ce qu'il reste à faire : - message queue, protection des données

import sysv_ipc
from queue import Queue
import random
import timer
from multiprocessing import Process, Value, Array, Lock

key_BtoP = 128
key_PtoB = 129
mq_BtoP = sysv_ipc.MessageQueue(key_BtoP, sysv_ipc.IPC_CREAT)
mq_PtoB = sysv_ipc.MessageQueue(key_PtoB)

class Player(Process):
def init(self, Pile, ID):
self.Pile = Pile
cartes_main = []
for i in range(6):
mains.append(pioche())

```
def run_player(Player):
    start = time.time()
    while len(Player.cartes_main) != 0:
        while timer < 10000:
            if mq_PtoB.size:
                lire message(=cartemsg) que player to Board
                for carte in Player.cartes_main:
                    if cartemsg = carte:
                        Player.carte_main.remove(carte)
                    elif cartemsg = 100 + carte
                    main.append(pioche())
        cartes_mains.append(pioche[0])
        end = time.time()
        timer = end - start
        timer = 0

def carte_joue():
    if mq_PtoB.size() != 0:
        return True
    return False
```

class Board:

def init(self, numCard, numPlayers,):
self.card = numCard
processes = []
for i in range(0, numPlayers):
p = Player(Pile,)
processes.append(p)
p.start()
# il faut start les processes

```
def run(self):
    message = 0
    while(! is__finished()):
        # Message queue Board to Player
        while message:
            message_BtoP = str(value_BtoP).encode()
            mq_BtoP.send(message_BtoP)

        # Message Queue Player to Board
        while True:
            message_PtoB, t = mq_PtoB.receive()
            value_PtoB = message_PtoB.decode()
            value_PtoB = int(value_PtoB)
            if value_PtoB:  # Value_PtoB sera un tableau avec 2 cases : la première
                print("received:", value_PtoB)
                numJoueur = value_PtoB[1]
                if is_valid(self.card, value_PtoB[0]):
                    self.card = value_PtoB[0]
                    message = 1
                    value_BtoP = int(value_PtoB[0])
                else:
                    # Si mauvais on renvoie le numéro de la carte + 100
                    value_BtoP = int(100+value_PtoB[0])
                mq_PtoB.empty()

            else:
                print("exiting.")
                break
    mq_BtoP.remove()
    mq_PtoB.remove()
```

def is_finished(pile, lock):
with lock
if (len(pile) == 0):
return True
return False

def is_valid(board_card, player_card):

if ((player_card == board_card + 1)
or (player_card == board_card - 1)
or (abs(player_card) == abs(board_card))):
return True
return False

def pioche(pile, lock):
with lock:
pile_content = pile[0]
pile.pop(0)
return pile_content

if name == "main":
# Creating Message Queue Board to Player

```
# Initialisation Pile
# les numéros négatifs représenteront les bleus tandis que les numéros négatifs ser
pile = Array('i', range(-10, 10))
lock = Lock()
random.shuffle(pile)
numJoueur = int(input("Entrez le nb de joueur :"))
theBoard = Board(1, 2, lock)
```