**Comparison of Various Pathfinder Algorithms' Experimental Performance Using Different Datasets**

Damla Büşra Çelik **150140102** [*celikdam@itu.edu.tr*](mailto:celikdam@itu.edu.tr)

Korel Chairoula **150140910** [*chairoulak@itu.edu.tr*](mailto:chairoulak@itu.edu.tr)

Yunus Güngör **150150701** [*gungoryun@itu.edu.tr*](mailto:gungoryun@itu.edu.tr)

**1. Introduction**

Pathfinding algorithms makes daily life easier. It can be easy for an intelligent brain to find, but sometimes there can be lots of variables and paths so that can confuse a human which can result in failure. Computers make this job easier for people to find their paths. The most efficient path to these destinations are calculated by specific algorithms called pathfinding algorithms. There are several pathfinding algorithms such as A\*, BFS (Breadth First Search), DFS (Depth First Search) and Dijkstra’s algorithm in which their specifications will be discussed in the background section. Our aim in this project is to present the characteristics of several pathfinding algorithms and help designers to choose the most suitable and efficient algorithm in their projects. The following sections of this document includes a background information about pathfinder algorithms following by methods in which explains the steps. Before conclusion, in section schedule and cost the progress of this project is discussed.

**2. Background**

**2.1 BFS (Breadth First Search) Algorithm**

Breadth First Search (BFS) is a search algorithm that checks parents or root first, then checks child nodes to find nodes which provides necessary condition or conditions. An adaptation of this algorithm to pathfinding is to convert available paths to a tree which has costs when moving node to node. After that shortest path can be calculated searching each node for total cost to get there. Algorithm's complexity is O (bd) at worst case because breadth-first search must travel all the nodes and if you consider an average case, half of the nodes which has depth of d, needs to be searched so complexity is still O (bd). [Korf 1985]

**2.2 DFS (Depth First Search) Algorithm**

Depth- First Search(DFS) is an algorithm that searches graphs or trees.  It has the method of checking the child nodes first, then the parent(root)nodes. It has the method of going as deep as it possibly can, that’s why it’ called depth-first search. Algorithm’s complexity is O (bd) at worst case (for implicit graphs with branching factor b and depth d). [Massachusetts Institute of Technology, 2017]

**2.3 Bidirectional**

This type of pathfinding algorithm starts from both start and finish. So, first of all you need to have access to finish to run this algorithm.  When both paths which begins from start and finish algorithm is finished. But this algorithm does not guarantee the most efficient path.

**2.4 Dijkstra’s Search Algorithm**

Dijkstra’s pathfinding algorithm starts from the starting point, by marking infinite every distance from the starting point, which means that those other points have not been visited yet. Initially, the starting point is marked zero and after marking the starting point, other distances between other nodes are labeled iteratively (after the starting point other points will be visited), when a closest distance is found, the distance is updated. When visiting all the nodes and updating all the distance between those nodes are terminated, the shortest path is found. Algorithm’s complexity is O (E + |v |log| v|) where E is the number of edges and v is the number of vertices. [Cornell University, 2017]

**2.5 A\* Search Algorithm**

A\* estimates pretty well its route to the destination point. Let’s assume that n is our destination point. A\* uses the function f(n) = g(n) + h(n) to find its way from starting point to the destination point with the least cost. g(n) is the complete cost calculated from the starting point to the arrived point and h(n) is the approximate cost from the starting distance to the destination point n. Using g(n) and h(n) A\* finds and chooses its optimal route in every step taking account to f(n). [Patel, 2017]

**3. Methods**

**3.1 Random Algorithm**

This algorithm used as a control group rather than a competitor. Its pure aim is to travel around the maze randomly, without considering anything until a path is found.

**3.2 Creating the labyrinth**

The labyrinth that we are going to test the pathfinding algorithms will be a 2D maze. An algorithm will be implemented to create a random maze with size of 100x100 and these labyrinths will have different characteristics such as crooked, flat and crooked combined with flatmazes with one input – output and one input – multiple outputs.

**3.3 Steps**

In this project, we are going to implement our pathfinding algorithms using C++ programming language. We will produce experimental data by testing our pathfinder algorithms in three different types of labyrinths several times in which we will be using these data to compare their speed according to the system’s time, efficiency and energy cost. In addition to that, as in every algorithm the complexity of each will also be calculated and extra data will be gathered using tools provided by the OS (e.g. Windows Performance Monitor – MacOS Activity Monitor). After gathering empirical data, we will analyze and create a report.

**4. Schedule and Cost**

**4.1 Work packages**

This section includes parts of the projects that can be done individually.

**Work package 1:** Preparation of the coding environment such as interfaces and constant patterns that will be used throughout the project’s development.

**Work package 2:** Implementing an algorithm that generates random mazes into the project environment which has been already created before.

**Work package 3:** Testing project environment and maze generation algorithm with Windows Performance Monitor and MacOS Activity Monitor to gather data as CPU usage, energy cost and memory usage.

**Work package 4:** After the implementation of the maze generation algorithm, we will be generating different mazes and eliminate the ones which does not fit to the specifications.

E.g. For experiment 1 the specifications are one input – multiple outputs crooked map.

For experiment 2 the specifications are one input – one output and flat map with flat lines and long corridors. For experiment 3 the specifications are one input – multiple outputs like a geographical map with several crooked - long roads. Until the desired maze will be generated, algorithm will be work continuously and will be eliminated by us.

**Work package 5:** All the pathfinding algorithms mentioned in **section 2 (BFS, DFS, Bidirectional, Dijkstra’s and A\* algorithms)** will be implemented by Yunus, Korel and Büşra. All these pathfinding algorithms will be tested in experiment 1, experiment 2 and experiment 3.

**Work package 6:** In this stage, Yunus is going to work with experiment 1. The tests of the crooked maze with one input – multiple outputs will be done by the pathfinding algorithms **(BFS, DFS, Bidirectional, Dijkstra’s and A\* algorithms)** separately several times and data according to the tests such as efficiency, power consumption, memory status and time will be gathered.

**Work package 7:** In this stage, Büşra is going to work with experiment 2. The tests of the flat maze with one input – one output will be done by the pathfinding algorithms **(BFS, DFS, Bidirectional, Dijkstra’s and A\* algorithms)** separately several times and data according to the tests such as efficiency, power consumption, memory status and time will be gathered.

**Work package 8:** In this stage, Korel is going to work with experiment 3. The tests of the geographical maze (crooked roads and long corridors) with one input – multiple outputs will be done by the pathfinding algorithms **(BFS, DFS, Bidirectional, Dijkstra’s and A\* algorithms)** separately several times and data according to the tests such as efficiency, power consumption, memory status and time will be gathered.

**Work package 9:** After the completion of the experiments mentioned above, the produced data will be analyzed and chart will be created according to the efficiency, power consumption, time and memory usage.

**Work package 10:** In this stage, the experiment reports will be written individually.

**Work package 11:** All the reports will be gathered and concatenated and the final presentation will be prepared.

**Work package 12:** For the prevention of statistical errors the prepared presentation will be proofread and the mistakes will be corrected to finalize the presentation.

**Work package 13:** All the experiment, will be presented in weeks between 12 and 15 according to attached week.

**4.2 Schedule**

|  |  |  |
| --- | --- | --- |
| No: | Work packages | Planned Time (Spring Term 2017) |
| **1** | Creating necessary structures for work environment | **5. Week** |
| **2** | Labyrinth Implementation | **6. Week** |
| **3** | Testing Tools | **6. Week** |
| **4** | Labyrinth Elimination | **7. Week** |
| **5** | Implementation of pathfinding algorithms | **9. Week** |
| **6** | Experiment 1 | **9. Week** |
| **7** | Experiment 2 | **9. Week** |
| **8** | Experiment 3 | **9. Week** |
| **9** | Data Analysis | **10. Week** |
| **10** | Presentation Write-up | **11. Week** |
| **11** | Concatenating Experiment Reports | **12. Week** |
| **12** | Proofreading | **12. Week** |
| **13** | Presentation | **12., 15. Week** |

**4.3 Key Responsibility Roles**

|  |  |  |  |
| --- | --- | --- | --- |
| Work packages/Names | YUNUS GÜNGÖR | KOREL CHAIROULA | DAMLA BÜŞRA ÇELİK |
| Creating necessary structures for work environment | I | RA | I |
| Labyrinth Implementation | CI | I | RA |
| Testing Tools | CI | CI | RA |
| Labyrinth Elimination | RAC | RA | RA |
| Implementation of pathfinding algorithms | RA | RA | RA |
| Experiment 1 | RA | I | I |
| Experiment 2 | I | I | RA |
| Experiment 3 | I | RA | I |
| Data Analysis | RA | I | I |
| Presentation Write-up | RA | RA | RAC |
| Concatenating Experiment Reports | CI | RA | CI |
| Proofreading | RA | RA | RA |
| Presentation | RA | RA | RA |

**R:** Responsible **A:** Accountable **C:** Consulted **I:** Informed

**4.4 Research**

In the research phase, we will examine pathfinder and labyrinth generating

algorithm’s codes. Also, we will make a research that examines the strong and weak

sides of these algorithms such as which algorithm to use when and where.

**4.5 Coding, Testing and Analyzing**

We will implement six algorithms **(BFS, DFS, Bidirectional, Dijkstra’s and A\*)** which the implementations of these algorithms will be divided equally among us. Furthermore, we will build a testing environment for the algorithms in which it will include a random labyrinth generating algorithm. The elimination of the labyrinths mentioned in **Work packages 3** will be done by all of us. The operating system’s performance measuring tools will be used to gather the data and the gathered data will be analyzed, examined and the graphical charts will be created.

**4.6 Documentation and Submission**

All the gathered data from **section 4** will be examined carefully and will be documented. The previously done calculations will be re-calculated in order to prevent statistical data error and inconsistencies. The finalized document will be proofread until the document is perfect and ready to submit.

**5. Conclusion**

All in all, in this project, we will compare several pathfinder algorithms with various data sets to examine their behavior. In order to accomplish it, we will implement these algorithms and create a testing environment to gather statistical data. Basically, these algorithms will be tested in three different types of labyrinths with different complexities several times to get accurate statistics. In our final report, which algorithm is likely to be used where, their performance and difference between will be our final step of this project.

**6. References**

CS312 Lecture 20 Dijkstra's Shortest Path Algorithm. (n.d.). Retrieved March 05,

2017, from

[https://www.cs.cornell.edu/courses/cs312/ 2002sp/lectures/lec20/lec20.htm](https://www.cs.cornell.edu/courses/cs312/%202002sp/lectures/lec20/lec20.htm)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY Department of Electrical Engineering and Computer Science 6.034 Artificial Intelligence. (n.d.). Retrieved March 05, 2017, from [**http://www.ai.mit.edu/courses/6.034b/searchcomplex.pdf**](http://www.ai.mit.edu/courses/6.034b/searchcomplex.pdf)

Patel A. 2017. “Amit’s Thoughts on Pathfinding”. Retrieved from:

<http://theory.stanford.edu/~amitp/GameProgramming/index.html>

Richard E. Korf. 1985. Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence, 27 (September 1985), Pages 97-109,

DOI: <http://dx.doi.org/10.1016/0004-3702(85)90084-0>