UCU. APPLIED SCIENCES FACULTY

Problem set guidelines

1. Should you have any questions regarding this homework, please post them on Piazza. Chances are your classmates have them too. By posting and answering questions on Piazza, you help your peers to get better understanding of the class materials.
2. You can discuss the problems with your classmates but do not share your code or the answers and do not use someone else's solutions.
3. You can submit writing assignments in any form convenient for you. It could be LaTeX, MS Word, PDF or image. Please make sure it is of good enough quality.
4. For the coding assignment, please submit your **ai.py** file along with any additional module files if it requires any.

Technical details

1. The coding assignment uses Python 3.6. You can use either the official distribution or Anaconda, which contains the majority of the required packages pre-installed for you.
2. Clone the following GitHub repository to start working on the assignment:

   ```
   https://github.com/YuriyGuts/ucu-ai-checkers
   ```

3. It is recommended that you create a **virtual environment** for your coding assignment.
   If you are using Anaconda, run:

   ```
   conda create –n checkers python=3.6
   activate checkers
   ```

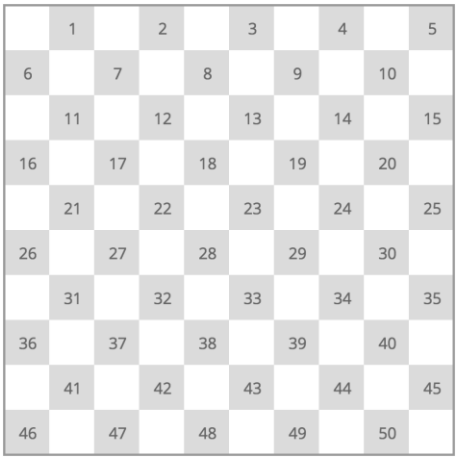
   If not, you can install the **virtualenvwrapper** package and run the following:

   ```
   mkvirtualenv checkers --python=python3.6
   workon checkers
   ```

1 Building a Game AI for Checkers

[70 points]

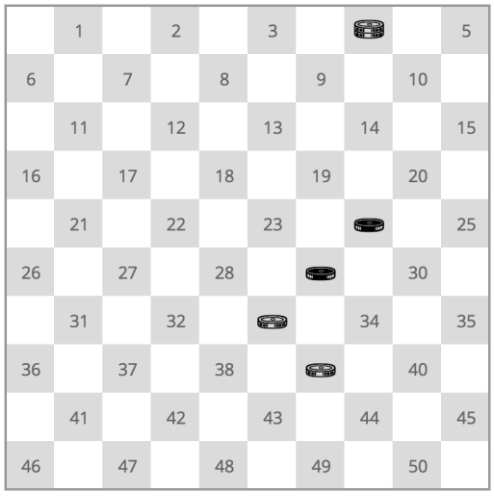
In this homework, you are going to develop an AI algorithm that plays the game of international checkers. Please read the rules of the game carefully before you proceed. Checkers rules can vary from country to country, and you might be familiar with some other version.



We are going to use the standard checkers notation as shown on the figure on the right.

Your AI must be implemented as an HTTP API service that receives the current state of the board in the request, and must respond with a valid next move the specified player should make.

Here is an example of a request your AI will receive for the board below, as well as an example of a response:



Request:

```
POST http://localhost:5000/move
```

Request body:

```
{
    "board": {
        "4": {"player": "white", "class": "king"},
        "24": {"player": "black", "class": "man"},
        "29": {"player": "black", "class": "man"},
        "33": {"player": "white", "class": "man"},
        "39": {"player": "white", "class": "man"}
    },
    "playerTurn": "white"
}
```

Response:

```
{
    "type": "ForwardMove",
    "startIndex": 29,
    "endIndex": 28,
}
```

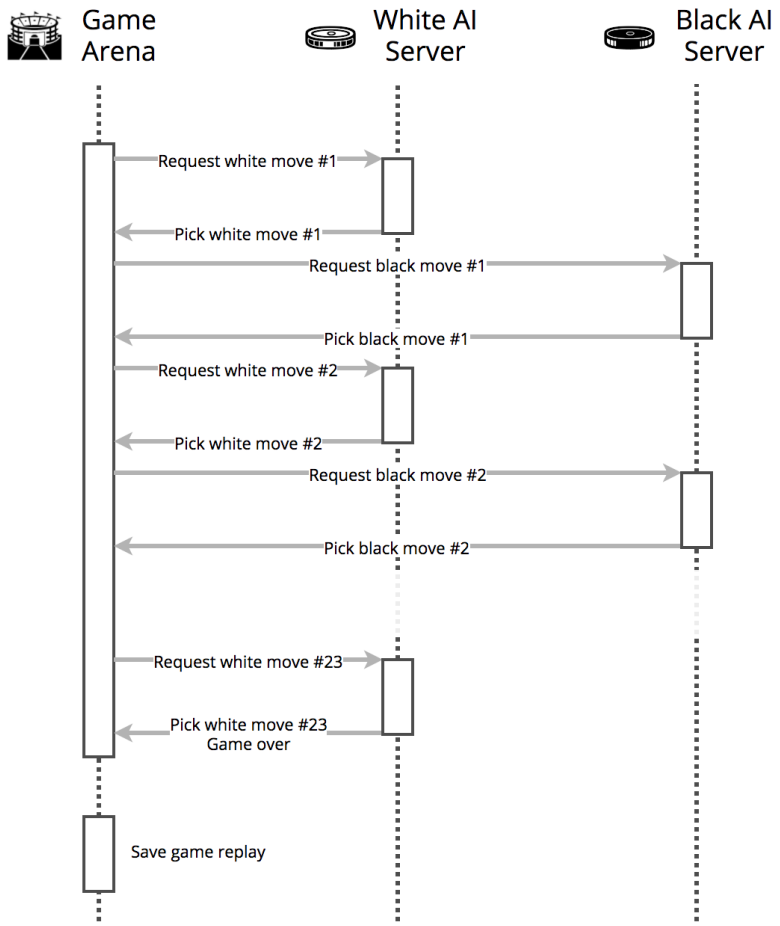
1.1 Development Environment

The logic for identifying valid moves, as well as other utilities for handling the game board are already implemented in the **libcheckers** Python package.

The **ucu-ai-checkers** repository you cloned before contains two pieces of software.

The first one, **ai-server**, is an example implementation of the AI service. By default, it contains a very simple AI that picks a random move every time, therefore having almost no intelligent behavior.

The second one, **game-arena**, is an environment that allows you to test different AIs and have them play against each other. The arena can also work in replay mode and play back one of the games it has observed and saved before.



The arena can run one or more games between the same opponents, display game statistics, and optionally visualize the game progress on a GUI window (check `python arena.py -h` for details).

1.2 Getting Started

To set up your development environment, run this in your virtualenv:

```
cd ucu-ai-checkers
pip install -r requirements-base.txt
```

If you use a Unix-like operating system (Ubuntu, macOS, etc.), install the Unix-only dependencies as well:

```
pip install -r requirements-unix-only.txt
```

Now, run the AI server on port 5000:

Unix (Anaconda or no virtualenv):

```
cd ai-server
uwsgi --ini uwsgi.ini --socket 0.0.0.0:5000
```

Unix (virtualenv):

```
cd ai-server
uwsgi -H $VIRTUAL_ENV --ini uwsgi.ini --socket 0.0.0.0:5000
```

Windows:

```
cd ai-server
python app.py 0.0.0.0 5000
```

After that, test that the arena works:

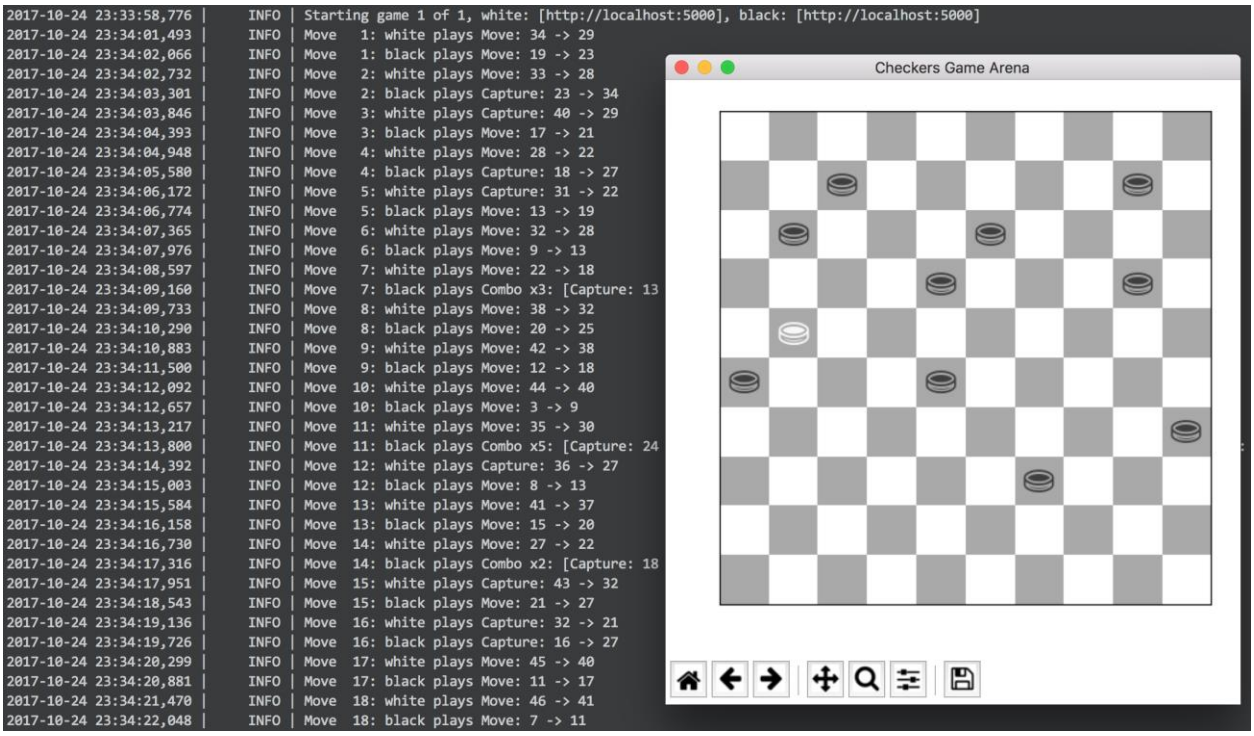
```
cd game-arena
python arena.py compete --gui http://localhost:5000 http://localhost:5000
```

If all goes well, you should see a log of moves with no warnings, as well as a visualization of the checkers board (see figure below).

Hint: on macOS, you may get the following error:

```
RuntimeError: Python is not installed as a framework. The Mac OS X backend will not be able to
function correctly if Python is not installed as a framework. See the Python documentation for
more information on installing Python as a framework on Mac OS X. Please either reinstall Python
as a framework, or try one of the other backends. If you are using (Ana)Conda please install
python.app and replace the use of 'python' with 'pythonw'. See 'Working with Matplotlib on OSX' in
the Matplotlib FAQ for more information.
```

If you do, run `echo "backend: TkAgg" > ~/.matplotlib/matplotlibrc` to resolve this.



After the game finishes, you can replay it by running:

```
python arena.py replay game-xxxxxxxx-xxxxxx-y.replay
```

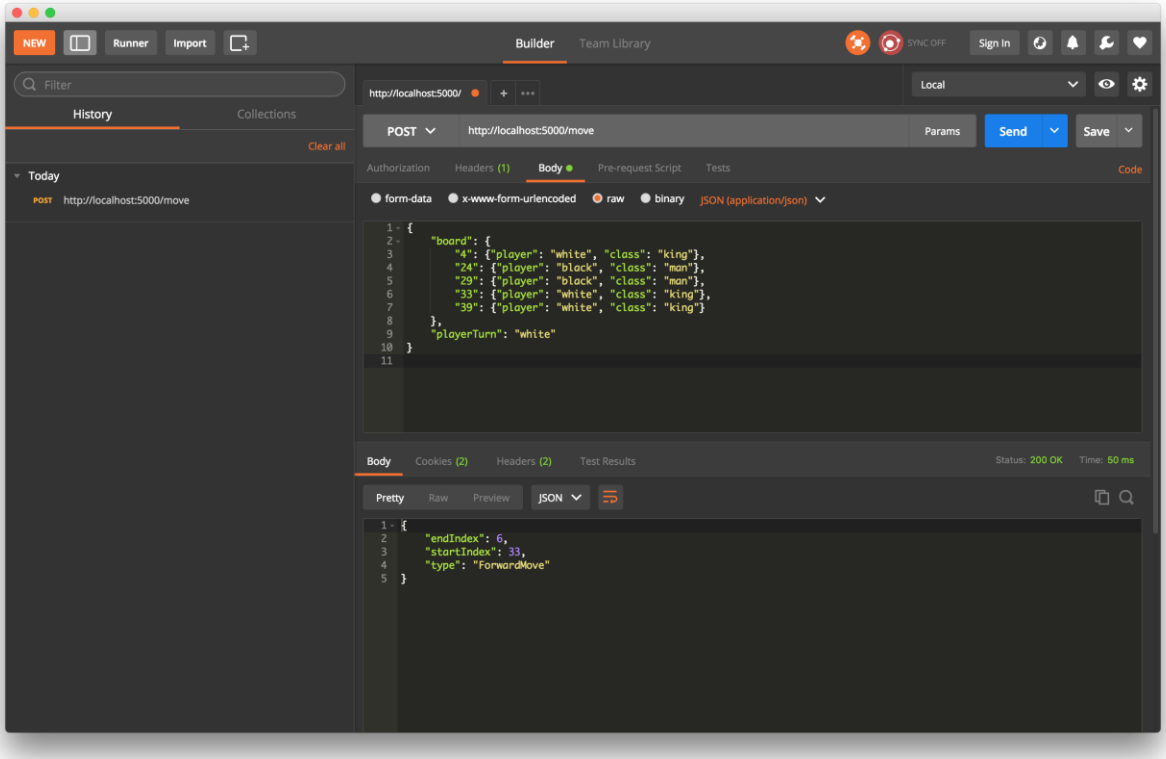
1.3 Developing the AI

The only file you need to change and submit is **ai.py**. You can also split your AI implementation into multiple modules, but make sure you submit all of them. You can run **app.py** in the debugger and use the Postman extension for Chrome to try out different HTTP requests for your server.

The three functions that will be the most useful for you are board.get_available_moves, board.get_player_squares, and move.apply. However, you are welcome to utilize any other functions from **libcheckers** if that helps you develop a better algorithm.

Warning: the arena will ignore your move and pick a random move instead, if:

1. Your server returns a non-OK HTTP status code (200).
2. Your server returns an invalid move.
3. Your server returns a malformed response entity.
4. Your server takes longer than 10 seconds to process the request. Don't search too much!



To test multiple versions of the AI at the same time, you can copy the **ai-server** folder multiple times and run all these servers on different ports. Just make sure to pass the correct white player URL and black player URL when running the arena.

1.4 Tasks

1. **[25 points]** Implement a minimax-based AI agent that beats the random AI agent at least 9 times out of 10 games.
2. **[20 points]** Implement alpha-beta pruning to speed up your search.
3. **[10 points]** Implement a heuristic evaluation function for pseudo-terminal states of the game.
4. **[10 points]** Maintain good code style. Give meaningful names to variables and functions. Split long routines into smaller, more manageable ones. Comment the code if the comments provide explanatory value that cannot be achieved through code alone.
5. **[5 points]** Tune your search so that it doesn't exceed the 10-second timeout. You are allowed at most 10 timeouts over 10 played games. The final evaluation will run on a c4.2xlarge AWS instance (should be quite close to the performance of a modern MacBook Pro).

[Extra Credit] After the submission deadline, we will run a tournament between the submitted AIs. The two finalists in this competition will be awarded **+10** and **+5** points respectively.

2 Reflections on the Checkers AI

1. **[20 points]** What options for implementing the heuristic evaluation function did you consider? What worked well and what did not? What idea would you try implementing if you had more time?
2. **[10 points]** According to the game rules, if a player has multiple options for capturing the opponents' pieces, it should pick the longest possible sequence. Find the place in **libcheckers** responsible for this. Step through it with a debugger if you'd like. Would the correctness of the algorithm change if we used a stack instead of the queue?