APPENDIX:

Example of the despondent machine issue (computer failing to block human from winning) demonstrated with code and debugger output when considering code at commit beginning 4c573 (Christie, 2022).

1. Failing test

```
def test_get_white_response_blocks_between_stones(self):
    # GIVEN
    winning_score = 3
    depth = 6

    board_state = [
        ["●", "o", "+", "+", "+"],
        ["+", "+", "+", "+", "+"],
        ["●", "+", "+", "+", "+"],
        ["+", "+", "+", "+", "+"],
        ["+", "+", "+", "+", "+"],
    ]

    # WHEN
    x, y = get_white_response(board_state,
winning_score=winning_score, depth=depth)
    board_state[x][y] = "o"
    print("\n\n\n***TEST BOARD STATE***")
    [print(f"{row}") for row in board_state]

    # THEN
    actual = (x, y)
    expected = (1, 0)
    self.assertEqual(expected, actual)
```

2. Console output when test fails

```
robogo $ djanrun test
games.tests.test_view.HelpersTestCase.test_get_white_response_blocks
_between_stones
Creating robogo_web_run ... done
Creating test database for alias 'default'...
System check identified no issues (0 silenced).



***TEST BOARD STATE***
['●', 'o', 'o', '+', '+']
['+', '+', '+', '+', '+']
['●', '+', '+', '+', '+']
['+', '+', '+', '+', '+']
['+', '+', '+', '+', '+']
F
======================================================================
==
FAIL: test_get_white_response_blocks_between_stones
(games.tests.test_view.HelpersTestCase)
--------------------------------------------------------------------
--
Traceback (most recent call last):
  File "/code/games/tests/test_view.py", line 129, in
test_get_white_response_blocks_between_stones
    self.assertEqual(expected, actual)
AssertionError: Tuples differ: (1, 0) != (0, 2)

First differing element 0:
1
0

- (1, 0)
+ (0, 2)

--------------------------------------------------------------------
--
Ran 1 test in 10.175s

FAILED (failures=1)
Destroying test database for alias 'default'...
ERROR: 1
```

3. `import pdb; pdb.set_trace()` added to code under test

```
def get_white_response(board_state, winning_score=WINNING_SCORE,
depth=DEPTH):
    root_node = GoNode(
        node_id="root_node",
        score=None,
        children=[],
        board_state=board_state,
        player_to_move="minimizer",
    )
    game_tree = GoTree(root_node)
    try:
        # using build_and_prune
        open_moves = sum(x == "+" for x in
list(itertools.chain(*board_state)))
        if open_moves < depth:
            depth = open_moves
        game_tree.build_and_prune_game_tree_recursive(
            parent=game_tree.root_node,
            depth=depth,
            node_ids=set(),
            winning_score=winning_score,
        )

        logger.info(f"root node: {game_tree.root_node.__str__()}")
        for child in game_tree.root_node.get_children():
            if child.get_move_coordinates() == (1, 0):
                for next_child in child.get_children():
                    logger.info(next_child.__str__())
        try:
            import pdb; pdb.set_trace()
            white_move_node = game_tree.root_node.get_optimal_move()
        except Exception as e:
            logger.error(f"Couldn't get optimal move {e}")
        print_node = white_move_node
        try:
            for i in range(depth):
                logger.info(f"Move {i}")
                for row in transpose_board(print_node.board_state):
                    # for row in print_node.board_state:
                    logger.info(row)
                if not print_node.is_leaf_node():
                    print_node = print_node.get_optimal_move()
                else:
                    break
        except Exception as e:
            logger.error(f"Error printing board: {e}")

    except Exception as e:
        logger.error(f"get_white_response failed with error: {e}")
        return

    logger.info(f"white_move_node: {white_move_node.__str__()}")

    assert (
        type(white_move_node) == GoNode
    ), f"White move node isn't of type GoNode for node:
{white_move_node.get_node_id()}"
    white_move = white_move_node.move_coordinates
    logger.info(f"white_move: {white_move}, best_score:
{white_move_node.get_score()}")
    return white_move
```

4. console work with debugger showing the game path from the root node when trying to block a black win

```
(Pdb) [print(row) for row in game_tree.root_node.get_board_state()]
['●', 'o', '+', '+', '+']
['+', '+', '+', '+', '+']
['●', '+', '+', '+', '+']
['+', '+', '+', '+', '+']
['+', '+', '+', '+', '+']

(Pdb) [print(row) for row in
game_tree.root_node.get_children()[3].get_board_state()]
['●', 'o', '+', '+', '+']
['o', '+', '+', '+', '+']
['●', '+', '+', '+', '+']
['+', '+', '+', '+', '+']
['+', '+', '+', '+', '+']
```

```
(Pdb) [print(row) for row in
game_tree.root_node.get_children()[3].get_children()[7].get_board_st
ate()]
['●', 'o', '+', '+', '+']
['o', '+', '+', '+', '+']
['●', '●', '+', '+', '+']
['+', '+', '+', '+', '+']
['+', '+', '+', '+', '+']

(Pdb) [print(row) for row in
game_tree.root_node.get_children()[3].get_children()[7].get_children
()[7].get_board_state()]
['●', 'o', '+', '+', '+']
['o', '+', '+', '+', '+']
['●', '●', 'o', '+', '+']
['+', '+', '+', '+', '+']
['+', '+', '+', '+', '+']

(Pdb) [print(row) for row in
game_tree.root_node.get_children()[3].get_children()[7].get_children
()[7].get_children()[10].get_board_state()]
['●', 'o', '+', '+', '+']
['o', '+', '+', '+', '+']
['●', '●', 'o', '+', '+']
['+', '●', '+', '+', '+']
['+', '+', '+', '+', '+']
```

Note: at the stage shown above, white can't block black

```
(Pdb) x =
game_tree.root_node.get_children()[3].get_children()[7].get_children
()[7].get_children()[10].get_children()

(Pdb) [print(row) for row in x[3].get_board_state()]
['●', 'o', '+', '+', '+']
['o', 'o', '+', '+', '+']
['●', '●', 'o', '+', '+']
['+', '●', '+', '+', '+']
['+', '+', '+', '+', '+']

(Pdb) [print(row) for row in
x[3].get_children()[13].get_board_state()]
['●', 'o', '+', '+', '+']
['o', 'o', '+', '+', '+']
['●', '●', 'o', '+', '+']
['+', '●', '+', '+', '+']
['+', '●', '+', '+', '+']

(Pdb) [print(row) for row in
x[3].get_children()[14].get_board_state()]
*** IndexError: list index out of range
```

Note: we see that alpha-beta pruning worked as no more children nodes were generated beyond a win state (as demonstrated by the list index out of range message when trying to look at subsequent node).

```
(Pdb) [child.get_score() for child in
game_tree.root_node.get_children()]
[100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100]
```

Note: all children of root node have a score of 100 which means all paths lead to a win for the human, the computer therefore simply selects the first losing move

REFERENCES:

Christie (2022). 'robogo, commit: 4c5731ae4320c0327befac50bae0e5c8a3a5f345 '. Github [online] Available at: https://github.com/nchristie/robogo/commit/4c5731ae4320c0327befac50bae0e5c8a3a5f345#diff-327ffd98a1f7c7084535515e3190e95b48a5b718f63232916279e113f507af2aR90-R119 [Accessed 23 August 2022]