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

Discuss and define an architecture for you game.
1. What roles and function the server and clients will take?
2. How will you decompose the software (a diagram might be useful here)?
3. Are the clients considered trusted? Eg. Will the client software be trusted to enforce the games rules and semantics? What considerations will impact the performance and scalability of the server?

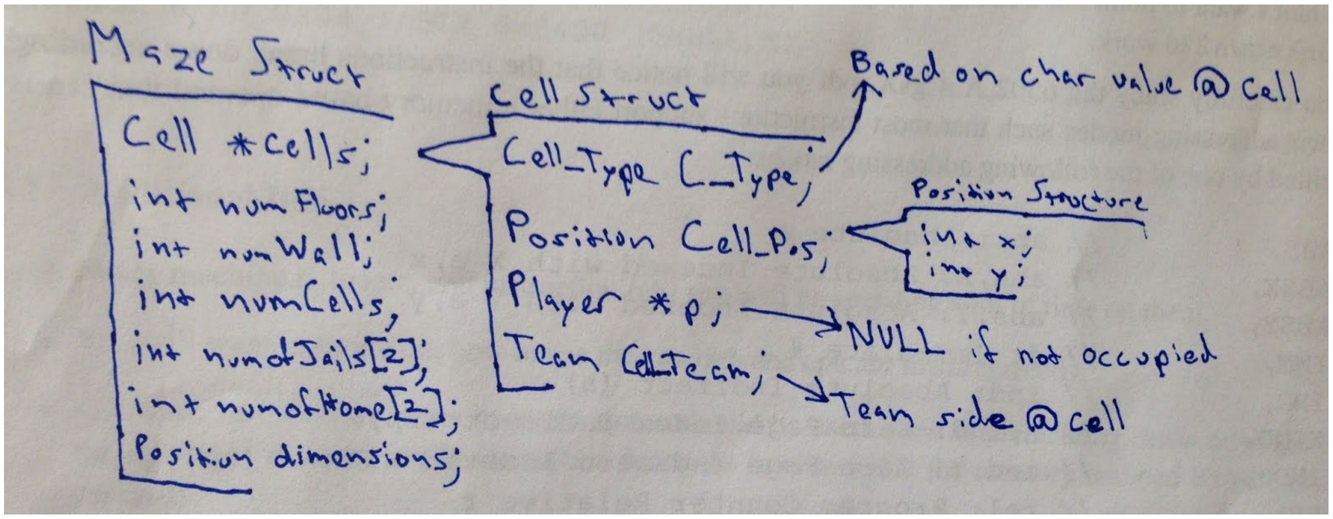
---

role- what the program is supposed to do
function- just carry out specific actions for the programs
server- first parse the map, sets up listening sockets, takes requests from client to update map (in order to update map) updates players when needed through event channel
server can dump map- prints out the map
client- sends requests, move, jump etc. initiates connection to server(rpc port and event port). receives updates.
initiate with <ip, rpc port> when your running the program you need to supply it with ip and port. ip wheres the server. and waht port the server is listening on. connet to external place. got to know wehre going.
sets up connections to server (rpc + event channels)
can receive updates from server when server has more upto date map via event port
can send request via rpc port
Are the clients considered trusted? Eg. Will the client software be trusted to enforce the games rules and semantics? What considerations will impact the performance and scalability of the server?
truested- cant affect game negatively, cant affect other ppls game. assume that it is trusted up to the point where if he is impersonating somebody. client still has your address. server will know.
1. logic side done on the server. client cant affect other ppls maps or the servers maps unless its a valid request. possiblity or impersonation. assuming position of other player
sending map all at once. scale of player. new player, keep on broadcasting. as of now sending on map, future sending updates. performance lower in current version, only thing taht could impact performance is number or players, how many requests sent and received. two ppl moving in same exact time.

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.



Some diagrams here help to understand the data structures used for the game as well as general protocols.

Trustworthiness in an important aspect when dealing with clients. It can affect the game negatively as well as other players game state if the client can tamper with the game logic. Our assumption of the client is that it is trusted up to the point where plausible impersonation of a player may occur. Although you can impersonate a player, the game logic is still sound because the server will handle the impersonators requests. Upon receiving this request, the server maintains the game logic by making sure if it is a valid request or else it will reject it. When the request is terminated, no one 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 state. Without a way for some adversary to impersonate the client it is assumed it is always 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. As the game progresses and more clients join the game, the performance will be impacted by the amount of updates the server has to send out. Overall, a greater number of players will impact the game's performance and scalability in a negative way.