Artificial Intelligence
Fall 2020
Project #1
Grading Sheet

Name: Jonathan Lam

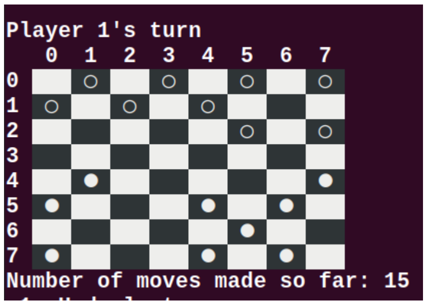
Email: lam12@cooper.edu

Grade: 97

Notes:

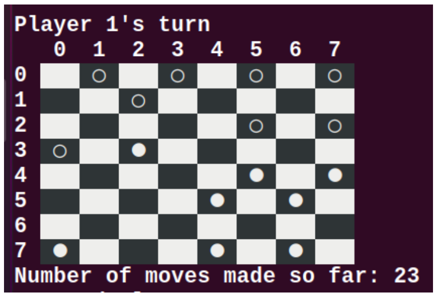
You have implemented Checkers. You used C++ first, then ported it to Scheme. As discussed in an email chain, I will at least start off grading the C++ version, because it tends to search a bit deeper, but I'll try out the Scheme version at the end. I am using an Ubuntu 20.04 virtual machine with 2 GB of RAM on my home desktop.

I am playing my first game moving first, as black, giving the program (which moves second) 5 seconds per move. The interface is black and white but looks quite good. The early moves by the program seem fine. It is reporting searches to depth 10 early. There were some early trades. When the program has a forced move, it makes it right away. The program did have to abandon one square of its back row to avoid losing material, which may put me at a slight positional advantage. Here is the early situation on one of my turns:



```
Player 1's turn
   0  1  2  3  4  5  6  7
0     ○     ○     ○     ○
1  ○     ○     ○
2                 ○     ○
3
4     ●                 ●
5  ●           ●     ●
6                 ●
7  ●           ●     ●
Number of moves made so far: 15
```

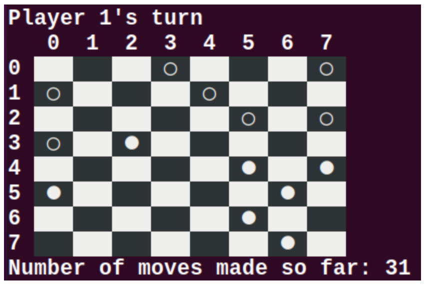
A bit later, I forced a trade that may put me in a position where I will eventually gain material; here is the updated board on my move:



```
Player 1's turn
   0  1  2  3  4  5  6  7
0     ○     ○     ○     ○
1        ○
2                 ○     ○
3  ○     ●
4                 ●     ●
5              ●     ●
6
7  ●           ●     ●
Number of moves made so far: 23
```

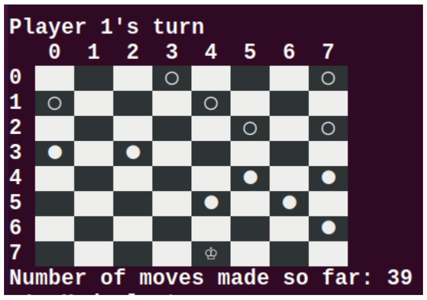
That piece at 3,2 it possibly too far forward; I will start to attack it (starting with 0,5->1,4) and possibly pull ahead. When the trading started, the computer was searching to depth 9, which was not far enough to see that I will eventually gain material (if I am correct that I will, which is far from certain); so, even if I manage to pull ahead, this is not due to a bug.

Nope. The program manages to defend, and we are still even. Here is the situation later:



```
Player 1's turn
   0  1  2  3  4  5  6  7
0           ○           ○
1  ○           ○
2                 ○     ○
3  ○     ●
4                 ●     ●
5  ●                 ●
6                 ●
7                    ●
Number of moves made so far: 31
```

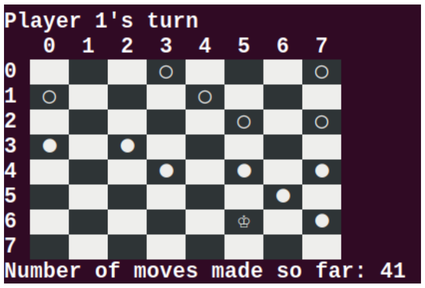
I think about 0,3->1,2 to attach more, but after 6,5->5,4, I don't think I have a path to gain material, and my back row would be exposed. So instead, I will play 3,0 -> 4,1 and work toward getting a king. (Another option was 2,5->3,6 to force a two-for-two trade.) I go on to get the first king. (Cool little character icon that looks like a crown!) Here is the updated board:



```
Player 1's turn
   0  1  2  3  4  5  6  7
0           ○           ○
1  ○           ○
2                 ○     ○
3  ●     ●
4                 ●     ●
5              ●     ●
6                       ●
7              ♔
Number of moves made so far: 39
```

The program just searched to depth 11; very good for the middle game! We'll see if my king is enough of an advantage for me to win, as I start chasing its pieces.

I don't know if I like its next move:



```
Player 1's turn
   0  1  2  3  4  5  6  7
0           ○           ○
1  ○           ○
2                 ○     ○
3  ●     ●
4           ●     ●     ●
5                    ●
6                 ♔     ●
7
Number of moves made so far: 41
```

Now, after 2,5->3,6, it will have to jump me and I will get a double jump in return, plus have a path to another king. However, looking at its options, none were good. I should go on to win here if I don't make a mistake.

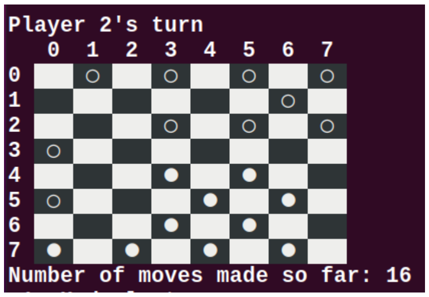
Soon, I am ahead five pieces to three; I have three kings, the program has two. I'll win unless I make a major blunder, but let's see how hard the program makes it for me!

The program is playing fine, putting off the inevitable. I have been careful, and I managed to force a couple of trades. Now I am ahead three pieces to one, and I have two kings (and can get a third easily). The program moved its king to a double corner, so I will have to force it out.

Near the end game, the program starts to make its move very fast. Interestingly, the depth search is 7, which I don't think is to the end of the tree, but perhaps the program sees that it will definitely lose somehow (maybe with my extra piece, there is a way to win quicker than I see). In any case, the end of the game goes quickly. The program puts off its loss as long as it can. The program exits gracefully after the loss.

I start a second game as "red" (white on the board), with the computer as black, again giving it 5 seconds per move. It's playing fine. It is searching to depths around 10.

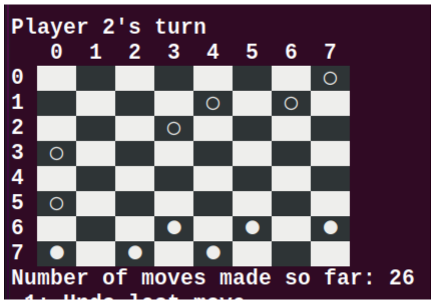
I may notice one minor flaw (not bug, but something non-ideal) with its heuristic. Look at this situation:



```
Player 2's turn
   0  1  2  3  4  5  6  7
0     ○     ○     ○     ○
1                    ○
2           ○     ○     ○
3  ○
4           ●     ●
5  ○           ●     ●
6           ●     ●
7  ●     ●     ●     ●
Number of moves made so far: 16
```

It just moved from 1,4 to 2,3. Now, I will move from 4,2 to 3,4; it will jump, I will double jump, it will jump. In the trade, it will abandon one of its back-row squares. I looked back at your writeup, and you do say it values keeping the back row protected; but maybe that weight should be a bit higher. Last game, it also moved one of those pieces early. That may be why I was eventually able to get a king, and ultimately won that game.

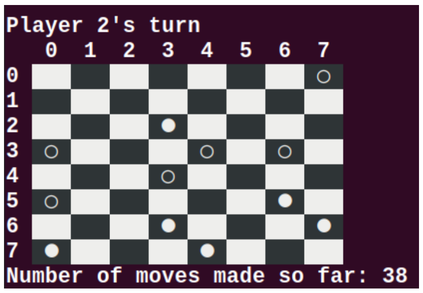
Here is the situation a bit later:



```
Player 2's turn
   0  1  2  3  4  5  6  7
0                       ○
1              ○     ○
2           ○
3  ○
4
5  ○
6           ●     ●     ●
7  ●     ●     ●
Number of moves made so far: 26
```

As you can see, my back row is much better protected than the programs. Whether that will be enough for me to win (or even avoid losing), I don't know.

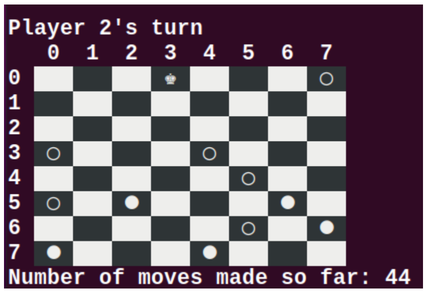
Here is the situation a bit later:



```
Player 2's turn
   0  1  2  3  4  5  6  7
0                       ○
1
2           ●
3  ○           ○     ○
4           ○
5  ○                 ●
6           ●           ●
7  ●           ●
Number of moves made so far: 38
```

I have the clearest path to a king, of course, but this is confusing. It may still be a theoretical draw. It can go from 3,6 to 4,5; then 4,3 to 5,4. Anyway, I'll just walk ahead for now and get my king. Hopefully (for me) getting it one or two moves ahead gives me some leeway.

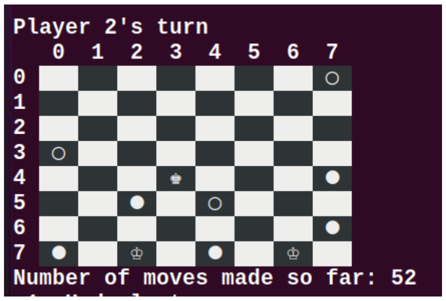
Things play out basically like I expected. Here is the updated board:



```
Player 2's turn
   0  1  2  3  4  5  6  7
0           ♚           ○
1
2
3  ○           ○
4                 ○
5  ○     ●           ●
6                 ○     ●
7  ●           ●
Number of moves made so far: 44
```

I will start utilizing my king. After it gets its king at 7,6, I can move from 5,6 to 4,7 so the program's king will be trapped.

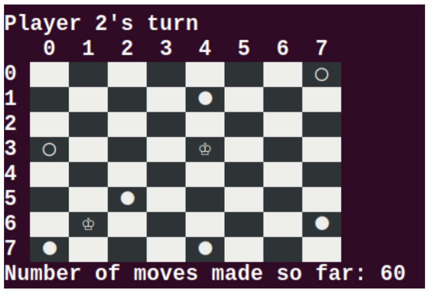
Just as I felt I might be headed to another win, I really like the program's latest move:



```
Player 2's turn
   0  1  2  3  4  5  6  7
0                       ○
1
2
3  ○
4           ♚           ●
5        ●     ○
6                       ●
7  ●     ♔     ●     ♔
Number of moves made so far: 52
```

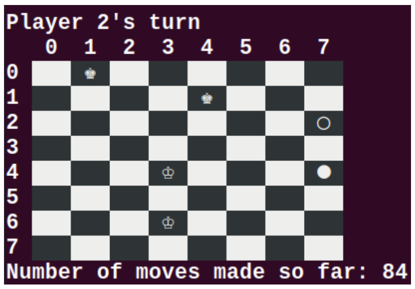
Note that I was temporarily ahead a piece. However, it just moved 4,5 to 5,4; this forces a trade, but it will get my king for a regular piece. Although I will still be temporarily ahead one piece, it will have two kings, I will have none, and its kings will no longer be trapped. If anything, I think a loss for me is more likely than a win now. We will see!

Here's the board a bit later:



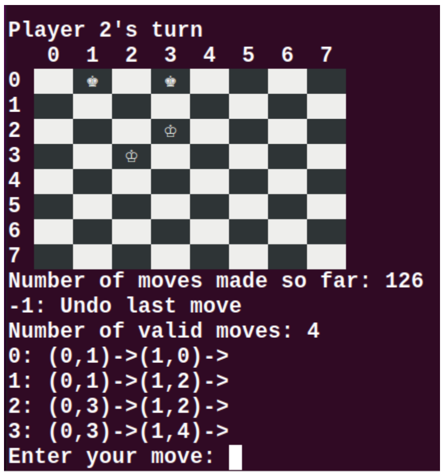
```
Player 2's turn
   0  1  2  3  4  5  6  7
0
1
2
3
4
5
6
7
Number of moves made so far: 60
```

I could just get the king at 3,0, and then after the program jumps me, move my king to 1,2 to it won't become trapped. That would probably be an even game, where we each have four pieces, although I would be down one king without much flexibility. I think better for me is to move 5,2 to 4,3 forcing a jump; then 7,4 to 6,3 forcing another jump (both of regular pieces); then I capture its king at 6,1. I think this gives me clear paths to get two kings (if I am not missing something). This has some risk (I cannot visualize future boards), but if I am processing it correctly, it will be a more clearly even game. This is what I will try. I do, and things play out as I hope. This is a clearly even end game position (but I'll play it out a bit to see what happens):



```
Player 2's turn
   0  1  2  3  4  5  6  7
0
1
2
3
4
5
6
7
Number of moves made so far: 84
```

Eventually, I force another trade, and now we each have two kings. One of my kings is in a double corner. I am just moving that king back and forth; the program is also moving one of its kings back and forth. I declare this game a draw! Here is the final situation:



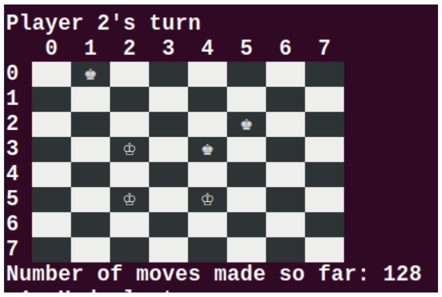
```
Player 2's turn
   0  1  2  3  4  5  6  7
0
1
2
3
4
5
6
7
Number of moves made so far: 126
-1: Undo last move
Number of valid moves: 4
0: (0,1)->(1,0)->
1: (0,1)->(1,2)->
2: (0,3)->(1,2)->
3: (0,3)->(1,4)->
Enter your move:
```

Since I haven't tried your undo move feature yet, I will test it now, just to see that it works. I type -1 a bunch of times (I didn't realize you can do it more than once). It works great! I see the game unplay a bunch of moves, then try moving forward again and it works. Nice feature!

Next, I try loading one of my boards with many, crazy multiple jumps available. (I see you already have them available for me in your "states" folder.) It loads fine, and all the legal moves are found. (I try this in person vs. person mode, which works fine, I can make moves for both players.)

Next, I load a board where I have one king in a double corner, and the program has two kings that start far away. The program marches its kings toward me, forces me out of the double corner (probably as quickly as it could), and goes on to win the game. The program handles the situation perfectly.

Next, I watch the program play a game against itself, giving it three seconds per move. The feature works fine. Both sides seem to be playing well. I can't really follow all the logic at this speed, but I notice no bad moves. Both sides get the first king at close to the same time. At first, I think the player that got the king a few moves ahead may be able to use it to pull ahead material, but the other side found a tactic to keep the material even. Now, both sides have three pieces, all kings. This should be a draw, but I'll let it play a bit longer. I let it play a bit longer, and I think there is some repetition now. I declare this game of computer vs. computer a draw. Here is the final situation where I end it:



```
Player 2's turn
   0  1  2  3  4  5  6  7
0     ♚
1
2                 ♚
3        ♔     ♚
4
5        ♔     ♔
6
7
Number of moves made so far: 128
```

I will try the Scheme version now! I install chezscheme. I don't think I need to change the permissions (as in your report); the play-game.ss file is already an executable. I try this command, as indicated in the report: ./ play - game .ss

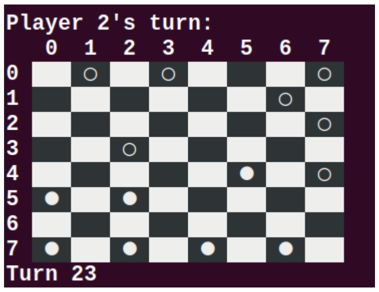
I get an error message from your program: "Exception: incorrect number of arguments to #<procedure char-ready?>"

I try it in interactive mode, and the same thing happens.

Ah, I see you addressed this in your email. I applied the fix that you sent me (one line changes in main.ss). Now, the game runs fine. I am starting a game moving second (as red/white), giving the program five seconds per move. The program is searching to depth 9 early (one less than when I started games in C++). I see you added two extra choices (quit, or use iterative deepening to find the best move), as mentioned in the writeup.

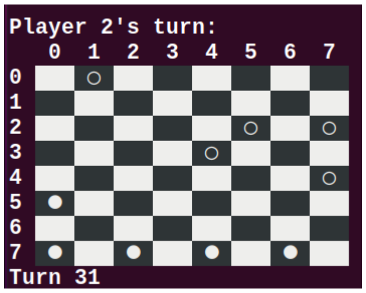
One thing I like about the Scheme version is that the program indicates its chosen move (which didn't happen in the C++ version); for example: "Best move: -1: (1,6)->(2,7)". I don't know why it shows -1, but the move is clearly indicated. (With the C++ version, at times I would scroll up to look at the previous board to see what move was made. Of course, this is not a big deal, but just one nice difference I noticed in the interface.)

As we enter the middle game, the program is searching to depth 8. Again, this is less than the C++ version, but still clearly enough to play well. We've done a bunch of trades, and we are even in the middle game:



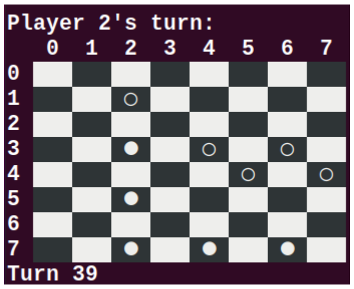
```
Player 2's turn:
   0  1  2  3  4  5  6  7
0     ○     ○           ○
1                    ○
2                       ○
3        ○
4                 ●     ○
5  ●     ●
6
7  ●     ●     ●     ●
Turn 23
```

Here is the situation a bit later:



```
Player 2's turn:
   0  1  2  3  4  5  6  7
0     ○
1
2                 ○     ○
3              ○
4                       ○
5  ●
6
7  ●     ●     ●     ●
Turn 31
```

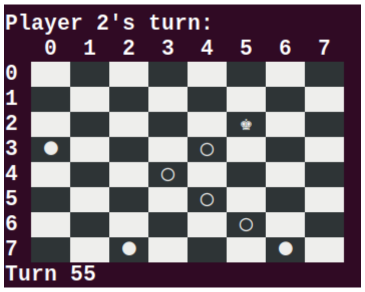
Note that my back row is still perfectly protected (although I will be forced to abandon one of those squares soon); the program's is mostly exposed. Although I have no clear path to a king (when I start matching 5,0 forward, it will likely use the piece at 0,1 to block), I think I have enough of an advantage here that I should get a king first. I don't know if I will win, but we'll see! Here is the situation a bit later:



```
Player 2's turn:
   0  1  2  3  4  5  6  7
0
1        ○
2
3        ●     ○     ○
4                 ○     ○
5        ●
6
7        ●     ●     ●
Turn 39
```

After I move from 3,2 to 2,3, I will get the first king and have a clear advantage (I think!).

The program hangs on for a while without losing a piece, but now we are here:

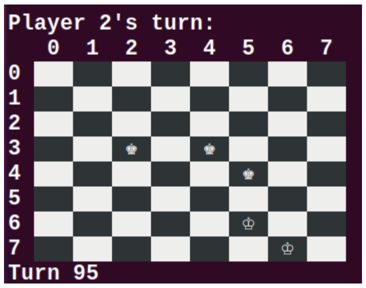


```
Player 2's turn:
   0  1  2  3  4  5  6  7
0
1
2                  ♚
3  ●           ○
4           ○
5              ○
6                  ○
7        ●           ●
Turn 55
```

Now, I will move from 7,2 to 6,3, forcing a jump, and getting a double jump in return.

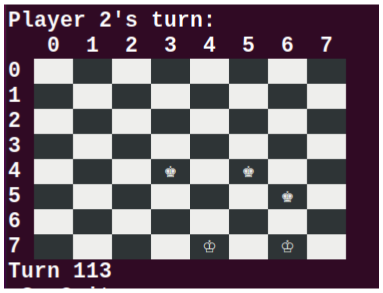
The program recently searched to depth 11, by the way; about as good as the C++ version!

Soon, I am ahead three pieces to two. The program has two kings, and I only have one, but I have clear paths to convert the other pieces. Now, I am ahead three kings to two kings. One of the programs' kings is in a double corner. (If both of the program's kings were in the two opposite double corners, that is a theoretical win for me, but tough.) The program now maneuvers its second king to be by the same double corner as the first game. This is not as tough as when they are in opposite double corners, but still kind of tough! Here is the current situation on my move:



```
Player 2's turn:
   0  1  2  3  4  5  6  7
0
1
2
3        ♚     ♚
4                  ♚
5
6                  ♔
7                     ♔
Turn 95
```

Of course, I am trying to force a trade (since two kings vs. one is an easy win for me); the program is doing a good job resisting. Here is the updated board:



```
Player 2's turn:
   0  1  2  3  4  5  6  7
0
1
2
3
4           ♚     ♚
5                     ♚
6
7              ♔     ♔
Turn 113
```

Finally, I can sneak into the double corner. This still may not be easy though.

A bit later, the program actually forces a trade. I will win easily now. My guess is that within its depth search, it saw that a trade was inevitable (although I had not yet seen how to force it)!

The program marches its king into the opposite double corner. It stays there until I force it out. I inevitably win, but the program pushes it off as long as possible.

I have enough to grade. This is an excellent project. All features have been implemented, plus some additional features (such as undo). The interface is nice. The program makes no clearly bad moves, and it finds good moves when available. It recognizes complex, multiple jumps. You also impressively implemented the game in both C++ and Scheme! The Scheme version searches almost as far with a time limit as the C++ version (I'd say typically, one depth less). Now, I played three games, and I won two with one draw. One of the wins was against the Scheme version. I think the main reason ai won the two games is that the program does not value protecting the back row strongly enough. In both games that I won, I made no mistakes (playing better than I usually play, I think), and got the first king; then I was able to convert that to a gain of a piece. Still, the program plays well. It also knows how to win when it is ahead two kings to one with the opponent's king in a double corner, and it does so quickly. Despite winning two games out of three, I think this program deserves an A+, or at least a borderline A/A+. It would be a clear A+ if it played a bit stronger, to avoid losing. I think with a tweak in the heuristic, it would be very hard to beat. I will count this as a 97. It's a great project.