FIGHT BOAT

Team Fight Boat

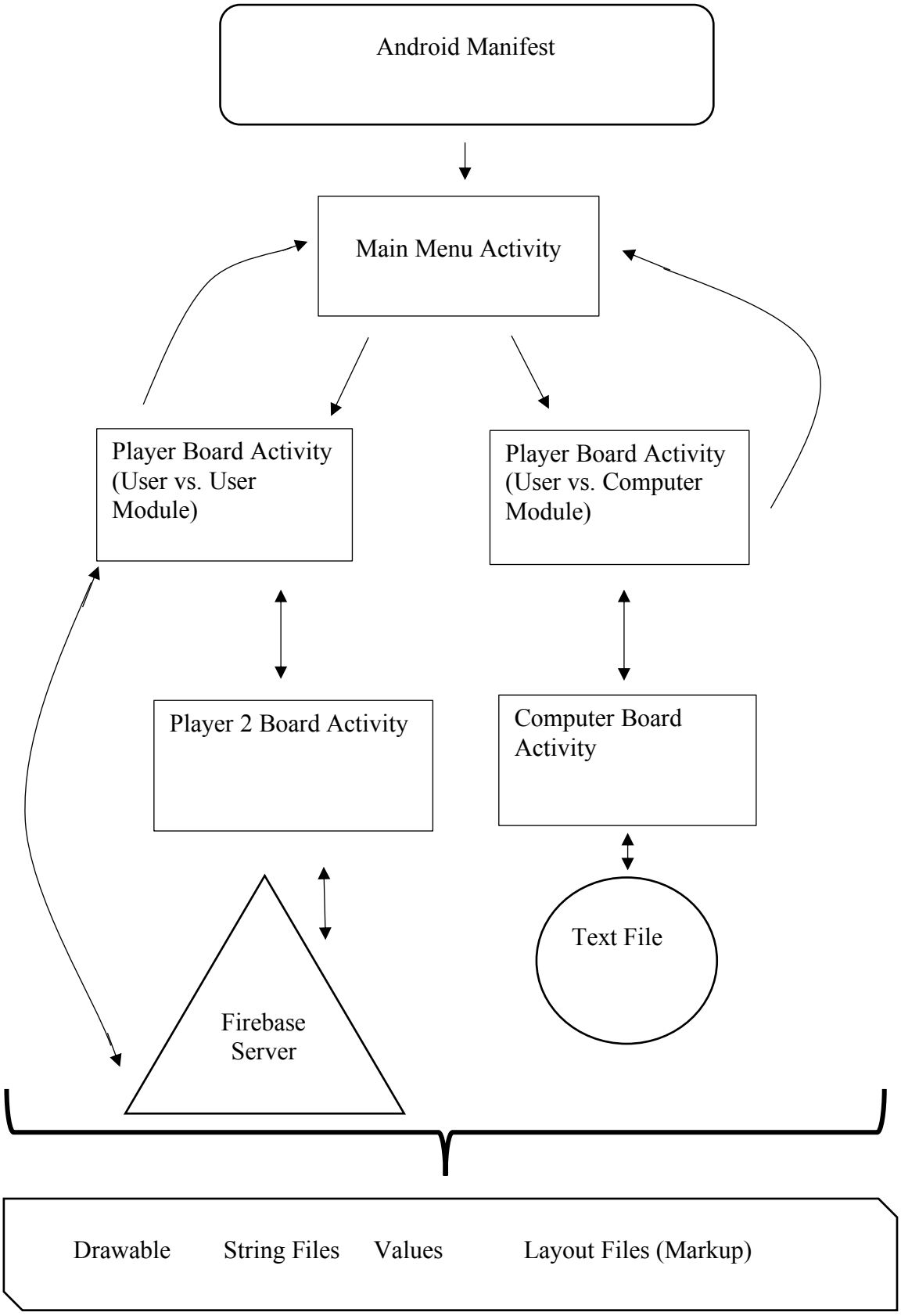
Sean Nemtzow: snemtzow
Ye Chen: chenye
Cullen Paulisick: clp0216
Terry Zhen: tzhen
Sean Brandenburg: sean333

Repository:
https://github.com/21ChenYe/FIGHTBOAT

Video:
https://youtu.be/IBMXd0RCG-4

Software Architecture

The overall architecture for our software follows a simple activity pathway. The main activity, initiated by the manifest, is a preliminary menu where the user is able to select a mode for gameplay, either wireless user vs. user or user vs. computer gameplay. From whichever of the choices, a different activity pathway is activated. For the user vs. computer mode, the activity pathway initiates an activity known as the "player map" that holds the ship view objects belonging to the user. The user has the ability to place ships on their grid either horizontally or vertically. From there, the player's map creates a child activity known as the "computer's map" that holds the computer's ship view objects. In a cyclical manner, the player's map activity and computer's map activity switch to one another respectively after each player/computer turn. The user will select a tile on the grid to "attack" during each turn, resulting in either a hit or a miss of the opponent's ships placed. The computer simulates this action during each of its turns. In order to save the data for each activity, attributes of the game grid and the computer's ships, including the location on the map, the length of the ship, and the amount of the ship that has not been sunk yet are written to a text file by the computer map activity. The ship data of the player map activity is automatically saved as a result of its parent relationship with the computer map activity. Furthermore, the activity switch after each turn that occurs between the player and computer map activities happens indefinitely until all of the ship objects for either the user or computer are completely defeated. At the conclusion of a game, an activity is initiated the gives the user the ability to return to the main menu activity or automatically restart the game. For the user vs. user game mode, a near-identical architecture to the user vs. computer module is utilized. The primary difference is that the computer map activity is replaced by a second player map activity that is allowed to place the boats on the computer's map. From there, the computer takes over and faces the primary player. Instead of a text file, however, the information regarding each of the players' respective game grids and ships is exchanged via a mutual Google Cloud Firebase server, enabling for wireless gameplay. The same information regarding grids and ship attributes that is written to the text file for the user vs. computer module is written to the server for the user vs. user module. In regards to the images used for the pop-up notifications, game grid, ships, and other extraneous interfaces, the images are stored in respective "drawable" .png files, accessed via the res folder of our repository. Additionally, accessed through the res file are the dimensions, strings, styles, and ship attributes continuously accessed throughout the project, held in different .xml files in the "values" directory. The .xml markup files for each activity used are accessed in the "layout" directory.


Android Manifest
Main Menu Activity
Player Board Activity (User vs. User Module)
Player Board Activity (User vs. Computer Module)
Player 2 Board Activity
Computer Board Activity
Text File
Firebase Server
Drawable
String Files
Values
Layout Files (Markup)

Overview

Fight Boat is an interactive game that allows for either user vs. user or user vs. computer gameplay. The premise of the game is simple: users position their boats on their grid and then select positions on the enemy's grid to attack. If a portion of the enemy's hidden boat is positioned on that specific point on the grid, then that portion of the boat is "hit". When all portions of the boats are "hit," then that boat is sunk. Alternating turns, the user competes to have all of the enemy's boats sunk before the enemy sink the user's ships. When all of the boats of the enemy are sunk, the user wins, but if the user has their boats sunk first, then they lose. The app supports single-player gameplay versus a computer bot, as well as a player vs. player module where players can play against their friends on other devices.

The application is extremely useful for quick gaming experiences. Users will have expediency when using the application, as games are straightforward, engaging and fast-paced. This app also seeks to capitalize upon the need for head-to-head, intimate gaming competition in the wake of massive multiplayer games' advent. Fight Boat is a handy and convenient game for those who enjoy concise, simple gaming on-the-go.

The market validity of the game is based around these principles. The primary market for the app is the younger generation of Generation Z/Millennials, with who mobile gaming is most popular. The confrontational, head-to-head gameplay will heavily appeal to this market as a fresh take on mobile games, most of which utilize either single player or massive-multiplayer online platforms, rather than the head-to-head module featured in Fight Boat. Another important market motif of Fight Boat is the "meme-ability" of the game. The light mannerism of the game, as well as the punchy, ironic title makes it a viable candidate for meme-based publicity. The increasingly important role that memes play in pop culture is made apparent by examples such as "Old Town Road" by Lil Nas X. This song, an outright meme, catapulted to the number one spot on the Billboard charts and garnered the artist several record deals. The reality is that memes are becoming an increasingly better way to make money and market products.

For all of the reasons above, Fight Boat is an extremely approachable app with a broad market.

Component Description

GUI (Front End)

The user interface is primarily defined by .png image files accessed through the "drawable" files in the "res" directory, imprinted upon button view objects. For the main menu, the user interface includes two buttons with distinctive text styles for accessing the different game modules, as well as a basic title screen and logo image. The player board activity, which includes the game grid and the ship objects, constructs the grid for gameplay by allocating a 10 by 10 two-dimensional array of "tile" objects created within the activity. Each of the tiles is fitted with a respective .png image. The .png file images for the tiles that are not inhabited by ship objects represent "open" ocean space and are colored blue. Tiles inhabited by a ship are fitted with a .png file image that represents the portion of the boat covering that particular tile. For notifications regarding a "hit" or "miss," or activity switches that initiate/terminate games, pop-up activities are called that display short text items.

Interface (Front End)

For connecting the different components of the project, GitHub was used as a version control unit for connecting the necessary code. Organizationally, code was integrated through this repository. The activities, such as the player board activities and the computer board activities are all connected through the repository of the GitHub. Activity switching and connection, on a more microscopic level, occurs via starting new Intents with methods like "setOnClickListener()" for button objects. In cases where button pressing is literally necessary for action, this method is activated based upon event-driven programming. In cases where activity switches are predicated upon virtual button-pressing (and not user-generated action), a virtual object is "created" and then activated while listening automatically so that a new intent can be created and the activity switch can occur. As mentioned above, data can be transferred between activities via writing to text files and accessing the remote Firebase server. Writing data concerning to these two components allows for sending data to the necessary activities.

Processor (Back End)

Since battleship consists of two identical maps with identical ships, representing the game in an app consists of a generous repetition. After leaving the main menu, the gameplay begins in the Main Activity, the player's map and ships. When the activity is created, the a matrix of custom View objects named Tiles are structured to represent the board and a fleet of custom view objects named Ships are arranged above the map. The Ships are drag sensitive and the Tiles are listening for drag events to accept the ship the objects. When a ship object is placed on a Tile, it paces the information about which type and which section of a ship it is. The tile uses this information to choose the right image to display. When all of the ships are placed on the map, the Main Activity launches a child activity which represents the other player's map. If the user is playing

multiplayer, it sends the database the positions of all of the ships. The other player's map is a child activity so that the main activity remembers which state it was in before. When the child activity is created for the first time, it randomly places its own ships, while keeping them hidden of course, and awaits a click of the user. If the user is playing multiplayer, instead of randomly placing ships, it reads from the database where it should place them. All of the tile objects are click sensitive and change their display when clicked depending on if they hold a ship or not. After the user clicks, the activity notifies the user if they missed or hit or sunk a ship. Before the child activity is destroyed, it writes the map's state and ship object information to a text file in a specified order. When it is created again, it reads from the text file to re-upload the map. When it returns to the Main Activity, it randomly chooses a coordinate on the map to hit, representing an enemy choosing their attack. The user is notified if the computer missed or hit or sunk any ships, and returns to the child activity when the user dismisses the notification. This process continues until either the player or the computer sinks all of their opponent's ships, which signals the end of the game. At the end of the game, the user is given a victory or defeat message and the options to return to the main menu, or restart the game.

Task Distribution

Back-End, Processing, Activity Organization – Sean Nemtzow

GUI, Graphic Design, Menu Activity– Ye Chen

GUI, User Experience Design, Audio Curation – Terry Zhen

Interfacing, Server Operation – Sean Brandenburg

Documentation/Report, Menu Activity, Notifications – Cullen Paulisick

Effort Assessment

Sean Nemtzow – 23%

Sean Brandenburg – 20%

Terry Zhen – 18%

Ye Chen – 19%

Cullen Paulisick – 20%

Timeline

Week 1:

Began preliminary designs for user interface and gameplay concepts, graphic design; also initiated architecture design for front-end

Week 2:

Started organization of activities and activity switches, began initial back-end/processing coordination; improved interface and user experience

Week 3:

Finalized user interface and user experience details, worked on back-end and processing with primary gameplay activities and activity switches; began server creation

Week 4:

Finalized back-end/processing with gameplay activities, finished interfacing and GUI as well; connected server to gameplay

Compilation Instructions

In order to compile this app first go into the SDK manager in android studio and go to the sdk tools tab. Then check all of the google play tools. Then ensure that the device you are using to emulate is API 28 with Google Play. Then click the run selected program button and the app should load onto the emulator that you select.