Heuristic Analysis

Finding a heuristic performing better than the AB_Improved agent has been a difficult task. Proving the heuristic is statistically better than AB_Improved was as leghty, and resource consuming as it was difficult.

When I was about to give up, I turned to 2500 years old wisdom and the Sun Tzu heuristic took shape. It was challenging to find the right model and even more challenging to find the right parameters. I tried the heuristic with three sets of parameters. The main lesson from working with the parameters was: ignore the General at one's own peril.

Variant 1	**Variant 2**	**Variant 3**
dispersive_ground_weight = 2.0	dispersive_ground_weight = 2.0	dispersive_ground_weight = 4.0
facile_ground_weight = 4.0	facile_ground_weight = 2.0	facile_ground_weight = 16.0
contentious_ground_weight = 2.0	contentious_ground_weight = 2.0	contentious_ground_weight = 2.0
open_ground_weight = 2.0	open_ground_weight = 2.0	open_ground_weight = 2.0
serious_ground_weight = 8.0	serious_ground_weight = 2.0	serious_ground_weight = 64.0
difficult_ground_weight = 4.0	difficult_ground_weight = 2.0	difficult_ground_weight = 16.0
hemmed_in_ground_weight = 8.0	hemmed_in_ground_weight = 2.0	hemmed_in_ground_weight = 64.0

The third variant was the only one proven to be statistically better than the AB_Improved agent. When Sun Tzu says run, one must run, when Sun Tzu says plunder, one must plunder :)

XI. THE NINE SITUATIONS

1. Sun Tzŭ said: The art of war recognises nine varieties of ground:

```
(1) dispersive ground;
(2) facile ground;
(3) contentious ground;
(4) open ground;
(5) ground of intersecting highways;
(6) serious ground;
(7) difficult ground;
(8) hemmed-in ground;
(9) desperate ground.
```

1. When a chieftain is fighting in his own territory, it is dispersive ground.
2. When he has penetrated into hostile territory, but to no great distance, it is facile ground.
3. Ground the possession of which imports great advantage to either side, is contentious ground.
4. Ground on which each side has liberty of movement is open ground.
5. Ground which forms the key to three contiguous states, so that he who occupies it first has most of the Empire at his command, is ground of intersecting highways.
6. When an army has penetrated into the heart of a hostile country, leaving a number of fortified cities in its rear, it is serious ground.
7. Mountain forests, rugged steeps, marshes and fens—all country that is hard to traverse: this is difficult ground.
8. Ground which is reached through narrow gorges, and from which we can only retire by tortuous paths, so that a small number of the enemy would suffice to crush a large body of our men: this is hemmed-in ground.
9. Ground on which we can only be saved from destruction by fighting without delay, is desperate ground.

1. On dispersive ground, therefore, fight not. On facile ground, halt not. On contentious ground, attack not.
2. On open ground, do not try to block the enemy's way. On ground of intersecting highways, join hands with your allies.
3. On serious ground, gather in plunder. On difficult ground, keep steadily on the march.
4. On hemmed-in ground, resort to stratagem. On desperate ground, fight.

Isolation game types of "ground"

```
(1) dispersive ground - the opponent has more moves than the player
(2) facile ground - the player has more moves than the opponent
(3) contentious ground - the number of common moves
(4) open ground - center moves
(5) intersecting highways - will ignore for a two player game
(6) serious ground - the opponent has much less moves than the player
(7) difficult ground - precious few available moves for the player
(8) hemmed-in ground - very, very few available moves for the player
(9) desperate ground - dire situation
```

Isolation game heuristics

1. penalize the dispersive ground -
2. encourage the facile ground
3. penalize the contentious ground
4. encourage the open ground, penalize the common moves
5. ignored
6. encourage++ the serious ground
7. penalize ++ the difficult ground
8. penalize +++ the hemmed-in ground
9. forfait

The Sun-Tzu heuristic

This heuristic is based on Sun-Tzu recommendations above. We modelled the recommendations via intial weights and the ratio of player vs opponent moves. Through a good amount of trial and error we noticed the heuristic performs differenty over 21,000 games for different intial weights. We kept the weights for the centrality (open ground) the same.

```
    def sun_tzu_heuristic(game, player):

        if game.is_winner(player) or game.is_loser(player):
            return game.utility(player)

        dispersive_ground_weight = 4.0   #fight not
        facile_ground_weight = 16.0      #fight and halt not
        contentious_ground_weight = 2.0  #attack not, don't be in the enemy's way
        open_ground_weight = 2.0         #don't be in enemy's way
        serious_ground_weight = 64.0     #PLUNDER!
        difficult_ground_weight = 16.0   #keep going, don't fight
        hemmed_in_ground_weight = 64.0   #Run, Forest, run!

        central_column = math.ceil(game.width/2.)
        central_row = math.ceil(game.height/2.)
        num_all_available_moves = float(game.width * game.height)
        num_blank_spaces = len(game.get_blank_spaces())
        decay = num_blank_spaces/num_all_available_moves

        player_moves = game.get_legal_moves(player)
        opponent_moves = game.get_legal_moves(game.get_opponent(player))
        num_player_moves = len(player_moves)
        num_opponent_moves = len( opponent_moves )
        common_moves = len(find_common_moves(game, player))

        dispersive_ground = num_opponent_moves/num_player_moves if num_player_moves !=0 else num_opponent_moves
        facile_ground = num_player_moves/num_opponent_moves if num_opponent_moves !=0 else num_player_moves
        contentious_ground = common_moves/num_blank_spaces if common_moves !=0 else 1.
        serious_ground = num_player_moves/num_opponent_moves if num_opponent_moves !=0 else num_player_moves
        difficult_ground = num_opponent_moves/num_player_moves if num_player_moves !=0 else num_opponent_moves
        hemmed_in_ground = num_opponent_moves/num_player_moves if num_player_moves !=0 else num_opponent_moves

        dispersive_ground = dispersive_ground if dispersive_ground !=0 else 1
        facile_ground = facile_ground if facile_ground !=0 else 1
        serious_ground = serious_ground if serious_ground != 0 else 1
        difficult_ground = difficult_ground if difficult_ground !=0 else 1
        hemmed_in_ground = hemmed_in_ground if hemmed_in_ground !=0 else 1

        player_weight=1.0
        opponent_weight = 2.0

        #handle open_ground/center moves
        for move in player_moves:
            if move[0] == central_row or move[1] == central_column:
                player_weight *= (open_ground_weight * decay)

        for move in opponent_moves:
            if move[0] == central_row or move[1] == central_column:
                opponent_weight  *= (open_ground_weight * decay)

        #factor ground weights only once
        player_weight *= facile_ground_weight*serious_ground_weight
        opponent_weight *= dispersive_ground_weight*contentious_ground_weight*difficult_ground_weight*hemmed_in_ground_weight

        player_weight *= (decay * facile_ground) * (decay * serious_ground)
        opponent_weight *= (decay * dispersive_ground) * (decay * contentious_ground) * (decay * difficult_ground) * (decay * hemmed_in_ground)

        return float((num_player_moves * player_weight) -
            (num_opponent_moves * opponent_weight))
```

The competing heuristics

This is the code of the other two competing heuristic:

```
    def added_centrality_sum_heuristic(game, player):
        opponent = game.get_opponent(player)
        opponent_moves = game.get_legal_moves(opponent)
        player_moves = game.get_legal_moves()
        if not opponent_moves:
            return float("inf")
        if not player_moves:
            return float("-inf")
        return float(len(player_moves) - len(opponent_moves) +
            sum(centrality_measure(game, m) for m in player_moves) +
            common_moves_measure(game, player) +
            interfering_moves_measure(game, player))

    def weighted_centrality_heuristic(game, player):

        if game.is_winner(player) or game.is_loser(player):
            return game.utility(player)

        central_column = math.ceil(game.width/2.)
        central_row = math.ceil(game.height/2.)
        num_all_available_moves = float(game.width * game.height)
        num_blank_spaces = len(game.get_blank_spaces())
        decay = num_blank_spaces/num_all_available_moves

        player_moves = game.get_legal_moves(player)
        opponent_moves = game.get_legal_moves(game.get_opponent(player))
        num_player_moves = len(player_moves)
        num_opponent_moves = len( opponent_moves )

      '''
        opponent_weight - avoid choices where the opponent has more moves
        center_weight - favorize the center more
        '''
        player_weight = 1
        opponent_weight = 2
        center_weight = 2

        for move in player_moves:
            if move[0] == central_row or move[1] == central_column:
                player_weight *= (center_weight * decay)

        for move in opponent_moves:
            if move[0] == central_row or move[1] == central_column:
                opponent_weight  *= (center_weight * decay)

        return float((num_player_moves * player_weight) -
            (num_opponent_moves * opponent_weight))
```

In [14]:
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sn
```

The data set contains the win rates for 350 runs of the tournament.py. Each tournament plays 70 games for each custom agent in competition. There are about 23,100 games played for each heuristic.

In [15]:
```
tournament_results = pd.read_csv('variant-3.csv')
tournament_results.head()
#tournament_results["AB_Custom3"]
```

Out[15]:

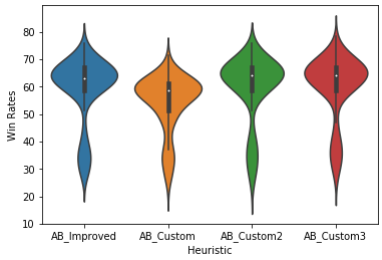
	AB_Improved	AB_Custom	AB_Custom2	AB_Custom3	Unnamed: 4
0	68.6	64.3	67.1	68.6	1.0
1	61.4	48.6	67.1	67.1	NaN
2	58.6	60.0	64.3	61.4	NaN
3	60.0	62.9	67.1	62.9	NaN
4	62.9	51.4	67.1	60.0	NaN

In [16]:
```
tournament_results[[0, 1, 2, 3]].describe()
```

Out[16]:

	AB_Improved	AB_Custom	AB_Custom2	AB_Custom3
count	350.000000	350.000000	350.000000	350.000000
mean	59.111429	54.246000	59.467143	59.953143
std	12.059109	10.490024	12.413517	11.917610
min	25.700000	21.400000	21.400000	24.300000
25%	58.600000	51.400000	58.600000	58.600000
50%	62.900000	58.600000	64.300000	64.300000
75%	67.100000	61.400000	67.100000	67.100000
max	75.700000	71.400000	75.700000	78.600000

In [17]:
```
sn.violinplot(data=tournament_results[[0, 1, 2, 3]], split=True)
plt.ylabel('Win Rates')
plt.xlabel('Heuristic')
plt.show()
```



Hypothesis

We'd like to determine if there is a significant difference in the mean win rates recorded for the AB_Improved heuristic and the win rates recorded for the Sun Tzu heuristic with the parameters above. The AB_Improved heuristic scored a mean average of $\bar{x}_h = 59.111429$. The mean average for the Sun Tzu Heuristic is $\bar{x}_f = 59.953143$ We choose as our test criterium an alpha level of 5%. So our hypothesis states:

Hypothesis		$\alpha = .05$
$H_0:$	$\mu_h = \mu_f$	There is no significant difference in mean win rates for the two heurisics
$H_a:$	$\mu_h \neq \mu_f$	There is a significant difference in mean win rates for the two heuristics

The hypothesis is written to denote the whole population, not just the sample.

In [18]:
```
t_statistic, p = stats.ttest_rel(tournament_results["AB_Improved"], tournament_results["AB_Custom3"])
t_statistic, p
```

Out[18]:
```
(-2.3923770969291529, 0.017268012809475139)
```

In [19]:
```
# To test our alternative hypothesis
alpha = .05
p < alpha / 2 # two sided
```

Out[19]:
```
True
```

The Sun Tzu heuristic is statistically bettr than the AB_Improved heuristic

We perform the same check for the AB_Custom heuristic.

In [20]:
```
t_statistic, p = stats.ttest_rel(tournament_results["AB_Improved"], tournament_results["AB_Custom"])
t_statistic, p
```

Out[20]:
```
(12.918557526176274, 1.7975440964753182e-31)
```

In [21]:
```
# To test our alternative hypothesis
alpha = .05
p < alpha / 2 # two sided
```

Out[21]:
```
True
```

In this case we could say that the AB_Custom heuristic is worse than the AB_Improved heurstic since it has a statistically lower win rate.

We perform the two-sided hypothesis check for the center with decay heuristic [AB_Custom2]:

In [22]:
```
t_statistic, p = stats.ttest_rel(tournament_results["AB_Improved"], tournament_results["AB_Custom2"])
t_statistic, p
```

Out[22]:
```
(-1.0125620249553686, 0.31197087896992376)
```

In [23]:
```
# To test our alternative hypothesis
alpha = .05
p < alpha / 2 # two sided
```

Out[23]:
```
False
```

AB_Custom2 heuristic is not statistically better than the AB_Improved heuristic.

Recommendations

Based on the results above, the recommended heuristic is the Sun Tzu heuristic:

1. It achieves better win rates than the AB_Improved agent
2. It performs better than the other two heuristics
3. The results are statistically significant
4. Although AB_Custom2 (center based heuristic with decay) achieved better average win rates, the results are not statistically significant

Future possible work

Apparently higher weights inspired by Sun Tzu recommendations (plunder and run) produce better results, as the three variants considered above show. This is an interesting hypothesis, perhaps worth to be checked.

In [ ]:
```
```