Assassin's Decree

Story

In 2016. A mysterious underground assassin organization can't find enough killers to accomplish their missions. In order to grow their assassination business. This organization decides to make their assassin-on-demand service. This is a high pay high risk service. The assassin must finish the mission in 2 minutes to get the reward or they will be hunted.

Killing or being killed? Run or die? Game begins...

Features

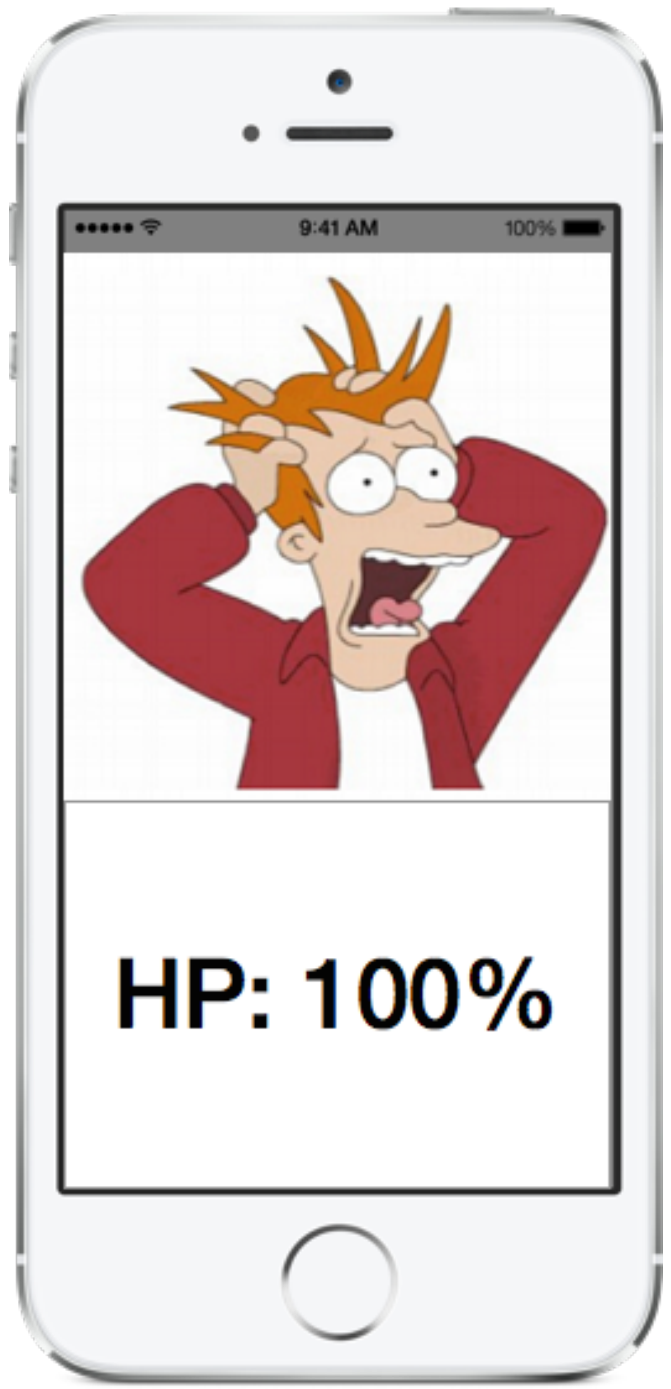
- Player will be randomly chosen as victim or assassin
- Assassin can track the locations of victims on the map
- Assassin will get a notification if a victim is nearby
- Assassin can attack the victim with different weapons
- Weapon has a range and the damage is depend on the distance to the victim
- Assassin can reload the weapon by pairing with device over BLE
- Assassin will be dead if no kill in 120 seconds

Features

- Victim can escape the assassination by running out of the range of the weapons.
- Victim can kill the assassin by not being killed in 120 seconds
- The locations of victims must be updated to the assassin in realtime

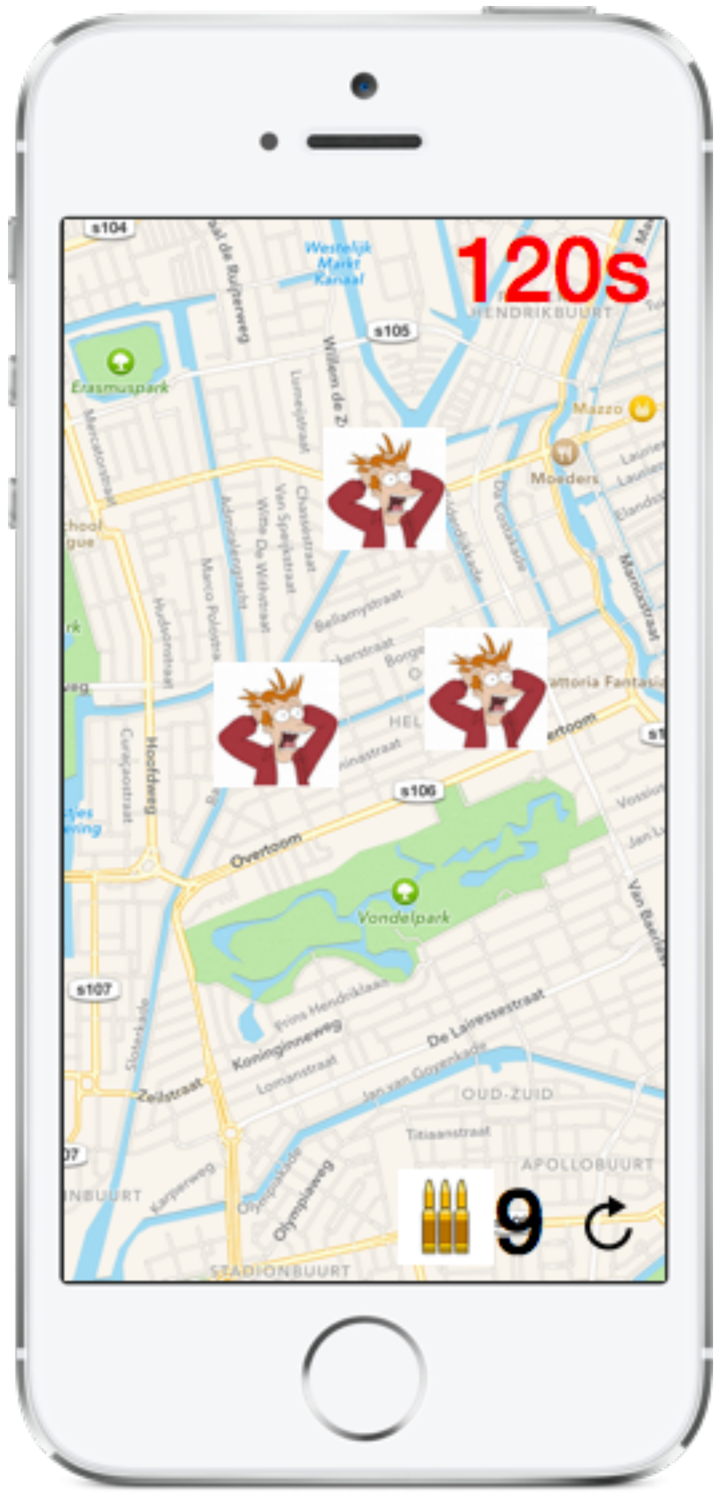
UI Victim

- Shows current hit points
- Picture indicate the status:
  alive or dead
- If Victim is dead, shows a button to restart the game.



UI Assassin

- Track locations of victims in realtime
- Current weapon load
- Time left to kill or die
- Reload the weapon button
    - Discovered BLE devices to connect
- Click a victim -> weapon list:
    - If victim in the range of weapon, weapon enabled to attack



Solutions for region detecting of assassins

1. Geofencing in victim side:
    - high device power consuming
    - limitation of tracking location numbers
2. Update locations of victims and assassins and store them in mongodb on the server. Mapreduce the locations for assassins and notify if needed.
    - server side load
    - too much implementation for this challenge
3. Update victim location to the assassin side and calculate the distance.
    - reduce server load since the client device has enough power to perform this operation.
    - since assassin tracks the location of victims, so the implementation of this solution will be optimal.

We choose solution 3.

Technical Requirements

- MVVM architecture
- Reactive functional programming (RxSwift)

In order to have more fun with this challenge, I choose to use MVVM + RxSwift to keep the code compostable.

Technical Requirements

- Node.js + socket.io backend server for realtime messaging. Redis datastore for storing the players.
- Victim App Client update their location to the server and then emit to the Assassin Client
- Assassin App Client receiving the location changes.
- Assassin App Client update the victim damage to the server and then emit to the client.

Tasks

- Node.js messaging backend
  - on("updateLocation") update location
  - on("damage") update new hit points to victims
- Victim Model + View Model + Victim View & View Controller
- Assassin Model + View Model + Assassin View & View Controller

Time Plan

- developer 1 to setup & implement the backend. (1 Hour)
- developer 2 to develop the models of victim & assassin, and their view models. (1 Hour)
- developer 1 then use the victim model & view model from developer 2 to finish the ViewController & View for victim (2 Hours)
- developer 2 do the rest. (2 Hours)