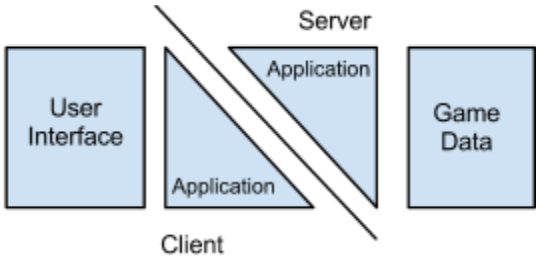
Architecture

For our implementation of the Capture the Flag game, we decided that the safest course of implementation would be to have the server side of the application perform most of the actual running of the game; we reasoned that this would be better than performing most of the calculations on the client side because it resulted in less risk of players becoming desynchronized with each other, which would ultimately render the game unplayable. However, we thought it would be good to include many of the standard error checking procedures on the client side so that we could save unnecessary transmission time between client and server. Our general architecture scheme ends up looking similar to example c from figure 2-5 in the textbook:



While the server will be doing the brunt of the actual gameplay, the client will be assisting with simple error checking procedures. If the server suffers in performance as the number of players increases, then we can transfer more responsibility to the client.

Client

The client will take commands in from the user. Then the client will check to make sure that the command makes sense, i.e. that no nonsensical command has been entered. Provided that the command is valid, the client will encode the information and transmit it to the server. After the server has responded, the client will output the result of the server calculation, whether the player's move was valid or not, and prompt the user for another command.

Server

The server will decode the information sent from the client to determine what the player's move is. The server will then check to see if the player's move is valid, i.e. the player isn't trying to move through a wall or pick up his own flag, etc. If the move is invalid, the server will send back to the client an invalid move error. If the move is valid, the server will update the game map with the player's move. The server will then encode and send to

the client an updated game map. Whenever any player performs an action, the server will send updated game maps to ALL clients in order to keep the clients synchronized with each other.

Depending on the number of clients playing the game, the server can be subject to performance decreases. However, having the clients error check to make sure the players are inputting valid commands will save the server several additional, and possibly meaningless, calculations. This does depend, however, on the fact that the server will trust the client to successfully error check. Because we are enforcing all of the actual gameplay rules on the server side, the server doesn't need to trust the client to do more than check to see if the player is actually inputting valid commands, which eliminates a good deal of variability coming in from the client.

One of the weaknesses of this style of architecture is that the scalability of the system isn't amazing, as it relies heavily on the performance of the server. So as more players start to access the server, the server will have to work harder to keep the game running. However, having all gameplay processes on a single server is very good for administration scalability, because as the number of players increases the difficulty of managing the system stays the same. Another issue that could arise in terms of scalability would be physical location, which could impact the latency and response of the server to far outlying clients; this doesn't seem to be a huge issue on its own, but if compounded with an increase in the number of players it could be an issue worth looking at.