Take-Home Exam:
An Artificial Intelligence for Reversi

Functional Programming 1 — 1DL330

2018

1 Introduction

Your task for this take-home exam is to implement a Reversi AI in SML.

This is an individual exam. It is acceptable to discuss abstract solution methods with others, and to get inspiration from the Internet or other sources, provided

- you give proper credit (via an explicit comment in your sources) whenever you use someone else's ideas, and
- you have constructed your submission yourself. It is not acceptable to share or copy source code.

See the course's Ethics Rules for further guidelines.

2 Game Rules

Reversi (also known as Othello) is a board game for two players. We will call the players Black and White.

The game is played on an 8-by-8 board. Each field of the board is either empty, marked black, or marked white.

We will number the fields of the board from 0 to 63, from top left to bottom right (see Figure 1).

Initially, fields 28 and 35 are marked black, fields 27 and 36 are marked white, and all other fields are empty.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Figure 1: The fields of the board are numbered from 0 to 63.

Players take alternate turns. Black moves first.

A move consist of choosing an empty field. This field is marked with the moving player's color. Then, all fields lying on a straight, contiguous (horizontal, vertical or diagonal) *line of fields marked in the opponent's color* that extends from the newly marked field to *another field marked in the moving player's color* change their color (from the opponent's to the moving player's). A move is valid only if at least one non-empty field changes its color. See `http://en.wikipedia.org/wiki/Reversi#Rules` for diagrams and further explanations.

If one player cannot make a valid move, (s)he must pass (skip the move). Play passes back to the other player.

When neither player can move, the game ends. (This occurs when all fields of the board have been marked, or when neither player can mark a field so that this would cause a non-empty field to change color.)

The player with the most fields marked in his/her color at the end of the game wins. If there are equally many black and white fields, the game is a draw.

3 Your Task

You are to write a single file called `exam.sml` that declares a structure Reversi_AI.

Your file must not (re-)declare any other top-level identifiers (e.g., for signatures, structures, values, types). That is, your file must be of the form

```
structure Reversi_AI =
struct
  (* ALL your code goes here, inside the structure *)
end
```

Your file will be loaded in an environment where the types player and move have been defined as follows:

```
datatype player = Black | White
datatype move = Pass | Move of int (* 0 ... 63 *)
```

Do not define these types yourself!

Your structure Reversi_AI must match (i.e., provide implementations for all types and values in) the following signature:

```
sig
  type T (* the internal state of your AI *)
  val author: string
  val nickname: string
  val init: player -> T
  val think: T * move * Time.time -> move * T
end
```

The individual components of this signature are specified as follows:

- T encodes the internal state of your AI, i.e., any information that your AI wants to keep track of between different invocations of think. (This information will likely include a representation of the board and current player.) Your structure may define T any way you see fit.
- author is your (first and last) name. Use ASCII characters only (no Swedish vowels!).
- nickname is an arbitrary string (of at most 20 ASCII characters) chosen by you to identify your AI. It will be used to report the evaluation results (see Section 6) pseudonymously.

- init takes an argument that indicates whether your AI will be playing Black or White, and returns the initial state of your AI.
- think takes three arguments: the current state of your AI, your opponent's last move, and the time remaining for your AI to play the rest of the game. The move argument will be Pass when (i) your AI is playing Black and making the first move of the game, or (ii) your opponent had to skip his/her last move. think returns a pair (m,t), where m is the move that your AI wants to take, and t is your AI's internal state after taking this move. (t will be passed to the *next* invocation of your AI's think function, along with your opponent's response to m.) m must be Pass if (and only if) your AI cannot otherwise make a valid move.

Your file must compile when submitted to the Poly/ML compiler, version 5.5.2 (where the types player and move have been defined as detailed above).

Near the top of your file must be a comment containing a brief (about 100-500 words) description of your implementation. All functions must be specified according to our Coding Convention.

Your implementation must not perform any kind of input or output (e.g., via the console, file system or network). Make sure to remove any print calls that you may have added for debugging before you submit.

Your implementation must not spawn additional threads or processes.

Your implementation may use types and values provided by the SML Basis Library. However, it must not use mutable data structures (e.g., references, arrays) or compiler-specific features (such as the PolyML structure).

4 Testing

For your convenience, we provide a testing framework that allows your AI to play games (for instance, against itself). To use this framework,

1. Log in via SSH to any of the Solaris x86_64 machines listed at `https://www.it.uu.se/datordrift/faq/unixinloggning`. Download

   - `http://user.it.uu.se/~tjawe125/reversi-2018/reversi` and
   - `http://user.it.uu.se/~tjawe125/reversi-2018/play.sml`

   into a new folder. (Note that the testing framework will probably *not* work on other operating systems, such as GNU/Linux.)

2. Make the `reversi` file executable:
   `chmod +x reversi`

3. Augment the `LD_LIBRARY_PATH` shell variable:
   `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/it/sw/ml/lib`

4. Create two files `black.sml` and `white.sml` that contain AIs for Black and White, respectively. For instance:
   `cp exam.sml black.sml && cp exam.sml white.sml`

5. Run `./reversi`

We also provide a file that allows a human player (i.e., you) to play for either (or both) sides. Simply download `http://user.it.uu.se/~tjawe125/reversi-2018/human.sml` and rename the file to `black.sml` or `white.sml`.

Please report any problems with the testing framework to tjark.weber@it.uu.se.

5 Submission

Please submit your file `exam.sml` via the "Exam" assignment on the Student Portal. The last time to submit is **Friday, 2018-10-26** at 18:00.

6 Evaluation

Your AI will compete against a very simple computer player that chooses its moves (pseudo-)randomly. Each AI will play 20 games: 10 as Black and 10 as White.

If your AI ever returns an invalid move in any of the games (or otherwise does not conform to the requirements stated in Section 3), it will be disqualified.

For each game, your AI will receive a score between -64 and 64. First, if the game is not a draw, all empty fields will be marked in the winner's color. Your AI's game score is then defined as

$$\text{(number of fields marked in your color)} - \text{(number of fields marked in your opponent's color)}$$

Your AI will be given 5 minutes per game. The time it takes to load your SML source file and to call your AI's init function is counted against this limit. If your AI runs out of time, it will receive a score of -64 for that game (and your opponent will receive a score of 64). If your AI times out on every game, it will be disqualified. All games will be played on reasonably recent hardware.

If your AI causes stability issues due to excessive memory requirements (> 1 GB) during a game, we may treat this similar to a timeout. (We do not expect memory usage to be an issue in practice, and will not actively monitor it unless problems arise.)

Your AI's total score is defined as the sum of its game scores.

7 Grading

If your AI was disqualified, the exam will not improve your final course grade (i.e., your course grade will be 3, provided you passed the lab assignments).

Otherwise, the exam grade depends on your AI's total score:

Evaluation result	Disqualified	Not disqualified	
		total score < 500	total score ≥ 500
Exam grade	(3)	4	5

With this grading scheme, you can decide yourself whether you merely want to implement the rules of the game or whether you want to implement a moderately strong AI, and hence, which grade you want to aim for.

8 Tournament

We will also conduct a tournament between all submitted AIs. Every AI will play against every other AI, once as Black and once as White. The tournament rules (e.g., timing, scoring) are as detailed in Section 6.

We will report the tournament results on the Student Portal, using your AI's nickname to identify it. These results may bestow bragging rights, but will not affect the grading.

Hints

For grade 4:

To attain a grade of 4, it is sufficient to return *any* valid move (or `Pass` if no valid move exists). This merely requires you to implement the rules of the game. It can be done without game tree search (and without an evaluation function for game positions).

The board can be represented in many different ways. A 64-tuple of fields is one (but probably not the most convenient) possible solution. Consider using lists or vectors instead.

Also consider using a 10-by-10 board internally, whose border fields stay empty. This can greatly simplify the implementation of the straight-line color change rule.

Test your code thoroughly before you submit.

For grade 5:

Two key components in a strong Reversi AI are (i) the search algorithm, and (ii) the evaluation function.

There are many strong Reversi programs. Do not try to reinvent the wheel. Instead, research existing techniques. A good starting point is `http://en.wikipedia.org/wiki/Computer_Othello`. Remember to give credit when you use someone else's ideas.

Test your code thoroughly before you submit. Buggy code may easily be worse than no code.

If you are aiming for a very strong AI, make sure it doesn't run out of time. The evaluation hardware may be slower (or faster) than your testing hardware.

Do not go overboard. We understand that one could spend a lifetime perfecting a Reversi AI, while you only have a few days.