CompSys302

Multiplayer Game Development

Group 9

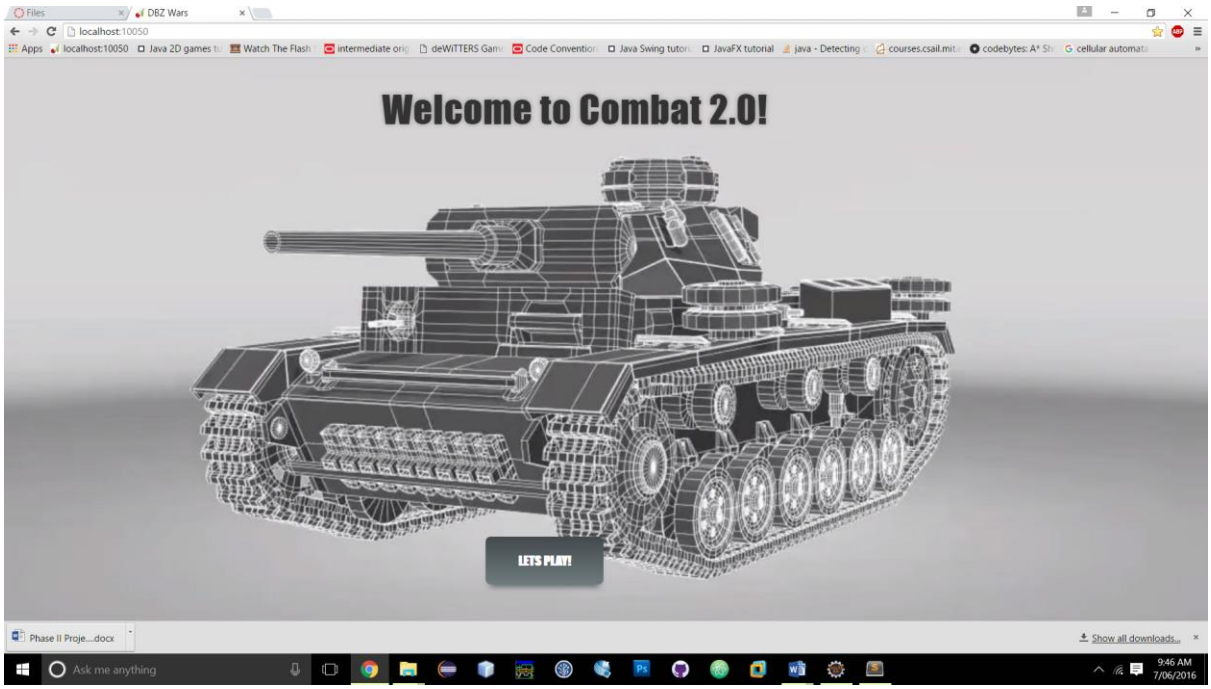
Table of Contents

Summary

The point of creating this webserver is so that we can enable the ability for users of our tank game to challenge users of other tanks games across a network. Keeping in mind that this is still designed for and eight year old, we decided to keep or UI relatively simple. The system developed allows users to play against others users who are on the same network in central-server/p2p style hybrid system. Several protocols were made to ensure that our tank game works well with others. These include data transmission and retention for drawing objects, challenging others, responding, understanding the game rules of the respective parties, etc.

Given our system could accomplish this, it also had to be able to send a receive requests from the central server Application Programming Interfaces'. These include being able to login in, view other users, log off etc.

Meeting the Requirements

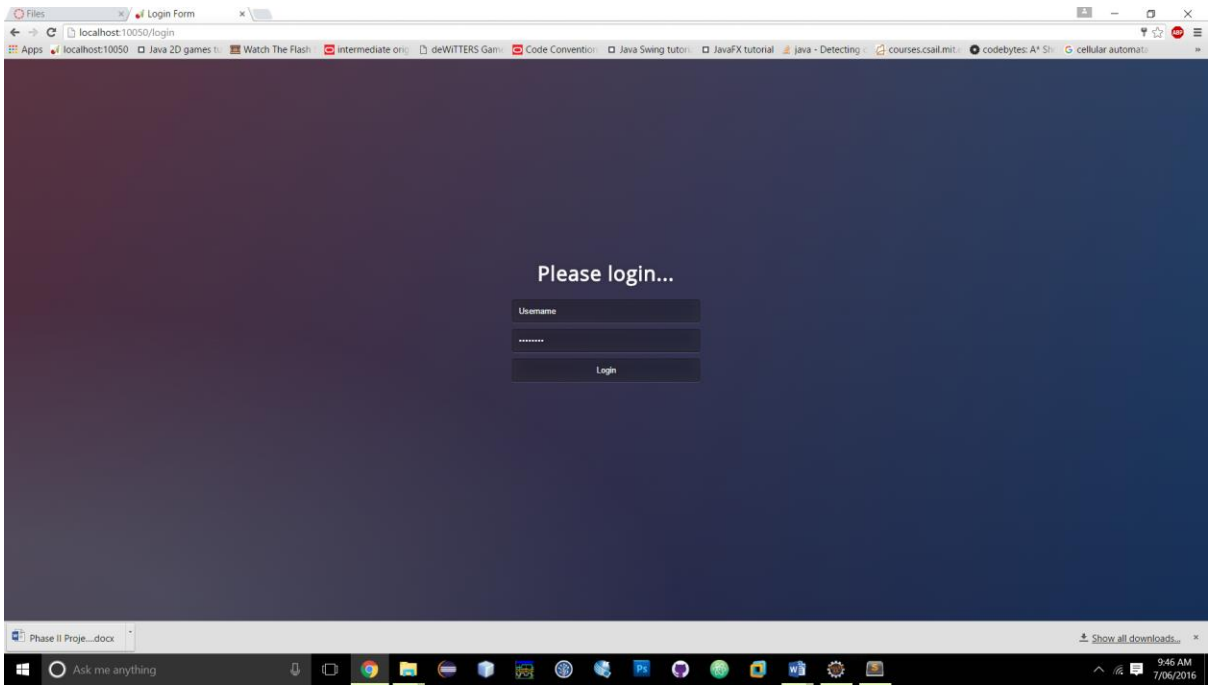


The client's main request was to provide for them a working online multiplayer game which their son can use to play against his friends. By ensuring that there is a secure server where his son and his friends can login into, view have put to rest the risk of large companies trying to steal his son's personal information. The server allow for him and his friends only to be visible to each other and communicate amongst themselves. It ensures security by hashing the password using SHA-256 algorithm for password encryption.

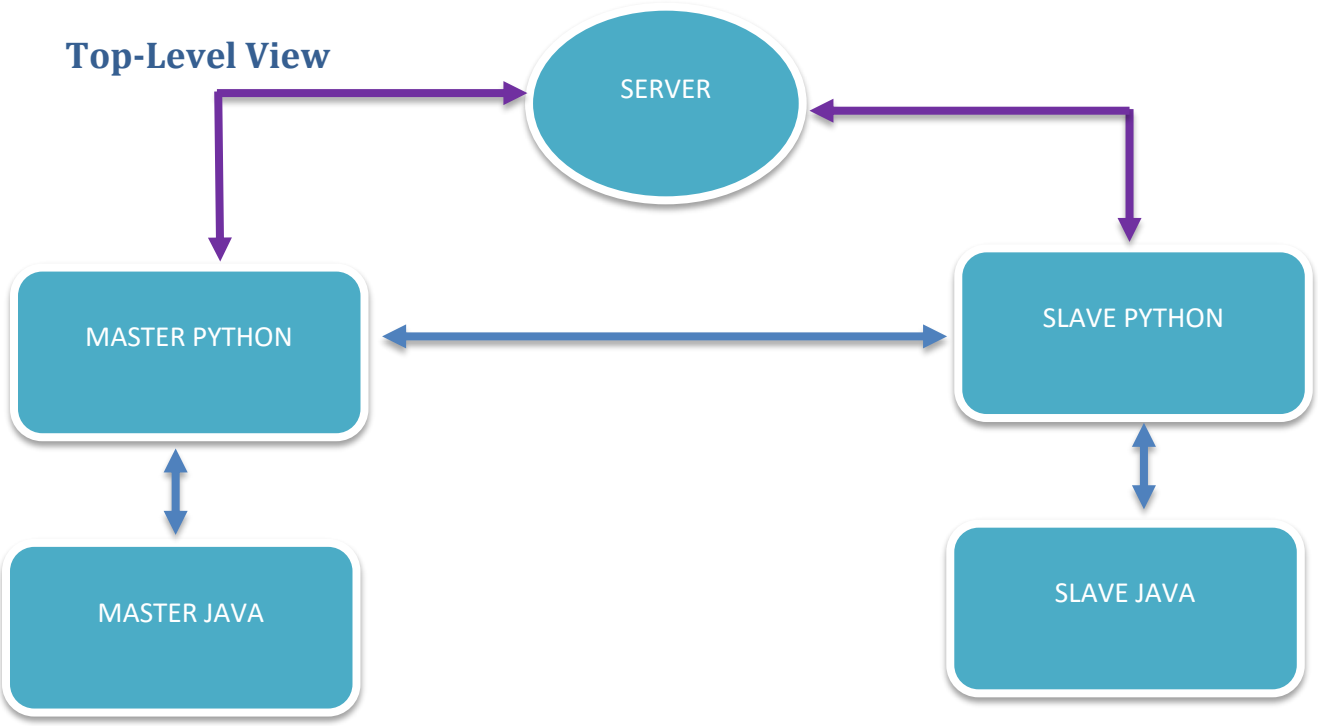
Our application allows the client's son to play against his friends keeping in mind who the master is and who the slave is. Upon the slave's cherrypy execution and challenge request is sent from a master cherrypy server, after starting up the slave side Java game, and then starting up the master side java game, the client's son and his friend can play against each other. Keeping in mind that they are on the same secure network.

The user interface we have designed is also impressive to look at for an eight year old, and fairly simple to use. The homepage has a simple play button, which when clicked, takes the user to a login screen. After successful sign in, there is a tool bar, for easy navigation to friends list, homepage, and challenge requests and also a button to sign out. The online players are a list going down the page, with buttons to challenge them and to add them as friends. Additionally, all their important details are written next to them if that information was required by the parent.

Safety is also a major concern to our client. We have also implemented encryption if our client was extra paranoid about his son's personal information. He can choose to login to server by passing an extra parameter to ensure extra protection of his son's details. This ensures that even if the data was being backtracked through some external ip, the thief would have no idea what it is due to the encryption.



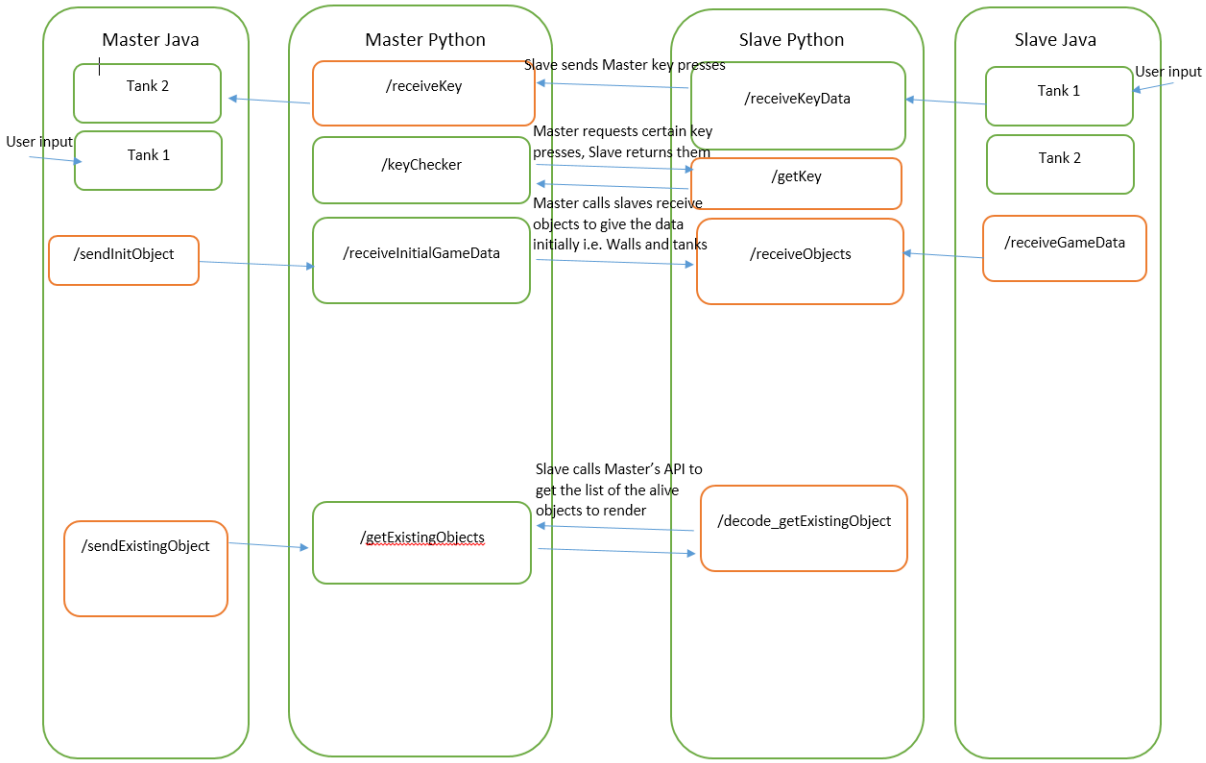
Top-Level View



This diagram shows the interactions between the different systems. Firstly, the user must log into the server and report their location. The server keeps track of all the online users who are available to connect with. When requested using /getList API the server returns the IP and port numbers of all the available users. This can be used to form a P2P network. Once the P2P network is formed the master and slave can send the required data over to play the multiplayer game.

There are two main applications running simultaneously, Java and Python. Python handles everything to do with the P2P network. Whereas the Java does all the computation based on the key presses which are passed down from the python game.

There are multiple API's implemented to pass data between the two Python games, a diagram below describes their data flow via sockets:



*Notes :

All the initial game data, i.e. walls and tanks are sent.

Socket Port 10071

All the alive game data, i.e. tanks, powerups, bullets are sent. NO WALLS

Socket Port 10071

Features

Some of the features we decided to implement was an auto refreshing list of players. This way the user is always up to date with who is online and who isn't, so that if they were to challenge someone, that request would actually go through to that user and the potential for a game starting up is higher.

Encryption is also implemented should the user decide to use this option to ensure extra security of his personal information.

We have also implemented a friends list so that our client's son can add his close friends and remove them at will. Upon exiting the application, the user is automatically logged out as it ensure his security that no one can start up a browser and play his game.

Our web design homepage and login screen are aesthetically very pleasing to the eye and they entice the user to want to use our application to play our game. We designed it to be as simple as possible but to also give the feel of a children's because ultimately it is them who will use this application.

Protocol

Initially we felt that the protocol was good enough for a project of this scale but it turned out that implementing some of the API's was a lot more difficult than we had anticipated. Developing API's like these was very difficult in the fact that to test just a single line of changed code on the slave java game, we would have to run the entire system and check for such a little thing to see if it had fixed the issue or not. This severely slowed down our development cycle and we were already short on time.

We also felt that some of the API's were a little redundant such as getKey() and getExistingObjects() as receivekey() should more or less be sufficient for the same purpose and existing objects is quite similar to getObjects() which again we could have used to get images for the slave to render. We personally felt that it would have been easier just to use one function rather than have one call the other.

Another protocol we thought should've been kept as constant was intra-app communication. We realise that it is a decision choice that we had to make, but we felt that if this was also kept constant, there may have been some higher potential of different groups' games interacting with each other.

Something we believe that should've been discussed more in depth in class, is how people would go about sending challenges to others and how they would accept or decline the proposal. I.e. if someone were to call someone's respond() api without having proposed a challenge a game may run when not intended to. This is less than ideal, and it isn't a perfect system, but there was still no proper discussing about how we would go about accepting challenges from others.

Overall we did feel that the majority of the API's were highly suitable to the p2p aspect of the tank game and they did serve their purpose well. In the future however, we believe that there should also be some api's to test other api's without having to run the entire system just to check for a minor code change.

Development Issues

One of the major issues that occurred during the development of this application was properly implementing intra-app connection. I.e. initiating proper communication between the cherrypy application and the java game. The approach we took was to implement sockets between the two applications to transfer data between them. This was extremely frustrating as we had to time the execution of our java games to start at a particular moment otherwise our sockets would start sending data to a port that wasn't open yet. Or we'd start listening for data on two of the same ports

and they'd cause errors. Eventually we found a way around this by creating multiple sockets and trying to start them at accurate times to start our games, however they're still less than ideal, and there is considerable lag on the slave side.

Another is that we didn't really understand what the parameters meant in the application protocol. We had assumed they were inputs to the functions and that we would call our own api's to initiate challenges. Hence we had coded for this. We only realised later on that the parameters are what needed to be JSON encoded in dictionary style with a key known to everyone so that they could receive data from us and send to us in the same format to create a uniform system. We had realised this quite late, so we had to refactor our code quite a lot, which ate up a lot of time.

Suitability of Tools

As we have mentioned in our first report, java is a very nice tool to develop more than aesthetically pleasing and functional 2d games. In terms of the project overall, java is a tool that can deliver a game not too complex for our client's son and not too dull that it is boring to play. Java is also cross-platform so regardless of the computer our client has, he'll be able to run our application. Again, as we have mentioned in our previous report, Java is not the best platform to develop extravagant games on, however for this type of role, it is ideal. One of the major advantages to java over other programming languages is that it can easily communicate with other applications and send and receive data through its easy to use socket class and functions. This made developing the application slightly easier, as some of the hard work was already done for us.

Developing our web server in the python language was also more than suitable for this role. Using the cherrypy webserver interface, meant that a lot of the difficult aspects of initiating a web server was abstracted away with simple function calls. We personally felt that java and cherrypy work quite well together as they are both object oriented design languages so it doesn't take too long to get used to cherrypy. One of the major advantages of using cherrypy is that it allowed us to code in the python language. Python is one of the most widely used languages in the world with a wealth of online support for almost any issue. This meant that development times were kept as short as possible, as help was easy to find based on our issues. Another positive that made python ideal was the fact that you can have a running console to check for syntax and simple code execution in the terminal itself. We found this feature particularly useful. One of the only complaints I have about python is that because it is an interpreted language, there are no semi-colons to indicate when to stop reading a line. It is all based around indenting which we weren't quite used to after working so much with other languages that require semi-colons.

Possible Improvements

One of the major improvements we feel is important, is how we have formatted our code. There is 3 different languages in one script. Python, Html and CSS. Next time we feel that it would be wiser to have all these 3 separate and just import the html and css files into the python module so that our code doesn't look too clattered and all over the place.

In terms of improving on the development side of things, if we were to do this project again and learn it from scratch again we would do a few things differently. Firstly we would have liked to focus less time on learning python, and more time on understanding what a web server is and how it works. Secondly what are API's and who calls which API's. And lastly and most importantly, how a client server system actually works. I believe it was our lack of knowledge in these aspects rather than our lack of knowledge of python that made this project quite challenging to grasp at times.

We also felt that as we developed the code, we had hardcoded a fair amount of code, like the ip address and port number for e.g. Ideally for future improvement, we will restrict ourselves from doing this as this masks some issues that may arise due to writing code with a sort of variable implementation.

One of the major improvements we felt that we could implement for the future is to have a scheduled plan of attack for projects such as these and try to stick to them as much as possible without missing deadlines and due dates. We felt we dived head first into this project, especially the second half, without really having much idea of what exactly was going on. We realised that it is very difficult to write code especially when you're not sure what you want your code to do. So one of the mitigations for this is to understand exactly, what the project requires of you before even starting to write any code, otherwise by the time you understand what is required, it is too late, and the client will not be too happy.