CSC148 winter 2015

A2, assert, call stack

week 10

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/148/W14/
416-978-5899

March 18, 2015

Outline

A2 examples

testing

making assertions

function call stack

one minimax solution

- starting game state has some moves
- generate all the game states reachable in one move
- find score using outcome or minimax (what is the base case?)
- find state with lowest score (opponent's point of view)
- return $-1\times$ lowest score and the move that achieved it

tippy apply_move

The type contract says to return a new GameState. Another approach would be to update the GameState, but reverse every move explored in minimax.

Careful copying the board!

analyze test parameters

Minimax on subtract square instance instance has two parameters to vary: the next player and the current total:

```
sm = StrategyMinimax()
sm.suggest_move(SubtractSquareState('p1', current_total=8))
```

What test cases should we check?

choosing test cases

What information can you get from the following tests, whether they pass or fail:

```
sm = StrategyMinimax()
sm.suggest_move(SubtractSquareState('p1', current_total=9))
sm.suggest_move(SubtractSquareState('p1', current_total=10))
sm.suggest_move(SubtractSquareState('p1', current_total=11))
```

Compare the possible results to what a random suggest_move might provide.

increasing confidence

How many passed tests should it take to make you confident that the code works? How many failed tests?

setUp, tearDown

For a sequence of tests we need to create some objects, and destroy them multiple times. For example:

```
StrategyMinimax()
SubtractSquareState()
SubtractSquareMove()
... etc.
```

These are **fixtures** of a **TestCase**, and we use methods **setUp** and **tearDown** to reduce repeated code.

state assumption

What do we assume is true about prev_node and cur_node at the end of the while loop?

```
if lnk.size > 0:
    prev_node = None
    cur_node = lnk.front
    while cur_node and not cur_node.value == v2:
        prev_node = cur_node
        cur_node = cur_node.nxt
```

Make the assumptions into assertions.

```
assert <condition> , <error_string>
---> if not condition: raise AssertionError(error_string)
```

type contracts

To a certain extent you can enforce type contracts using **assert**. What assertion would make sense at the beginning of this function?

```
def list_between(node, start, end):
    ''' (BTNode, object, object) -> list

    Return a Python list of all values in the binary search tree
    rooted at node that are between start and end (inclusive).

    ...

    '''
```

tracing calls

Trace the print statement in function.py. Here are the rules:

- evaluate the arguments to each function, left to right
- function calls whose bodies cannot be completely evaluated go on the stack, which may add function calls to stack
- names are resolved "inside out" — look for the most local assignment (e.g. function definition or parameters), then to enclosing assignment.

example

```
N = 5
M = 2

def f(N):
    return M * g(N + 1)

def g(N):
    return M * N

def h(N):
    return f(g(N))


if __name__ == '__main__':
    print(h(1))
```

assignment 3

assignment 3 is a3.pdf.

A recurring theme is redundancy...