Phase 1 Software Progress Report

Card Czar Android App

CMSC 495

Group 2 Final Project

Kenneth Mikkalson

Alton Hinton

Shawn Henson

Sarah Holley

Tara Lawson

Richard Wysong

Table of Contents

[1.0 Schedule and Milestones 3](#_Toc430531457)

[1.1 Milestone Progress 3](#_Toc430531458)

[1.2 References 3](#_Toc430531459)

[2.0 Lessons Learned 4](#_Toc430531460)

[2.1 Barriers and Resolution 4](#_Toc430531461)

[2.1.1 Development Environment 4](#_Toc430531462)

[2.1.2 Developer Education 4](#_Toc430531463)

[2.1.3 Android Communication 4](#_Toc430531464)

[2.1.4 Apache HttpComponents™ 5](#_Toc430531465)

[2.1.5 Intent Crashes 5](#_Toc430531466)

[2.1.6 Configuration Management System 5](#_Toc430531467)

[2.1.7 Threading 5](#_Toc430531468)

[2.2 Project Reevaluation 6](#_Toc430531469)

[3.0 Phase 1 Functionality 7](#_Toc430531470)

[3.1 Functionality Description 7](#_Toc430531471)

[3.1.1 Overall Application Description (end goal for the project) 7](#_Toc430531472)

[3.1.2 Phase 1 Functionality 7](#_Toc430531473)

[3.2 Application screenshots of sample runs 8](#_Toc430531474)

[3.2.1 Non-dealer test run 8](#_Toc430531475)

[3.2.2 Dealer test run 13](#_Toc430531476)

[3.3 Source Code 19](#_Toc430531477)

[3.3.1 Source Code Overview 19](#_Toc430531478)

[3.3.2 Getting Started – User to Gameplay 20](#_Toc430531479)

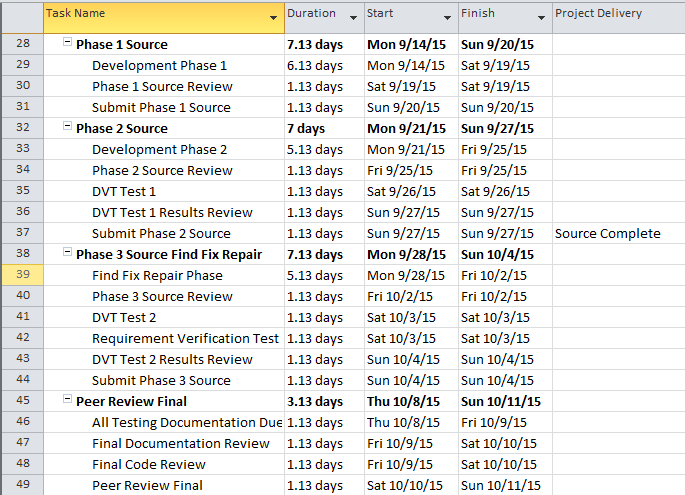
[3.3.2 Getting Started – Host to Gameplay 23](#_Toc430531480)

[3.3.3 Gameplay – GameplayActivity onCreate 24](#_Toc430531481)

[3.3.4 Gameplay – Turn 25](#_Toc430531482)

[3.4 Build Process 27](#_Toc430531483)

# 1.0 Schedule and Milestones



## 1.1 Milestone Progress

All previous milestones related to documentation and planning have been accomplished on time or ahead of schedule. Weeks 5-7 will include three phases of source code development. Fully functioning software is expected at the end of Phase 2, reserving phase 3 source for bug fixes.

In parallel with the development effort, a portion of the team has been updating documentation based on previously provided comments by the professor and peer group review. Those documents include the Project Plan and Test Document.

## 1.2 References

* Project Plan, Card Czar Android Application
* Test Plan, Card Czar Android Application
* Software Design Document, Card Czar Android Application

# 2.0 Lessons Learned

Group two has worked through several significant roadblocks in the development of the Card Czar application mainly stemming from the sum total of zero experience with Android programming.

## 2.1 Barriers and Resolution

### 2.1.1 Development Environment

Android Studio was chosen as the development environment because of integrated smartphone emulator support. Considerable time was spent on installing Android Studio because:

* Target hardware did not have sufficient memory (the recommended 2GB was not sufficient)
* Target hardware did not have virtualization capability for the emulator (not even mentioned in the installation requirement)

With mediocre hardware, several hours were spent on installation, only to realize that the hardware was insufficient. Multiple installs on slower hardware took hours with a slow Internet connection.

Additionally, low memory availability resulted in issues getting the Intel® Hardware Accelerated Execution Manager to provide sufficient memory for the emulator. This slowed setup of the development environment in some cases.

### 2.1.2 Developer Education

It was assumed that prior Java programming skills would make for a quick transition to Android development. As it turned out, gaining the prerequisite knowledge took considerable time. The developers watched about 9 hours of YouTube videos on Android development to close that gap.

### 2.1.3 Android Communication

It was assumed that, with Android OS being targeted primarily for the smartphone market, communicating with Android would likely be easy. We found that it wasn’t as communication-friendly as we initially believed. Days were spent experimenting with:

* Bluetooth, which is not supported by Android Studio emulators
* Direct network communications
* Google Play (this stopped early at the $25 fee for setting up a certain account with Google)

Fortunately, a team member stumbled across a helpful IBM developerWorks article by Jack D. Herrington that described use of the Apache HttpComponents™ libraries: <http://www.ibm.com/developerworks/library/x-androidtictactoe/>

From there, it was decided that a LAMP server would be used for communications, though it was soon found out that this entailed problems of its own.

### 2.1.4 Apache HttpComponents™

Though code from the developerWorks article ran fine, using legacy libraries seemed less than ideal. A considerable number of hours were spent trying to use more up-to-date Apache libraries. One member went so far as to uninstall Android Studio and install an older version after reading about issues with the API23 version of Android Studio. Various Apache libraries were tried with various commands in various API configurations, and in the end, the team was only able to get http communications working with the older http libraries using older command sets.

### 2.1.5 Intent Crashes

At first, code for intents was adapted from the YouTube series “Android Apps Development Tutorial for Beginners” by Bucky Roberts. The underlying cause of the crashes was never pinpointed, but the first intent implementation in Card Czar led to the app crashing. To alleviate this, typical Java onClick Listeners were implemented. As the number of Activities grew, it was found that this too would lead to infrequent crashes. Eventually, a standard Android methodology was employed and the intent crashing went away entirely, but it was one of the larger bugs that took a notable amount of time to solve.

### 2.1.6 Configuration Management System

Though GitHub is a feature-rich configuration management system, it did present some delays due to the team’s inexperience with the website. The first repository created was empty—members wasted time trying to deposit codebases into the empty repo, but there never was a solution found for doing this with Android Studio.

Additionally, we stumbled with trying to perform actions on an individual-created repo—it took some time to figure out that a group repository needed to be made to perform these actions.

While the time spent overcoming these obstacles was minimal compared to many of the problems we encountered, it nonetheless slowed progress.

### 2.1.7 Threading

A considerable amount of time was spent trying to get threading to work properly. Initial designs included a thread called Turn, which would call a subsequent thread for performing communication actions with the LAMP server. Unable to get a working solution, the nested thread design was abandoned and the team focused on simply trying to get the Turn thread to work (it would have been easier yet to run the turn interactions in the main thread, however, this was seen to be extremely poor Android application design).

After considerable frustration, it was found that problems with wait() and notify() methods vanished by using “extends Thread” instead of “implements Runnable” for the Turn class.

## 2.2 Project Reevaluation

Despite considerable setbacks, the team generally stayed the course.

The decision to create an Android app, despite the lack of experience, made for a challenging project, but the difficulties encountered provided substantial learning opportunities. The large scope was initially intimidating, but after overcoming the many roadblocks, the team is satisfied that the project is of appropriate scope for the class.

The only notable decision the team chose not to pursue was the use of Bluetooth for Android communication. Though the use of Bluetooth was abandoned due to apparent time constraints, the use of a stable central database alleviated concerns about the host quitting, as the LAMP server provides consistent, accessible data. Additionally, PHP provided powerful tools that simplified coding in several cases.

# 3.0 Phase 1 Functionality

## 3.1 Functionality Description

### 3.1.1 Overall Application Description (end goal for the project)

The Card Czar Android application is a spin-off of the popular card game Apples to Apples (drawing some inspiration from the similar card game, Cards Against Humanity).

The Card Czar is an android application multiplayer card game that can deliver hours of fun.  It allows a user to create a new game room and invite friends to play with them. For a game to begin there must be at least 3 players. Once a game is started each player is dealt at random a hand of response responses that will be used to respond to the random bait that is chosen for the round. A bait card (text) is chosen and the Card Baron (this is the dealer—either the winner from the last round, or host if it’s the first round) then waits for players to submit responses to that bait. The Card Baron (dealer) then selects from one of the responses and the player who submitted the winning response gains 1 point and becomes the Card Baron for the next round. The first player to get to 5 points wins and is crowned the Card Czar.

### 3.1.2 Phase 1 Functionality

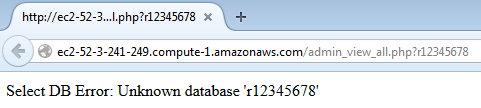
This section describes the functionality being delivered as of Phase1. All of the application’s activity layout screens have been coded and are working, except for GameplayActivity, which only handles gameplay while a player is a dealer (Card Baron). Note that the other completed Activity Java classes have only the level of functionality necessary for bare minimum operation. Additionally, the LAMP server has been created and deployed to an AWS instance. This server hosts the middleware, PHP files, which the application will call to interact with the database. The PHP files have not all been completed—not all of the non-dealer actions have been coded. The delivered code for phase 1 can be compiled and run through an emulator to begin showing basic gameplay functionality and interaction with the LAMP server.

The gameplay functionality is not complete yet so a full game cannot be played without using PHP tools that were designed for testing. Full functionality should be completed in the next phase of development. Additionally, security, such as setting up HTTPS communication with the LAMP server, has not yet been implemented in this phase of development. A breakdown of a sample game follows.

## 3.2 Application screenshots of sample runs

### 3.2.1 Non-dealer test run

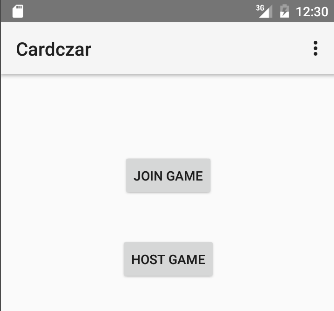
1. Using one of the PHP admin tools (not part of the finished product), check to see that there is no database for the room that we will create:

  
Figure 3.2.1.1: admin\_view\_all.php output

1. Next, an API23 emulator is started:

  
Figure 3.2.1.2: Selecting an appropriate emulator in Android Studio

1. The initial menu to host or join a game is displayed.

  
Figure 3.2.1.3: Main Display when app starts

1. To create a game for the non-host, we must first host a game (the user cannot get to the GameplayActivity without a valid room (database). To do this, we first click HOST GAME and the following screen appears:

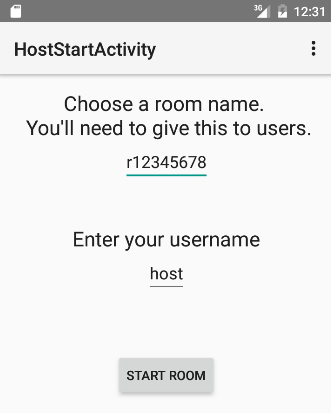
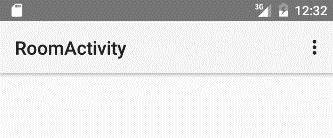
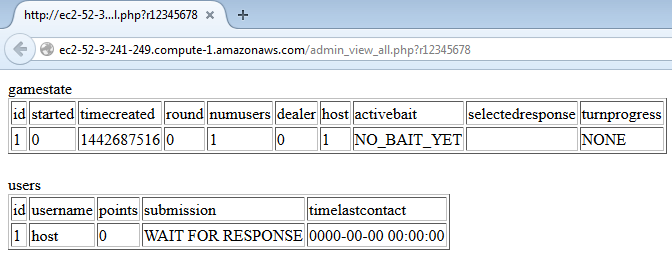


Figure 3.2.1.4: Main Display when app starts

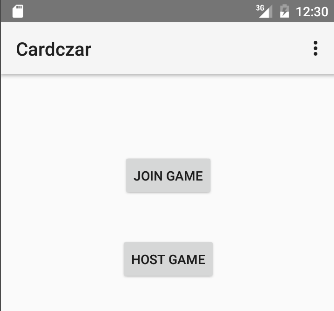
1. By clicking START ROOM, the following screen appears:

  
Figure 3.2.1.5: RoomActivity display

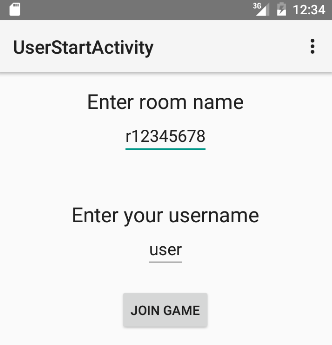
1. Room creation is verified by using admin\_view\_all.php. Notice that the game has not started (started = 0).

  
  
Figure 3.2.1.6: Admin view before the game has started

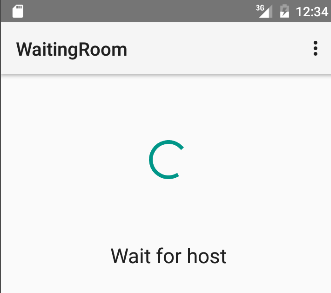
1. By clicking on the back arrow twice, we can return to the initial screen.

   
Figure 3.2.1.7: Returning to the Main display

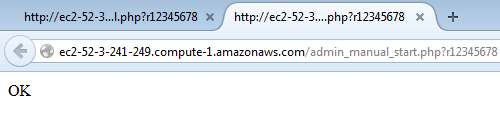
1. Clicking JOIN GAME shows the UserStartActivity:

   
Figure 3.2.1.8: UserStartActivity

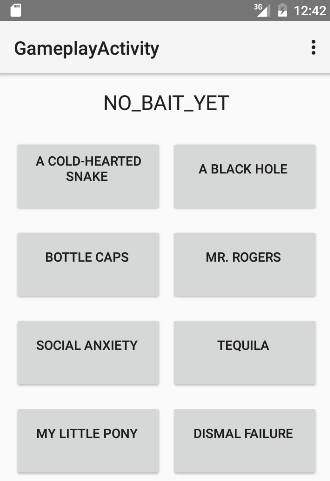
1. Clicking JOIN GAME shows the WaitingRoomActivity which shows a progress bar cycling until the game is started by the host:

   
Figure 3.2.1.9: WaitingRoomActivity

1. To start the game without a host, the admin\_manual\_start.php is used:

   
Figure 3.2.1.10: Starting the game without an emulator

1. Without any emulator input, the WaitingRoomActivity intends GameplayActivity:

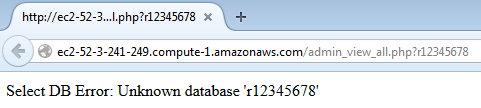
   
Figure 3.2.1.11: Sample display of non-dealer viewing responses

1. Finally, the database must be cleared manually:



### 3.2.2 Dealer test run

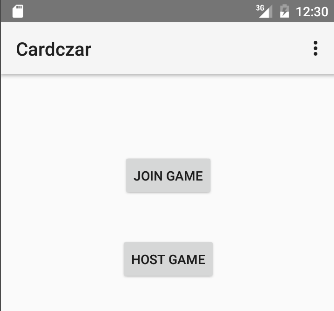
1. Check to see that there is no database for the room that we will create:

   
Figure 3.2.2.1: See that room hasn’t been created

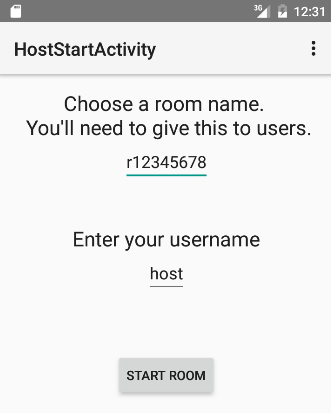
1. Start the appropriate emulator:

   
Figure 3.2.2.2: Start the API23 emulator

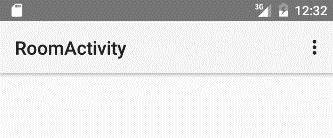
1. Run the program and the Main Activity is displayed:

   
Figure 3.2.2.3: Initial display

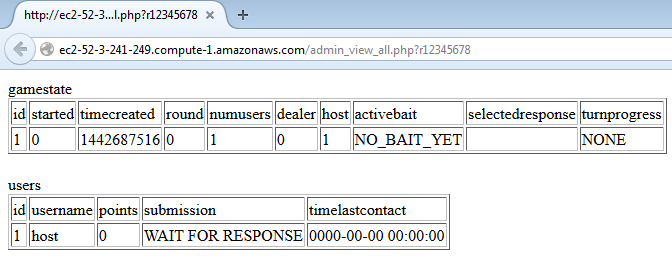
1. HOST GAME is selected, displaying the following screen:

   
Figure 3.2.2.4: Display of HostStartActivity

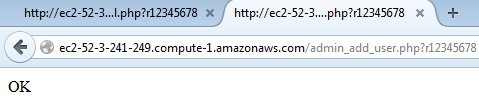
1. RoomActivity is shown, but it’s empty:

   
Figure 3.2.2.5: Empty RoomActivity

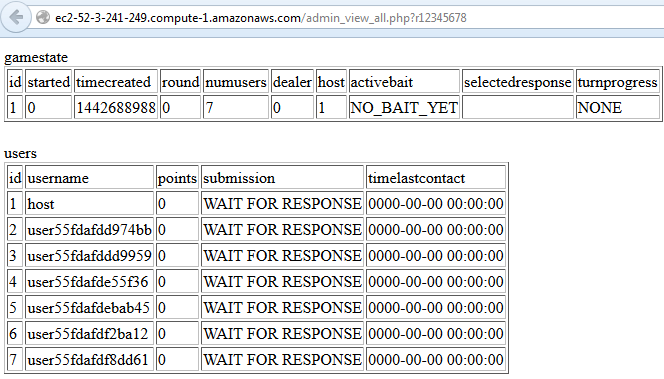
1. Looking at the database, we see that the only player is the host:

   
Figure 3.2.2.6: Database viewing reveals only one user is present

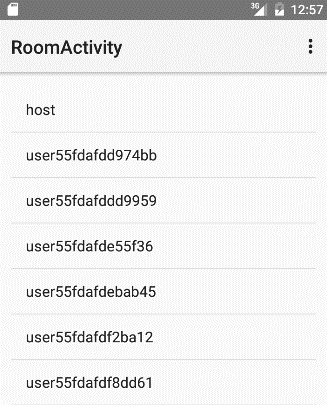
1. To add users, admin\_add\_user.php is called 6 times:

   
Figure 3.2.2.7: Adding users with admin\_user\_add.php

1. Looking at the database verifies that 6 users were inserted:

   
Figure 3.2.2.8: Verify users in database

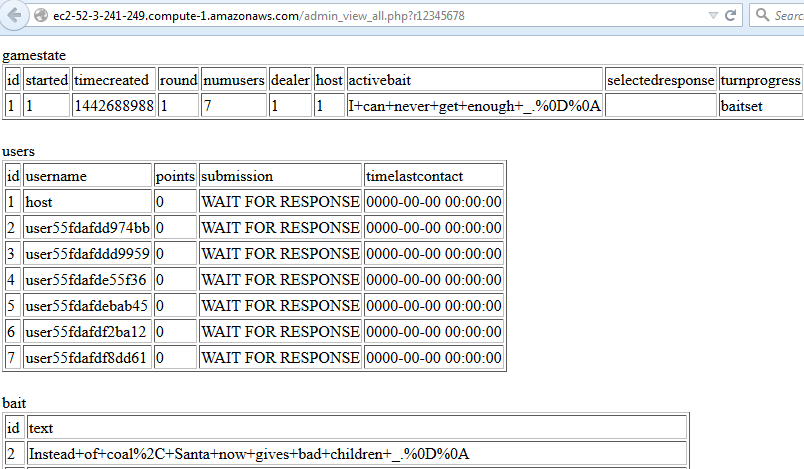
1. Pressing the refresh button, they can be seen in the emulator:

   
Figure 3.2.2.9: Verify users in emulator

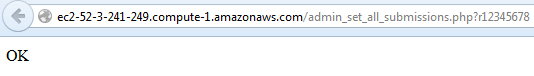
1. By clicking START GAME, the host can proceed to Gameplay Activity:

   
Figure 3.2.2.10: Dealer view of GameplayActivity

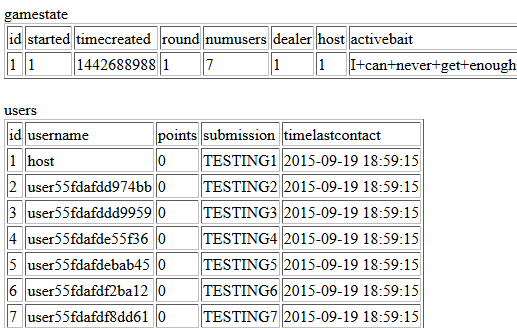
1. Looking at the database, started = 1 and bait id=1 was deleted and moved to activebait:

   
Figure 3.2.2.11: Verify game started properly in database

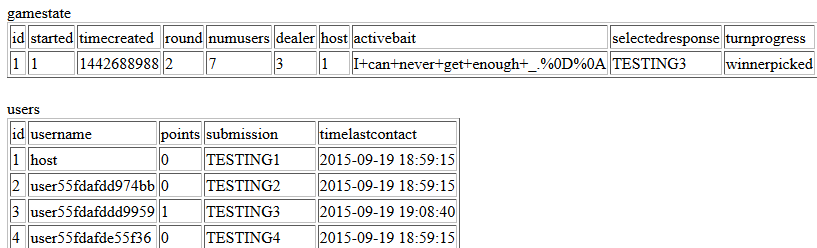
1. Set all submissions to some value so that dealer can choose a response:

   
Figure 3.2.2.12: Set submissions to TESTING#

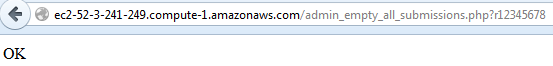
1. Verify in database that submission fields have changed:

   
Figure 3.2.2.13: Verify subsmissions

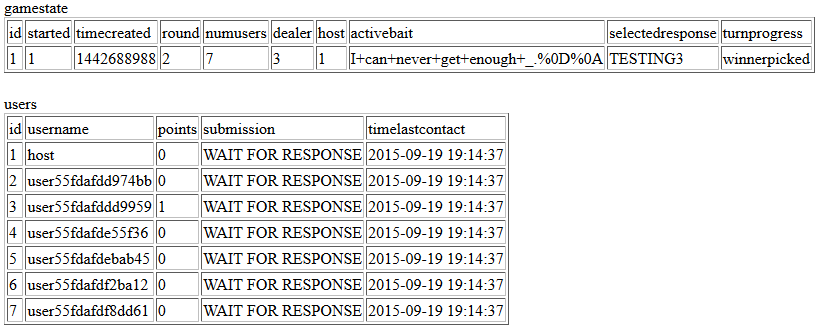
1. After selecting TESTING3, we can see in database that round winner is chosen:

   
Figure 3.2.2.14: Round is incremented, round winner points incremented

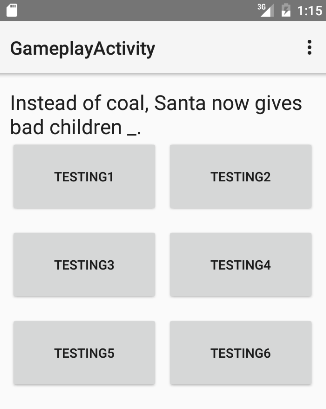
1. To continue gameplay, users normally reset their submission back to WAIT FOR RESPONSE. In this case, an admin tool does this manually to progress the game:

   
Figure 3.2.2.15: Manual setting of submissions to WAIT FOR RESPONSE

1. Verify in database that all submissions have been reset:

   
Figure 3.2.2.16 Verify all submissions are WAIT FOR RESPONSE

1. GamePlayActivity shows the new bait (see the bug? buttons weren’t changed):

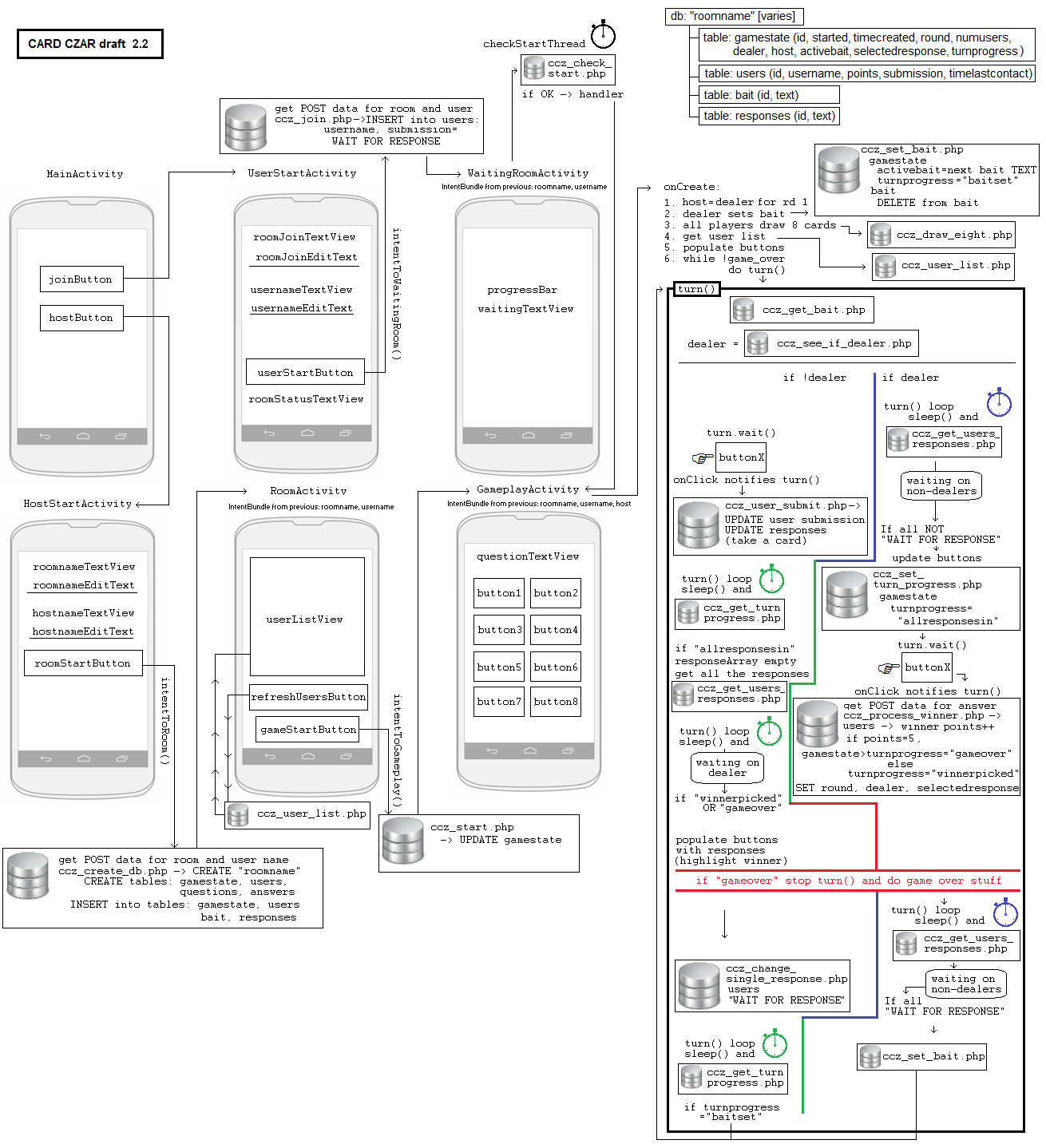
   
Figure 3.2.2.17: Emulator displays new bait without any keys pressed

## 3.3 Source Code

The phase 1 source code can be viewed at <https://github.com/cmsc495fall/Cardczar/tree/Phase1>. The Android application files are stored in the app subdirectory of this project.

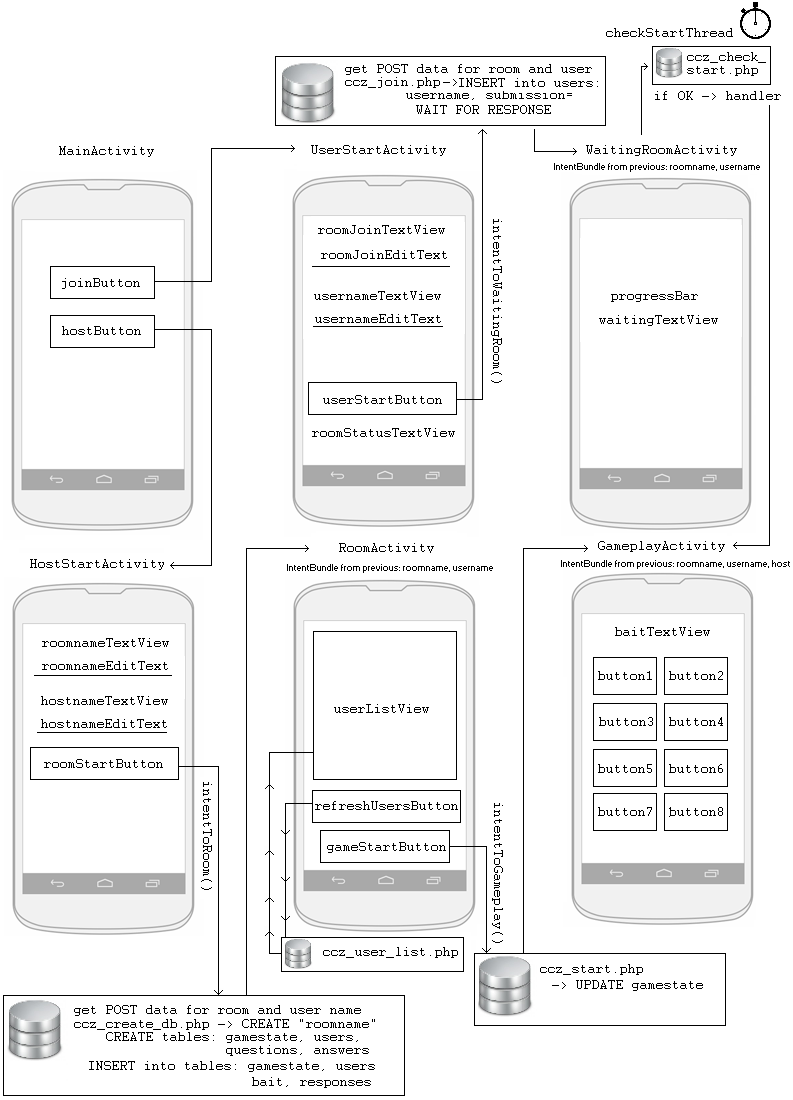
## 3.3.1 Source Code Overview

A broad overview of how the application functions can be seen below:

   
Figure 3.3.1.1: App overview

## 3.3.2 Getting Started – User to Gameplay

MainActivity is the first Activity displayed. From there, users must choose whether to be a host or non-host (referred to user in the source code). Non-hosts click the joinButton, which intends UserStartActivity.

   
Figure 3.3.1.1: App overview

Here’s the xml that maps the button to the appropriate code:

<**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="@string/joinButtonText"  
 android:onClick="intentToUserStart"**

and the method that intends the target Activity:

**public void** intentToUserStart(View view) {  
 Intent userIntent = **new** Intent(**this**, UserStartActivity.**class**);  
 startActivity(userIntent);  
}

The code is similar for all intents, though later intents pass data from Activity to Activity.

Next is the UserStartActivity. This is where the user enters the room name (which correlates to the database used for the individual game) and the username. The user clicks the userStartButton, which executes code from the Apache HttpComponents libraries:

**try** {  
 String url = **"http://ec2-52-3-241-249.compute-1.amazonaws.com/ccz\_join.php"**;  
 HttpClient httpclient = **new** DefaultHttpClient();  
 HttpPost post = **new** HttpPost(url);  
 List<NameValuePair> urlParameters = **new** ArrayList<>();  
 urlParameters.add(**new** BasicNameValuePair(**"roomname"**,roomJoinEditText.getText().toString()));  
 urlParameters.add(**new** BasicNameValuePair(**"username"**,usernameEditText.getText().toString()));  
 post.setEntity(**new** UrlEncodedFormEntity(urlParameters));  
 HttpResponse response = httpclient.execute(post);  
 **result** = EntityUtils.*toString*(response.getEntity());  
 Log.*d*(**"Result of join request"**, **result**);  
} **catch** (IOException e) { e.printStackTrace(); }

This code calls ccz\_join.php when the Button is clicked, which executes the following key pieces of code. First, data from the Activity is stored in variables:

// GET POST DATA  
$db\_name = urlencode($\_POST["roomname"]);  
$username = urlencode($\_POST["username"]);

ccz\_join.php then attempts connection to the database:

// SELECT DB  
if (mysql\_select\_db($db\_name, $link)) {  
 $connected = TRUE;  
} else {  
 echo 'Error selecting database: '.mysql\_error();  
 $connected = FALSE;  
}

If the attempt is unsuccessful, the desired “OK” output will not be returned to the Activity, and the Activity will not intend the next Activity. If it does connect successfully, the following code is executed:

// UPDATE TABLES  
if (connected) {  
  
 // ADD USER TO users  
 $query = "INSERT INTO users (username, points, quit, submission, timelastcontact) VALUES ('".$username."', 0, FALSE, 'WAIT FOR RESPONSE', ".time().");";  
 if (mysql\_query($query, $link)) {  
 echo "OK";  
 $inserted = TRUE;  
 } else {  
 echo 'Error: ' . mysql\_error() . "\n";  
 $inserted = FALSE;  
 }  
  
 // INCREMENT NUMUSERS IN gamestate  
 if ($inserted) {  
 $tablecontents = mysql\_query("SELECT \* FROM gamestate");  
 if ($myrow = mysql\_fetch\_array($tablecontents))  
 $numusers = $myrow["numusers"];  
 $numusers++;  
 $query = "UPDATE gamestate SET numusers=".$numusers." WHERE id=1; ";  
 mysql\_query($query, $link) or die("Update DB Error: ".mysql\_error());  
 }  
}

After the user is added to the user table and gamestate is updated, ccz\_join.php returns “OK” and UserStartActivity intends WaitingRoomActivity. WaitingRoomActivity loops via the CheckStartThread class:

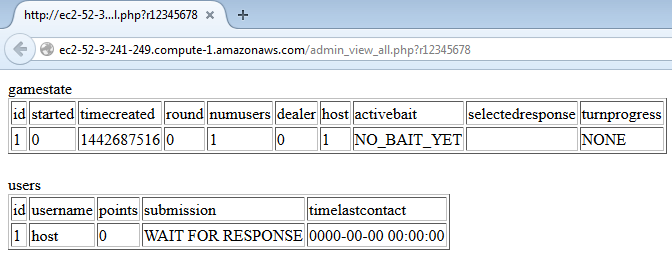
**class** CheckStartThread **implements** Runnable {  
 **private volatile boolean running** = **true**;  
  
 @Override  
 **public void** run() {  
  
 **while** (**running**) {  
 **try** {  
 Thread.*sleep*(1210);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 *// Get gamestate:started value from LAMP* **try** {  
 String url = **"http://ec2-52-3-241-249.compute-1.amazonaws.com/ccz\_check\_start.php"**;  
 HttpClient httpclient = **new** DefaultHttpClient();  
 HttpPost post = **new** HttpPost(url);  
 List<NameValuePair> urlParameters = **new** ArrayList<>();  
 urlParameters.add(**new** BasicNameValuePair(**"roomname"**, **roomname**));  
 post.setEntity(**new** UrlEncodedFormEntity(urlParameters));  
 HttpResponse response = httpclient.execute(post);  
 **result** = EntityUtils.*toString*(response.getEntity());  
 …

When the gamestate variable started=1 (true), then GamePlayActivity is intended via a handler:

**handler**=**new** Handler(){  
 @Override  
 **public void** handleMessage(Message msg) {  
 Intent roomIntent = **new** Intent(getApplicationContext(), GameplayActivity.**class**);  
 Bundle extras = **new** Bundle();  
 extras.putString(**"roomname"**, **roomname**);  
 extras.putString(**"username"**, **username**);  
 extras.putBoolean(**"host"**, **false**);  
 roomIntent.putExtras(extras);  
 startActivity(roomIntent);  
 }  
}; *// end handler*

## 3.3.2 Getting Started – Host to Gameplay

Again, MainActivity is displayed, since it’s the starting Activity. The user then intends HostStartActivity for room creation. This is where the host provides his/her username and the room name. When the host clicks the roomStartButton, ccz\_create\_db.php is called. ccz\_create\_db.php gathers the POST data and creates a database that is named after the room name. A sample of the database created is shown below:

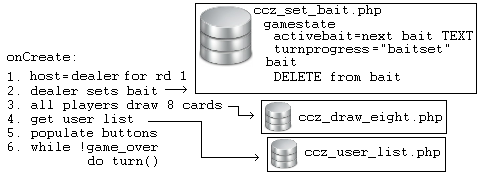
  
  
Figure 3.3.2.1: Admin view before the game has started

If there are no problems creating the database, HostStartActivity intends RoomActivity, which displays the players that have joined the room when the host refreshes the view. RoomActivity gets the username and room name and when the host clicks the gameStartButton, ccz\_start.php changes gamestate started to 1 and uses a bundle to communicate with the next Activity:

**if** (Objects.*equals*(**result**, **"OK"**)) {  
 Intent roomIntent = **new** Intent(**this**, GameplayActivity.**class**);  
 Bundle extras = **new** Bundle();  
 extras.putString(**"roomname"**, **roomname**);  
 extras.putString(**"username"**, **username**);  
 extras.putBoolean(**"host"**, **true**);  
 roomIntent.putExtras(extras);  
 startActivity(roomIntent); }

## 3.3.3 Gameplay – GameplayActivity onCreate

The GameplayActivity begins with some onCreate actions:

  
  
Figure 3.3.3.1: GameplayActivity onCreate summary

The first action is to set the host as the dealer for round 1. It does this after extracting the bundle from the preceding Activity:

*// Get vars from previous activity  
// SET DEALER*Bundle extras = getIntent().getExtras();  
**roomname** = extras.getString(**"roomname"**);  
**username** = extras.getString(**"username"**);  
**host** = extras.getBoolean(**"host"**);  
**if** (**host**) { **dealer**=**true**; } **else** { **dealer**=**false**; }

For the remaining code, the assumption is only for dealer=true.

The dealer sets the bait via ccz\_set\_bait.php, which starts with getting the bait from the bait table and deleting the row:

// GET bait TEXT, ECHO bait TEXT, then DELETE ROW  
$textcontents = mysql\_query("SELECT text FROM bait LIMIT 1") or die("set bait Select from bait Error: ".mysql\_error());  
$row = mysql\_fetch\_assoc($textcontents);  
$bait\_text = $row[text];  
echo urldecode($bait\_text);  
$query = "DELETE FROM bait where text = '$bait\_text'";  
mysql\_query($query, $link) or die("set bait Delete row error: ".mysql\_error());

Then it changes activebait in gamestate, while progressing the turn by setting turnprogress to “baitset”:

// SET activebait and turnprogress IN GAMESTATE  
$query = "UPDATE gamestate SET activebait='$bait\_text', turnprogress='$turnprogress' WHERE id=1;";  
mysql\_query($query, $link) or die("set bait Set activebait/turnprogress error: ".mysql\_error());

Note that the bait is returned to the dealer via an echo call (echo urldecode($bait\_text);), which is put into the Activity String variable “result”.

Next, players (dealer or not) draw eight cards via ccz\_draw\_eight.php:

for ($i = 1; $i <= 8; $i++) {  
 if ($myrow = mysql\_fetch\_array($tablecontents)) {  
 $response\_text = $myrow[text];  
 echo urldecode($response\_text)."|";  
 $query = "DELETE FROM responses where text = '$response\_text'";  
 mysql\_query($query, $link) or die("draw eight Delete row error: ".mysql\_error());   
 } else { echo "Select rows error"; }  
} // end for

The dealer gets a list of users from ccz\_user\_list.php to set visibility to true on the appropriate number of buttons (for example, if there are 6 additional users, there should only be 6 buttons visible). Then the Turn thread is started.

## 3.3.4 Gameplay – Turn

Turn is called from the onCreate method and loops through the turns as follows:

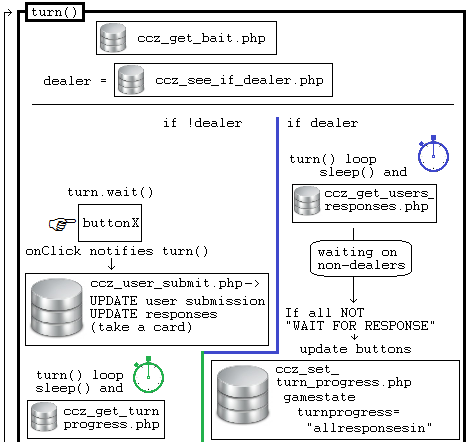


Figure 3.3.4.1: Turn progression part 1

Each player starts by getting the bait for the turn via ccz\_get\_bait.php. Then each player determines if they are the dealer or not (note: this function has not yet been implemented).

Though figure 3.3.4.1 shows the logic for non-dealers, it has not been implemented and therefore only the dealer logic will be covered in this section.

The dealer first runs a sleep/httpclient loop similar to the one in WaitingRoomActivity. In this case, it checks user responses via ccz\_get\_users\_responses.php, waiting for all users to respond to the bait. When all responses are in (ccz\_get\_users\_responses.php outputs “allresponsesin”), a handler updates the GUI to show the user responses in the dealer’s Activity.

Next, the dealer’s Turn thread must wait for dealer input. Unlike previous wait methods used, this uses wait/notify, rather than a sleep/httpclient loop. It performs a thread wait, and is notified by a button onClick.

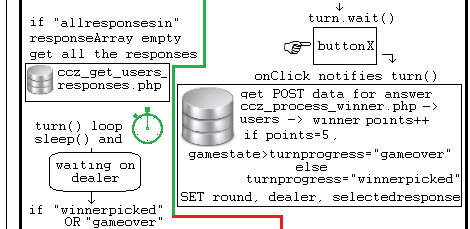


Figure 3.3.4.2: Turn progression part 2

Once the thread is woken up, it calls ccz\_process\_winner.php, which updates several key database fields:

1. Turnprogress is changed to winnerpicked (gameover is not yet fully implemented) to notify users that the dealer has chosen a winner for the round:

// GET user values  
$contents = mysql\_query("SELECT \* FROM users WHERE id=$user") or die("process winner Select from users Error: ".mysql\_error());  
$row = mysql\_fetch\_assoc($contents);  
$selected\_response = $row[submission];  
$points = intval($row[points]);  
$points++;  
if ($points==2) { $turn\_progress = "gameover"; } else { $turn\_progress = "winnerpicked"; }

1. The winner’s point value is incremented:

$query = "UPDATE users SET points=$points WHERE id=$user;";  
mysql\_query($query, $link) or die("process winner Set points in users error: ".mysql\_error());

1. The round is incremented:

// GET user values  
$contents = mysql\_query("SELECT \* FROM gamestate") or die("process winner Select from gamestate Error: ".mysql\_error());  
$row = mysql\_fetch\_assoc($contents);  
$round = $row[round];  
$round++;

1. And the database selectedresponse is set, so that all users can see the selection:

// SET selectedresponse and turnprogress IN gamestate  
$query = "UPDATE gamestate SET round=$round, dealer=$user, selectedresponse='$selected\_response', turnprogress='$turn\_progress' WHERE id=1;";  
mysql\_query($query, $link) or die("process winner Set selectedresponse/turnprogress error: ".mysql\_error());

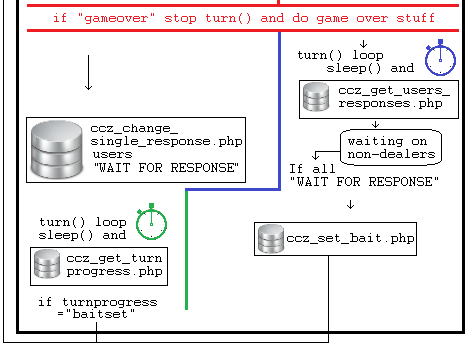


Figure 3.3.4.2: Turn progression part 3

The “gameover” code had not been implemented, but is planned for execution at this point.

Now the dealer waits for the users to set their responses to “WAIT FOR RESPONSE”. This signals that all users have pulled the selectedresponse from the server. After this, it is safe to set the bait via ccz\_set\_bait.php. Note that turnprogress and various stopping points are implemented so that users do not get ahead of any processes. If, for instance, there were no waiting points and communication via the turnprogress variable, a user could pull the next round’s bait variable before it has been set by the dealer.

Next week, we hope to finish up the non-dealer code and gameover code so that we have a fully functional product. Stay tuned!

## 3.4 Build Process

The team is using Gradle (a tool bundled with Android Studio) to build their android application.