COMP6481/COMP8481
Class 2: NIM Game in Python

Marek Grześ

January 4, 2023

All the resources for this class are on `raptor` in `/courses/comp6481/class2_nim_part1/`. Very simple Python concepts are used in this class, and there is no `numpy`. The learning objective is to devise a heuristic solution to a simple game and to implement it in Python, extending the existing framework.

1 Introduction

Nim is a mathematical game of strategy in which two players take turns removing objects from distinct heaps or piles. We will consider a special case of the game which is commonly played in practice with only one pile. This special case is better called the subtraction game. An upper bound is imposed on the number of objects that can be removed in a turn. A player can only remove 1 or 2 or ... or k at a time. Depending on the version being played, the goal of the game is either to avoid taking the last object or to take the last object.

In our class, we will make the following assumptions:

1. there is 1 pile of objects/stones,
2. a player can take 1 or 2 objects/stones,
3. the goal of the game is to take the last object or objects.

The Nim game with these assumptions can be found here `https://education.jlab.org/nim/`, and you are encouraged to play it a few times before engaging in the class exercises.

2 Nim Game Implementation

Listing 1 presents a terminal-based implementation of the Nim game, which allows a human player to play against a random opponent. This implementation is available on `raptor` in `/courses/comp6481/class2_nim_part1/nim1pile_no_learning_stud.py`.

Listing 1: Nim Game Implementation

```
'''
    Author: Marek Grzes, University of Kent
    Date created: 23/12/2022
    Date last modified:
    Python Version: 3.10
'''
```

```
7
8  import sys
9  import getopt
10 import random
11
12 # config variables
13 pile_size = 10
14 whose_turn = "computer" # computer or human
15 opponent = "random"     # random or expert or learning
16
17
18 def expert_move(n):
19     '''
20     :param n: state
21     :return: action (not action id)
22     '''
23     # TODO: implement your expert move here
24     return random.randint(1,2)
25
26
27 def who_is_first():
28     if random.choice([True, False]) is True:
29         return "computer"
30     else:
31         return "human"
32
33
34 if __name__ == "__main__":
35     opts, args = getopt.getopt(sys.argv[1:], "hg:o:", ["help", "goes-first=", "opponent=
       "])
36     for opt, arg in opts:
37         if opt == '-h':
38             print(sys.argv[0] + ' --goes-first [human|computer] --opponent [random|
                   expert]')
39             sys.exit(0)
40         elif opt in ("-g", "--goes-first"):
41             whose_turn = arg
42         elif opt in ("-o", "--opponent"):
43             opponent = arg
44
45     episode = 0
46     while True:
47         if episode > 0:
48             # after one episode, we randomise who goes first
49             whose_turn = who_is_first()
50         print("New game no {} starts. {} plays first. The opponent is {}.".format(
               episode+1, whose_turn, opponent))
51         curr_pile_size = pile_size
52         while True:
53             print("-" * 20)
54             print("Current pile has " + str(curr_pile_size) + " stones: " + "*" *
                   curr_pile_size)
55             if whose_turn == "human":
56                 move = -1
57                 if curr_pile_size == 1:
58                     print("Only action 1 is possible.")
59                     move = 1
60                     input("Please press ENTER:\n")
61                 else:
```

```
                    while not move == 1 and not move == 2:
                        move = int(input("Please enter 1 or 2 and press ENTER:\n"))
                print("The {} player moves next and removes {}.".format(whose_turn, move
                    ))
                curr_pile_size -= move
                if curr_pile_size == 0:
                    print("You won!")
                    break
            else:
                move = -1
                if opponent == "random":
                    move = random.randint(1, 2)
                    if curr_pile_size == 1:
                        move = 1
                elif opponent == "expert":
                    move = expert_move(curr_pile_size)
                    if curr_pile_size == 1:
                        move = 1
                else:
                    print("Wrong opponent. See the --opponent parameter.")
                    sys.exit(1)
                if (curr_pile_size == 1 and move != 1) or (move != 1 and move != 2):
                    print("Wrong action.")
                    sys.exit(1)
                print("The {} player moves next and removes {}.".format(whose_turn, move
                    ))
                curr_pile_size -= move
                if curr_pile_size == 0:
                    print("The {} player won!".format(opponent))
                    break
            if whose_turn == "human":
                whose_turn = "computer"
            else:
                whose_turn = "human"
        again = input("Play again y/n and press ENTER:\n")
        if again == "n":
            break
        episode += 1
```

You are encouraged to study this code to understand all the major details because you will be asked to extend it later on. It may be helpful to run it a few times and play a few games to see what kind of output is printed. The output in Listing 2 below shows how to run this program on `raptor`.

Listing 2: Program execution with default parameters

```
mg483@raptor [/courses/comp6481/class2_nim_part1] $ python3 nim1pile_no_learning_stud.py
New game no 1 starts. computer plays first. The opponent is random.
--------------------
Current pile has 10 stones: **********
The computer player moves next and removes 2.
--------------------
Current pile has 8 stones: ********
Please enter 1 or 2 and press ENTER:
1
The human player moves next and removes 1.
--------------------
Current pile has 7 stones: *******
The computer player moves next and removes 1.
--------------------
```

```
Current pile has 6 stones: ******
Please enter 1 or 2 and press ENTER:
```

3 Expert Strategy

When our initial code is run without any command line parameters (like in Listing 2), the human player will play against the random player. In this section, you are asked to implement your expert strategy that will play better than the random player. We understand that you may not be able to find a fully optimal solution that will work for all states of the game. However, you should be able to create a few `if-then` rules that will allow your computer agent to be somewhat better than the random player. The `expert_move(n)` function in Listing 1 is the only place where your modifications to the code are needed. This function takes the current state of the game as input (i.e. `n` is the current number of stones on the pile), and it should return 1 or 2, which represent the number of stones to be removed from the pile.

Your task in this section can be summarised as follows:

1. Implement your strategy in `expert_move(n)` in `nim1pile_no_learning_stud.py`.
2. Test your strategy using the `--opponent expert` command line parameter. This step is shown in Listing 3 below. Note the difference between the first two lines in Listings 2 and 3.

Listing 3: Program execution with an expert opponent

```
mg483@raptor [/courses/comp6481/class2_nim_part1] $ python3 nim1pile_no_learning_stud.py
    --opponent expert
New game no 1 starts. computer plays first. The opponent is expert.
--------------------
Current pile has 10 stones: **********
```

If you have finished all the sections of this document early, you can start working on the next worksheet.

Note that our class next week builds on this class. Make sure that you complete this worksheet before the next class.