CS408 Project Document Revised (KAIST 2017 Spring)

Due: 05/10 23:59

Team Project Name: Tom and Jerry game (Transmitter Hunting, Virtual Foxhunt game)

Project Summary: Implementation of the virtual gears and the system for the sport called Foxhunt game, where 'hounds' hunt down 'a fox' by observing the virtual signal calculated from the signal simulation using physical GPS and Map data.

Game Reference: https://en.wikipedia.org/wiki/Transmitter_hunting

GitHub: https://github.com/SuminHan/Tom-and-Jerry-Simulation

Team Members:

20090484 Seung-hwan Song[SH], 20091238 Jung-woo Yang[JW], 20130690 Sumin Han[SM]

1. Project Overview

A. Motivation

Our team wanted to utilize various technologies that we learned throughout our courses or experienced personally. Some of these technological experiences include the Unity 3D game engine, TCP-IP bidirectional communication, and MPI parallel programming. Also, we discussed augmented reality (AR) games such as 'Pokemon GO!', which was one of the most popular games released last year (2016). Therefore, we agreed to make an AR game for mobile platforms which utilizes the various technologies that we have learnt.

One of our members, Seung-hwan, proposed the foxhunt game, which is a sport originated from England. This sport is similar to hide-and-seek. The biggest distinction is the usage of the signal generator and the detectors. Suppose Tom hunts down Jerry. Jerry carries the signal generator while running away. Tom observes the signal shown on the detector while hunting. However, it's very inefficient to carry all these equipment every time we play. It would be much more convenient if we can encapsulate the core features of the game into a mobile device instead.

Furthermore, the simulation of the signal generation and detection process does not look easy. If there are walls or forests, the signal can be disrupted or reflected. Also, the signal gets weakened by the distance. So we need to figure out a reasonable algorithm to calculate signal variation to achieve this simulation. We also need to calculate as many signals as possible to make the simulation more realistic. Plus, we have to reduce the delay for this whole calculation process since we want to make this game run on real-time. So we thought that this project could be challenging enough to satisfy the conditions for our needs.

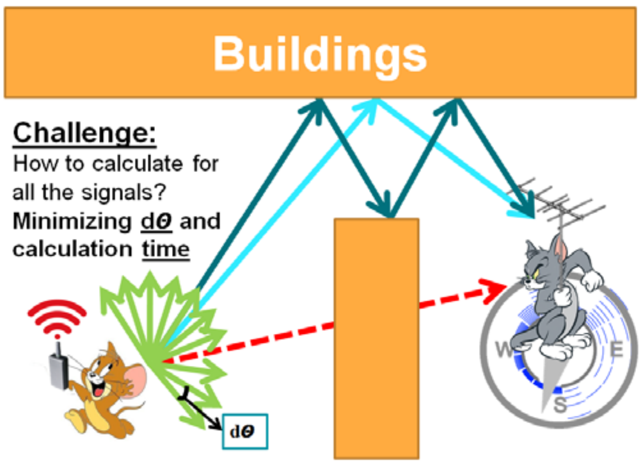

Buildings
Challenge:
How to calculate for all the signals?
Minimizing dθ and calculation time
dθ
W
E
S

Figure 1. Calculating signal reflection.

B. Problem Statement

This game is based on a real sport, so the communication between the client and the server should be on real time, which limits the calculation time for the signal process. As the signal can be affected by surrounding buildings or forests, the calculation should be done based on existing map data. The Open Street Map data (open map database) has a representation of buildings and forests as polygons created with nodes. We will calculate the reflection using these edge data of the polygons. Our plan is to use the queue data structure to implement this algorithm. By using a queue, we can use the Map-and-Reduce technique for calculation, which enables to run multiple cores (CPUs or GPUs) to greatly reduce the time consumption.

In addition, we will operate the server to connect with the clients using bidirectional communication like TCP as TCP is more stable than UDP and more appropriate for the real-time communication. The GPS positions of the players will be sent to the server, and the server will send back the calculated signal information to the players' smartphones. In this process, there could be exceptional cases such as disconnection or unstable connection. We should take care of those possible faults by various testing methods.

Finally, we will provide graphical user interface (GUI) for each player. If the player takes the role of a fox (Jerry), we will show his state (running, caught) and the state of signal generation on the map. If the player takes the role of a hound (Tom), we will show the signal detection graph from different points of view (on the compass, on the map, in a first-person view). To start us off, we will use the Unity 3D map demo code created by a GitHub user, 'Reinterpretcat'.

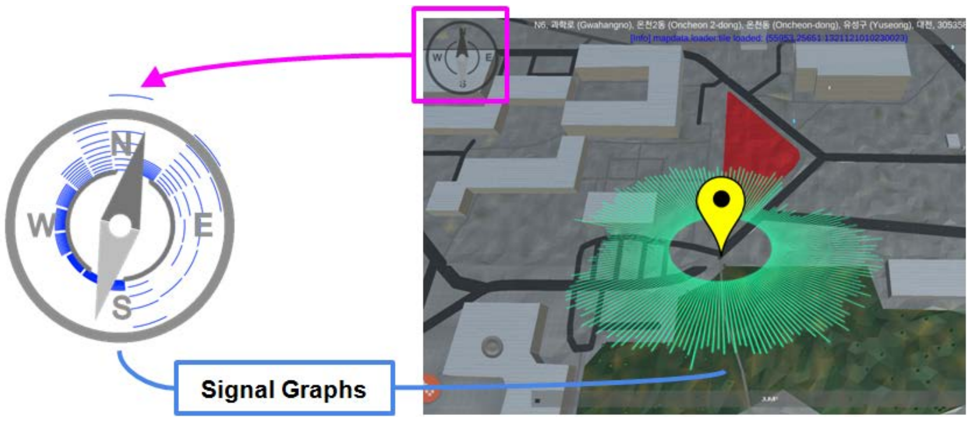

N6, 과학로 (Gwahangno), 온천2동 (Oncheon 2-dong), 온천동 (Oncheon-dong), 유성구 (Yuseong), 대전, 30535
[info] mapdata.loader:tile loaded: (55053,25651 13211210102300023)
W
E
N
W
E
S
Signal Graphs

Figure 2. Possible screen for signal detection.

C. Goal Statement

These are the list of what we are going to achieve in this project.

1. Signal simulation calculation using parallel computing.
2. TCP real-time connection between the clients (Android) and the server.
3. Graphical User Interface of 3D map based on Unity 3D game engine.
4. Screen for the observer which pinpoints the positions of both fox and hound.

If we finish, we will record a video that we actually play this game outside on the campus as a final achievement.

D. Overall Plan

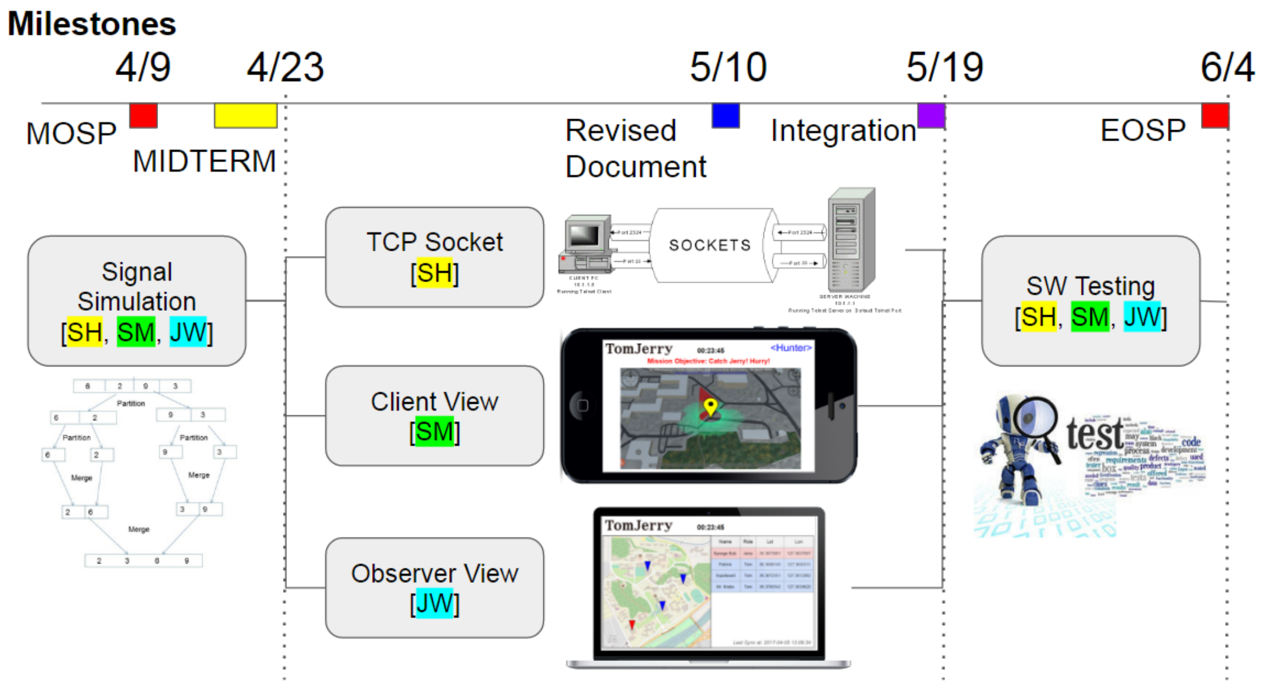
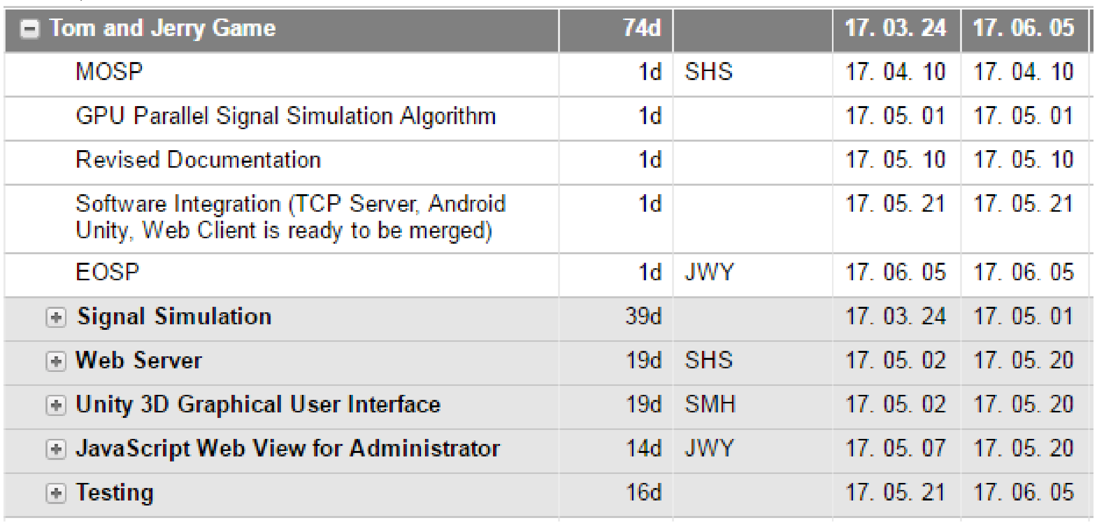


Figure 3. Overall plan with milestones.

We will apply the **incremental model** for our project.

[Basic milestones: **MOSP: 17.04.10 / Revised Documentation: 17.05.10 / EOSP: 17.06.05**]

First, We will spend time to implement the signal simulation until the midterm period. Since the signal simulation is the core function for our project, we need to check its feasibility as soon as possible. **(Milestone: 17.05.01)** Next, we will separate our job to implement TCP socket[SH], Android client view[SM], and Observer View[JW]. Each process may apply the waterfall model. (Milestone: **(Milestone: 17.05.21)** Then we will procure enough time to test and actually play with it. We will go outside on a sunny day and play the actual Foxhunt with our own device. We will also take a video to show others about our game. **(Milestone: 17.06.04)**



Tom and Jerry Game	74d		17. 03. 24	17. 06. 05
MOSP	1d	SHS	17. 04. 10	17. 04. 10
GPU Parallel Signal Simulation Algorithm	1d		17. 05. 01	17. 05. 01
Revised Documentation	1d		17. 05. 10	17. 05. 10
Software Integration (TCP Server, Android Unity, Web Client is ready to be merged)	1d		17. 05. 21	17. 05. 21
EOSP	1d	JWY	17. 06. 05	17. 06. 05
Signal Simulation	39d		17. 03. 24	17. 05. 01
Web Server	19d	SHS	17. 05. 02	17. 05. 20
Unity 3D Graphical User Interface	19d	SMH	17. 05. 02	17. 05. 20
JavaScript Web View for Administrator	14d	JWY	17. 05. 07	17. 05. 20
Testing	16d		17. 05. 21	17. 06. 05

Figure 4. Overall plan with duration (start and end date).

E. Schedule and Task Management Plan

We specified actual tasks that are required to implement each function. (Also, you can see our full Gantt chart in the following URL: https://goo.gl/DkTAiv)

MileStones:

- MOSP **(17.04.10)**
- GPU Parallel Signal Simulation Algorithm **(17.05.01)**
- Revised Documentation **(17.05.10)**
- Software Integration (TCP Server, Android Unity, Web Client) **(17.05.21)**
- EOSP **(17.06.05)**

a. Signal Simulation (17. 03. 24 ~ 17. 05. 01)

1. OpenCL(NVIDIA CUDA) installation
2. OSM XML data crawling and refinement
3. Sequential Reflection algorithm
4. GPU Parallel Implementation

b. Web Server (17. 05. 02 ~ 17. 05. 20)

1. C/C++ on Win7 server environment set up
2. TCP socket implementation
3. Android real-time connection testing
4. Real-time calculation and communication
5. Login and DB setup

c. Unity 3D Graphical User Interface (17. 05. 02 ~ 17. 05. 20)

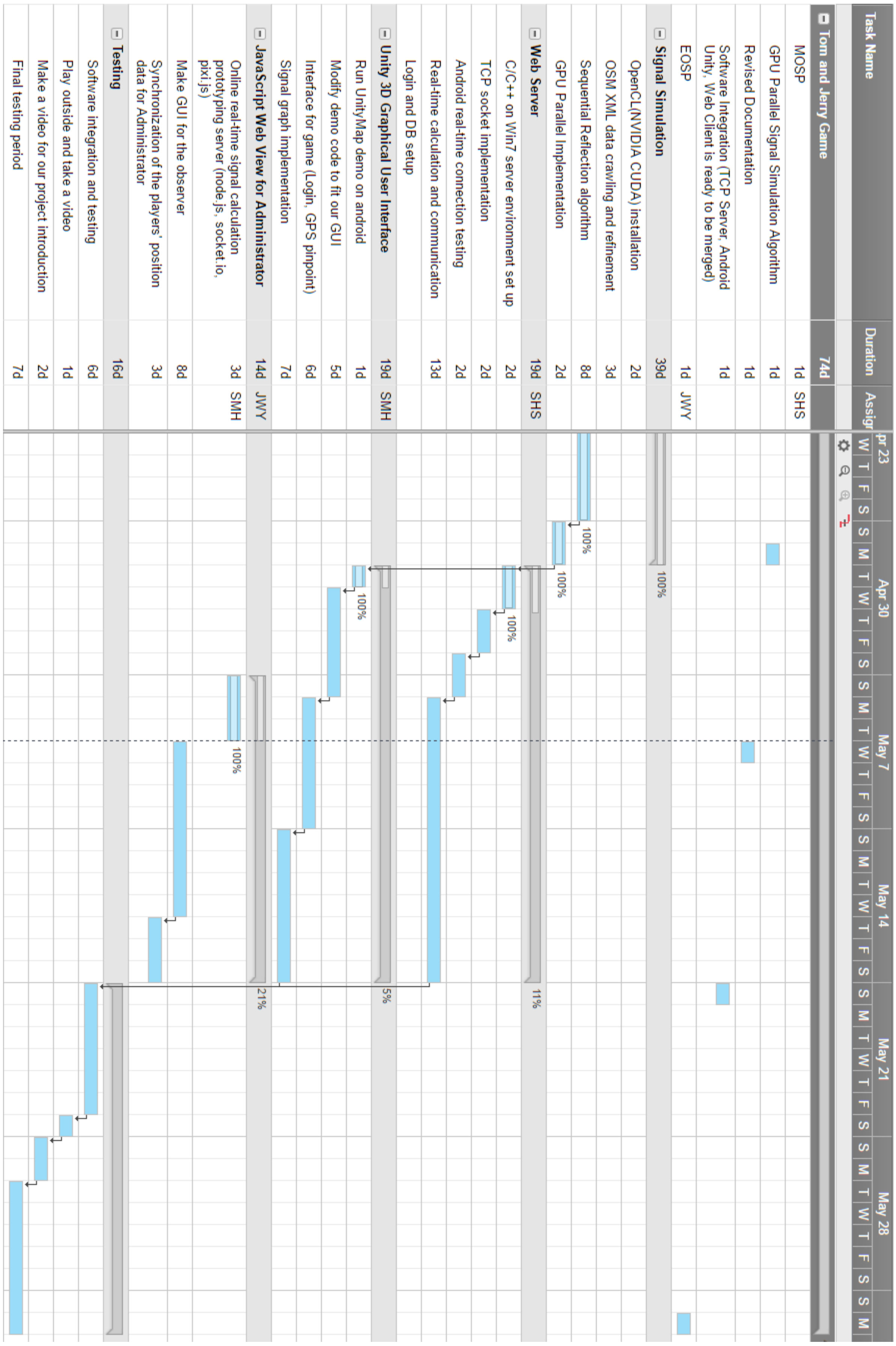
1. Run UnityMap demo on android
2. Modify demo code to fit our GUI
3. Interface for game (Login, GPS pinpoint)
4. Signal graph implementation

d. JavaScript Web View for Observer (17. 05. 07 ~ 17. 05. 20)

1. Online real-time signal calculation prototyping server (node.js, socket.io, pixi.js)
2. Make GUI for the observer
3. Synchronization of the players' position data for Administrator

e. Testing (17. 05. 21 ~ 17. 06. 05)

1. Software integration and testing
2. Play outside and take a video
3. Make a video for our project introduction
4. Final testing period



Task Name	Duration	Assign
Tom and Jerry Game	**74d**	
MOSP	1d	SHS
GPU Parallel Signal Simulation Algorithm	1d	
Revised Documentation	1d	
Software Integration (TCP Server, Android Unity, Web Client is ready to be merged)	1d	
EOSP	1d	JWY
Signal Simulation	39d	
OpenCL(NVIDIA CUDA) installation	2d	
OSM XML data crawling and refinement	3d	
Sequential Reflection algorithm	8d	
GPU Parallel Implementation	2d	
Web Server	19d	SHS
C/C++ on Win7 server environment set up	2d	
TCP socket implementation	2d	
Android real-time connection testing	2d	
Real-time calculation and communication	13d	
Login and DB setup		
Unity 3D Graphical User Interface	19d	SMH
Run UnityMap demo on android	1d	
Modify demo code to fit our GUI	5d	
Interface for game (Login, GPS pinpoint)	6d	
Signal graph implementation	7d	
JavaScript Web View for Administrator	14d	JWY
Online real-time signal calculation prototyping server (node.js, socket.io, pixi.js)	3d	SMH
Make GUI for the observer	8d	
Synchronization of the players' position data for Administrator	3d	
Testing	16d	
Software integration and testing	6d	
Play outside and take a video	1d	
Make a video for our project introduction	2d	
Final testing period	7d	

Figure 5. Gantt chart.

F. Final Deliverables

The final product of this project will obviously include this documentation, MOSP and EOSP presentations. There will also be a user client, a server program and a demonstration video of our program. The user client is run on a smartphone. It will consist of a login screen and a play interface. The server client (observer) is run on a PC, with a web-based environment. The schedule to produce our final deliverables is specified above, in section E, Schedule and Task Management Plan. Every deliverables including source code and documentation will be published well organized on our GitHub.

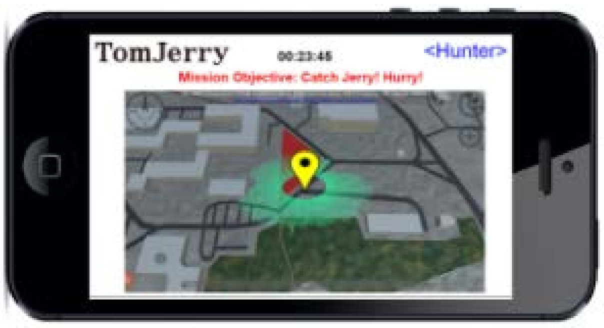
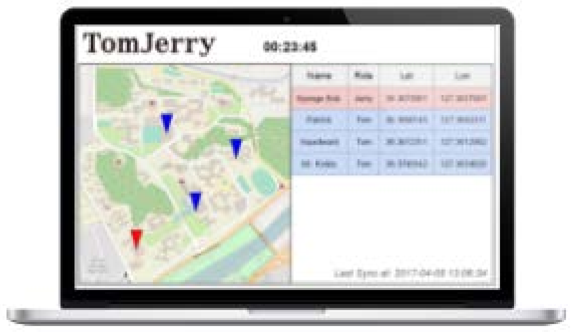
 TomJerry 00:23:45 <Hunter> Mission Objective: Catch Jerry! Hurry!	
Smartphone application for the player.	Web application for the observer.

Figure 6. Expected screenshots for runner and observer during the game.

G. Internal Roles

First, SH, JW, SM will work with the signal simulation algorithm together. However, as SM has experience of working with a parallel algorithm, he may bring related materials to learn. We all have to learn about related mathematical theories to find the most effective way to detect signal reflection and detection. In this process, we expect to deal with matrix calculation of vectors (the signal vector and the edges of the buildings which are polygons).

After we successfully solve this, we will separate our workload equally. SH will take care of TCP socket implementation since he took Computer Networks course. SM will implement Unity 3D GUI since he had made a 3D game using Unity game engine in another class. JW will work on JavaScript web viewer for observer to manage and control the users.

This section should provide a table that shows what roles are required for the project and which member would take the role.

Name	E-mail	Role
Seung-hwan Song	sik2603@kaist.ac.kr	Coder (Java Server, TCP Socket)
Jeong-woo Yang	jwy1991@kaist.ac.kr	Coder (HTML, JS)
Su-min Han	hsm6911@kaist.ac.kr	Coder (Unity, Parallel), Project Manager

Chart 1. Internal roles.

2. Software Requirements Specification

A. Use Case Diagram

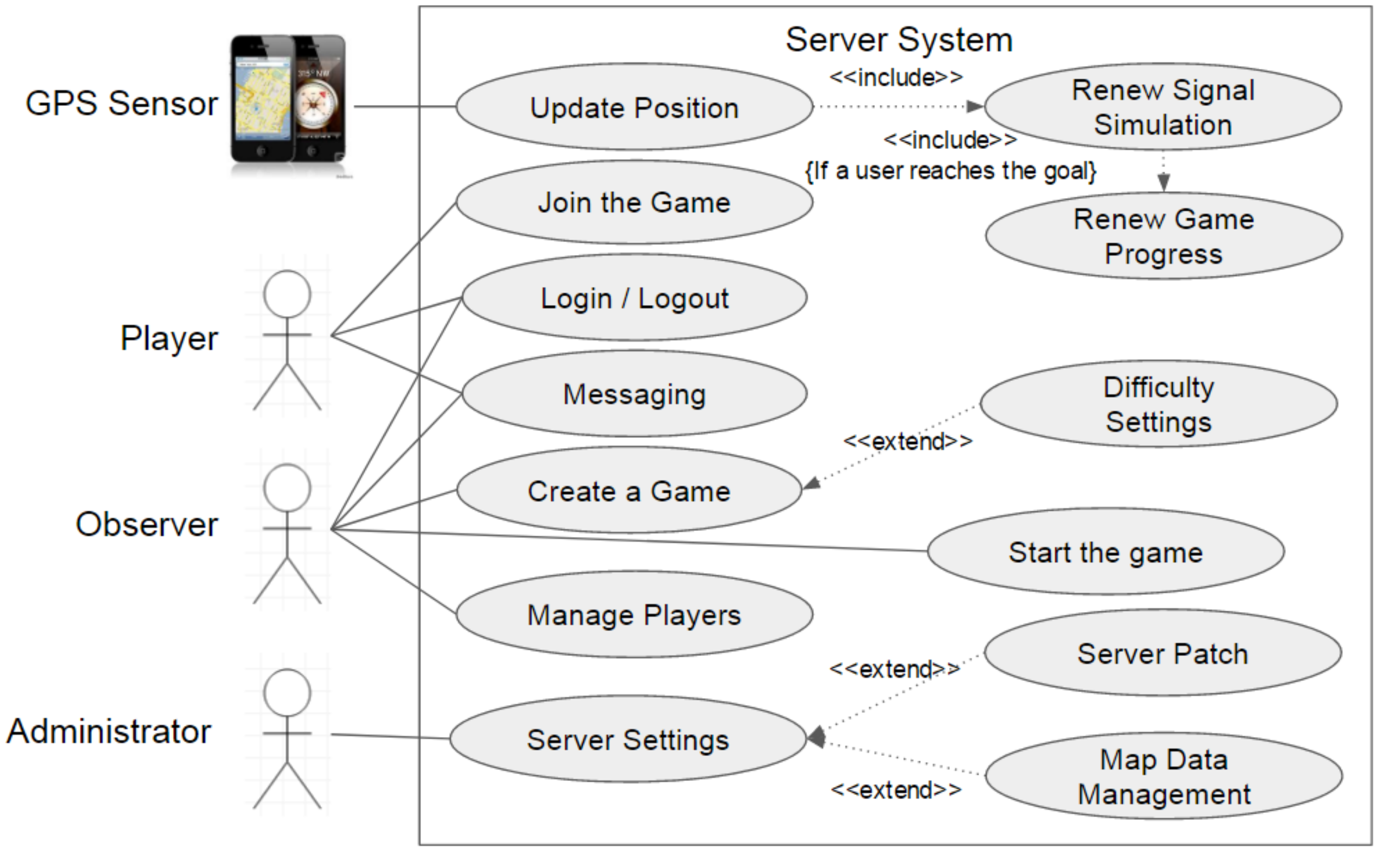


Figure 7. Use Case Diagram.

B. Use Case Description

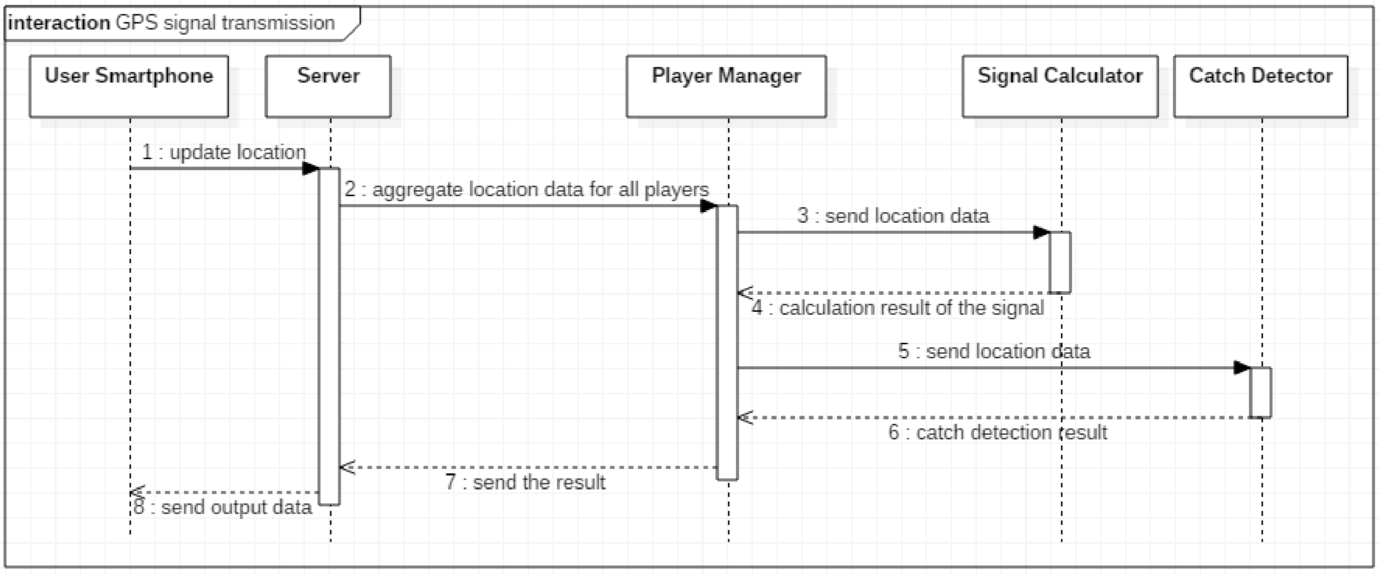


Figure 8. Sequence diagram of GPS signal transmission.

This is the sequence diagram for our key use case, which is the process of sending the information to the server and receive the calculated data. You may refer the terminologies such as “Player Manager”, “Signal Calculator”, “Catch Detector” from SW architecture figure (Figure 12).

Use case	B.1. Update Position
Primary actor	GPS sensor of each smartphone having player client
Goal in context	Send the location of a player to continue radio signal simulation.
Preconditions	Wireless internet connection and GPS sensor must be available. A game must be on process.
Trigger	The GPS sensor installed in a smartphone periodically measures the location. If the sensor works without problem, it triggers this use case. This usecase also includes the catching action of the Hunter.
Scenario	1. The mobile smartphone automatically uploads its GPS position data to the server in every 3 seconds. 2. The server reacts to the data and gathers it for all the players, and be ready to run the signal calculation algorithm. 3. The signal calculation module runs using inputted position data using parallel algorithm utilizing the GPU. 4. The signal strength data for each player is outputed. 5. The catch detector calculates the distance between the players and checks whether the hunter caught the runner. (This process is controversy because of the GPS error so we're looking for exact distance calculation methods like sending real signal or taking picture by the hunter). 5.a. If the runner was caught, goto 8). 6. The server sends the signal strength data to the user smartphone. 7. The Unity uses that data to draw the signal graph on the map. 8. If the game is over, show the winner on the screen, and ends the game.

Use case	B.2. Login
Refer UI	E.a.1, E.b.1
Primary actor	Game player, Observer
Goal in context	Establish an Internet connection with the server to login.
Preconditions	Wireless internet connection and GPS sensor must be available. The client application must be installed before. Player must be logged off
Trigger	A player executes the client application.
Scenario	1. The game client is turned on. 2. The user enters his ID and password. 3. The game server receives the player's information and allows further operations if valid.

	4. Server responds to the client. If logged in successfully, the user is ready to use other functionalities.

Use case	B.3. Logout
Primary actor	Game player, Observer
Goal in context	Finish connection with the server.
Preconditions	The client application must be turned on.
Trigger	The player selects the logout menu.
Scenario	1. The client sends signal to server for logout. 2. The server rejects the player from any game he/she joined. 3. Deactivate player status on game he/she joined.

Use case	B.4. Messaging
Primary actor	Game player, Observer
Goal in context	Communicate with other players
Preconditions	Wireless internet connection must be available. The user must be logged on
Trigger	A player selects the chat menu.
Scenario	1. A player enters a message, and sends it to server. 2. The server checks the receiver options for the message 3. Transmit the message to allowed players.

Use case	B.5. Join the Game
Refer UI	E.b.2
Primary actor	Player
Secondary actor	Other game players
Goal in context	Join the game
Preconditions	Wireless internet connection and GPS sensor must be available. The user must be logged on
Trigger	Player clicks to join the game.

Scenario	1. The player chooses the game he wants to join. 2. If there’s a password, he should type it correctly to enter(or there will be error message shown). 3. The player enters the room and wait until the observer starts the game.

Use case	B.6. Create a Game
Refer UI	E.a.2
Primary actor	Observer
Goal in context	Prepare a game to play with other players.
Preconditions	Wireless internet connection and GPS sensor must be available. The user must be logged on
Trigger	A player selects the ‘Create a game’ menu.
Scenario	1. The observer creates a group for a new game. 1.a. He can set a password if he want. 2. Selects the location where game is held, and options. Those information is sent to the server. 3. The observer waits until the server matches suitable players. 4. The observer starts the game if all players are ready. Send the initialization signal to the host. 5. The server decides the goal(location of beacons), and starts the simulation. 6. The server starts to send game status to the players.

Use case	B.7. Starts the Game
Refer UI	E.b.3
Primary actor	Observer
Goal in context	Starts the game by clicking on the button.
Preconditions	All players are ready.
Trigger	The observer starts the button.
Scenario	1. Observer clicks on the start button when evereybody is ready.

Use case	B.8. Server Settings
Primary actor	Administrator

Goal in context	Manage the overall server settings, modify map data.
Preconditions	The user must be logged on.
Trigger	Observer executes the management program
Scenario	1. Administrator updates the system or source code.

Use case	B.9. Manage the players
Refer UI	E.b.4
Primary actor	Observer
Goal in context	Modify the player status. (May be useful for debugging)
Preconditions	The user must be logged on.
Trigger	Observer executes the management program
Scenario	1. Watch for the player’s action and manage them.

C. Functional Requirements

1. Radio signal simulation

Our software(game) revolves around trying to catch a simulated radio signal. Therefore, it is the essential functional requirement for our software. As we planned to operate this program with our campus as the setting, we would have to consider geographical obstacles such as trees and buildings. This will be done through an algorithm involving mathematical and physical calculations. We will test for the combination of the possible positions of the players such as between the building, across the woods, or in the open space. Whether the signal is calculated reasonably good or not can not be judged by the computer, so this test should involve human judgement. The KAIST has 117 buildings according to the OSM data, so we need to check at least 117*117 combinations theoretically, but we may try with rather feasible player’s location, so this number would be reduced around 300 test cases (reject the case if the players are too far away). Also we should be careful not to have the spark signal. If a signal has unreasonablly high intensity, we will dismiss that value. Finally, the total signal calculation process should be done in maximum 2 seconds since the synchronization is done for 3 seconds (1 extra second for data send, and receive time, and file input/output etc.) Our goal is to provide a simulation of around 50,000+ signals for each player considering the hardware performance (which could result in delays in processing or displaying).

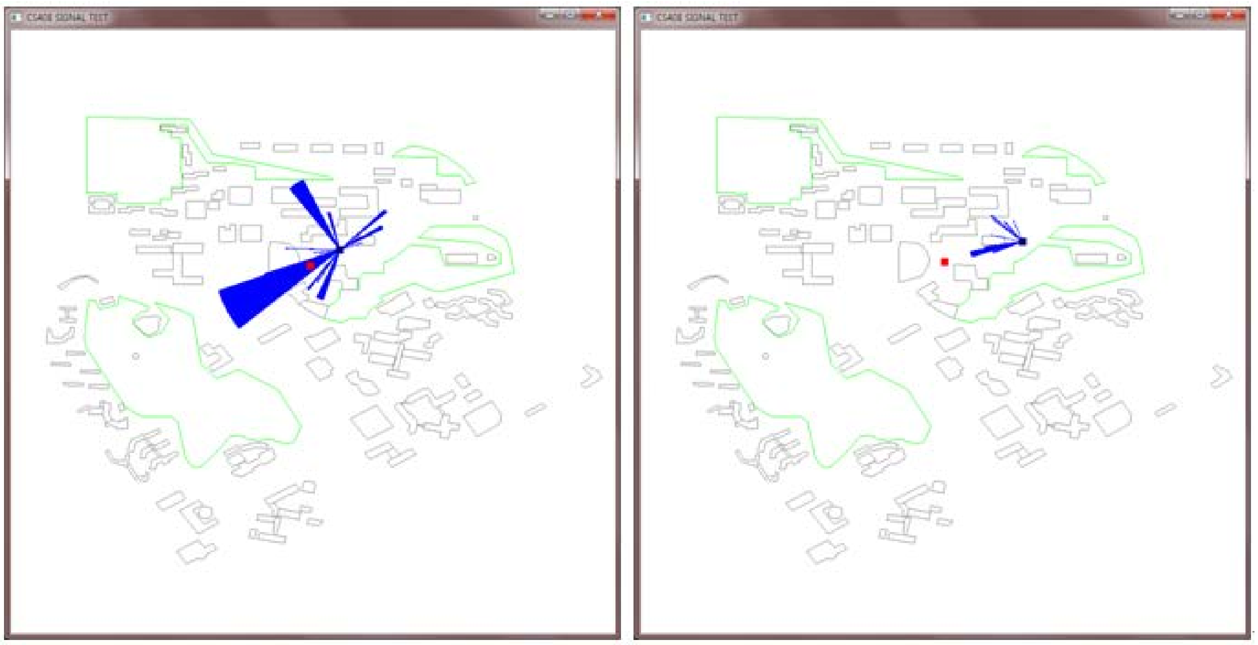


Figure 9. Only human can judge whether the calculation is reasonable or not. So we should check for the combination of possible positions of the players.

2. **Catch Detection**

   As the goal of this game is to catch the runner, we need a precise method to detect whether the player actually caught the runner. In fact, the GPS has error for 5-10 meter, we will still use the GPS distance for the catch detection. Later if we have more time, we may apply other methods like taking picture and detect the unique code of the runner (e.g. the fox is running wearing red shirt so the observer admits it in analogue style or using image recognition tools if available) or using other mobile sensors if it's possible.

3. **Server-client TCP communication**

   Our game involves mobile devices (smartphones) as the signal detectors. Therefore, we need consistent intercommunication between the game server (which has the information about the source) and the users' devices which are used to track down the signal source. The TCP communication latency for our project is every 3 seconds to refresh the signal of the target. So the calculation should be done in 2 seconds as it's mentioned above. Also the CUDA program, which is the NVIDIA GPU program, should be written in C language, the TCP server should be written in C language too. (Otherwise, we can choose another language for the server program, but this may require modulization of the signal calculation program and this requires more time for communication between the processes. However, in the current development, we modulized signal calculation unit to be used in the prototyping of the server using Node.js server.)

4. **Graphical User Interface (GUI)**

   Our game needs to show the current user's location because we are making a game based on the real physical space. Since Tom has to track Jerry's signal, Tom's smartphone should show the signal graph on the map. A simple and effective GUI is required for better user experience. The overall layout of the GUI is shown below in section E, which deals with UX/UI. We would try to keep the layout as simple as possible, because a game would only need to display minimal information needed for the player to play. We benchmarked Starcraft 1 battle.net UI, because it is simple and it has been used for almost 20 years without any inconvenience.

5. **Login system**

   The original signal hunt game is a social game, so we would like to facilitate this aspect of the original game through registration of users. With dedicated IDs, users would be able to find their friends or meet new people to play together. Also, this is done to prevent confusion caused by users using multiple devices. This also makes the game more enjoyable, as registering the users will prevent them from cheating, as cheaters could get banned or have other measures taken against them. The login system should, like all login systems, only authorize the login of authorized (identified) users. Also, it should discriminate between the administrator and normal users. We will use SQLite for the DataBase technology for its simplicity.

6. **Game Room System**

   In order to start the game, the observer should create a game room. Then each player will join the room using their smartphone. There can be game password if the observer wants to make a private game. The observer can also ban player before start from game room. Refer to the section E: User Interface Requirements.

D. Software Quality Attributes

1. **Precision**

   This application needs to provide the users the accurate location of the target while still maintaining the fun. If we keep the error tolerance too high, then there will be no fun in tracking down the target, as the hunt will become too easy to accomplish. If we keep the tolerance too low, then it would be too stressful for the players as they would have to unfeasible for us as the error rate of the GPS is around 5 meters. We plan to keep our error radius as 5 meters, as it is the precision of the GPS signals, and it is a reasonable range that the players can “see” the target.

2. **Optimization**

   Our program is designed to run on a mobile environment. So, battery drain is an important issue for our app. The original game could go for more than 2 hours. Our players could be using their smartphones for other uses such as messaging or phone calls during the game session. Therefore, we must ensure that the program does not consume too much battery. The standard for optimization is around 10% battery drain per hour.

3. **Portability**

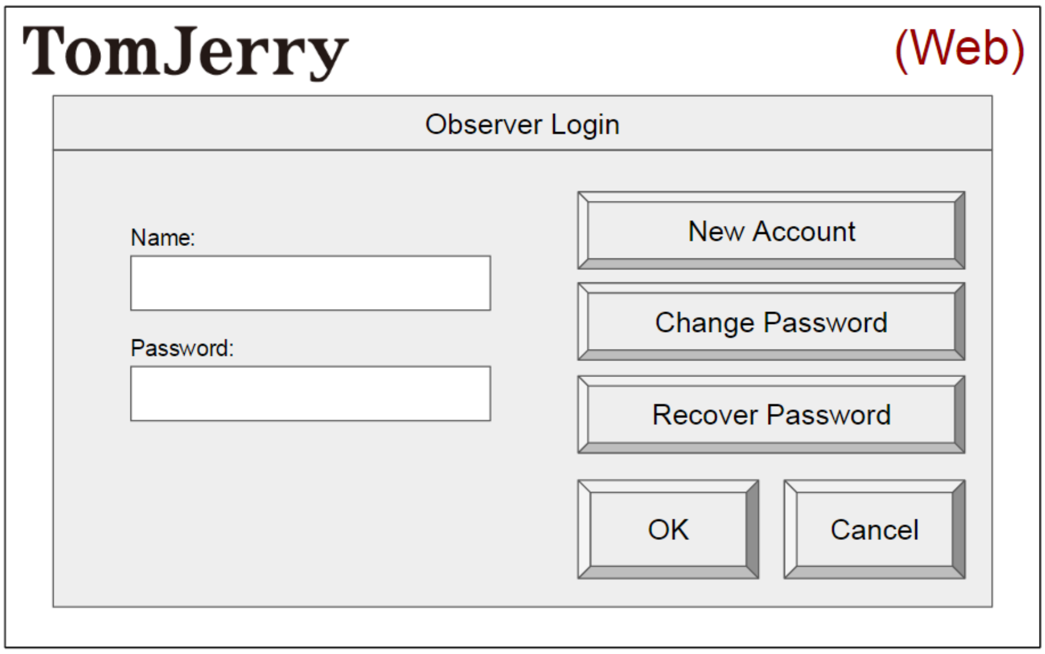
   As stated above, the program runs on a mobile environment. Around three-quarters of smartphone users in South Korea are android users. As this program is set for domestic purposes, we plan to make it run on an Android operating system for mobile devices.

E. User Interface Requirements (UI/UX)

We need some interface for user to easily create game and join the game. We benchmarked Starcraft I Battle.net user interface for our UI, because they shared some similar requirements.

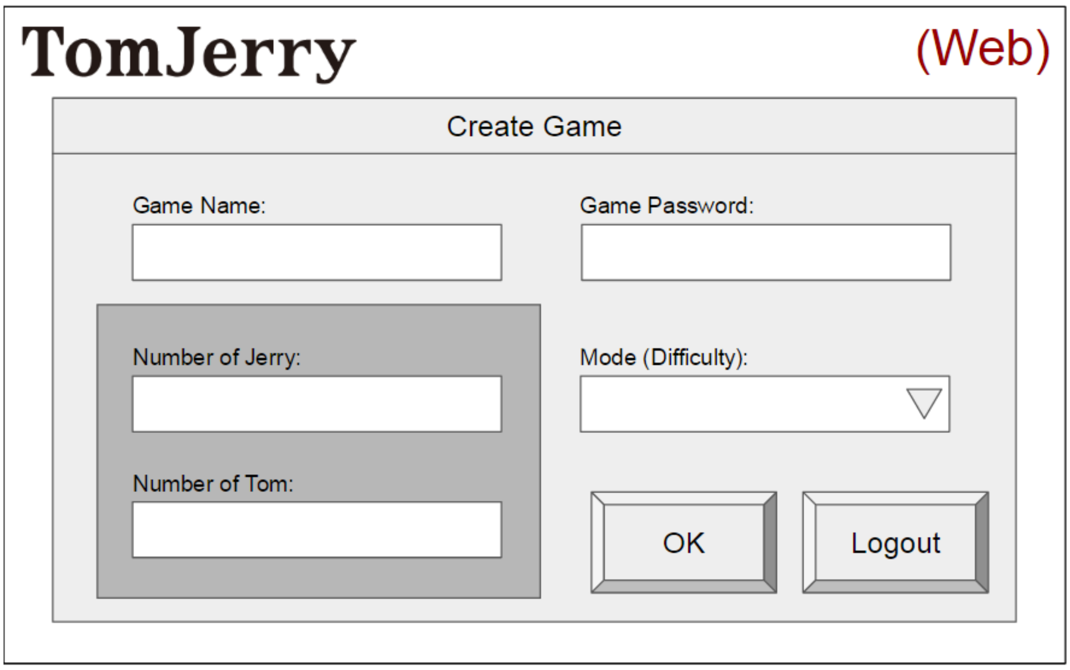
a. Web Observer View

1) Login



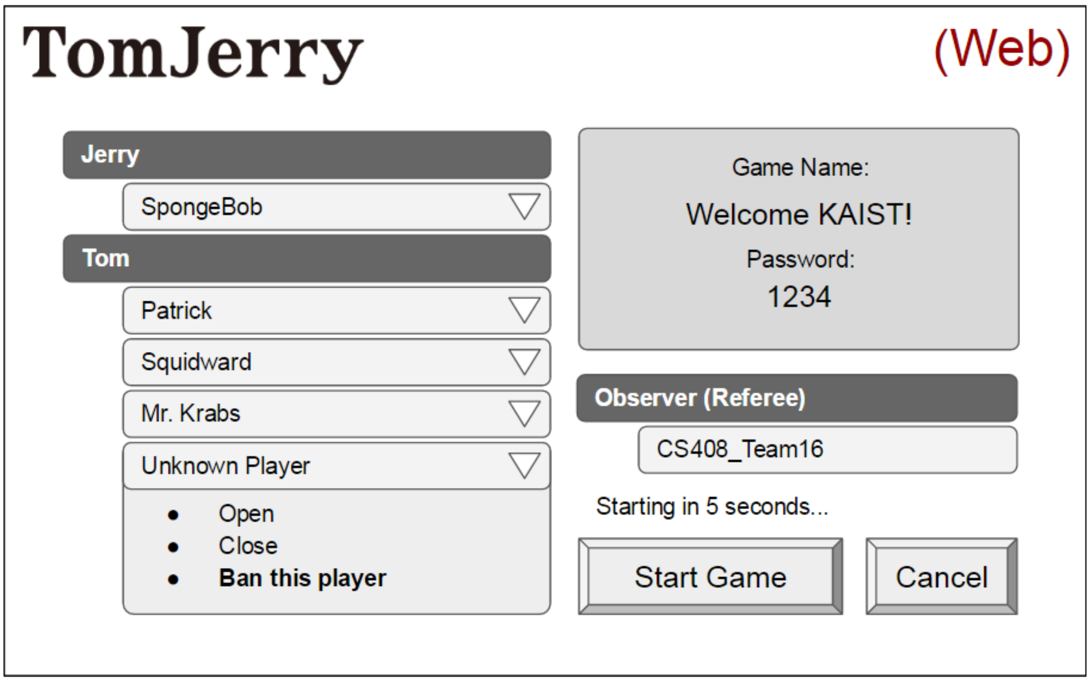
➢ This is a login page. You can create new account or change and recover password. Each button will link to typical process for each purpose.

2) Create the Game



➢ After log in, observer can directly create game, setting game name, game password, number of Jerry and Tom (foxes and hounds).

3) The Game Room



TomJerry (Web)

Jerry
- SpongeBob

Tom
- Patrick
- Squidward
- Mr. Krabs
- Unknown Player
    - Open
    - Close
    - **Ban this player**

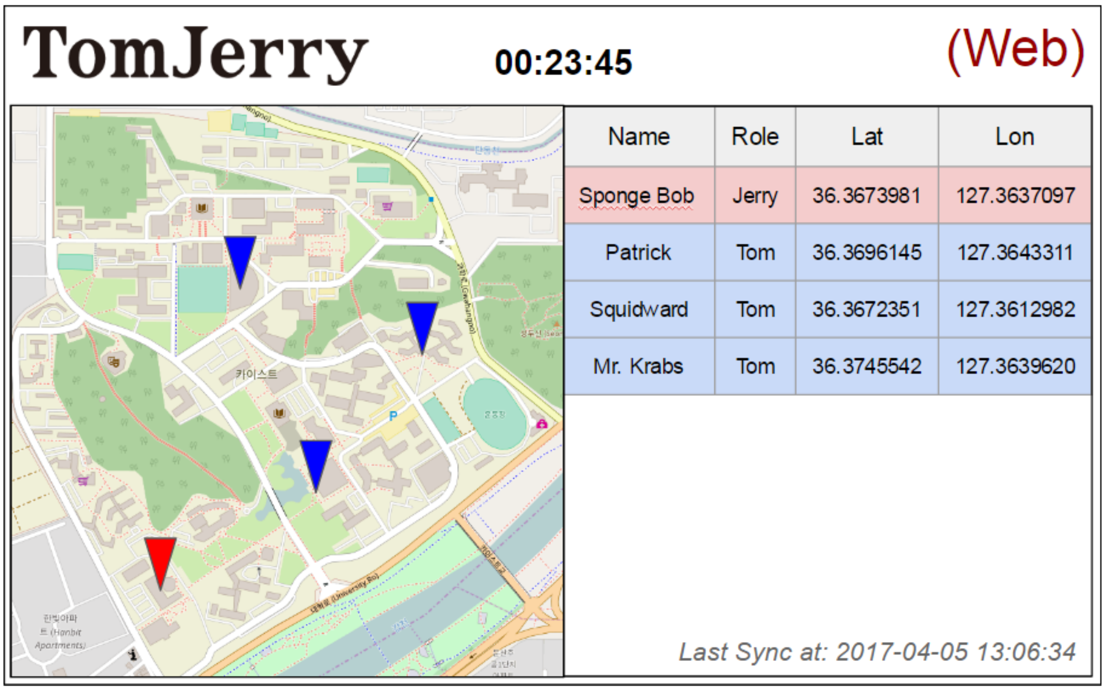
Game Name:
Welcome KAIST!
Password:
1234

Observer (Referee)
- CS408_Team16

Starting in 5 seconds...

➢ You can wait for other player. You can close or open or ban the player by click and drag down the triangle button on the right side of the nick name.

4) During the Game



TomJerry 00:23:45 (Web)

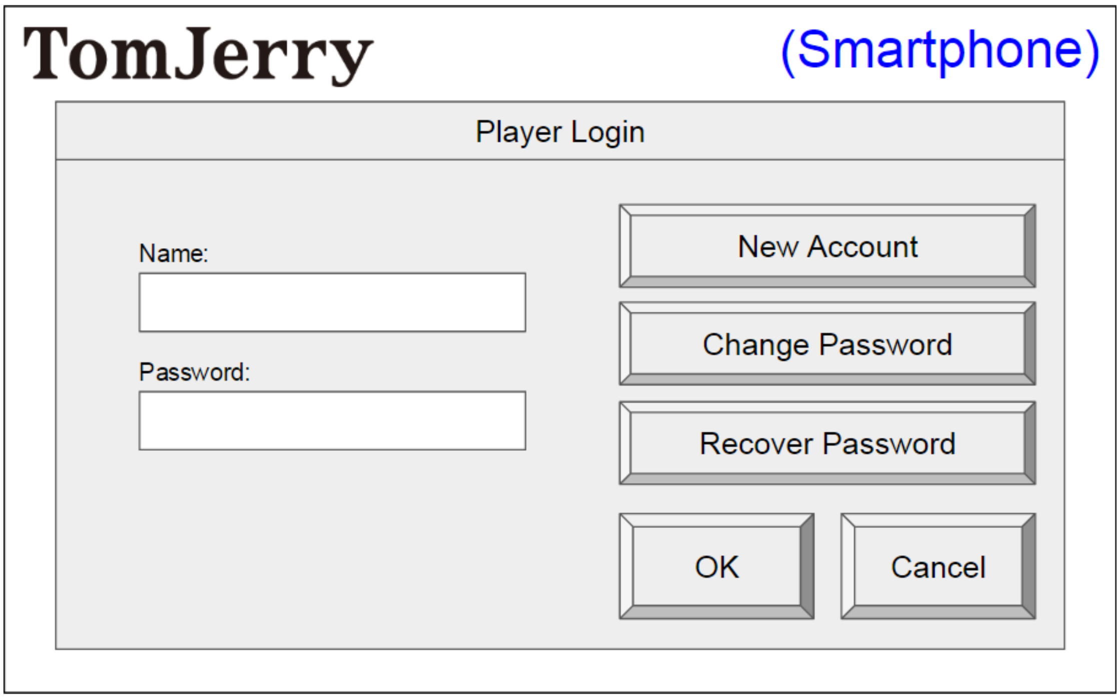
Name	Role	Lat	Lon
Sponge Bob	Jerry	36.3673981	127.3637097
Patrick	Tom	36.3696145	127.3643311
Squidward	Tom	36.3672351	127.3612982
Mr. Krabs	Tom	36.3745542	127.3639620

Last Sync at: 2017-04-05 13:06:34

➢ After game starts, you can see the position of each player on the map.

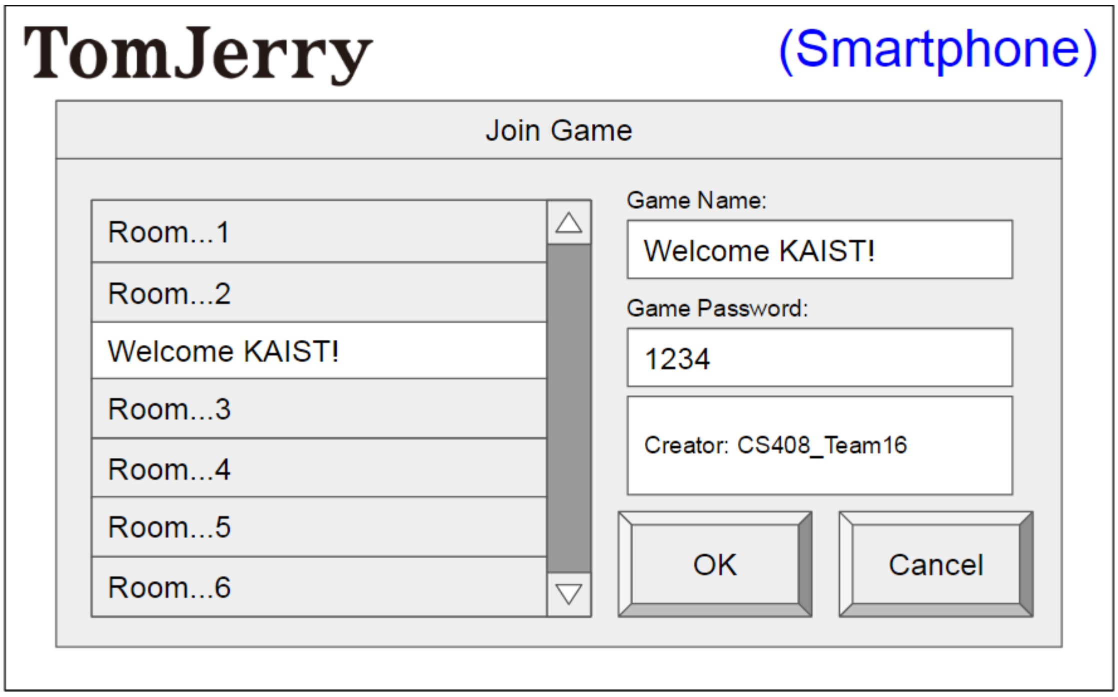
b. Client Smartphone View (Player)

1) Login



➢ You can log in through smartphone. If you click “new account”, “change password”, “recover password” button, it will link to the same web page shared with web view buttons.

2) Join the Game



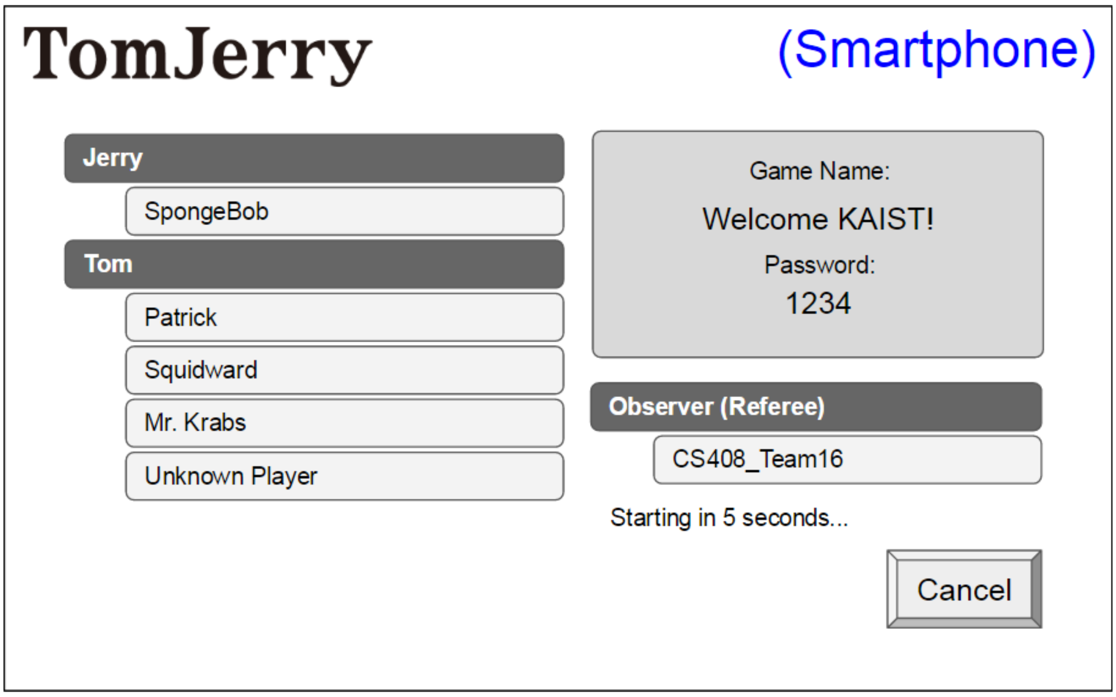
Game Name: Welcome KAIST!

Game Password: 1234

Creator: CS408_Team16

➢ You can join the game. You can also type the game password. If you fail, it will show an error message.

3) The Game Room



TomJerry (Smartphone)

Jerry
SpongeBob

Tom
Patrick
Squidward
Mr. Krabs
Unknown Player

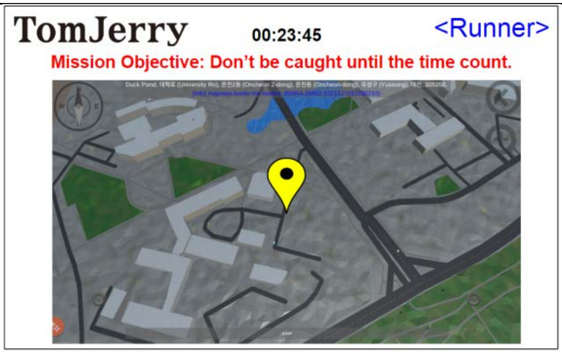
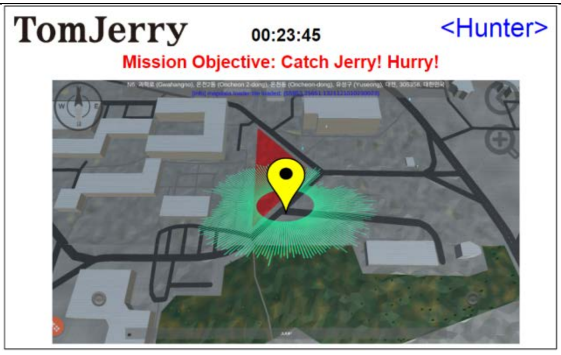
Game Name:
Welcome KAIST!
Password:
1234

Observer (Referee)
CS408_Team16

Starting in 5 seconds...

➢ After you enter the game room, you can show other players waiting. However, you can't ban other players or slot state. Also the player can't start the game (there's no START button unlike Web view).

4) Players' view

 TomJerry 00:23:45 <Runner> Mission Objective: Don't be caught until the time count.	 TomJerry 00:23:45 <Hunter> Mission Objective: Catch Jerry! Hurry!
Jerry's view has a map with pinpoint on it.	Tom's view is similar to Jerry's, but with signal visualization

➢ The screens of Jerry and Tom are different as their roles.

F. Software Interface Requirements

1. Server
    a. OS: Windows 7
    b. Server: C (TCP socket) → MicroSoft Visual Studio 2015, OpenGL library
    c. Database: SQLite → JDBC (Java DataBase Connectivity) driver
    d. CPU: 4 cores (Intel(R) Core(™) i5-4590 CPU @ 3.30GHz)
    → Open MPI: Version 2.1.0 for parallel computing (on multi-cores)
    e. GP-GPU: NVIDIA GeForce GTX 650 Ti
    → CUDA 8.0v (OpenCL) for parallel computing (on GPU)
2. Web Observer's GUI
    a. HTML 5 : supports canvas object.
    b. Pixi.js: Version 4.4.3
    c. We need those browsers: since they support HTML5 required for Pixi.js.
        i. Internet Explorer 11+
        ii. FireFox 15+
        iii. Chrome 11+
        iv. Safari 5.1+
        v. Opera 19+
3. Client's GUI
    a. Development Tool: Unity 3D 5.5.2f1 (Unity Engine), Android Studio 2.3 (Android)
    b. Android 6.0 Marshmallow + (supports Unity 3D)

G. Design and Implementation Constraints

1. Hardware Limitations.

➢ CPU: **4 cores** (Intel(R) Core(™) i5-4590 CPU @ 3.30GHz)

➢ GP-GPU: NVIDIA GeForce GTX 650 Ti

If we choose the number of threads per block to 672, then each multi-processor can run 2 blocks, so 4 cores can run 8 blocks. 672*4*2 = 5376 GPU threads can run simultaneously.

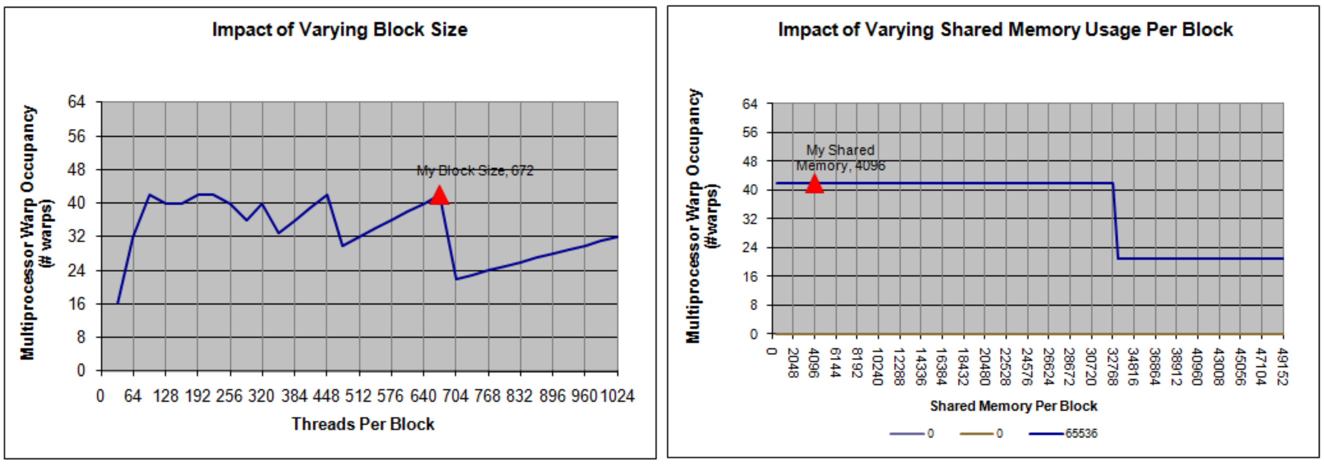


Figure 10. Test result of GPU performance on the PC. (672 threads and 4096 bytes of memory per block)

2. Application Memory and Storage Usage.

The property of the current Demo UnityMap application:

- Memory: 4.3 MB ← This is due to the dynamically downloaded map data while testing.
- Storage: 70 MB

➢ Since our application <u>does not download map data</u> dynamically, rather it will use pre-installed map data to draw. So we only need to consider the GPS and the Signal Data transmission. These data will be carried using JSON format to minimize the size.

➢ We have to renew signal data for each second, we may need to free the allocated memory using the Garbage Collector on Android (Java).

➢ Suppose we download 360 signal data from each 1 degree angle to create a circular signal graph. each signal is double float (8byte), so [360*8byte + JSON formatter = 3 KB / second]. This is equivalent to 10.5 MB per hour. So our plan is:

- MUST: No more than 20 MB
- PLAN: No more than 10 MB
- WISH: No more than 5 MB

➢ Also we don't think we will add much feature to current Demo. Our target Storage:

- MUST: No more than 100 MB
- PLAN: No more than 80 MB
- WISH: No more than 70 MB

where MB defined: MegaByte.

3. Software Architecture Design

A. Context Diagram

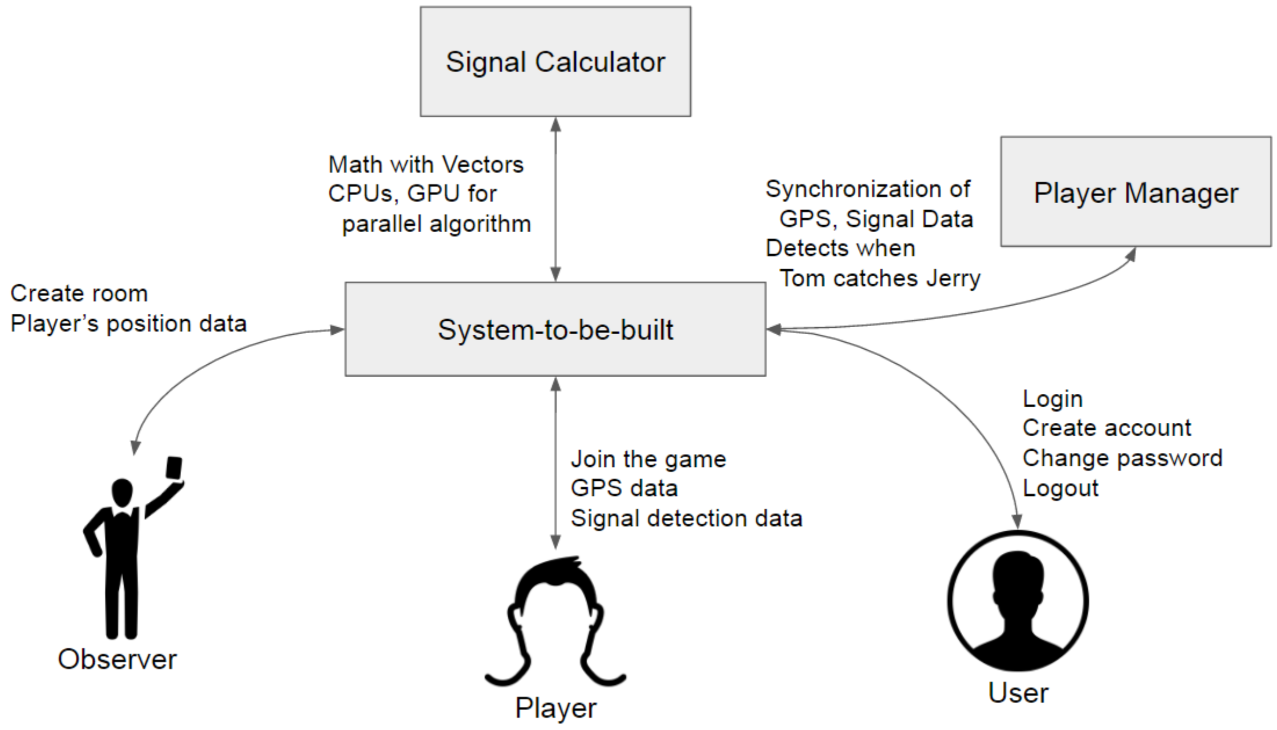


Figure 11. Context Diagram.

B. Architecture Diagram

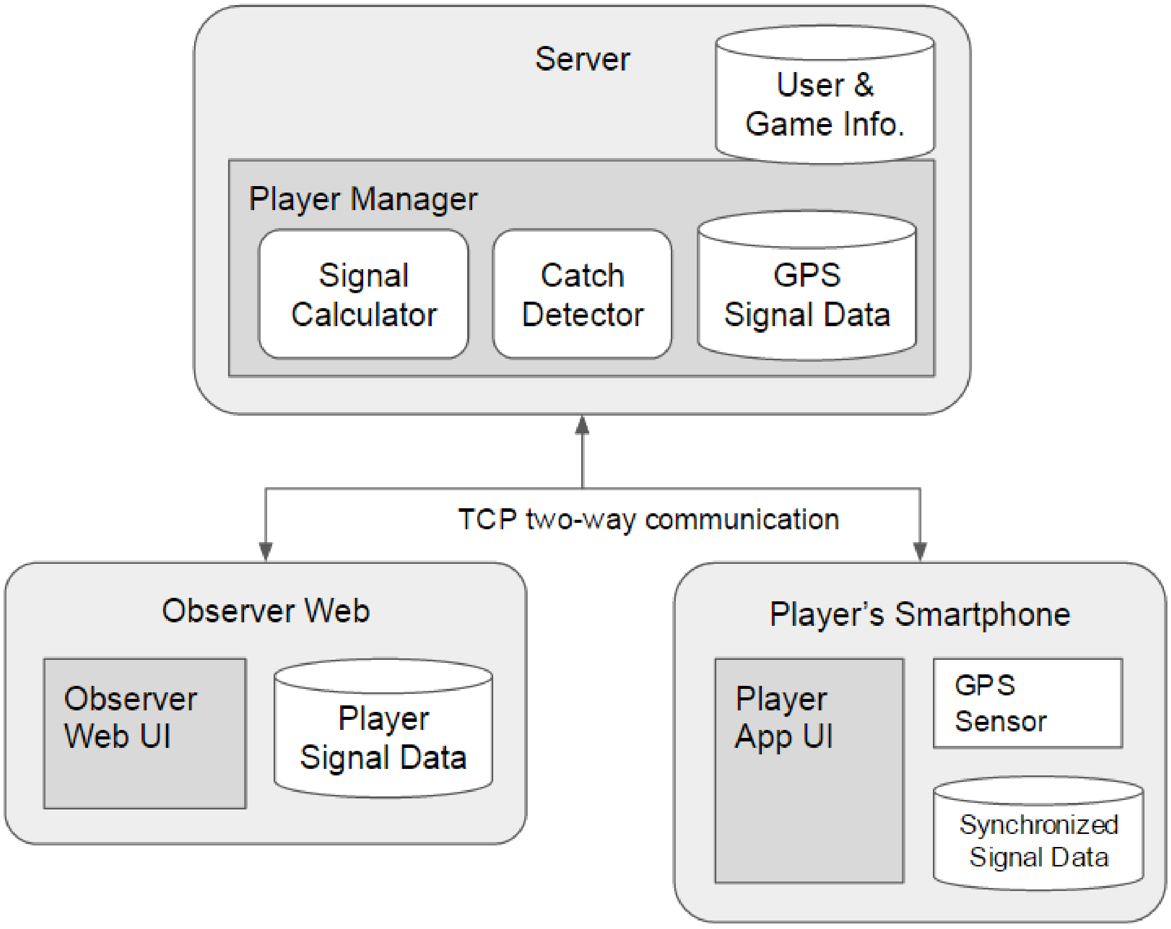


Figure 12. Architecture Diagram.

1) **Server**
    a) Signal Calculator: apply the parallel algorithm using OpenCL (NVIDIA-CUDA) to utilize GP-GPU to shorten the calculation time.
    b) Player Information Manager: controls the players' position data and signal data that will be sent to each player. Also this unit detects catch condition for the victory.
    c) User & Game Information: this database contains login and game information (e.g. player's nickname, player's role, game channel, game room number, etc.)
2) **Observer Web**
    a) Observer UI: use graphic renderer Pixi.js(http://www.pixijs.com/), a javascript graphic renderer that can run without extra installation on the webpage.
    b) Player Signal Data: manages players' position and signal data through TCP socket.
3) **Client Smartphone**
    a) Player UI: use Unity 3D game engine to display map and 3D signal graph.
    b) GPS Sensor: sends GPS data to the server through TCP socket.
    c) Signal Data Analyzer: receives strength data through TCP and draw the signal graph.

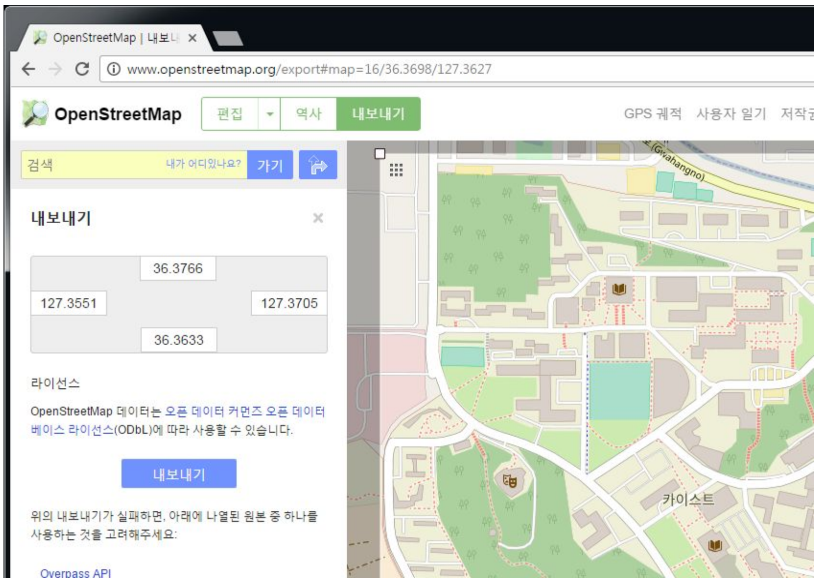
*Each component is connected through TCP two-way communication socket.

Appendix: Signal Simulation Algorithm

2017 CS408 Team #16 (last modified: 2017-05-10)

This paper introduces the method to calculate the strength of the signal detected from the generator by using GPU. All the project is saved in this URL: https://github.com/SuminHan/Tom-and-Jerry-Simulation

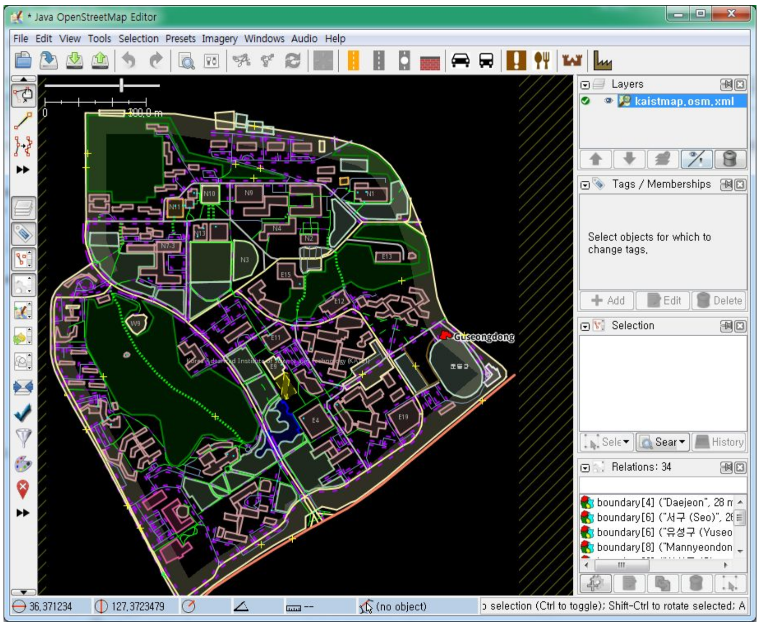
OpenStreetMap Download: www.openstreetmap.org



OpenStreetMap | 내보내기
www.openstreetmap.org/export#map=16/36.3698/127.3627
OpenStreetMap
편집
역사
내보내기
GPS 궤적
사용자 일기
검색
내가 어디있나요?
가기
내보내기
36.3766
127.3551
127.3705
36.3633
라이선스
OpenStreetMap 데이터는 오픈 데이터 커먼즈 오픈 데이터베이스 라이선스(ODbL)에 따라 사용할 수 있습니다.
내보내기
위의 내보내기가 실패하면, 아래에 나열된 원본 중 하나를 사용하는 것을 고려해주세요:
Overpass API
카이스트

Click on the extract button to download the KAIST map, osm.xml file.

Refine map using JOSM: http://josm.openstreetmap.de



* Java OpenStreetMap Editor
File Edit View Tools Selection Presets Imagery Windows Audio Help
Layers
kaistmap.osm.xml
Tags / Memberships
Select objects for which to change tags.
Add
Edit
Delete
Selection
Relations: 34
boundary[4] ("Daejeon", 28 m
boundary[6] ("서구 (Seo)", 2(
boundary[6] ("유성구 (Yuseo
boundary[8] ("Mannyeondon
36.371234
127.3723479
(no object)
selection (Ctrl to toggle); Shift-Ctrl to rotate selected; A

Remove redundant buildings and ways outside the KAIST using the JOSM tool.

OpenStreetMap xml to the edible data:

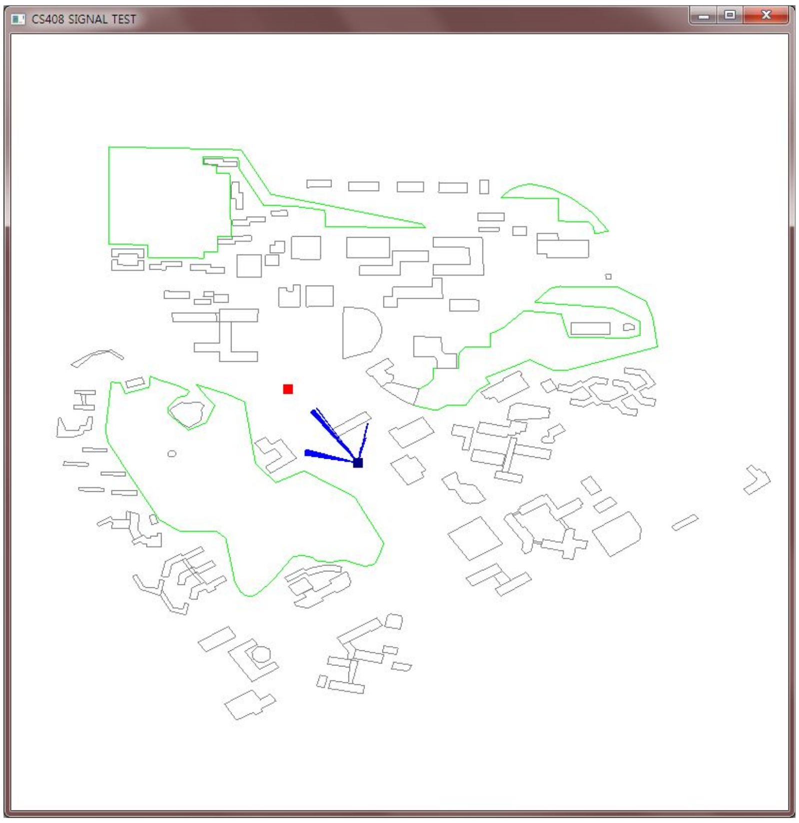
1. Install BeautifulSoup[1].

```
$ pip install beautifulsoup4
```

2. Export OpenStreetMap data that will be used.
3. Run our Python Code.
4. Then you can get the results: (e.g.)

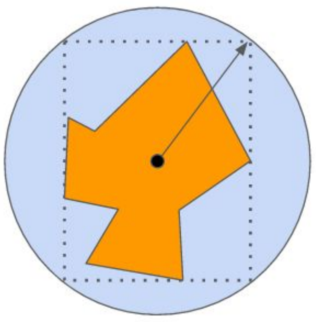
```
# i: nodes, buildings, forests
i     949   117   4
# nodes
n     36.3673981 127.3637097
n     36.3674462 127.3638063
n     36.3677069 127.3643303
n     36.3675444 127.364455
n     36.3674521 127.3642694
n     36.3674026 127.3641699
n     36.3672357 127.3638344
b E2-2     0     1     2     3     4     5     6     0
```

OpenGL: we use OpenGL to draw the KAIST map and the signals.

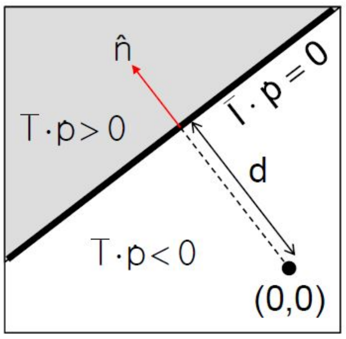


[1] https://www.crummy.com/software/BeautifulSoup/bs4/doc/ we use beautifulsoup to remove xml tags and refine it to edible format of data to be read from C code later.

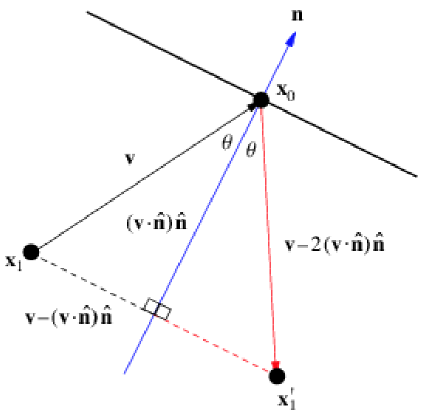
Pre-processing: We need to make a circle that encompass each polygon (i.e. building or forest). For each polygon, we first calculate the center of the vertexes by calculating the center by calculating ((maxx + minx)/2, (maxy + miny)/2). Then we calculate the maximum radius to calculate the circle like the figure.[2]



1) pre-processing (on loading)
   a. x0 = (maxx + minx) / 2
      y0 = (maxy + miny) / 2
   b. radius r, center(x0, y0)
2) calculate if |T · p| < r:
   c. then: calculate for each edges in the building
   b. else: ignore that building



Reflection Algorithm: calculate the intersection point p' using matrix calculation, and also calculate the vector reflected **v'** = **v** - 2 (**v** · n)n.



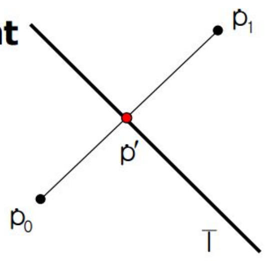
$\mathbf{n}$
$\mathbf{x}_0$
θ
θ
$\mathbf{v}$
$(\mathbf{v}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}}$
$\mathbf{x}_1$
$\mathbf{v}-2(\mathbf{v}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}}$
$\mathbf{v}-(\mathbf{v}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}}$
$\mathbf{x}_1'$

- **Interpolate to get new point**

$$p' = p_0 + t\,(p_1 - p_0) \qquad T \cdot p' = 0$$

$$T \cdot (p_0 + t\,(p_1 - p_0)) = 0$$

$$t = \frac{-(T \cdot p_0)}{T \cdot (p_1 - p_0)}$$



Detection Algorithm: there are three major process for the signal calculation.

a. Can a signal possibly hit the detector? : dist_t
b. Can a signal possibly blocked by the forests? : dist_b
c. Can a signal possibly reflected by the buildings? : dist_r

For each possible case (a, b, c), we need to compare the distances from the signal: dist_t, dist_b, dist_r to determine whether the signal will be hit the detector, blocked by the forests, or reflected by the buildings. Here's the pseudocode:

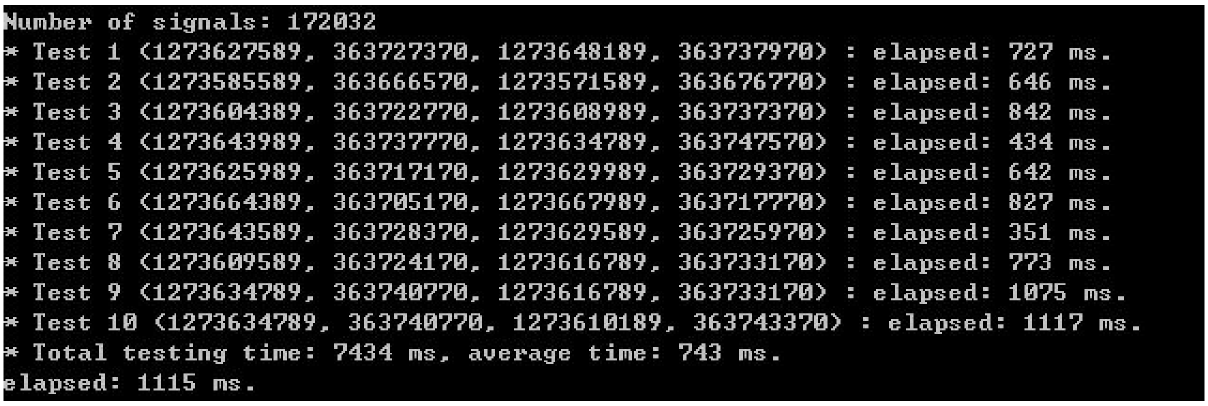
```
while (signal is alive) {
      if (reflected) {
            if (blocked) {
                  if (detected && dist_t < dist_b && dist_t < dist_r) {
                        si->ss = si->ss + dist_t;
```

[2] Using the line equation, we can easily calculate the distance from a line. By comparing the distance and the radius of the circle, we can see whether the line meets the circle or not.

```
                        break;
                  }
                  if (dist_r < dist_b) {
                        reflected_signal->ss = si->ss + dist_r;
                        *si = *reflected_signal;
                        continue;
                  }
                  else {
                        kill(si);
                        break;
                  }
            }
            else {
                  if (detected && dist_t < dist_r) {
                        si->ss = si->ss + dist_t;
                        break;
                  }
                  else {
                        reflected_signal->ss = si->ss + dist_r;
                        *si = *reflected_signal;
                        continue;
                  }
            }
      }
      else {
            if (blocked) {
                  if (detected && dist_t < dist_b) {
                        si->ss = si->ss + dist_t;
                        break;
                  }
                  else {
                        kill(si);
                        break;
                  }
            }
      }
      if (detected)
            si->ss = si->ss + dist_t;
      else
            kill(si);
      break;
}
```

GPU Parallel Programming: We are using GPU: NVIDIA GeForce GTX 650 Ti. We tested with 172032 signals using 672 blocks. Then for each signal index, we should use: int i = threadIdx.x + (blockIdx.x * blockDim.x);

Test Result: (max reflection iteration: 10)



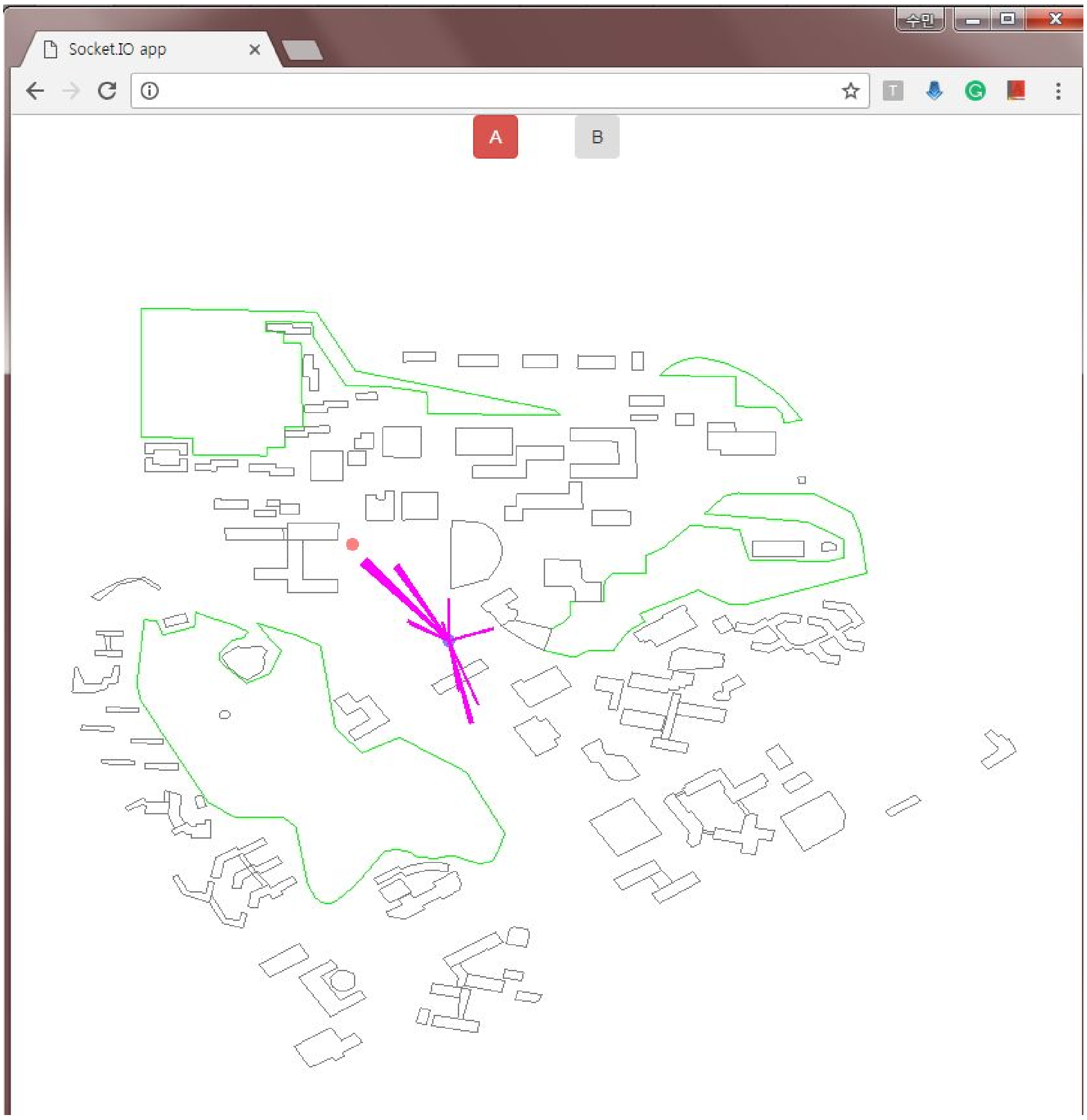
```
Number of signals: 172032
* Test 1 (1273627589, 363727370, 1273648189, 363737970) : elapsed: 727 ms.
* Test 2 (1273585589, 363666570, 1273571589, 363676770) : elapsed: 646 ms.
* Test 3 (1273604389, 363722770, 1273608989, 363737370) : elapsed: 842 ms.
* Test 4 (1273643989, 363737770, 1273634789, 363747570) : elapsed: 434 ms.
* Test 5 (1273625989, 363717170, 1273629989, 363729370) : elapsed: 642 ms.
* Test 6 (1273664389, 363705170, 1273667989, 363717770) : elapsed: 827 ms.
* Test 7 (1273643589, 363728370, 1273629589, 363725970) : elapsed: 351 ms.
* Test 8 (1273609589, 363724170, 1273616789, 363733170) : elapsed: 773 ms.
* Test 9 (1273634789, 363740770, 1273616789, 363733170) : elapsed: 1075 ms.
* Test 10 (1273634789, 363740770, 1273610189, 363743370) : elapsed: 1117 ms.
* Total testing time: 7434 ms, average time: 743 ms.
elapsed: 1115 ms.
```

Online Realtime Signal Calculation Server Prototyping

We used node.js (server), socket.io (websocket), and pixi.js (graphic renderer). (see WebGraphic and SeqProgram in the GitHub).

```
function runCalculation(){
    var prog = "SeqProgram.exe " + position.gx + " " + position.gy + " " +
position.ax + " " + position.ay;
    exec(prog, (err, stdout, stderr) => {
        if (err) {
            console.error(err);
            return;
        }
        signal = JSON.parse(stdout);
    });
}
runCalculation(); //initalize
```

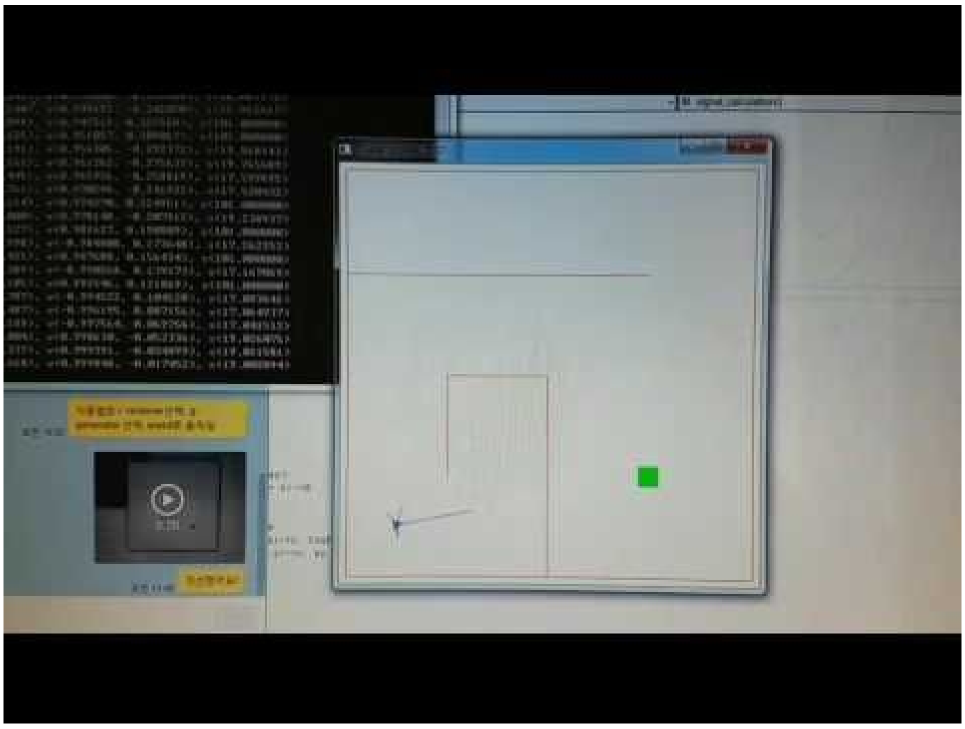
The user can choose the fox (runner) or the hound (chaser) and move using keyboard ‘asdw’ or ‘left, right, up, down’ keys. Then the signal is calculated by executing the SeqProgram.exe and get the JSON array of the signal output. Then the output is sent to the client, and the client javascript code interprets and draw the signal graph.



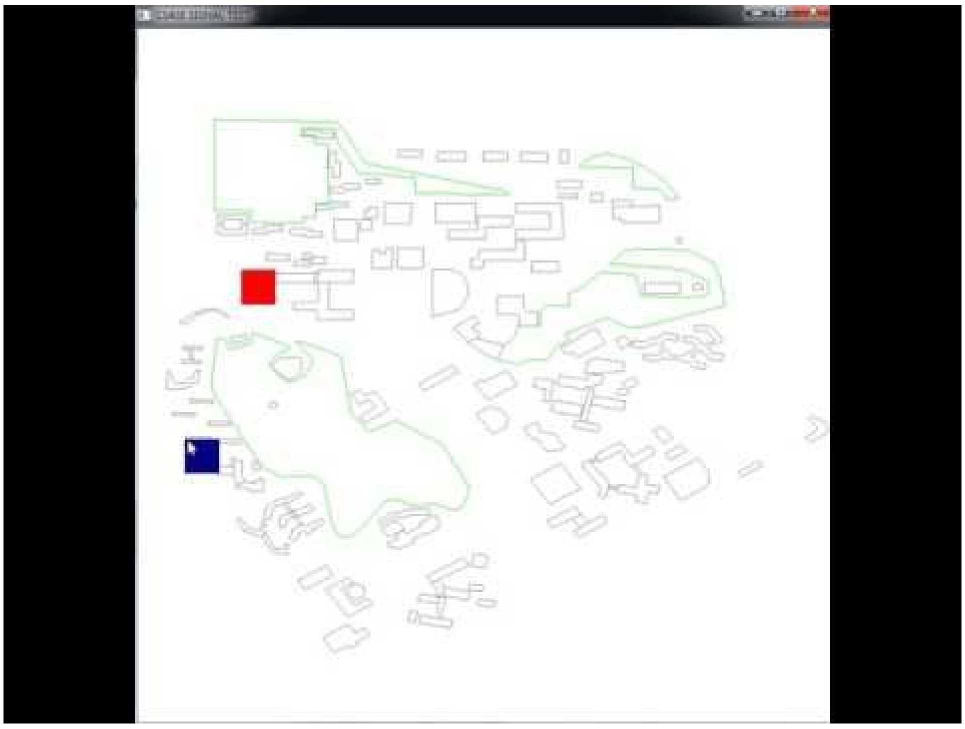
We can draw the signal graph using Pixi.js on the web similar to OpenGL in the kernel. This prototype only used Sequential Code, not GPU yet.

Ongoing Improvement

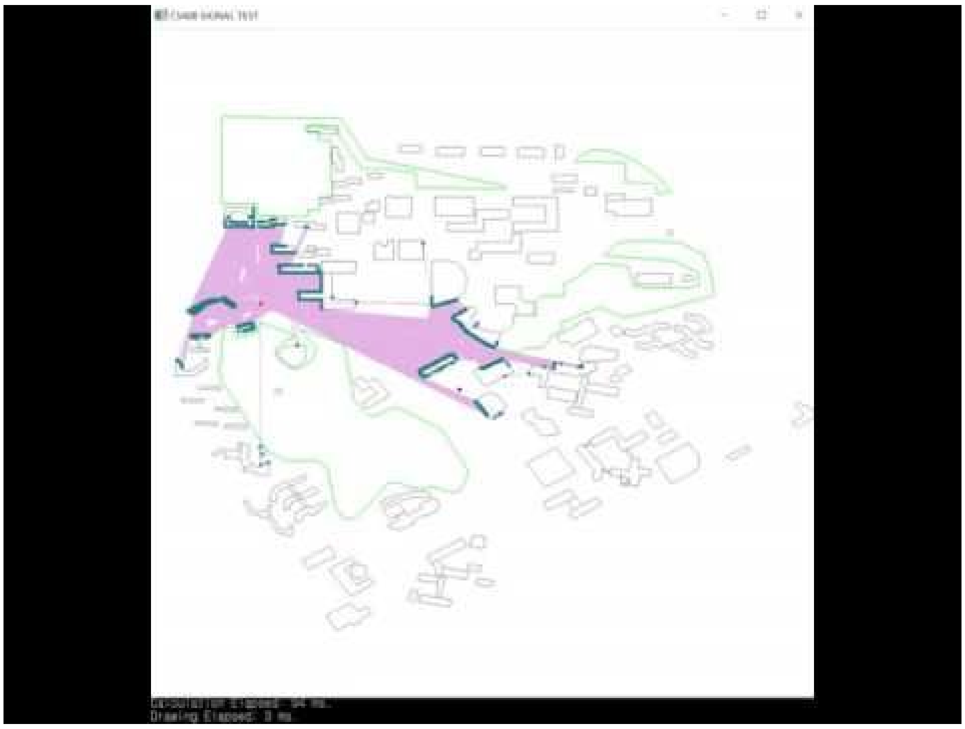
- [4/22] Reflection Simulation on: # of signals:360
  https://youtu.be/6H-wo_vZulA



- [4/25] Signal blocking by forests on KAIST MAP
  https://youtu.be/OocRdbA8gdY

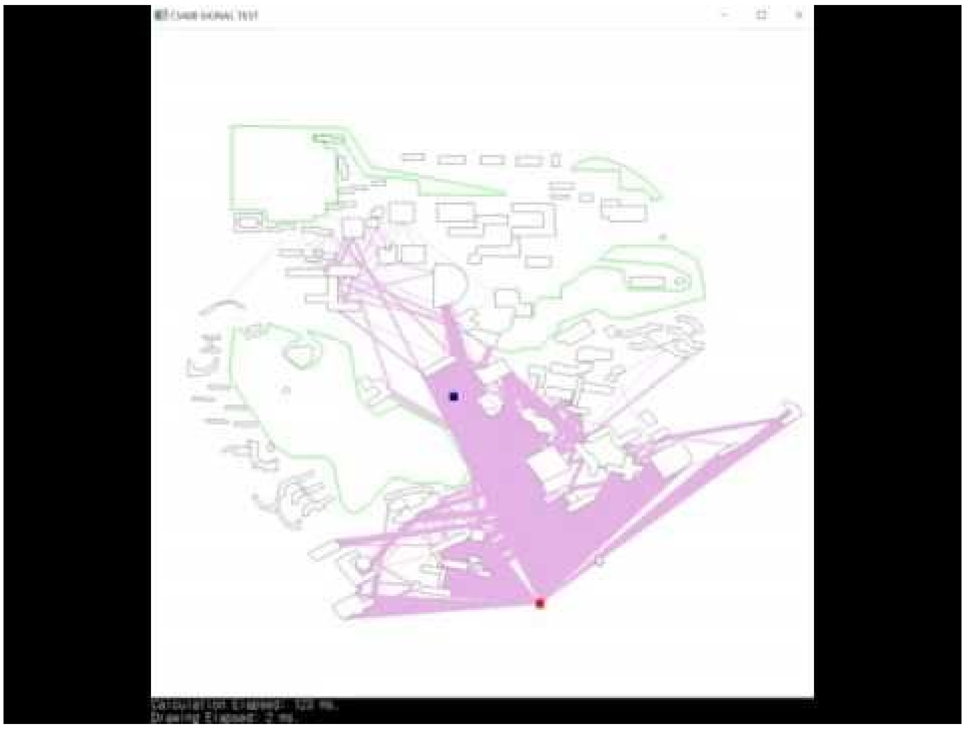


- [4/29] Signal reflectionon KAIST MAP (sig#3600, long double, sequential)
  https://youtu.be/gqyN_1NY7mE



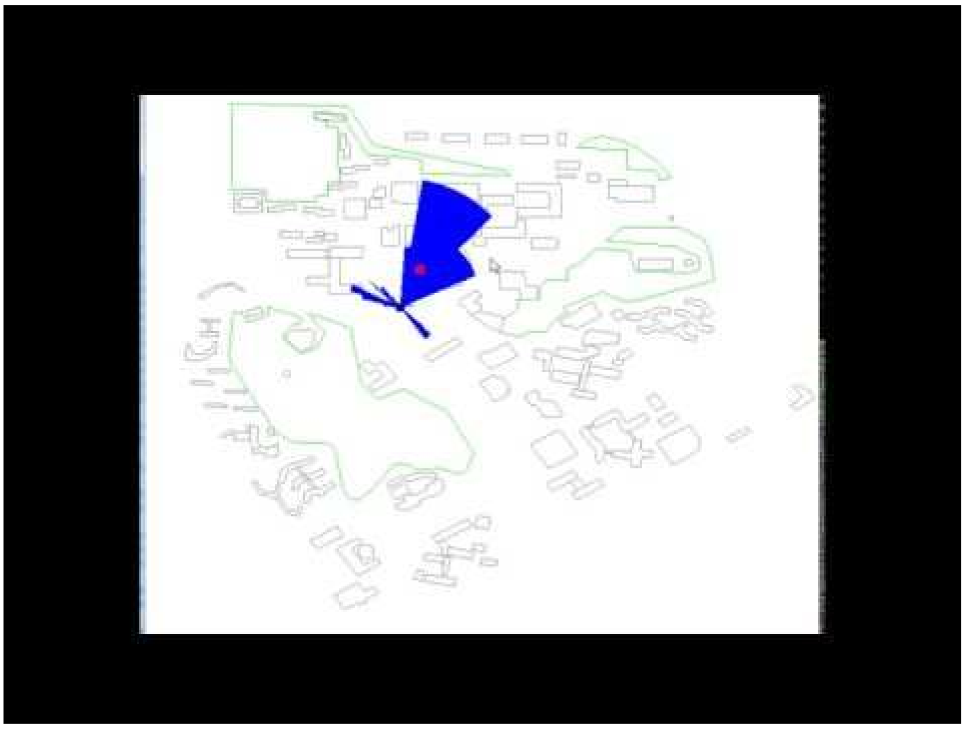
```
Calculation Elapsed: 94 ms.
Drawing Elapsed: 3 ms.
```

- [4/30] Signal reflectionon KAIST MAP (sig#3600, 64bit int, sequential)
  https://youtu.be/A-gF2tzhoB0



```
Calculation Elapsed: 123 ms.
Drawing Elapsed: 2 ms.
```

- [5/1] Signal reflectionon KAIST MAP using NVIDIA GeForce GTX 650 Ti. https://youtu.be/2uKSR1LshKs
    - Number of signals : 172032, max reflection: 10



- [5/8] Web Online Realtime Graphical Simulation Prototoype https://youtu.be/h7f6DAYvjwM