Build a Game Playing Agent - Knights Isolation

James McGuigan

Unit Tests

```
$ python3 -m unittest -v

test_get_action_midgame (tests.test_my_custom_player.CustomPlayerGetActionTe
st)
get_action() calls self.queue.put() before timeout in a game in progress ...
ok
test_get_action_player1 (tests.test_my_custom_player.CustomPlayerGetActionTe
st)
get_action() calls self.queue.put() before timeout on an empty board ... ok
test_get_action_player2 (tests.test_my_custom_player.CustomPlayerGetActionTe
st)
get_action() calls self.queue.put() before timeout as player 2 ... ok
test_get_action_terminal (tests.test_my_custom_player.CustomPlayerGetActionT
est)
get_action() calls self.queue.put() before timeout when the game is over ...
ok
test_custom_player (tests.test_my_custom_player.CustomPlayerPlayTest)
CustomPlayer successfully completes a game against itself ... ok

----------------------------------------------------------------------
Ran 5 tests in 22.290s

OK
```

Advanced Heuristic

Code:

```
class CustomPlayer(BasePlayer):
    heuristic_fn         = 'heuristic_area'  # or 'heuristic_liberties'
    heuristic_area_depth = 4
    heuristic_area_max   = len(Action) * 3

    @staticmethod
    @lru_cache(None, typed=True)
    def liberties( state, cell ):
        """add a @lru_cache around this function"""
        return state.liberties(cell)

    @classmethod
    @lru_cache(None, typed=True)
    def heuristic_area( cls, state, player_id):
        own_loc = state.locs[player_id]
        opp_loc = state.locs[1 - player_id]
        own_area = cls.count_area_liberties(state, own_loc)
        opp_area = cls.count_area_liberties(state, opp_loc)
        return own_area - opp_area

    @classmethod
    @lru_cache(None, typed=True)  # depth > 1 exceeds 150ms timeout (without
caching)
    def count_area_liberties( cls, state, start_loc ):
        depth     = cls.heuristic_area_depth
        max_area  = cls.heuristic_area_max

        area      = set()
        frontier  = { start_loc }
        seen      = set()
        while len(frontier) and len(area) < max_area and depth > 0:
            seen     |= frontier
            frontier |= set(chain(*[ cls.liberties(state, cell) for cell in
 frontier ]))
            area     |= frontier
            frontier -= seen
            depth    -= 1
        return len(area)
```

> What features of the game does your heuristic incorporate, and why do you think those features matter in evaluating states during search?

The main analogy is with the game of Go. The goal is to surround your opponent and capture a larger territory.

Recursively computing liberties several moves ahead shows the area of the board that the opponent could potentially escape to.

At `depth=1` this `heuristic_area()` is equivalent to `#my_moves - #opponent_moves`

Early in the game the opponent effectively has access to the entire board, making this heuristic ineffective, hence `max_area` is used in addition to depth to shortcircuit the computational cost of expanding breadth-first search to all possible future moves on a mostly empty board when neither side is trapped.

This heuristic is endgame focused. It solves the simplified subproblem of local search without an adversary, and provides an upper-bound estimate for the difference in maximum number of total moves each player has remaining. It reaches maximum value for moves that trap a player within a self-contained section whilst leaving the other a means of escape. The player in the smaller territory will run out of moves first.

The other major impact on the performance of this agent is the addition of alphabeta pruning with the aggressive use of `@classmethod @lru_cache()`. This avoids the computation expense of recomputing previously explored subtrees, and by caching on `@classmethod` parts of the cache can even be reused between runs. The net effect of this is to increase the maximum depth of iterative deepening before the timeout to beyond what `MinimaxPlayer(depth=3)` can compute.

Comparison with Baseline

As we can see:

In [1]:

```
%load_ext autoreload
%autoreload 2

from collections import deque
from isolation import Agent, DebugState, Isolation, play
from my_custom_player import CustomPlayer
from run_match import play_matches
from sample_players import MinimaxPlayer, RandomPlayer
import numpy as np

class LibertiesPlayer(CustomPlayer):
    heuristic_fn = 'heuristic_liberties'

class HeuristicAreaPlayer(CustomPlayer):
    heuristic_fn = 'heuristic_area'

def run_match(agent1, agent2):
    class Args:
        fair_matches = True
        rounds       = 25
        time_limit   = 150
        processes    = 8
        debug        = False

    agent1 = Agent(agent1, agent1.__name__)
    agent2 = Agent(agent2, agent2.__name__)
    wins, num_games = play_matches(agent1, agent2, Args)

    print("{} won {:.1f}% of matches against {}".format(
       agent1.name, 100. * wins / num_games, agent2.name))
```

LibertiesPlayer vs MinimaxPlayer

- same heuristic
- this shows the added performance effect just from using alphabeta pruning, iterative deepening and lru caching
- 64% winrate

In [2]:

```
run_match(LibertiesPlayer, MinimaxPlayer)
```

```
Running 50 games:
++-++++++++++-+--+++++++++--+++++++++-++++++-++-+++-+-++-++-
Running 50 games:
++++++++++++---++++-+++++++-+-+++++-++++++++++++---+++++--+++++++
LibertiesPlayer won 79.0% of matches against MinimaxPlayer
```

HeuristicPlayer vs MinimaxPlayer

- comparing heuristic_area against reference implementation: `#my_moves - #opponent_moves` and `depth=3`
- 88% winrate showing a significant improvement over LibertiesPlayer

In [3]:

```
run_match(HeuristicAreaPlayer, MinimaxPlayer)
```

```
Running 50 games:
-++++++++++-++++++++++++++++++++++++-+--+++++++--+++-++
Running 50 games:
++++++++++-++---++++++++++++++++++-++++++++++++++++-+++++-
HeuristicAreaPlayer won 88.0% of matches against MinimaxPlayer
```

HeuristicPlayer vs LibertiesPlayer

- comparing heuristic_area against `#my_moves - #opponent_moves` but with equal iterative deepening and caching
- hyperparameter tuning found that: `heuristic_area_depth=4 + heuristic_area_max=8*5` provide the best performance
- 76% winrate, shows that the heuristic contributes an equal amount to the performance optimizations alone

In [4]:

```
run_match(HeuristicAreaPlayer, LibertiesPlayer)
```

```
Running 50 games:
+++-+-+++---+++++-++-++++++++++-++-+-++++-+-++++++++++-+++++--++
Running 50 games:
-+--+-++++++-++++-++++++++++---++++++++++-++++++++++++--+
HeuristicAreaPlayer won 76.0% of matches against LibertiesPlayer
```

Depth Analysis

> Analyze the search depth your agent achieves using your custom heuristic. Does search speed matter more or less than accuracy to the performance of your heuristic?

Compared to MinimaxPlayer:

- On the first turn, iterative deepening has trouble with the analysing beyond `depth=1`, so essentially makes a random move.
- In the early game, during the first dozen turns, iterative deepening can manage `depth=3-4` which is competitive with MinimaxPlayer.
- Towards the midgame, the depth slowly grows upto 8.
- During the last few moves of the endgame, the depth can grow unbounded, essentially to the victory condition.

In [9]:

```
class LibertiesVPlayer(CustomPlayer):
    heuristic_fn  = 'heuristic_liberties'
    verbose_depth = True
class HeuristicVPlayer(CustomPlayer):
    heuristic_fn = 'heuristic_area'
    verbose_depth = True

agents = (
    Agent(HeuristicVPlayer,  "HeuristicVPlayer"),
    Agent(MinimaxPlayer,     "MinimaxPlayer"),
)
initial_state = Isolation()
winner, game_history, _ = play((agents, initial_state, 150, 0))
print()
print('-'*50)
print('winner: ', winner.name)
# print('game_history: ', game_history)
```

```
HeuristicVPlayer | depth:
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3 4
HeuristicVPlayer | depth: 1 2 3
HeuristicVPlayer | depth: 1 2 3 4
HeuristicVPlayer | depth: 1 2 3 4
HeuristicVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3 4
HeuristicVPlayer | depth: 1 2 3 4 5 6
HeuristicVPlayer | depth: 1 2 3 4 5 6
HeuristicVPlayer | depth: 1 2 3 4 5 6 7
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 4
2 43 44 45 46 47 48 49
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 4
2 43 44 45 46 47 48 49
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 4
2 43 44 45 46 47 48 49
--------------------------------------------------
winner:  HeuristicVPlayer
```

```
Comparing HeuristicPlayer with LibertiesPlayer
- LibertiesPlayer tends to do iterative deepening with a 1-3 depth lead over
HeuristicPlayer
- LibertiesPlayer's `#my_moves - #opponent_moves` heuristic is much cheaper to
implement
- HeuristicPlayer heuristic is effectively adding an extra 4 layers of hidden
depth, but on a simplified subproblem
- This accounts for HeuristicPlayer having a 76% winrate vs LibertiesPlayer
- Performance optimization is still of utmost importance with a 150ms timeout
```

In [11]:

```
agents = (
    Agent(HeuristicVPlayer,  "HeuristicVPlayer"),
    Agent(LibertiesVPlayer,  "LibertiesVPlayer"),
)
initial_state = Isolation()
winner, game_history, _ = play((agents, initial_state, 150, 0))
print()
print('-'*50)
print('winner: ', winner.name)
# print('game_history: ', game_history)
```

```
HeuristicVPlayer | depth:
LibertiesVPlayer | depth: 1 2
HeuristicVPlayer | depth: 1 2
LibertiesVPlayer | depth: 1 2 3 4
HeuristicVPlayer | depth: 1 2
LibertiesVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3
LibertiesVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5 6 7
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3
LibertiesVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3
LibertiesVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3
LibertiesVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5 6
HeuristicVPlayer | depth: 1 2 3
LibertiesVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5 6 7
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5 6
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5 6
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5 6
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5 6
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5 6
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5
HeuristicVPlayer | depth: 1 2 3
LibertiesVPlayer | depth: 1 2 3 4 5 6
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5 6
HeuristicVPlayer | depth: 1 2 3 4
LibertiesVPlayer | depth: 1 2 3 4 5 6 7
HeuristicVPlayer | depth: 1 2 3 4 5
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9
HeuristicVPlayer | depth: 1 2 3 4 5 6 7
```

```
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22 23 24
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22 23 24 25 26 27 28 29 30 31
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 4
1 42 43 44 45 46 47 48 49
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 4
1 42 43 44 45 46 47 48 49
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 4
1 42 43 44 45 46 47 48 49
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 4
1 42 43 44 45 46 47 48 49
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 4
1 42 43 44 45 46 47 48 49
HeuristicVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 4
1 42 43 44 45 46 47 48 49
LibertiesVPlayer | depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
8 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 4
1 42 43 44 45 46 47 48 49
--------------------------------------------------
winner:  HeuristicVPlayer
```

In [ ]: