INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Challenge: Snake search algorithm comparison.

Course: Artificial Intelligence

Team:

Carlos Roberto Cueto Zumaya	A01209474
Alan Kuri García	A01204805

Professor:

Ruben Stranders

Due Date:

November 20, 2018

Pathfinding is a necessary component for most games today, characters, vehicles, animals are just some things that move in a goal-oriented manner and the game needs to identify paths that avoids obstacles at the most efficient way possible. But there is a complication, games run on limited resources, movements have to be computed in real time and there are multiple elements to compute a path for, not only that, but it also needs to look realistic. Many games in practice use A* as it has good performance, its implementation is rather easy and the paths are always optimal as long as the heuristic function is admissible, however, its overall time depends on the grid size and complexity, many 3D games (e.g. Dark Souls, Gears of War, Halo, etc...) are not perfect grids, graphics are defined by a mesh so a new challenge arises, it might require dropping of a ledge or running upstairs or as you can see on the image below, climb a hill but the enemies are not aware of that new dimension.

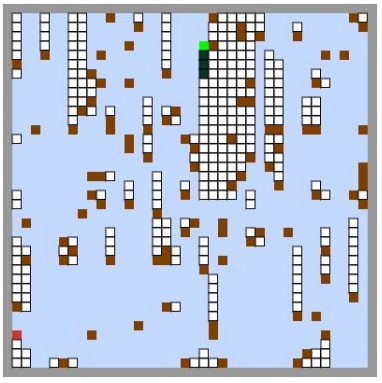
Since the algorithms are specifically designed and optimized for 2D grid-based pathfinding were each cell is connected cardinally with four neighbors, the developers needed to modify the existing algorithms to work with 3D; the navigation mesh is an invisible object in the game world which covers every space where entities can move around, it works by removing all the environment that should not be navigable making sure to generate a walkable path. Now with powerful hardware it's possible to do ray casting from every corner so they became nodes and the rays becomes an edge.

Several real-time algorithms had been proposed, these are used on RTS games where players maneuver units and structures to secure areas of the map and/or destroy their opponents resources, the two most used are TBA* and HCDPS, the first one is a time-sliced version of the A* that adds the ability to control move planning and the second one performs offline abstraction defining representative regions of the map with a compressed path database between all regions with minimal memory per agent.

We thought about Snake, a 2D game where the player maneuvers a snake which grows by eating food and with the line itself being a primary obstacle.

DFS

DFS runs horrible when the grid gets bigger, since the time complexity is proportional to the total number of vertices and edges O(N*M), that means that the snake will be forced to traverse the entire grid if the food it's at the opposite side, therefore its complete but not optimal.

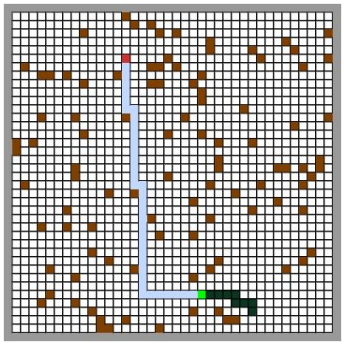


BFS

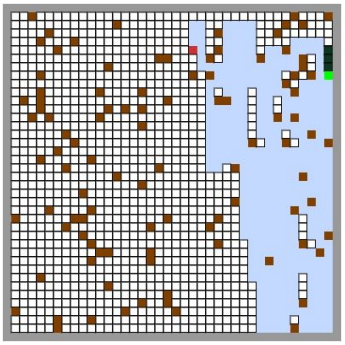
BFS runs good and fast for the first few foods eaten but since it must store at least an entire level of the tree in the queue it needs to explore the nodes of that level and when the snakes get medium length it can easily lead to a dead end because there may not be more moves available. Also, the algorithm does not consider the placement of the food since we used a cost of 1 for every movement optimizing the path, there is a high chance of having a dead end but after all it is complete and optimal.

A*

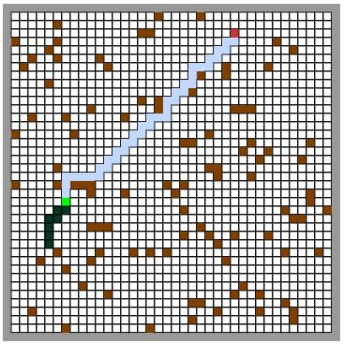
A* is guaranteed to find an optimal path if it exists, we used the Euclidean and Manhattan distance as heuristics, up to some extent, it is comparable to BFS as both can find the optimal path, however, A* only expand nodes that can potentially lead to an optimal path with less processed nodes it can still lead to a dead end if the snake is long enough also because there may not be more moves available.



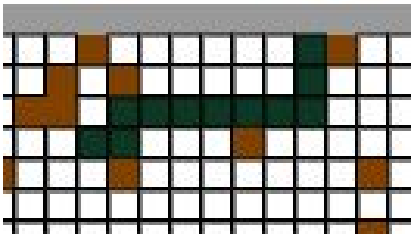
Typical BFS path



Typical DFS path



Typical A* path



Stuck scenario

Setup

To run our project, you need to install python at least 3.6.6 and the newest version django 2.1.3.

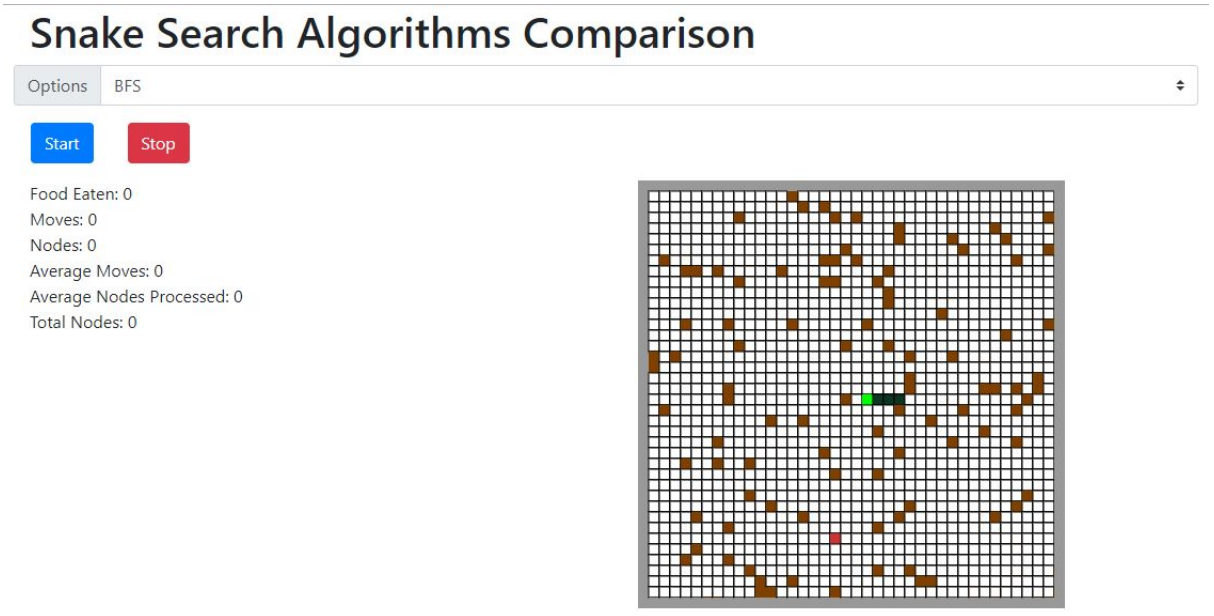
To install all the necessary requirements, type *pip install -r req.txt* on the root folder.

```
(venv) D:\Users\crcz\Documents\Tec\Semestre9\AI\snake_search_algorithms>pip install -r req.txt
```

To execute server locally type *python manage.py runserver* on the root folder and access through http://localhost:8000/

```
(venv) D:\Users\crcz\Documents\Tec\Semestre9\AI\snake_search_algorithms>python manage.py runserver
```

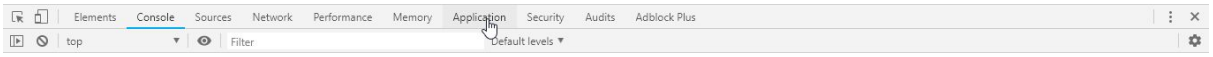
The page will load the game and all the current stats, and the search algorithm options where you can change to DFS, BFS or A*.



To start or stop an evaluation you just need to click on  or  .

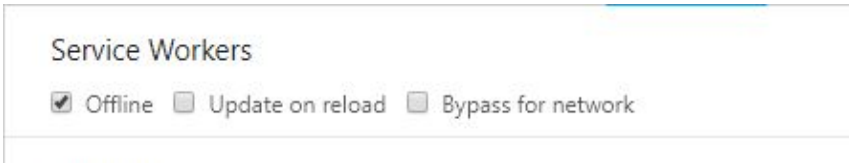
To enable javascript worker:

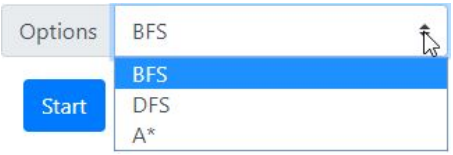
- Right click anywhere on the chrome page and select inspect or type the shortcut ctrl+shift+i
- Left click on the Application tab



- Under the tab Service Workers

- Check the Offline option





Analysis

To compare the performance of each search algorithm we create 3 test cases where everything was the same except the number of foods the snake had to eat. The metrics to evaluate the quality of the different pathfinding techniques are:

- Time taken
- Distance traveled (moves)
- Number of nodes explored
- Numbers of node expanded
- Average moves/food
- Average nodes/food

Test 1

Search	Config	Time sec	Moves	Nodes Processed	Average Moves	Average Nodes Processed	Total Nodes Expanded
BFS	Grid: 40x40 Square size: 10x10 Obstacles: 100 Food: 5	13	118	3,397	23.8	679.4	259,131
DFS		108	1,056	1,107	21.2	221.4	810,888
A* Heuristic: Euclidean Distance		13	118	844	23.6	168.8	57,867
A* Heuristic: Manhattan Distance		13	118	630	23.6	126	43,263

Test 2

Search	Config	Time sec	Moves	Nodes Processed	Average Moves	Average Nodes Processed	Total Nodes Expanded
BFS	Grid: 40x40 Square size: 10x10 Obstacles: 100 Food: 10	26	241	6,878	24.1	687.8	943,602
DFS		415	4,069	4,261	406.9	426.1	9,980,103
A* Heuristic: Euclidean Distance		26	241	1,706	24.1	170.6	234,749
A* Heuristic: Manhattan Distance		26	241	1,198	24.1	119.8	168,759

Test 3

Search	Config	Time sec	Moves	Nodes Processed	Average Moves	Average Nodes Processed	Total Nodes Expanded
BFS	Grid: 40x40 Square size: 10x10 Obstacles: 100 Food: 20	52	486	13,706	24.3	685.3	3,688,689
DFS		1,004	9,868	10,805	493.4	540.25	56,024,370
A* Heuristic: Euclidean Distance		53	488	3334	24.4	167.2	928,016
A* Heuristic: Manhattan Distance		51	486	2,410	24.3	120.5	657,004

On the third test we realize that the A* with euclidean distance is not admissible, it should at least perform the same or better but not worse than BFS, conversely, the manhattan heuristic is admissible, it found the optimal solution and as fast as or faster than BFS but with less memory consumed, for games it's really good that's why its preference for pathfinding, and even with other heuristics it could be faster.

By no means these search algorithms are bulletproof, there are a lot of cases that need to be taken into consideration when expanding the nodes, so that the next movement doesn't represent higher difficulty. A lot of improvements and optimizations are possible, we think that an easy method iis run at different stages each algorithm, for example, run BFS to find and eat the first foods but when the snake is at medium length maybe 4 or 5 blocks and switch to A* to keep maximum reliability and guaranteeing optimality, having a look ahead movement will be a good feature too, if the next movement happens to be in a dead end with no adjacent movement left the snake could choose another path sacrificing optimality, but we would never use DFS, at least if it doesn't have a cutoff depth value, but then, if the food is at a higher depth there will be no solution found. We can say that the implementations should balance the search time, memory used and the actual path found, and since the experience and success of a video game usually depends on how playable it is, it may represent.

References

Barret, S. (1990). *Path Finding.* Retrieved from Stanford Computer Science: http://www-cs-students.stanford.edu/~amitp/Articles/PathFinding.html

Botea, A., Bouzy, B., Buro, M., Bauckhage, C., & Nau, D. (1998). *Pathfinding in Games.* Retrieved from UFR DE MATHÉMATIQUES ET INFORMATIQUE: http://www.math-info.univ-paris5.fr/~bouzy/publications/dagstuhl2012-cpf-report.pdf

Bourque, L. (2016). Retrieved from AI-Snake-Game: https://github.com/louisbourque/AI-Snake-Game

IronEqual Game Studio. (2017). *Pathfinding Like A King — Part 1.* Retrieved from Medium: https://medium.com/ironequal/pathfinding-like-a-king-part-1-3013ea2c099

Hui, Y., Prackash, E., Chaudhari, S. (2004). *GAME AI: ARTIFICIAL INTELLIGENCE FOR 3D PATH FINDING*. Retrieved from IEEE: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1414592

Kang, S. (2017). Retrieved from Snake-Star: https://github.com/sacert/Snake-Star

Kong, S. Mayans, J. (2016). *Automated Snake GAme Solvers via AI Search Algorithms.* Retrieved from University of California: http://sites.uci.edu/joana1/files/2016/12/AutomatedSnakeGameSolvers.pdf

Lanctot, M., & Ng Man, N. (2006). *PATH-FINDING FOR LARGE SCALE MULTIPLAYER COMPUTER GAMES.* Retrieved from McGill University : https://pdfs.semanticscholar.org/cd9e/7101f0e586e16f2d8111471fc7a587fd9b33.pdf

Lee, W., & Lawrence, R. (2013). *Fast Grid-based Path Finding for Video Games.* Retrieved from University of British Columbia: https://pdfs.semanticscholar.org/224a/912c7ff65a6362607a9cb97c15da90f296af.pdf

Mathew, G. (2015). *Direction Based Heuristic For Pathfinding In Video Games.* Retrieved from K.S.R Institute for Engineering and Technology: https://www.researchgate.net/publication/277888023_Direction_Based_Heuristic_for_Pathfinding_in_Video_Games/fulltext/55e0770608aede0b572e32c8/277888023_Direction_Based_Heuristic_for_Pathfinding_in_Video_Games.pdf?origin=publication_detail

Pinter, M. (2001, March 14). *Toward More Realistic Pathfinding.* Retrieved from Gamasutra: The Art & Business of Making Games: http://www.gamasutra.com/view/feature/131505/toward_more_realistic_pathfinding.php