# MCS 275 Project Two : playing mancala
due Monday 27 February at 1PM

The goal of this project is to use a game tree for mancala. Mancala is a 2-player board game.

Two players face each other with a board in between them. The board has two rows. A row has six holes and one store. The store of the player is to the right of the six holes. At the start of the game, the stores are empty. In each hole there are four stones. The initial board can be represented as

```
0  4  4  4  4  4  4
   4  4  4  4  4  4  0
```

Players take turns, following these three rules:

1. In each turn, a player picks up all stones in one of the six holes at the side of the player. All stones are dropped one by one in successive holes, starting at the next hole, in counter clockwise direction. In dropping stones, skip the store of the other player.
2. If the last stone in the sequence lands in the store of the player, then the player gets an extra turn.
3. If the last stone dropped is in an empty hole on the side of the player, then the player can move all stones in the opposite hole to the store of the player.

The game ends when all holes are empty on one side. At the end of the game, all stones remaining in holes go to the store at the right. Who has the most stones in store at the end of the game wins.

By default, an object of the class `Board` is constructed with the initial board. We can also provide specific values for the number of stones in the holes and the stores, as illustrated below.

```
>>> from board import Board
>>> b = Board()
>>> b
   0  4  4  4  4  4  4
      4  4  4  4  4  4  0
>>> a = Board(rowone=[3, 0, 6, 1, 2, 3], storeone=3, \
... rowtwo=[1,2,3,6,0,2], storetwo=1)
>>> a
   3  3  0  6  1  2  3
      1  2  3  6  0  2  1
```

The second initialization is useful for testing.

The `main()` function of the script `board.py` launches an interactive test, which could run as follows:

```
$ python3 board.py
The initial board :
   0  4  4  4  4  4  4
      4  4  4  4  4  4  0
Enter 1, 2, 3, 4, 5, 6 : 3
Give a row, either 0 or 1 : 1
The move is valid.
Last stone in row 1 and hole 7
The player gets an extra turn.
The board after the move :
```

```
   0  4  4  4  4  4  4
      4  4  0  5  5  5  1
A test board :
   2  5  0  3  2  6  7
      0  8  2  3  0  5  3
Enter 1, 2, 3, 4, 5, 6 : 4
Give a row, either 0 or 1 : 0
The move is valid.
Last stone in row 0 and hole 2
The player gets stones from opposite row.
The board after the move :
  10  5  1  4  0  6  7
      0  0  2  3  0  5  3
$
```

The feedback messages, such as `"The move if valid."` are printed by the test function in the script `board.py`. No printing statements occur in the methods of the class `Board`.

In the test above, by selecting a row, the user identifies with player zero or one. By selecting a hole, the user can test specific moves, such as the special case when the last stone ends in the store, or the special case where a player receives stones from the opposite row.

The class `Board` defines the data structure to represent a game board with all methods to perform the move of one player. In particular, the class should define the methods to code the three rules of the game: (1) to distribute stones in counter clockwise fashion, (2) to decide whether a player gets an extra turn; and (3) to move stones from a hole of the opponent to the store. The `Board` should also provide a method to decide whether the choice of a hole by a player is valid, that is: the hole contains at least one stone. Define a method to check whether the game is over, that is: when one row of holes is empty.

The constructor of the class `Player` takes on input a board (an instance of the class `Board`) and an identification number which is either zero or one. This identification number corresponds to a row on the board. The human player in the game will always be identified as player one, while the opponent, the computer, is player zero. The third data attribute of an instance of the class `Player` is the difficulty level, which is an integer number, either $-1$ for human input, 0 for a random value, or a positive number for the depth of the game tree to compute the next best move.

Who begins the game is decided by a coin flip, via the outcome of `randind(0, 1)`. To choose the hole to pick the stones from, define three different methods:

1. Prompt the user for an integer from one to six, check if the entered number is in `range(1,7)` and if the corresponding hole is not empty. If empty, prompt the user to try again. Provide an exception handler to guard against type errors. This method defines level $-1$ of the player.

2. Choose uniformly at random with `randint(1,6)`. If the chosen hole is empty, run `randint(1,6)` again. This random choice is strategy zero. When the computer runs this strategy, a human player can easily win the game. This method defines level 0 of the player.

3. Enumerate all choices of holes and compute for each choice the difference between the number of stones in the store between the current player and the number of stones in the store of the opponent. A positive difference indicates a winning move, while a negative difference is likely to lead to a loss. The first hole in this enumeration that leads to the largest difference then defines the choice of hole to pick the stones from. This methods runs when level is larger than zero. The level of the player determines the maximum number of turns to which the enumeration will apply. This level also equals the maximum depth of the game tree.

An interactive session to test a move of a player could run as below:

```
>>> from player import Player
>>> from board import Board
>>> b = Board()
>>> p = Player(b, 1, -1)
>>> p
Player 1 plays at level -1 at board:
   0  4  4  4  4  4  4
      4  4  4  4  4  4  0
>>> p.play()
enter 1, 2, 3, 4, 5, 6 : 3
Player 1 picks up 4 stone(s).
Player 1 gets an extra turn.
   0  4  4  4  4  4  4
      4  4  0  5  5  5  1
enter 1, 2, 3, 4, 5, 6 : 4
Player 1 picks up 5 stone(s).
>>> p
Player 1 plays at level -1 at board:
   0  4  4  4  4  5  5
      4  4  0  0  6  6  2
```

The method `play()` prompts the user for a hole as the player `p` to which the method was applied to was instantiated with level `-1`. This `-1` corresponds to the interactive mode of playing. At level `0`, the player will select a hole at random. At level `1` and higher, the player will compute the next best move with a recursive game tree.

By default, `play()` without options runs in `verbose` mode. To turn off the print messages, which are useful for verify the correctness and debugging, run `play()` as `play(False)` or more explicitly as `play(verbose=False)`. Running `python3 player.py` at the command prompt asks the user three questions, as illustrated below.

```
$ python3 player.py
Welcome to our mancala game!
Run with two computer players ? (y/n) n
Give the level of difficulty (>= 0) : 2
Verbose mode ? (y/n) n
   0  4  4  4  4  4  4
      4  4  4  4  4  4  0
   1  5  5  5  0  4  4
      4  4  4  4  4  4  0
   2  0  5  5  0  4  4
      5  5  5  5  4  4  0
enter 1, 2, 3, 4, 5, 6 : 2
   2  0  5  5  0  4  4
      5  0  6  6  5  5  1
enter 1, 2, 3, 4, 5, 6 : 3
   2  0  5  5  0  5  5
      5  0  0  7  6  6  2
   3  1  6  6  1  0  5
      5  0  0  7  6  6  2
```

```
   4  0  6  6  1  0  5
      5  0  0  7  6  6  2
   9  1  7  7  2  1  0
      0  0  0  7  6  6  2
enter 1, 2, 3, 4, 5, 6 :
```

In the above session, the coin flip went in favor of player zero, who started the game. In verbose mode, what gets printed depends on the logic of your code. When verbose is `False`, the board is printed each time the player is prompted for a move. After each change of the board, the board is printed as well. The same board should not be printed twice in a row.

Running the game with two computer players, after each turn, the user is asked to continue or not.

```
$ python3 player.py
Welcome to our mancala game!
Run with two computer players ? (y/n) y
Give the level of difficulty (>= 0) : 2
Verbose mode ? (y/n) n
   0  4  4  4  4  4  4
      4  4  4  4  4  4  0
   0  4  4  4  4  4  4
      4  4  0  5  5  5  1
   0  4  4  4  4  5  5
      4  4  0  0  6  6  2
   1  5  5  5  0  5  5
      4  4  0  0  6  6  2
   2  6  6  6  1  0  5
      4  4  0  0  6  6  2
   3  0  6  6  1  0  5
      5  5  1  1  7  6  2
Continue to the next round ? (y/n)
```

At the end of the game, both in the interactive mode and with two computer players, the final board is printed and the winner is announced:

```
The final board :
  27  0  0  0  0  0  0
      0  0  0  0  0  0 21
Player 0 wins.
```

If the store on the second row is larger than the store on the first row, then player 1 wins. In some cases the game may also end in a draw:

```
The final board :
  24  0  0  0  0  0  0
      0  0  0  0  0  0 24
The game ended in a draw.
```

Some important points:

1. Since this project consists in two parts, you may (not must) collaborate in pairs.
   Both authors will receive the same number of points. A pair consists of two, not three or more.
2. The solution consists of two files: the first file is the script `board.py` and the second file is the script `player.py`. You must define the class `Board` in the script `board.py`. The script `player.py` contains the definition of the class `Player` and the main program which launches the game.
3. Submit the two files `board.py` and `player.py` in an email attachment.
4. Every function in your program must have a documentation string.
   Moreover, provide a documentation string for each class and for each module.
5. Handing in an incomplete but working program is better than handing in a program that crashes or does not run at all.
6. The first line of your Python script must be

   ```
   # MCS 275 Project Two by <Authors>
   ```

   where you replace the `<Authors>` by your names.
7. If you work in a pair, then only one author should submit.
8. Email your solution to the project to `janv@uic.edu` before 1PM on Monday 27 February so the date of the email is proof of an on time submission. Also bring a printed version of your solution to class.
9. There is an automatic extension of the deadline till 5PM on the same day. However, late submissions are penalized with ten points off. Submissions after 5PM will not be graded.

If you have questions or difficulties with the project, feel free to come to my office for help.