Mauraders Map: Proposal, CS 5412

Haotian Pan, Yize Li, Hang Chu

February 10, 2014

1 Goals

We want to create an online game Zombie City played on webpages.

2 Motivation

To provide a distributed gaming platform for Zombie City and other similar games on webpage for any portable devices:

- Simple framework on webpage for real-time interactive multiplayer games.
- Service guarantees for developers-availability, fault-tolerance, consistency

3 Analytical description

3.1 Game modeling

3.1.1 Players

- State: location, human or zombie bit, ID, bullets
- Actions: move, shoot, collect
- Interactions: zombie eat humans, human shoot zombies

3.1.2 Environment

- Grid maps based on reality
- Bullets placed in the world for humans to pick up
- Obstacles: walls, etc.

3.1.3 Events

- Move
- Pick up bullets
- Shoot or get shot, bite or get bitten
- Get to the safe zone

3.1.4 Rules

- Humans win by shooting down all zombies
- Zombies win by eating all humans
- Human could shoot a zombie a few blocks away in a line
- Zombie could only attack humans in their adjacent blocks, and turns them into zombies by only one bite

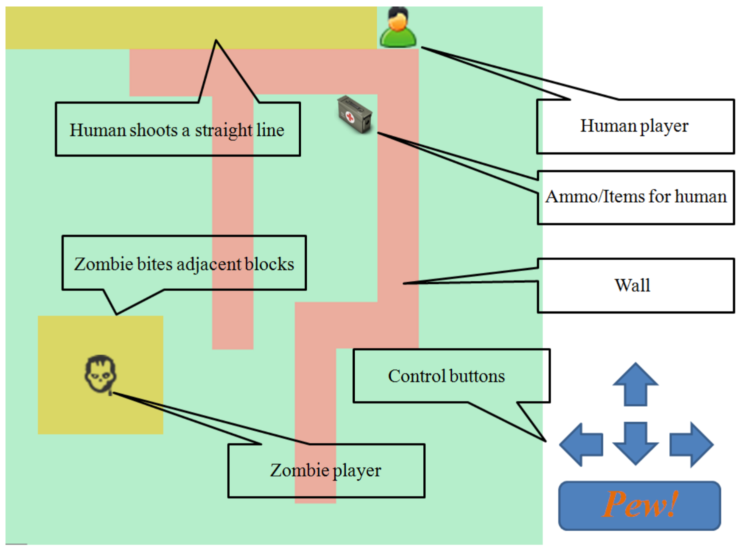

Figure 1: A simple illustration of the game concept and GUI (simulate map version)

3.2 Server

Given the game model, create an instance of a game:

- Using existing game abstraction library
- Clients connect to/select/play games
- Superimpose live data (collect, decode, merge), apply updates on the fly -¿ new state
- Compact and serve requesting clients

3.3 Client

Find server, select/connect to game, play

- Learn how to utilize APIs of the phone from the webpage to get sensor data

- Implement connections, IDs, and participation
- Find how to stream information (location, actions, items) with low bandwidth costs

4 Milestones & Dates

4.1 Milestone 1

4.1.1 Create the game map

According to our game model, the map is 2D. It can be either manually crafted, or can be retrieved from real-world map such as Google Maps.

4.1.2 Create game GUI

We will implement a webpage-based GUI for gamers, which includes: a map display with real-time player status, buttons for taking actions such as move or attack (we will also try locating by GPS to substitute manual moving control).

4.1.3 Traditional single server-client framework

We will build a simple gaming framework, using only one server. Every user is connected to this server.

4.1.4 Testing maximum response speed

We will test the real-time performance of the simple framework.
Date: March 5

4.2 Milestone 2

4.2.1 Improve communication efficiency

In the previous stage we only test response speed, in this stage we will improve communication efficiency to meet the 100 ms requirement.

4.2.2 Multi server support

Only one server cannot be called a cloud. We will implement strategies to enable our framework working at multiple servers. Different game instances will be hosted on different servers.

4.2.3 Load balancing

We dont want to exhaust one server while other servers are barely used, so we will design and implement strategies for assigning servers, such that the work will be distributed to servers averagely.
Date: March 26

4.3 Milestone 3

4.3.1 Error tolerance (replicating servers)

In this case, we will duplicate the data of each server to two other redundant servers. Changes in any one of the servers would cause the other two to synchronize. This ensures that clients would not receive bad data after one server crashed.

4.3.2 Clients can communicate with every replica that handles their games instance

Typically, we want to do synchronizations between servers whenever a client has communicated with one server. Though this is impossible according to TAs notes, we want to try and see why.

4.3.3 Clients should witness changes in a consistent way

For this part, we may actually implement with the isis tool. We may come across some protocols to guarantee consistency between clients.

4.3.4 Maintain 100 ms delay limit

Assume that we have made several changes from milestone 4.2.1 in order to meet the requirements, we still want to check and maintain a 100 ms delay limit.
Date: April 14

4.4 Milestone 4

4.4.1 Recovery protocol for failed server replicas

We will ensure that the recovered replicas get all the information that the current operational replicas have to keep consistency.

4.4.2 Improve efficiency

We then will try to decrease the delay in response as possible as we can.

4.4.3 Adjusting gaming model

Making the game funnier and more vivid to play with.
Date: May 6

5 Tools

5.1 Front end

HTML5, javascript, webSocket, Box2D

5.2 Back end

Isis2, Mono, Visual Studio

6 Evaluation & Visualization

6.1 Demo

- Setup the service on the server
- Connect clients to the server and use the service
  - Move around, use environment objects if found, avoid zombies, avoid obstacles
  - Basic rules about the game, (i.e. how to move and to use items)
  - Show the interactions between players(human becoming a zombie, shooting at zombie)
- Run test cases:
  - Multiple clients single server (stress test for capacity checking)
  - Multiple servers; no replication (Load Balancing)
  - Replicated servers
    - Introduce faults: availability, latency, consistency, load
  - Recovery
    - Load restoration

6.2 Poster

- Architecture of our system
- Technologies we use and how did we use it to achieve our goal.
- Experiment results:
  - Scalability
  - Latency
  - Availability under heavy load
  - Failure handling
  - Recovery

7 Team coordination

- Team members: Hang Chu (hc772), Haotian Pan(hp343), Yize Li (yl2376)
- Scheduling: Two meetings per week, 3-4 hours for each meeting.
- Yize Li: Web game design, communication framework of single server implementation and multiple server implementation.
- Haotian Pan: GUI, consistency, game maps.
- Hang Chu: GUI, communication framework and consistency strategies.
- All: game model, replication, recovery