Jake Lucic

2103435  AI for Games

Spike report spike 12

**Number:** Spike 12

**Spike Title:** Graphs and Search

**Personal:** Jake Lucic (2103435)

**Context:**

Graph can be used to represent many aspects of a game, and selecting and using the right type of graph search is critical to finding solutions to problems in games.

**Knowledge/Skill Gap:**

The developer needs to be able to use appropriate graph search algorithms to solve problems for characters and controllers game situations.

**Goals:**

Clearly demonstrate the appropriate use of the Dijkstra’s (search for item) and A\* (search to position).

• Modify the graph search lab code, or create your own simulation.

• Add in a simple moving agent to moves to each way-point in a successful graph search result.

• Be able to demonstrate either search-for-item or search-to-point examples.

• Make sure your agents (or their graph searching algorithms) correctly consider wall, mud or water tiles in the map.

• Display path cost for comparison

• Clearly demonstrate the need for different search algorithms

**Technologies, Tools, and Resources used:**

In this task, the technologies used are listed below:

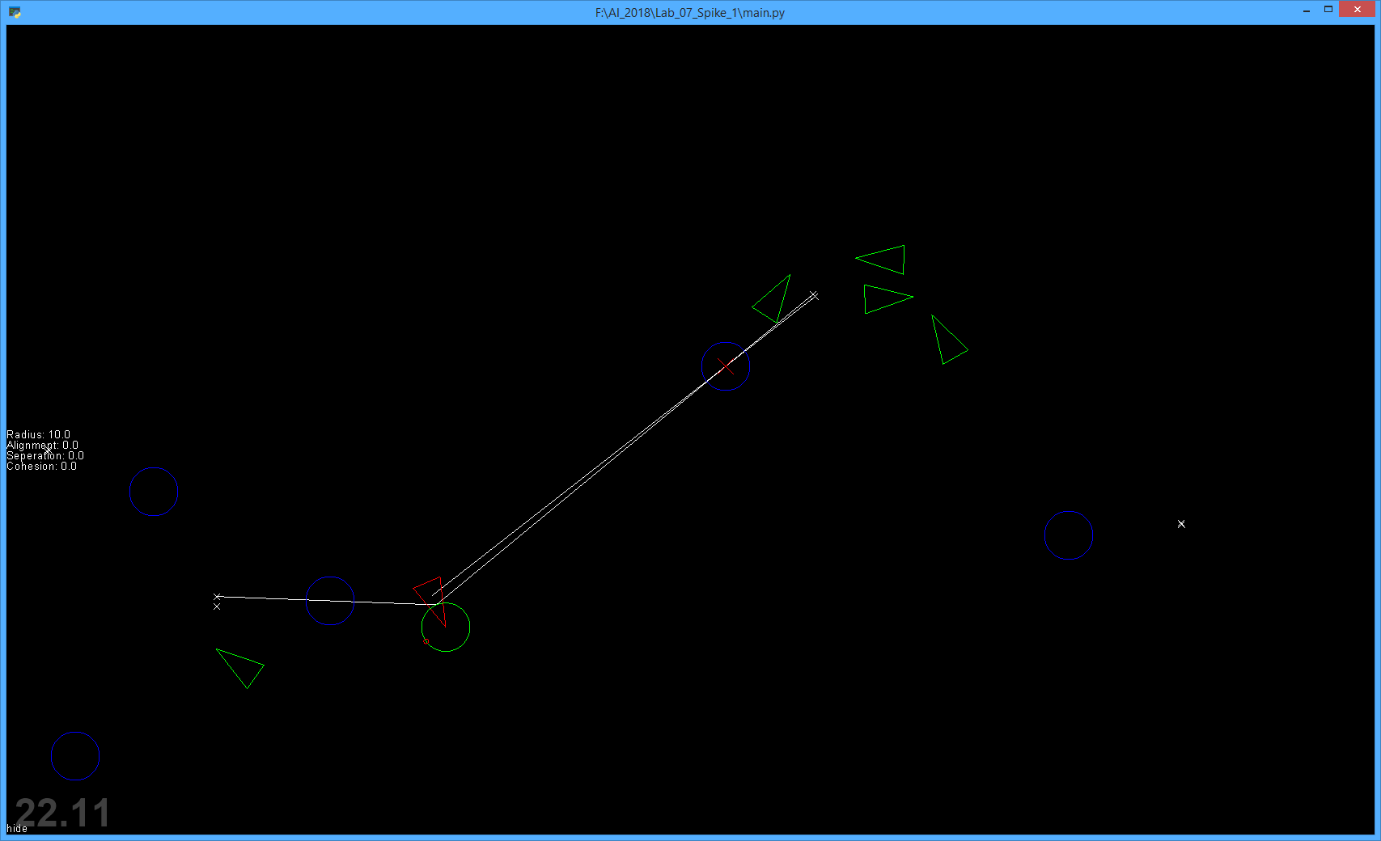
* Simple Code base, the not-yet-functional code that required fixing.
* Python IDLE v3.6.4 / Python language / PyCharm
* Path Following, Wandering Lecture
* Tactical Action Selection and Predictive Shooting Lecture

**Tasks undertaken:**

* Read the code base and understand what each function does
* Study the different key words and variable names
* Follow the lectures on Path Following, and Tactical Action Selection
* Approach Task using similar process below

**Process:**

I approached this task by creating objects that the Agent could hide behind. I followed that they should be made as circles. The reason as to why it should be a circle is the fact that to get your Agent to hide behind it, having a radius and a centre point, makes the math to hide behind the obstacle a lot easier.



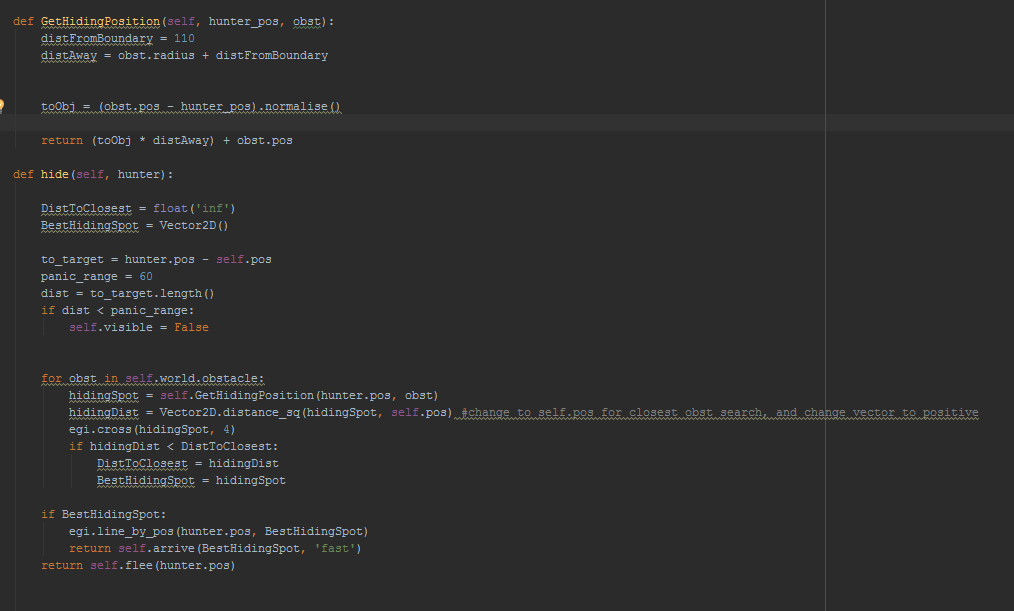
In this image, you can see the dark blue circles, these are the obstacles that the hiding Agents (Green) hide behind. The hunter agent is the red agent, and it is simply on wander mode.

The white lines stemming from the Hunter, points to target hiding positions, there is one for each agent.

As the Hunter and the Hiders move, the hiding position also moves.

The Hiders look for the closest obstacle to themselves, and then by using the function getHidingPosition, this sets a distance from the radius of the circle. This is then updated every update cycle so it constantly returns the “hidden” side of the obstacle from the Hunter Agent.

Using Slight Tactical Evaluation as an approach to this task, my selection was for each of the Hiders to select the closest obstacle to them. Although, I also found a way to get all of the Hider Agents to hide from the furthest obstacle away from the Hunter Agent. This is explained in full in the image on the next page



**Here is the code for GetHidingPosition and Hide Agent mode**

**Extension**

I also was able to get the Hiding agents be able to be swallowed by the Hunter Agent simply by creating a collision circle, comparing if the Agent type was a Hunter or a Hider, then if they entered that circle, they would simply not be visible (This is also visible in the hide mode function. Although if I had to approach this task again, especially to optimise – I would just remove them from the list, so that they would not update/render in the next cycle.

**Added Controls:**

H: Adds the hunter into the world. Please place only one.

i: Adds Hider agents.

0: Sets the Hider Agents into Hide mode. Please have a hunter (H) before selecting this option