GladOS AI Documentation

Introduction: The monster intelligence will allow each monster to make decisions about their movements based on our game's path planning. There were a number of path planning algorithms that came to mind. The first one thought of was Dijkstra's algorithm for deciding the shortest path between any two locations on a given map. After researching more types of path planning algorithms the most suitable one was the A* path finding algorithm.

A* Path Finding: The key components of the A* path finding algorithm are having a rectangular grid along with a start location and end location. Once the path finding algorithm has all three of these things it begins the search.

1. Beginning at the starting node, add it to an open list of nodes to be considered.
2. From the starting node, search for all nodes that are accesable.
3. Add every accessable node to the open list.
4. Remove the starting node from the open list and add it to the closed list.
5. Proceed to the desired adjacent square and repeat the process.

The process of choosing the correct adjacent square consists of using three different values: F, G, and H. The lowest F value is the preferred square and hence the square that is moved to.

F: The final score used in determining which adjacent square will be moved to next.

G: The movement cost to move from the starting location to the current node.

H: The H value is the heuristic. It is a calculation of the estimated movement cost to move from any given node to the ending node. In our implementation of the A* path finding algorithm the heuristic is decided using the recommended Manhatten Distance calculation.

Calculating F, G, and H:

G: This value is calculated every time the monster moves a node. The G values to get from the starting node up to the node the monster is currently on are summed up.

H: The next value to be calculated is the H value which as mentioned previously is calculated by figuring out the Manhatten Distance from any node to the ending node.

F: The final value calculated is the sum of both the G and H values.

Every time an A* path needs to be found the starting and ending locations are picked and the calculations are done on a per node basis until the end is reached.

Implementation:

There are many ways to implement A* path finding. The way we have implemented this path finding is using three different classes.

AInode.java
ManDistance.java
AIastar.java

AInode.java

The AInode.java class is representative of a node that knows its coordinates, its parent, neighbors, and finally is able to calculate its F value with the stored G and H values. Along with the stored values an AInode implements comparable. The reason the AInode implements comparable is so that we can call a compare to between two AInodes and decide which one has a lower F value.

ManDistance.java

The ManDistance.java class only had one function and that is to calculate the manhatten distance using two locations along with the height and width of the game map.

AIastar.java

The AIstar.java class is responsible for controlling all of the core data structures of the pathfinder. These consist of the following:

HashMap<FlatMap, AINode> closedList;
LinkedList<AInode> openList;
LinkedList<FlatMap> finalPath;

There is a specific reason each one of these data structures are used. The closedList is created using a HashMap data structure because we do not need to access the

data in any specific order instead we just want a fast lookup to check if a node is contained in the closedList(meaning it has already been checked).

A LinkedList is used for the openList because we wanted to have a very fast way of adding and removing nodes. Linked Lists are generally much faster than arrays when it comes to adding and removing elements.

Finally a LinkedList was used for the finalPath in order to have fast addition of locations in the order that the monster would move.