TÉCNICO LISBOA

Artificial Intelligence and Decision Systems (IASD)
Mini-projects, 2018/2019

Assignment #1

Version 1.3 (27-Oct-2018)

Atari Go

1 Introduction

Go is an ancient board game, believed to be originated in China about 4000 years ago, from which it spread over the world. It remains extraordinarily popular in Asia. Even though the rules of Go are very simple, it gives rise to complex strategies. Go is a two player board game. Players place, in each turn, one stone on a board. Under certain conditions, stones are captured by the adversary. The rules of Go can be found here:

`https://senseis.xmp.net/?BasicRulesOfGo`

Until very recently computer Go remained a hard problem, namely at professional level, due to the large branching factor and long term implications of plays at the beginning, requiring very deep search trees. However, recent methods such as Monte Carlo tree search and deep learning enabled the development of top level computer Go. Google DeepMind's AlphaGo beat in 2017 the top ranked human Go player, Ke Jie.

This mini-project addresses a simplified version of Go, called Atari Go. While Go scoring is based on both surrounded territory and captured stones, the Atari Go finishes when the first stone is captured. More information on Atari Go can be found here:

`https://senseis.xmp.net/?CaptureGo`

2 Objectives

The goal of this mini-project is to develop an Atari Go game engine, to interface an existing alpha-beta search with cutoff algorithm implementation[1].

The implementation should be done in Python version 3. No extra modules, besides the Python Standard Library, are allowed.

In particular, the game engine is to be implemented as a class with name `Game` containing the following methods:

[1] Function `alphabeta_cutoff_search` of the file `games.py` in the GitHub repository `https://github.com/aimacode/aima-python`.

`to_move(`s`)` returns the player to move next given the state s

`terminal_test(`s`)` returns a boolean of whether state s is terminal

`utility(`s`,` p`)` returns the payoff of state s if it is terminal (1 if p wins, -1 if p loses, 0 in case of a draw), otherwise, its evaluation with respect to player p

`actions(`s`)` returns a list of valid moves at state s

`result(`s`,` a`)` returns the sucessor game state after playing move a at state s

`load_board(`f`)` loads a board from a file object f (see below for format specification) and returns the corresponding state

Players are represented with the integers 1 for black and 2 for white. State representation (s) is up to the students. Move representation (a) is the following: a tuple `(`p`,`i`,`j`)` where $p \in \{1, 2\}$ is the player, $i \in \{1, \ldots, N\}$ is the row, and $j \in \{1, \ldots, N\}$ is the column, assuming that `(1,1)` corresponds to the top left position and the size of the board is N. *Note that, unlike the Go rules, passes are not considered in this mini-project.*

The file format for boards is a text file containing the following lines: first line has two space separated values, $n\,p$, where n is the board size and p is the next player to move, the remaining lines represent the rows of the board as a string of numbers, where 0 represents a free place, and 1 or 2 represents a stone of the corresponding player. Example:

```
4 1
0010
0122
0210
0000
```

3 Evaluation

The deliverable for this mini-project has two components:

- a single Python file, called `go.py`, implementing the above mentioned `Game` class, and
- a report in the form of a short questionnaire.

Both components are submitted to a Moodle platform. Instructions for this platform are available at the course webpage.

The grade is computed in the following way:

- 40% from the public tests

- 40% from the private tests
- 10% from the questionnaire
- 10% from the code structure

Deadline: 5-Nov-2018 (Projects submitted after the deadline will not be considered for evaluation.)