UNLEASH

Sheldon Roddick & Ben Baxter

Contents

Description

General Overview

Unleash is a multiplayer, real-time, augmented reality game for Android OS. The basic premise of the game is that players are in a combat arena where they must race each other to power ups which spawn in their local area. When enough of these power ups are collected the player is able to unleash an attack on his/her opponent. The last person standing is the winner.

Technical Overview

Unleash runs on a new wireless protocol that has been in development over the course of the Unleash project. This protocol handles automatic and dynamic connection of multiple devices over the WiFi Direct framework. A one-to-many network is created using Java sockets on the WiFi Direct group network framework. This network is used to transfer locational data, among other packets, to each device in order to update each user of the game's state.

The game interface is a Google Maps Fragment on which custom markers are drawn to represent players and game entities. A second fragment is inflated on top of the map fragment for the game's "Unleash" command, and to show the player's power level.

Market Survey

The current mobile gaming market is saturated with copies of the same game concepts with new art pasted over them. It seems like the entire market is rehashed puzzle games and countless twists on the traditional “iCopter” game (Flappy Bird being the biggest one). There are very few largely innovative products in the mobile gaming market, despite there being a growing demand for such games. That is where this game sits.

Augmented reality is a fairly new field in general, and the mobile platform is the perfect ground to base an augmented reality project in. Due to the devices being so portable, it offers a huge array of content to be tapped into.

There aren’t any games on the market that do anything similar to what Unleash does from a gameplay perspective. Concept-wise though, Google’s “Ingress” game has implemented the concepts of real-world interactivity through an augmented reality interface to drive their point-of-control game. There aren’t any other applications that allow players to interact with each other in a fast-paced, competitive, real-world game through an augmented reality interface.

Code Description

APIs

Unleash employs the use of the Google Maps API as the base interface through which the game is played. Along with the Fused Location API, it is used to draw the game entities in an easily-readable way, on a map that represents the player's local area.

The WiFi Direct API is used to create the network framework on which the communications network is built. The purpose of the custom communications network being built on top of WiFi Direct is to allow players to connect to each other and play Unleash without a pre-existing wireless connection or an intermediary switch or routing device.

Procedure

From start to finish, the Unleash game and its underlying connection and communication protocol goes through many steps to connect users for multiplayer gameplay.

From the title screen the players press the "Join Game" button, which launches the Google Maps fragment activity along with the Join Game fragment activity. The Join Game activity is inflated on top of the Google Maps activity, rendering the map invisible to the user until later on. Meanwhile, the Maps activity (hereafter referred to as "Play") has launched the Fused Location service which it uses to track the location of each player.

The devices begin their search for other users running WiFi Direct requests, and call out to them for connections. Once one device connects to another, they decide which one will host the WiFi Direct group. After they have made their decision the host begins his "Host Service," which detects incoming connection requests from other users. Once a connection attempt is detected, a "ClientService" is created on the host device for each user (one per user that connects to the host). These Client Services handle data transfer between the host and that device. The other users, meanwhile, have started a "ClientDeviceService" on their own device which attempts to form a socket connection to the host device. After starting their respective services, the clients will stop trying to discover other WiFi P2p connections, while the host will continue trying to discover other people to form the one-to-many group network.

Once this connection has been made (signified by a packet sent by the client with the header "255"), the host tells the user what their user number is for data concurrency purposes. Full data transfer can now begin.

After all users are connected, they press their "ready" buttons, sending packets to the host that let the host know whether they are ready or not. Once all users are ready the host is able to start the game. Once the start game packet is detected by the users, they get rid of their Join Game fragment and are presented with the map screen. It is now that the game begins, with the host generating location data for power-ups. These power up locations are transmitted to the users and they then draw the power ups at the same location.

Once a user has a high enough power level to attack another user, they can press the Unleash button. When this button is pressed, it sends an UnleashAttack packet to the host, who then broadcasts it to all devices. The devices draw a series of concentric circles around the epicenter of the attack, representing the shock wave. Users caught in this shock wave are killed, making them unable to interact with the game.

When there is only one user left alive the end game is triggered. The end game screen shows the name of the user who won the game.

Spec Sheet

• WiFi Direct	WiFi Direct is a relatively new P2P technology that allows devices to connect to each other by forming groups, and removing the necessity for an intermediary wireless switch/router. It has a projected speed of up to 250Mbps when employing the 802.11 networking protocol.
• Fused Location Provider	Used to provide an efficient and easy way of tracking the players' location accurately enough to play the game.
• Google Maps	The Google Map allows us to visualize the players' local area so they may recognize buildings, roads, and landmarks during gameplay.
• Multithreaded Communication Handling	The one-to-many nature of the network that is created in this app forces a multithreaded approach to handling the network communication services in conjunction with the Map/UI thread

Features

• Multiplayer gameplay	Play with your friends anytime, anywhere!
• Router-free connectivity	No need for a router or switch to connect with your friends, just hit connect and play!
• Personalized names	Choose your name when the app first launches, this is how you will appear to your friends when you win!
• Google Maps caching	Once you've loaded your local area map from Google services you won't need to be connected to the internet to launch the game.
• Music	Music plays to accompany the battle with your friends
• Tutorial	A thorough tutorial explaining how to play the game
• Bug Reporter	Allows the user to choose which email program they want to use to send the bug report. Opens a new message in that program with a default message template for the user to complete.

Test Cases

Connection Rates

Note: Devices have had their persistent and remembered P2P groups manually forgotten between test sessions. It is possible to do this programmatically, however the method call is hidden in Android OS. This means that it may be removed without notice on any update, causing large compatibility issues. We have opted for manual group forgetting for this reason.

Two Devices:

Connection rate is virtually 100% when connecting two devices. Sometimes it can take longer for the P2P group to form, however they will almost certainly connect.

Three Devices:

Again, connection rate is near 100%. Once two devices have connected, the host will reach out to the third for a connection. The only issue here is the possibility of a cyclic connection request situation (where R is the request, device1 R device2, device2 R device3, device3 R device1).

Four+ Devices:

As more devices are added the possibility of isolated groups occurs. The possibility of having the hosts communicate to join groups (see figure A) and determine a “Master Host” has been discussed, however this is a massive undertaking which isn’t possible at this point due to time constraints.

Having people join one at a time (two people join, create a group, and then have other join one at a time so there is a maximum of two people broadcasting services at any time) removes the issue of isolated groups and maintains a high connection rate.

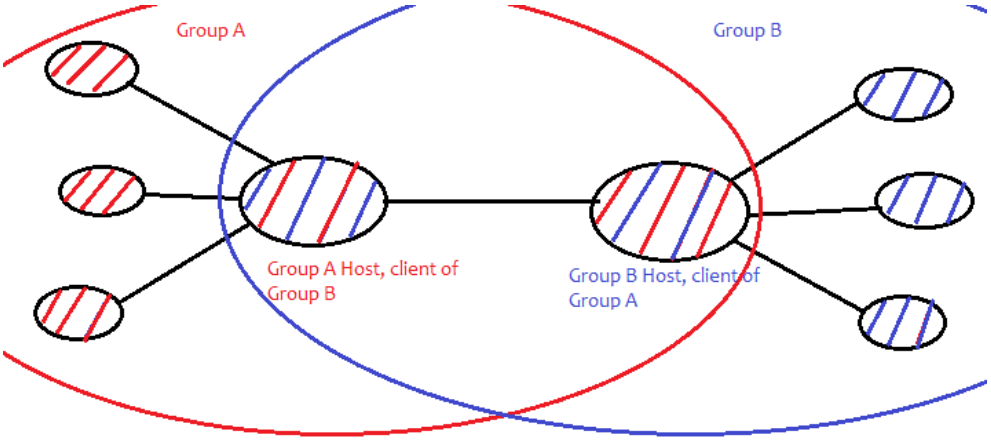

Figure A – Conceptual WiFi Direct hybrid devices, host of one group and client of the other.

Performance

Network Performance

No specific quantitative tests have been done on the throughput of the network, however some qualitative observations have been made.

The data transfer speed has so far been sufficient to support all of the devices we have tested with. Network logs have shown consistent performance with minimal latency time in the majority of cases.

Occasional packet delay has been observed, however this is to be expected on a network designed namely for file-sharing. This delay has not had any effect on gameplay, as locational information is updated quickly enough for the delayed packet to become negligible.

Data integrity has been solid on the network with the exception of one bug (detailed later in the report).

Device performance

Early in development certain connection calls and other tasks weren't being handled in an acceptable manor, causing large performance issues and hang-ups. Since then the application has been optimized to the point where it runs very well on modern Nexus devices.

The devices are responsible for drawing and updating the location of multiple users as markers on Google Maps in fairly rapid succession. The devices we tested the application with have done very well in this regard, updating the positions of users seamlessly.

The battery life has been affected as little as possible, however since it is a resource-intensive application it does still affect it. Measures have been taken to ensure that only necessary processes and services are running (locational services, networking services, etc. only run while the game is in progress).

Permissions

ACCESS_WIFI_STATE
Need to detect the WiFi state on the device for connection purposes.

CHANGE_WIFI_STATE
Need to change the WiFi state on the device during connection.

ACCESS_NETWORK_STATE
Again, needed to detect the network state for connection

CHANGE_NETWORK_STATE
Need to change the network state during connection.

INTERNET
Needed to open network sockets for connecting to other devices

WRITE_EXTERNAL_STORAGE
Needed to use SharedPreferences for detecting if it is the first time the application has been started (showing tutorial), and to store the user's name.

ACCESS_COARSE_LOCATION
Needed for accessing the user's location

ACCESS_FINE_LOCATION
Needed for accessing the user's location more accurately during gameplay

READ_GSERVICES
(com.google.android.providers.gsf.permissions)

Needed for using Google's services.

Marketing Strategy

It has been shown that the best way of making your mobile application widely used is to make it free to pick up and play. Major hit games like Flappy Bird and Clash of Clans would not have seen anywhere near the amount of players that they currently have if they had been paid apps. In accordance with this, we are making Unleash free for everyone to download and play.

Apps need to make money though, and so we have a few ideas for this.

First of all is ad revenue. No one likes ads, but almost everyone accepts them to some extent as a way for a development team to make money. If Unleash starts to become popular, we will add in some non-intrusive advertisements on the title screen, where they will be well out of the way of any game-relevant options to prevent accidental pressing of these ads. (Nothing makes us more frustrated with a game than ads in places where it's easy to accidentally press them and disrupt gameplay).

The next revenue-earning strategy is simple, hats. Buying items to use in-game for real money is nothing new. However, most companies do this very poorly. They offer exclusive items that give distinct advantages over non-paying players, something that simply cannot be justified in a game. To allow a person to, as the phrase goes, "pay to win" is contrary to every game design principle that there is. However, some companies do this very well. Valve does this incredibly well with Dota 2 and Team Fortress 2, in which no purchased items will actually affect gameplay. Their cash shop items are simply cosmetic alterations to characters. Adding a paid option to Unleash for players to customize their player icon (probably by importing images) would be a way to generate revenue in Unleash.

The third option is not directly tied to the Unleash application, but rather involves stripping the networking protocol that we have developed out of Unleash and making it more robust. Once this is done it could be sold or licensed to other game development studios for large amounts of money, as it is a highly complex protocol that takes a significant amount of time and knowledge to implement.

Bugs

ReadObjectNoData

Referenced on page 9

ReadObjectNoData is a method of the ObjectInputStream that is called when there is some sort of integrity issue with the serialized data. When the receiver (the client reading from the input stream) has a version of the class that the object it is trying to read is derived from, it recreates that object using the default constructor of its local class.

This is causing an issue where, due to some unknown cause, an issue in the data arises which causes the receiver to register the user's location as 1.0, 1.0 (default latitude and longitude of the User class).

The effect that this has on the game is significant. If this bug happens, the receiver will appear to not draw the user which is sending this corrupted data on the map. It is drawing the user, however it is drawing the marker at location 1.0, 1.0. This obviously means that unless you are here you won't see the marker.

We have debugging logs that trace the data all the way down to the line where we send it (which output the correct data all of the time), and then on the line that we receive it at (which is where the bug occurs). There is no other code that we are running between these locations, so the bug is not of our own doing.

The perplexing part of this bug is that it doesn't happen all of the time. This leads us to believe it could be caused by some sort of perceived noise on the network that the WiFi Direct code may pick up.

Group Formation Failure

It is possible for a group formation to fail outright, causing the whole connection procedure to fail with it. This catastrophic failure is due to the nature of WiFi Direct's group formation protocol. If the Group Owner Intent (GO Intent and GOI in the following figures) is the same, there is a high likelihood of the group formation failing (approx. 50%).

The Group Owner Intent is determined by the steps outlined in figure B, and the handling of that intent is outlined in figure C. These images are taken directly from Google's patent page for determining group owner intent.

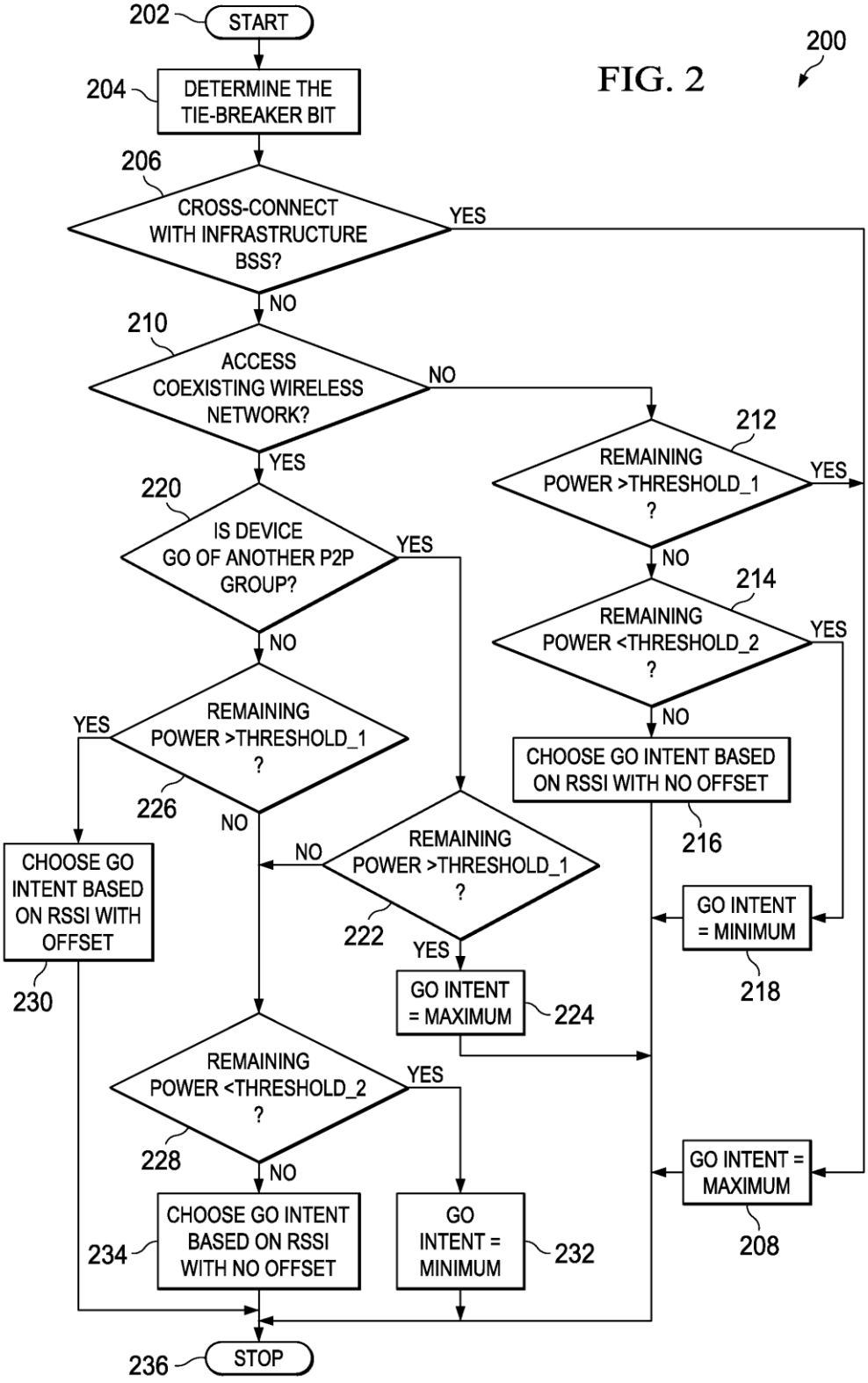
FIG. 2
200
202
START
204
DETERMINE THE TIE-BREAKER BIT
206
CROSS-CONNECT WITH INFRASTRUCTURE BSS?
YES
NO
210
ACCESS COEXISTING WIRELESS NETWORK?
NO
YES
212
REMAINING POWER >THRESHOLD_1 ?
YES
NO
214
REMAINING POWER <THRESHOLD_2 ?
YES
NO
216
CHOOSE GO INTENT BASED ON RSSI WITH NO OFFSET
218
GO INTENT = MINIMUM
220
IS DEVICE GO OF ANOTHER P2P GROUP?
YES
NO
222
REMAINING POWER >THRESHOLD_1 ?
NO
YES
224
GO INTENT = MAXIMUM
226
REMAINING POWER >THRESHOLD_1 ?
YES
NO
230
CHOOSE GO INTENT BASED ON RSSI WITH OFFSET
228
REMAINING POWER <THRESHOLD_2 ?
YES
NO
232
GO INTENT = MINIMUM
234
CHOOSE GO INTENT BASED ON RSSI WITH NO OFFSET
208
GO INTENT = MAXIMUM
236
STOP

Figure B

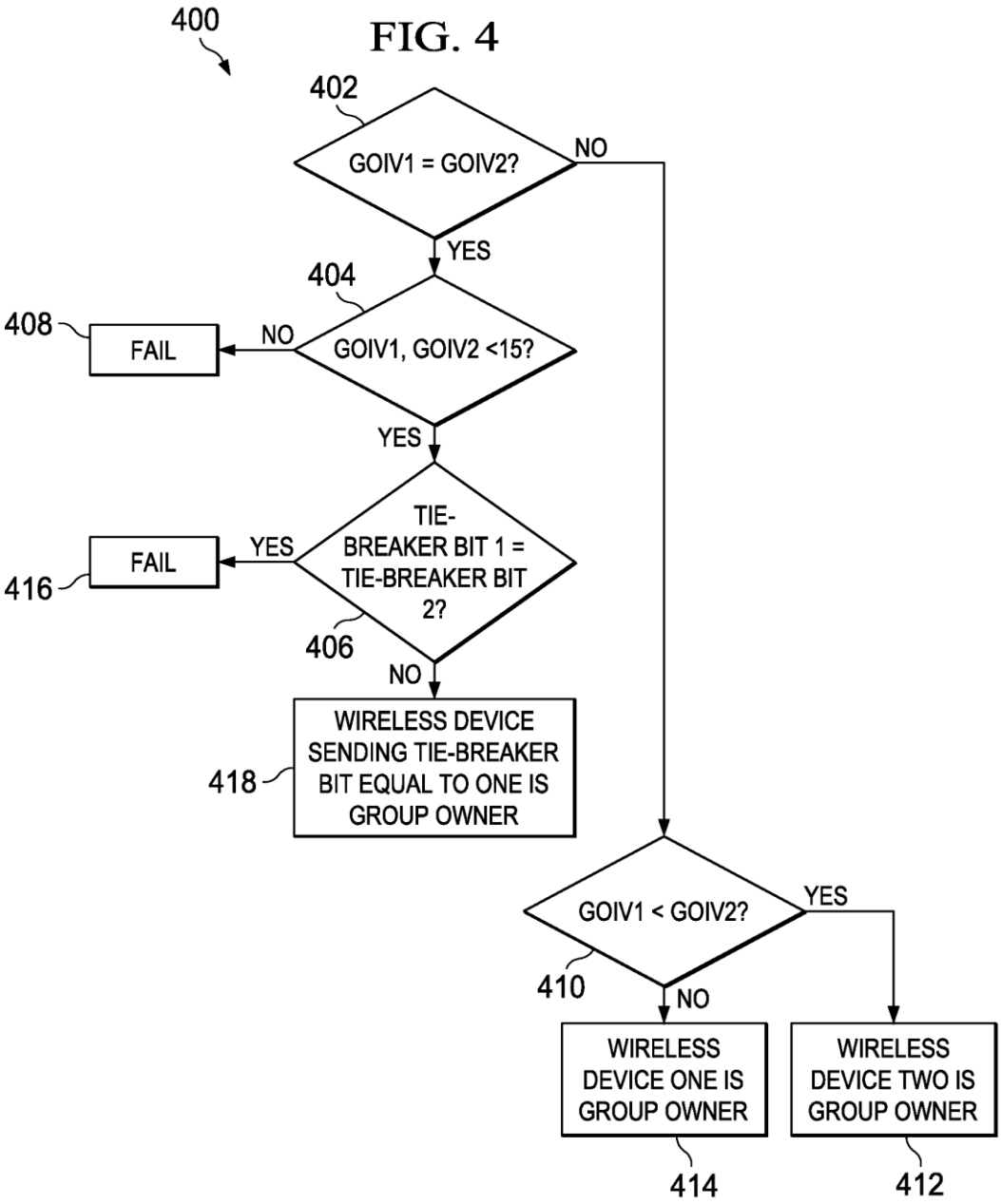

Figure C

As these figures outline, there is a high likelihood of failure. Figure B shows that being assigned a group owner intent of 15 (max) is not uncommon, especially with newer devices, which will cause the connection to fail if they attempt to connect.

Measures have been taken to prevent this (assigning a random GOI of 1-14, namely) however failure is still a possibility.