Project Plan [20 pts]
Write up a proposal file (in the file proposal.txt, or .docx, or .pdf) which should include the following components:

Project Description [5 pts]: The name of the term project and a short description of what it will be.

Dynamic Defense: A unique tower defense with simple (nonexistent) graphics that allows buildings to be placed open-map and enemies to pathfind freely across the placed buildings to reach the destination. Refer to the classics bloons td and fieldrunners for reference.

Structural Plan [5 pts]: A structural plan for how the finalized project will be organized in different functions, files and/or objects.

<u>What I currently have:</u>
main.py - main file for scene rendering, most game logic, main grid for building placement, etc
geometry.py - a helper file I made for rendering rectangles (as canvas polygons) that can rotate, be set as child of other rectangles, look towards a point, etc etc
ui.py - a helper file for my buttons, text, and other ui
towers.py - stores all tower classes
levels.py - stores all levels
enemy.py - stores enemy properties

My structure is pretty complete, save for any additional map editor files/save files I may add after hitting MVP with the pickle module.

Note: no images are used, because I tried using images with alpha and anything >200x200px significantly dropped the framerate (a single 500x500 image with alpha values lowered the fps from ~50 to ~20)

Algorithmic Plan [5 pts]: An algorithmic plan for how you will approach the trickiest part of the project. Be sure to clearly highlight which part(s) of your project are most complex.

- Pathfinding: what's tricky about this pathfinding is that I have to make sure all building placements are legal, e.g. I can't place a building if any existing enemy paths get completely blocked off. Thus, I had to pathfind every frame for every enemy when checking building placement. I wrote the script myself using A* pathfinding techniques as a guideline.
    - Enemy paths are recalculated upon any building change, and since they don't teleport to a certain grid they must "slide" towards it. This means adding interpolation between two points.
- Geometry.py rotation/parent child: in order to rotate points, a LOT of trig was used. The same goes for moving in an object's local position (when rotated, forward is different), setting a child to pivot with the parent when rotating, etc.
- Complex OOP structure: towers inherit from common parents, ui inherits from common parents, all classes have to have their own render function, etc
- Layering system: when rendering, I made a system so that creating a rectangle class in appstarted() means I don't have to deal with redrawall() at all. This means I designate a layer

for the object at the beginning and lower layers always must be rendered first. UI also renders on top of all geometry objects.

Version Control Plan [5 pts]: A short description **and image** demonstrating how you are using version control to back up your code. Notes:

I use Github (Desktop) to organize my CMU files. The repo is currently private and will be public after the course is all over and graded.