# **INFO6023 – Game Algorithms & Gems – Mid-term – Winter 2025**

Instructor: Michael Feeney

## The exam format:

* You may use any resources you feel are necessary to complete the exam, but you are to answer the questions **on your own, individually**. I will be looking for plagiarism (i.e. copying) very carefully. There is *no possible way* that the specific code to answer these questions, or the output to the screen, would be very similar to the look of another student’s code. Remember, this is a test and there are very clear policies about cheating on tests.
* You may also ***not*** use code that’s taken *mostly* from some existing framework (like downloading the code from the OpenGL book, Learn OpenGL, Open Scene Graph, or something like that)  
  + <http://www.fanshawec.ca/admissions/registrars-office/policies/cheating-policy>
  + <http://www.fanshawec.ca/sites/default/files/assets/Ombuds/cheating_flowchart.pdf>

How can I make the determination that it’s not “your” code? Simple: If I suspect it’s not yours then I’ll ask you to “defend” it one-on-one where I’ll ask you questions and see if you know what you’re doing or why the code is the way it is. If you have no idea what’s going on, then it’s pretty certain you didn’t write it yourself. You *can* use any code *provided in class* or *you* wrote since starting this program *this September 2024.*

* You ***may*** use simple utility libraries like assimp, loadPNG, JSON/XML loaders, sound, etc. **No** boost, though.
* You may **not** use the auto keyword. Doing so will give you a mark of zero.
* You may **not** use generative AI tools to create any of the code for this exam (“you” didn’t write it).
* It is an “open” exam. You have access to anything in any book, internet resource, or anything on your computer, or that has been uploaded in class, including projects you’ve already completed.
* The questions are of equal weight. The exam has **six** **(6)** questions and **ten (10)** pages.
* The questions build on each other so you can put them in one solution/project.  
  If you feel you need further clarification, please include a readme file (and a video if you’d like, though not required).
* ***PLEASE*** delete any temporary files that Visual Studio generates (to reduce the upload size)
* Do ***NOT*** do some clever “*oh, you just have to comment/uncomment this block of code*” nonsense. However, if the questions ***CLEARLY AND OBVIOUSLY*** build on each other, you may combine them (like if one question places objects, then the next one moves objects around with the keys) – even so, **MAKE IT 100% CLEAR** to me what questions the solution is attempting to answer. **I do NOT want to edit the code in any way.**
* For applications: if it doesn’t build and run, *it’s like you didn’t answer it*. I’ll correct trivial, obvious problems (like you clearly missed a semicolon, etc.), but you need to be sure that it compiles and/or runs.
* You have until **11:59 PM** on **Friday, February 28th** to submit all your files to the drop box on Fanshawe Online.   
  This is an exam, not a project, so please submit by the deadline.
* Unless otherwise indicated, all these solutions assume that you are creating/using a C++ project using Visual Studio 2022 using the OpenGL 4.x API (with GFLW, glad, and glm).
* I will be building using the default Visual C++ settings (C++ 14).
* I’ll be in the classroom during the usual class time, and outside that I’ll check e-mail and discord.

## For all questions:

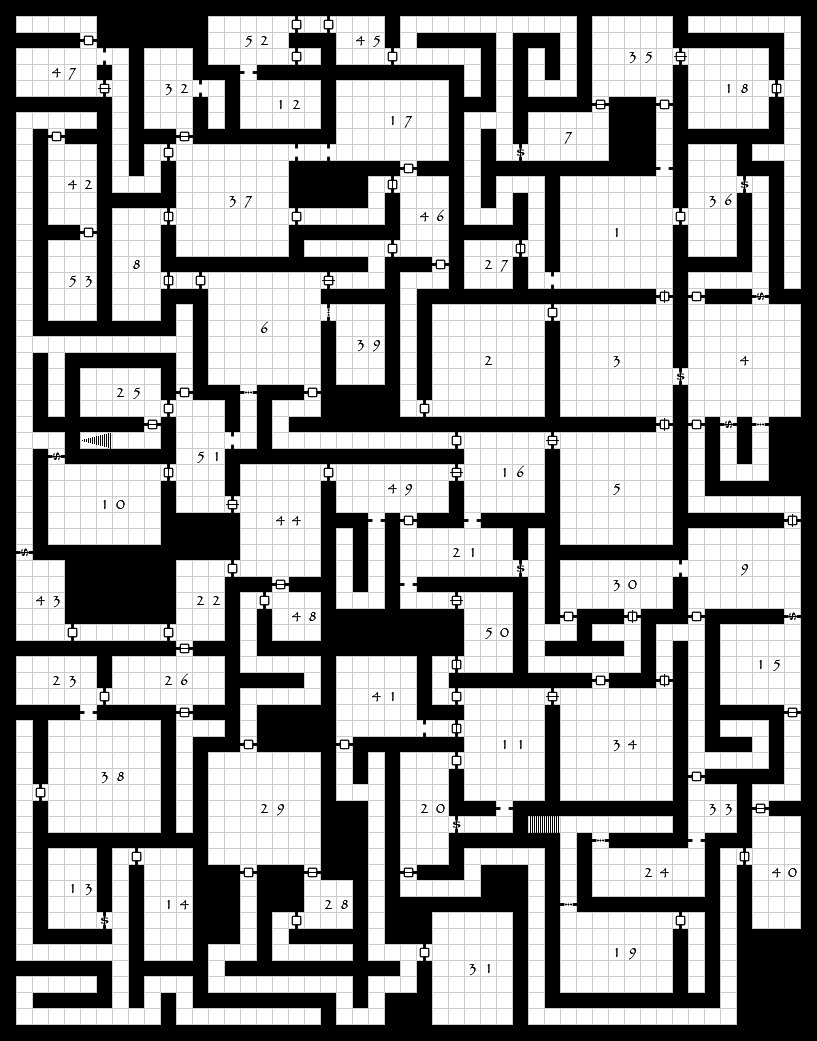
* You may ***NOT*** use any threading libraries like boost, C++11, (TBB) Threading Building Blocks, etc. The only threading and synchronization mechanisms allowed are the basic Win32 or C run-time calls (CreateThread, \_beginthreadex, critical sections, interlocked exchange, etc.).
* Your submission must run using the **Release** **x64** build of Visual Studio 2022 with the default settings (C++ language is C++**14** by default).

## Questions:

This is a variation of the type of output the cMazeMaker program uses.

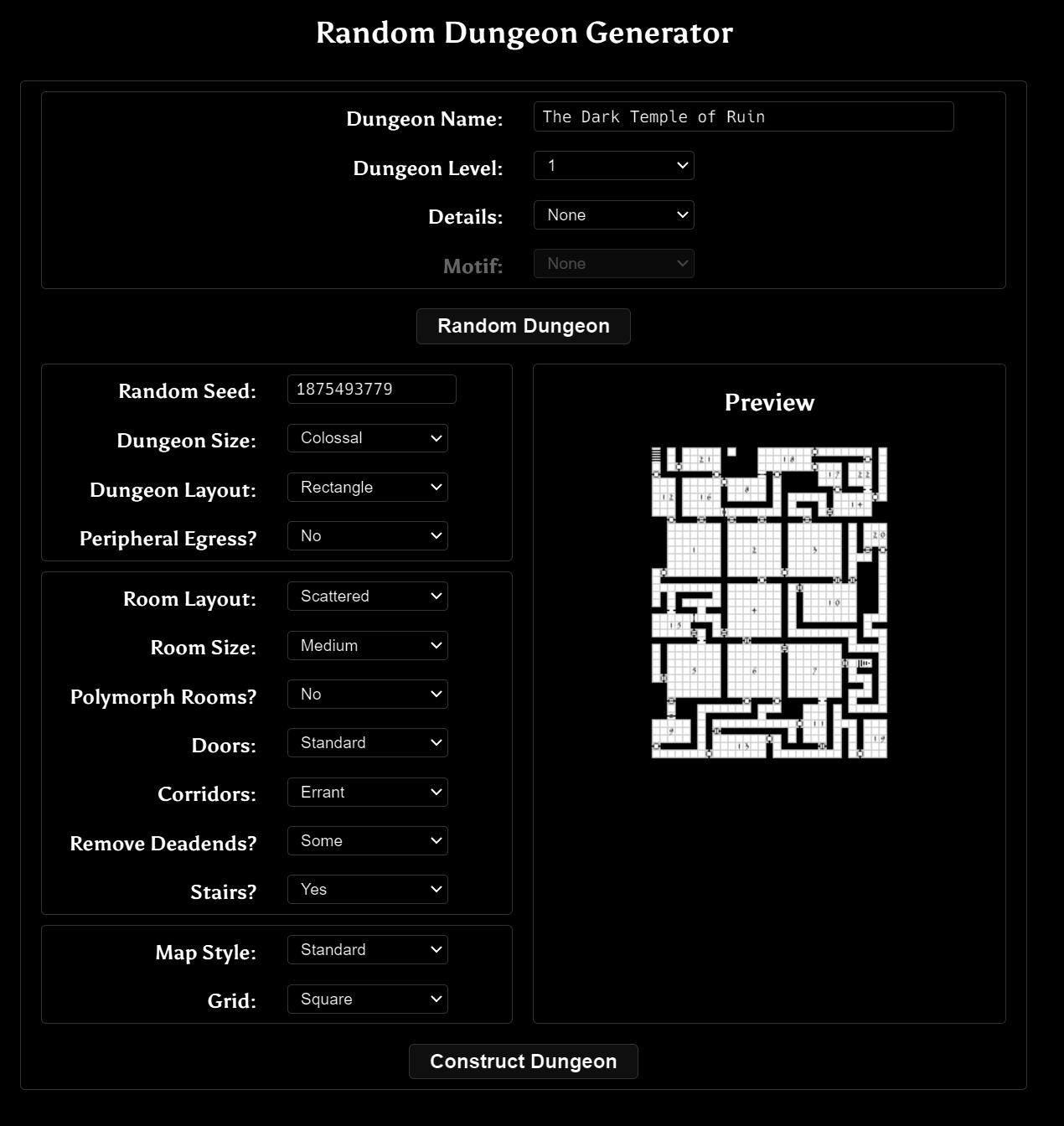
Instead of that, you will use the donjon Random Dungeon Generator located here: <https://donjon.bin.sh/fantasy/dungeon/>

You are to simulate a bunch of explorers looking for treasure in an underground dungeon.

As an example, here is the “The Lost Crypts of Sorrows” dungeon as generated by the site above.

A few things of note:

* It’s a Dungeons and Dragons (DnD) map generator so has a number of other things that you are going to ignore, like stairs, types of doors, etc.
* On the map, there’s some stairs, but not a clear “entrance” and “exit”. You are going to assume that:
* The **entrance** is the lowest left most cell/square of the map
* The **exit** is the highest right most cell/square of the map.

1. (25 marks) Generate your dungeon by:   
   * Going to the <https://donjon.bin.sh/fantasy/dungeon/> site. You’ll see this 🡪
   * Change “**Dungeon Size**” to “**Colossal**”
   * Change “**Polymorph Rooms**” to “**No**”
   * Change “**Room Size**” to “**Large**”
   * Change “**Stairs**” to “**No**”
   * Leave everything else as is.
   * Click “**Construct Dungeon**”

Note that if you refresh your page, the dungeon will go away BUT if you type in the same random seed, it will generate the same\* dungeon.

(\*OK that’s not 100% true: it will generate the same room layout, but will change the name, the type of doors, and some other details.)

**BEFORE YOU CONTINUE**, make a note the “Random Seed” number.

Once generated, go to the bottom of the screen and you’ll see a “Download” button that has options when you click on it.

**Save all the files** it generates and *include them with your submission* (the maps, TSV, HTML, PDF, JSON, etc. ALL of it.)

**Also, tell me what seed you used** (you did remember to make a note of that, right?)

You will only need the “**TSV**” file to finish the rest of the exam (I have no idea what that is, but it’s the file my little dungeon program can read, so we’ll use that).

**Build the donjon\_tvs\_converter*\_2* program and feed it your TSV file.** You can either run the program with the file as a command line parameter or just drop your TSV file onto the executable. It will spit some output to the screen as well as to a file with “(cleaned)” in the name.

This “cleaned” file is the one you are going to need.   
  
This Internationally Famous and Award Winning program will take the sort of whacky formatted TVS file, make some changes and additions, and then output something like the cMazeMaker program you’ve used before.

The original TVS file is a 2D grid structure (like the cMazeMaker output).

It also contains a bunch of different types of doors, though.

I’ll be referring to the “cleaned” file (i.e. the output of the donjon\_tvs\_converter***\_2*** program) for the rest of the exam, as the original file is tricky to parse, and you won’t need it. i.e. if you see “TSV” file from now on, assume it’s the “cleaned” TSV file from the donjon\_tvs\_converter***\_2*** program.

The output lists the size of the dungeon, if that’s useful to you (but you could also get that from the file output, too).

It also lists the items that were in the original file. Note that these are multiple characters in length. It will also list a *single* character equivalent.

The ‘.’ character represents cells that are walls, and the “F” represents cells that are *not* walls (I’m guessing “F” means “floor”, but I have no idea).

The first dungeon output is the dungeon with the ‘F’, the ‘.’, and the other items as single characters.

The second dungeon output is only ‘X’ and ‘.’ characters where the ‘X’ is a wall and the ‘.’ is *not* a wall; i.e. this is just like the cMazeMaker output. If there’s a door, it replaces it with a ‘.’

|  |
| --- |
| ***NOTE:*** *There are also “f” and “w” letters (for “food” and “water”), but for the sake of interpreting the map, treat these as “.” characters (i.e. not wall or floor).* |

The dungeon output is delimited with “DUNGEON\_BEGIN” and “DUNGEON\_END”.

You can choose whatever file you’d like and you can even edit the file for your own use BUT you can NOT edit the dungeon.

You can even “hard code” the dungeon into your code if you’d like. Like paste the text into a set of vectors or an array or something.

I’ve giving you the file to help make things easier for you.

You may also use the code from the donjon\_tvs\_converter***\_2*** if that’s helpful.

**You are going to mimic the adventure of Theseus in the labyrinth of Crete.**

**See:** [**https://en.wikipedia.org/wiki/Minotaur**](https://en.wikipedia.org/wiki/Minotaur) **(the “Theseus myth” part) for more info.**

The TLDR is: Minotaur monster is in a maze, people are thrown in the maze as a sacrifice.

One day, a guy named Theseus goes into the maze and not only kills the Minotaur, but finds his way out of the labyrinth. First person to do it, apparently. Pretty bad-ass.

So from now on, I’ll use “maze” or “labyrinth” instead of “dungeon”.

1. (50 marks) Present this dungeon maze on screen as an OpenGL output.

I’d suggest that you just use cubes or flat quads, placing one where there’s an “X” (from the 2nd TVS output file).

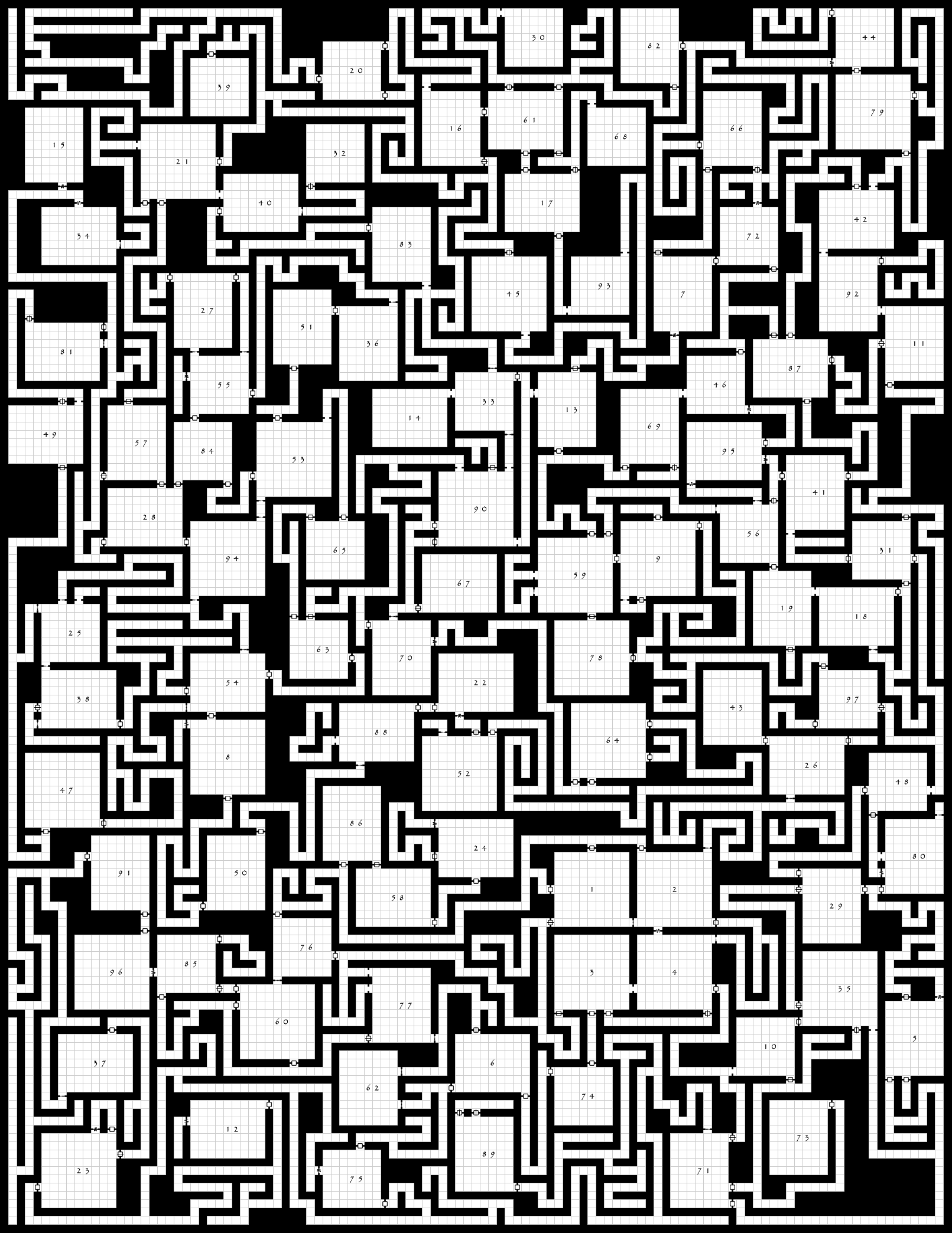
I’d also suggest that you just place the camera so it’s looking down onto the dungeon, but that’s up to you. You *will* have to have the capability to get close to the dungeon *and* “zoom out” to see the entire dungeon at once.

You won’t get any marks for “beauty” here, but you have to have sufficient lighting, colours, textures, or whatever, so that I can *clearly* see what’s going on.

I don’t even care if there’s only 2D squares/quads representing floors and walls, just as long as I can see what’s happening.

However, making “everything” wireframe is *not* allowed. You *can* use the odd wire frame object (if you feel that’s useful) but making wireframe makes it almost impossible to see anything.

1. (100 marks) Place Theseus and the Minotaur in the maze (labyrinth):



Place them at opposite “ends” of the maze, approximately where these red circles are:

So not *right* in a corner, but maybe in the top right “10%” of the maze, if that makes sense.

Set up the movement of the Minotaur in the following way:

* The Minotaur randomly picks a horizontal or vertical direction and moves until it hits a wall or intersection *UNLESS...*  
  + Theseus is within 10 squares away (i.e. within a circle with radius of 10 “tiles) in which case it can “smell” Theseus, and instead of picking a random direction, it will pick a horizontal or vertical direction that will move it closer to Theseus, *whatever is larger*.  
      
    For example, if Theseus is 8 units to the right and 5 units down, the Minitor will move to the *right*. This will have the Minitor either zero in on Theseus
  + If Theseus is within 3 tiles *and* there’s no wall in the way, then Minitor will attack.

Demonstrate this by placing some small mesh at the Minitor and Theseus’ locations.

Show the “state” of the Minitor (can **smell Theseus** or not, or has **attacked**) in some way.   
I’d suggest changing the colour or adding another small mesh above the Minitor mesh.

In the next question, Theseus will move on his own, but for now, add some key controls to allow the movement of Theseus around the maze.

Theseus *can’t* move through walls, so you’ll have to handle that.

Set up a game loop so that we can see the Minitor moving around. I’m suggesting no more than a few tiles a second *at most*. If this is DnD, then the tiles are 5 feet across. One tile/second (5 feet/second) is around 5.5 kph. Full grown bulls (the Minitor is ½ man, ½ bull) run at 20-24 kph.

What I *don’t* want is some instant teleportation of the Minitor...

1. (100 marks) Theseus moves himself:

We’ll assume that since the Minitor is a monster, if doesn’t need food or water (except for eating people, I suppose), but Theseus *does* need to eat and drink.

The donjon\_tvs\_converter\_2 program has placed 50 pieces of food and 100 bottles (?) of water throughout the maze. These are marked with “f” (for food) and “w” (for water).

We’ll also assume that Theseus has a map of the maze (in the story, Ariadne helped him navigate the labyrinth, so maybe she gave him a map or something?)

Theseus will choose a path one of two ways: Either just like the Minitor, or he’s moving directly to food/water.

* He has a health of 100. Every time he moves a tile, he uses 1 health.
* Because he has a map, he knows where the food and water is, too.
* Normally, he wanders around just like the Minitor, moving in a horizontal or vertical direction until he hits a wall or an intersection, then picks another direction to move.   
  (in other words, he uses the same “movement code” as the Minitor)
* If his health is <75%, then his path is chosen like this:
* He will find the closest food or water (remember, he has a map)
* He will *plan his path* before moving:
  + He will make a note of each square he’ll need to move.   
    You can do this by keeping a list of locations/grid coordinates in a vector or something.
  + Starting from the square he’s on, he randomly picks a adjacent square that’s not a wall or a square he’s already checked (more on this in a moment) and makes a note of this new location on his path (i.e. adds it to the vector/whatever).
  + He can check if the location is already looked at if it’s already in the vector.   
    (i.e. before he adds it, he can search to see if it’s already there).
  + If there are no move paths to take (they are all walls or they are already on his path), then he’s reached a dead end.
    - In that case, he starts the search all over again.
  + Eventually, one of these “plans” will lead to food/water...
  + Note that *no movement happens while this search is happening.*
* Once he has found a valid path, he moves along that path (the one in the vector) on square at a time.
  + On the console, print out how many paths Theseus investigated.
* If he gets to the food, he gains +2 health, and +1 if it’s water. Remove the item from the map once he’s consumed it. (Note he *still* expends energy when he’s moving)
* Indicate the two “movement states” with either a colour or some other mesh above Theseus. Also indicate if his health is <75% or not.

1. (50 marks) Theseus and Minitor run into each other:

If Theseus and the Minitor meet, then one of two things happens:

* If Theseus’ health is >50% then he kills the Minitor and continues wandering around looking for an exit. It’s up to you if you want to give Theseus an exit or just have him wander around until he starves to death.
* If his health is <50% then the Minitor kills Theseus and continues moving around looking for more people to eat.

Indicate this in some way: changing the colours, changing the mesh, etc.

Also, print something to the console to indicate which character “died”.

*And now for the part you KNEW was coming...*

1. (500 marks) Thread the the “I’m hungry” search that you used in question 4.  
     
   When Theseus’ health is <75%, spawn at least 32 thread, each one trying a different path to the fool/water.  
     
   There are three issues you will face:
2. Preventing your “random” selection code from picking the same value every time. Remember that rand() *and* the C++11 random library is *not* thread safe, so you’ll have to add some code to make it thread safe/re-entrant.
3. Determine if the search reaches a dead end and has to be restarted. If this happens, you can either have the thread start its search again, or exit and another thread is spawned.  
     
   BUT remember that you will still have to keep track of how many possible paths were checked, so if the thread starts again, you’ll have to note that.
4. The counter that is checking the possible paths likely needs some protection so when you increment it, it’s accurate.

That’s it.