



Self-playing Snake Game Based on Pathfinding algorithms

Yang Sui, Jin Xu, Runlin Hou

Snake Game

- Snake Game can be divided into several parts:
 - Map
 - Snake
 - Food
- Implement Snake Game using Graph Theory
 - Map, undirected graph, each edge has the same weight
 - Snake, body of snake would be considered as unreachable vertex
 - Food, marked as a normal vertice

Principle

- Snake length will be initialized as 3
- New food will be generated on random position after the last food was taken
- The snake can not hit the wall and its own body

Goal

- Get score
- Be alive
- Less movements

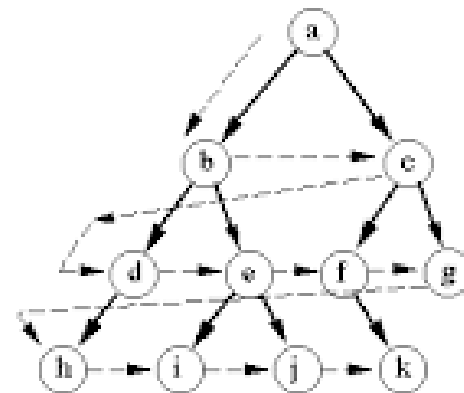
Algorithms

- BFS
- A^*
- Hamilton

Breadth-First Search

- BFS will try to traversal all the map to find the target.
- All the vertices that is going to be visit will be added to a FIFO queue

Dijkstra is same with BFS when the weights of edges are the same



Breadth-first search

Breadth-First Search

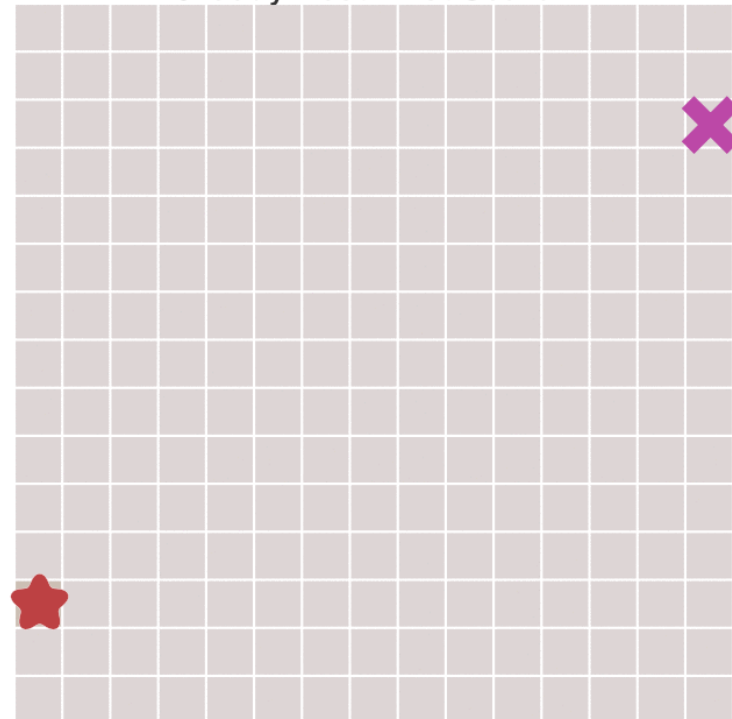
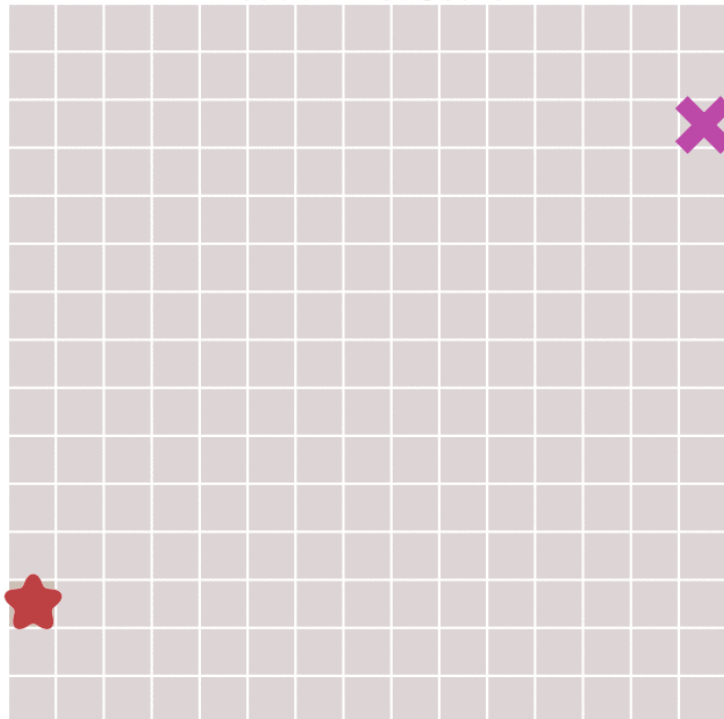
- Visit the root vertice
- Visit the adjacencies of root vertice
- Visit the adjacencies of adjacencies until there is no un-visited vertices

Implementation on Snake Game

- `Connected(ver)` will return a list that the vertices `ver` is connected to
 - It will be avoided to return where the vertices is wall or been occupied by body
- `Bfs(a, b)` will find the shortest path from `a` to `b` using BFS
- Strategy
 - 1st target would be the food
 - If there is no path to the food, trace tail
 - If there is no path to tail, randomly walk

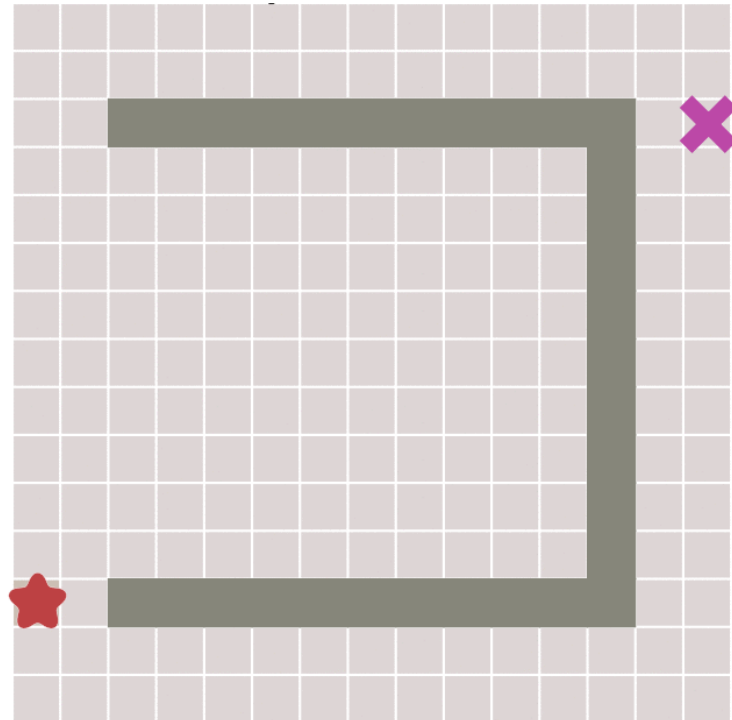
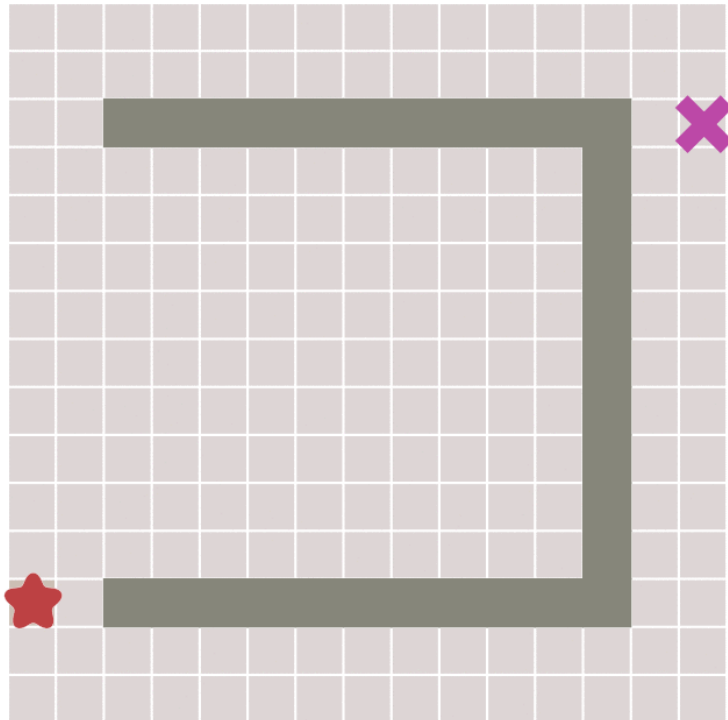
A* Algorithm

- Dijkstra's Algorithm + Greedy Algorithm



A* Algorithm (cont)

- Dijkstra's Algorithm + Greedy Algorithm



A* Algorithm (cont)

Dijkstra's Algorithm

- All possible directions
- Always shortest
- Slow

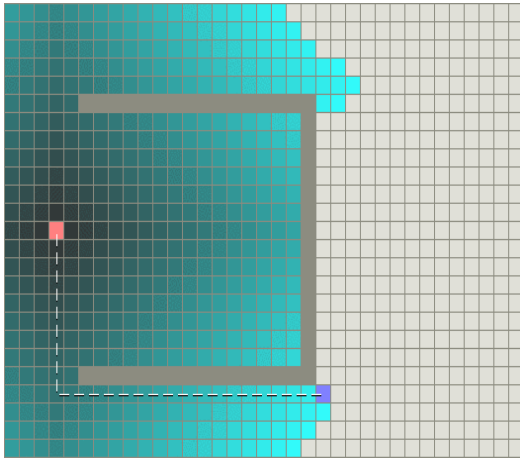
Greedy Algorithm

- The most possible direction
- Not guaranteed shortest
- Fast

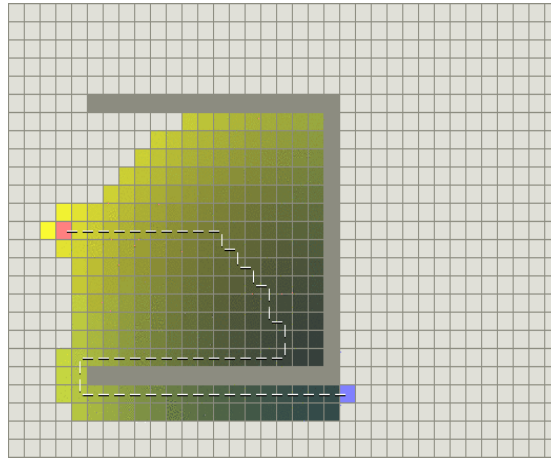
◆ Use a heuristic function to guide the Dijkstra's searching path

A* Algorithm (cont)

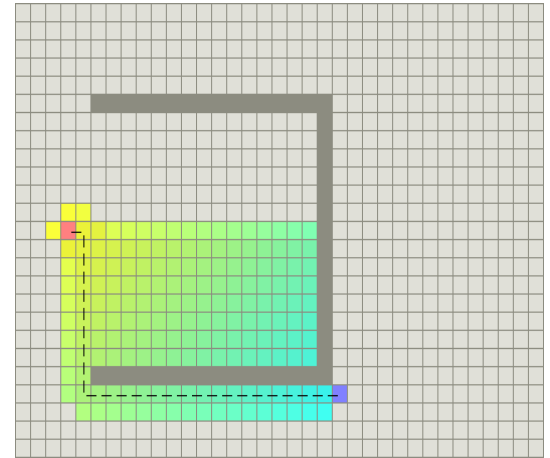
Dijkstra's Algorithm



Greedy Algorithm



A* Algorithm



A* Algorithm (cont)

- Also use a priority queue, but not just adding weights

$$f(n) = g(n) + h(n; d)$$

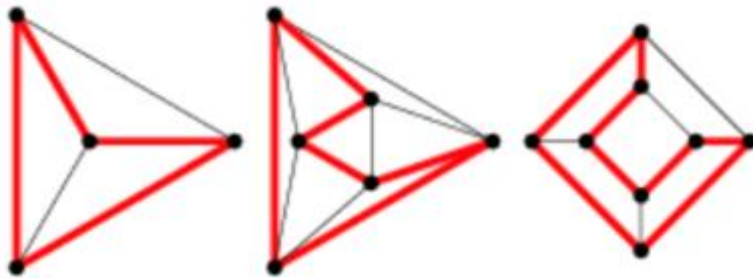
- The cost from starting point to vertex n

- Heuristic estimate the cost from n to destination d

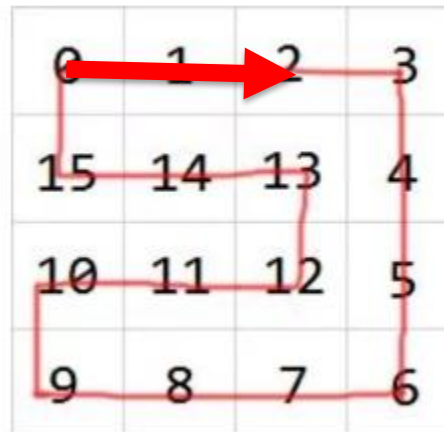
- Manhattan distance
- Euclidean distance

Hamilton Algorithm: Alive Longer

- A Hamiltonian cycle, also called a Hamiltonian circuit, through a graph that visits each node exactly once.

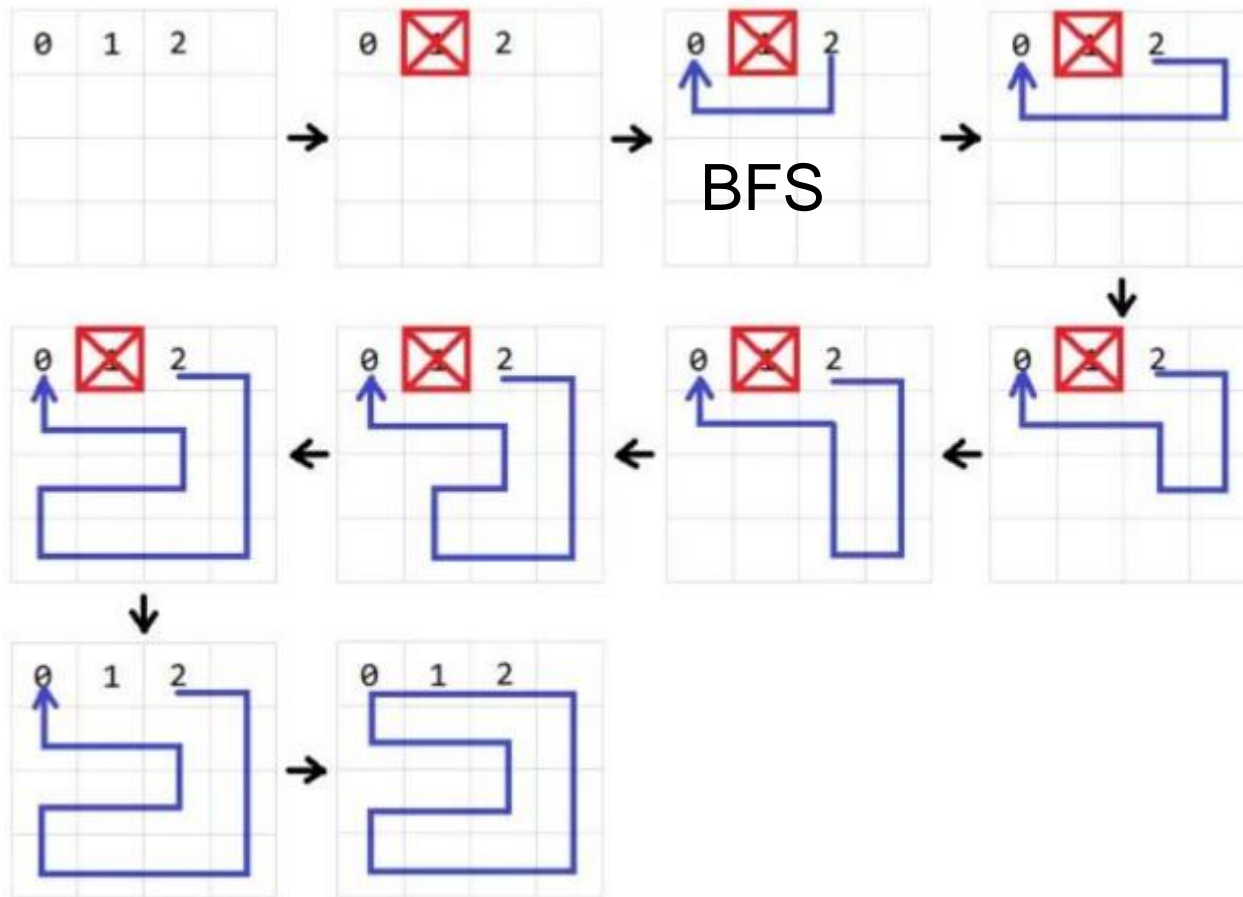


In Snake Game:



→ HEAD
INITIAL
SNAKE

Build the Hamilton circle.



Short Way: Not always Longest Way.

1. Empty screen, with the ordered hamiltonian cycle precomputed and order numbered

1	2	3	4
0	7	6	5
15	8	9	10
14	13	12	11

2. As the snake moves, it follows the rule of ensuring that the head does not overtake the tail in the hamiltonian cycle ordering

1	2	3	4
0	7	6	5
15	8	9	10
14	13	12	11



3. All calculations and movements can be considered to be done in a flattened 1D space

0	1	2		4	5		7	8		10	11		13	14	15
---	---	---	--	---	---	--	---	---	--	----	----	--	----	----	----



4. For the next move, we can consider the shortcuts possible for the head, only needing to consider the position of the tail. There is thus no need for detecting collisions with the rest of the body, except in the case that we have grown faster than expected and the head is past the tail.

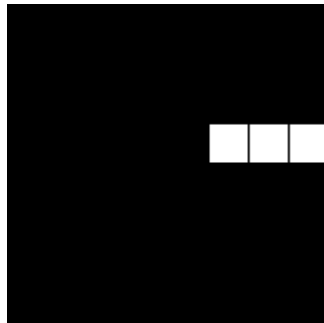
0	1	2		4	5		7	8		10	11		13	14	15
---	---	---	--	---	---	--	---	---	--	----	----	--	----	----	----

Invalid shortcut since it's between the tail and the head

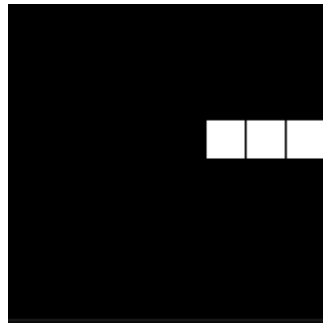


Performance: Alive

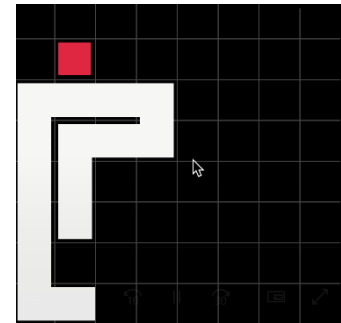
- Hamilton algorithm can finish the map every time
- A*, BFS can only reach half of the map



BFS



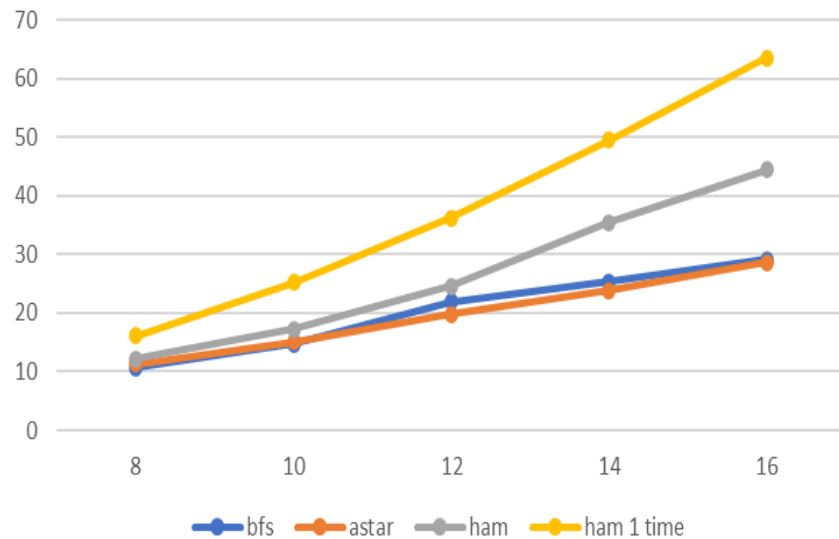
A*



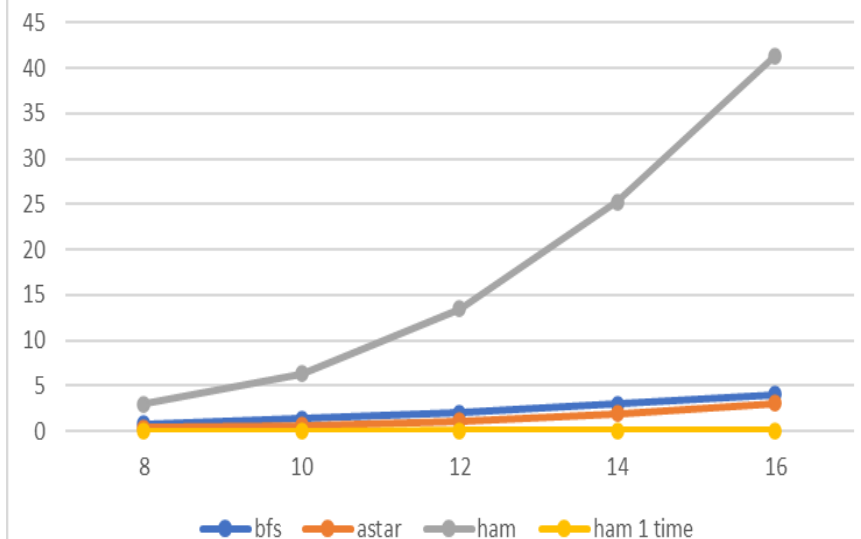
Hamilton

Performance: Move Length & Runtime

Moves per score



runtime(ms) per score



Conclusion

- Move Path Length (less is better) :
 - $\text{BFS} \approx A^* < \text{Hamilton 1time} < \text{Hamilton}$
- Alive(longer is better) :
 - $\text{Hamilton 1time} = \text{Hamilton} > A^* > \text{BFS}$