Capture the Flag

Design Document

Authors: Luke Colburn, Tyler Johnson, Chris LaBauve

Revision History

Date	Version	Description	Author(s)
2014-02-11	0.1	Initial draft	Luke Colburn, et al.
2014-02-13	0.2	Appended to functional / nonfunctional requirements	Luke Colburn, et al.
2014-02-17	0.3	Added definitions, table of contents	Chris LaBauve
2014-02-17	0.4	Added introduction, project description	Tyler Johnson
2014-02-25	0.5	Added diagrams and diagram descriptions	Luke Colburn, et al.
2014-02-25	1.0	Cleaned up document and added table of contents page numbers	Chris LaBauve
2014-03-19	1.1	Added/updated class diagrams, sequence diagrams, and collaboration diagrams and diagram descriptions	Luke Colburn, et al.
2014-03-20	2.0	Formatted diagrams and updated TOC	Chris LaBauve

Table of Contents

Revision History **1**

1 Introduction **5**

1.1 Definitions **5**

1.1.1 Player 5

1.1.2 Avatar 5

1.1.3 Character 5

1.1.4 Puzzle 5

1.1.5 Asymmetric play 5

1.1.6 Game session 5

1.1.7 Buffalo 6

1.2 Project Goal & Inspiration **6**

2 Description **6**

3 Functional Requirements **6**

3.1 Vital Functional Requirements **6**

3.1.1 Puzzle 6

3.1.2 Character selection 7

3.1.3 Side-scrolling 7

3.1.4 Screen Orientation 7

3.1.5 High Score Tracking 7

3.1.6 Local Multiplayer 7

3.1.7 Team Play 8

3.1.8 Player communication 8

3.1.9 Specific Character Mechanics 8

3.1.10 Mechanic Combinations 8
3.1.11 Controls 8
3.2 Non-Vital Functional Requirements 9
3.2.1 Multiplayer Matchmaking 9
3.2.2 Friends List 9
3.2.3 Private Game 9
3.2.4 Competitive Play 9
3.2.5 Asymmetric Competitive Play 9
3.2.6 Gravity Shift 9
4 Non-functional Requirements 10
4.1 Vital Non-functional Requirements 10
4.1.1 Movement 10
4.1.2 Graphics 10
4.1.3 Synchronous Gameplay 10
4.1.4 AndEngine 10
4.1.5 Box2D 10
4.2 Non-Vital Non-Functional Requirements 10
4.2.1 Multiplayer Matchmaking 10
4.2.2 User System 11
4.2.3 Device Accelerometer 11
5 Use Cases 12
5.1 Interact with Main Menu 12
5.2 Join Game 13
5.3 Play Game 15
5.4 Play on Team 16

5.5 Play Against Team **18**

6 Diagrams **19**

6.1 Class Diagrams **19**

6.1.1 Game 20

6.1.2 Android UI 21

6.1.3 Networking 22

6.1.4 Game Host and Game Client 23

6.2 Sequence Diagrams **24**

6.2.1 Create Game 24

6.2.2 Display Friends 25

6.2.3 Game Client 26

6.2.4 Game Host 27

6.2.5 Display High Scores 28

6.2.6 Join Local Game 29

6.2.7 Join Matchmaking Game 30

6.2.8 Join Private Game 31

6.3 Collaboration Diagrams **32**

6.3.1 Game 32

6.3.2 Network 33

6.4 Activity Diagram **34**

1 Introduction

1.1 Definitions

1.1.1 Player

A player is a single human interacting with the software described in this document.

1.1.2 Avatar

An avatar is a representation of a player within the software. Once a player enters play mode of the game, her avatar may be any of the game's cast of characters. A player may change avatars in the course of a puzzle.

1.1.3 Character

A character is any of the cast of individual playable entities in a puzzle. A player must select one character to be her avatar within a game session.

1.1.4 Puzzle

A puzzle is a map with a certain play type (asymmetric or not), tailored to be solved by a certain number of players in potentially many ways by combining different character abilities.

1.1.5 Asymmetric play

Asymmetric play is a form of play in which one or more players have a vastly different role or set of goals from the other players (see 1.1.7 Buffalo).

1.1.6 Game session

A game session is a single play session involving between one and four players with a single objective. A game session is over either when the players achieve the objective or when any player quits.

1.1.7 Buffalo

“To outwit, confuse, deceive, or intimidate” (http://en.wiktionary.org/wiki/buffalo#Verb). This is the name given to the player that antagonizes the team trying to solve the puzzle in an asymmetric puzzle. His in-game representation will probably be a very grumpy-looking buffalo.

1.2 Project Goal & Inspiration

The primary goal of this project is for players to communicate and play in real-time to solve puzzles with other players on their phone or tablet. Typically, game developers for mobile games have resorted to turn-based or asynchronous methods to include multiplayer gaming. We feel that a fully real-time game over LAN or the internet (via 4G, 3G, or local wireless) is technically feasible and allows for a game that is more engrossing than typical mobile multiplayer games, such as "Draw Something" or "Words With Friends."

2 Description

Capture the Flag is a synchronous, multiplayer puzzle game developed for mobile platforms and inspired by cooperative games such as Portal. The puzzles included will require a specific number of players to play and interact in real time. Each player will be able to select one character from an assortment of characters, each of which has a particular property or skill that the player can use. Each puzzle will be able to be solved with a particular combination of characters. The puzzles will typically be platform-based levels with a starting point and several "flags" distributed to distinct locations throughout the level. In order to complete the levels successfully, the players will need to collect each of the flags using their characters' unique skills.

3 Functional Requirements

3.1 Vital Functional Requirements

3.1.1 Puzzle

- The objective of each puzzle is for a team to collect all the flags on the map (some maps will only have one flag).
- Players do this by combining abilities to solve puzzles
- Assortment of different characters to choose from, each with different mechanics
- Limited character selection pool per puzzle
- Each puzzle is designed to be solved by a specific number of players

3.1.2 Character selection

- Players start each puzzle as a blank avatar
- Players may run and investigate the puzzle as this avatar, but he has no abilities
- When they decide what abilities are needed, there is a chamber at the start of the puzzle the characters can enter and swap to characters with the needed abilities

3.1.3 Side-scrolling

- The game will be oriented to show all the action from the side (as opposed to a bird's eye view)
- In this way, it will provide a platforming experience, where players can jump from platform to platform and objects will be affected by gravity

3.1.4 Screen Orientation

- For simplicity, we will force landscape orientation. This also works well for our gravity shift requirement, as the map will always be locked to the screen in one orientation and will not need to change based on accelerometer readings.

3.1.5 High Score Tracking

- Each individual player will have a record of high scores (fastest time to completion) achieved in each puzzle game arrangement
- High scores will be associated with the other players in the game

3.1.6 Local Multiplayer

- Players can connect with other players over a local area connection so that up to four may be engaged in a single play session at one time

3.1.7 Team Play

- Games may consist of 2-4 players on a team to solve a puzzle

3.1.8 Player communication

- Players may “point” to specific locations on the map. This is done by dragging an “!” icon onto a part of the map.
- Players may also alert other players of their location (an indication to “come here”) by dragging the “!” onto their avatar
- Players are always aware of the direction in which of other offscreen characters lie; there are arrows pointing to them.

3.1.9 Specific Character Mechanics

- Umbrella character can float while airborne
- Wind character can blow gusts of air
- Water character can shoot streams of water, can fill up jars, etc. to weight down a lever e.g.
- Plank character can lie flat on a ledge to provide a platform

3.1.10 Mechanic Combinations

- Wind character may keep umbrella character afloat with wind bursts
- Gusts of air may redirect stream of water
- Open umbrella can split a stream of water into two separate streams
- Water character may stand on top of plank character so that she can direct her stream downwards

3.1.11 Controls

- Tap & Hold a side (left or right) of screen to move
- Swipe up to jump
- Double-tap the screen in one of the four cardinal directions to perform special ability

3.2 Non-Vital Functional Requirements

3.2.1 Multiplayer Matchmaking

- Players may join a game with random people on the Internet
- They may join games based on criteria, such as number of players, play style (2v2, 3v1, etc.), or a specific puzzle

3.2.2 Friends List

- Players may retain a list of friends so that they can play with the same people again

3.2.3 Private Game

- Players may create private games open only to friends or specifically invited players

3.2.4 Competitive Play

- Games can be two teams of two racing to beat one another to solving the puzzle
- Games can also be two single players racing to solve a 1-person puzzle (if we create single-player puzzles)

3.2.5 Asymmetric Competitive Play

- If only three players are playing, two can be on a team to solve the puzzle and the one other will have some other role entirely as the Buffalo, creating obstacles for the other two. If he can prevent them solving the puzzle in a time limit, he is the winner.
- The Buffalo player can make use of the “gravity shift” mechanic to hinder his opponents
- This can also work with 3v1 if four players are playing

3.2.6 Gravity Shift

- Certain puzzle stages will allow players to alter their personal gravity based on device orientation
- Other stages will give the gravity control to only one of the players but he controls the gravity for everyone

4 Non-functional Requirements

4.1 Vital Non-functional Requirements

4.1.1 Movement

- Gravity: Objects in the game have a weight associated with them which determines how they are affected by gravity
- Collision

4.1.2 Graphics

- Graphics will be sprite-based; we can get open-source sprite resources from http://opengameart.org/

4.1.3 Synchronous Gameplay

- Uses UDP socket connections for networked gameplay

4.1.4 AndEngine

- We will use the AndEngine library as a third-party resource so that we don't need to write all the game abstraction code ourselves
- Provides extensions for both Physics and Multiplayer http://www.andengine.org/blog/showcase/

4.1.5 Box2D

- http://box2d.org/
- Physics engine that we can plug into AndEngine so that we can focus on developing game mechanics and not basic physics models
- This is the physics engine used in Angry Birds
- Can handle gravity and collisions--can even do physics constraints (like links of a chain)

4.2 Non-Vital Non-Functional Requirements

4.2.1 Multiplayer Matchmaking

- To be able to matchmake players, a central server will be needed to connect players to one another and to assign hosts
- This server can potentially be provided by Google Play Game Services

4.2.2 User System

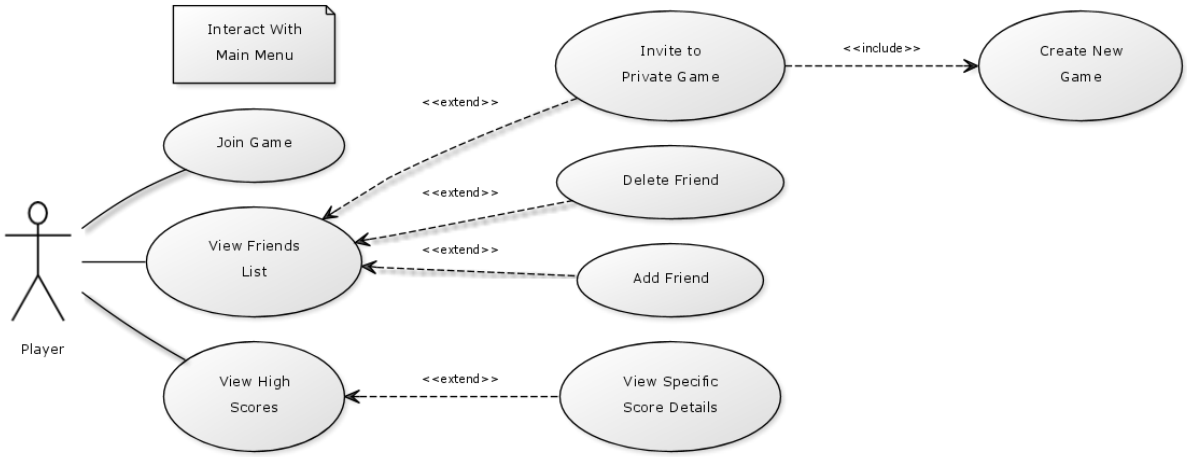
- If we have friends and private games, a system to identify users uniquely is necessary
- This can also be provided by Google

4.2.3 Device Accelerometer

- In order to support the Gravity Shift feature (3.2.6), the device running the application will need an accelerometer to detect physical orientation.

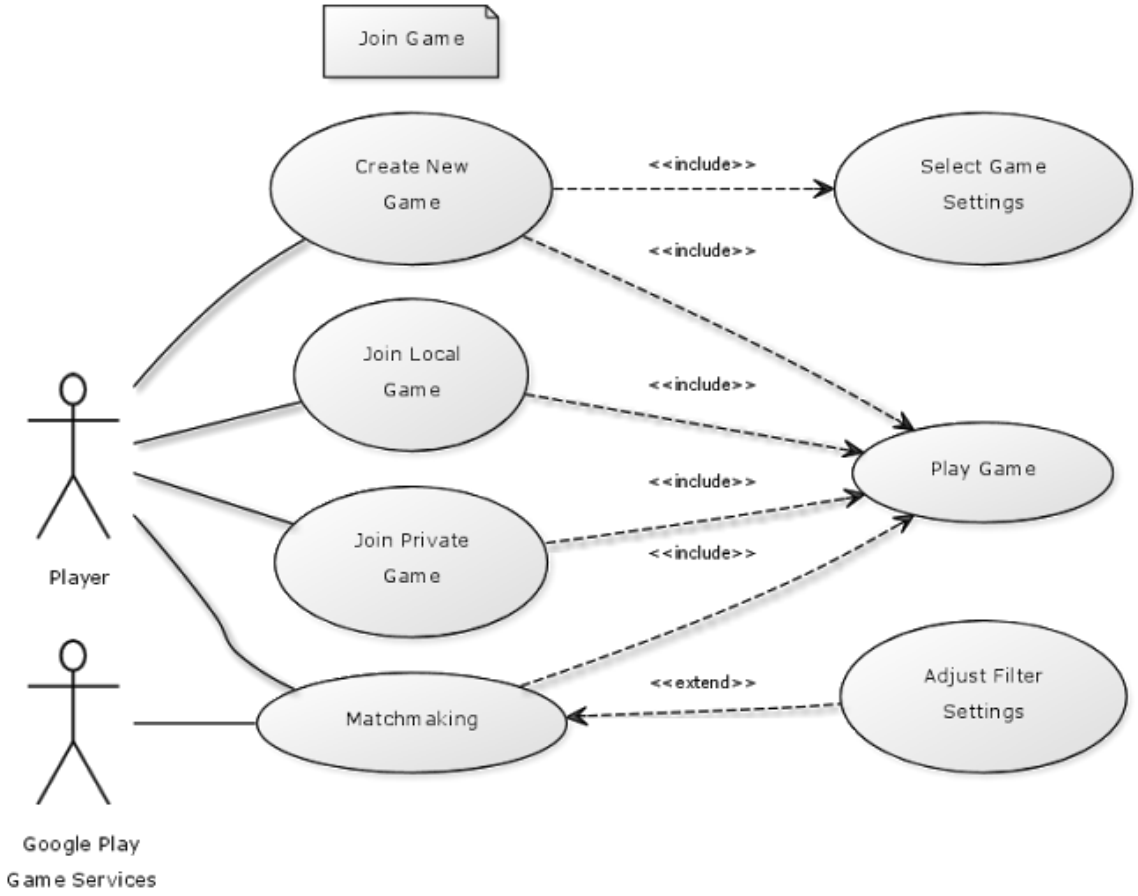
5 Use Cases

5.1 Interact with Main Menu



- When in the main menu, a player may view his high scores. When looking at high scores, he may optionally check further details for a specific high score.
- A player, from the main menu, may also view his friends list. He may then delete a friend. He may also add a new friend. He may also select 1, 2, or 3 of his friends to immediately invite to a private game; when he sends the invites, he must then create his new game (detailed in 5.2 Join Game).

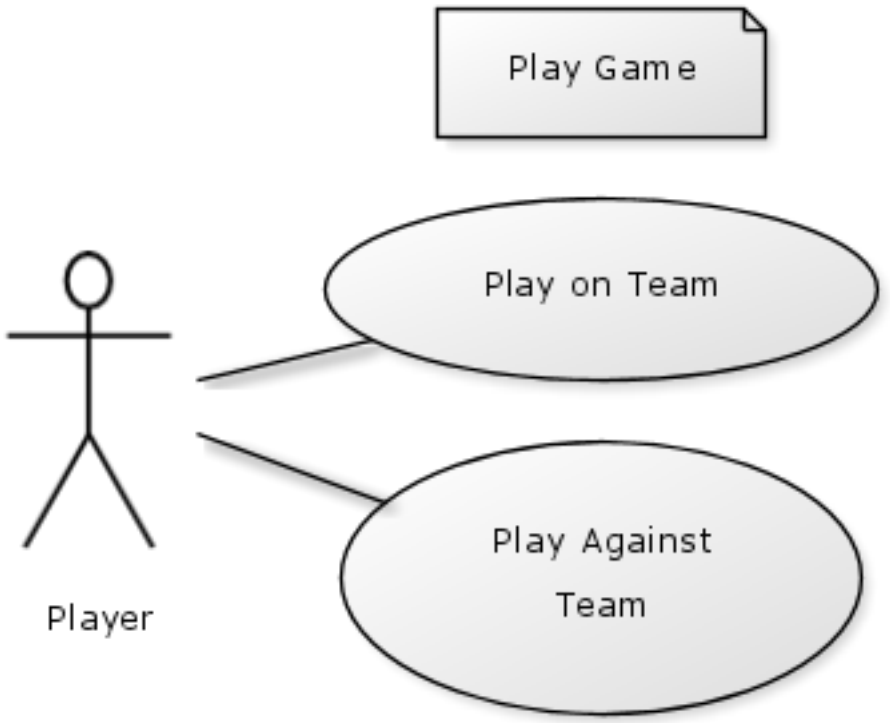
5.2 Join Game



- When attempting to join a game, a player may create a new game. In order to do this, it is required that he select game settings, including puzzle to play and privacy settings (friends only, invite only?, public) then, when the number of required other players join the game, he may proceed to playing the game (5.3 Play Game).
- A player may, at this stage, also join a local or a private game. A local game is defined as a public game played on the local Wi-Fi network. A private game is currently defined as a game hosted by a friend. The semantics of this definition are subject to change, as we may define different privacy options, such as “invite only.” Once again, once enough other players have joined, the player may play the game (5.3 Play Game).

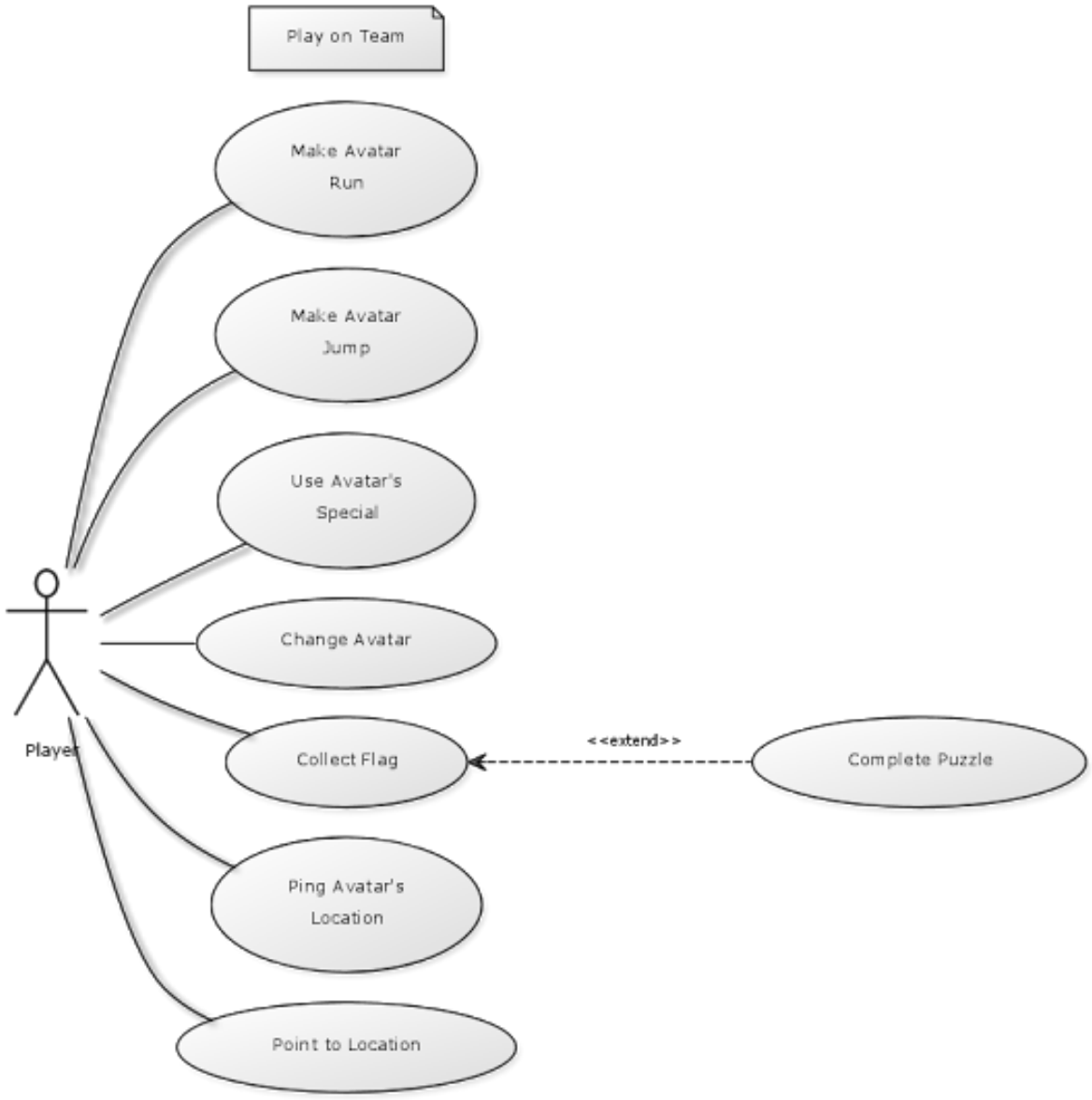
- A player may choose to join a matchmaking game, which will be generated by the external Google Play Game Services actor. Before commencing the search for potential games, a player may adjust filter settings for a preferred match type; for example, a player may only want to join a match that has asymmetric play with three players. Alternatively, he may specify to the system that he does not care which type of puzzle he joins. Once Google Play Game Services matches players into one game, they may then play the game (5.3 Play Game).

5.3 Play Game



- Once a player is in a game, he will be assigned a role as a team member if it is a strictly team-oriented puzzle. If the puzzle is asymmetric, he may be assigned the role of a team member or the role of the Buffalo, with the task of preventing the team from solving the puzzle (see 5.4 Play on Team and 5.5 Play Against Team for more details).

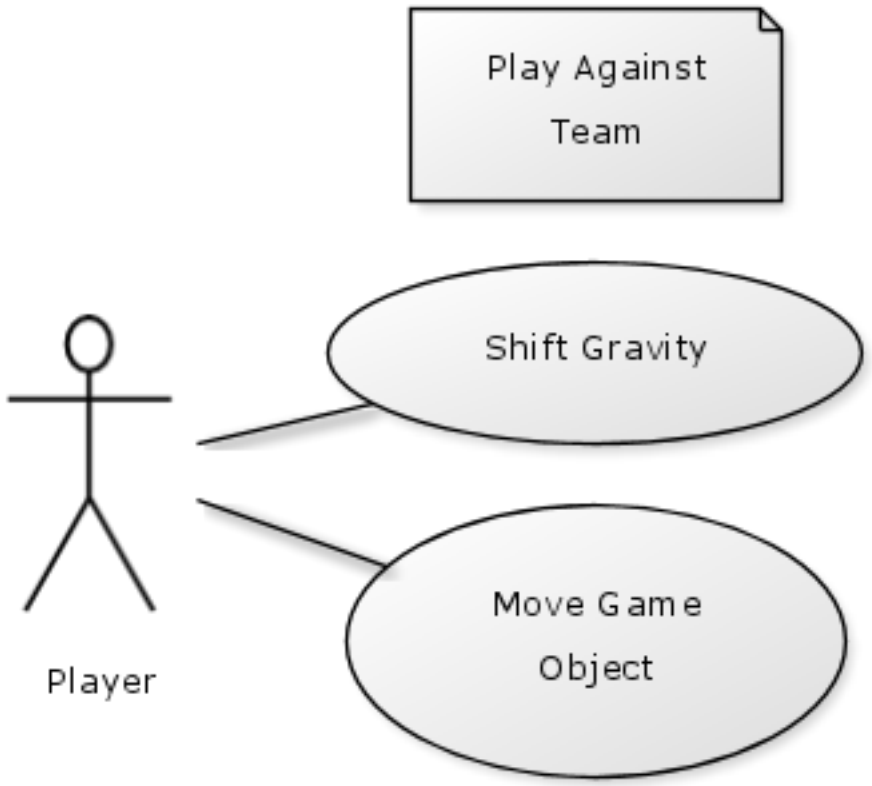
5.4 Play on Team



- Once a player is inside a game sessions and has been assigned the role of team player, he has several options, mostly related to controlling his avatar:
    - A player may make his avatar run or jump.
    - A player may use his avatar's special ability (sometimes in combination with another avatar's ability).
    - A player may direct his avatar to go to a specific checkpoint on the map to select a new character with a different ability.

- A player may collect one of the flags on the map; if it is the last flag left, the puzzle has been completed
- A player may use a ping interface to either alert other players of his location or to alert other players that their assistance is needed at a particular location.

5.5 Play Against Team

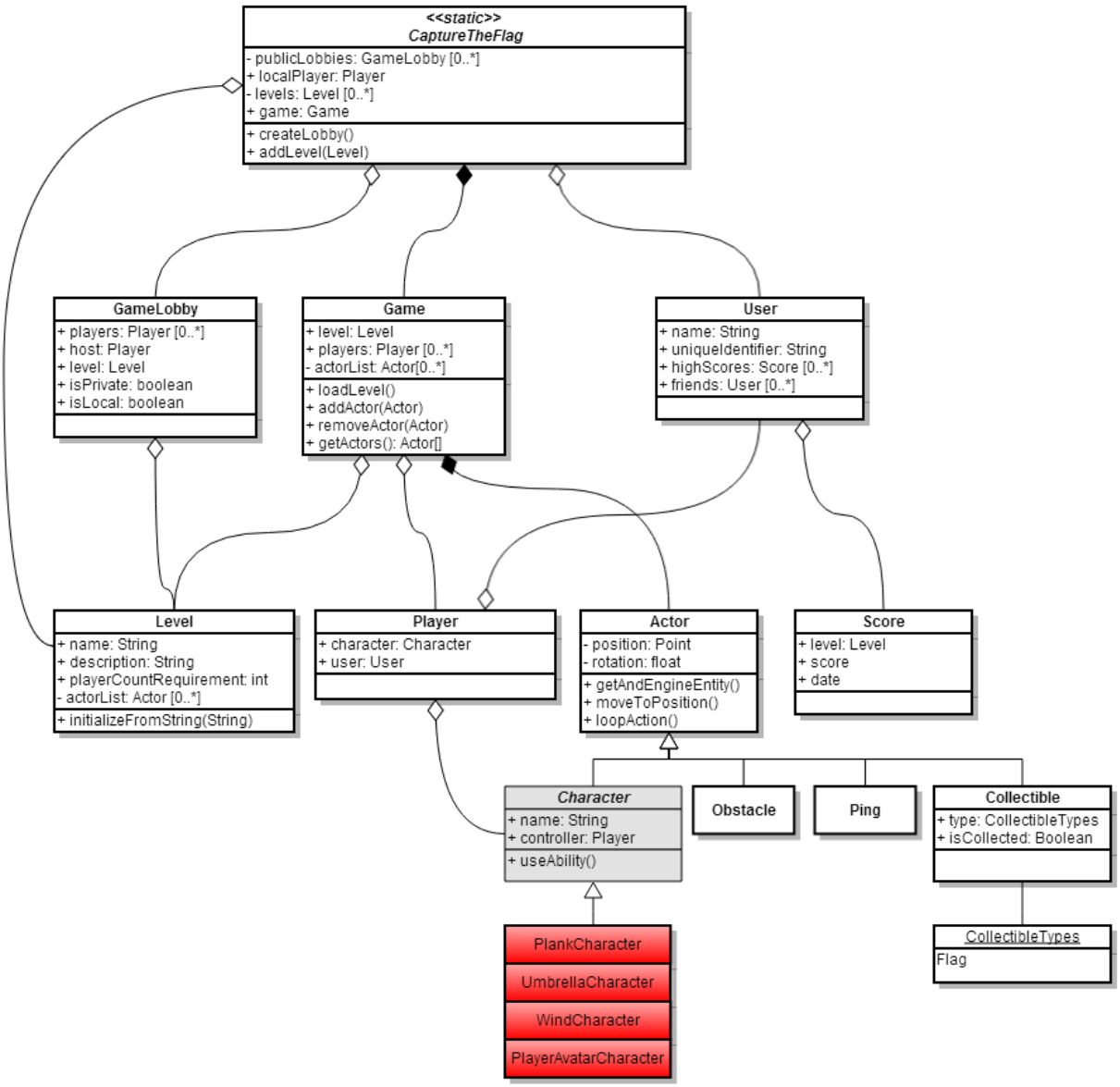


- When a player is inside a game session and assigned the role of the Buffalo, he has two primary actions (so far) that allow him to hinder the team:
    - He may shift gravity to point in any of the four cardinal directions, affecting the other players drastically (but not necessarily rendering the puzzle unsolvable). This action will probably need a cooldown timer, as constantly changing gravity wells will probably make the puzzle too difficult.
    - The Buffalo may also move certain objects on the map (such as blocks) and put them out of reach of the team players or build walls in front of them.

6 Diagrams

6.1 Class Diagrams

6.1.1 Game

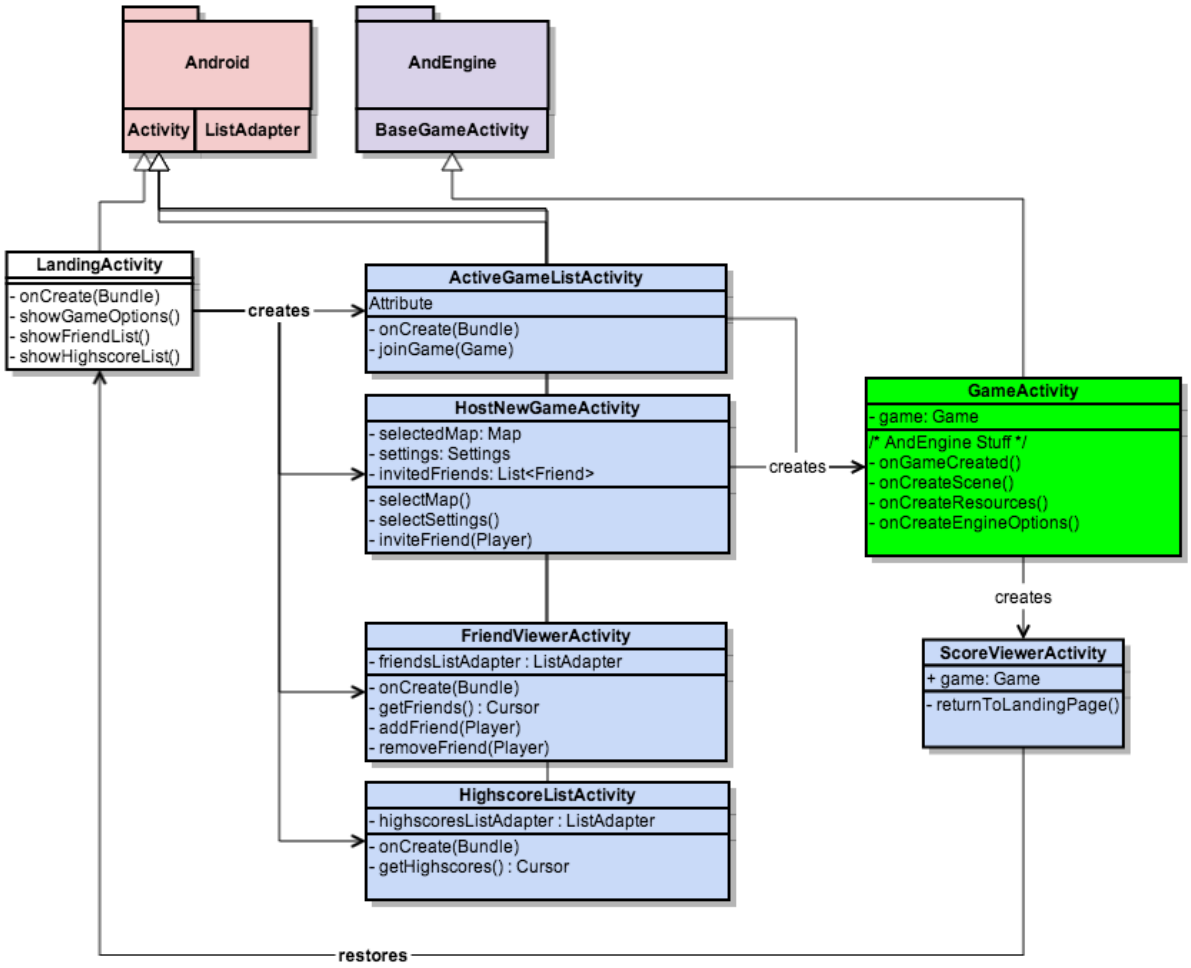


This diagram outlines the data model for the different entities of the game and their relationships.

The static CaptureTheFlag class serves as the interace between the UI activities and the game activity.

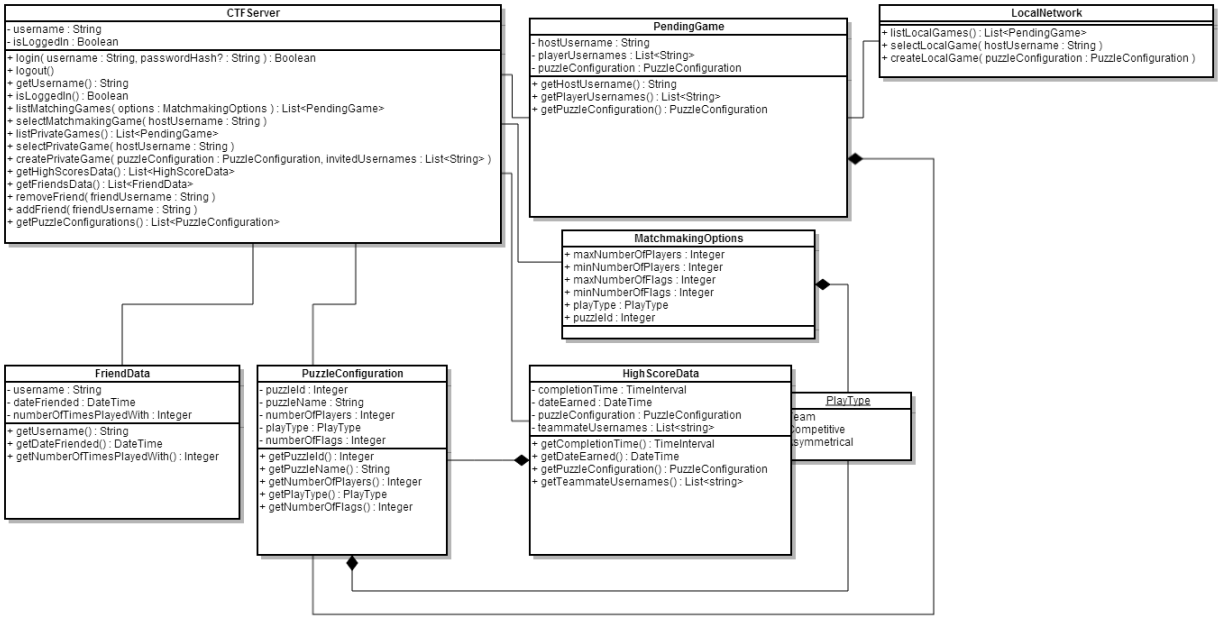
The Actor system models anything that can be displayed on the screen. Characters are a specific type of actor with the ability to be controlled by a player. Any other type of displayable may subclass Actor. The red classes above are examples of Characters available for play by the player. Each Level (aka Puzzle) has a list of Actors and iterates through them each game tick to update state.

6.1.2 Android UI



This diagram encompasses the different activity subclasses that will be created and how they are related to each other. Each activity represents a screen of the program.

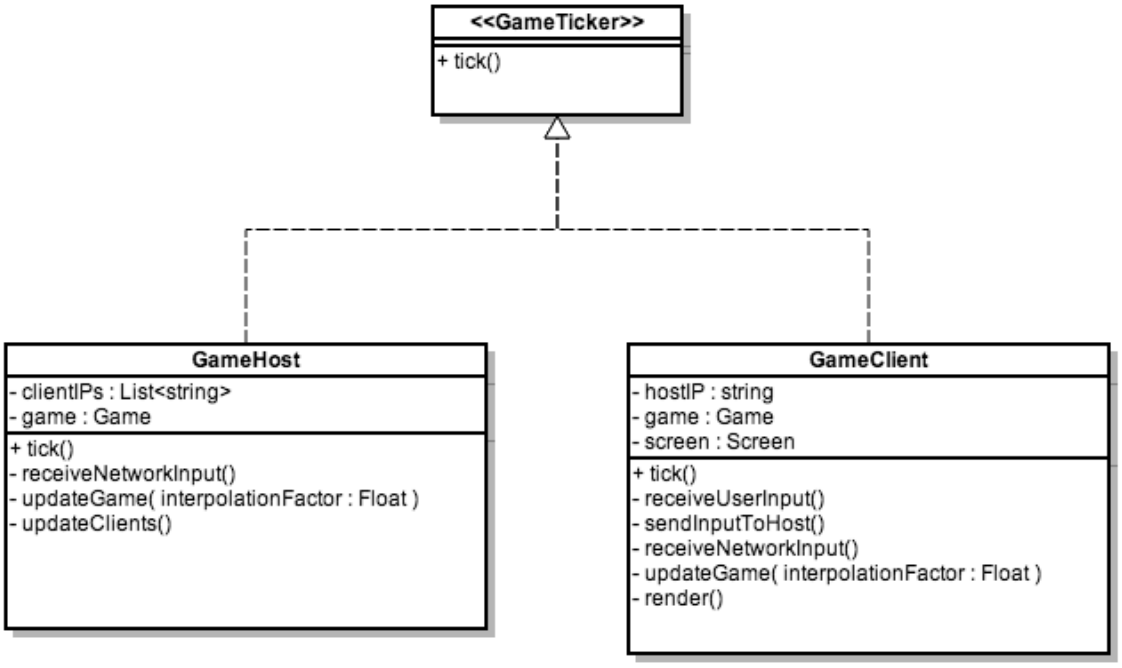
6.1.3 Networking



The *CTFServer* class will be a static class that provides a convenient interface to the server for our game (where user data and games hosted over the Internet reside—that is, players looking for games over the Internet, private or matchmaking, must find them on the server. Games will still be hosted by players, not by our server). It exposes several functions that get information from the server or request an action to be performed. This is also where the user login submission is handled.

The *LocalNetwork* class is analogous to the *CTFServer* except that it is only for searching, joining, and creating games on a local network—this is for connecting to players nearby without the need for any kind of user account.

6.1.4 Game Host and Game Client

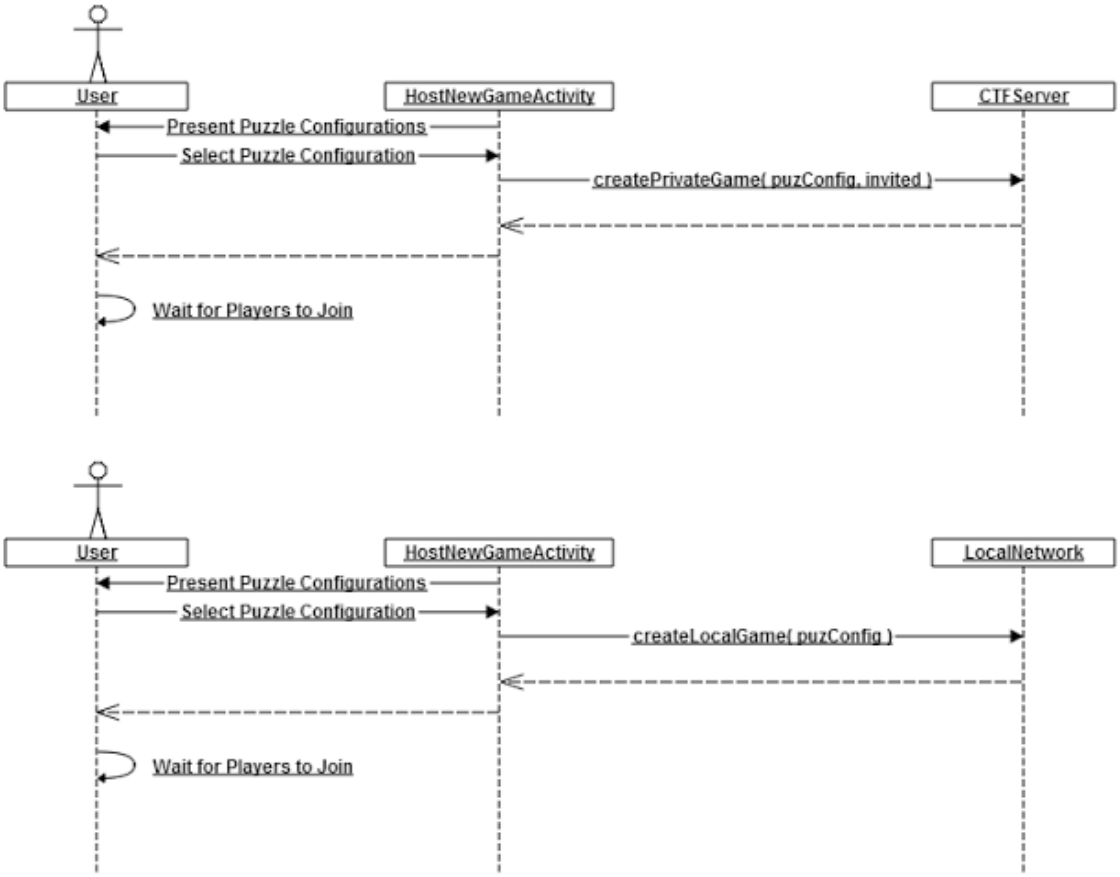


These classes basically hold a game state and contain the logic for updating that state. The *tick* function in their common interface is called on every tick of the game logic within the game loop. They share a common interface because any device hosting a game will also serve as a client for the game—that way, the GameActivity will simply need to hold an array of *GameTicker*s that it will tick every time it goes through its loop. The array is either size 1 (for a client) or size 2 (for a host; the array holds one of each of *GameHost* and *GameClient*.

Both classes need to receive input from one another and send input to one another. The *GameHost* will be updating the game based on input from the clients and then broadcasting information about the new state. The *GameClient* will be reading input directly from the user and will send that straight to the host device—when it receives update information from the *HostDevice*, it updates its local copy of the game state to match the authoritative version and then render's the game on the screen for the user.

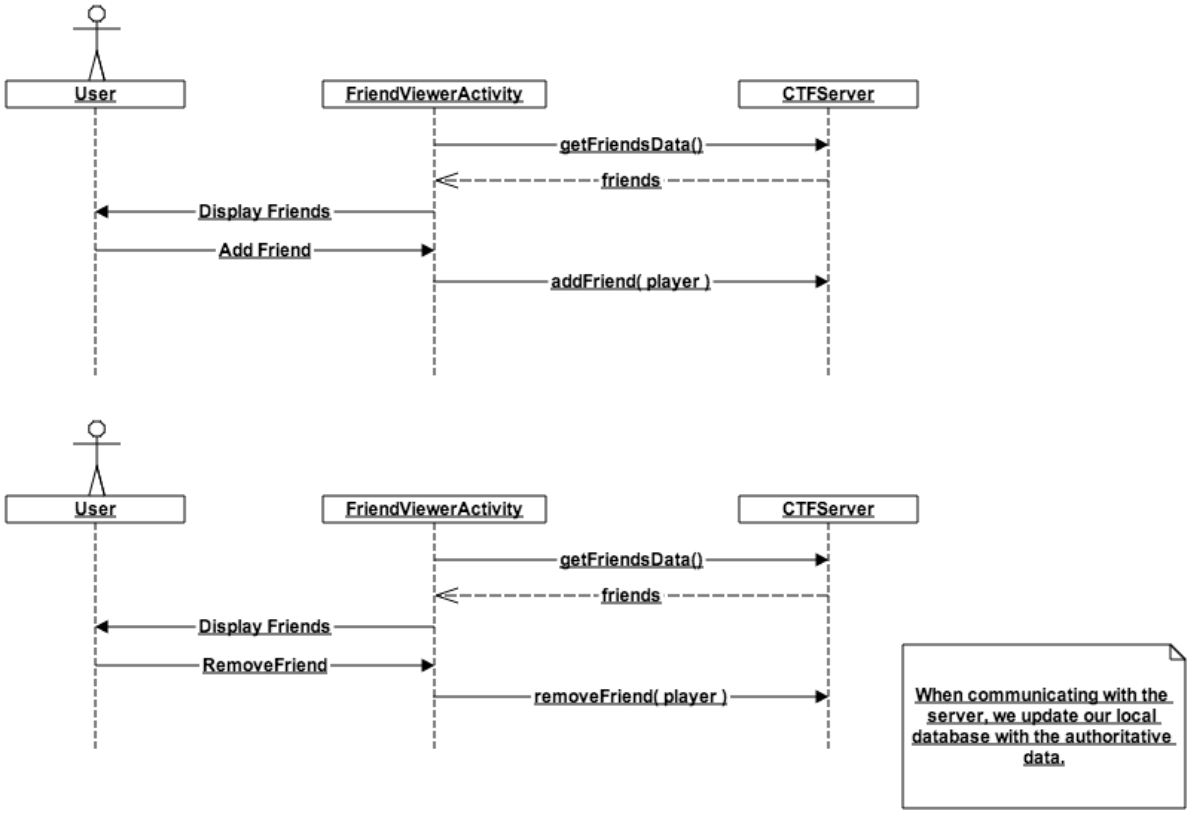
6.2 Sequence Diagrams

6.2.1 Create Game



These sequence diagrams depict the flow of logic for creating a private game and a local game. They are quite similar except that the *HostNewGameActivity* is using the *CTFServer* class for private games and the *LocalNetwork* class for local games. Notice that there is no use case for creating a matchmaking games—these are automatically created on the server when they are needed.

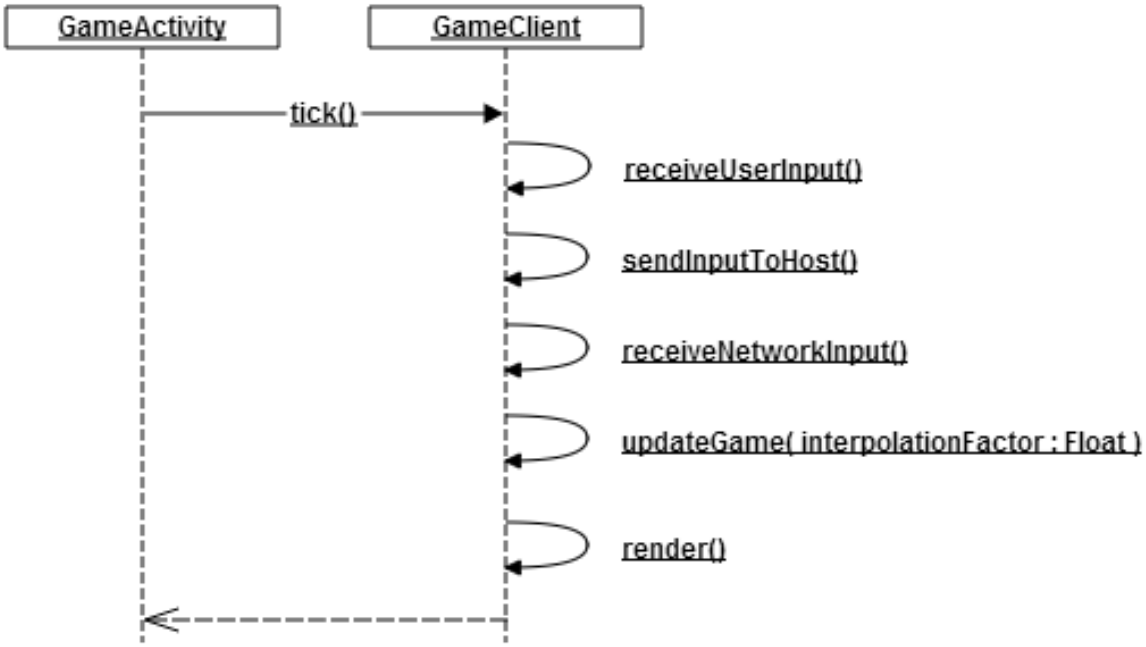
6.2.2 Display Friends



This diagram simply shows a user interacting with the friends screen in the menu—if she decides to add or remove a friend, the activity makes the corresponding request to the server.

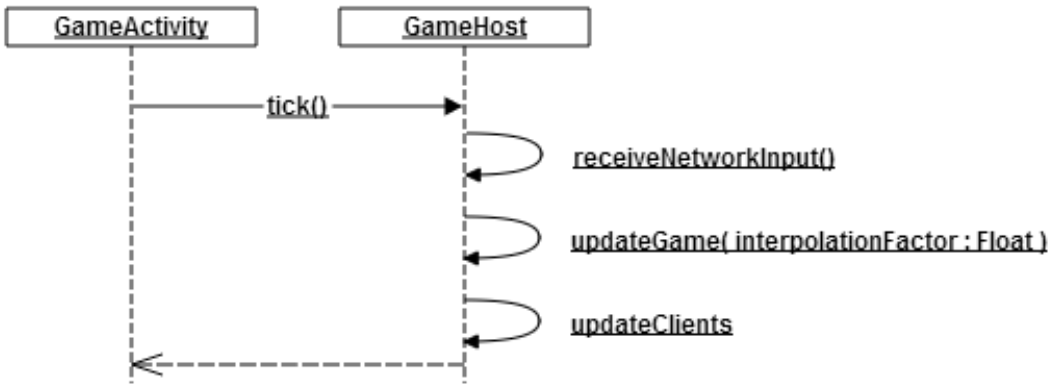
A user will not be able to add or remove a friend if she is not logged in to her account. She may still view her friends, however, as they get cached to the local database every time the device accesses the server. Perhaps even pending adds/deletes can be placed in the local database to be pushed the next time the user makes a connection with the server.

6.2.3 Game Client



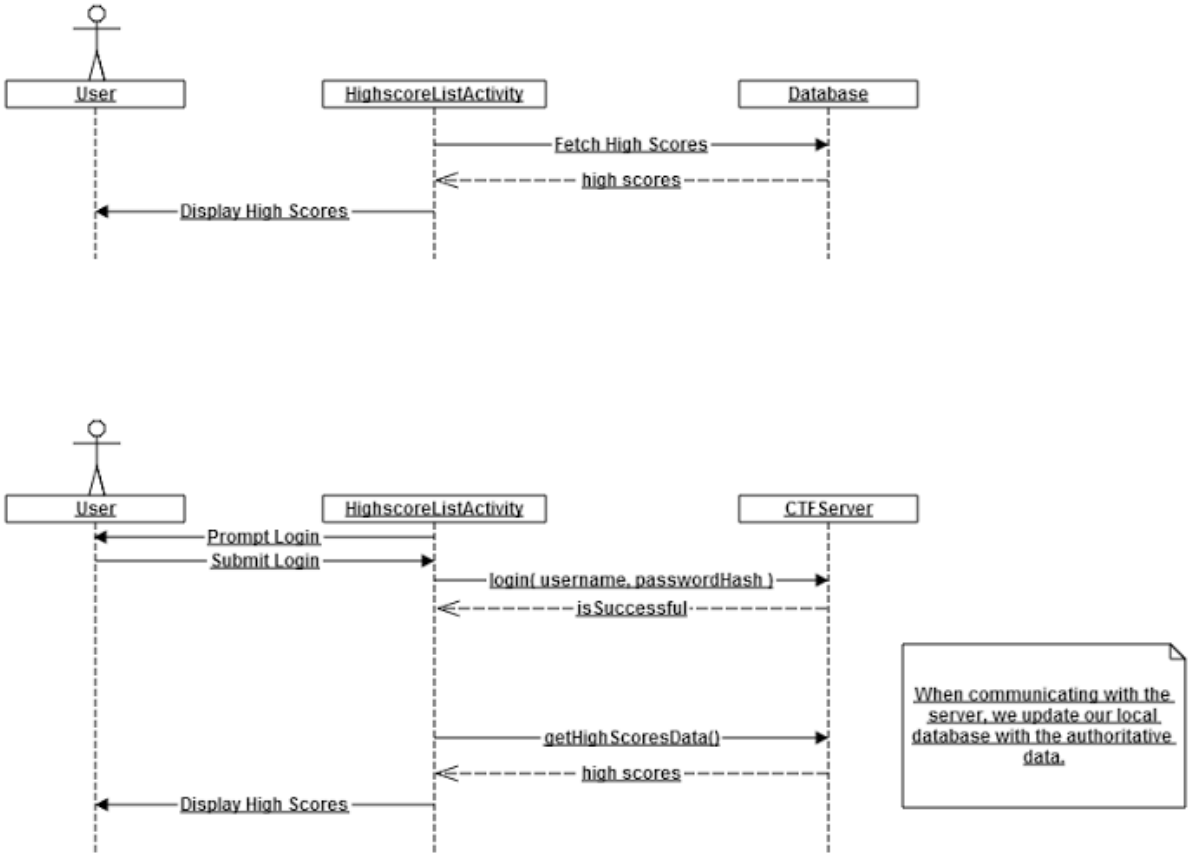
This simply shows the order of execution for the methods that a *GameClient* runs on every tick of the game. The *interpolationFactor* is intended to smooth out any animation or movement that happens either early or late in the framerate.

6.2.4 Game Host



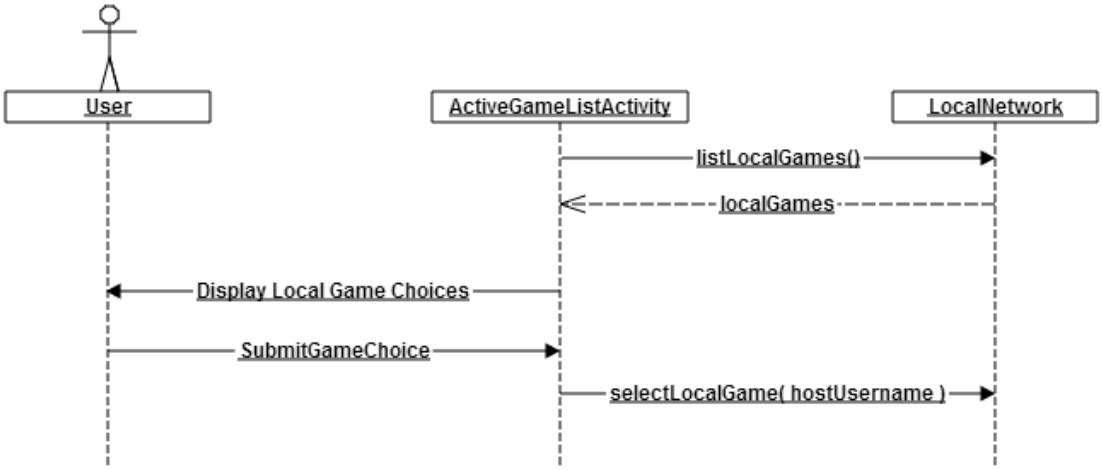
Just as with the *GameClient* diagram, this shows the order of execution for the methods that take place within a host device every tick of the game.

6.2.5 Display High Scores



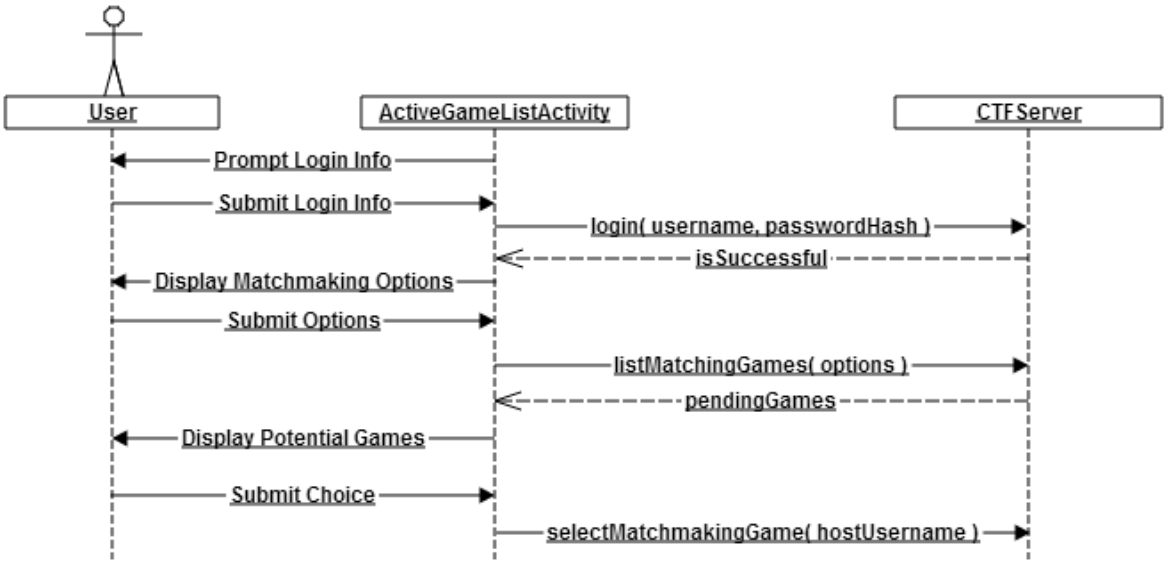
As could have been done with the *Display Friends* diagram, this one showcases the two ways the system shows high scores. If the user is logged in, the *HighscoreListActivity* requests the high scores from the *CTFServer* (assuming the user is logged in) and displays the returned data. The device caches the high score data locally regularly so that (as depicted in the top sequence), a user may view high scores directly fetched from his local database.

6.2.6 Join Local Game



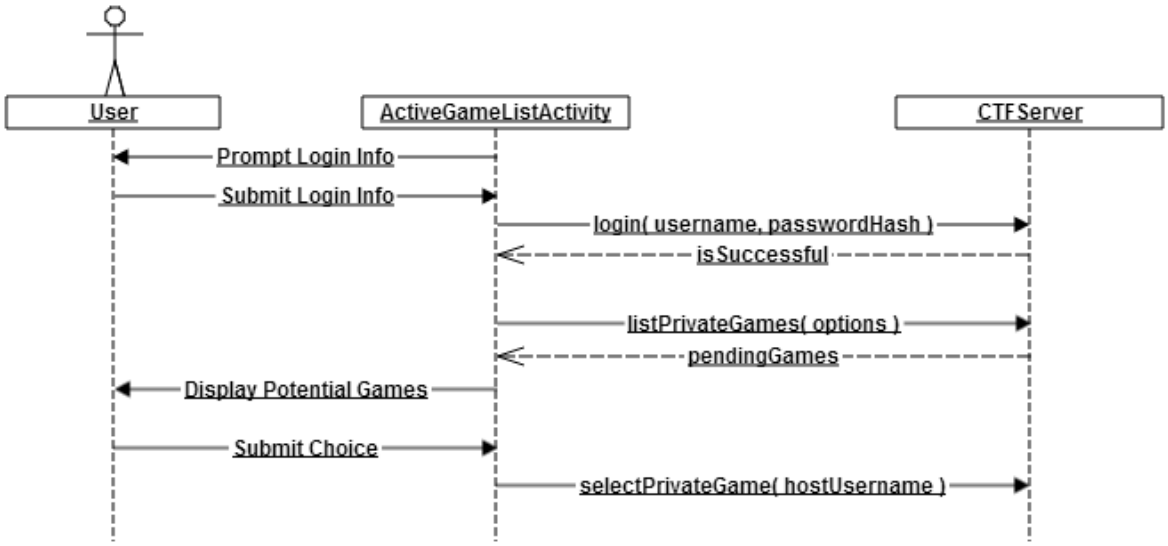
Joining a local game is rather straightforward—the *ActiveGameListActivity* gathers the local games from the local network and then displays those to the user. The user selects on of these games and the *ActiveGameListActivity* notifies the host of the game that the user wishes to join.

6.2.7 Join Matchmaking Game



To be able to play a matchmaking game, a user must be logged in. Once he his, he selects his filtering options for a matchmaking game and the *ActiveGameListActivity* forwards his suggestions to the server, which responds with a list of potential games. The user selects one and the server is notified that the user is joining the game.

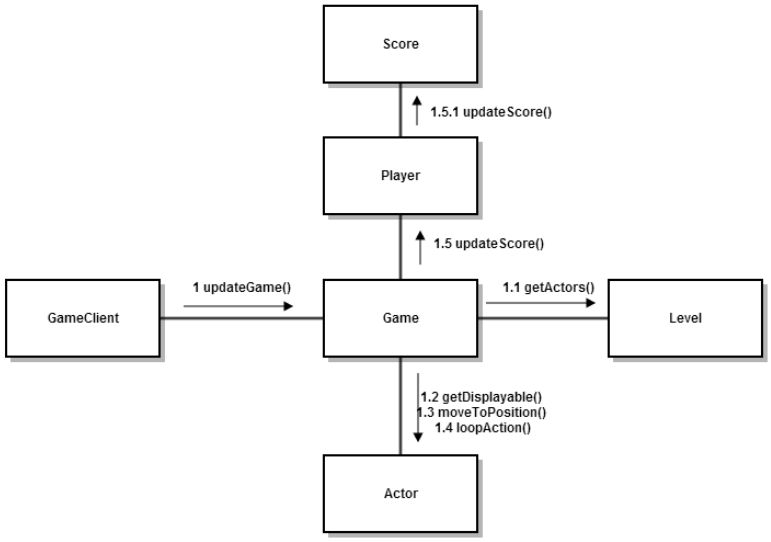
6.2.8 Join Private Game



Joining a private game is very similar to joining a local game—potential games are displayed and the user selects one. The only difference is that the Local Network is replaced with the *CTFServer* and, accordingly, the user must log in beforehand.

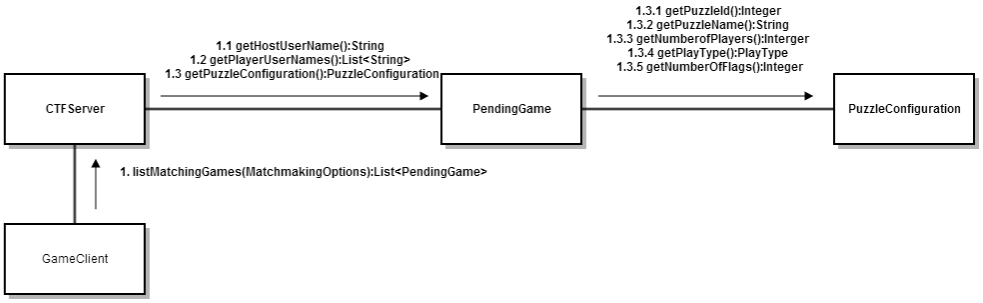
6.3 Collaboration Diagrams

6.3.1 Game



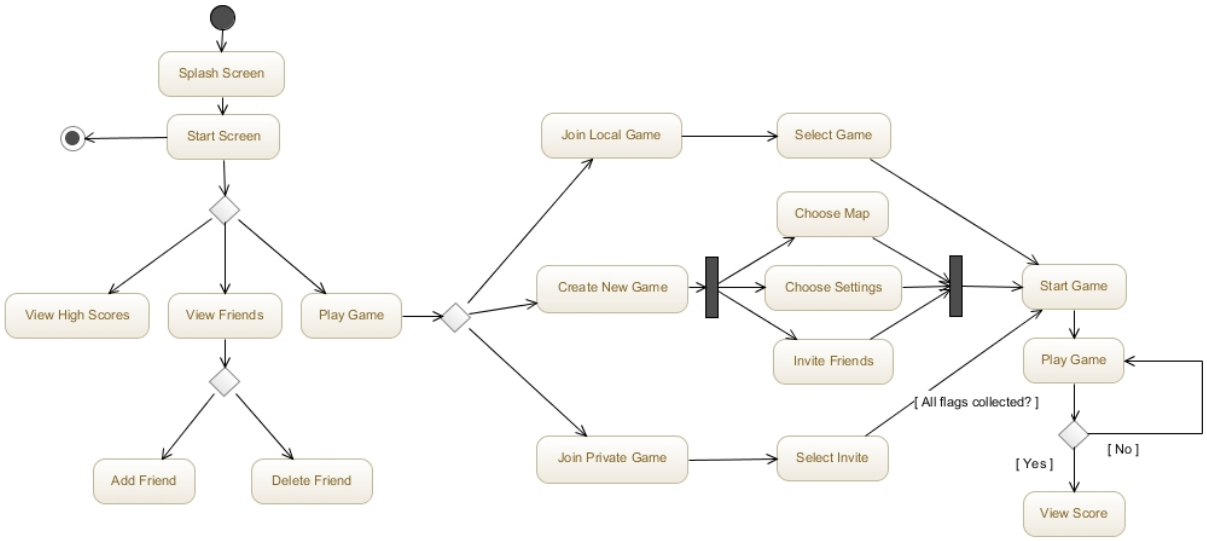
This diagram shows the collaboration between classes in the main game activity when the client is updating the local game. After receiving the game data from the game host, the GameClient class will call updateGame() on the Game class. The Game class will then get all of the actors from the Level class with the getActors() call, and iterate over them, calling getDisplayable(), moveToPosition() and loopAction() on each in order to update them. Once it has finished updating all of the Actors, the Game class will update the local score by calling updateScore() on the Player class, which will call updateScore() on the Score class.

6.3.2 Network



This diagram models the collaboration between classes for getting a list of games in matchmaking. The GameClient class starts by sending a MatchmakingOptions object to the listMatchingGames() function in the CTFServer. The CTFServer then gets the relevant PendingGame objects by calling getHostUserName(), getPlayerUserNames() and getPuzzleConfiguration(). Lastly, the PendingGame class will get the info from the PuzzleConfiguration class by calling getPuzzleId(), getPuzzleName(), getNumberOfPlayers(), getPlayType(), and getNumberOfFlags().

6.4 Activity Diagram



This diagram outlines the primary flow of the application. After the splash screen, user will see a start screen with three options: View High Score, View Friends, and Play Game. Selecting ‘View High Score’ will show the high score list, and selecting ‘View Friends’ will allow a user to add and delete friends. When a user selects ‘Play Game’, they will have three more options: Join Local Game, Create New Game, and Join Private Game. The ‘Create New Game’ screen will allow a user to invite friends, choose the map, and change other game settings. Selecting ‘Join Private Game’ will let a user select a game from a list of current invites, and ‘Join Local Game’ will let a user select from current public games on the local network. Once the game has been started, users will play until all of the flags have been collected for that map. Users will finish at the View Score screen, where they will be able to see the score for their last game and high scores for that map.