**Simplistic AI**

There are many types of AI, most of which are not neural Net based. I find that when people think about AI they automatically think of GANs (Generative Adversarial Networks), the type of AI that the Boston Dynamics Robots and what Tesla use. You may notice a few different types of projects on Youtube.com where people build projects to learn how to use GANs and other types of Neural Networks. They offer more insight into the uses of Neural Nets and how it feels to get started with it.

The only type of AI that isn’t Neural Net based that I noticed is mentioned often enough is the A Star Path Finding Algorithm. This is where we can finally learn about the simpler types of AI. A Star uses a mix of two AI algorithms that I will cover independently from each other first.

---

**Graphs & The Simplest Types of AI**

Most AI require some sort of graph that represents something. In short, a graph is a set of nodes (states) and edges (moves). To build a graph in terms of AI, we need to have some sort of scene in mind, for example, let’s say we start from the beginning of a maze. We can build a graph of moves starting with where we would start the maze.

Diagram

Description automatically generated

This is a relatively small graph for a maze, but you can see exactly how it could be solved. The problem is how it can be solved by a computer though. If we have a small graph like this, we can very quickly find a solution. There is one solution called Depth First Search (DFS) that will be the quickest at finding a solution in this graph, but it may not give an answer quite as short though. Breadth First Search (BFS) does just that. It will always find the best answer, but it may take extra time to find it. Let’s assume our program searches through moves with a preference of order for up, down, left, then right.

Diagram, schematic

Description automatically generated

We can build a graph from this game board that will show all the possible moves from each tile, we know that we won’t have to go back to any tiles because that would be a waste of moving forwards.

Shape, circle

Description automatically generated

Notice how we list each move as so. DFS and BFS would go through this graph specifically, but in different ways. Let’s see how each one works.

Diagram

Description automatically generated

Above is the Depth First Search. It will seemingly slide down the graph to the left and climb back up to the highest left-most node. It stops the moment it finds a solution that includes the goal/end.

Diagram

Description automatically generated

This is the Breadth First Search. Notice how when it goes down each step, it will search the rest of nodes on that layer. In this situation we only had one end case. Both BFS and DFS found that same solution. But one was clearly faster than the other with BFS looking at two extra nodes. If there was an End goal directly Right of 0,2 then then BFS would have found that solution instead taking less moves. DFS would still have found our original solution that has more moves.

One last detail, we don’t have to build these graphs ahead of time, we can always build these graphs as we need their nodes, as to put it, dynamically. But if we have a set size tree with all the possible moves then it’s possible. An example of where it’s no possible, games like Chess or Go can’t just search down a tree and choose every best move. Tic-Tac-Toe can show every possible move on a manageable graph, but we need to use a different type algorithm to find the best solution.

---

**More Complexity**

Imagine how a game of tic-tac-toe or chess go back and forth. Imagine you are trying to figure out your best possible move. You also must keep in mind your opponents move. So, this is where we start using something like a Minmax Algorithm would handle tic-tac-toe just fine. For a larger game like chess or Go, we need to only look towards something slightly different where we also trim the tree using Alpha-Beta pruning in addition to Minmax as our base algorithm. I won’t go into detail for these algorithms on this page.

The last few are for AI algorithms that go back and forth between two players. For other algorithms like the Google Maps program, we use something like Alpha-Beta Pruning, Google has their very own algorithm for pruning away Streets and Highways off a route trying to find the route that give the least amount of time added to your travels.

Sometimes we also find that some of our most basic algorithms are not fast enough or are searching down the longest amount of moves to find an answer. We can find better/smaller strings of moves using several different processes that rely on *heuristics*!

Uniform Cost Search (UCS), Greedy Best First (GBF), A-Star. These are all examples of Heuristic informed searches. Cost and Heuristic functions make these algorithms exactly what they are. UCS uses Cost and tries to minimize its search based solely on that. Cost may include distance that it has already traveled, you could also use this idea for Toll Roads, if you had multiple routes you can go with one of them being a toll road, you might optimize cost by going the slightly longer, Toll free road.

Greedy Best First Searches solely on the heuristic function. The Heuristic Function is just a method which returns the fastest possible distance from its current state to its end state. Then it will decide based on what move it needs to make, just trying to move quickly as possible to the end. For example, a video game NPC trying to move towards you as quickly as possible, if it hits a wall on its way to you, it will likely keep moving along that wall instead of backing away from it. On a Maps algorithm, you could think of it as finding the fastest route, no matter what toll roads or accidents have happened to occur.

A-Star Algorithm uses both ideas. So now your Maps Algorithm won’t just go the fastest route (Heuristic) or the least costly route (Cost). Now it balances both of those things. Imagine you have 3 Routes, One is the shortest distance but has an accident on it, Two is a toll road with a medium distance, Three is no Toll or accident but is the Longest distance. GBF would likely go for One, A Star would likely go for Two, and UCS would go for 3.

The thing about the last three algorithms, is that they might get stuck looking for the best possible method at a local Maximum. This happens often in big datasets. So, we need to use something like Hill Climbing or Annealing to stray from the *best* path and going a slightly better or worse direction. Hill Climbing is a great example, imagine you are a climber that can only see 5 meters ahead of you because of a bad storm. You are trying to get to the tallest part of the mountain you are climbing. So, the most likely thing you will do is just keep going up the hills until you can only see that in either direction the hills start to slope downwards. But what if only 25 meters away there was a taller hill? You wouldn’t be able to see that. In Hill Climbing and Annealing you try solutions that might not make the best sense. Hill Climbing can seem a little random, but it does a good job at finding better, newer solutions. Annealing may work better in other instances though, because the idea is that it starts by looking at a broad range of possible values, but then it slowly creeps onto the closest answer.