Distributed Systems Spring 2013

TEAM

For Example

Formally The Engineers & Floppy Disk

MEMBERS:

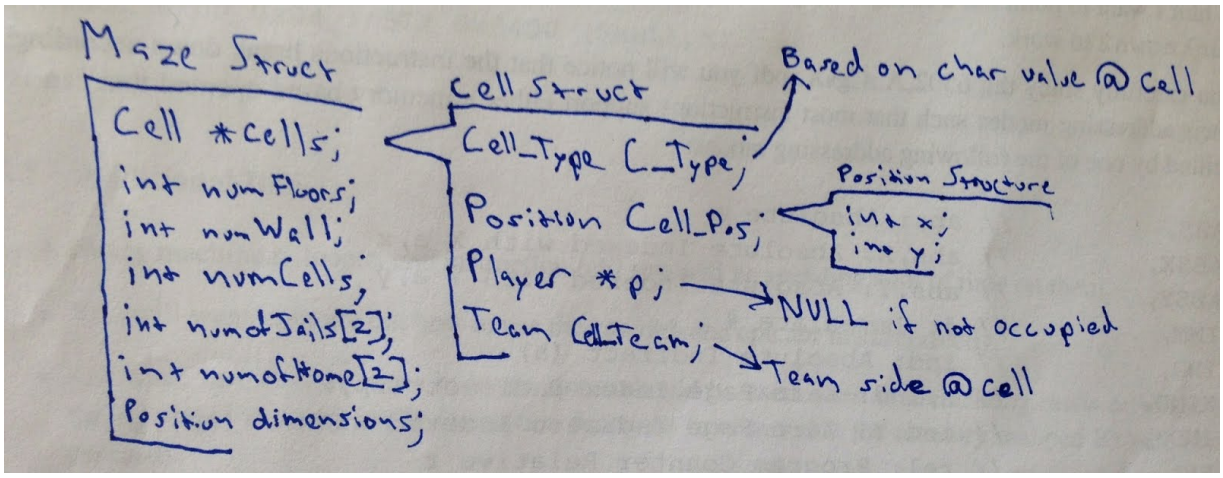
Name	Email
Gurwinder Singh	gsinghny@bu.edu
Matthew Lee	matt2lee@bu.edu
Alejandro Pelaez Lechuga	apelaez@bu.edu
Amelia Martinez	mely91@bu.edu
Yanolsh	yanolsh@bu.edu
Jcmartin	jcmartin91@bu.edu

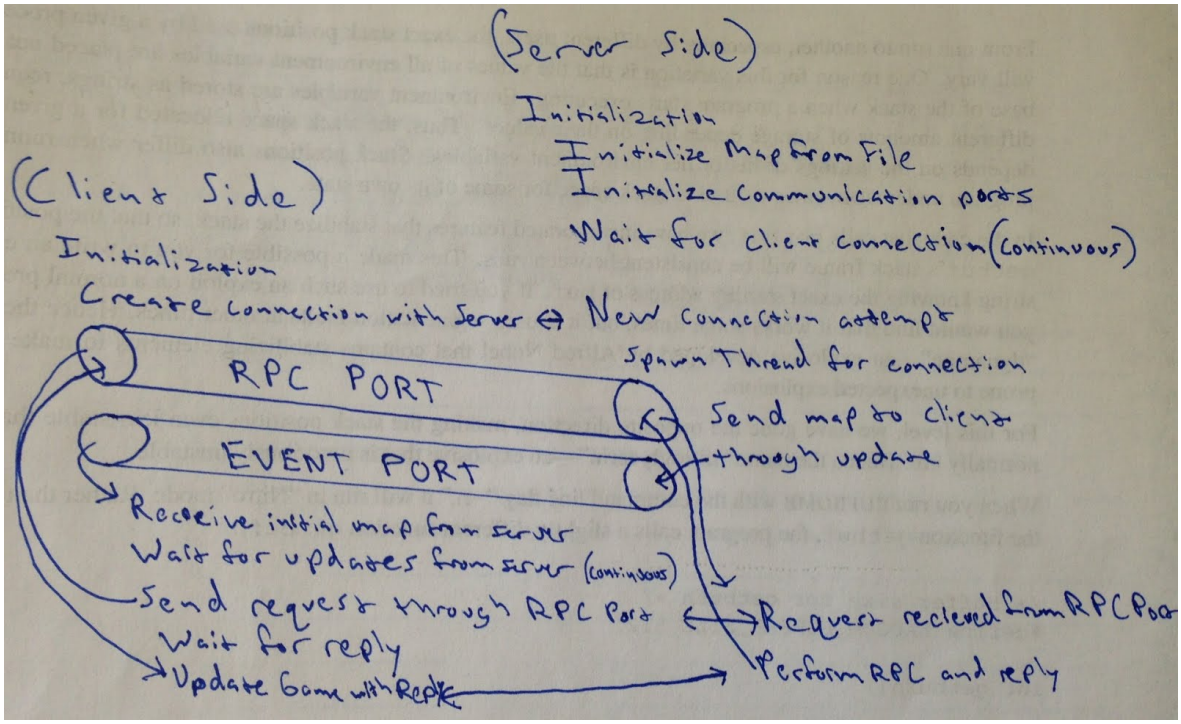
The game of Capture the Flag can be programmed in different ways according to its architecture. Our game of Capture The Flag is run on a server and a client in which their role will be to work together to execute functions and achieve a working game through rpc method calls. The server will maintain a state of the game which it shares with the client that have initiated contact with the server in order to play the game.

When discussing the role of the server it is wise to begin with it's initiation. Primarily, a map file is parsed, character by character, to be stored as the 'master' state of the game. During this process, the server uses a Maze structure to store each individual character as a cell; each cell contains information specific to its location and properties such as it's type and its position relative to the map. While filling the maze structure with cells, it also updates values in the maze, for example both total number of floors and total number of walls are tracked. The server initialization also includes the setup of listening sockets to see when a request for connection is made. If a client connects to the server's ip and open rpc port a connection is made. This grants the ability for the server to take requests from the client to update the map. Requests can be use for various actions like joining and moving, but the validity of this request (such as moving into a valid cell) is maintained by the server's game logic. Furthermore, if the map is successfully updated, the server sends out this update to all players connected through the event port. The server can also execute a dump function which essentially prints out the current state of the map. These functions are based in a second layer of code, or our library, and can be called by our server to handle initialization, requests and updates. Ultimately the server uses rpc based requests from clients and game logic to maintain a stable and up-to-date game for all clients.

As mentioned above, client can send requests and get updates from the server when needed. Since the server uses sockets to listen for clients, the client initiates a connection with the server using an ip address and rpc port. Once the client tethers a connection with rpc ports and event ports, requests and updates can be sent and received. However, before any requests

are made the client is initially sent the current state of the game from the server. A valid player requests can include right, left, up and down movements as well as advanced functions like joining the game and spawning, jailbreaking, tagging , manual item pick up, use item, and drop item. Such requests are sent to the server where the information is used to update the map and other players. Fundamentally, the client sends a request, and the server returns an update as a response.





In order to better understand the data structures used as well as the running of the game some diagrams have been included. Some of the structures used for the map are visible in the first diagram, while the second contains the server and client side from initialization to a request. In this instance, the server would update clients through the event port after replying to a client.

Trustworthiness in an important aspect when dealing with clients. It can affect the game negatively as well as other player's game state if the client can tamper with other clients. As of now, we are under the assumption the client is trusted up to the point where plausible impersonation of a player may occur. However, although one may be able to impersonate another client, since the game logic is on the server and it is secure, the impersonator will not be able to crash the game on behalf of the other clients. Upon receiving an impersonated request, the server maintains the game logic by treating it as a normal request and making sure if it is a valid one or else it will reject it. If a request is rejected by the server, no one on the event channel gets affected by the rejected request. The impersonator's request can at most affect the client being impersonated. If the request is valid, the server will update the other clients with the victim's new location or state. Without a way for some adversary to impersonate the client our clients are assumed to be trusted.

There are many considerations that must be evaluated when impacting the performance and scalability of the server. Initially the performance is bounded by the sending of the map when a client initializes a connection. Since the whole map is sent to the client only once, the data needed to be sent over updates is immensely reduced for later updates. As the game progresses and more clients join the game, the performance will be impacted merely by the number of clients requesting updates from server. Overall, a greater number of players will

impact the game’s performance and scalability in a negative way.