**Spike:** 04

**Title:** Graphs, Search & Rules

**Author:** Ben Holmes, 103024841

**Goals / deliverables:**

1. Tic-tac-toe code modified to represent the game state as a graph.

2. An AI that randomly searches the graph

3. An AI that improves the efficiency of a basic random search

4. An AI that improves the effectiveness of a basic random search

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

* Visual Studio Code
* Python 3.12.2

**Tasks undertaken:**

* Download and install Visual Studio code
* Download and install Python
* In visual studio code, install python extensions
* Run code

**What we found out:**

For deliverable 1, I added a class called Board\_State that is the node of a graph with relevant information stored.

A computer screen shot of text

Description automatically generated

This is the initialisation method that takes the current\_player, a board list, and the previous\_move, adds them to its variables and then checks how many moves have occurred to store that information.

A black background with white text

Description automatically generated

This is the function used to provide current board state as the Board\_state class.

This deliverable is necessary for the lab as you need to be able to store the board state and useful information in order to effectively use the graph method.

For deliverable 2 I added this function

A computer screen shot of a program code

Description automatically generated

It grabs the current board state and stores it in the list, it then does a random move in that board\_state, stores the new board state, checks to see if there is a victory, and loops back if not. It will then pass the entire path as the result.

A screen shot of a computer code

Description automatically generated

This is the function that uses that random path function.

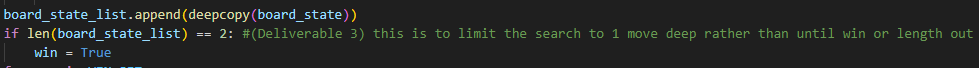
This deliverable is the most basic implementation of the graph method, its very basic, but is the foundation for the rest of the deliverables to build on in order to use the graph effectively and efficently.

For deliverable 3 the do\_move function of the Board\_State class was changed to switch between the two player’s moves (the if else statement referring to self.node\_player). This was so that it could account for the opponent making moves between its moves and to prevent illegal moves causing the ai to think it would win.

A computer screen with text

Description automatically generated

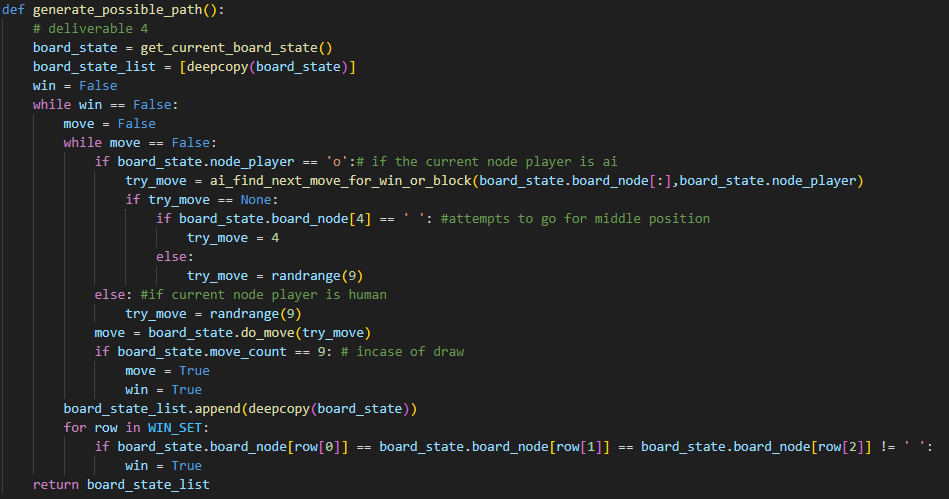
Also these two lines were added in the generate\_possible\_path\_stupid()



To limit the search to only one move deep to speed up the processing.

Deliverable 4 was the main part of the spike, where the graph was actually created and used, at least partially. This was so that the ai could improve its effectiveness

3 functions were created



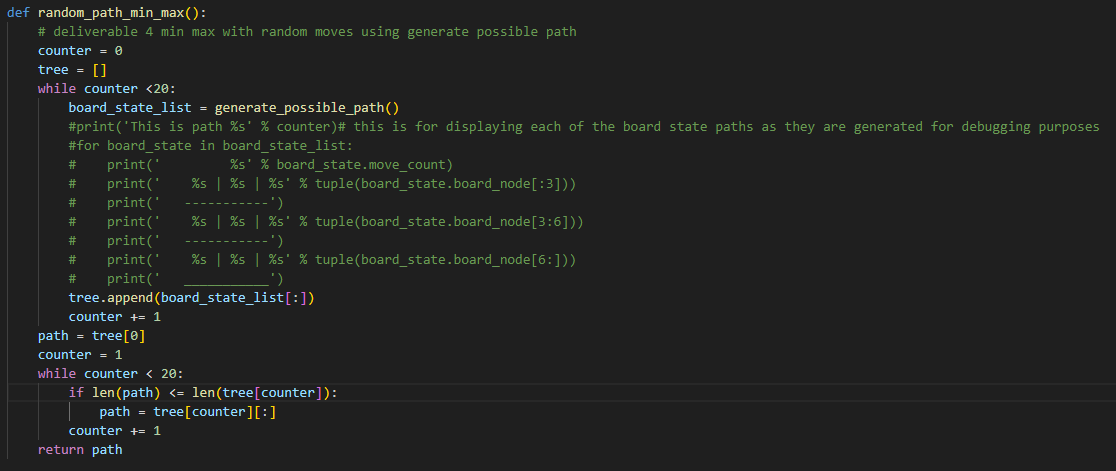
Generate\_possible\_path was the core processing function, creating each of the paths for the tree, it used random moves for the human opponent, would attempt to go for the centre position if it was open, and would use ai\_find\_next\_move\_for\_win\_or\_block to help decide which move to make for victory

A screenshot of a computer

Description automatically generated

This function checked all available moves for the ai to see if it resulted in a victory and returned the move if so, and if there was no possible victory it would cycle through again as if it were the opponents next turn to see if they had a victory and would block that move if it existed.

The third function was random\_path\_min\_max



Which generated a 20 long list of paths using generate\_possible\_path, it then searches through each path to find the shortest path, and then passes that path as the result.

A black background with white text

Description automatically generated

It then takes the first move of the path and passes that on to the board

This deliverable is the core of the graph methodology where the ai uses the minmax algorithm along with searching for a possible win/block move to do in order to make and efficient and effective ai.