COMP30024 Artificial Intelligence

Project Part B: Playing the Game

Chris Leckie and Sarah Erfani
Modified by: Matt Farrugia
Last updated: 12 April 2017

Task

The aim of this part of the project is to design and implement a Java 1.8 program to play a complete game of Slider. You are encouraged to use any of the game-playing techniques discussed in lectures and tutorials, to develop your own Slider strategies, and to conduct your own research into different game-playing algorithms. Before you read this specification, please ensure you have read "Project Specification: Rules of the Game of Slider".

Your program should be a collection of Java classes, including one class implementing the interface `aiproj.slider.SliderPlayer`. This interface specifies methods for receiving a starting board state, receiving moves made by an opponent, and requesting the next move for your agent to make (both in the form of `aiproj.slider.Move` objects). This interface is used by the provided driver program, `aiproj.slider.Referee`, which contains a main method to conduct a game between two implementations of the `SliderPlayer` interface. Source files for the driver, the interface, and the move class are available on the LMS. More details on these components are provided below.

Your submission may also contain any additional programs you have used to assist in the development of your game playing program. For example, if you conduct any machine learning to optimise your program's competitive performance, you should include with your submission any programs you developed for these purposes. These programs may be implemented in languages other than Java.

Interface

For your program to be able to play a game of Slider, you must have a class which correctly implements the provided interface `aiproj.slider.SliderPlayer`. To do so, your class must meet these conditions:

1. You must import `aiproj.slider.SliderPlayer` and `aiproj.slider.Move` at the start of your source file.
2. Your class must be declared with a declaration of the form:

   ```
   public class ClassName implements SliderPlayer {/*class definition*/}
   ```

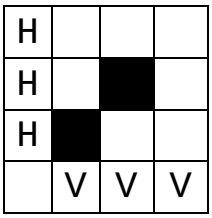
   where `ClassName` is the name of your class.
3. Your class must implement the three methods described in the `SliderPlayer` interface. These methods are also described in the remainder of this section.

The `SliderPlayer` interface includes 3 methods that you need to implement:

`public void init(int dimension, String board, char player);`
This method is called by the Referee (driver) once at the beginning of a game to initialise your player. You should use this opportunity to set up your own internal representation of the board, and any other state you would like to maintain for the duration of the game.
The input parameters are as follows: `dimension` is an integer representing the dimension of the board. `board` is a string describing the initial board state. `player` is either `'H'` (if you are the horizontal player for this game) or `'V'` (if you are the vertical player for this game).

The format for the string `board` is like the format used in project part A. However, these is no first line containing the board dimension. Also, the input will be a string; you will not read it from standard input. As an example, the following call to `init` would describe a game as the horizontal player on the following board:

```
init(4, "H + + +\nH + B +\nH B + +\n+ V V V\n", 'H')
```



Note: A useful tool for reading a string in Java in a similar manner to reading from standard input is `java.util.Scanner`, which has a constructor that takes a string.

`public void update(Move move);`
This method is called by the Referee to inform your player about the opponent's most recent move, so that you can maintain your internal board configuration. The input parameter, `move`, is an object of type `aiproj.slider.Move` which is described later in this section.
Note: In some situations, the value of `move` passed to update may be `null`:

- If the opposing player's last move was a pass (because they had no legal move), or
- In the first turn of the game, when the opponent has not yet made a move.

Your program must be capable of handling these cases.

`public Move move();`
This method is called by the Referee to request a move by your player. Based on the current state of the board, your player should select its next move and return it as an object of type `aiproj.slider.Move`.

The helper class `aiproj.slider.Move` is described in `Move.java`. It's also described here:

`aiproj.slider.Move`
This class encapsulates a possible Slider move, and includes the following fields:

- `i` and `j`; integers representing the column and row* of the piece to be moved, and
- `d`; a `Move.Direction` representing the direction in which this piece is to be moved.

`Move.Direction` is an enumerated type, providing symbolic constants:
`Move.Direction.RIGHT`, `Move.Direction.LEFT`, `Move.Direction.DOWN`, and `Move.Direction.UP`.

***Note**: for part B, `i` is to be used for board columns, and `j` for board rows. This differs from the coordinate system described in the original game specification, but mimics standard mathematical usage (Cartesian coordinates). Please keep in mind that if you are reusing code from project part A, you may need to swap some `i`s and `j`s while setting up your board.

Referee

The `aiproj.slider.Referee` class is the driver program, which will allow you to actually play a game of Slider between two implementations of the `SliderPlayer` interface. Its `main` method has the following structure:

1. Get the board dimension and class names given as command-line arguments
2. Load a new board of the given dimension
3. Instantiate and initialise a Horizontal player and a Vertical player
4. While the game has not ended:
5. Update the current player with the opponent's previous move
6. Ask the current player for their next move
7. Apply this move to the board, if it is legal (otherwise break the game loop)
8. Swap the current player, and repeat this loop
9. Display the result of the game

The Referee program takes input in the form of command-line arguments in the following order:

- `N`, an integer representing the dimension of the board.
- `playerH`, the fully qualified package/class name of a class implementing the `SliderPlayer` interface, to be the Horizontal player for this game.
- `playerV`, the fully qualified package/class name of a class implementing the `SliderPlayer` interface, to be the Vertical player for this game.
- (optional) `sleepTime`, the number of milliseconds to delay between rendering boards, to be used to slow down the output so that you can more easily watch your player make decisions.

To run your implementation of the SliderPlayer interface with the Referee, you can import the Referee source file (and the other provided source files) into your Java editor. From there, you should be able to specify command line options and run the Referee inside your editor.

Alternatively, you can run the Referee via the command line. Here's an outline of the steps for compiling and running your project on a Unix terminal (such as bash or MinGW):

1. **Compiling:** let's assume you have a terminal located in the directory with all your source files. That is, listing the contents of the directory shows `Referee.java`, `Move.java`, `SliderPlayer.java`, and all your .java files used for implementing the interface.
   Then, you should be able to compile these files using the following commands:

   ```
   mkdir -p bin
   javac -d bin/ *.java
   ```

   This will create a folder called `bin` (if it does not exist already), and place in it your compiled `.class` files under the correct directory structure according to their package names.

2. **Running:** let's now assume that your Slider Player implementation was a class named `MySliderPlayer`, and its package statement (at the top of the .java file) was `package ai.partB;`. Then, you should be able to run the Referee with your Slider Player playing against itself on a board of dimension 5 with the following command:

```
java -cp bin aiproj.slider.Referee 5 ai.partB.MyPlayer ai.partB.MyPlayer
```

   This will tell Java to look inside the `bin` directory for its class files (`cp` stands for class path), and to run the main program `aiproj.slider.Referee`, with the three command-line arguments `5`, `ai.partB.MyPlayer`, and `ai.partB.MyPlayer`.

Naming

As your implementation of the `SliderPlayer` interface will be run against other programs, it is important that your classes have unique names (don't use `ai.partB.MyPlayer`!). One way to ensure this is to encapsulate all your Classes inside a package structure which somehow incorporates the usernames of you and your partner. You are encouraged to be as creative and witty as possible when it comes to naming your Slider Player class.

Board states

Your agent should be able to play a valid game on boards of various dimensions $N > 3$. However, for the performance testing of your project, we will only use board sizes $N = 5$, $N = 6$ and $N = 7$. So, your design and strategies should be optimised for these board sizes.

The initial board state will be passed to your program at the beginning of a game by the Referee. The board state will be as described in the game specification document. In addition, the following rules will be used to select 'blocked' position(s) at the beginning of each game:

On a board of size $N > 3$, zero, one, or two positions will be blocked for the duration of the game. The provided Referee will choose which of these options to select at runtime.

- When one position is to be blocked, a random position will be selected from the cells that are *not* along the edges of the board. That is, the position to be blocked will not be in the topmost row, the bottommost row, the leftmost column, or the rightmost column. In other words, the position will be a randomly selected cell (i, j) where $0 < i < N\text{-}1$, and $0 < j < N\text{-}1$.
- When two positions are to be blocked, they will both be chosen from the same board region as for one position (i.e. *not* along the edges of the board). In addition, the two positions will either both be on the main diagonal (i.e. of the form (i, i) and (j, j)), or they will both be placed symmetrically with respect to the main diagonal (i.e. of the form (i, j) and (j, i)).

Performance constraints

The following constraints will be strictly enforced on your implementation during testing. This is to prevent your programs from gaining an unfair advantage by using a large amount of memory and/or computation time. Please note that these limits apply to each player for an entire game—they do *not* apply on a per-turn basis.

Board Size	**Memory Limit**	**Time Limit (CPU Time)**
N=5	750KB	15 seconds
N=6	1500KB	30 seconds
N=7	1500KB	30 seconds

During marking, the memory constraint will be set using an `-Xmx` Java runtime command line option to set the maximum heap size, and the time constraint will be managed by the Referee.

Any attempt to circumvent these constraints will not be allowed. For example, your program must be single threaded, and must run in isolation (without attempting to connect to any other programs, for example, via the internet, to access additional resources). If you are not sure whether a particular strategy will be allowed, please seek clarification early.

Assessment

Part B will be marked out of 22 points, and contribute 22% to your final mark for the subject. Of the 22 points, 4 points will be for the quality of your code and comments (e.g., intuitive and consistent variable names, sensible use of classes, pseudocode and time complexity), 4 points will be for the correctness and reliability of your code (e.g., always playing a legal game, not making any illegal moves, not violating the performance constraints), 7 points will be based on the results of testing the performance of your game playing agent against other agents that we shall provide, and 7 points will be for the creativity of your solution (to be discussed in lectures). Please note that part of your submission will be a `comments.txt` file, in which you will have an opportunity to provide an explanation of your solution and any creative aspects of your design. Please note also that a game playing agent that only makes random moves will not be sufficient to pass Part B.

There should be one submission per group. You are encouraged to discuss ideas with your fellow students, but your program should be entirely the work of your group. It is not acceptable to share code between groups, nor to use the code of someone else. Further, you should not host your code publically online where other students might find it. You may make use of the library of classes provided by the Russell and Norvig textbook site, provided you make appropriate acknowledgements that you have made use of this library. If your program is found to be suspiciously similar to someone else's or a third party's software, you may be subject to investigation and, if necessary, formal disciplinary action. Please refer to http://academichonesty.unimelb.edu.au/ or ask Sarah or Matt if you need further clarification on this point.

Teamwork

As part of your project, you and your partner must create and submit a brief *project plan* containing the following:

- A breakdown of how you plan on sharing the workload of this project between members.
- A summary of where and how regularly you plan on communicating about the project (e.g. weekly meetings after tutorials, video chat, email).
- A list of upcoming deadlines for work in other subjects between now and the end of semester.

You must email this document along with your names and student numbers to Matt Farrugia (matt.farrugia@unimelb.edu.au) **by 4.00pm, Wednesday 26th April 2017**.

Additionally, you should keep minutes/meeting notes recording topics discussed at all project meetings. Finally, each group member will have an opportunity to individually comment on their experience as a part of the final submission process. In the event of a dispute between partners about individual contributions to the project, these documents will help us reach a fair resolution. Note that it's in your best interest that any issues are identified early, so please reach out as soon as anything goes wrong.

Questions and clarifications

Any questions about this specification, the provided java files, the compilation and running process, using packages, or anything else project-related should be directed to the relevant LMS Discussion Board (for questions of a general nature) or to Matt Farrugia (matt.farrugia@unimelb.edu.au) via email (please use the subject header “COMP30024 Project B”).

Clarifications and answers to questions about the project will be made available on the LMS and will be considered as part of the formal specification for this project. Any significant revisions will be accompanied by an announcement and email, but you should regularly check the LMS Discussion Board for additional clarifying information.

Submission

The final submission deadline for part B is **4.00pm Wednesday 17th May 2017**. As for part A, a separate document partB-submit-2017.pdf will describe the submission instructions for part B.

Tournament

After the submission deadline, we will endeavour to run a Slider tournament to determine which submissions perform the very best. The details of the tournament are still being finalised, and more information will be released in a separate document closer to the submission deadline.