

PA2 – Maze

Student Information

Integrity Policy: All university integrity and class syllabus policies have been followed. I have neither given, nor received, nor have I tolerated others' use of unauthorized aid.

I understand and followed these policies: Yes No

Name:

Date:

Submission Details

Final **Changelist** number:

Verified build: Yes No

Number Tests Passed:

Required Configurations:

Discussion (What did you learn):

Verify Builds

- Follow the Piazza procedure on submission
 - Verify your submission compiles and works at the changelist number.
- Verify that only MINIMUM files are submitted
 - No – Generated files
 - *.pdb, *.suo, *.sdf, *.user, *.obj, *.exe, *.log, *.pdb, *.db
 - Anything that is generated by the compiler should not be included
 - No – Generated directories
 - /Debug, /Release, /Log, /ipch, /.vs
- Typical files project files that are required
 - *.sln, *.suo,
 - *.vcxproj, *.vcxproj.filters, *.vcxproj.user
 - *.cpp, *.h
 - CleanMe.bat

Standard Rules

Submit multiple times to Perforce

- Submit your work as you go to perforce several times (at least 5)
 - As soon as you get something working, submit to perforce
 - Have reasonable check-in comments
 - Seriously, I'm checking

Write all programs in cross-platform C++

- Optimize for execution speed and robustness
- Working code doesn't mean full credit

Submission Report

- Fill out the submission Report
 - No report, no grade

Code and project needs to compile and run

- Make sure that your program compiles and runs
 - Warning level ALL ...
 - NO Warnings or ERRORS
 - Your code should be squeaky clean.
 - Code needs to work "as-is".
 - No modifications to files or deleting files necessary to compile or run.
 - All your code must compile from perforce with no modifications.
 - Otherwise it's a 0, no exceptions

Project needs to run to completion

- If it crashes for any reason...
 - It will not be graded and you get a 0

Leave Project Settings

- Do NOT change the project or warning level
 - Any changing of level or suppression of warnings is an integrity issue

Leaking Memory

- If the program leaks memory
 - There is a deduction of 20% of grade
- If a class creates an object using new/malloc
 - It is responsible for its deletion
- Any **MEMORY** dynamically allocated that isn't freed up is **LEAKING**
 - Leaking is **HORRIBLE**, so you lose points

No Debug code or files disabled

- Make sure the program is returned to the original state
 - If you added debug code, please return to original state
- If you disabled file, you need to re-enable the files
 - All files must be active to get credit.
 - Better to lose points for unit tests than to disable and lose all points
- Disable your debug printing otherwise you will lose points

Due Dates

- See Piazza for due date and time
- Submit program performance in your student directory assignment supplied.
- Fill out your this **Submission Report** and commit to performance
 - **ONLY** use Adobe Reader to fill out form, all others will be rejected.
 - Fill out the form and discussion for full credit.

Goals

- Required:
 - Multi-Threading Maze program - Using C++ 11 threading model
 - Required to use 2 or more threads besides main() thread.
 - Any C++ 11 threading functionality allowed

- Write-up
 - if multithreading IS working
 - 2-3 page pdf of how your systems work
 - Detailing your application (see below)
 - if your system isn't working
 - Describe how your system should work
 - Convince me that you understand what you are doing, but are having implementation details

Assignments

1. Simple summary

- a. Maze program is provided that demonstrates two solutions
 - i. Single Threaded Breadth First Solver
 - 1. *STMazeSolverBFS*
 - ii. Single Threaded Depth First Solver
 - 1. *STMazeSolverDFS*
- b. Each of these solvers will be able to run the series of sample mazes
 - i. 5x5, 10x10, 20x20, ... 1Kx1K, 2Kx2K, 5Kx5K, 10Kx10K, 15Kx15K
 - ii. See sample maze project
- c. Goal
 - i. Create a multi-threaded solver from either of the existing single threaded solvers (provided) or create your own, that performs better in time than the existing single threaded solutions.
 - ii. Your solution must use two or more threads (not including the main thread).
 - 1. The lifetime of the threads can vary.
 - 2. Threads can run the same functional code, but in different contexts.
 - 3. You can have different threads doing different roles
 - iii. The majority of your timing improvement **must** come from the division of work between threads.
 - 1. Optimizing existing single threaded solutions with little to no work done in the external threads does not count.

2. Some Rules

- a. Common sense
 - i. Remember the spirit of this assignment, to take a large scale project and make it a multithreaded solution
- b. Goal isn't to find an algorithm from the internet on concurrent mazes.
 - i. The goal isn't to win the performance contest at all cost.
 - ii. You should be able to create a good performing solution from the material provided.

- c. You can receive a very good grade
 - i. If you systematically create a concurrent multithreaded solution
 - ii. Get the multithreading solution working
 - iii. Analyze the performance - understand and explain its the behavior from a performance / memory / threading perspective
- d. The maze creation
 - i. Needs to be as is, in the original format, no processing in the maze creation or loading
 - ii. The underlying data structure is atomic int
 - 1. You can create additional structures and data
 - a. But that's inside the timed section
- e. Your timing begins
 - i. Solver construction / Solver execution
 - 1. There you can change the default data if you desired (you are on the clock - timed)
 - ii. You can change/ refactor / or create a different algorithm for your maze solver
 - 1. I was able to refactor the BFS solution
 - 2. I was able to speed up the DFS solution as well

3. Testing

- a. Four test files are provided that will be use in testing:
 - i. Maze15Kx15K_E.data
 - ii. Maze15Kx15K_J.data
 - iii. Maze20Kx20K_B.data
 - iv. Maze20Kx20K_D.data
- b. Running the tests
 - i. This maze data is accessed inside the // Maze_DevelopmentData directory
 - 1. 4 maze files from above will be used
 - ii. A script will be executed for performance testing.
 - 1. Test_Constest.bat - script
 - iii. There is a Flag in main.cpp that must be set to run the submission tests
 - 1. `#define FINAL_SUBMIT 1`
 - iv. Run the script
- c. Code review
 - i. Everyone's code will be reviewed
 - ii. Understanding how you accomplished multithreading
 - iii. Using the written document and comments to understand your code
- d. Remember:
 - i. You need at least two working threads (besides main thread) to receive full credit.

- ii. Optimizing the single threaded model isn't sufficient.

4. Paper

- a. 2-3 page pdf paper (more pages is OK)
- b. Necessary items to cover:
 - i. Description of the application
 - 1. Your overall approach / strategy
 - 2. Diagrams
 - ii. Thread creation process
 - 1. Who creates the threads
 - 2. Names you use in code
 - 3. Each thread responsibilities
 - iii. Communication between different threads
 - 1. What is signaling, callbacks, mutexes, synchronization operations
 - iv. Complete Data movement
 - 1. Follow the data through the whole process to the actual playing
 - 2. Atomics, scope, visibility
 - v. Challenges you had and what you learned
 - 1. Please explain your hardships and lessons learned here.
- c. If for some reason your system isn't working
 - i. Your paper and descriptions need to be VERY detailed to convince me that you know what you are doing.
 - ii. Explain what went wrong with your approach.
 - 1. Why are you in this situation (be honest)
 - iii. Expect paper size to be 6-10 pages diagramming every problem and how you would solve it.

5. ***Make sure it builds for Debug/Release configurations***

- a. Suggestion: Implement and develop for Release/x64
- b. After that configuration works → verify all two configurations:
 - i. Debug x64 ← checking memory leaks here..
 - ii. Release x64 ← used in performance test

6. ***Make sure it Doesn't leak memory in Debug mode***

- c. Suggestion:
 - i. Occasionally test your code in Debug mode with memory tracking on

Validation

Simple checklist to make sure that everything is submitted correctly

- Is the project compiling and running without any errors or warnings?
- Does the project run **ALL** in all configurations without crashing?
- Is the submission report filled in and submitted to performe?
- Follow the verification process for performe
 - Is all the code there and compiles “as-is”?
 - No extra files
- Is the project leaking memory?

Hints

Most assignments will have hints in a section like this.

- Baby steps
 - You’ll be in trouble if you don’t
- This is so slow and painful, takes forever to get working.
 - You cannot escape the agony of this part
 - Just do it.
- Hard to debug print for this project
 - Suggest using stream or sprintf to a buffer
 - Faster, doesn't hit thread performance that much
 - Convert to I/O at the end of application