Troy Daniels

AI Project

Chess:

- This chess game is implemented with all rules except en passant. All pieces move as they should. Any illegal move will be prevented by the game including anytime a player is in check. Pawns get promoted automatically as queens when they reach the enemies last row. Castling is enabled with all constraints.
- To start a game choose the file with the opponent you wish to play and run that file. Your options are HumanVsHuman, HumanVsMinimax (really alpha-beta player) and HumanVsRL. To play the RL, there is a parameter where the RL player is initialized that is the name of a file containing trained weights (string). Enter the correct file of weights you wish to play. Each file will be explained further below.
- To make a move look at the coordinates of each location. The rows are labeled by numbers and columns by letters. After the game starts input the location of the piece you wish to move (letter then number) followed by the location you wish to move the piece with no space in-between (letter then number).
- Chess is a particularly hard game for AI players because of the extremely large state space and the complexity of the game. In class we mentioned that the average branching factor for chess is around 35. This causes the alpha-beta player's runtime to grow very fast as the depth increases. As for the reinforcement learner, it will take a very long time to explore all of these states

causing a slow convergence. The complexity of the game also makes it hard for a value function or features to represent a good state.

Alpha-Beta:

- The alpha-beta player value function is based on the addition of the scores- comparison of pieces, and central control.
    - The pieces are valued at - Queen = 9, Rook = 5, Bishop = 3, Knight =3, pawn = 1 and the king is not included in the comparison. Each player's total amount of points are added up and then the opponents total is subtracted from the players total for the comparison score.
    - The central control is weighted much less than the comparison of pieces. This made sense to me because in a game, going up in pieces is more important than central control. Central control is calculated by giving 0.01 to each piece on the outside square of the board. Then 0.02 for the square inside that, followed by 0.03 and then 0.05 for being in the center. Check out the table below.

0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.01
0.01	0.02	0.03	0.03	0.03	0.03	0.02	0.01
0.01	0.02	0.03	0.05	0.05	0.03	0.02	0.01
0.01	0.02	0.03	0.05	0.05	0.03	0.02	0.01

0.01	0.02	0.03	0.03	0.03	0.03	0.02	0.01
0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.01
0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

- The alpha-beta player is an adept opponent. The depth of the search can be set in the parameters of the initialization of the player. The highest depth where the alpha-beta player makes a move in a reasonable amount of time is 4. At 4 the max amount of time I've seen it take to make a move is about 15 minutes. I have been playing chess for about a year now and am rated around 1400. Although the alpha-beta player at depth 4 almost always loses to me, the games are very competitive. It struggles more in the beginning and end of the game as it has a tendency to make useless moves. It performs very well in the mid-game. The alpha-beta player at depth 3 is most definitely a lesser opponent. However, it is still decent and makes a move with almost no wait. I have seen it beat a friend who is probably rated around 800. Any depth below three, the alpha-beta player becomes easy to beat. I played players of different depths against each other but often the players would just get into loops of repeating moves. If a player does win, it is always the higher depth player. Also, I found that ordering the moves, where attacks are first, helped the runtime of the alpha-beta player due to quicker pruning.

Reinforcement Learner:

- All of the trained players are based on one of two feature vectors. The first being the general feature vector (G) and has a length of 6. This feature vector is based on comparison of pieces, central control, number of pawns around the king (king defense), number of squares the player is attacking, number of opponents pieces the player is attacking and number of pieces of the players pieces being attacked. The other feature vector used is the encoded feature vector (E) and has a length of 70. It contains the same 6 features as the general one (why get rid of them, the learning can always decrease the weights on them if they aren't useful) and 64 other features that encode the board. Each feature the 64 represents a location on the board where the value is based on the piece in that location (positive for your piece, negative for the opponent's piece). For both feature vectors all values are scaled to be in-between 0-1. I was much more focused on the encoded feature vector as it gave the RL more power in deciding what was important.
    - Training of these players was very hard. I was only able to get around 150K games played if I ran it all day.

Players:

	Games Played	ε (explore rate)	opponent	Learning rate	Q-learning
QEncodedPlayer #1	800K+	0.1	QEncodedPlayer #2	0.1	yes

QEncodedPlayer #2	800K+	0.2	QEncodedPlayer#1	0.1	yes
EncodedPlayer #1	250K	0.1	GeneralPlayer #1	0.1	no
GeneralPlayer #1	250K	0.1	EncodedPlayer #1	0.2	no
QEncodedPlayer #2	550K	0.3	Alpha-Beta (Depth 3)	0.1	yes

- Upon playing each player against each other, I found that QEncodedPlayer#2 is the best. However, these players are not good. Nine out of ten games end in a draw (I also have an extra constraint that if there are more than 2000 moves in a game it is a draw). I have played against all of them. The players rarely make a move that makes sense. For the most part they seem like random moves. Furthermore, the reinforcement players lost every game (not including draws) against the depth 1 alpha-beta player. Perhaps, they need to be trained more or something could be off. I noticed that some of the weights became negative for features they are clearly positive for the player. For example, all players set the piece comparison weight as negative. This seems wrong but I was unable to figure out why. I am sure that the complexity of chess also plays a role. I hope to build a neural network when I have the time in hopes that it can find a better function approximator.