Program:	**CPA2**
Course:	**INFO3135 "Data Structures & Algorithms", Fall 2020**
Professors:	**Michael Feeney**
Project # 2:	**Cities connected by roads.**
Weight:	**15% of your final mark**
Due Date:	**Saturday, December 19th, @ 11:59 PM**

Description and Purpose

You are to write an application that will take some map & terrain data, store this information as a graph and/or tree structure, and use that to answer specific questions involving navigating from place to place.

In the 2nd ½ of INFO-3135, we focused on trees, hashes, and graphs, which are useful in this project.

Your classes will be added to a Win32 (windows API) 64-bit application, running in both Debug and Release mode, on Windows 10, which will "exercise" ("call") these various methods on these classes. i.e. be sure that your classes will compile and link when placed into a 64-bit Win32 console application.

You will submit:

- Your amazing **Visual Studio solution.**
- A **short video highlighting your code** (more details later in the document)

There's seven (7) parts (questions?) to this project, with the marks indicated.

The "helper program" is in the `INFO3135_Project2HelperProgram.7z` file.

Details

In the theme of Mike Acton's "3 big lies", this project centres about The Data (Lie #3: "Code is more important than data"), this project centres around some input and output data.

In project #1 (and the mid-term), there was a large focus on the API, in the sense that you needed to match the function/method signatures, class names, and how data was passed/created (passing copies of pointers or copies of instances).

While most of you did great with this, a large number *changed* the API, effectively.

That's a big "no no" with any sort of software development. I mean, beyond having "the thing work properly", it might be *the* biggest "rule": *Use* the API as is, and *don't* change it.

But the API, particularly with all sorts of web stuff (JSON, etc.), can often be the structure of the data as well.

With this project, there *won't* be certain classes, functions, methods, whatever you have to match, but you *will* have to match the format of the data you accept *and* the data you generate.

So the *data* is "the API" this time. And... be careful **not** to change it.

This will also help in a few other ways:

- It will help me mark it: more obvious when you are "close" vs. "completely clueless".
- Easier for *you* to see what you've done and why.
- It gives you more freedom and creativity with how you write your code (maybe?).

Using MeshLab (https://www.meshlab.net/), I've generated a number of "terrains" (specifically, the "hybrid multifractal terrain", with scale of 2.0, various "seed" values, and the rest defaults. If you're curious how this terrain is generated, you read the book it's based on, or ask your friendly professor who's doing his Masters on stuff directly related to stuff and could bore you silly with *"oh please make him shut up"* [1] details).

You do ***not*** have to generate the terrains or really do anything with MeshLab other than look at the output your tool, and my code, generates (more on this later).

You will use the Project2HelperProgram (from FOL) to generate and test your project.

This program will take various files from you, generate the test data, verify what you've produced, and help you visualize your solution.

Briefly, here's the steps:

1. Using your student number, you will choose three (3) "terrain" files, passing them to the program.
2. It will generate a "TerrainDataInput.txt" file that your program will load and manipulate.
3. Some of the output of your program can also be passed to this program, which will generate a 3D mesh file that MeshLab can display, to help you see if your program output is correct.

   Note it doesn't "mark it", per se, but if this program doesn't work, then you won't get 100% (i.e. you can certainly get marks for certain questions, but it will only test *some* of your output).

[1] My parent's and grown children's faces when I start talking about my topic ☺

Even more details (what you actually have to do):

1. **(10 marks):** Divide the last six (6) digits of your student number into three pairs of numbers.

   For example, if your SN is **8506394**, then this would be "50", "63", and "94".

   If your SN has zeros, then include these. i.e. 0**0**2**0**4**0**6 would give "2", "4", and "6" as the three numbers.

   "Hash" these numbers with 26: i.e. the integer modulus/remainder, mod(), or "%" operator.

   (I picked 26 as there's 26 letters in the English alphabet – I had some other idea about these terrain models and names, but it didn't work out, but that's "why 26?")

   Use these numbers to pick the three (3) "terrain" files you'll need.

   So 8506394 gives you:

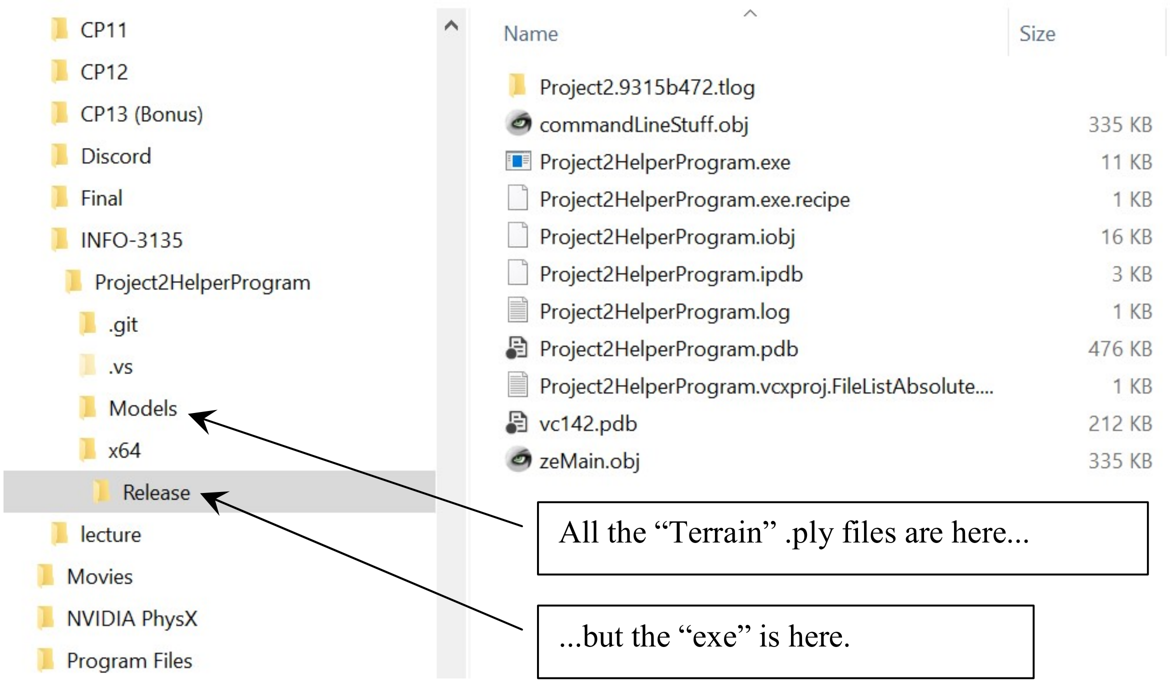
Digits	**Hash value**	**"Terrain" file you'll use**
50	50 % 26 = **24**	Terrain_seed_**24**_W.ply
63	63 % 26 = **11**	Terrain_seed_**11**_J.ply
94	94 % 26 = **16**.	Terrain_seed_**16**_O.ply

   There's nothing to "submit" for this question, but I'll take note of what files you've used, and assign those marks to this question.

   WARNING: Be SURE to use these specific files in the later questions, though, since I'll be using them for marking, and if you use different ones, you'll get different results (i.e. I'll assume your answers a wrong).

2. (10 marks): Build the Project2HelperProgram program, then run it by passing your student number (as text) and these three file names as command line parameters. This will generate a TerrainDataInputXXXX.txt file, where the "XXXX" is your student number.

   Note: Keep in mind that programs assume where they are run is the "working directory", so you will have to place the "exe" file into the same folder that has the "Models" folder (which has the "ply" files).



Name	Size
Project2.9315b472.tlog	
commandLineStuff.obj	335 KB
Project2HelperProgram.exe	11 KB
Project2HelperProgram.exe.recipe	1 KB
Project2HelperProgram.iobj	16 KB
Project2HelperProgram.ipdb	3 KB
Project2HelperProgram.log	1 KB
Project2HelperProgram.pdb	476 KB
Project2HelperProgram.vcxproj.FileListAbsolute....	1 KB
vc142.pdb	212 KB
zeMain.obj	335 KB

All the "Terrain" .ply files are here...

...but the "exe" is here.

In other words, the "exe" and "Models" folder have to be in the same folder, so you have to either move the "Models" folder into the folder with the "exe" or move the exe into the "Models" folder.

Life is all about choices, so you get to make one! Hazzah!

It doesn't matter (well, it shouldn't...) if it's debug or release or which "bit-ness" it is, but I did all my testing on 64 bit Release.

For the student above, you would type in (all on one line, in the CMD/Command prompt):

```
Project2HelperProgram.exe 8506394 Terrain_seed_24_W.ply
Terrain_seed_11_J.ply Terrain_seed_16_O.ply
```

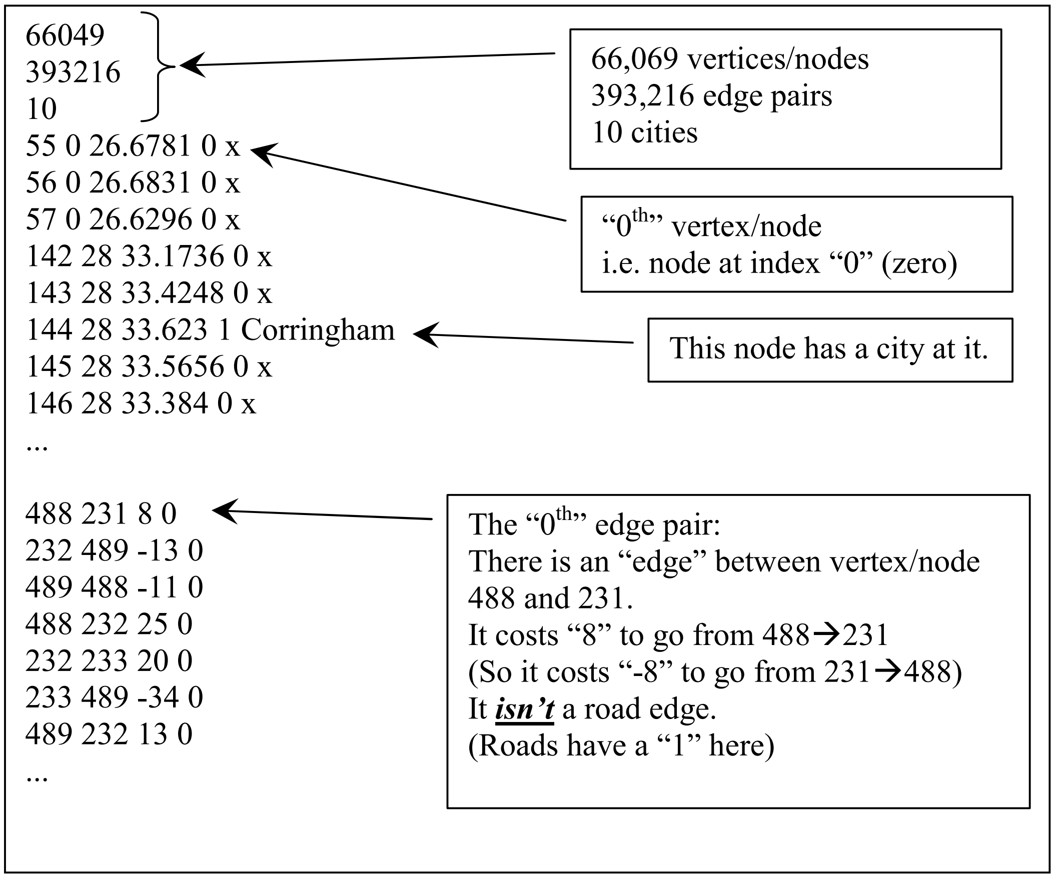
This would spit out a file called **`TerrainDataInput8506394.txt`**.

The file that's generated represents a graph with map information that you'll have to work with.

It's in the following format:

- Number of Vertices/Nodes, which is an unsigned integer.
- Number of edge pairs, which is an unsigned integer.
- Number of cities, which is an unsigned integer.
- A list of vertices, starting with index 0 (zero), in this format:
  - X location (float)
  - Y location (float)
  - Height or Z (float)
  - Is there a city here, which is either a "0" (false) or "1" (true)
  - The name of the city (which is one word, no spaces), or the letter "x"
- A list of edges, starting with index 0 (zero), in this format:
  - Start vertex/node ID, which is an unsigned integer.
  - End vertex/node ID, which is an unsigned integer.
  - "Cost" to go from one node/vertex to the other, an *signed* integer.
  - Is a road, which is "0" (false) or "1" (true)

Here's (part of) an example file:



```
66049
393216
10
55 0 26.6781 0 x
56 0 26.6831 0 x
57 0 26.6296 0 x
142 28 33.1736 0 x
143 28 33.4248 0 x
144 28 33.623 1 Corringham
145 28 33.5656 0 x
146 28 33.384 0 x
...

488 231 8 0
232 489 -13 0
489 488 -11 0
488 232 25 0
232 233 20 0
233 489 -34 0
489 232 13 0
...
```

66,069 vertices/nodes
393,216 edge pairs
10 cities

"0th" vertex/node
i.e. node at index "0" (zero)

This node has a city at it.

The "0th" edge pair:
There is an "edge" between vertex/node 488 and 231.
It costs "8" to go from 488→231
(So it costs "-8" to go from 231→488)
It ***<u>isn't</u>*** a road edge.
(Roads have a "1" here)

Some things to note about this graph representation (in the file):

- The graph is ***not*** directed, so each edge is bidirectional.
- In other words, you can go in either direction.
- BUT, each edge is only listed once.
- The "cost" is in the order listed by the nodes, so:
    - If the 1st three numbers are: 234 292 17
    - You can go from node/vertex 234 to 292, but it "costs" 17
    - You can go from 292 to 234 but it's the reverse, so "costs" ***-****17*.
- The graph is a grid, basically like the terrain that MeshLab spits out. It is:
    - It's a bunch of points, representing points on a terrain.
    - It's on the X/Y axis, with one corner at 0,0 going positive.
    - The coordinates range from 0.0 to 256.0 on the X & Y axis.
    - The coordinates range from 0.0 to 128.0 on the Z/height axis.
- Some of the vertices/nodes have cities on them.

3. (50 marks) Print out the names and locations of the cities. Specifically:

    - Scan through the vertices, and if it's a city, print out the name and the location.
    - That's pretty much it.

4. (100 marks) Print out the TerrainDataInputXXXX.txt file in the exact same format that you read it in.
   Why? Because you'll need this to visualize your results later.
   You can use a bunch of tools to see if this is correct:

    - Windows "fc" (for "file compare"): https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/fc

      Pro Tip: Use the "ASCII comparison", "binary". It's the default, anyway
    - Notepad++ compare plugin: https://sourceforge.net/projects/npp-compare/
    - There's a bunch of other ways to do this: https://thegeekpage.com/12-best-free-file-comparison-tools-for-windows-10/ and I'm sure your classmates have other suggestions if you don't like these.

5. (100 marks) Find the city closest to the origin and the one farthest from the origin, in the following way:
   - You can steal the static getDistance() method on the cMesh::sVertex class. It's pretty much the same as the 2D version you had for the mid-term, but it's in "fancy 3D"!

     Essentially, you pass two 3D coordinates and it returns a float, indicating distance.

   - To find the city "closest" to the origin, go through all the cities, one by one, comparing their location with the location/vertex/node at 0,0,0.
   - To find the city "farthest" to the origin, to the same thing, but check for the *farthest* from the origin.
   - Hint: When searching, make the assumption that the 1st city in your "list of cities" is the correct one, then compare with that one (is it closer or farther?), then update that.

   - Print out the names and locations of the cities, indicating which they are (closest or nearest).

6. (200 marks) Build a "road" between those two cities (in question 4). Here's how you do that:
   - Starting from the first city, pick the "cheapest" vertex/node that goes towards the other city.
   - From that node, look at all the edges associated with that vertex/node to determine which vertex/nodes you can reach.
   - Firstly, you only want to go "towards" the other city, not "away".
     i.e. no matter what the "cost", you *don't* want to go "away" from the destination city (vertex/node).
   - You can test this using the same technique as question 4: compare the vertex/node you're on with some candidate vertices/nodes (you can even test *all* of the connecting vertices/nodes).

     If the node goes "away" – i.e. it would move you *further* than where you are, ignore it.

     Of the nodes that move *towards*, pick the lowest cost edge.

     Note: Keep in mind that the edges list cost in the order they are listed, so if you are going "the other way", then the cost is reversed.
   - Keep track of this overall cost (by keeping a running total of the costs).
   - As you go through this, keep a list of "edges" by index.
     Remember that the edges are listed in the TerrainDataInputXXXX.txt file are listed from 0 to n.

     You will need to keep track of these indices.
   - Keep going until you reach the other city.
   - What you will end up with is:
     i. A list of indices, indicating the edges (i..e. index of the edges, as listed in the file)
     ii. A total cost
   - Note: It's almost certain that the road will "wind" as it tries to find the cheapest local path.
     What you are doing is a form of "greedy algorithm", where it takes the best, immediate option, rather than considering the overall cost.
   - **Update** and write to a TerrainDataInputXXXX.txt file (with a different name, like don't overwrite the original file).

     Specifically, update any edge so that "is road" is now "1" instead of "0".

If you want to visualize your amazing road, you can the Project2HelperProgram program can take your TerrainDataInputXXXX.txt and generate a mesh that you can see in MeshLab (like you don't have to do anything – it'll just "appear").

This is what the "is road" value in the edge is for. If it's a "1", it will colour the mesh with black/brown (the rest of the mesh will be green). The city nodes will have a city model hovering above them.

7. (250 marks): Continue building roads to connect the rest of the cities.

   Do this in the following manner:

   - 1st, you want to connect the two (2) cites from question 5 to their nearest neighbours.
   - From each of these cities, pick the two closest cities.

     It's unlikely that one of your "picks" is the city in question 5, but in case you do, pick a different city (since you already have a road, right?)
   - Build roads towards them in the same manner you did in question 5.

   At this point, six (6) cities should be connected by five (5) roads, with four (4) cities left (with no roads).

   - Repeat this process with the four (4) remaining cites that *don't* yet have roads (i.e. pick the "closest" two cities and build roads towards them).

   IF there are any cities that only have one (1) road:

   - Build a road to the nearest city that <u>*doesn't*</u> have a road *to this city.* (the one with only one road)
   - Repeat this until all cites have at least two (2) roads.

   Check to see that all cities can reach all other cities. You can do this in a number of ways, starting at one city and performing (I'd suggest making a function/method that does this, but you do you):

   - Breadth first traversal, noting which cities have been "visited", OR...
   - Depth first traversal, noting which cities have been "visited", OR...
   - Randomly jumping from vertex/node to vertex/node, randomly, like a million times or something, and if it's the 1st time there, noting that the city has been "visited", THEN...
     i. See if there are any "unvisited" cities. If there *aren't*, then you're good.
   - "Collapsing" the "cities connected by roads" graph, eliminating cities that have been visited. When you're done, if you have more than one city, then you *can't* get to all the cities.
   - Some other, clever way, that you've come up with, but you <u>***can't***</u> just "*connect all the cities together with a ton of roads*". That's <u>not</u> the solution. You *might* have three (3), possibly four (4) roads, but this process should connect all but the "farthest" cites with only two (2) roads.
     (Note that the roads might "cross" each other, but don't worry about that.)

   Keep repeating this until:

   - All cites are connected <u>AND</u>
   - They have at least two (2) roads

   Like Question 6, **Update and save out your TerrainDataInputXXXX**.txt file by placing a "1" on any edge that now has a road (again, output to a different filename name, so you don't overwrite the original file).

That's it. That's all the questions.

What you submit and how I'll mark it:

- **Your entire Visual Studio solution** (**PLEASE** remove the "extra" files from it, making it smaller), and compress it.

- **A short recorded video** *briefly* pointing out where you've done what and with what (Example: "Here's where I did question 3, where I used a combination of a hashing function, a smart array, and the power of human kindness...")
  - **I only need to see your screen and hear your amazing "radio" voice.**
  - If can be any format, as long as it plays in the VLC player (for Windows/Linux).
  - **This is non-negotiable.** While I heard some people say they don't have a web cam (which you are required to have, anyway) there's no way you don't have a microphone and/or can't record your screen. And you can also record your screen from your phone, if needed.
  - **TO BE CLEAR:** I ***don't*** want an image from your webcam – I want a recording of ONLY your screen, showing visual studio, with your code in it.
- **If you are missing either of these, your submission won't be marked and you will receive a mark of zero.**

- Any additional notes you think I need to mark this.

- I have another program that takes your information, solves the problem, and compares it with what your program calculates/outputs.

  - I'll be using the same input files you had (the "terrain" files).
  - I'll use the same output files you generate.
  - I'll scan through your code looking for:
    - Violations in what I've asked you *not* to do (auto, any STL containers/algorithms, damn boost, etc.)
    - Seeing that the containers you're using are actually yours.
  - I'll be running your program on a Dell Precision workstation with a Xeon E-2186M (6 core) 2.9-4.8GHz CPU, 64 G RAM, and an ASUS GTX 1080, with all your code and data files running off the five (5) disk 1 TB WD Blue SSDs RAID 0 array swap/temp drive, in case you care.

Internal data structures/code restrictions:

- You can ***NOT*** use the "`auto`" keyword (nor can you use a #define/typedef to circumvent this). I will use find and replace to change "auto" to "HelloKitty", then try to recompile. If it doesn't compile/build, then you get a mark of zero. **This is absolutely non-negotiable.**

- You can ***NOT*** use *any* variant of the STL **containers** or **algorithm** libraries, boost, or any other library (standard or otherwise), other than your own. In other words:
    - You **can** use any STL *streaming* (iostream, fstream, stringstream, etc.) or *string* libraries, as well as the C++ 11 random library (*though regular rand() is completely fine for this project*).
        - **If there is another *standard* library you'd like to use, please check with me first.**
        - **Keep in mind that the main intent of this course is for you to make your own data structures and algorithms, so if it's a library that's really a "data structure" and/or basic algorithm, my answer is going to me "no".**
    - You can ***NOT*** use any of the STL vector, list, hash, map, (container) libraries.
    - You can ***NOT*** use any STL "algorithm" (or equivalent) libraries.
    - You can ***NOT*** use boost. I don't have it installed, so if you've got it in there, it won't build, and if it won't build, then you get zero. **This is absolutely non-negotiable.**

- Any "container" data structure must be *your own* (smart array, linked lists, maps, hashes, trees, graphs, whatever). If you are unsure, then ask yourself "did I write this?", and if the answer isn't an unequivocal "yes!", then you *can't* use it.
    - In your video, point out where these data structures are being used.
    - You **can** use any code that has been provided to you *by me in this course*.
    - You have to demonstrate *both* of these structures being used. In other words, you *can't* ONLY use a smart array for everything.
    - You can use any combination of data structure you'd like, including something you've come up with yourself. You do ***not*** have to use *all* the data structures you've see in INFO-3135.

Requirements of your data structures:

- There are none. “Go to town” with your creativity.
- There are no requirements for doing any template stuff, although you might find it useful, maybe?

  Honestly, I can’t really see how, but I’m that’s just me.

  I’d *strongly* suggest that you *don’t* immediately start “templating” everything, as it’s just going to make things much, much more difficult, and for not much of a pay off.

- There’s no “performance” requirement, unless things get ridiculous – like if your program is still running after minutes, I’m might conclude it’s locked in an endless loop...

- I don’t care about comments, formatting, or any other pointless trivialities.

  These are certainly important, as you have to meet whatever “code style” that your company/contract dictates, but it’s usually not hard to do that, and lots of places honestly don’t care, or simple “peer pressure” and/or mimicry sorts this out in the end.

  IMHO, when stuff like this is marked/required – unless it’s 1st term – it’s almost always because the instructor is going for “low hanging fruit” for marking; marking “formatting” or “code conventions” is an easy thing to do, even if it doesn’t make you a “better” programmer.

  On caveat to that is one student put “**`F___ you, Feeney`**” in the comments, and I laughed... then wrote up a code of conduct violation, and a mark of zero.

  (If you think that’s harsh, imagine if I had been their boss and/or hiring them for a contract, and then imaging how that might have played out in *that* situation.)

  I care that it works, mainly.

  Just “get it done”.

  You’re welcome. Welcome to what happens “in industry”.

Project Corrections

If any corrections or changes are necessary they will be posted to the course web site and you will be notified of any changes in class. It is your responsibility to check the site periodically for changes to the project. Additional resources relating to the project may also be posted.

75/10-year old "squinty eye" plagiarism test:

I have very little tolerance for plagiarism, but many students are unclear about what it is.

Basically, it's submitting somebody else's work as your own.

There is sometimes some confusion over this because you could argue nothing is actually "unique" (see: http://everythingisaremix.info/ for a fascinating overview of this).

The whole point of assignments/tests/projects in this course (or any course, really) is to try to see if you are actually able to ***do*** the coding that's asked of you. In other words: How competent are you? Handing me someone else's code and/or making a trivial change isn't good enough.

Also, it's illegal:

- http://www.plagiarism.org/ask-the-experts/faq/
- http://definitions.uslegal.com/p/plagiarism/
- http://en.wikipedia.org/wiki/Plagiarism
- https://www.legalzoom.com/articles/plagiarism-what-is-it-exactly

In other words, I'm not going to be drawn into a giant debate over how "different" your code is from mine or anyone else's, if any sensible person (including me) would conclude that the code/application is pretty much the same thing, then it is. It is up to my discretion to decide this.

- While you may freely "borrow" mine (or anyone other) code ***but*** your code should be "sufficiently" different from mine (you might want to replace the word "sufficiently" with "significantly").
- In other words, you *cannot* simply use an existing game engine (or part of a game engine) to complete this assignment; it should be either completely new of **significantly** modified.
- How will I determine this?
    - If I showed your application and/or your source code to either my pragmatic 83-year-old mother, or a typical 10-year-old, or even some random person walking down the hallway (i.e. a non-expert), and they looked at it, tilted their heads, squinted their eyes, and said "you know, they look the same," then they ***are*** the same.
    - Another test would: How much time it would take for a "competent programmer" (for example, *me*) to make the changes you are submitting? The point here is that I don't "care" if you tell me "But it took me *weeks* to make the changes!" Fine, but if I can make those same changes in 10 minutes, then not a lot of work has been done (certainly **not** sufficient work – these projects should show take **days** of work having been done).