Stuart Minshull

AI Nanodegree, Udacity

July 1st, 2017

Isolation Knight-Variant: Heuristic Analysis

As part of the AI Nanodegree Project 'Build a Game-Playing Agent', I wrote a program that used artificial intelligence techniques such as alpha-beta pruning and iterative deepening search. To evaluate the states found as game trees were traversed, I developed three custom heuristic based on my experiences playing the Isolation Knight-Variant with friends and against computer players.

During gameplay, I noticed three notable strategies that tended to have a beneficial effect on the game: picking moves that left you with the most moves on the next turn, trying to stay near the center, and staying close enough to the opponent that it was difficult for them to trap you inside a small partition. My heuristics attempt to distill these strategies into quickly executable scores that an AI agent can use to evaluate the 'goodness' of a move.

To evaluate the strength of each heuristic, I implemented the provided function prototypes inside of game_agent.py. Then, I increased the ***NUM_MATCHES*** variable inside of tournament.py to **10** to have a large sample size and larger confidence in the performance of my heuristics. Hence, each other agent was played 20 times for a total of 140 matches per heuristic.

Custom Heuristic 1: 'Hybrid Score' | 65.0%-win rate

This heuristic was the last heuristic I developed, and is a blend of all three aforementioned strategies: to prefer moves with the most child moves, to prefer moves near the center, and to prefer moves closer to your opponent. This heuristic calculates *center_score*, the the square Euclidean distance of the player to the center, *opp_score*, the square Euclidean distance of the player to the opponent, and *move_score*, the number of legal moves a player would have after taking the current moves.

The three scores are combined in a weighted sum, at a ratio of:

$$hybrid\_score = 1.5 * move\_score + 1.75 * center\_score + 0.7 * opp\_score$$

The weights of each subscore were determined experimentally through repeated trials of the game-playing tournament.

This heuristic achieved a 65.0% total win rate in the tournament. The performance of this heuristic against each is shown in the breakout table below:

Opponent	Won	Lost	Win Rate
Random	19	1	95%
MiniMax_Open	7	13	35%
MiniMax_Center	20	0	100%
MiniMax_Improved	13	7	65%
Alpha-Beta_Open	10	10	50%
Alpha-Beta_Center	15	5	75%
Alpha-Beta_Improved	7	13	35%
Total	**91**	**49**	**65.0%**

Fig 1: Performance of Custom Heuristic 1: 'Hybrid Score' in the agent tournament

```
# Create a composite score that blends preferring the center, preferring
# moves with more descendants, and staying close to the opponent

# Minimize the combined distance between the center of the other board
# The idea is to keep to the center, but do not let the other player wall
# you in - stay close enough to your opponent that he can't box you out.
# While doing this, prefer moves with more open possibilities.

# Check win/loss conditions
if game.is_loser(player):
    return float("-inf")

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

# Find the center of the board
center_width = game.width/2.0
center_height = game.height/2.0

# Find the player's position
player_height, player_width = game.get_player_location(player)

# Find the opponent's position
opp_height, opp_width = game.get_player_location(game.get_opponent(player))

# Find the NEGATIVE Square Euclidean distance to the other player, as we want
# to prefer positions closer to our opponents
opp_score = -((opp_height - player_height)**2 + (opp_width - player_width)**2)

# Find the NEGATIVE Square Euclidean Distance to the center of the board,
# as we want to prefer positions closer to center
center_score = -((center_height - player_height)**2 + (center_width - player_width)**2)

# Find the number of open moves
move_score = len(game.get_legal_moves(player))

# Combined

return (1.5*move_score + 1.75*center_score + 0.7*opp_score)
```

Fig 2: Implementation of the Hybrid Score heuristic

Custom Heuristic 2: 'Enemies Close, Center Closer Score' | 63.6%-win rate

This was the second heuristic I developed – by staying closer to the center of the board, we minimize the chance that we get trapped in a corner or very small partition. In addition, we prefer moves closer to our opponent: due to the knight movements of the game, being directly adjacent to an opponent makes it difficult for them to trap you in a partition. However, chasing an opponent around too much can lead you into easy traps, so this heuristic prefers staying to the center by overweighting the center score.

This heuristic is calculated as a weighted sum of the square Euclidean distance from the player to the opponent and the square Euclidean distance from the player to the center of the board.

$$eccc\_score = opp\_score + 1.75 * center\_score,$$

Note that we cast both scores negative – we want to prefer moves *closer* to the center and the opponent, so as the distance grows, we want the heuristic to evaluate better. The agent interprets larger numbers as better moves, so we add the negative sign to make smaller distances evaluate better than larger distances.

The weights of each sub-score were determined experimentally through repeated trials of the game-playing tournament.

This heuristic achieved a 63.6% total win rate in the tournament. The performance of this heuristic against each other agent is shown in the breakout table below:

Opponent	Won	Lost	Win Rate
Random	18	2	90%
MiniMax_Open	7	13	35%
MiniMax_Center	17	3	85%
MiniMax_Improved	8	12	40%
Alpha-Beta_Open	12	8	60%
Alpha-Beta_Center	17	3	85%
Alpha-Beta_Improved	10	10	50%
Total	**89**	**51**	**63.6%**

Fig 3: Performance of Custom Heuristic 2: 'Enemies Close, Center Closer' in the agent tournament

```
# Minimize the combined distance between the center of the other board
# The idea is to keep to the center, but do not let the other player wall
# you in - stay close enough to your opponent that he can't box you out,
# but prefer moves that are farther from the corners to avoid traps

# Check win/loss conditions
if game.is_loser(player):
    return float("-inf")

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

# Find the center of the board
center_width = game.width/2.0
center_height = game.height/2.0

# Find the player's position
player_height, player_width = game.get_player_location(player)

# Find the opponent's position
opp_height, opp_width = game.get_player_location(game.get_opponent(player))

# Find the NEGATIVE Square Euclidean distance to the other player, as we want
# to prefer positions closer to our opponents
oppdist = -((opp_height - player_height)**2 + (opp_width - player_width)**2)

# Find the NEGATIVE Square Euclidean Distance to the center of the board,
# as we want to prefer positions closer to center
centdist = -((center_height - player_height)**2 + (center_width - player_width)**2)

# Prefer center positions more
return float(oppdist + 1.75*centdist)
```

Fig 4: Implementation of the Enemies Close, Center Closer heuristic

Custom Heuristic 3: 'Open-Center Score' | 57.9%-win rate

This was the first heuristic I developed – we want to prefer moves that leave us the most possibilities on the next turn, and stay to the center to avoid an opponent trapping us in a partition or in a corner. However, while the Open-Center score performed better than the AB Improved baseline, without accounting for the strategic gain of being closer to your opponent, this heuristic performed the worst of my three heuristics.

We calculate this heuristic as the weighted sum of the number of legal moves a player has after taking the current move, and the Manhattan distance of the player to the center

$$oc\_score = 1.5*move\_score - ((abs(center\_h - player\_h) + abs(center\_w - player\_w))$$

Note that we subtract the Manhattan distance, as we want to prefer moves closer to the center. The Manhattan distance is minimal when the player is in the center of the board, and maximal in the corners. As heuristics that are more positive are considered better, we subtract higher Manhattan distances to get the desired behavior.

This heuristic achieved a 57.9% total win rate in the tournament. The performance of this heuristic against each other agent is listed in the breakout table below:

Opponent	Won	Lost	Win Rate
Random	20	0	100%
MiniMax_Open	8	12	40%
MiniMax_Center	16	4	80%
MiniMax_Improved	5	15	25%
Alpha-Beta_Open	7	13	35%
Alpha-Beta_Center	17	3	85%
Alpha-Beta_Improved	8	12	40%
Total	**81**	**59**	**57.9%**

Fig 5: Performance of Custom Heuristic 3: 'Open-Center' in the agent tournament

```
# Open-centered heuristic - by avoiding the corners of the board,
# we keep our options open & hopefully avoid being trapped.
# Minimize the Squared Euclidean distance between the player
# & the center of the board, while still preferring moves that keep
# open the most possible child moves

# Check win/loss conditions
if game.is_loser(player):
    return float("-inf")

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

# Find the center of the board
center_width, center_height = game.width/2.0, game.height/2.0

# Find the player's position
player_height, player_width = game.get_player_location(player)

# Get the number of open moves
moves = len(game.get_legal_moves(player))

# Return the # of number of moves minus the Square Euclidean Distance to
# the center of the board, as we want to prefer positions closer to center
# but also prefer ones that leave open more moves
return float(1.5*moves - ((abs(center_height - player_height)) + abs((center_width - player_width))))
```

Fig 6: Implementation of the Open-Center heuristic

Heuristic Recommendation:

While the 'Hybrid Score' had the highest win rate of my three heuristics at 65.0%, it performed notably worse than its average when playing against other alpha-beta players (Fig. 7). The Hybrid Score heuristic achieved a win rate of only 53.3% against other alpha-beta players, compared to a win rate of 66.7% against Minimax players.

Compare this to the performance of the 'Enemies Closer, Center Closer (ECCC)' heuristic. While it had a lower total win rate of 63.6%, it maintained a win rate of 65.0% against other alpha-beta players while winning 53.3% of the time against Minimax players.

Given that Alpha-Beta players are guaranteed to achieve performance greater than or equal to MiniMax agents, given the same board state and time restrictions, we can say that Alpha-Beta players play more optimally in situations with limited time. As game players, we should assume that our opponents play as optimally as possible. In such situations, the ECCC score performs better.

The 'Enemies Closer, Center Closer' heuristic is also very simple to calculate, requiring the calculation of only 2 board positions using basic arithmetic. This allows the agent to look at more states per turn, improving overall performance. More complex heuristics, such as one that evaluates whether partitions exist and if a move would place you inside one, require significantly more computation. In those cases, the heuristic may provide more sophisticated evaluation of the 'goodness' of a move: however, limiting our search depth limits the visibility of our agent into moves that may evaluate well, but lead to traps or terminal states just a few levels beyond our search depth. The 'ECCC' heuristic provides decent evaluation of the goodness of a move while still enabling agents to search deeply within the game tree.

As additional benefit of using the ECCC Score is that that because it does not examine the number of moves as an input, its calculation is not affected by the phase of the game (beginning, mid, end). Other scores, such as the AB-Improved or Hybrid Score heuristics, suffer from the fact that at the end of the game most moves will only have 1-2 child moves. This leads to many possible moves being evaluated with similar scores, making the critical move choices at end game more challenging or randomly selected with the left-first rule. The ECCC heuristic does not suffer from this problem.

Lastly, with heuristics that consider move-number, the agent may select a move that has the most child moves, but allows the opponent to trap it in a partition or corner. This could cause a losing terminal state to occur when a move with less child states (but that avoids traps) would succeed. ECCC's preference for trap avoidance over move count helps mitigate this risk,

Based on the listed factors, and because the 'Enemies Close, Center Closer (ECCC)' heuristic performs better against stronger opponents, I recommend the use of the ECCC heuristic.

Heuristic	Opponent	Won	Lost	Win Rate
AB-Improved	*Random*	17	3	85%
AB-Improved	*MiniMax_Open*	6	14	30%
AB-Improved	*MiniMax_Center*	15	5	75%
AB-Improved	*MiniMax_Improved*	10	10	50%
AB-Improved	*Alpha-Beta_Open*	9	11	45%
AB-Improved	*Alpha-Beta_Center*	11	9	55%
AB-Improved	*Alpha-Beta_Improved*	9	11	45%
AB-Improved	***Total***	**77**	**63**	**55.0%**
Custom1: HS	*Random*	19	1	95%
Custom1: HS	*MiniMax_Open*	7	13	35%
Custom1: HS	*MiniMax_Center*	20	0	100%
Custom1: HS	*MiniMax_Improved*	13	7	65%
Custom1: HS	*Alpha-Beta_Open*	10	10	50%
Custom1: HS	*Alpha-Beta_Center*	15	5	75%
Custom1: HS	*Alpha-Beta_Improved*	7	13	35%
Custom 1: HS	***Total***	**91**	**49**	**65.0%**
Custom2: ECCC	*Random*	18	2	90%
Custom2: ECCC	*MiniMax_Open*	7	13	35%
Custom2: ECCC	*MiniMax_Center*	17	3	85%
Custom2: ECCC	*MiniMax_Improved*	8	12	40%
Custom2: ECCC	*Alpha-Beta_Open*	12	8	60%
Custom2: ECCC	*Alpha-Beta_Center*	17	3	85%
Custom2: ECCC	*Alpha-Beta_Improved*	10	10	50%
Custom 2: ECCC	***Total***	**89**	**51**	**63.6%**
Custom3: O-C	*Random*	20	0	100%
Custom3: O-C	*MiniMax_Open*	8	12	40%
Custom3: O-C	*MiniMax_Center*	16	4	80%
Custom3: O-C	*MiniMax_Improved*	5	15	25%
Custom3: O-C	*Alpha-Beta_Open*	7	13	35%
Custom3: O-C	*Alpha-Beta_Center*	17	3	85%
Custom3: O-C	*Alpha-Beta_Improved*	8	12	40%
Custom 3: O-C	***Total***	**81**	**59**	**57.9%**

Fig 7: Full tournament history for the baseline AB-Improved agent and my three heuristics