heuristic_analysis

June 19, 2017

1 Adding an Endgame

My first idea was to implement an endgame where the agent calculates the length of the longest possible chain of moves. I tested this with severeral scores: #### AB_Custom_1 Nullscore, always outputs zero #### AB_Custom_2 Nullscore + Endgame #### AB_Custom_3 Improved Score + Endgame #### AB_Custom_7 Center Score #### AB_Custom_8 Center Score + Endgame

The results were the following:

In [ ]:

Match #	Opponent	AB_Improved		AB_Custom_1		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	94	6	90	10	92	8	92	8
2	MM_Open	77	23	65	35	67	33	70	30
3	MM_Center	86	14	81	19	84	16	91	9
4	MM_Improved	68	32	68	32	54	46	74	26
5	AB_Open	53	47	43	57	46	54	51	49
6	AB_Center	63	37	44	56	46	54	54	46
7	AB_Improved	55	45	40	60	38	62	56	44
	Win Rate:	70.9%		61.6%		61.0%		69.7%	

In [ ]:

Match #	Opponent	AB_Improved		AB_Custom_7		AB_Custom_8	
		Won	Lost	Won	Lost	Won	Lost
1	Random	95	5	93	7	89	11
2	MM_Open	79	21	72	28	76	24
3	MM_Center	87	13	86	14	87	13
4	MM_Improved	56	44	65	35	62	38
5	AB_Open	57	43	42	58	49	51
6	AB_Center	67	33	47	53	56	44
7	AB_Improved	56	44	42	58	39	61
	Win Rate:	71.0%		63.9%		65.4%	

As we can see the results stayed pretty much the same. Nullscore and Improved Score got a little worse when adding an endgame and Centre Score got a little bit better. I suppose that adding an endgame is computationally too expensive and therefore reduces the depth of the search thus leading too not so good results.

2 Center Score with Manhatten Distance

Next I tried to implement center score just using the manhatten distance instead of the euclidean distance because the manhatten distance is less computationally expensive: #### AB_custom_4 Center Score with euclidean distance #### AB_custom_5 Center Score with manhatten distance #### AB_custom_6 Center Score with Manhatten distance + Endgame

The results were the following:

```
In [ ]: Match #   Opponent    AB_Improved  AB_Custom_4  AB_Custom_5  AB_Custom_6
                                Won | Lost   Won | Lost   Won | Lost   Won | Lost
            1       Random      92  |   8    90  |  10    93  |   7    89  |  11
            2       MM_Open     83  |  17    70  |  30    64  |  36    65  |  35
            3      MM_Center    89  |  11    88  |  12    84  |  16    85  |  15
            4     MM_Improved   69  |  31    70  |  30    65  |  35    57  |  43
            5       AB_Open     48  |  52    51  |  49    44  |  56    37  |  63
            6      AB_Center    62  |  38    51  |  49    54  |  46    47  |  53
            7     AB_Improved   46  |  54    45  |  55    33  |  67    40  |  60
          ----------------------------------------------------------------------
                    Win Rate:      69.9%        66.4%        62.4%        60.0%
```

```
In [ ]: As we can see the results actually got worse.
```

3 Improved Score with Center Score as a Tie Break

This heuristic is basically the same as Improved Score yet it uses Center Score as a Tie Break, that is we add a number in between 0 and 0.9 from a scaled Center Score to the Improved Score. That way if the results for Improved Score are identical for a certain state of the game a decision is made through the Center Score.

3.0.1 AB_custom_9

Improved Score plus Centre Score as tie break, valueing greater distances to the center as positive ### AB_custom_10 Improved Score plus Centre Score as tie break, valueing smaller distances to the center as positive

The results were as follows:

```
In [ ]:  Match #   Opponent    AB_Improved  AB_Custom_9 AB_Custom_10
                                Won | Lost   Won | Lost   Won | Lost
            1       Random      97  |   3    95  |   5    96  |   4
            2       MM_Open     76  |  24    73  |  27    70  |  30
            3      MM_Center    85  |  15    91  |   9    85  |  15
            4     MM_Improved   74  |  26    77  |  23    71  |  29
            5       AB_Open     56  |  44    53  |  47    53  |  47
            6      AB_Center    56  |  44    58  |  42    60  |  40
            7     AB_Improved   44  |  56    51  |  49    51  |  49
          ----------------------------------------------------------
                    Win Rate:      69.7%        71.1%        69.4%
```

As we can see Custom_Score_9 slightly outperforms Improved Score.

4 Knight Move

With the next score I tried to have a look at the mechanics of the game. I get the coordinates of the players location and look at every 3x3 square that contains these coordinates except for when the position would be in the middle of the square. For each of these 3x3 squares I calculate the longest chain of moves the player could make within these squares. In Custom Score 11 I take the maximum of these lengths and in Custom Score 12 I add them up. Custom Score 13 plays Centre Score at the beginning of the game and then switches to the algorithm of Custom Score 12.

AB_custom_11 Knight Move with maximum length #### AB_custom_12 Knight Move with sum of lengths #### AB_custom_13 same as AB_custom_12 but plays center score at the beginning of the game.

The results were as follows:

```
In [ ]: Match #   Opponent    AB_Improved AB_Custom_11 AB_Custom_12
                                Won | Lost   Won | Lost   Won | Lost
            1       Random      91  |   9    92  |   8    89  |  11
            2       MM_Open     76  |  24    75  |  25    76  |  24
            3      MM_Center    87  |  13    89  |  11    86  |  14
            4     MM_Improved   70  |  30    69  |  31    66  |  34
            5       AB_Open     45  |  55    48  |  52    52  |  48
            6      AB_Center    55  |  45    53  |  47    61  |  39
            7     AB_Improved   41  |  59    46  |  54    44  |  56
        ------------------------------------------------------------
                  Win Rate:      66.4%        67.4%        67.7%
```

```
In [ ]:  Match #   Opponent    AB_Improved AB_Custom_13
                                Won | Lost   Won | Lost
            1       Random      88  |  12    91  |   9
            2       MM_Open     72  |  28    77  |  23
            3      MM_Center    88  |  12    88  |  12
            4     MM_Improved   74  |  26    74  |  26
            5       AB_Open     53  |  47    47  |  53
            6      AB_Center    61  |  39    59  |  41
            7     AB_Improved   48  |  52    53  |  47
        ------------------------------------------------
                  Win Rate:      69.1%        69.9%
```

As we can see the scores slightly outperformed Improved Score, Custom Score 12 having the largest distance to Improved Score.

5 Conclusion

We use Custom Score 12 as Custom Score because it performed best in comparison to Improved Score looking at the difference of the percentages of games won (67.7-66.4=1.3). Furthermore it won 44 out of 100 games against Improved Score suggesting that both scores are of similiar strength. It also takes into account the mechanics of the game by looking at how the knight can move in chess.

6 Implementation

```
In [ ]: def custom_score_1(game,player):
            if game.is_loser(player):
                return float("-inf")

            if game.is_winner(player):
                return float("inf")
            return 0

        def custom_score_2(game,player):
            if game.is_loser(player):
                return float("-inf")

            if game.is_winner(player):
                return float("inf")

            def going_on(game_state,l=0,m_max=0):
                #print("begin")
                #print("l={}".format(l))
                legal_moves=game_state.get_legal_moves(player)
                #print("legal_moves: {}".format(legal_moves))
                if len(legal_moves)==0:
                    if l>m_max:
                        m_max=l
                    #print("m_max={}".format(m_max))
                    return m_max
                for m in legal_moves:
                   #print("m: {}".format(m))
                   game_state._active_player=player
                   m_max=going_on(game_state.forecast_move(m),l+1,m_max)
                #print("max={}".format(m_max))
                return m_max

            if len(game.get_blank_spaces())<=15:
                game_state=game.copy()
                x,y =game_state.get_player_location(player)
                #print("x={},y={}".format(x,y))
                m_max=going_on(game_state)
                #print("final_max={}".format(m_max))
                return m_max
            else:
               return 0



        def custom_score_3(game,player):
            if game.is_loser(player):
```

```
            return float("-inf")

        if game.is_winner(player):
            return float("inf")

        def going_on(game_state,l=0,m_max=0):
            #print("begin")
            #print("l={}".format(l))
            legal_moves=game_state.get_legal_moves(player)
            #print("legal_moves: {}".format(legal_moves))
            if len(legal_moves)==0:
                if l>m_max:
                    m_max=l
                #print("m_max={}".format(m_max))
                return m_max
            for m in legal_moves:
               # print("m: {}".format(m))
                game_state._active_player=player
                m_max=going_on(game_state.forecast_move(m),l+1,m_max)
            #print("max={}".format(m_max))
            return m_max

        if len(game.get_blank_spaces())<=15:
            game_state=game.copy()
            x,y =game_state.get_player_location(player)
            #print("x={},y={}".format(x,y))
            m_max=going_on(game_state)
            #print("final_max={}".format(m_max))
            return m_max
        else:
            own_moves = len(game.get_legal_moves(player))
            opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
            return float(own_moves - opp_moves)
```

In [ ]:
```
def custom_score_4(game,player):

        if game.is_loser(player):
            return float("-inf")

        if game.is_winner(player):
            return float("inf")


        y, x = game.get_player_location(player)
        return float((3 - y)**2 + (3 - x)**2)

    def custom_score_5(game,player):
```

```
        if game.is_loser(player):
            return float("-inf")

        if game.is_winner(player):
            return float("inf")


        y, x = game.get_player_location(player)
        return float(abs(3 - y) + abs(3 - x))

    def custom_score_6(game,player):

        if game.is_loser(player):
            return float("-inf")

        if game.is_winner(player):
            return float("inf")

        def going_on(game_state,l=0,m_max=0):
            #print("begin")
            #print("l={}".format(l))
            legal_moves=game_state.get_legal_moves(player)
            #print("legal_moves: {}".format(legal_moves))
            if len(legal_moves)==0:
                if l>m_max:
                    m_max=l
                #print("m_max={}".format(m_max))
                return m_max
            for m in legal_moves:
               # print("m: {}".format(m))
                game_state._active_player=player
                m_max=going_on(game_state.forecast_move(m),l+1,m_max)
            #print("max={}".format(m_max))
            return m_max

        if len(game.get_blank_spaces())<=15:
            game_state=game.copy()
            x,y =game_state.get_player_location(player)
            #print("x={},y={}".format(x,y))
            m_max=going_on(game_state)
            #print("final_max={}".format(m_max))
            return m_max

        y, x = game.get_player_location(player)
        return float(abs(3 - y) + abs(3 - x))

In [ ]: def custom_score_7(game,player):
```

```
    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")


    y, x = game.get_player_location(player)
    return float((3 - y)**2 + (3 - x)**2)



def custom_score_8(game,player):

    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    def going_on(game_state,l=0,m_max=0):
        #print("begin")
        #print("l={}".format(l))
        legal_moves=game_state.get_legal_moves(player)
        #print("legal_moves: {}".format(legal_moves))
        if len(legal_moves)==0:
            if l>m_max:
                m_max=l
            #print("m_max={}".format(m_max))
            return m_max
        for m in legal_moves:
           # print("m: {}".format(m))
            game_state._active_player=player
            m_max=going_on(game_state.forecast_move(m),l+1,m_max)
        #print("max={}".format(m_max))
        return m_max

    if len(game.get_blank_spaces())<=15:
        game_state=game.copy()
        x,y =game_state.get_player_location(player)
        #print("x={},y={}".format(x,y))
        m_max=going_on(game_state)
        #print("final_max={}".format(m_max))
        return m_max

    y, x = game.get_player_location(player)
    return float((3 - y)**2 +(3 - x)**2)
```

```
In [ ]: def custom_score_9(game,player):

            if game.is_loser(player):
                return float("-inf")

            if game.is_winner(player):
                return float("inf")

            y, x = game.get_player_location(player)
            d= float((3 - y)**2 + (3 - x)**2)/10.
            own_moves = len(game.get_legal_moves(player))
            opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
            return float(own_moves - opp_moves)+d



        def custom_score_10(game,player):

            if game.is_loser(player):
                return float("-inf")

            if game.is_winner(player):
                return float("inf")

            y, x = game.get_player_location(player)
            d= float((3 - y)**2 + (3 - x)**2)/10.
            own_moves = len(game.get_legal_moves(player))
            opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
            return float(own_moves - opp_moves)-d


In [ ]: def check(x,y,blank,move_list,move_list2):
            l=0
            l2=0
            x1=x
            y1=y
            for u,v in move_list:
                    x1=x1+u
                    y1=y1+v
                    if (x1,y1) in blank:
                        l=l+1
                    else:
                        break
            if l!=7:
                x2=x
                y2=y
                for u,v in move_list2:
                    x2=x2+u
```

```
            y2=y2+v
            if (x2,y2) in blank:
                l2=l2+1
            else:
                break
    return max(l,l2)


def custom_score_(game,player):
    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")



    blank=game.get_blank_spaces()
    x,y = game.get_player_location(player)

    a=[(2,1),(-2,1),(1,-2),(1,2),(-2,-1),(2,-1),(-1,2)]
    b=[(1, 2), (1, -2), (-2, 1), (2, 1), (-1, -2), (-1, 2), (2, -1)]
    c=[(2,-1),(-2,-1),(1,2),(1,-2),(-2,1),(2,1),(-1,-2)]
    d=[(1, -2), (1, 2), (-2, -1), (2, -1), (-1, 2), (-1, -2), (2, 1)]
    a2=[(-2, -1), (2, -1), (-1, 2), (-1, -2), (2, 1), (-2, 1), (1, -2)]
    b2=[(-1, -2), (-1, 2), (2, -1), (-2, -1), (1, 2), (1, -2), (-2, 1)]
    c2=[(-2, 1), (2, 1), (-1, -2), (-1, 2), (2, -1), (-2, -1), (1, 2)]
    d2=[(-1, 2), (-1, -2), (2, 1), (-2, 1), (1, -2), (1, 2), (-2, -1)]
    e=[(2,-1),(-1,2),(-1,-2),(2,1),(-2,1),(1,-2),(1,2)]
    f=[(2,1),(-1,-2),(-1,2),(2,-1),(-2,-1),(1,2),(1,-2)]
    e2=[(-2, -1), (1, 2), (1, -2), (-2, 1), (2, 1), (-1, -2), (-1, 2)]
    f2=[(-2, 1), (1, -2), (1, 2), (-2, -1), (2, -1), (-1, 2), (-1, -2)]
    g=[(1,-2),(-2,1),(2,1),(-1,-2),(-1,2),(2,-1),(-2,-1)]
    h=[(-1,-2),(2,1),(-2,1),(1,-2),(1,2),(-2,-1),(2,-1)]
    g2=[(1, 2), (-2, -1), (2, -1), (-1, 2), (-1, -2), (2, 1), (-2, 1)]
    h2=[(-1, 2), (2, -1), (-2, -1), (1, 2), (1, -2), (-2, 1), (2, 1)]

    l_max=0

    if (x+2<=6) and (y+2<=6):
        l=check(x,y,blank,a,b)
        if l>l_max:
            l_max=l

    if ((x+2<=6) and (y-2>=0)) and l_max!=7:
        l=check(x,y,blank,c,d)
        if l>l_max:
            l_max=l
```

```
    if ((x-2>=0) and (y-2>=0)) and l_max!=7:
        l=check(x,y,blank,a2,b2)
        if l>l_max:
            l_max=l

    if ((x-2>=0) and (y+2<=6)) and l_max!=7:
        l=check(x,y,blank,c2,d2)
        if l>l_max:
            l_max=l

    if ((x+2<=6)and((y+1<=6)and(y-1>=0))) and l_max!=7:
        l=check(x,y,blank,e,f)
        if l>l_max:
            l_max=l

    if ((x-2>=0)and((y+1<=6)and(y-1>=0))) and l_max!=7:
        l=check(x,y,blank,e2,f2)
        if l>l_max:
            l_max=l

    if ((y-2>=0)and((x+1<=6)and(x-1>=0))) and l_max!=7:
        l=check(x,y,blank,g,h)
        if l>l_max:
            l_max=l

    if ((y+2<=6)and((x+1<=6)and(x-1>=0))) and l_max!=7:
        l=check(x,y,blank,g2,h2)
        if l>l_max:
            l_max=l


    return float(l_max)

def check_a(x,y,blank,move_list,move_list2):
    l=0
    l2=0
    x1=x
    y1=y
    for u,v in move_list:
            x1=x1+u
            y1=y1+v
            if (x1,y1) in blank:
                l=l+1
            else:
                break

    x2=x
```

```
        y2=y
        for u,v in move_list2:
            x2=x2+u
            y2=y2+v
            if (x2,y2) in blank:
                l2=l2+1
            else:
                break

        return l+l2


    def custom_score_(game,player):
        if game.is_loser(player):
            return float("-inf")

        if game.is_winner(player):
            return float("inf")



        blank=game.get_blank_spaces()
        x,y = game.get_player_location(player)

        a=[(2,1),(-2,1),(1,-2),(1,2),(-2,-1),(2,-1),(-1,2)]
        b=[(1, 2), (1, -2), (-2, 1), (2, 1), (-1, -2), (-1, 2), (2, -1)]
        c=[(2,-1),(-2,-1),(1,2),(1,-2),(-2,1),(2,1),(-1,-2)]
        d=[(1, -2), (1, 2), (-2, -1), (2, -1), (-1, 2), (-1, -2), (2, 1)]
        a2=[(-2, -1), (2, -1), (-1, 2), (-1, -2), (2, 1), (-2, 1), (1, -2)]
        b2=[(-1, -2), (-1, 2), (2, -1), (-2, -1), (1, 2), (1, -2), (-2, 1)]
        c2=[(-2, 1), (2, 1), (-1, -2), (-1, 2), (2, -1), (-2, -1), (1, 2)]
        d2=[(-1, 2), (-1, -2), (2, 1), (-2, 1), (1, -2), (1, 2), (-2, -1)]
        e=[(2,-1),(-1,2),(-1,-2),(2,1),(-2,1),(1,-2),(1,2)]
        f=[(2,1),(-1,-2),(-1,2),(2,-1),(-2,-1),(1,2),(1,-2)]
        e2=[(-2, -1), (1, 2), (1, -2), (-2, 1), (2, 1), (-1, -2), (-1, 2)]
        f2=[(-2, 1), (1, -2), (1, 2), (-2, -1), (2, -1), (-1, 2), (-1, -2)]
        g=[(1,-2),(-2,1),(2,1),(-1,-2),(-1,2),(2,-1),(-2,-1)]
        h=[(-1,-2),(2,1),(-2,1),(1,-2),(1,2),(-2,-1),(2,-1)]
        g2=[(1, 2), (-2, -1), (2, -1), (-1, 2), (-1, -2), (2, 1), (-2, 1)]
        h2=[(-1, 2), (2, -1), (-2, -1), (1, 2), (1, -2), (-2, 1), (2, 1)]

        l_sum=0

        if (x+2<=6) and (y+2<=6):
            l=check_a(x,y,blank,a,b)
            l_sum=l_sum+l

        if ((x+2<=6) and (y-2>=0)):
```

```
            l=check_a(x,y,blank,c,d)
            l_sum=l_sum+l

        if ((x-2>=0) and (y-2>=0)):
            l=check_a(x,y,blank,a2,b2)
            l_sum=l_sum+l

        if ((x-2>=0) and (y+2<=6)):
            l=check_a(x,y,blank,c2,d2)
            l_sum=l_sum+l

        if ((x+2<=6)and((y+1<=6)and(y-1>=0))):
            l=check_a(x,y,blank,e,f)
            l_sum=l_sum+l

        if ((x-2>=0)and((y+1<=6)and(y-1>=0))):
            l=check_a(x,y,blank,e2,f2)
            l_sum=l_sum+l

        if ((y-2>=0)and((x+1<=6)and(x-1>=0))):
            l=check_a(x,y,blank,g,h)
            l_sum=l_sum+l

        if ((y+2<=6)and((x+1<=6)and(x-1>=0))):
            l=check_a(x,y,blank,g2,h2)
            l_sum=l_sum+l

        return float(l_sum)
```

In [ ]:
```
def custom_score_13(game,player):
    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    blank=game.get_blank_spaces()

    if len(blank)>=30:
        x,y = game.get_player_location(player)
        return float((3 - y)**2 + (3 - x)**2)
    else:
        x,y = game.get_player_location(player)

        a=[(2,1),(-2,1),(1,-2),(1,2),(-2,-1),(2,-1),(-1,2)]
        b=[(1, 2), (1, -2), (-2, 1), (2, 1), (-1, -2), (-1, 2), (2, -1)]
        c=[(2,-1),(-2,-1),(1,2),(1,-2),(-2,1),(2,1),(-1,-2)]
        d=[(1, -2), (1, 2), (-2, -1), (2, -1), (-1, 2), (-1, -2), (2, 1)]
```

```
a2=[(-2, -1), (2, -1), (-1, 2), (-1, -2), (2, 1), (-2, 1), (1, -2)]
b2=[(-1, -2), (-1, 2), (2, -1), (-2, -1), (1, 2), (1, -2), (-2, 1)]
c2=[(-2, 1), (2, 1), (-1, -2), (-1, 2), (2, -1), (-2, -1), (1, 2)]
d2=[(-1, 2), (-1, -2), (2, 1), (-2, 1), (1, -2), (1, 2), (-2, -1)]
e=[(2,-1),(-1,2),(-1,-2),(2,1),(-2,1),(1,-2),(1,2)]
f=[(2,1),(-1,-2),(-1,2),(2,-1),(-2,-1),(1,2),(1,-2)]
e2=[(-2, -1), (1, 2), (1, -2), (-2, 1), (2, 1), (-1, -2), (-1, 2)]
f2=[(-2, 1), (1, -2), (1, 2), (-2, -1), (2, -1), (-1, 2), (-1, -2)]
g=[(1,-2),(-2,1),(2,1),(-1,-2),(-1,2),(2,-1),(-2,-1)]
h=[(-1,-2),(2,1),(-2,1),(1,-2),(1,2),(-2,-1),(2,-1)]
g2=[(1, 2), (-2, -1), (2, -1), (-1, 2), (-1, -2), (2, 1), (-2, 1)]
h2=[(-1, 2), (2, -1), (-2, -1), (1, 2), (1, -2), (-2, 1), (2, 1)]

l_sum=0

if (x+2<=6) and (y+2<=6):
    l=check_a(x,y,blank,a,b)
    l_sum=l_sum+l

if ((x+2<=6) and (y-2>=0)):
    l=check_a(x,y,blank,c,d)
    l_sum=l_sum+l

if ((x-2>=0) and (y-2>=0)):
    l=check_a(x,y,blank,a2,b2)
    l_sum=l_sum+l

if ((x-2>=0) and (y+2<=6)):
    l=check_a(x,y,blank,c2,d2)
    l_sum=l_sum+l

if ((x+2<=6)and((y+1<=6)and(y-1>=0))):
    l=check_a(x,y,blank,e,f)
    l_sum=l_sum+l

if ((x-2>=0)and((y+1<=6)and(y-1>=0))):
    l=check_a(x,y,blank,e2,f2)
    l_sum=l_sum+l

if ((y-2>=0)and((x+1<=6)and(x-1>=0))):
    l=check_a(x,y,blank,g,h)
    l_sum=l_sum+l

if ((y+2<=6)and((x+1<=6)and(x-1>=0))):
    l=check_a(x,y,blank,g2,h2)
    l_sum=l_sum+l

return float(l_sum)
```