LAB 6 - Building Games with AI

Avec la mise à jour d’easyAI, beaucoup de code du lab6 est obsolète.  
En consultant la documentation officielle dispo sur le GitHub du projet, j’ai pu refaire les programmes.

<https://github.com/Zulko/easyAI/blob/master/README.rst>

Quelques exmples de problèmes que j’ai rencontrés :

from easyAI import id\_solve

Old

from easyAI import solve\_with\_iterative\_deepening

New

from easyAI.AI import TT

Old

from easyAI.AI import TranspositionTable

New

from easyAI.AI import TwoPlayersGame

Old

from easyAI.AI import TwoPlayerGame

New

game.play()

Bug lors de l’execution de cette fonction

AttributeError Traceback (most recent call last)  
[<ipython-input-11-dfc344d4fcfa>](https://localhost:8080/#) in <cell line: 0>()  
 **60** # Start the game  
 **61** game = LastCoinStanding([AI\_Player(tt), Human\_Player()])  
---> 62 history = game.play()

Construction d'un bot pour jouer à Last Coin Standing

from easyAI import TwoPlayerGame, Human\_Player, AI\_Player, Negamax

class GameOfBones( TwoPlayerGame ):

""" In turn, the players remove one, two or three bones from a

pile of bones. The player who removes the last bone loses. """

def \_\_init\_\_(self, players=None):

self.players = players

self.pile = 20 # start with 20 bones in the pile

self.current\_player = 1 # player 1 starts

def possible\_moves(self): return ['1','2','3']

def make\_move(self,move): self.pile -= int(move) # remove bones.

def win(self): return self.pile<=0 # opponent took the last bone ?

def is\_over(self): return self.win() # Game stops when someone wins.

def show(self): print ("%d bones left in the pile" % self.pile)

def scoring(self): return 100 if game.win() else 0 # For the AI

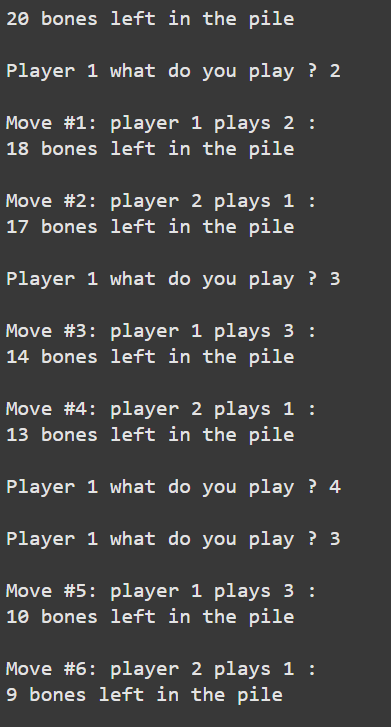
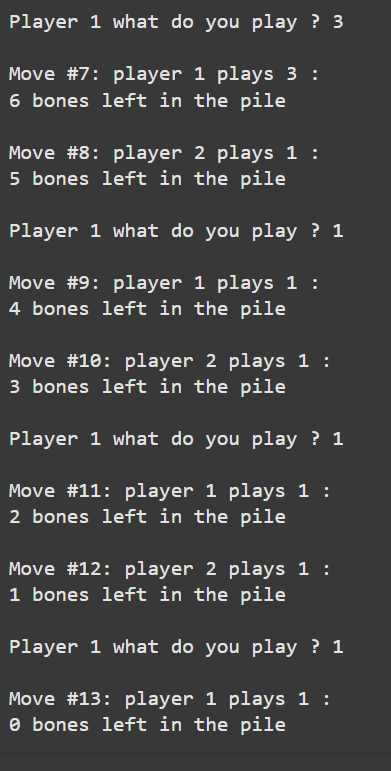
# Start a match (and store the history of moves when it ends)

ai = Negamax(13) # The AI will think 13 moves in advance

game = GameOfBones( [ Human\_Player(), AI\_Player(ai) ] )

history = game.play()

Voici le résultat (j’ai perdu la partie contre le bot).

Construction d'un bot pour jouer au Tic-Tac-Toe

from easyAI import TwoPlayerGame

from easyAI.Player import Human\_Player

class TicTacToe(TwoPlayerGame):

"""The board positions are numbered as follows:

1 2 3

4 5 6

7 8 9

"""

def \_\_init\_\_(self, players):

self.players = players

self.board = [0 for i in range(9)]

self.current\_player = 1 # player 1 starts.

def possible\_moves(self):

return [i + 1 for i, e in enumerate(self.board) if e == 0]

def make\_move(self, move):

self.board[int(move) - 1] = self.current\_player

def unmake\_move(self, move): # optional method (speeds up the AI)

self.board[int(move) - 1] = 0

def lose(self):

""" Has the opponent "three in line ?" """

return any(

[

all([(self.board[c - 1] == self.opponent\_index) for c in line])

for line in [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9], # horiz.

[1, 4, 7],

[2, 5, 8],

[3, 6, 9], # vertical

[1, 5, 9],

[3, 5, 7],

]

]

) # diagonal

def is\_over(self):

return (self.possible\_moves() == []) or self.lose()

def show(self):

print(

"\n"

+ "\n".join(

[

" ".join([[".", "O", "X"][self.board[3 \* j + i]] for i in range(3)])

for j in range(3)

]

)

)

def scoring(self):

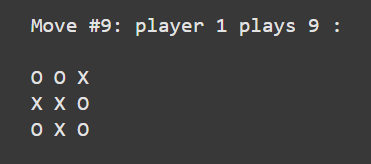
return -100 if self.lose() else 0

if \_\_name\_\_ == "\_\_main\_\_":

from easyAI import AI\_Player, Negamax

ai\_algo = Negamax(6)

TicTacToe([Human\_Player(), AI\_Player(ai\_algo)]).play()

Au bout du tour 9 (logiquement le dernier tour), nous avons fait match nul avec le bot.  
  


Construction de deux bots pour jouer au Puissance 4

try:

import numpy as np

except ImportError:

print("Sorry, this example requires Numpy installed !")

raise

from easyAI import TwoPlayerGame

class ConnectFour(TwoPlayerGame):

"""

The game of Connect Four, as described here:

<http://en.wikipedia.org/wiki/Connect_Four>

"""

def \_\_init\_\_(self, players, board=None):

self.players = players

self.board = (

board

if (board is not None)

else (np.array([[0 for i in range(7)] for j in range(6)]))

)

self.current\_player = 1 # player 1 starts.

def possible\_moves(self):

return [i for i in range(7) if (self.board[:, i].min() == 0)]

def make\_move(self, column):

line = np.argmin(self.board[:, column] != 0)

self.board[line, column] = self.current\_player

def show(self):

print(

"\n"

+ "\n".join(

["0 1 2 3 4 5 6", 13 \* "-"]

+ [

" ".join([[".", "O", "X"][self.board[5 - j][i]] for i in range(7)])

for j in range(6)

]

)

)

def lose(self):

return find\_four(self.board, self.opponent\_index)

def is\_over(self):

return (self.board.min() > 0) or self.lose()

def scoring(self):

return -100 if self.lose() else 0

def find\_four(board, current\_player):

"""

Returns True iff the player has connected 4 (or more)

This is much faster if written in C or Cython

"""

for pos, direction in POS\_DIR:

streak = 0

while (0 <= pos[0] <= 5) and (0 <= pos[1] <= 6):

if board[pos[0], pos[1]] == current\_player:

streak += 1

if streak == 4:

return True

else:

streak = 0

pos = pos + direction

return False

POS\_DIR = np.array(

[[[i, 0], [0, 1]] for i in range(6)]

+ [[[0, i], [1, 0]] for i in range(7)]

+ [[[i, 0], [1, 1]] for i in range(1, 3)]

+ [[[0, i], [1, 1]] for i in range(4)]

+ [[[i, 6], [1, -1]] for i in range(1, 3)]

+ [[[0, i], [1, -1]] for i in range(3, 7)]

)

if \_\_name\_\_ == "\_\_main\_\_":

# LET'S PLAY !

from easyAI import AI\_Player, Negamax, SSS

ai\_algo\_neg = Negamax(5)

ai\_algo\_sss = SSS(5)

game = ConnectFour([AI\_Player(ai\_algo\_neg), AI\_Player(ai\_algo\_sss)])

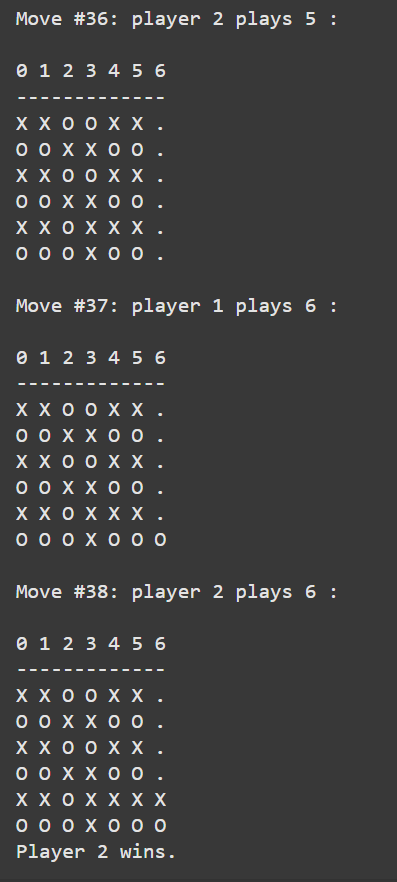
game.play()

if game.lose():

print("Player %d wins." % (game.opponent\_index))

else:

print("Looks like we have a draw.")

Voici le résultat :  


C’est effectivement le bot 2 qui gagne la partie.

Construction de deux bots pour jouer à Hexapawn

from easyAI import TwoPlayerGame

# Convert D7 to (3,6) and back...

to\_string = lambda move: " ".join(

["ABCDEFGHIJ"[move[i][0]] + str(move[i][1] + 1) for i in (0, 1)]

)

to\_tuple = lambda s: ("ABCDEFGHIJ".index(s[0]), int(s[1:]) - 1)

class Hexapawn(TwoPlayerGame):

"""

A nice game whose rules are explained here:

<http://fr.wikipedia.org/wiki/Hexapawn>

"""

def \_\_init\_\_(self, players, size=(4, 4)):

self.size = M, N = size

p = [[(i, j) for j in range(N)] for i in [0, M - 1]]

for i, d, goal, pawns in [(0, 1, M - 1, p[0]), (1, -1, 0, p[1])]:

players[i].direction = d

players[i].goal\_line = goal

players[i].pawns = pawns

self.players = players

self.current\_player = 1

def possible\_moves(self):

moves = []

opponent\_pawns = self.opponent.pawns

d = self.player.direction

for i, j in self.player.pawns:

if (i + d, j) not in opponent\_pawns:

moves.append(((i, j), (i + d, j)))

if (i + d, j + 1) in opponent\_pawns:

moves.append(((i, j), (i + d, j + 1)))

if (i + d, j - 1) in opponent\_pawns:

moves.append(((i, j), (i + d, j - 1)))

return list(map(to\_string, [(i, j) for i, j in moves]))

def make\_move(self, move):

move = list(map(to\_tuple, move.split(" ")))

ind = self.player.pawns.index(move[0])

self.player.pawns[ind] = move[1]

if move[1] in self.opponent.pawns:

self.opponent.pawns.remove(move[1])

def lose(self):

return any([i == self.opponent.goal\_line for i, j in self.opponent.pawns]) or (

self.possible\_moves() == []

)

def is\_over(self):

return self.lose()

def show(self):

f = (

lambda x: "1"

if x in self.players[0].pawns

else ("2" if x in self.players[1].pawns else ".")

)

print(

"\n".join(

[

" ".join([f((i, j)) for j in range(self.size[1])])

for i in range(self.size[0])

]

)

)

if \_\_name\_\_ == "\_\_main\_\_":

from easyAI import AI\_Player, Human\_Player, Negamax

scoring = lambda game: -100 if game.lose() else 0

ai = Negamax(10, scoring)

game = Hexapawn([AI\_Player(ai), AI\_Player(ai)])

game.play()

print("player %d wins after %d turns " % (game.opponent\_index, game.nmove))

C’est le bot numéro 1 qui gagne au bout du 12ème tour.  
