Ain Shams University
Faculty of Engineering
CESS Program
CSE386: Artificial Intelligence

Project Description

Project Name: Intelligent Chess Game.

Grading weight: This project accounts for 20% of the course mark and will be graded out of 20 marks (15 marks for the implementation and 5 marks for the report). A bonus up to 5 marks to be added to **the project mark** will be given for implementing at least **two** of the *bonus* features suggested below. Please do not be tempted to implement all the bonus features, as this will cost you too much time. Also note that these bonus marks will only affect the project marks.

Project Overview: The goal of this project is to implement a Chess game using the Minimax algorithm with alpha-beta pruning. This requires you to design an appropriate chess board evaluation function to be used as the algorithm's utility function. Your game should allow a human player to play against your algorithm. The algorithm should use a depth-first strategy when exploring the game tree to ensure efficient memory usage.

The project can be implemented using any high-level language of your choice (C/C++, Java, C#, python, etc.). It can also be implemented using Prolog or a combination of Prolog and any other high-level language (Prolog code can be called from programs written in other languages).

The game should allow either players to start first. The game can either be console based or GUI-based (preferred but not required and not counted as a bonus feature). In case your game does not support various difficulty levels nor a "play now" button (see bonus features below), it should be adjusted to respond with a move in 30 seconds or less.

A chess competition between all teams willing to participate will be held after the submission deadline. Competition rankings will not affect your marks in any way.

Team Size: 1 to 5 students (preferably 2 or more)

Important Plagiarism notice: You have to write your own code from scratch. Projects based on others code will receive a grade of **zero** in the entire project and report (even if the code is heavily re-factored/modified, etc.). Examples of such sources include (but is not limited to) code coming from the following sources: other teams, previous year projects, open-source software, Internet, tutors, etc.

Project Deadline: Thursday December 21st 2017, 11:59 pm. Projects and reports should be submitted via the LMS (only one student from each team should submit the project). *If needed*, a meeting with the course TA to evaluate your work will be scheduled in the following days.

Project Report: In addition to your team member names, the report should include:

1. A brief description the game and of your implementation including any bonus features included.
2. A detailed description of the utility function(s) used by your algorithm.
3. A user guide with snapshots.

4. A summary of how the work was split among your team members (who did what exactly).
5. Any additional documentation you might find useful (including code documentation, descriptions of difficulties encountered, tricks used, etc.).
6. Optionally, you can include a section about your experience working on this project. This section will NOT affect your grade in any way.

***Bonus* features:**

1. Implement an iterative-deepening variation of the alpha-beta algorithm to support both: a "play now" button and a time limit setting. If a "play now" button is supported, clicking it would force the game to respond immediately with the best move found so far. Also if a time limit setting is supported, the game will be required to respond with the best move found when the time limit has expired (in each turn).
2. Support various difficulty levels corresponding to different game tree depths or to different utility functions. Unless you have implemented the bonus feature #1, at least one the difficulty levels should be guaranteed to return a move in 30 seconds or less.
3. Support a verbose mode in which each time the computer plays, the following information is displayed: The maximum depth, the average branching factor of all nodes explored, the number of leaf nodes evaluated and their utility values, the number of cutoffs that happened and the level at which each cutoff occurred.
4. Support an alpha-beta optimization mode allowing the game to sort the children of each node according to a fast utility function to maximize the chance of alpha-beta pruning
5. Support a networking mode allowing two instances of the game (running on the same machine or on different machines) to play against each other without any human intervention. This function will be particularly useful if multiple teams agree on a communication protocol (or use a standard one) allowing their games to play against other.
6. Support game saving and loading in addition to starting the game from any board state (not necessary the standard chess state)

What and how to submit: Submit a compressed folder (zip or rar) on the LMS. The folder must contain: 1) All your code and 2) Your report (PDF).