Theory of Computer Games (Fall 2018) Homework #2

National Taiwan University

Due Date: 14:20 (UTC+8), December 27, 2018

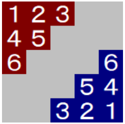
Homework Description

In this homework, you are required to

1. Implement an agent of **Einstein Würfelt Nicht! (Kari)** using Monte-Carlo Tree Search.
2. Beat the random AI and the greedy AI.

Basics

- The game is played on a 5×5 board. Initially there are 6 red cubes and 6 blue cubes on the board.



1	2	3		
4	5			
6				6
			5	4
		3	2	1

1. Each cube has a number between 1 and 6, and there are no two cubes of the same color sharing the same number.
2. The 12 numbers are determined randomly, but are guaranteed to be centrosymmetric.

- In each turn the first player chooses a red cube to move, and subsequently (if the game is not over) the second player chooses a blue cube to move.

Moves

1. In turn $1, 3, 5, 7, \ldots$, a player is restricted to choose cube $1, 3, 5$ to move; in turn $2, 4, 6, 8, \ldots$, a player is restricted to choose cube $2, 4, 6$ to move.
2. The first player can only move a cube to the **east** adjacent square, the **south** adjacent square, or the **southeast** adjacent square. The second player can only move a cube to the **west** adjacent square, the **north** adjacent square, or the **northwest** adjacent square.
3. If there is another cube in the adjacent square, that cube is **captured**. A player is not allowed to capture a cube of himself.
4. If there is no movable cube, a player should **pass** in that turn. A player is not allowed pass if there is a movable cube.

Game Over

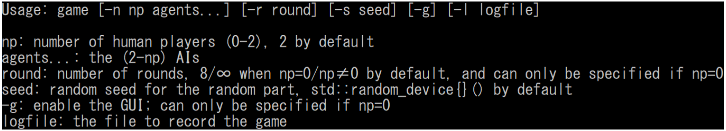
The game is over when

1. a red cube reaches the southeast corner (first player wins), or
2. a blue cube reaches the northwest corner (second player wins), or
3. the last red cube is captured (second player wins), or
4. the last blue cube is captured (first player wins).

This game always yields exactly one winner.

Execution Files

- Under directory `einstein_kari`, use `make` to build the execution files `game`, `greedy`, and `random`.
- Execution file `game` supports AI-AI mode, AI-human (1P) mode, and human-human (2P) mode.



```
Usage: game [-n np agents...] [-r round] [-s seed] [-g] [-l logfile]

np: number of human players (0-2), 2 by default
agents...: the (2-np) AIs
round: number of rounds, 8/∞ when np=0/np≠0 by default, and can only be specified if np=0
seed: random seed for the random part, std::random_device{}() by default
-g: enable the GUI; can only be specified if np=0
logfile: the file to record the game
```

- To begin with, use

```
$ ./game -n 1 greedy
```

to start the game with the agent `greedy`.

game-Agent Communication

- An agent receives the last move of the opponent from game and sends its move accordingly back.
- We've handled most parts of the communication. All you have to do is receive messages by simply reading from stdin and send messages by simply writing to stdout.
- Read everything character-by-character; if you expect a message of length k to be received, read one character k times instead of directly read a string.
- Remember to flush every time after writing a message to stdout.

Frame of an Agent

```
while true do
    receive $R_1$
    if $R_1$ = “end of game” then
        break
    end if
    receive $R_2$
    $B \leftarrow$ the initial board given $R_2$
    while true do
        if $R_1$ = “second player” or this is not the first turn then
            receive $R_3$
            if $R_3$ = “win” or “lose” then
                break
            end if
            do the opponent’s move $R_3$ on $B$
        end if
        choose a move $M$
        do the move $M$ on $B$
        send $M$
    end while
end while
```

Formats of Received / Sent Messages

1. R_1: a single character.
   - ‘e’: end of game
   - ‘f’: you are the first player in this round
   - ‘s’: you are the second player in this round
2. $R_2 := R_2[1:6]$: a permutation of “123456”.
   - number of $(1,1),(5,5) = R_2[1]$
   - number of $(1,2),(5,4) = R_2[2]$
   - number of $(1,3),(5,3) = R_2[3]$
   - number of $(2,1),(4,5) = R_2[4]$
   - number of $(2,2),(4,4) = R_2[5]$
   - number of $(3,1),(3,5) = R_2[6]$
3. R_3: can be “ww” (win), “ll” (lose), “00” (pass), or *nd* (otherwise), where
   - $n =$ number of cube to be moved
   - $d =$ direction: 1 (horizontal), 2 (vertical), 3 (diagonal)
4. M: a 2-sized string, can be “00” (pass) or *nd* (otherwise) only.

Misc

- You can assume that every move your agent receives is valid.
- Your agent should send a valid move within 10 seconds. If game receives an invalid move, or doesn't receive a move within the time limit, your agent will be killed, and your opponent wins immediately.

Code

- You're required to implement the following algorithms:
  - UCB score and UCT
  - Progressive Pruning **or** RAVE
- Your execution file should be named with your student ID, with all alphabets in lower case, e.g., `b07902000`, not `B07902000`.
  - If your programming language is python3, add `#!/usr/bin/env python3` in the first line and remove `.py` from the filename.
- Your agent can use at most 1 thread.
- Your agent will be tested by

```
$ ./game [your_id] [our_agent] -r 3
```

Report

- Your report should contain the following:
    - How to compile your code into an agent (if your code must be compiled). Don't upload the compiled executable file!
    - What algorithms and heuristics you've implemented.
- Your report should be named `report.pdf`.

Directory Hierarchy

- [your_id] // e.g. b07902000
  - source // the directory contains your code
  - report.pdf
- Compress your folder into a zip file.

Grading Policy

- Basics:
  - Beat the agent `random`.
  - Beat the agent `greedy`.
  - `report.pdf`
- Bonus:
  - Beat the agent `hidden`.
  - Ranked high in class.