Intermediate Report II

Marauder's Map, CS 5412

Haotian Pan(hp343), Yize Li(yl2376), Hang Chu(hc772)

April 13, 2014

1. Summary of Progress in Intermediate Report I

In the last report, we have already made a game able to run. Several players could join one game instance and can only move in the area and observe each other moving. The game is played on the web browser and thus can be played both using portable device and computer. The game turned out to be fast in response, and have a special mechanism to achieve server replica, e.g. to continue the game when one server crashes down.

2. Current Status

In this part, we are going to show our completed works since last report in details.

2.1 Improved Game Logic

As can be seen in our previous report, our game was a tank game. This is not the outcoming we had wanted for a Human vs. Zombies game, though the underlying mechanism is the same. To improve this aspect of our project, we implemented more details of the game logic. Overall speaking, there are 3 major changes:

- a. We added two types of players: human players and zombie players.
- b. We added a “war fog” effect, such that a play can only see surrounding environment instead of the whole map.
- c. We implemented the attack function, so now players from different sides can kill each other.

2.2 Robust Data Replicating Strategy

In our first attempt, we stored players’ information in every client’s disk, and let the client send the stored information to server in order to retrieve a game. We were also using JSON files to record these data to server. However, this turned out to be a bad way to do server replication since players may cheat by sending a modified version of data to server while retrieving a game. Therefore, we have decided to use the *memcached* technique.

2.3 Load Balancing with Router

Users can choose which game server to go to manually on a webpage by clicking the corresponding link or connect to the game server assigned to them by the router by clicking a special link. We have already implemented a random router and a round-robin router, but both are not satisfying enough.

2.5 Errors fixed

In the last report, we made every player send their data including ID, location, velocity and orientation back to the new leading server which just took up the job of running game logic and communicating and all the other tasks that the previous leader was doing before crash down. It turns out to be a bad idea, because the consistency can be easily violated if players take advantage of this and send incorrect data to the new leader. Thus we adapted our technical approaches by adopting memcached to store user data and provide easy and safe access to data stored to all the servers in the same group.

3. Milestone Status

In this section, we report the status of our previously planned milestones: finished before Intermediate Report I, finished between Intermediate Report I and II, ongoing, not started.

3.1 Milestone 1

- Create the game map
- Create game GUI
- Traditional single server-client framework
- Testing maximum response speed

3.2 Milestone 2

- Improve communication efficiency
- Multi server support

In Intermediate Report I this is only partially finished. In our current implementation, we set multiple servers and share/replicate game data using memcached. We also implemented protocols of assigning idle server into service when previous leader server fails, this protocol is able to recover game data by retrieving from memcached.

3.3 Milestone 3

- Error tolerance (replicating servers)

In our Intermediate Report I this is partially finished. We now finished this by using the memcached approach described above.

- Clients should witness changes in a consistent way

In our Intermediate Report I we marked this as finished. However, this was not actually finished as at that time we did not have data replication protocols, all game data was stored locally and game cheating would be a severe problem. (We would like to thank our TA Stavros Nikolaou for pointing out and explaining this.) We now have data replicating implemented.

- Maintain 100 ms delay limit

3.4 Milestone 4

- Recovery protocol for failed server replicas
- Improve efficiency
- Sophisticated load balancing
- Adjusting gaming model

4. Demonstration

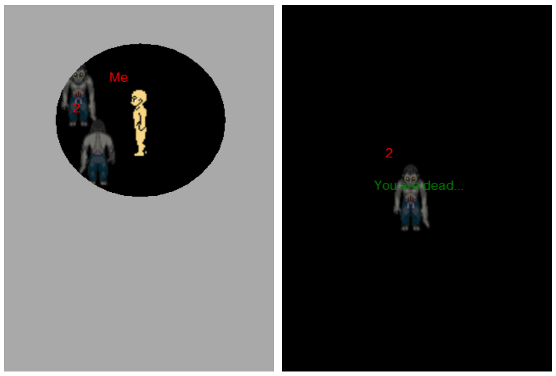
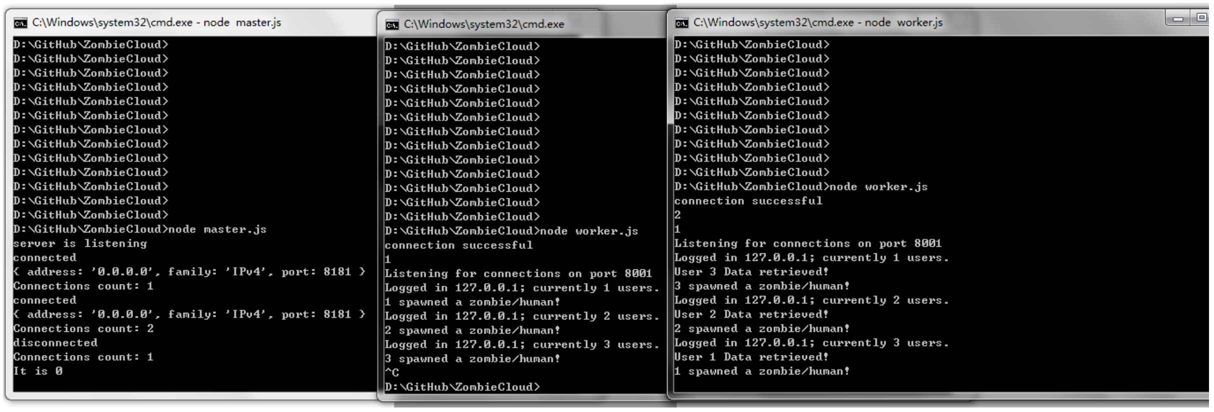


Figure 1. Improved game logic. We added two types of player, “war fog”, attacking and killing.



```
C:\Windows\system32\cmd.exe - node  master.js
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>node master.js
server is listening
connected
{ address: '0.0.0.0', family: 'IPv4', port: 8181 }
Connections count: 1
connected
{ address: '0.0.0.0', family: 'IPv4', port: 8181 }
Connections count: 2
disconnected
Connections count: 1
It is 0
```

```
C:\Windows\system32\cmd.exe
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>node worker.js
connection successful
1
Listening for connections on port 8001
Logged in 127.0.0.1; currently 1 users.
1 spawned a zombie/human!
Logged in 127.0.0.1; currently 2 users.
2 spawned a zombie/human!
Logged in 127.0.0.1; currently 3 users.
3 spawned a zombie/human!
^C
D:\GitHub\ZombieCloud>
```

```
C:\Windows\system32\cmd.exe - node  worker.js
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>
D:\GitHub\ZombieCloud>node worker.js
connection successful
2
1
Listening for connections on port 8001
Logged in 127.0.0.1; currently 1 users.
User 3 Data retrieved!
3 spawned a zombie/human!
Logged in 127.0.0.1; currently 2 users.
User 2 Data retrieved!
2 spawned a zombie/human!
Logged in 127.0.0.1; currently 3 users.
User 1 Data retrieved!
1 spawned a zombie/human!
```

Figure 2. Server running screenshot. Master server (left) assigns an idle worker server (right) as the new leader server when previous leader server (middle) fails. The new assigned leader is able to recover user data to keep consistent user experience.