Game Engines

MGAE-E2013

IT-University of Copenhagen

Jacob Claudius Grooss, jcgr@itu.dk

May 22, 2013

1 Introducton

This report has been written as part of a project on the Game Engines E2013 (MGAE-E2013) course on the Games course at the IT-University of Copenhagen.

For the project, I have written a program in C# that visualizes one agent chasing another through the use of the A* algorithm. I have worked, and shared ideas, with Jakob Melnyk (jmel).

2 Features

In this section I will describe the features of the program: Pathfinding, visualization and user input.

2.1 Pathfinding

The program uses the A*[1] algorithm to simulate two agents, one chasing the other. The A* algorithm allows the chasing agent to find the shortest path to his target while avoiding obstacles and, depending on his speed, catch the fleeing agent.

2.2 Visualization

The map the agents are traversing is visualized through simple ASCII art, as seen in figure 1. Agent A is represented by the capital `A`, agent B by the capital `B` and the walls by capital `X`. The paths the two agents take are represented by non-capital `a` and `b`.

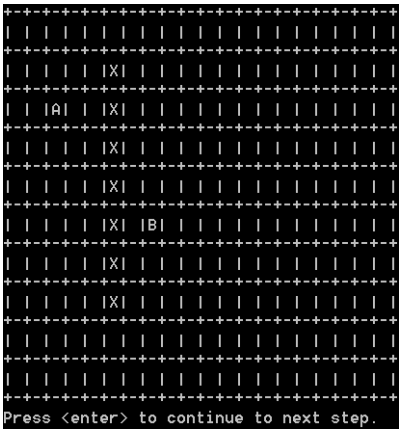

Figure 1: The initial visualization.

2.3 User input

The program accepts very simple user input. When the program is started, it lets the user specify the start position and speed of each of the agents. After that, it waits at every step for the user to press the `enter` button, upon which the simulation continues.

[1] http://en.wikipedia.org/wiki/A*_search_algorithm

3 Overview

In this section I will be giving an overview of the program. I will cover the classes of the program and how the pathfinding is done.

3.1 Classes

The program contains five classes. They are `Agent`, `Node`, `Map`, `AStar` and `Program`.

3.1.1 Agent

The `Agent` class represents the agents running around. It contains variables that specify the agent's position, speed and the path the agent currently is moving. The class has two methods, one for chasing another agent and one for fleeing from a given agent. The difference between the two is describes in section 3.2.

3.1.2 Node

The `Node` class represents the nodes the agents are walking on. Each node contains its x- and y-coordinates (used when the position of a node needs to be found without asking the map), a value that indicates if the node can be paased or not and values that are used by the `AStar` class for determining the best path to another node. It has methods for comparison with other nodes and a method for determining the distance to another node.

3.1.3 Map

The `Map` class represents the map through a 2D array of nodes. The class furthermore has methods for determining if a node (or x/y position) is within the bounds of the map, a method for drawing the map in ASCII art and a method for finding neighbours to a given node.

3.1.4 AStar

The `AStar` class contains the pathfinding algorithm, along with the method used to reconstruct the path when a route has been found. It is based on the pseudocode from the Wikipedia article about A*, but has been modified to accept a map as input as well, as it uses the map to find the neighbour nodes.

3.1.5 Program

The `Program` class starts the application and loads both the map and the agents. After loading it enters a loop, in which the agents are told to chase or flee from the other agent. When the loop ends, the program writes at which position the fleeing agent is caught, after which it terminates.

3.2 Pathfinding

Pathfinding is done in the following two ways:

- **Chasing** - When an agent is chasing another agent, it uses the basic A* algoritm to find the shortest route to the target agent.
- **Fleeing** - When an agent is fleeing, it goes through its neighbour nodes, looking for one that puts it further away from the chasing agent. It chooses the one that gives the most distance and then it checks the new position's neighbours the same. This iteration happens ten times. After the ten iterations, the agent uses the A* algorithm to find the shortest route from its current position to the position it found to be furthest away from the chasing agent.

4 Problems

A major problem in the program lies with the fleeing agent. Most of the time it will act in a smart way, until the chasing agent is on the same row or column as the fleeing agent. At this point, the fleeing agent will prefer moving either up or left (depending on if they are on the same row or column) until it reaches the edge of the map. Here it will move away from the edge, only to go back to where it came from.

This issue only occours when the fleeing agent has a speed of 1. It could be prevented by a better prediction of where a certain node will lead it, instead of blindly going for the one that puts it the furthest from the chasing agent.

5 Conclusion

The program I have written shows simulates one agent chasing another through the use of the A* pathfinding algorithm. It allows the user to specify the start position and speed of the two agents and will visualize each step of the chase.

5.1 Future works

While the program can do simple pathfinding/visualization, there are a lot of things that could be changed:

- **GUI** - The GUI could be improved tenfold (or more). Currently it just consists of printing to the console. While it gives a basic view of what is going on, it is by no means pretty or user-friendly. Changing it to include graphics would be a huge improvement and would also make user interaction (see next point) a lot easier.
- **User Interaction** - As it is, the user is only allowed to interact with the program in a very limited way: At first by selecting the start position and speed of the agents, then by controlling the visualization by pressing `enter`. Letting the user open/close nodes or changing the position/speed of the agents while the program is running would make it more interesting.
- **AI** - When it comes to movement, the chasing agent is doing fine, but the fleeing agent has some issues (as described in chapter 4). The method for finding a suitable path away from the chasing agent would be an improvement worth spending time on developing.
  On the same note, implementing states for the agents ("idle", "chasing", "fleeing", etc.) would also be worth looking into, as it would better simulate how beings act.
- **Terrain types** - All nodes (excluding the closed ones) are treated equally during pathfinding. Introducing different types of "terrain" (water, forest, plains, mountains, etc.) would make the simulation more interesting, as the agents would have to take the terrain types into account. This would require the use of a graphics-based UI instead of the current text-based one, but would be well worth the time.

6 Appendix

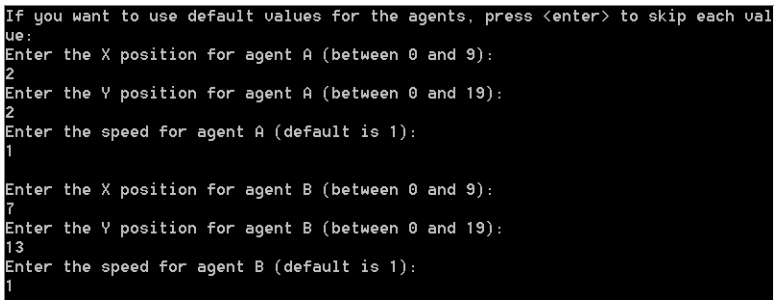

Figure 2: Choosing the start position and speed of the agents.

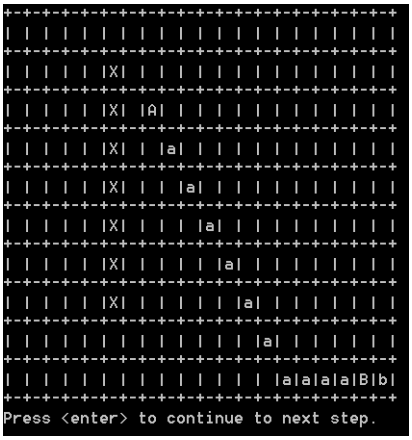

Figure 3: A few steps into the simulation.

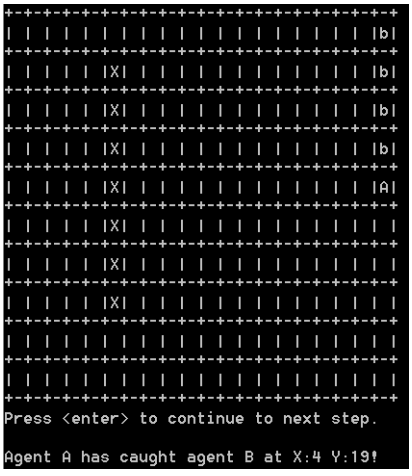

Figure 4: The end of the simulation.