Introduction to Artificial Intelligence (236501)

Winter 2012

Homework Assignment 2: Quarto

Assignment Goals:

- To understand the dynamics of a non-cooperative multi-agent environment.
- To review the pros and cons of multi-agent search algorithms.

Notes:

- Due date: 2/1/2012.
- This assignment is for submission in pairs or singles. No trios will be allowed.
- **Copying of any kind will not be tolerated, and will result in expulsion from the course.**
- Feel free to ask questions via email: omerlevy@cs.technion.ac.il

Problem Setting:

Quarto is a two-player zero-sum game, in which players place pieces on a 4x4 board in order to achieve a pattern (a "quarto"). Each piece has 4 binary attributes (height, shape, color, and fill) and each of the 16 pieces is distinct from the others. In each turn (excluding the first and last), the current player places a piece and then selects the piece his opponent will place in their turn. The first player to create a row, column, or diagonal with at least one common attribute wins.

Online implementations of Quarto are available, such as: http://quarto.freehostia.com/en/

In this assignment, you will design and implement a Quarto game-playing agent. This agent will provide a single action each turn. As with any AI assignment, your agent will be subject to time constraints:

1. **Setup Time** – The time it takes your agent to prepare for the game.
2. **Turn Time** – The time it takes your agent to decide upon its turn.

Setup time and time between turns are not interchangeable; i.e. time cannot be accumulated. Since these factors are not constant, you will have to design an **anytime** agent: one that provides better actions as more computation time is given.

Assignment Instructions:

1. Understand the game space, and write a short theoretical analysis about its characteristics. Points to consider:
    - What is the branching factor?
    - How does the branching factor change throughout the game?
    - Which actions would you consider as “threats”?

2. Design and implement at least **three** different evaluation functions for the following tasks:
    - Utility Assessment
    - Node Reordering
    - Quiescence Search

    There must be at least **one** function for each task. Explain the intuition behind them.

3. Implement the following extensions to the alpha-beta algorithm:
    - **Node Reordering**
    - **Quiescence Search**

    Further detail about these algorithms is provided in the next section.

4. Write a **comparative** empirical analysis of how alpha-beta’s extensions perform in Quarto under an anytime setting. To do so, you will need to create five game-playing agents:
    - Dummy (provided)
    - Anytime alpha-beta (no extensions)
    - Anytime alpha-beta with node reordering
    - Anytime alpha-beta with quiescence search
    - Anytime alpha-beta with node reordering and quiescence search

    You may then run the following experiment:
    - For each pair of agents, run a series of Quarto games with the following turn time limits: $t(seconds) = 0.25, 0.5, 1, 2, 4, 8, 16, 32, 64$.
    - Measure wins, losses, and ties.

    How would you use these findings to improve the algorithm?
    Your analysis should show which algorithm performs better and explain **why**. When writing your analysis, consider how the turn time parameter affects each agent’s performance and explain **why** it does so. How would you use these findings to design a better anytime agent?
    Note that setup time is irrelevant for each of the above agents, since no setup is required. In addition, statistical significance is not required, since your results are not based on a sample.

5. In accordance with the conclusions from your research report, you will implement a game-playing agent for Quarto. This agent will compete against agents designed by your peers in a tournament. Winners of the competition will be awarded **bonus points to their final grade**.
   You are not limited to the algorithms presented in these instructions, and are encouraged to use your imagination and creativity. While your assignment's grade will not be affected by how well your agent does in the competition, malfunctioning agents will result in point reduction. Participation is mandatory.

Alpha-Beta Extensions:

Node Reordering

As shown in class, the order in which child nodes are expanded can affect the amount of pruning that alpha-beta performs, and ultimately, the search depth. This extension orders each set of child nodes according to a given evaluation function, deciding which will be extended first, second, and so forth.

Quiescence Search

Some game states may be classified as "unstable". In chess, for example, a state in which the king is in "check", or when a threat to a high-value piece (such as the queen) is in place, may be interpreted as an unstable state. These states are unique in the sense that the balance of power may shift within one or two turns if defensive action isn't taken. It is therefore preferable to keep on expanding such states, at the expense of ignoring other, more stable, states.

Provided Code:

The following code is provided for your convenience:

- `game.py` – An interface for games.
- `game_agent.py` – An interface for game playing agents.
- `game_runner.py` – Runs games between agents.
- `quarto_game.py` – An implementation of Quarto.
- `example.py` – An example of basic game-playing agents playing Quarto.
- `alpha_beta.py` – A basic implementation of the alpha-beta algorithm.

<u>Submission Instructions:</u>

Dry: Research Report

- This is the most important part; it will determine your grade.
- Limit your report to **8 pages**.
- The report should contain the following sections:
  - Introduction – Where you will present your theoretical analysis of the game and the evaluation functions you will use to play it.
  - Empirical Results – Where you will present the experiments' results. Use graphs or tables to present your results, and add textual explanations where necessary. Analyze your results and try to draw deep and profound conclusions from them.
  - Agent Proposal – Where you will explain how your competition agent works, and argue why it should perform well (and kick the other agents' @$$).
- Do not write code in the report.
- Submit the printed report to the course's inbox next to the coffee cart.
- Submit an electronic version of your report and all **experimentation code** to the "Research Report" assignment in the course website.

Wet: Competition

- You will write your agent in the Python programming language. The agent must inherit from the `GameAgent` class in the provided code.
- No preprocessed data is allowed. No use of network resources is allowed. Any attempt of sabotage or cheating will be dealt with harshly.
- Preprocessing is allowed in the `setup()` function, within the given time limit.
- You will submit (electronically) a zip file containing:
  - `submissions.txt` – Name, ID, email (comma delimited, each student in a separate row)
  - `agent.py` – Contains the implemented agent. Must contain exactly one class that inherits from `GameAgent`.
  - Other Python files, in the same directory (no sub-directories).
- Make sure that your code assumes a flat hierarchy; i.e. that the code provided by the staff is in the same directory as yours.
- The submission should **not** contain any of the provided code.
- Submit the **competition code** to the "Competition" assignment in the course website.

Good luck!