**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 = [

["●", "○", "+", "+", "+"],

["+", "+", "+", "+", "+"],

["●", "+", "+", "+", "+"],

["+", "+", "+", "+", "+"],

["+", "+", "+", "+", "+"],

]

# WHEN

x, y = get\_white\_response(board\_state, winning\_score=winning\_score, depth=depth)

board\_state[x][y] = "○"

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\*\*\*

['●', '○', '○', '+', '+']

['+', '+', '+', '+', '+']

['●', '+', '+', '+', '+']

['+', '+', '+', '+', '+']

['+', '+', '+', '+', '+']

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()]

['●', '○', '+', '+', '+']

['+', '+', '+', '+', '+']

['●', '+', '+', '+', '+']

['+', '+', '+', '+', '+']

['+', '+', '+', '+', '+']

(Pdb) [print(row) for row in game\_tree.root\_node.get\_children()[3].get\_board\_state()]

['●', '○', '+', '+', '+']

['○', '+', '+', '+', '+']

['●', '+', '+', '+', '+']

['+', '+', '+', '+', '+']

['+', '+', '+', '+', '+']

(Pdb) [print(row) for row in game\_tree.root\_node.get\_children()[3].get\_children()[7].get\_board\_state()]

['●', '○', '+', '+', '+']

['○', '+', '+', '+', '+']

['●', '●', '+', '+', '+']

['+', '+', '+', '+', '+']

['+', '+', '+', '+', '+']

(Pdb) [print(row) for row in game\_tree.root\_node.get\_children()[3].get\_children()[7].get\_children()[7].get\_board\_state()]

['●', '○', '+', '+', '+']

['○', '+', '+', '+', '+']

['●', '●', '○', '+', '+']

['+', '+', '+', '+', '+']

['+', '+', '+', '+', '+']

(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()]

['●', '○', '+', '+', '+']

['○', '+', '+', '+', '+']

['●', '●', '○', '+', '+']

['+', '●', '+', '+', '+']

['+', '+', '+', '+', '+']

**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()]

['●', '○', '+', '+', '+']

['○', '○', '+', '+', '+']

['●', '●', '○', '+', '+']

['+', '●', '+', '+', '+']

['+', '+', '+', '+', '+']

(Pdb) [print(row) for row in x[3].get\_children()[13].get\_board\_state()]

['●', '○', '+', '+', '+']

['○', '○', '+', '+', '+']

['●', '●', '○', '+', '+']

['+', '●', '+', '+', '+']

['+', '●', '+', '+', '+']

(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]