UNIVERSITEIT ANTWERPEN

Academiejaar 2023-2024

Faculteit Toegepaste Ingenieurswetenschappen

5-Artificiële Intelligentie

Lab 4: Chess Project

Arne Troch, Bavo Lesy & Jens Duym

Bachelor of Science in de industriële wetenschappen: Electronica-ICT

Universiteit Antwerpen

Premise

In this lab, you will be tackling the game of chess. Chess is an interesting challenge for the field of AI that has prompted many researchers to have a go at it. The complexity of the game together with the massive state space renders many basic AI-techniques useless. For human players, chess is more a game of intuition. This intuition allows humans to assess chess positions in the blink of an eye. Computers do not natively have the ability to do this, placing them at an immediate disadvantage. These difficulties make chess an extremely interesting playground for creating and developing AI-techniques. For the next few weeks, you will have a go at this playground. Extend your basic knowledge of AI by looking into solutions to this challenging problem, and then create your own version of them!

Deliverables

This project will be evaluated based on a report as well as an oral defense during the examination period. Both aspects have the same weight towards your final score for this lab (50/50).

Report

Use this report to show what you have learned:
- Explain the techniques you used; where did you get your inspiration from, why do they work, how could you make them better?
- Explain your implementation of these techniques.
- Explain other ideas that you had but that maybe did not fully work or you were not able to implement due to time constraints, show us your creativity!

This report does not need to be too long (try not to go over 5 pages). Don't write entire books about every single detail or line of code. Use images to aid you in your explanation! ***The deadline for the report is 29/01/2024.***

Oral defense

You will be asked to give a short presentation (about 5 minutes) highlighting the parts about your implementation that you find the most interesting. Afterwards we will spend +- 10 minutes discussing what you presented. This will not be too detailed or technical, we just want to see if you understand what you implemented. We also expect you to upload a ZIP-file containing your code.

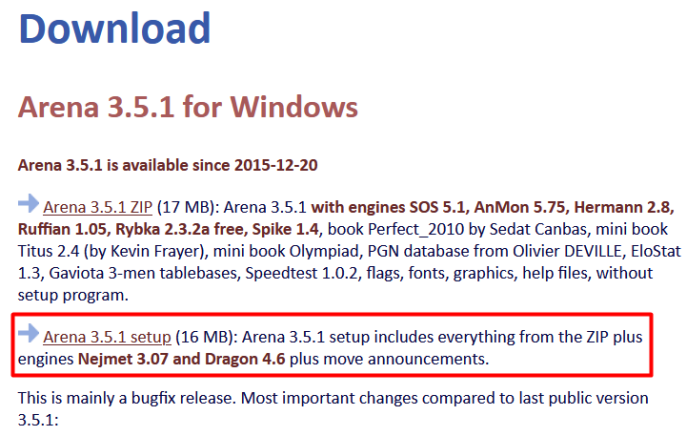
Prerequisites

1. In this lab we will use the python-chess library to implement the chess logic for us. You will need to install this using *pip install python-chess* (if you use conda, you can create a new environment for this lab and install it there).
2. Install the provided library using the command "*pip install -e .*" from the top level folder.
3. We have provided you with a framework that abstracts away some of the details of this library. You can download the ZIP-file containing this framework on Blackboard.

   The first thing you need to do after unzipping is edit the *run_engine_conda.bat* file (if you use conda that is). This file will allow your agent to interact with a chess GUI (which we will

install later). Open the file and replace "YOUR_ENVIRONMENT_NAME" with the name of your conda environment (if you don't use conda you don't have to do anything).

4. We will also use a graphical interface so that you are able to play against your own agent (or see it in action against other bots). You are free to choose one to your liking, we provide a small tutorial to the Arena GUI below:
   a. Download and run the installer from the link provided above



   b. Open Arena and go to *Engines -> Install New Engine…*

   c. Navigate to the folder where you unzipped the framework and select the .bat file you just edited (or the unedited one, depending on if you use conda or not). Select type UCI when prompted.
   d. Start a new game (*File → New*) to play against your agent! The agent we implemented in the framework is very basic and naive, it is up to you to make him smarter!

Lab

Time to dive into the code. The framework we provided has a generic structure which you can use to develop your own agents. You are not obliged to follow this structure, though it might help get started. Below, we provide a quick rundown of the main structure with the most important parts. The code itself is also documented quite extensively, so don't be afraid to go through the code to understand what is going on!

Agents

Agents contain the logic to search through the game tree (this is where you would place a minimax). Agents use a Utility to give values to positions, these values are then used to select the best action according to your search tree.

Utilities

Utilities contain the logic to give a value to a state of the game (heuristics). This value determines how advantageous a position is for either side. This value can be calculated very naively (#black pieces vs #white pieces) or more advanced (taking into account things like position, state of game, importance of pieces, …)

Engines

The *uci_engine.py* script handles all the background stuff to allow you to interact with a GUI. You probably never have to touch this file.
The *example_engine.py* script shows how you can use this script to actually interact with a GUI. This script also shows how you should use the Agent and Utility classes.

Examples

This folder contains three example scripts to show you how you can use (or train) your agent. The scripts show how to play against yourself, how to play against the stockfish engine and how to import game from *pgn* files. To play against stockfish, you have to install it from https://stockfishchess.org/ and reference the path where you installed it in the *play_stockfish.py* script.

We suggest you take your time going through all of the code, keeping in mind each scripts' general purpose as we describe it here. For each aspect (*agent, utility* and *engine*), we provided an example to show how you can create your own implementation of them. The examples folder shows how you can combine your agents and utilities to create a functioning whole. Once understand how it all ties together, you can quickly and effortlessly start creating your own implementations of *agent* and *utility*.

You do not HAVE to make use of any of the code/structure we provided. If you want to do your own thing, go for it. The only thing we expect in the end is a program that is able to communicate using the UCI protocol (like the example in *chess_engines*).