Nim, nim.py and games.py

Homework 4
Problem 4

The History of Nim Games

• Believed to have been created in China; unknown date of origin
• First actual recorded date- 15th century Europe
• Originally known as Tsyanshidzi meaning “picking stones game”
• Presently comes from German word “nimm” meaning “take”

Adapted from a presentation by Tim Larson and Danny Livarchik

Rules of Nim

• Impartial game of mathematical strategy
• Strictly two players
• Alternate turns removing any number of items from any ONE heap until no pieces remain
• Must remove at least one item per turn
• Last player to be able to remove a wins
• Variations:
  – Initial number of heaps and items in each
  – Misere play: last player who can move loses
  – Limit on number of items that can be removed

Demonstration

Player 1 wins!
**Theoretical Approach**

- Theorem developed by Charles Bouton in 1901
- This states that in order to win, the goal is to reach a nim-sum of 0 after each turn until all turns are finished
- Nim Sum: evaluated by taking the exclusive-or of the corresponding numbers when the numbers are given in binary form
- Exclusive-or is used for adding two or more numbers in binary and it basically ignores all carries

---

**Tree for (2,1)**

---

**Tree for (2,2)**

---

**games.py**

- Peter Norvig’s python framework for multi-player, turn taking games
- Implements minimax and alphabeta
- For a new game, subclass the Game class
  - Decide how to represent the “board”
  - Decide how to represent a move
  - A state is (minimally) a board and whose turn to move
  - Write methods to (1) initialize game instance, (2) generate legal moves from a state, (3) make a move in state, (4) recognize terminal states (win, lose or draw), (5) compute utility of a state for a player, (5) display a state
Assumptions about states

- `games.py` assumes that your representation of a state is an object with at least two attributes: `to_move` and `board`.
- The `Struct` class defined in `utils.py` can be used to create such instances:
  - `s = Struct(foo='a', to_move=1, board=[[1][2][3]])`
  - Access the attributes as `s.to_move`, etc.

Caution

- Python lists are mutable objects.
- If you use a list to represent a board and want to generate a new board from it, you probably want to copy it first:
  - `new_board = board[:]`

Players

The `games.py` framework defines several players:

- `random_player`: chooses a random move from among legal moves.
- `alphabeta_player`: uses `alpha_beta` to choose best move, optional args specify cutoff depth (default is 8) and some other variations.
- `human_player`: asks user to enter move.

Variations

```python
def make_alphabeta_player(N):
    """returns a player function that uses alpha_beta search to depth N"
    return lambda game, state: alphabeta_search(state, game, d=N)
```

# add to the PLAYER dictionary player function named ab1, ab2, ... ab20
# that use alpha_beta search with depth cutoffs between 1 and 20
for i in range(20):
    PLAYER['ab'+str(i)] = make_alphabeta_player(i)
```