# Super Awesome KU Adventure Game

Authors: Zane Ralston and Jamie Robinson

GitHub repo: https://github.com/Cybermite/lab2

## Introduction:

The objective of the game is to pick-up the basketball and take it to Allen Fieldhouse. The objective of these labs was to allow for multiple users to be within the same world, to interact with each other, and to allow the server information to persist if it crashed or needed to restart.

## Non-interference:

In lab 3, we were tasked to allow multiple users to be in the same universe. The way we accomplished this was to create a cookie for each user on the server side and send it back to them. The contents of the cookie is a string version of the server's date and time appended to the current user count in the universe. Cookie-parser is a nice middle-ware used with express that allowed us to see the user's cookie easily on the server side. This allow us to make sure that the user always had a cookie for the userid and to make sure that the user was always in the server's system. Any time a get, put, or delete request are sent from the client, the server will always make sure the user has a valid userid within our system. The system would store these userids within a dictionary of users. Each user would have a separate userid, inventory, and location. This allow each user to be able to interact with the shared universe. This also added a simple form of interaction between the users. If they dropped an item within the same room, other users could view it. This naturally lead to more advance forms of interaction between users.

## Interaction:

Building off this simple form of interaction, we added the ability to view the other users within a room and steal items from other users' inventories. This lead to issues with keeping all the users synchronized with each other and the universe. The current system was not robust enough to handle these issues in real time. This lead to the discovery of express.io, which allowed us to use web sockets and rooms within express. The rooms allowed us to group clients by their location. The web sockets allowed us to have the clients send signals instantly back to the server. The server could then disperse the appropriate signal to the other clients within the same room from which the signal originated. This would allow the client to dynamically change the page depending on which signal is received. Since the clients were group within rooms, only the clients within the same room received the signal of a change in the environment. This eliminated the need for the clients to continuously pull information from the server every few seconds. This removed unnecessary web traffic, and provided clients with the most up-to-date information about the room and universe the second it was changed. The next task was to add persistence to the server to allow the server to restart without losing clients data.

### Persistence:

In lab 4, we added persistence to the server. We accomplished this by writing the users and campus JSON objects to separate files. We wrote to these files any time anything was changed within the universe. On start-up of the server, it attempts to read from these files. This lead to users hogging the basketball. We got around this by not allowing a user to have the basketball on start-up of the server. It would go back to Fraser Hall if a user had the basketball, and the Allen Fieldhouse victory text would be reset on start-up. This way the game's objective could still be accomplished by taking the basketball to Allen Fieldhouse.

### What we learned:

- Javascript: Neither of us had real experience with much javascript, so we learned javascript as we progressed in the project.
- Express: We learned about the basic web request of GET, PUT, and DELETE, and how to apply them to a web service.
- Cookies: We learned more about how cookies work. How to create them on both the client and server side. We ultimately used the server side version by using a middle-ware called Cookie-parser.
- Express.io: Express.io is a combination of Express and Socket.io. We learned how to have the server and clients dynamically communicate with each other by passing event signals when needed. This lead to signals being passed to all the clients rather than only the ones that needed to update their information. This lead us to learn about another option within Express.io. This option was join rooms. These rooms would be the user's current location. This allowed for the signals to be broadcast to only the clients within their room and not everyone on the server.
- Server and client sides: Neither of us knew much about the separation of the two. We learned about securing the server from malicious clients and preventing server crashes.