Games Programming – Homework 01

Search Algorithms and *Finite State Machines*

Name:	Bennet Kuhlmann
Date:	08.12.20
E-Mail:	bennet.kuhlmann@ue-germany.de
Matriculation Number:	95407984
Subject:	Games Progamming
Lecturer:	Kevin Hagen
Type:	Homework
Deadline:	11.12.2020

Index

1.

Search Algorithms Graph Theory

1) What is the difference between a graph and a tree?

2) List and explain all components of a tree.

Search Algorithms

1) What defines a search algorithm?

2) Explain the general search algorithm from the lesson in your own words.

Breadth First Search

1) List three use-cases of breadth first search in games.

2) What is an uninformed search algorithm and why does breadth first classify as one?

3) Explain the differences between depth first and breadth first search. Show how they work differently in a search tree.

Dijkstra Algorithm

1) How does Dijkstra compare to breadth first search and what are key the differences?

2) Name the search algorithm properties applicable to Dijkstra.

3) Implement another type of grid node (currently we have GROUND, WATER, WALL) and assign a unique cost to it. Make it so, that a designer could tweak the cost for all types separately.

Greedy Best First Search

1) How does this algorithm compare to the Dijkstra algorithm and what are the key differences?

2) What is a heuristic?

A*

1) What is the difference between A & A*?

2) What makes a heuristic admissible?

3) Based on the algorithms we build in the course, implement A* on your own.

2.

Finite State Machines (FSM)

FSM Theory

1) Describe in your own words what a Finite State Machine, using a simple example (not the Light Switch from the lecture)

2) Explain the use and functionality of transition tables for FSM.

3) Given is the following FSM:

M = (K, Σ, δ, s, F) ...

… K = {s, q1, q2, q3, q4}

… Σ = {a, b}

… s = {s}

… F = {q2, q4}

… δ = {(s, a) → q1, (s, b) → s, (q1, a) → q2, (q1, b) → q1, (q2, a) → s, (q2, b) → q3, (q3, a) → q4, (q3, b) → q3, (q4, a) → q3, (q4, b) → q4

The input is as follows: “abbabaababbbaab” Visualize the FSM as a graph and show the process it goes through and where it terminates.

Does it stop it a final state?

State Pattern

1) Which problem does the State Pattern solve?

2) How can you implement the State Pattern? Describe the solution and use an UML to support your explanation.

3) What is the difference between implementing states in a static vs an instantiated manner?

4) In class we worked on a simple state machine for a quest giver NPC. Currently it’s a quite linear FSM. Make it more interesting by implementing a fail state: Once you talk to the NPC you only have a given timeframe to collect the quest item. If you don’t collect it before that timeframe, the quest fails. The timeframe should be tweakable in the inspector (serialized field!).

Go ahead and make the necessary changes to the FSM

Search Algorithms Graph Theory

1)

In essence, a Tree is a structured version of a Graph. The difference between those two is, that in the case of a Tree, there is only one path from the Leaf to the Root, there are no other ways to path towards that Root. A Tree are a collection of structured Nodes. A Graph, on the other hand, consists of loosely connected Nodes, which may or may not cause one to make multiple steps back instead of forward until the Root is reached.

2)

- Nodes - Three types of Nodes exist within a Tree, each with a different purpose and function:

    - Root – The starting point of the Tree, there may only exist exactly one Root.

    - Leaf / Leaves – The points at the end of the Tree, from which nothing else connects any further.

    - Inner Nodes - Certain points inbetween Root and Leaves, which connect at least two other Nodes with each other. They act as a bridge that connects Root and Leaves and may branch into multiple more (there does not necessarily have to be a direct path from Root to Leaf).

- Edges – Edges connect Nodes and may reveal whether a correct path between two Nodes exists.

Search Algorithms

1)

A Search Algorithm is defined as a program or module, which searches through connected Nodes or data in order to find a pre-set value / keyword. It will search through those

connected Nodes or data until that value / keyword is found, or until all possible end points have been reached.

2)

The General Search Algorithm begins at a starting point and stores all accessible, connected Nodes in a list. Within the list, the General Search Algorithm will then check each of the listed elements, until either the value / keyword has been found or it is out of elements to check. In the latter case, it will take the children of these elements and repeat until it reaches the end.

Breadth First Search

1)

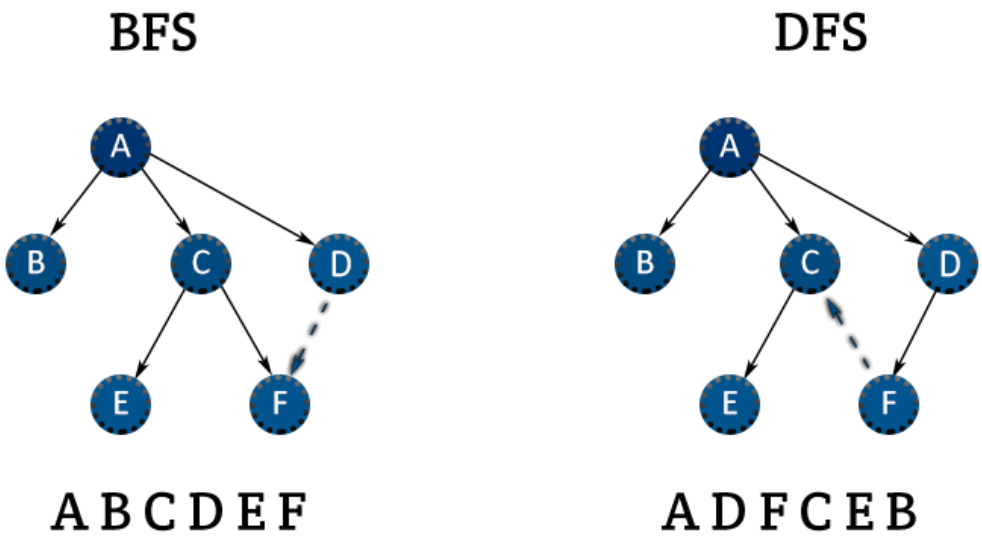
- Level generation, mostly procedural (Rogue-like/lite)
- P2P Networks aka. Torrent networks
- Game path finding

2)

An “Uninformed Search Algorithm” has no knowledge or information about anything in the search space. That means it will not keep states or domain specific information in mind during the search. It will essentially go through the space it is supposed to search “blind”, with only its goal in mind until that one is met or the entire space has been searched without finding the goal.

“Breadth First” counts as an uninformed search algorithm, because it starts at the root and continues searching for a pre-set goal node by node, equally in all directions. The search algorithm knows nothing else but its goal and will keep nothing but that in mind until the goal is reached or the algorithm is unsuccessful after the goal has not been reached and the space was completely searched.

3)



(Fig. 1 – BFS vs DFS)

The key difference between “Breadth First” and “Depth First” is, that the breadth first algorithm starts at the root and searches “line-by-line” or node by node from the root to the nodes below the root and so on.

The depth first algorithm, meanwhile, will choose a branch and continue searching the nodes connected to that branch until it finds a dead end. It will then start again with the next node, which branches from the root.

Dijkstra Algorithm

1)

The Dijkstra Algorithm, unlike the breadth first algorithm, is an informed search algorithm, as it will make decisions based on path cost instead of mere length. The breadth first algorithm, as an uninformed search algorithm, will not take cost into account and simply go by path length. The Dijkstra Algorithm optimizes the search path.

2)

- Optimizing, takes the shortest path to the pre-set goal
- Informed, as it will make informed decisions during the search

3)

Added “Decay” to GridNode and added its color to the BreadthFirstSearchSettings class.

Greedy Best First Search

1)

Greedy Best First Search is a search algorithm akin to the Dijkstra Algorithm, the difference here is, that Dijkstra cares about optimization and path costs, while (as the name implies) the Greedy Best First Search has only the goal in mind and reaching without a care for anything else.

The good thing about the Greedy Best First Search is, that it will find the goal faster than the Dijkstra Algorithm, however, this may come at the cost of being presented with a result that can be far worse than what the Dijkstra Algorithm would find in a longer time.

2)

“Heuristic” describes a process, which can help to produce “semi-decent” results by speeding up an operation through shortcuts. It is mainly used to speed up an existing process and has a use in processes where optimization is irrelevant or impossible.

A*

1)

Basically the only difference is, that if the heuristic of an algorithm is admissible, it is an A* algorithm, if it is not admissible, it is an A algorithm.

2)

The heuristic of an algorithm is admissible if the projected cost of the path is not higher than cost of reaching the goal.

Finite State Machines (FSM)

FSM Theory

1)

A Finite State Machine is a process, which can be in several “states”, with a minimum of two different states it can be in. A Finite State Machine will always have one start and at least one ending state.

A decent example in reality might be a traffic light, which transitions from red to yellow to green (aka. It has three states it can be in: stop -> transition -> go and the other way around).

2)

A transition table helps to visualize the transitions of the FSM based on the state it is currently in and the given input.

State Pattern

1)

The State Pattern determines under what conditions the state is supposed to change and provides the necessary information for that to happen to the states. It also solves the issue of “Spaghetti Code” (or “Yandere-Dev Code” or “Riot Games Coding), by acting as a foundation upon which further code can be built and new states be implemented without it devolving into utter chaos.

2)

A possible solution for the implementation of the State Pattern would be to split the individual states into groups which form an own, smaller state machine. These machines could then be connected to one another to transition from smaller state machine to smaller state machine, while these smaller state machines in themselves run their transitions. For example, one could group the states “Walking” and “Running” into one of these smaller machines and call it “Movement”. Inside of it, based on player input, it would transition between walking and running while active. If the player then would then engage someone in combat, it would switch to the “Combat” state machine, which contains “Attacking” and “Blocking”, because that machine would be connected to the “Movement” machine and vice versa. It would transition between smaller groups and not individual states in essence.

(Fig 2. – GML)

3)

Static States are, as their name implies, static and global. Every time their value gets called, they will send the same, current value of the requested variable in return. Instantiated States, on the other hand, are instance-unique values, which are being created as they are instantiated, this also means they can not be global. An example would be a persistent scoreboard in a Shoot'em Up game, where the score persists from start until death/end of the game. This score would be a static state. An instantiated state in the same example would be a level specific score, which gets reset after the level (but not the game) ended.