# **INFO6023 – Game Algorithms & Gems – Mid-term – Winter 2024**

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**. 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.   
  + <http://www.fanshawec.ca/admissions/registrars-office/policies/cheating-policy>
  + <http://www.fanshawec.ca/sites/default/files/assets/Ombuds/cheating_flowchart.pdf>
* The questions are **NOT** of equal weight. There are six (6) pages with four (4) questions
* You are to submit a **single** project that answers all the questions:
  + You are ***not***to submit a separate project per question.   
    (if you do this, I will either mark the first one, or just give you a mark of zero.)
  + You are ***not***to have any code and/or reconfigurations to demonstrate different questions.   
    I will ***not***modify your code and/or files and/or file structure *in any way* and will mark your submission *as is* – meaning that if that untouched build + run doesn’t answer certain questions, then you’ll get zero on those questions.

|  |
| --- |
| *(Note: Having separate projects that contain your own DLLs or libraries that your code uses are 100% OK.)* |

* 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.
* This exam should nominally take you three (3) hours to complete (i.e. if it was physically scheduled in a classroom, it would be for three (3) hours). You have until **11:59 AM** on **Friday, March 1st** to submit all your files to the appropriate drop box on Fanshawe Online. This is an exam, so don’t miss that deadline.
* Also be **SURE** that you are actually submitting the correct files pointing to the correct folders, etc.
* I’ll be monitoring e-mail, discord, and will be in my office (6th floor) for part of the day, if you have any questions.

## For all questions:

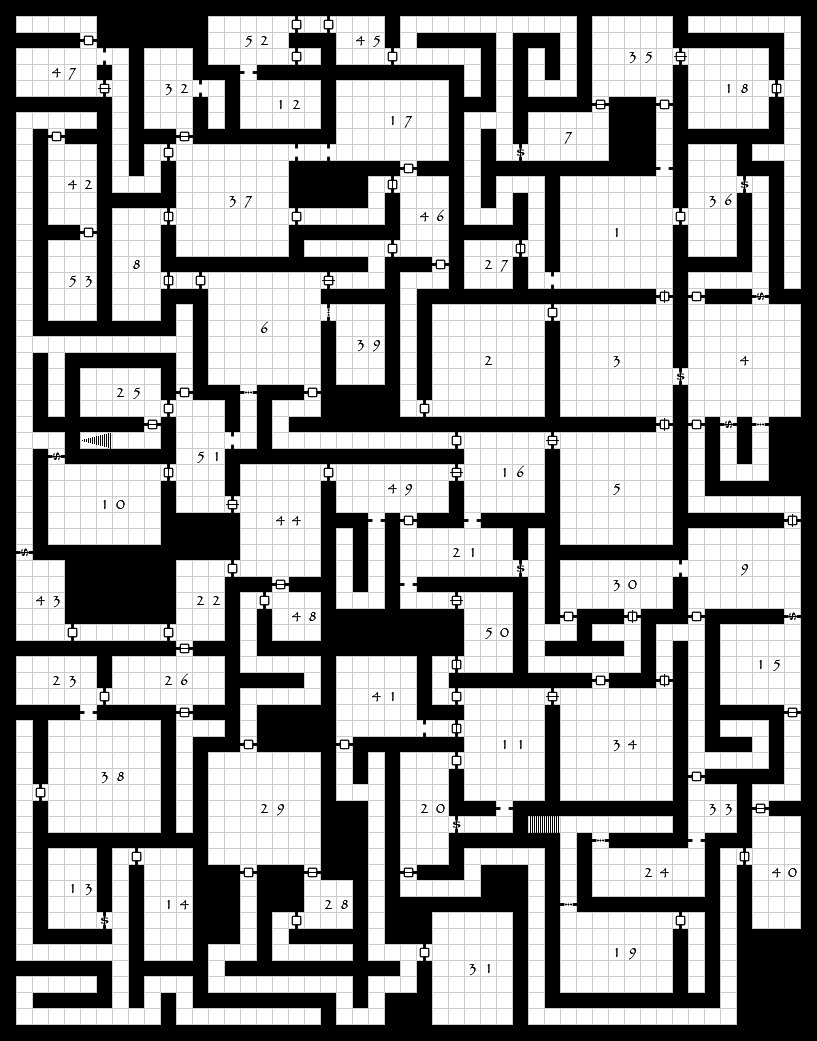
* You may ***NOT*** use any threading libraries like boost, C++11, (TBB) Threading Building Blocks, etc. The only threading and synchronization mechanisms allowed are the basic Win32 or C run-time calls (CreateThread, \_beginthreadex, critical sections, interlocked exchange, etc.).
* Your submission 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.

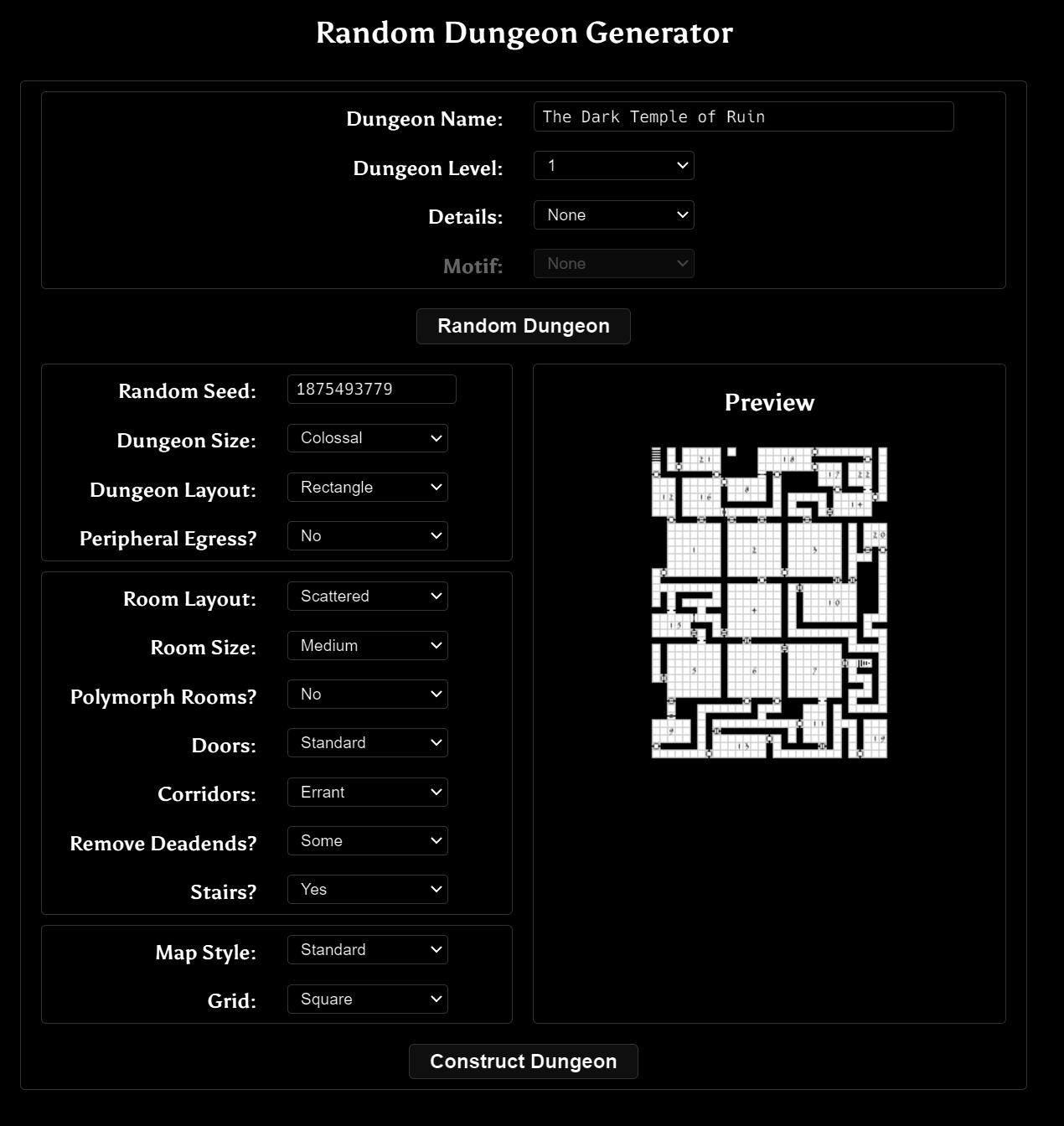
Instead of that, you will use the donjon Random Dungeon Generator located here: <https://donjon.bin.sh/fantasy/dungeon/>

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**”
   * 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 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 (that I wrote this afternoon) will take the sort of whacky formatted TVS file and make it into something like the output of 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” (i.e. the output of the donjon\_tvs\_converter 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 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 ‘.’

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 if that’s your preference.

1. (50 marks) Present this dungeon 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.

Oh, “everything is a wireframe” sort of thing us *not* allowed, either. You can use wire frame a little bit if you’d like, but making everything wireframe will make it almost impossible to see what’s happening.

1. (100 marks) Have a single treasure hunter collect all the treasure and leave.

Randomly place 250 treasure items in the dungeon. You do this by randomly picking a location and seeing if it *isn’t* a wall. On the output, place a differently coloured shape or model in the cells/rooms that have treasure.

Make sure they are “randomly” scattered, and not just piled in one place, more than one treasure item in the same cell/room, or in a line or something.

Your “treasure hunter” can also be something very simple, but make it distinct from the treasure and the rest of the dungeon (different model/colour/whatever).

Place the treasure hunter at the “entrance” of the dungeon (left most cell/square).

Have them wander through the dungeon until they’ve collected all the treasure. You can use any technique you’d like for this, but I’d suggest a random (“drunken”) walk or some variation of that. It’ll take a little while, but it’ll get done.

“Collecting” the treasure means they got the cell/room with some treasure in it. Remove the treasure once they “collect” it.

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

1. (250 marks) Now place 64 treasure hunters into the simulation, all looking for the *same* treasure, with each treasure hunter’s code running on it their own thread of execution.   
     
   The challenge here will be to:

* ...prevent your “random” selection code from picking the same value every time. Remember that rand() or the C++11 random library is *not* thread safe, so you’ll have to add some code to make it thread safe/re-entrant.
* Determine when a piece of treasure has been picked up by a treasure hunter. It’s virtually certain that two treasure hunters will find the same treasure at the same time, but you can only give the treasure to one.
* You will also have to have the treasure hunters “know” when all the treasure has been collected, so they know when to leave.

As each treasure hunter exits the dungeon, have them print out how much treasure they collected. Like how many of the 250 treasure items they individually found.

That’s it.