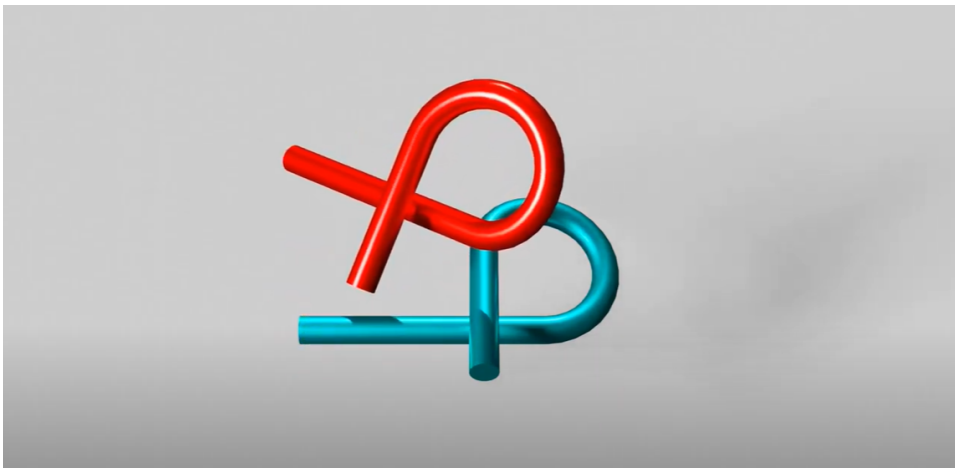
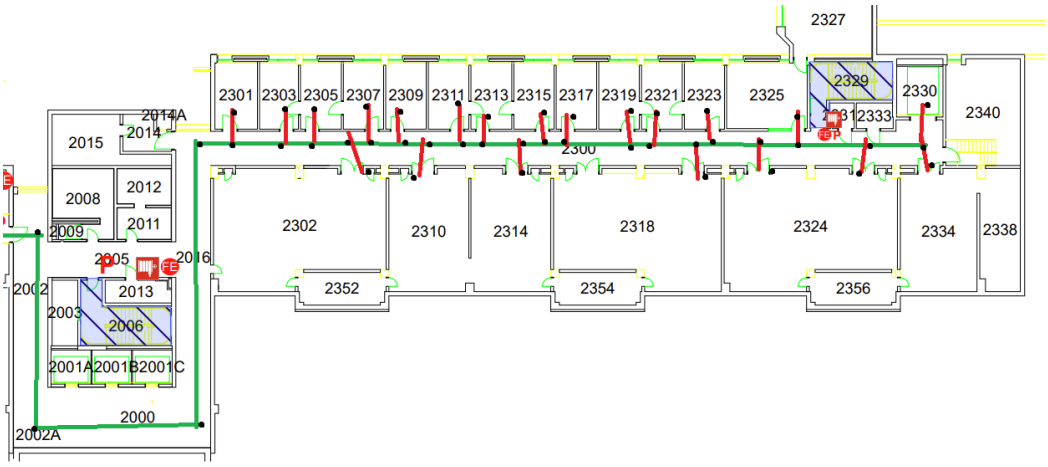
March 6, 2023 Meeting Notes

1. **3-dimensional puzzle solver** - [example video](#) → Maybe
   a. 
   b. User interface to control puzzle pieces
   c. Could have hard-coded solution for user if needed

2. **Arcade game (must port game from 1990's) --> MayBe**
   a. Run and shoot game

3. **Application for mapping buildings of CMU→ No--**
   a. 



b.

c. User enters their location and location they want to go to and application maps best route to get there

d. Application has three main components:

   i. An annotation software that allows the user to create a graph, that would be attached to the .png map. This graph is what is used to find routes

   ii. A path planning algorithm that searches along the graph to find the optimal path.

      1. Start with a simple code for this then optimize it, if we want. Can be as easy or difficult as we want. Below is an easy implementation:

         a. Create an array (double[num_nodes]) that records the distance from each node to the start node.

         b. Create a vector of all search nodes (intialize with the start node)

         c. For each search_node in search_nodes, calculate the distance to all nodes that neighbor search_node (search_nodes total distance + edge distance)

         d. If the new distance to the neighbor nodes is less than their saved distance, update with the lower distance, and add the neighbor node to new_search_nodes.

         e. Search_nodes = new_search_nodes

         f. Repeat until no new_search_nodes. (We now have a minimum distance to each node in the map from the original node)

         g. Walk back from end node to start node, going to the neighbor node with the smallest distance from the start.

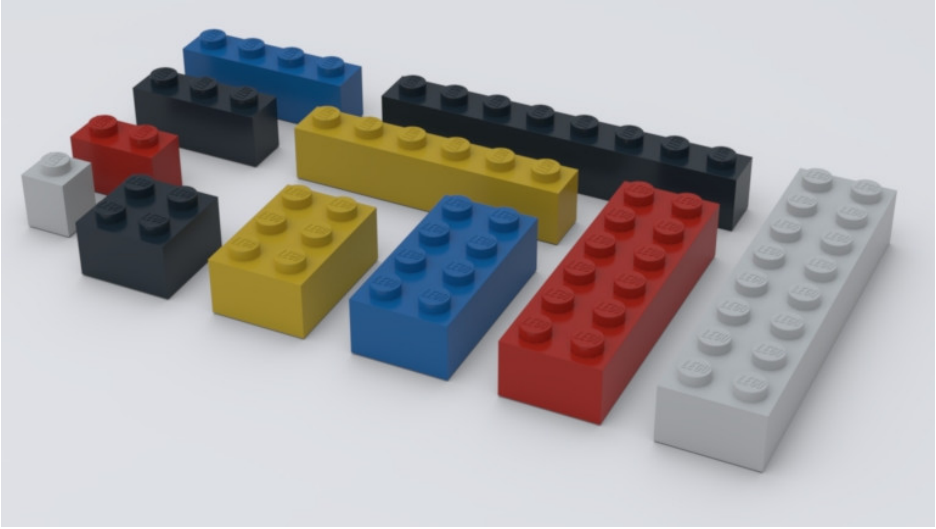
   iii. A GUI that shows the user an optimal path given a start and end location

e. Additional possiblesteps include:

i. Let the user load in different augmented maps, generate new augmented maps, and add to current augmented maps (add an extra floor, etc.) directly in the GUI
ii. Have an avatar that moves along the path (simple animation)
iii. Take pictures of key locations (such as the turn onto the NS bridge), so that the user can pull up images of landmarks on the map.

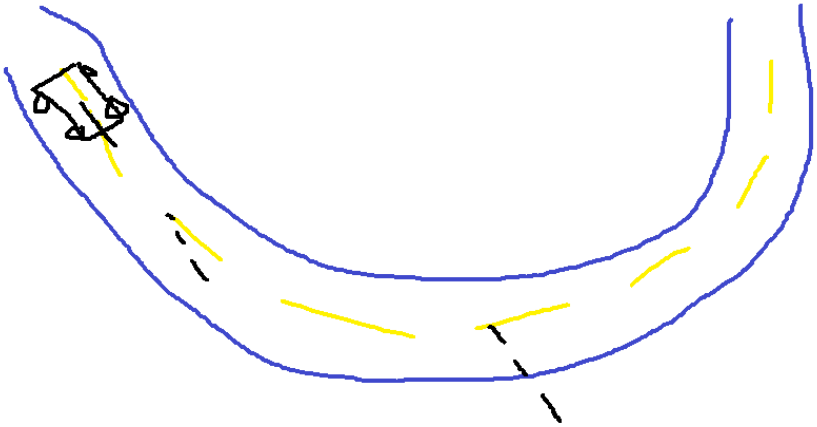
4. **Arduino powered car for something more hardware related - No**

5. **Lego slicer→ GOOD**
   a. Given a 3d object, how would I make this out of legos?
   b. Voxelization
   c. Like how a 3d printer takes 3d object and converts it into 2d slices, our program would convert an input .stl
   d. To increase scope, have the app run from a GUI
   e. Main components:
      i. GUI: allow user to load in STL, and choose settings including what possible bricks to use
      ii. Slicer: First voxelize the STL, then pick blocks to fill up the voxel space.
      iii. 3D viewer: Render the assembled structure. Allow the user to change their view and look at it from other angles.
   f. Additional possible components
      i. Structural integrity
   g. https://printablebricks.com/ ← library of .stl's with popular bricks (for people who want to 3d print their own LEGO's)
   h. https://grabcad.com/library/classic-lego-brick-1 ← more bricks on GrabCAD



6. **Learning to drive using Neural Networks and Genetic Algorithms→ No**
   a. Enviroment: Car, road.
   b. State:
      i. Current speed.

ii. Distance from center of car to center of road (how far to the left/right the car is from the center line).
iii. angle between the car and the road at the car (how far off from straight the car is).
iv. distance between the car and the center line X1 distance ahead of the car (how much you'll need to turn for a future curve). (small scale)
v. distance between the car and the center line X2 distance ahead of the car (how much you'll need to turn for a future curve). (large scale)



c.
d. Have a small NN, State inputs (5ish) to two outputs (steering wheel and acceleration).
e. Train weights using genetic algorithm.
    i. Intialize N networks, random weights.
    ii. Test them all. Keep the highest scoring ones (~25%)
    iii. Generate N new networks, by altering the weights from the survivors. (Just add some noise). If 25% survive, each survivor has 4 children.
    iv. Repeat untill the car drives good.
f. Components:
    i. Simulation environment. Originally this can be a simple, hardcoded track (just need the center line, represented as a collection of points), but eventually we'd want to have it randomly generated?
        1. Also need to implement basic bicycle style mechanics for a car. (Integrate acceleration to get speed, use the steering angle to determine direction of motion)
    ii. NN implementation: Create a function that takes in the weights and does the NN inference. This should be pretty straightforward, since we're not doing any backprop.
    iii. Genetic algorithm implementation. Given the performance of the previous generation, generate the new generation
        1. Originally can just use direct descendants, adding noise to the weights

2. Eventually may want to implement Cross Entropy Method (sample NN weights from a gaussian distribution)

iv. GUI: Make a nice rendering, showing off the training. Allow the user to visualize the performance of different generations, etc.