Artificial Intelligence - Homework 1

The main directory contains 2 subdirectories, one for the chess game and another one for the tiles game. The two main files are named ***main.ipynb***

The dependencies are **python-chess**, and the libraries used in the **Jupyter Lab**

Tiles

For the tiles game, I wrote a simple implementation of the game using **numpy** arrays.

The goal state is the board with the empty cell in the first position like this:

0	**1**	**2**
3	4	5
6	7	8

Algorithms implemented

The algorithms implemented right now are:

- Depth-First Search
- Breadth-First Search
- Depth-Limited Search
- Random Search
- Best-First Search
- A*

For the **Best-First** and **A*** algorithms you need to provide also one heuristic. The ones available are:

- Difference-based heuristic (*how many tiles are out of position?*)
- **Manhattan Distance** heuristic (see also)

To improve efficiency and avoid time wasting I also used a function to check if a specific board is solvable (any 8-puzzle instance is solvable if the number of inversions is even)

Comparison

I tested the execution speed mostly for the 3x3 grid cases.

For the solvable instances (even the more complex ones) the A* and the Best-First are fast and can a solve in milliseconds in the best cases and ~6sec in the worst case.

Other algorithms for complex instances don't solve in a reasonable time (>10 min execution time) and in simple cases (not too deep) they solve in milliseconds.

Modularity

To ensure modularity I created an abstract class that contains the methods that every Search Algorithm implementation should provide and then you can easily switch between different algorithms.

Chess

I want to remember that ***only*** the code used for the board is imported and not the minimax algorithms as said in the previous correction of the homework.

Same Heuristic against

The two agents played against each other 100 games for each configuration.

Now it's possible to play smoothly even at depth 4 due to several optimizations made to the algorithm such as Alpha-Beta Pruning and other techniques to speed up Python. I used 3 different Heuristic functions:

1. The **SimpleHeuristic** was just a simple difference calculation so the agent tries to maximize his number of pieces without any interest in the type
2. The **WeightedHeuristic** introduced the concept of weight so each piece has a weight to prioritize for example the number high valued pieces like the king or the queen
3. The **CheckHeuristic** adds to the previous heuristic the concept of check and checkmate so it will prioritize the check and checkmate willing state

Report

depth\Heuristic	SimpleHeuristic win Ratios	WeightedHeuristic win ratios	CheckHeuristic win Ratio
1	W=7% B=6% D=87%	W=7% B=8% D=85%	W=11% B=8% D=81%
2	W=10% B=7% D=83%	W=7% B=4% D=89%	W=11% B=6% D=83%

Different heuristics against

The two agents played against each other 100 games for each configuration.

I used the previous 3 heuristics and tried to make them beat each other.

Report

(Added the row for the 3 vs 3 depth, and updated stats)

depth\ Heuristics	SH vs WH	SH vs CH	WH vs CH
1	SH=9% WH=6% Draw=85%	SH=8% CH=10% Draw=82%	WH=9% CH=7% Draw=84%
2	SH=6% WH=7% Draw=87%	SH=4% CH=6% Draw=90%	WH=10% CH=5% Draw=90%
3	SH=10% WH=9% Draw=81%	SH=8% CH=5% Draw=87%	WH=3% CH=7% Draw=90%

Modularity

To ensure modularity I created an abstract class that contains the methods that every Heuristic implementation should provide and then you can easily switch between different heuristics.

Even in the main program modularity is strongly encouraged and it's easy to switch from one board game to another just by providing a Board instance that gives some basic methods.

Each Heuristic class implements a method called `evaluate` that given a state returns the corresponding evaluation of the state.

The algorithm logic for minimax was moved onto the `Agent.py` file as suggested in class, so now the agent computes the entire move on itself and uses the heuristic only for the evaluation.

Representation

The `python-chess` library provides the board implementation and a list of legal moves that can be performed on each turn.