal Values of DeckTo & DeckFrom:
"hand","my-winnings", "turn-in"
"white-stack","black-stack"
Cards: int[ ]

The game operates as a state machine receiving deck deltas. A deck delta represents moving cards from one source to another. A deck delta of {DeckTo: hand, DeckFrom: white-stack, Cards: { 74, 135 }} indicates said player drawing 2 cards with Id's 74,135 and inserting them into his or her hand.

ActionDelta
Deck: DeckDelta Player: PlayerDelta

The action delta is a wrapper class used to parse both a DeckDelta or PlayerDelta from the same JSON input stream. Since JSON libraries can only decode a single class at a time we must use a wrapper class to conditionally accept either a DeckDelta or PlayerDelta at the same time. It is an error to send an Action Delta with both Deck & Player set.

Client Side Classes

Cah extends *Activity*
in : Queue<MessageDelta> out : Queue<MessageDelta>
onCreate() performOnBackgroundThread()

Cah is the primary Android Activity class and is instantiated by the Android OS.

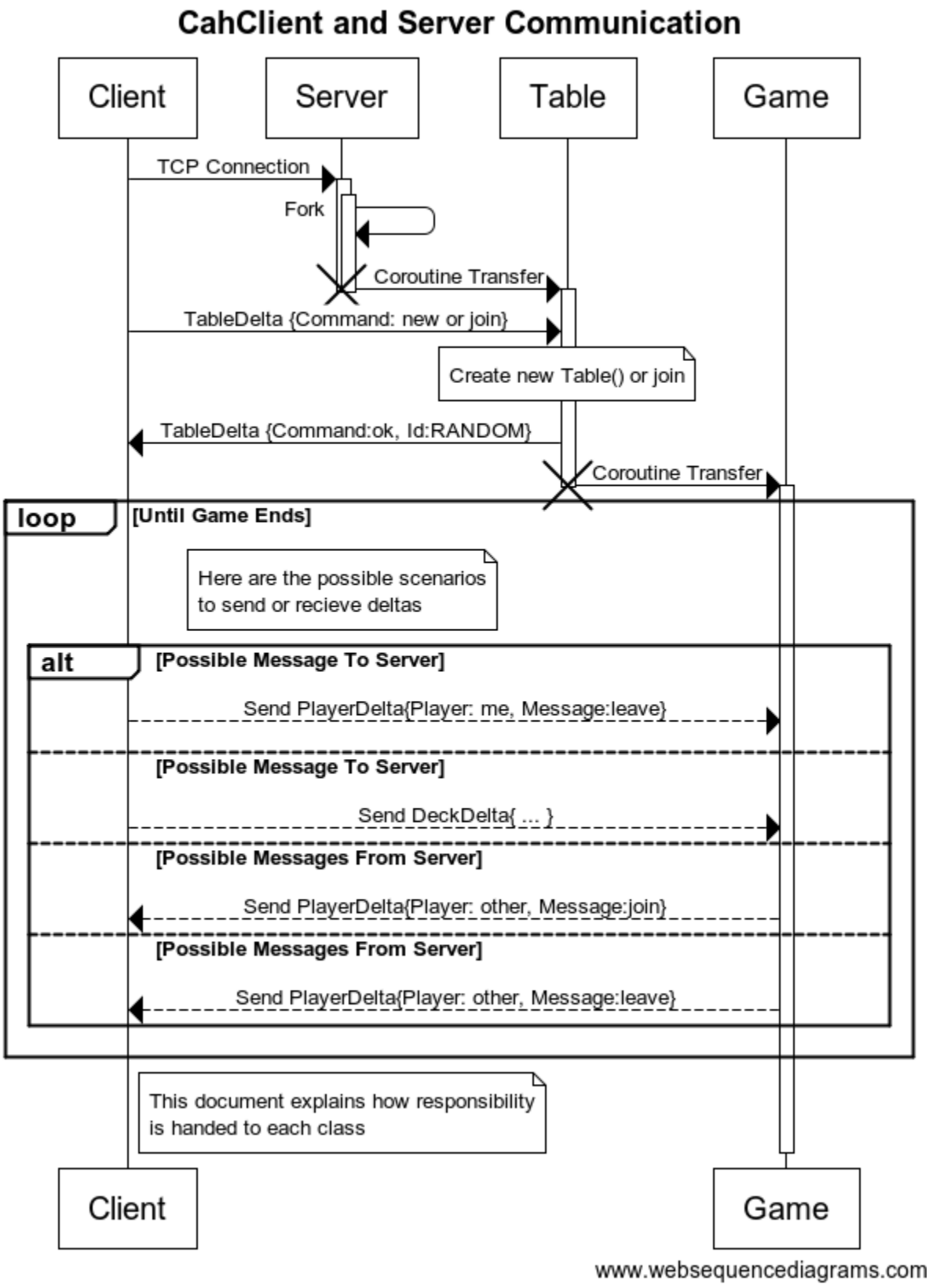
CahClient implements *Runnable*
incoming: Queue<MessageDelta> outgoing: Queue<MessageDelta>
run()

CahClient implements the Runnable interface and is a self contained thread that communicates MessageDelta's to a Cah Server. CahClient opens a socket to a CahServer. Incoming player actions are read from the incoming MessageDelta queue, encoded to JSON and sent to the server. Cah Client creates another internal thread that decodes JSON responses from the server and sends them as native Java objects on the outgoing MessageDelta queue.

Sequence Diagram

Client Server Diagram: Handling Join/Create Requests

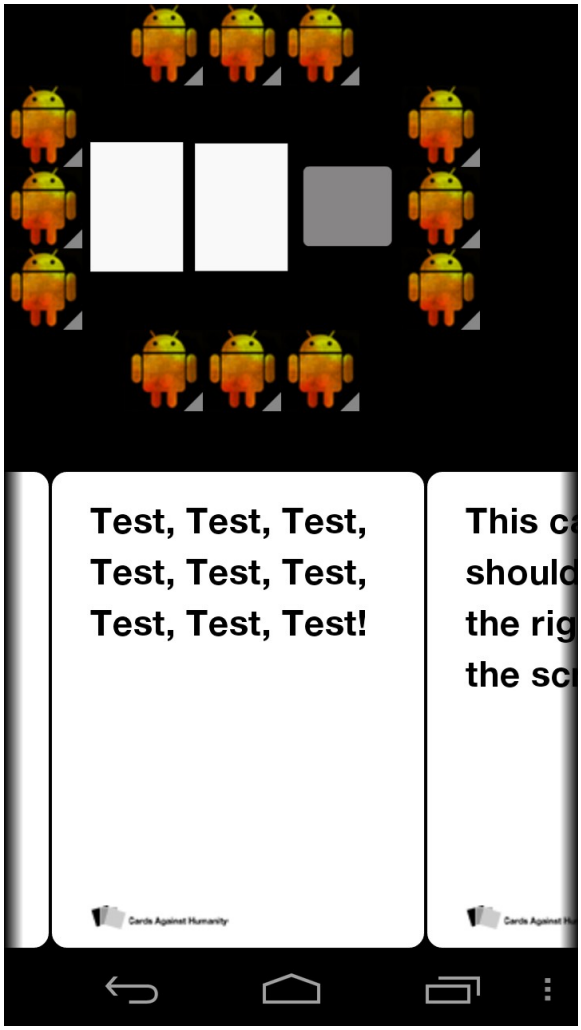
The preceding diagram depicts the communication between the Client and the Server when creating and joining tables. The flow chart depicts actions and the decisions made based on those actions.

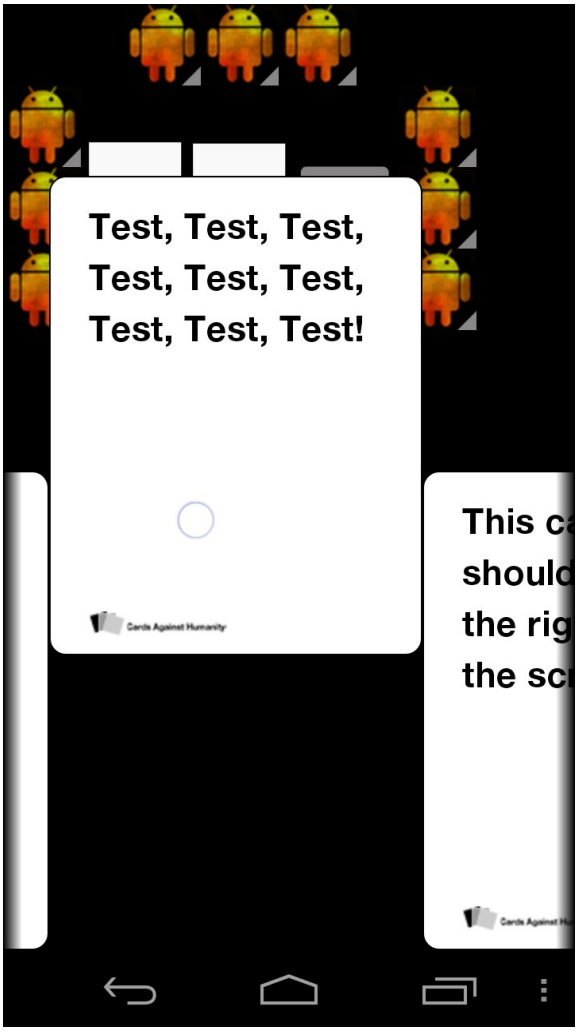

CahClient and Server Communication
Client
Server
Table
Game
TCP Connection
Fork
Coroutine Transfer
TableDelta {Command: new or join}
Create new Table() or join
TableDelta {Command:ok, Id:RANDOM}
Coroutine Transfer
loop
[Until Game Ends]
Here are the possible scenarios to send or recieve deltas
alt
[Possible Message To Server]
Send PlayerDelta{Player: me, Message:leave}
[Possible Message To Server]
Send DeckDelta{ ... }
[Possible Messages From Server]
Send PlayerDelta{Player: other, Message:join}
[Possible Messages From Server]
Send PlayerDelta{Player: other, Message:leave}
This document explains how responsibility is handed to each class
Client
Game
www.websequencediagrams.com

User Interface Analysis

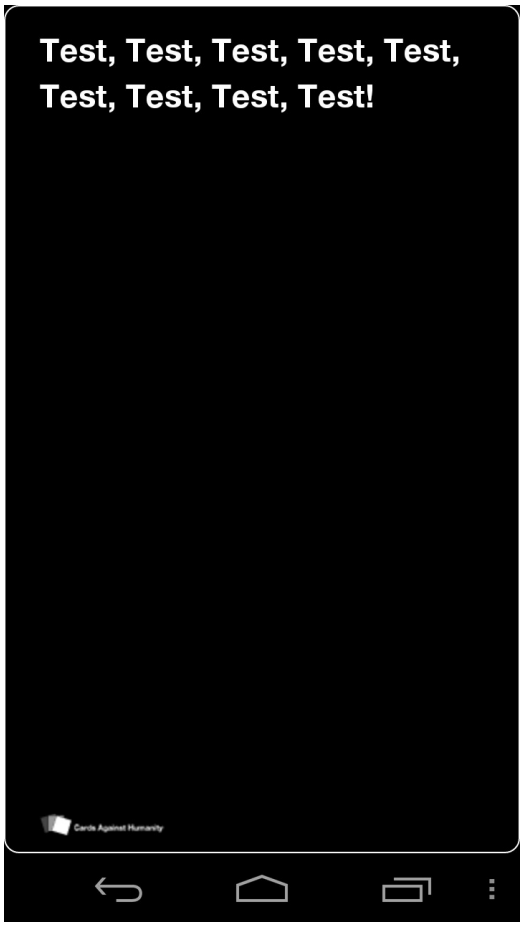
LINK TO LIVE DEMO: http://www.cs.purdue.edu/homes/etemplin/Cah.apk

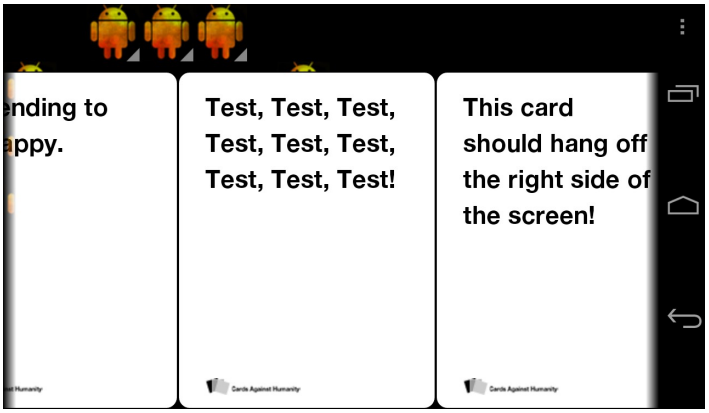
This is a live demonstration of our user interface as a minimum viable product. It is an experimental throw-away program used to exercise our thoughts and to get a better idea of how the app will function. It is not a working game and only a demonstration of our UI principles.





Screenshot 1: Main gameplay UI. Android icons will be user's pictures.	**Screenshot 2:** Card moved towards table by dragging it upwards.





Screenshot 3: Black card is shown full screen on the Czar's phone while waiting for other player's card selection.	**Screenshot 4:** An alternative way to choose which card you would like to play.