HurdleJumpr:
OO Software System Design

Hesham H. Salman Jonathon Kissinger Sean Mead Troy Johnson

October 23, 2014

1 External Design Specifications

The design for layout for the user game will be relatively simple. When the user launches the game, they will be presented with the menu in 1. This menu will give the user several options. The user will have the option play the game, view details about the creators of the game, and check the results of their data was most recently sent in for processing. If the user selects to play the game they will be presented with a layout similar to 2. This is the game layout the user will be presented with when they start playing the game.

Figure 2 is the basic game layout. However, more details may be added later if time permits. The design will be a two-dimensional side scroller game in which the user is represented by a character of some sort (a stick figure in our case in 2). The user will then have to jump over the black blocks that pop up as the user moves along from left to right. A figure of the user jumping can be seen in 3.

The time it takes from when a block first appears, to the time it takes the user to tap the screen to make their character jump over the block is what we will use as the measure for reaction time for the game. Once the user has finished playing, their reaction times are uploaded to the server for further processing. Once their results have been processed and the results are received on the user's mobile device, the user will see a screen similar to that of 4, where they are notified about their results.

The results of our data mining of their response time data may become more detailed later on depending upon how accurate we can get with out predictions. For now, we will just have them a low,medium, or high recommendation for getting checking out by a doctor for the possibility of a ADHD diagnosis.

2 Architectural Design Specifications

3 Detailed Design Specifications

3.1 Interface Specifications

- public interface SyncInterface
  - Relays the results of the sync to the calling thread.

3.2 Class Definitions

- class AddAppsTask extends AsyncTask
  - Adds applications to the main sqlite database.

- class AddResultTask extends AsyncTask
  - Adds results from gameplay to the main sqlite database.

- class SyncTask extends AsyncTask
  - Syncs the list of applications and gameplay results with the server.
- public class SyncService extends Service
  - Controls timed executions of the SyncTask.
- public class MainDatabase extends SQLiteOpenHelper
  - Controls thread-safe access to the database.
- Request Library
  - The Request Library consists of numerous abstractions for thread creation, tcp communication and object transmission.
    - public class Client extends DefaultHttpClient
      - Registers custom ssl connections.
    - public class ClientManager implements ClientStatusController
      - Abstracts the thread-safe creation of ThreadedClient
    - public class NameValueBuilder
      - Used to abstract form creation for POST requests.
    - public class Request
      - Abstracts the creation and execution of threaded network tasks.

3.3 Algorithms

Gathering Applications and adding them to the database. List¡ApplicationInfo¿ appInfoList = pm.getInstalledApplications(PackageManager.GET$_A CTIVITIES$); $List < Application > appList =$ $newArrayList < Application > ();$ $for(Applicationapp : appList)mdb.addApp(app)$;

long start = MainDatabase.getHelper(context).getMinimumDate(); long stop = System.currentTimeMillis();

boolean didCache = false; //whether the cache/caches have uploaded properly boolean didSync = false; //whether the current master db/dbs have uploaded properly

String srcPath = MainDatabase.getHelper(context).getPath(); //file path of the master database

long startSync = Epoch.getDayEpoch(start); //floor the requested range to the start of the day long stopSync = Epoch.getDayEpoch(stop);

if(stopSync¿System.currentTimeMillis()) //hopefully not an issue... will check and deny going back to the future. Log.d("CACHE",String.valueOf(didCache)); Log.d("SYNC",String.valueOf(didSync)); return false; if(startSync==stopSync) //check to see if the upload will be pointless return false;

long cacheStart = startSync; //make a copy of the start date because we want to manipulate it int days = Epoch.getDays(cacheStart,stop,Epoch.MILLIS); //the total number of days we want to upload

for(int d = 0; d ¡ days; d++) FullParser fp = new FullParser(); String dbPath = fp.build(context, (cacheStart), cacheStart + DAYMS); //completely parse one full day and return the path cacheStart += DAYMS; Log.d("CACHE",d+ " PATH: "+dbPath);

//at this point we have a single day cached and ready to upload //we will try up to the value of ATTEMPTS times to upload this file

for(int i = 0; i ¡ ATTEMPTS; i++) if(ClientManager.connect().status) //if we have connected to the server if(ClientManager.cache(dbPath).status) //if the server says the cache file is ok didCache = true; i = ATTEMPTS; if(!context.deleteDatabase(dbPath)) context.deleteDatabase(dbPath); //cleanup the file we created

//at this point all cache creation and uploads are complete

if(!context.deleteDatabase(tdbPath)) context.deleteDatabase(tdbPath); //cleanup

Log.d("COPY",String.valueOf(didCopy)); Log.d("CACHE",String.valueOf(didCache)); Log.d("SYNC",String.valueO

return didCachedidSync;

3.4 Data file specifications

Gameplay results and application data will be stored in a sqlite database locally. That database will sync with our server and will likely be deleted from the device. One table will be created for the application data and another for the gameplay.

4 Test Plan

1. Performance tests
   (a) a. User interface
      i. Ease of use - Ask those unfamiliar with the project, and no more than a beginners knowledge of the game to attempt to play the game. Ask user on how they think the game could be improved or what may have confused them.
      ii. Ease of use on touch device Make sure icons work fine on touch devices, i.e, make sure that all click-able icons that are used aren't so small that a user with rather large fingers would have trouble playing
      iii. Compatibility with other Android operating system versions Mobile game application should be compatible with at least Android Jelly Bean (Android 4.0) or higher. This means that the mobile game will have to be tested on mobile devices with Android Jelly Bean, Android Kit Kat, and possibly Android L (if released in time).
      iv. Other design goals as needed (as more detail is added to the SRS)
   (b) Overall system
      i. Response time should be quick. No more than 1 second without feedback to the user on their results. Wait times that are required to be longer than that should show a dialog giving an estimated wait time if possible, and a general spinner waiting dialog if not to show that the app is retrieving or uploading data [1].
      ii. Availability System should always be available other than when maintenance is being performed on the server. Test what happens if the system is shut down while a client is actively playing the game and/or uploading the results of their game. Other than when the server/system is down for maintenance, the system should always be up and running and therefore a user playing the game at any time should be able to upload the data from their game at almost any time.
      iii. Security - Normal security measures; to test this we could possibly invite a person with hacking knowledge to attempt to hack the system. Also, gather advice on the security aspects from the security professionals and ask them what we could do to increase the security of the system.
      iv. Recovery - How does the system respond if the clients connection is lost when uploading the users data to the server? v. Other design goals as needed (as more detail is added to the SRS)
   (c) Stress tests
      i. Manual - A test where 10 (hopefully more depending on how many testers we can get) logged-in users perform the same functions and see if any wait times or ramp up times occur. This test will help determine how the server reacts to periods of time where many people may be submitting their response time data to the server at the same time for processing.
      ii. Automatic Set up a testing scenario with some open source software testing tool that can simulate as many simultaneous connections as needed (hundreds? thousands?), similar to an integration test. This can method can also be used to determine the maximum number of concurrent users that the server can support under a given configuration.

iii. Automatic (Similar style as mentioned in b) Perform a long-running stress test that drives a continuous load on the server for an extended period of time. (Due to time constraints long-running may only consist of a week at most) The main purpose of this type of test would be to ensure the server accepting the data can sustain acceptable levels of performance over an extended period of time without exhibiting any slowdowns.

iv. User monkeyrunner to simulate thousands of pseudorandom presses in places on the game and see if the application breaks.

3. Functional tests

(a) Specific test cases:

i. User submits response time data with no connection to the server currently available

ii. User loses connection in the middle of uploading their data to the server

iii. User uploads their same data to the server multiple times

iv. User is the first to submit their data to the server (no data to compare against to help determine ADHD diagnosis)

v. User quits in the middle of the game, make sure data isnt submitted.

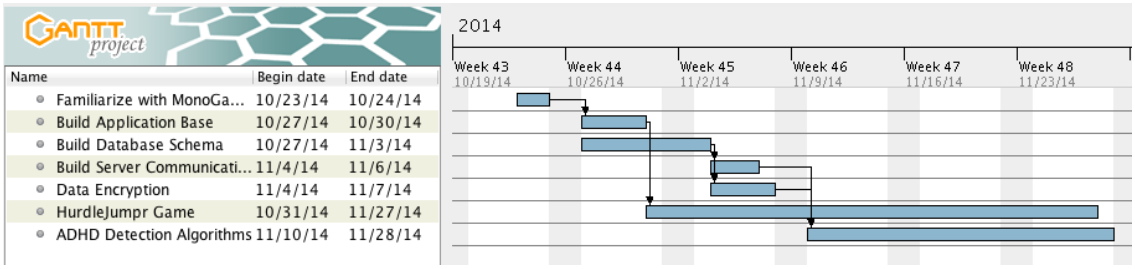
vi. User doesnt give permission to upload their information

5 References

[1]Nielson, Jakob. "Response Times: The 3 Important Limits." Nielsen Norman Group. Nielsen Norman Group, 1 Jan. 1993. Web. 23 Oct. 2014.

6 Appendices

7 Gantt Chart



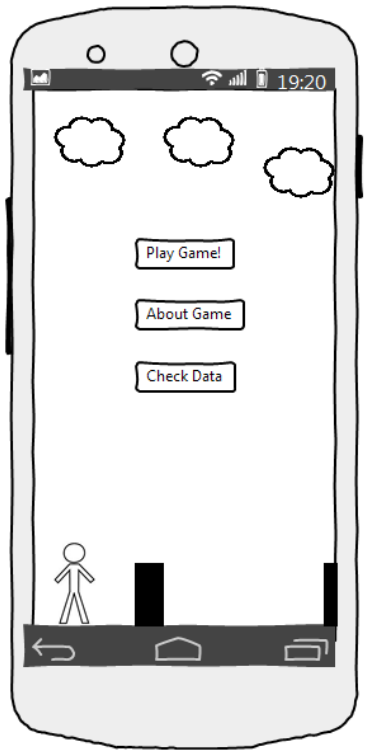


Figure 1: Menu Screen User uses to navigate

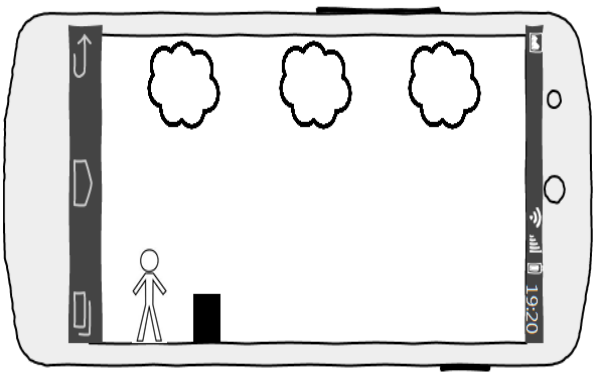


Figure 2: Typical screen layout for user playing game

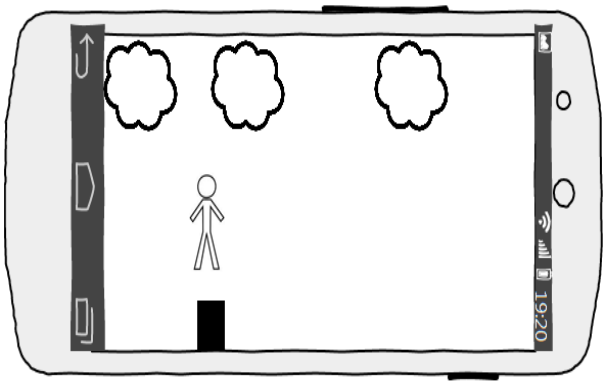


Figure 3: User will have to jump over the black blocks

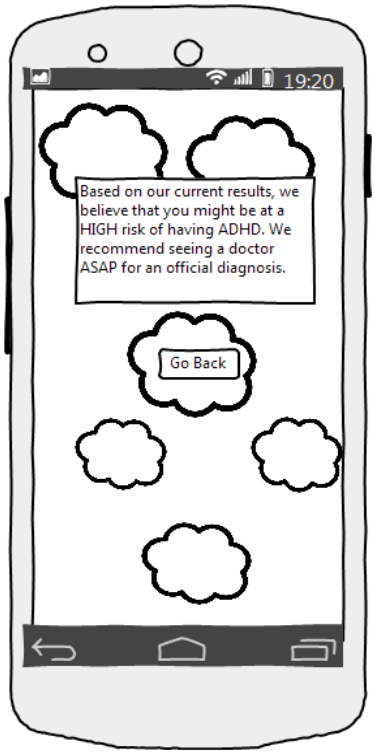


Figure 4: Screen where the user can view their results