Freeze Tag Frenzy

CS 490 | Masters Project | M.S. in Computer Science

An exploration of the process used to create a mobile real-time multiplayer game.

Lathaniel Mejias

Lathanielm@gmail.com

Northeastern Illinois University

Contents

1. Introduction 2

2. Background 2

3. Playing the Game 3

3.1. Rules 4

3.2. How to Win 4

3.3. Other Dynamics 5

3.3.1. Solid Objects 5

3.3.2. Hideable Objects 5

3.3.3. Camouflaging 5

3.4. Controls 5

3.4.1. Movement 5

3.4.2. Action Buttons 6

4. Implementation 6

4.1. Tiled Map 6

4.2. Animation 8

4.3. Collision Detection 9

4.4. Networking 9

5. Limitations 11

5.1. Lag and latency 11

5.2. Wi-Fi Networks 12

5.3. Score 12

6. Conclusion 13

7. References 14

1. Introduction

Multiplayer gaming, a branch of video games as old as the first video game itself (Baker, 2010), is a type of gaming that can now be split into various other branches. For instance, there are one-on-one, n-vs-n, turn-based, real-time, local multiplayer, online multiplayer, massively multiplayer online, and many others. A relatively new branch of multiplayer gaming that is quickly flourishing as of late is mobile multiplayer gaming. In 2012, Google released a Software Development Kit (SDK) and Application Programming Interface (API) called Google Play Services (Google Play Services, 2012). While still not as popular as single player mobile games, the release of these services has made the creation of multiplayer games more obtainable for the average developer.

The purpose of creating this game is to explore various techniques and methods used to develop an online multiplayer mobile game. Going forward, it may help to understand the origin of this game, followed by how to play the game. This will be followed by the implementation of the game including its tiled map, animation, collision detection, and networking. Lastly, no game is without its limitations, and this game is no exception—facing limitations in regards to lag, Wi-Fi, and scoring. After undertaking such a project, one has become more accustomed and familiar with the challenges and obstacles associated with developing online multiplayer games.

2. Background

As previously mentioned, multiplayer gaming started with the first video game, Nimrod, created by a UK electrical engineering firm in 1951. This paper, however, is more concerned with online multiplayer. Skipping forward a few decades, packet-based networking started to mature which eventually led to the creation of the Internet in 1983 (Cerf & Kahn, 1974) (Stewart, 2000). This opened the doors for online gaming, both single and multiplayer.

The roots of this game in particular date back to 2003. The original Freeze Tag Frenzy started as an online PC game created using a software suite for creating and playing online games called BYOND (Mejias, 2003). *Figure 1* and *Figure 2* feature screenshots of this original version. While much of the core gameplay is the same, a great deal of the functionality featured in the original was provided by BYOND and its API. This functionality included, but is not limited to, networking, communication, synchronization, graphics handling, and movement. As for this iteration, the networking is provided by Google Play Services, and some of the basic game features were built upon the LibGDX game engine (Zechner, n.d.); however, the communication, synchronization, and movement were handled as part of the creation of the project. Given more time to spend on core development, the original has additional features (after several updates) such as items, additional maps, and computer players. These features may reach the mobile platform in the future.

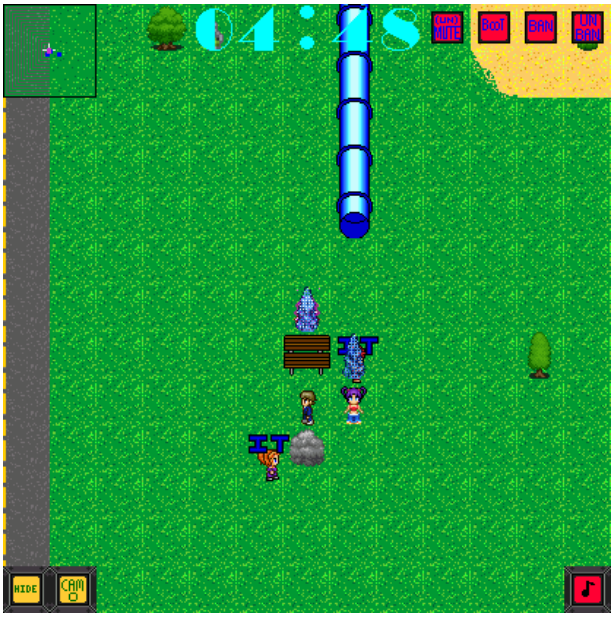

Figure 1

Figure 2

3. Playing the Game

While the game does not take much time to learn, it is not quite self-explanatory. The game has two different types of players, ITs and Non-ITs. This section will detail the rules, what it takes to win, and other factors affecting gameplay.

3.1. Rules

In order to explain the rules of the game, it is necessary to understand the different players involved. However, due to the similarity of the player terms, it is possible to misread these terms. Therefore, the following section makes use of colored text to simplify the explanation of the rules. *Figure 3* explains the meaning of these colors.

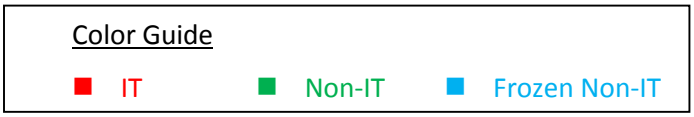

Figure 3

3.1.1. If an IT player runs into a Non-IT player, the Non-IT player will become frozen.

3.1.2. If an IT player runs into a Non-IT player that is currently frozen, the IT player will be able to pass through the Non-IT player. This behavior prevents cheating.

3.1.3. If a Non-IT player is frozen, it will be unable to move.

3.1.4. If a Non-IT player runs into an IT player or an unfrozen Non-IT player, they will not be able to pass through them.

3.1.5. If a Non-IT player runs into a frozen Non-IT player, that frozen Non-IT player will become unfrozen.

3.1.6. If a Non-IT player is to be frozen a third time, instead of becoming frozen, the player will become an IT.

3.2. How to Win

The concept behind winning is a simple one. If the round timer reaches 0:00 before all the Non-ITs are frozen or converted, the Non-ITs win the round. However, if all the Non-ITs are frozen at once, or there are no more Non-ITs remaining, the ITs win the round. If the IT or ITs drop out, the Non-ITs win.

3.3. Other Dynamics

3.3.1. Solid Objects

- If a player runs into a solid object, they will collide and stop moving.

3.3.2. Hideable Objects

- The Non-ITs have the option of hiding behind certain objects. These objects are not apparent just by looking.
- The ITs have the option to check behind hideable objects, expelling any Non-IT that may be hiding behind the object.

3.3.3. Camouflaging

- In certain areas, Non-ITs can attempt camouflage with their surroundings.
- If an IT runs into a camouflaged Non-IT, they will no longer be camouflaged and become frozen.

3.4. Controls

The controls for the game revolve entirely around the device's touchscreen.

3.4.1. Movement

- To move the player, simply touch one of the four regions pictured in *Figure 4* and the player will move in that direction.
- The player will continue to move as long as the direction section is being touched. Upon release, the player will stop moving in that direction.

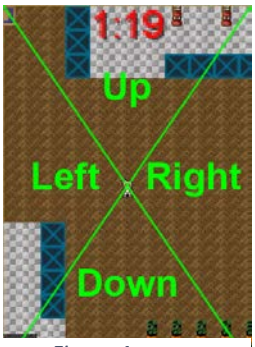

Figure 4

3.4.2. Action Buttons

Depending on whether the player is an IT or Non-IT, the player will have different action buttons available. These actions can be initiated by simply tapping the button.

IT Buttons

- **Check**: Allows the IT player to check behind hideable objects. If a Non-IT is hiding behind the checked object, the Non-IT will be forced out.

Non-IT Buttons

- **Hide**: Allows the Non-IT player to hide behind hideable objects.
- **Camo**: Allows the Non-IT player to possibly camouflage with its surroundings.

4. Implementation

Certain implementation aspects help make this game possible. These aspects include the use of a tiled map, animation, collision detection, and, most importantly, networking and multiplayer consideration.

4.1. Tiled Map

The main playing field is a map consisting of 30 rows and 30 columns of tiles. Each of these tiles measures 32 x 32 pixels (See *Figure 5*). What may not be evident is the use of layers. The map currently has five layers of tiles--not all layers are completely filled with tiles. The use of layers allows for the choice of which tiles should render in front of

which tiles. Also, if all tiles of a certain type, say obstacles, are exclusive to a layer, it makes looping through the layer much faster.

It is important to note that the map is not visible in its entirety as to add to the challenge of having to find the other players in order to tag them or unfreeze them. To make the game fair across devices with different resolutions, the viewport of the map is limited to a resolution 480 x 320 pixels. This gives all players an equal view of 15 x 10 tiles.

This design is not by accident. The tiled map helps game development in a few ways. Tiled maps allow for visibly divided locations and visible units of movement, if movement is locked to tiles—which, in this case, it is. The biggest consideration was the multiplayer aspect. This will be explained in more detail with the game limitations.

The map itself can be created with a GUI editor called Tiled (Lindeijer, n.d.). With this editor, one can drag and drop tiles onto the map as seen fit. The editor then stores this map as an XML file. This XML file is read into the game as various objects by the LibGDX game engine, such as Tiles, TiledMaps, and TileLayers.

Thus, to create a TiledMap object—the core of the tiled map, the following process is performed:

Parse XML file (created by GUI editor)
For each element in the file
    If element's name is "tileset"
        Add element to map's tileset
    Else if element's name is "layer"
        Create a layer object out of the element
            For each tile in map's tileset with an ID belonging to this layer
                Create a new tile
                Place tile in its respective position
    Else if the element's name is "objectgroup"
        Create an object group out of the element
        … (objectgroups are currently not used in this game)

Here is the tiled map in its entirety with grid lines that are not normally visible.

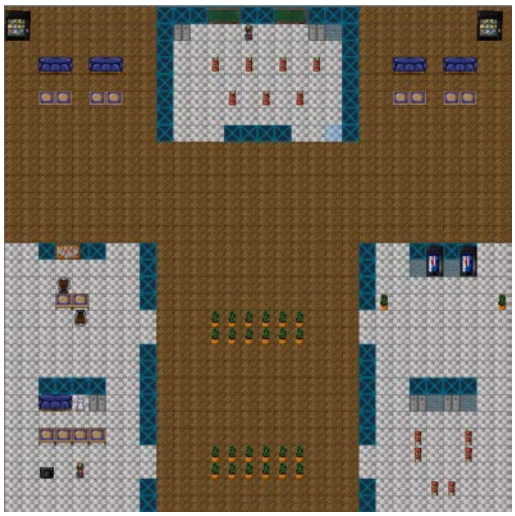

Figure 5

4.2. Animation

While animation may seem like solely a cosmetic feature, it also plays a role in giving the user feedback to certain actions. A superficial example is that when the player runs, it will make a running animation. A more functional animation example would be when the Non-ITs camouflage. When this happens, the player cannot move until the animation has finished, leaving the player temporarily defenseless and vulnerable. ITs, on the other hand, experience a useful animation. Whether an object is hideable or not is not apparent just from looking at them. The IT can attempt to "check" the object, if it animates, then it is hideable. This gives the IT ideas as to where the Non-ITs may be hiding at some point.

The implementation for an animation is not so different from a static image. In essence, an animation consists of several static images (referred to as "frames") that change (slightly) from one image to the next. To create an animation for this game, three properties must be defined: the frame duration, a list of the frames, and the play mode. Upon setting those attributes, the currently displayed image will change based on a monitored time value. The simplified process is as follows:

Update(timeChangeFromLastCall)
    If the user is walking
        *Add **TimeChangeFromLastCall** to **animationTime***

Else
*Set **animationTime** to **0***
*Set the character's **image** to the **walking** animation based on **animationTime***
*Using animationTime, calculate the current frame based on the animation's **frame duration***
Check play mode to get the correct order of the images (i.e. do they loop, go in reverse, go in random order, etc.)
*Set the character's **image** to this calculated current frame*

4.3. Collision Detection

Collision detection is a crucial element to the game—without it, the game would not be playable. Given that the game makes use of a tiled map, collision detection can be simplified slightly. Since the player can only move in four directions (up, down, left, and right), and they can only move on a tile by tile basis, collision detection can be boiled down to checking the tile adjacent to the player in the direction they are moving. For example, if the user is moving to the right, all that would be necessary to check is if the tile directly to the right of the player is free of obstacles. The simplified process is as follows:

*Set **tentativeCell** to the cell adjacent to the player in the direction they are facing*
*Check **tentativeCell** for solid tiles*
For each layer in the map
*Check the tile at the position of **tentativeCell** for solidity*
If tile is solid
Stop movement
Slow down player's animation for cosmetic effect
*Else if no solid tiles in **tentativeCell***
*Check for players with same position as **tentativeCell***
bumpIntoPlayer() //(handles the various collisions between players)
*Else if no one has the same position as **tentativeCell***
*Allow the player to walk to **tentativeCell***

4.4. Networking

The key to the multiplayer functionality is networking, without which would warrant users playing on the same device. While such a function would be possible, part of the gameplay lies in having to find the other players. If all players played on the same device, the location of each player would not be hidden from anyone. Thus, networking was used to make this game possible. This game implements Google Play Services for its real-time game networking (Google Play Services, 2012). The infrastructure behind this networking is called Peer-to-peer or P2P. P2P works by connecting each player to every other player in their own network. Within this network, players can communicate with each other directly without the need for a central server or stable host (Schollmeier, 2002).

This is important because mobile users are just that, mobile. If a user is playing a game, there is no guarantee that they are focused and dedicated to follow through to end. If a client-host structure were used, and the host suddenly decided to quit, all the other players would be disconnected. Depending on the frequency of this happening, this could anger and frustrate more serious players. P2P, on the other hand, allows any player to drop out of a game at any time without disconnecting any other players.

Excluding the player's location, there is a small, finite number of messages that the player can send. The information in these messages can vary from sending a step, changing direction, camouflaging, checking hideable objects, and so on. The simplified process for this is as follows:

```
SendMessageToEveryone(message):
For each player in the list of players
    If the player is not the current player
            Send a reliable message to the player along with the current roomID
```

The messages are constant bytes that are assigned to different actions. For example,

take_step_up *= {7,1}*
take_step_down *= {7,2}*
change_direction_up *= {6,0}*

While these numbers have no inherent value and can be chosen at random, once chosen, they must remain constant in order for all players to receive and interpret the message correctly.

5. Limitations

No game is without limitations, including this one. This does not mean, however, that some limitations cannot be managed. This game experiences some limitations with lag, certain Wi-Fi networks, and lacks the type of scoring models seen in many of today's mobile games.

5.1. Lag and latency

While lag is never good, some multiplayer games can get away with having lag. For instance, if a game is turn-based only, lag is easy to miss. This is because one player is not exactly sure when the other player has made a move. Therefore, it is hard to identify what causes any delay that may occur when waiting for an opponent's turn to complete. Real-time games, on the other hand, are not so lucky.

The whole concept behind real-time games is that they are non-stop and occurring in the present. That means as soon as a player makes a move locally, they expect the game to respond instantly, in other words, low latency. To make games fair, the same actions should be occurring simultaneously across players of the same game. Now, unfortunately, data transmission across a network currently takes more time than input to a local machine. Good real-time multiplayer games seek to make up for this lag and/or high latency. To compensate, some games try to predict the results of users' input. This game takes a different approach.

In order for a player's character not to fall behind in their peers games while moving, their character is given a faster speed in the remote games. The reason being, if it takes x milliseconds to transmit one step (movement across 1 tile), the remote character would be (x * the number of steps) milliseconds behind, especially since a player must complete a step before starting another one. This will quickly result in the games falling out of synchronization. However, since the remote characters move a slightly more

quickly, they will complete their steps sooner. So, even though they experience a slight delay with input, they will complete the step sooner, allowing them to be ready to start the next step upon receiving it. The downside to this method is that it could result in the remote characters' movement looking slightly choppy.

5.2. Wi-Fi Networks

While public Wi-Fi networks are quite ubiquitous these days, many of them are very restrictive in terms of what can be accessed with them. This is common practice in the workplace, libraries, and universities. One of the common restrictions is strict port access. This can be an issue for any game that uses Google Play Services for multiplayer as it requires access to ports TCP and UDP 5228. If these ports are blocked, the game will not be able to connect online. Perhaps, with future improvements in security, this will no longer be an issue.

5.3. Score

The original Freeze Tag Frenzy did not feature a scoring system at all. Instead, there was just a winning team for each round. However, that was quite a long time ago. These days, mobile users are constantly connected to the internet and to their friends via sites like social networking services. This makes it very easy for people to share their high scores among their friends, and the competitiveness of beating friends' scores could motivate someone that much more to play the game. Mainly due to time constraints, this sort of scoring has not been implemented in this iteration of the game, but may be included in the future.

6. Conclusion

Freeze Tag Frenzy was created for the entertainment of others, but also to explore the process of creating a real-time multiplayer game. Playing the game requires the explanation of a few rules and other dynamics, but there is hardly a learning curve. The game makes use of a handful of common game features, namely tiled maps, animation, collision detection, and—for multiplayer—networking. However, the game has its limitations with lag and high latency, restrictive Wi-Fi networks, and lack of an explicit scoring model. These limitations will be further explored before the game is available for download on the Google Play Store later in 2014. Even in its current state, the game can be quite entertaining.

7. References

Baker, C. (2010, June 2). *Nimrod, the World's First Gaming Computer*. Retrieved from Wired: http://www.wired.com/2010/06/replay/

Cerf, V. G., & Kahn, R. E. (1974). A Protocol for Packet Network Intercommunication. *IEEE Trans on Comms*.

Google Play Services. (2012, September 26). Retrieved from https://plus.google.com/+AndroidDevelopers/posts/J1A5hc1ZnS1

Lindeijer, T. (n.d.). *Tiled Map Editor*. Retrieved from Tiled Map Editor: http://www.mapeditor.org/

Mejias, L. A. (2003, August). *Freeze Tag Frenzy by Mr Tunahead*. Retrieved from BYOND: http://www.byond.com/games/MrTunahead/FreezeTagFrenzy

Schollmeier, R. (2002). A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. *Proceedings of the First International Conference on Peer-to-Peer Computing*.

Stewart, B. (2000, January 7). *TCP/IP Internet Protocol*. Retrieved from Living Internet: http://www.livinginternet.com/i/ii_tcpip.htm

Zechner, M. (n.d.). *LibGDX*. Retrieved from Badlogic Games: http://libgdx.badlogicgames.com/