Faculty of Computer and Information Helwan

Project : Maze Solver Problem Parallel Processing

Name : Marc Essam Erian
ID : 20150398

Name : Eyad Mohamed Shokry
ID : 20150156

Name : Antoin Ashraf Fathy
ID : 20150668

Name : Karim Mamdouh
ID : 20150392

December 15, 2018

CONTENTS

1 INTRODUCTION

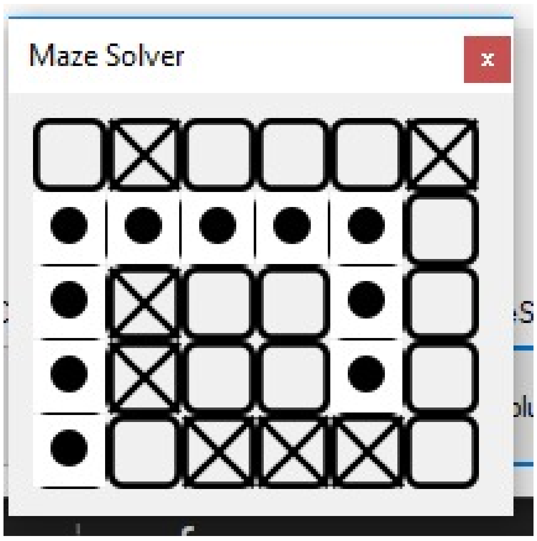
Firstly we have created 2 Mazes one of them is easy to solve of size 5 X 7 and the other maze is a complex one of size 10 x 10.
We Created a sequential function to solve the any maze it takes the maze from the file then it convert the data in the file into an interface Maze
"." represents an empty path.
"*" represents an Obstacle in the path where the car cannot move.

"^" represents the starting direction and starting point of our maze.
"E" represents ending point of the maze where the car should stop.

2 SCREENSHOTS

2.1 MAZE 5 X 7 :

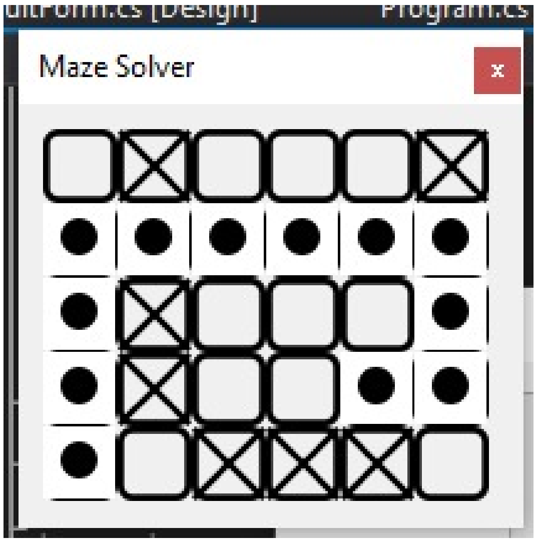
2.1.1 5 X 7 MAZE SEQUENTIAL :



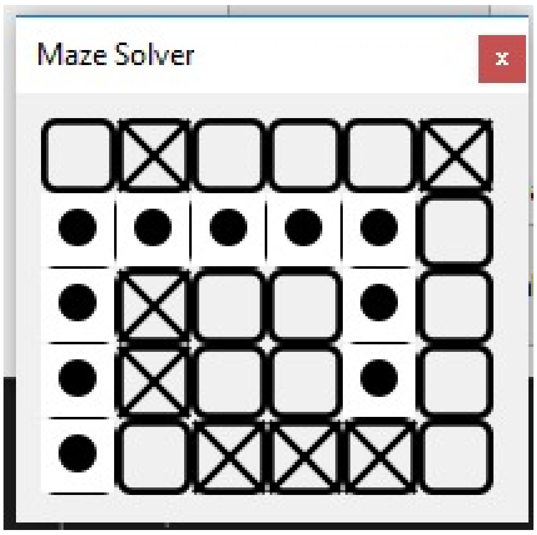
2.1.2 5 X 7 MAZE PARALLEL :

We made it run several times with several paths so as to make it parallel as it shows a new path each run till all the paths are shown.

2.1.3 First Run in Parallel :



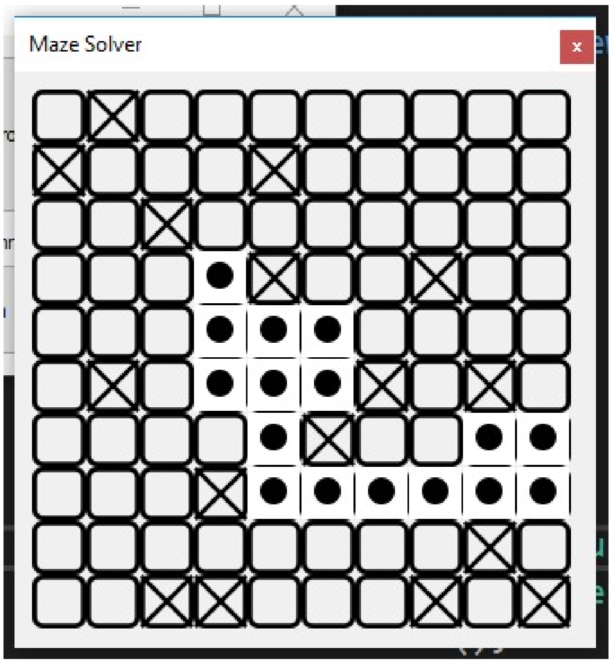
2.1.4 Second Run in Parallel :



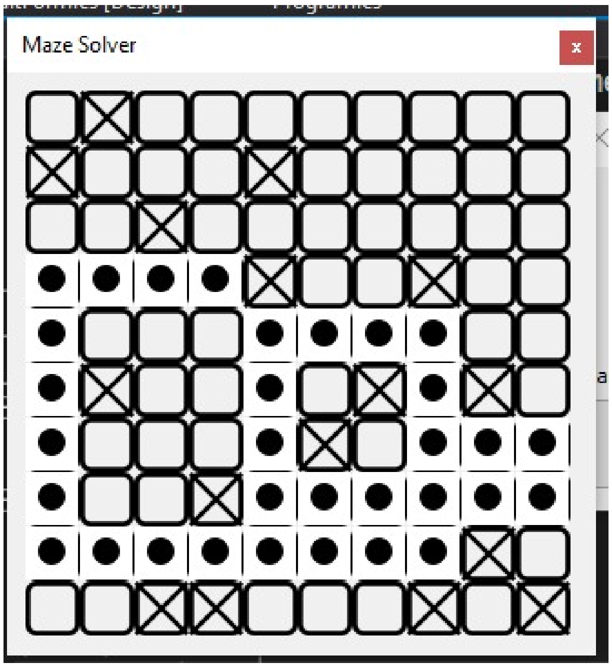
2.2 10 x 10 Maze :

In this maze we tried to make it harder for the algorithm to solve it by adding more obstacles in its path but it succeeded in solving it sequentially as well as in parallel.

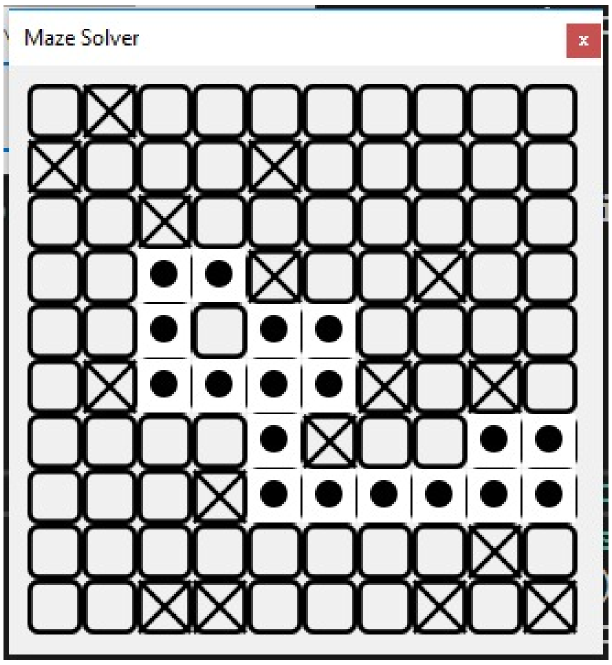
2.2.1 10 x 10 Maze run in sequential :



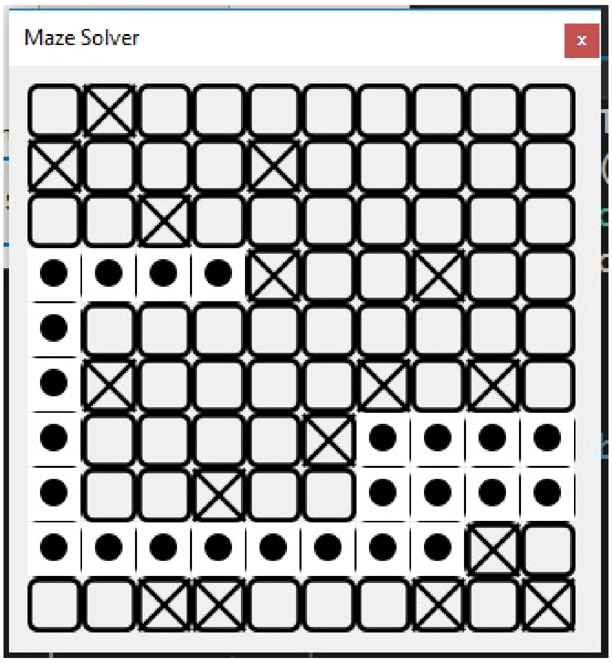
2.2.2 10 x 10 Maze run in parallel First Run :



2.2.3 10 X 10 MAZE RUN IN PARALLEL SECOND RUN :



2.2.4 10 X 10 MAZE RUN IN PARALLEL THIRD RUN :



3 Parallel Processing C# Constructs Used :

3.1 System.Threading Part 1:

we used threads in the parallel BFS algorithm while searching for paths while searching the nodes we pass each possible node to be visited to a thread in order to save the paths and show it in the interface.

3.2 CancellationTokenSource Part 2 :

We used this Construct in order to check the possible paths but in our problem we have obstacles when the car finds that it is in front of an obstacle and there is noway to reach the destination we cancel this task in order to try another path till we reach a complete path of the car from the starting point to the end point.

3.3 Tasks Part 2 :

Task is a class representing a single Operation also helps in threading. We created a list of tasks in order to save operations - Movements of the car - . We pass a list of nodes to the Task as well as the CancellationTokenSource so as to cancel operation if the car is in front of an obstacle and couldn't move to take a new path. This threading class helps our car to see the possible nodes to pass through at the same time.

3.4 Timer Part 2 :

We used Timer in order to compare between the time that the sequential algorithm spent to solve the maze with that of the parallel algorithm.

3.5 async await functions Part 3 :

In order to improve responsiveness and compute possible paths at the same time as well as avoiding performance bottlenecks if the car keeps on looping in its place without reaching the goal or terminating the program and in order not to interrupt a path being computed by another thread so await function is used.

3.6 WhenAny() Part 3 :

Creates a task that will complete when any of the supplied tasks have completed.We can summarize it that when a task completes another task is started automatically.

3.7 ManualResetEvent [Mutex] Part 5 :

We used it because each calling thread is blocked until it acquires ownership of the mutex, it must call the ReleaseMutex method to release ownership of the thread so as not to interrupt an ongoing task.

---

4 Parallel Processing Design Pattern

Our app needs to perform 100+ CPU-intensive operations, each taking roughly 3-5 minutes. Execution order doesn't matter.So we can Classify our Design Problem to Design Problem 1 referring to the design pattern lectures that we studied.
so we need to Create 1 task per core, as one finishes create another.

4.1 Pipeline :

- Linear flow from one task to another :

  we succeeded to do so by using await() in order to check if a task is ended or not and if so WhenAny() is used to create a new task

4.2 DataFlow :

- Generalized flow with one-to-many :

This happens when the car is at the starting point trying to find possible paths that it is trying to pass through. when it finds more than path to go through , we use threads to help it to save as many paths as possible.

4.3 Concurrent Data Structures :

- We used QUEUE in the BFS searching Algorithm in order to save nodes visited and thus finding possible paths.

---

5 Conclusion

After Comprehensive run of the Algorithms We concluded that parallel algorithm are much faster than sequential although this problem is a small one but When we compared the Times that each one spent to solve it we found that the parallel one was faster.

---

6 Credits :

- we used github in order to organize our work :

  Our Repository link : https://github.com/EyadMShokry/CarMazeSolver/
- we also used Latex to document our work in this project.
