Programming Languages Assignments (v1.0)

Eric Laermans (eric.laermans@UGent.be)

2017-03-02

Assignment 1 Using the Message-Passing Concurrency Model

In a first phase of the assignment you will write a program in Oz implementing the following game.

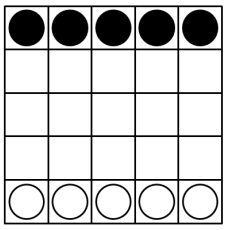

Figure 1: Initial configuration 5-by-5 board

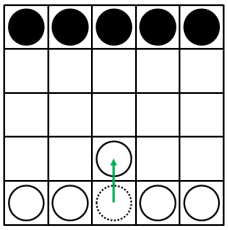

Figure 2: White player makes the first move

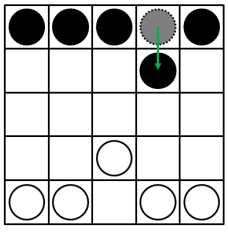

Figure 3: Black player makes the second move

Rules of the game

The rules of the game to be used for this assignment are:

1. The game is played on an N-by-M board (both N and M should be at least 3, and at most 8 to keep things practical). The default starting values will be $N = 5$ and $M = 5$.

2. There are two players:

   (a) Player1 (aka the white player), who makes the first move and uses the white pawns.

   (b) Player2 (aka the black player), who uses the black pawns and chooses the size of the board.

3. The initial configuration of the board is shown in Fig. 1: the top row is filled with black pawns, the bottom row is filled with white pawns.

4. Pawns can move foward (up for white pawns, down for black pawns) by 1 square on the board. Possible initial moves for the white and black players are shown in Fig. 2 and Fig. 3.

5. A pawn can capture a pawn of the opposite colour on a square diagonally in front of it on an adjacent column of the board, as shown in Fig. 4.

6. The first player who reaches the opposite side of the board (top row for white player, bottom row for black player) wins the game. This is shown in Fig. 5.

7. A player can also win the game by blocking the game, making it impossible for his opponent to move one of his pawns or to capture one of his opponent's pawns. This situation is shown in Fig. 6.

8. A game cannot end in a draw.

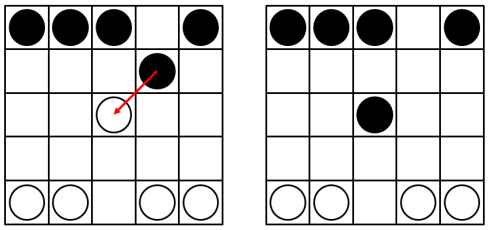

Figure 4: Black pawn captures a white pawn

Requirements

Your code will have to satisfy the following requirements:

- Write the program as three concurrent threads representing each player (the player threads) and a referee (referee thread).

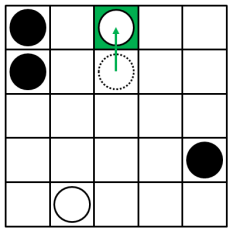

Figure 5: White player wins the game by reaching the opposite side of the board

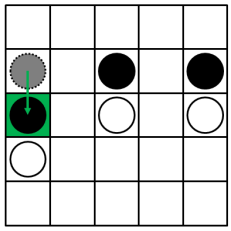

Figure 6: Black player wins the game by blocking the game (White player can't move or capture)

- The referee thread will verify and validate each move submitted by the players. If a move is rejected, the player will have to submit another attempt. If this attempt also fails, the referee will proclaim this player's opponent the winner of the game.
- The referee also proclaims the winner of the game. The referee thread is essential to the operation of the game.
- Each player thread and the referee thread must be written within the declarative model in Oz.
- Both player threads communicate with the referee thread using the message-passing concurrency model. Make sure all necessary information can be transmitted through this mechanism, but keep the communication as simple as possible (keep the second part of the assignment in mind!).
- Using functors (see book, sections 3.9.3 and 3.9.4) would be extremely useful (again, keeping the second part of the assignment in mind).

A first operating version (it need not be the final version, but you should make sure that the communication mechanisms don't change too much in later versions) of the code for this assignment must be available on **April 14, 2017**. A final version of the code must be available on **May 8, 2017**.

Assignment 2 Adapting Declarative Code and Integrating Other People's Code

The second phase of the assignment will have two aspects:

1. An additional rule for the game (to be announced on **April 15, 2017**).
2. A small tournament within the group (approximately 7 users) to which you will be assigned. The composition of the groups will be announced on **April 11, 2017**.

Assignment 2.1 Adapting Declarative Code

An additional rule for the game will be communicated. You will have to modify your code to comply with this additional rule.

The goal is to explore the impact of modified specifications on declarative code.

Assignment 2.2 Integrating Other People's Code

The goal of the tournament is to explore the modularity of declarative code. You will have to integrate the code of other players with your own code.

1. You will have to post your compiled code (other players of the group should not see your source code; this is why you had to use functors in the first phase of the assignment) in the group documents on Minerva. As the compiled code of Oz is bytecode, there should normally not be any incompatibility between different platforms. However, the Oz bytecode for version 1.4 is incompatible with the bytecode for version 2.0 of the language (and vice versa). I'll set up a poll on Minerva to ask you which version you use. All users within a single group will use the same version (1.4 or 2.0).

2. You will also have to communicate how other users can interact with your code (which functions are accessible). This information must also be posted in the group documents on Minerva.

3. You will have to agree on the information you exchange in the message-passing concurrency between the thread of your own code and the threads of the imported code (This is why the communication mechanisms of your final code should not change too much with respect to those of your originally posted code).

4. The integration will probably require some adaptation of each participant's code.

Tournament

Once the different codes have been made compatible, you will have the possibility to improve your code in order to perform better in the eventual tournament.

1. You test your player code with the player code of every other member of your group.

2. You select a different referee (different from both players) for each game.

3. You play the first move on your own computer.

4. You will face each opponent twice: once on your own computer (playing the first move), once on his/her computer (playing the second move).

5. The tournament will be performed using the final versions of the code (which must be available on **May 8, 2017** at the latest, both on Minerva's Dropbox and in the group documents).

6. Each victory will yield one point, each defeat will yield zero points (these are points for the tournament, not for the assignment).

7. Please, do not cheat...

Practically

Report

Consider your report as a paper describing the work you have done. So, don't forget to mention the hardware/software on which the programs are executed and don't forget a conclusion for each assignment. Also be honest about possible remaining issues with the code you have written. I prefer to read about it in the report than to discover it when I test the code.

Explain how your code works in the report. I should be able to understand it even without having a look at your code itself.

Discuss your experience with both the declarative model and the message-passing concurrency model for the development of the code. What were possible advantages of this choice? What were possible drawbacks? Would another model have been preferable?

Discuss the impact of the rule modification on your code. Would the impact have been similar if you had been allowed to use the stateful model?

Discuss the integration of your code with other players' code. What was the impact of the choice of the programming model (declarative code + message-passing concurrency)? Would another programming model have been a better option?

The assignment reports are due on **May 22, 2017** and should be done individually (except for the tournament part, which will require some limited cooperation within each group).

As the first exams are scheduled on May 29, it is impossible for me to *guarantee* feedback about the score of your assignments before the exam (especially because I have to score assignments for another two courses). If you want earlier feedback, please ask this in advance. However, in this case, I expect your report on May 15, 2017 (7 days before the normal deadline). Otherwise, it will be "best effort" only.

Scoring

The assignments (including the discussion of the results at the oral exam) will count for 35% of your global score for this course.

Evaluation criteria will be:

- the quality and the completeness of the report
- whether the code satisfies the requirements (e.g. declarativity)
- the performance of the code (the tournament results are not directly taken into account, but may be an indication, as I don't expect an extremely simple implementation to perform very well in the tournament)
- the timeliness of the deliverables

Deliverables

- **April 14**: first operating version of the code, to be uploaded on the Dropbox of Minerva (source code only) and in the Documents section of your group (bytecode only + specifications on the expected input)

- **May 8**: final version of the code, to be uploaded on the Dropbox and in the Documents section of your group (source code + bytecode + specifications)
- **May 15**: advanced deadline for the report, to be uploaded on Minerva's Dropbox (deadline only applicable to those who want guaranteed feedback before the exam)
- **May 22**: regular deadline for the report, to be uploaded on Minerva's Dropbox
- **May 22**: deadline for the publication of the results of the tournament: all games + classification (as a spreadsheet, text file,... uploaded on Minerva's Dropbox)

All deadlines should be understood as "**before 22h00 that day**".

Other Calendar Events

- **April 11**: announcement of group compositions
- **April 15**: announcement of additional rule to be implemented

Questions

If you have any further question, please contact me.

If you experience trouble in reaching other group members, or in collaborating with them, or if you experience any other problems, let me know the issues as soon as possible. The sooner I know about problems, the easier it will be to attempt to solve them. Don't wait until the report deadline to address these issues!