**1η εργασία Οικονομικής Θεωρίας και Αλγορίθμων**

**Ονοματεπώνυμο:** Μαρία-Βασιλική Πετροπούλου **ΑΜ:** 1072540

**Ονοματεπώνυμο:** Ίων-Απόστολος Μπουρνάκας-Δρακόπουλος **ΑΜ:** 1075475

**Θεωρητικό Μέρος**

Παρακάτω εξετάζεται ξεχωριστά κάθε περίπτωση Ν κενών κελιών, όπου Ν ≤ 5. Χρησιμοποιείται αναπαράσταση με στοίβες, τόσο της αρχικής παρτίδας όσο και των παρτίδων που ακολουθούν και κληροδοτούνται διαδοχικά από παίκτη σε παίκτη. Το πράσινο βέλος υποδεικνύει κίνηση που γίνεται από τον πράσινο, ενώ το κόκκινο βέλος κίνηση που γίνεται από τον κόκκινο παίκτη. Σε κάθε περίπτωση θεωρείται ότι ξεκινά πρώτος ο πράσινος παίκτης.

* **Περίπτωση 0 κελιών:**

Πρόκειται για την παρτίδα (0), η οποία αποτελεί τετριμμένη **παρτίδα ήττας**, καθώς ο πράσινος δεν μπορεί να κάνει καμία κίνηση και άρα το παιχνίδι τελειώνει με νίκη του κόκκινου.

* **Περίπτωση 1 κελιού:**

Πρόκειται για την παρτίδα (1), η οποία αποτελεί τετριμμένη **παρτίδα νίκης**, καθώς ο πράσινος έχει μόνο μία πιθανή κίνηση (να γεμίσει το μοναδικό κενό κελί που έχει απομείνει) και η κίνηση αυτή οδηγεί σε νίκη του, αφού κληροδοτεί στον κόκκινο την τετριμμένη παρτίδα ήττας της περίπτωσης 0:

1. (0) , ΠΝ

* **Περίπτωση 2 κελιών:**

Από αυτό το σημείο και ύστερα εξετάζονται πλέον και διαφορετικές περιπτώσεις αρχικών στοιβών, καθώς υπάρχουν περισσότερες από μία πιθανές διατάξεις των κενών κελιών στο ταμπλό. Οι δύο μη-συμμετρικές διατάξεις για την περίπτωση 2 κενών κελιών είναι οι εξής:

* Μη διαδοχικά κελιά: είναι **παρτίδα ήττας**, καθώς ο πράσινος έχει δυνατότητα να γεμίσει μόνο ένα από τα 2 κενά κελιά, και άρα αναγκαστικά κληροδοτεί στον κόκκινο την τετριμμένη παρτίδα νίκης της περίπτωσης 1. Η αντίστοιχη αναπαράσταση με στοίβες δίνεται παρακάτω.

(1,1) (1) (0) , ΠΗ

* Διαδοχικά κελιά (όπου κανένα κελί δεν ανήκει στην κύρια διαγώνιο): είναι **παρτίδα νίκης**, καθώς ο πράσινος μπορεί με την κίνησή του να γεμίσει και τα 2 κενά κελιά, και άρα να κληροδοτήσει στον κόκκινο την τετριμμένη παρτίδα νίκης της περίπτωσης 0.

1. (0) , ΠΝ

Σημείωση: για την συγκεκριμένη περίπτωση διαφαίνεται μία ακόμα υποπερίπτωση, αυτή όπου τα 2 κελιά είναι διαδοχικά αλλά κάποιο ανήκει στην κύρια διαγώνιο. Αυτή η περίπτωση αποτελεί και πάλι παρτίδα ήττας για τον πράσινο, καθώς τα κελιά της διαγωνίου δεν μπορούν να παιχτούν διαδοχικά με άλλα κελιά και άρα θα πρέπει αναγκαστικά να γεμίσει μόνο ένα κελί. Η αναπαράστασή της μέσω στοιβών μπορεί να γίνει ως εξής:

(1,1) (1) (0) .

Παρατηρούμε δηλαδή πως, αν και εκ πρώτης όψεως μοιάζει με ξεχωριστή περίπτωση, ουσιαστικά μπορεί να αναχθεί στην πρώτη από τις 2 “βασικές” υποπεριπτώσεις και άρα δεν αποτελεί καινούργια διακριτή υποπερίπτωση.

Επιπλέον, στις περιπτώσεις που ακολουθούν οι υποπεριπτώσεις παρουσιάζονται κυρίως με βάση τις αρχικές στοίβες και όχι την ακριβή διάταξη των κελιών στον χώρο. Αυτό αφενός γιατί είναι αρκετά δύσκολο να εξεταστούν όλες οι πιθανές διατάξεις των κελιών (ειδικά για την περίπτωση με τα 5 κενά κελιά), και αφετέρου επειδή κάθε περίπτωση διάταξης μπορεί τελικά να αναχθεί σε κάποια από τις “βασικές” υποπεριπτώσεις (όπως έγινε και με την προηγούμενη). Εξαιτίας αυτού δεν θα ξαναγίνει επίσης ξεχωριστή μελέτη για την κύρια διαγώνιο, και όταν αναφέρονται διαδοχικά κελιά θα θεωρείται πως κανένα από αυτά δεν βρίσκεται πάνω σε αυτήν.

* **Περίπτωση 3 κελιών:**

Παρακάτω δίνονται οι 3 μη-συμμετρικές υποπεριπτώσεις αρχικών στοιβών, καθώς και μία πιθανή διάταξη των κελιών για την καθεμία:

* (3): πρόκειται για μία μοναδική στοίβα με 3 κελιά, και η μοναδική διάταξη που είναι δυνατόν να αντιστοιχεί σε αυτήν είναι τα 3 διαδοχικά κελιά. Είναι **παρτίδα νίκης** καθώς ο πράσινος μπορεί με την κίνησή του να γεμίσει και τα 3 κελιά, κληροδοτώντας στον κόκκινο την τετριμμένη παρτίδα ήττας (0):

1. (0) , ΠΝ

* (2,1): πρόκειται για 2 στοίβες, μία με ένα και μία με 2 κενά κελιά. Μια πιθανή διάταξη για αυτή την αναπαράσταση είναι η περίπτωση 2 διαδοχικών κελιών και ενός “σπαστού” κελιού (δηλαδή ενός κελιού που δεν είναι διαδοχικό με κανένα άλλο). Είναι **παρτίδα νίκης**, καθώς ο πράσινος μπορεί, γεμίζοντας το κατάλληλο κελί, να κληροδοτήσει στον κόκκινο την παρτίδα (1,1). Η παρτίδα (1,1) αποτελείται από 2 στοίβες ίδιου μεγέθους, και άρα όπως γνωρίζουμε από την θεωρία (και από την αντίστοιχη περίπτωση για 2 κενά κελιά) αποτελεί παρτίδα ήττας για τον παίκτη που θα παίξει σε αυτήν.

(2,1) (1,1) (1) (0) , ΠΝ

* (1,1,1): πρόκειται για 3 στοίβες που αποτελούνται η καθεμία από ένα κελί, και μία πιθανή διάταξη για αυτήν είναι η περίπτωση όπου υπάρχουν 3 σπαστά κελιά. Είναι **παρτίδα νίκης**, καθώς όμοια με πριν ο πράσινος μπορεί να κληροδοτήσει στον κόκκινο την παρτίδα ήττας (1,1).

(1,1,1) (1,1) (1) (0) , ΠΝ

Όπως διαπιστώνουμε από την παραπάνω ανάλυση, η περίπτωση τριών κενών κελιών αποτελεί πάντα παρτίδα νίκης για τον παίκτη που παίζει σε αυτήν, εφόσον αυτός παίζει βέλτιστα.

* **Περίπτωση 4 κελιών:**

Παρακάτω δίνονται οι 5 μη-συμμετρικές υποπεριπτώσεις αρχικών στοιβών, καθώς και μία πιθανή διάταξη των κελιών για την καθεμία:

* (4): πρόκειται για μία μοναδική στοίβα με 4 κελιά, και μπορεί να αντιστοιχεί μόνο στην περίπτωση τεσσάρων διαδοχικών κελιών. Είναι **παρτίδα νίκης**, καθώς ο πράσινος μπορεί να γεμίσει τα 2 διαδοχικά μη-ακριανά κελιά, κληροδοτώντας στον κόκκινο δύο σπαστά κελιά, δηλαδή την παρτίδα ήττας (1,1).

(4) (1,1) (1) (0) , ΠΝ

* (3,1): πρόκειται για 2 στοίβες, μία με ένα και μία με 3 κελιά. Μια πιθανή διάταξη για αυτήν είναι 3 διαδοχικά κελιά και ένα σπαστό. Είναι **παρτίδα νίκης**, καθώς ο πράσινος μπορεί να γεμίσει 2 από τα 3 διαδοχικά κελιά, και όμοια με πριν να κληροδοτήσει την παρτίδα ήττας (1,1) στον αντίπαλο.

(3,1) (1,1) (1) (0) , ΠΝ

* (2,2): πρόκειται για 2 στοίβες των 2 κελιών η καθεμία. Μία πιθανή διάταξη για την συγκεκριμένη περίπτωση είναι δύο ζεύγη διαδοχικών κελιών, όπου τα δύο ζεύγη δεν τέμνονται μεταξύ τους με κανέναν τρόπο. Γνωρίζουμε από την θεωρία πως πρόκειται για **παρτίδα ήττας**, αφού αποτελείται από 2 στοίβες ίδιου μεγέθους. Πράγματι, ο κόκκινος μπορεί να κερδίσει το παιχνίδι παίζοντας ως “καθρέφτης”, αντιγράφοντας δηλαδή την προηγούμενη κίνηση του πράσινου. Παρακάτω δίνονται οι δύο πιθανές εξελίξεις της παρτίδας, οι οποίες όπως είναι αναμενόμενο καταλήγουν και οι δύο σε ήττα για τον πράσινο:

(2,2) (2,1) (1,1) (1) (0) , ΠΗ

(2,2) (2) (0) , ΠΗ

* (2,1,1): μία στοίβα 2 κελιών και 2 στοίβες με ένα κελί η καθεμία. Μπορεί να αντιστοιχεί σε διάταξη 2 συνεχόμενων και 2 σπαστών κελιών, και αποτελεί **παρτίδα νίκης**, αφού ο πράσινος γεμίζοντας τα δύο διαδοχικά κελιά μπορεί να κληροδοτήσει στον κόκκινο την παρτίδα ήττας (1,1).

(2,1,1) (1,1) (1) (0) , ΠΝ

* (1,1,1,1): 4 στοίβες των τεσσάρων κελιών η καθεμία, το οποίο μπορεί να αντιστοιχεί σε διάταξη τεσσάρων σπαστών κελιών. Είναι **παρτίδα ήττας**, καθώς οι παίκτες αναγκαστικά θα παίζουν από ένα κελί την φορά και το αρχικό πλήθος των κελιών είναι άρτιος αριθμός.

(1,1,1,1) (1,1,1) (1,1) (1) (0) , ΠΗ

* **Περίπτωση 5 κελιών:**

Παρακάτω δίνονται οι 7 μη-συμμετρικές υποπεριπτώσεις αρχικών στοιβών, καθώς και μία πιθανή διάταξη των κελιών για την καθεμία:

* (5): πρόκειται για μία στοίβα με 5 κελιά, και η μοναδική διάταξη που μπορεί να της αντιστοιχεί είναι να υπάρχουν 5 διαδοχικά κενά κελιά. Πρόκειται για **παρτίδα νίκης**, καθώς ο πράσινος μπορεί να γεμίσει τα 3 μη-ακριανά κελιά, και άρα να κληροδοτήσει στον κόκκινο την παρτίδα ήττας (1,1).

(5) (1,1) (1) (0) , ΠΝ

* (4,1): μία στοίβα τεσσάρων κελιών και μία στοίβα ενός κελιού, το οποίο μπορεί να πρόκειται για διάταξη τεσσάρων διαδοχικών κελιών και ενός σπαστού. Πρόκειται για **παρτίδα νίκης**, αφού και πάλι ο πράσινος μπορεί να γεμίσει τα κατάλληλα 3 κελιά από τα 4 διαδοχικά, αφήνοντας στον κόκκινο την παρτίδα ήττας (1,1).

(4,1) (1,1) (1) (0) , ΠΝ

* (3,2): πρόκειται για 2 στοίβες, μία των δύο και μία των τριών κελιών. Μπορεί να αντιστοιχεί σε διάταξη 2 διαδοχικών μεταξύ τους κελιών και τριών διαδοχικών κελιών, όπου τα δύο και τρία διαδοχικά κελιά δεν τέμνονται μεταξύ τους με κανέναν τρόπο. Είναι **παρτίδα νίκης**, καθώς ο πράσινος γεμίζοντας ένα ακριανό κελί από τα 3 διαδοχικά κληροδοτεί στον κόκκινο την παρτίδα (2,2), η οποία όπως έχει αναλυθεί ήδη αποτελεί παρτίδα ήττας. Παρακάτω δίνονται οι δύο πιθανές εξελίξεις της παρτίδας, οι οποίες όπως αναμένεται καταλήγουν και οι δύο σε νίκη του πράσινου:

(3,2) (2,2) (2,1) (1,1) (1) (0) , ΠΝ

(3,2) (2,2) (2) (0) , ΠΝ

* (3,1,1): μία στοίβα τριών κελιών και δύο στοίβες του ενός κελιού η καθεμία, το οποίο πιθανώς αντιστοιχεί σε διάταξη τριών διαδοχικών κελιών και δύο σπαστών. Είναι **παρτίδα νίκης**, αφού ο πράσινος γεμίζοντας τα 3 διαδοχικά κελιά κληροδοτεί στον κόκκινο την παρτίδα ήττας (1,1).

(3,1,1) (1,1) (1) (0) , ΠΝ

* (2,2,1): πρόκειται για 2 στοίβες των δύο κελιών και μία στοίβα του ενός. Μία πιθανή διάταξη που της αντιστοιχεί είναι 2 ζεύγη διαδοχικών κελιών τα οποία δεν τέμνονται μεταξύ τους με κανέναν τρόπο, και επιπλέον ένα σπαστό κελί. Πρόκειται για **παρτίδα νίκης**, αφού γεμίζοντας το σπαστό κελί ο πράσινος αφήνει στον κόκκινο την γνωστή παρτίδα ήττας (2,2). Παρακάτω δίνονται οι δύο πιθανές εξελίξεις της παρτίδας, οι οποίες όπως αναμένεται καταλήγουν και οι δύο σε νίκη του πράσινου:

(2,2,1) (2,2) (2,1) (1,1) (1) (0) , ΠΝ

(2,2,1) (2,2) (2) (0) , ΠΝ

* (2,1,1,1): μία στοίβα 2 κελιών και 3 στοίβες του ενός κελιού η καθεμία. Μπορεί να πρόκειται για διάταξη με 2 διαδοχικά κελιά και 3 σπαστά και αποτελεί **παρτίδα νίκης**, καθώς γεμίζοντας ένα από τα δύο διαδοχικά κελιά ο πράσινος κληροδοτεί στον κόκκινο την παρτίδα (1,1,1,1), η οποία όπως έχει αναλυθεί ήδη στην περίπτωση 4 είναι παρτίδα ήττας.

(2,1,1,1) (1,1,1,1) (1,1,1) (1,1) (1) (0) , ΠΝ

* (1,1,1,1,1): πέντε στοίβες του ενός κελιού η καθεμία, το οποίο μπορεί να αντιστοιχεί σε διάταξη με πέντε σπαστά κελιά. Εφόσον οι παίκτες θα παίζουν αναγκαστικά ένα κελί την φορά και ο αρχικός αριθμός των κελιών είναι περιττός, πρόκειται για **παρτίδα νίκης** του πράσινου.

(1,1,1,1,1) (1,1,1,1) (1,1,1) (1,1) (1) (0) , ΠΝ

Όπως διαπιστώνουμε από την παραπάνω ανάλυση, η περίπτωση πέντε κενών κελιών αποτελεί πάντα παρτίδα νίκης για τον παίκτη που παίζει σε αυτήν, εφόσον αυτός παίζει βέλτιστα.

**Προγραμματιστικό Μέρος**

Παραδοχες οσον αναφορουν τις συναρτησεις που δινόντουσαν ετοιμες στο template.

Αλλαξαμε την isBoardFull και ο νεος της κωδικας είναι όπως φαινεται από κατω.

def isBoardFull(*board*, *N*):

*# Function for checking if the board is full*

    for i in range(1, *N*\**N* + 1):

*# if the cell does not contain G or R then it is empty*

        if *board*[i] != 'G' and *board*[i] != 'R':

            return False  *# return False if there is an empty cell*

    return True

Επιστρεφει δηλαδη True αμα όλα τα tiles του πινακα εχουν γραφτει από πανω με το γραμμα G η R. Αυτό δειχνει ότι καποιος από τους δυο παικτες το εχει κρατησει και αρα αμα είναι όλα κρατημε το ταμπλο είναι γεματο.

Δεν χρησημοποιησαμε την getRowAndColumn μιας και ειχαμε φτασει σε σημειο στο προγραμμα μας, όταν δωθηκε αυτή η συναρτηση, που την ειχαμε υλοποιησει αυτή την λειτουργια με άλλες συναρτησεις.

Τωρα θα αρχισει η αναλυση του κωδικα με την σειρα που εκτελειται ο κωδικας όταν τρεχουμε το προγραμμα.

Στην αρχη του προγραμματος αφου εκτυπωθουν οι κανονες και ο τροπος που παιζεται το παιχνιδι στην οθονη το προγραμμα μας τρεχει τις παρακατω γραμμες του κωδικα πριν αρχισουν τα input του χρηστη.

maxNumMoves = 3

playNewGameFlag = True

while playNewGameFlag:

if not startNewGame():

break

N = getBoardSize()

nimBoard = initializeBoard(N)

initialNimBoard = nimBoard.copy()

playerLetter, computerLetter = inputPlayerLetter()

turn = whoGoesFirst()

diagonalCells = getDiagonalCells(nimBoard)

computerStrategy = howComputerPlays()

Αρχικα δηλωνουμε τον μεγιστο αριθμο κινησεων (ποσα κουτια μπορει να παιξει ο υπολογιστης η ο παικτης σε μια σειρα) και ένα flag που χρησημοποιεται για να ξερουμε ποτε ο παικτης επιλεξει να αρχισει το παιχνιδι από την αρχη. Στη συνεχεια το Ν που είναι το μεγεθος του board αλλα και το ιδιο το board oπως θα το τυπωνουμε αρχικοποιειται. Εχουμε ορισει όπως φαινεται άλλο ένα board που το ονομασαμε initialNimBoard το οποιο θα εχει την μορφη του αρχικου board καθολη την διαρκεια του παιχνιδιου (μιας και ο πινακας nimBoard αλλαζει κατά την εκτελεση του προγραμματος). Υστερα οι μεταλητες που με βαση τις εισαγωγες του χρηστη στα αρχικα μηνυματα κρατανε ποιο γραμμα θα εχει ο παικτης και ποιο ο υπολογιστης καθως και ποιος θα παιξει πρωτος που με βαση την whoGoesFirst είναι 50-50. Οσον αναφορα τωρα τον πινακα diagonalCells και την computerStrategy. Ο diagonalCells είναι ενας πινακας που με την βοηθεια της συναρτησης getDiagonalCells ανεξαρτητως του μεγεθους του board βρισκει και αποθηκευει τα κελια που είναι στην διαγωνιο του board, όπως φαινεται παρακατω και στην getDiagonalCells. Τελος η μεταβλητη computerStrategy επισστρεφει το ονομα της στρατηγικης που θα χρησημοποιεισει ο υπολογιστης που το παιρνει από την εισαγωγη του χρηστη στην αρχη του προγραμματος.

*# define a function that returns the name of the cells in the diagonal of the nimboard*

def getDiagonalCells(*nimBoard*):

*# get the length of the nimboard*

length = len(*nimBoard*)

*# get the number of rows and columns*

rows = int(math.sqrt(length))

*# create a list of the diagonal cells*

diagonalCells = []

*# create a list of the diagonal cells*

for i in range(0, rows):

cell\_name = *nimBoard*[i\*rows + i + 1]

number = re.findall(r'\d+', cell\_name)[0]

diagonalCells.append(int(number))

return diagonalCells

Συνεχιζουμε τωρα αφου ο χρηστης εχει υποβαλει τις προτιμησεις του (μεγεθος του board, σταρτιγικη υπολογιστη, και χρωμμα που προτιμαει να εχει). Τωρα εχουμε 3 περιπτωσεις. (μια για κάθε στρατιγικη που μπορει να παιξει ο υπολογιστης).

* Random Στρατηγικη

if computerStrategy == 'random':

while not isBoardFull(nimBoard, N):

if turn == 'player':

*# Player's turn.*

move = getPlayerMove(nimBoard, diagonalCells, initialNimBoard)

while (move == False):

move = getPlayerMove(

nimBoard, diagonalCells, initialNimBoard)

makeMove(nimBoard, playerLetter, move)

drawNimPalette(nimBoard, N)

if (isBoardFull(nimBoard, N) is True):

drawNimPalette(nimBoard, N)

print(

bcolors.HEADER + 'Congrats you have beaten the computer!' + bcolors.ENDC)

break

else:

turn = 'computer'

else:

*# Computer's turn.*

move = getComputerMove(

nimBoard, computerStrategy, initialNimBoard, playermoves)

print(move)

makeMove(nimBoard, computerLetter, move)

drawNimPalette(nimBoard, N)

if (isBoardFull(nimBoard, N) is True):

drawNimPalette(nimBoard, N)

print(

bcolors.HEADER + 'The computer has beaten you! You lose.' + bcolors.ENDC)

break

else:

turn = 'player'

Εστω ότι είναι η σειρα του παικτη τοτε καλείται αρχικα η συναρτηση getPlayerMove όπως φιανεται ο κωδικας της παρακατω.

def getPlayerMove(*nimBoard*, *diagonalCells*, *initialNimBoard*):

*# This function returns the player's move.*

*# the player will choose a tile that is empty and will fill it with his label. Then if he chooses to continue he will choose another tile and so on until he has picked 3 consecutive tiles in the same row or column*

possibleMoves = getAvailableCells(*nimBoard*)

choice = []

while (1):

print(bcolors.QUESTION +

'[Q4] Which is the first tile you choose:' + bcolors.ENDC)

num = input()

if num.isdigit():

num = int(num)

if num in possibleMoves: *# if the selected tile is available*

choice.append(num)

if num in *diagonalCells*: *# if a main diagonal cell is chosen as the first move, the player can choose no more tiles*

return choice

while (1):

print(bcolors.QUESTION +

'[Q5] Which is the second tile you choose:' + bcolors.ENDC)

num1 = input()

if num1.isdigit():

num1 = int(num1)

if num1 in possibleMoves and checkValidMove(*initialNimBoard*, choice + [num1]):

choice.append(num1)

while (1):

print(bcolors.QUESTION +

'[Q6] Which is the third tile you choose:' + bcolors.ENDC)

num2 = input()

if num2.isdigit():

num2 = int(num2)

if num2 in possibleMoves and checkValidMove(*initialNimBoard*, choice + [num2]):

choice.append(num2)

if checkValidMove2(*initialNimBoard*, choice):

return choice

else:

print(bcolors.ERROR +

'ERROR 8: The tiles you chose were not consecutive. Try again from the beginning...' + bcolors.ENDC)

return False

else:

print(bcolors.ERROR +

'ERROR 6: The tile you chose is not available or invalid. Try again...' + bcolors.ENDC)

else:

print(bcolors.MSG +

'Your input was not an integer so your turn was terminated.' + bcolors.ENDC)

if checkValidMove2(*initialNimBoard*, choice):

print(bcolors.GREEN +

'Move executed succesfully!' + bcolors.ENDC)

return choice

else:

print(bcolors.ERROR +

'ERROR 8: The tiles you chose were not consecutive. Try again from the beginning...' + bcolors.ENDC)

return False

else:

print(bcolors.ERROR +

'ERROR 5: The tile you chose is not available or invalid. Try again...' + bcolors.ENDC)

else:

print(bcolors.MSG +

'Your input was not an integer so your turn was terminated.' + bcolors.ENDC)

if checkValidMove2(*initialNimBoard*, choice):

print(bcolors.GREEN +

'Move executed succesfully!' + bcolors.ENDC)

return choice

else:

print(bcolors.ERROR +

'ERROR 8: The tiles you chose were not consecutive. Try again from the beginning...' + bcolors.ENDC)

return False

else:

print(bcolors.ERROR +

'ERROR 6: The tile you chose is not available. Try again...' + bcolors.ENDC)

else:

print(bcolors.ERROR +

'ERROR 7: Your input was not an integer. Please enter a valid choice!' + bcolors.ENDC)

Η συναρτηση αυτή μαζει και με καποιες άλλες που καλουνται μεσα από αυτή είναι υπευθυνη για να παιρνει τις κινήσεις του παικτη και να τις ελεγχει αμα είναι νομιμες. Αμα δεν είναι τοτε τυπωνονται τα αντιστοιχα λαθη και ειτε του δινεται η ευκαρια να ξαναπροσπαθησει ειτε του ολοκληρωνεται η σειρα και κρατιουνται οι επιλογες που εχει κανει μεχρι τοτε (εξαρταται το λαθος). Ας αρχισουμε την αναλυση της συναρτησης αυτης. Αρχικα καλουμε την συναρτηση getAvailableCells που παιρνει σαν ορισμα το board κάθε φορα. Αυτή η συναρτηση ουσιαστικα επιστρεφει κάθε φορα που την καλεις μια λιστα με όλα τα tiles του board που είναι ελευθερα και μπορει να γραψει πανω ειτε ο παικτης ειτε ο υπολογιστης. Στην συνεχεια δημιουργείται μια λιστα που ονομασαμε choice που θα είναι ουσιστικα η λιστα που επιστρεφει η συναρτηση getPlayerMove και θα περιεχει τα tiles της κινησης του παικτη. Τωρα όπως βλεπουμε στην συνεχεια ολη η υπολοιπη συναρτηση είναι μεσα σε μια while και μεσα της υπαρχουν και άλλες παρομοιες while. Αυτό γινεται γιατι εχουμε φτιαξει το προγραμμα ετσι ώστε αναλογως το θεμα που παρουσιαζεται με τις επιλογες του χρηστη να του δινεται σε καποιες περιπτωσεις η επιλογη να επαναλαβει ολο την κινηση η καποια μερη της, ετσι θα πρεπει να είναι μεσα σε μια while loop. Ετσι μεσα σε αυτή την πρωτη while εχουμε την επιλογη του πρωτου tile της κινηση του παικτη. Αναγκαζουμε τον παικτη να δωσει τουλαχιστον ένα tile για την κινηση του οποτε αυτό το κομματι είναι απαιραιτητο. Αμα ο χρηστης βαλει καποιο μη αποδεκτο input (κατι διαφορετικο από έναν ακεραιο που αντιπροσοπευει τον αριθμο του tile απο το board η ενα tile που δεν μπορει να επιλεξει γιατι είναι κατειλημμένο, δεν είναι δηλαδη στην λιστα availableMoves) τοτε τυπωνεται παλι το μηνυμα να εισαγει το πρωτο tile. Επισης αμα η επιλογη του είναι καποι tile της διαγωνιου τοτε ολοκληρωνεται η σειρα του κρατώντας μονο αυτό το tile. Τωρα για το δευτερο tile υπαρχουν και άλλες προυποθεσεις που πρεπει να τηρουνται. Αρχικα πρεπει παλι να είναι καποιος ακεραιος απο την λιστα availableMoves αλλα επισης πρεπει να είναι νομιμη η κινηση που αποτελει το πρωτο τile και το δευτερο μαζι. Αυτό τον ελεγχο τον κανει η συναρτηση checkValidMove που φαινεται από κατω.

def checkValidMove(*nimBoard*, *move*):

*# get the length of the move*

*# print(move)*

length = len(*move*)

*# check if the move is valid*

if length == 3:

if *move*[length-1] != *move*[length-2]:

*# check if the move is in the same row*

if *move*[length-1] in getRowCells(*nimBoard*, *move*[length-2]):

if *move*[length-1] in getRowCells(*nimBoard*, *move*[0]):

return True

*# check if the move is in the same column*

elif *move*[length-1] in getColumnCells(*nimBoard*, *move*[length-2]):

if *move*[length-1] in getColumnCells(*nimBoard*, *move*[0]):

return True

else:

return False

else:

if *move*[length-1] in getRowCells(*nimBoard*, *move*[0]):

return True

*# check if the move is in the same column*

if *move*[length-1] in getColumnCells(*nimBoard*, *move*[0]):

return True

else:

return False

elif length == 2:

if *move*[length-1] not in getDiagonalCells(*nimBoard*):

if *move*[length-1] in getRowCells(*nimBoard*, *move*[0]):

return True

*# check if the move is in the same column*

elif *move*[length-1] in getColumnCells(*nimBoard*, *move*[0]):

return True

else:

return False

Όπως ειδαμε πριν η συναρτηση καλειται με ορισμα το αρχικο board μιας και θελουμε να δουμε αμα είναι νομιμη η κινηση από αποψη θεσης των tile (εχουμε ηδη ελέγξει αμα απασχολειται ηδη το tile) και η λιστα choice που περιεχει τα ηδη περασμενα tiles της κινησης του παικτη μαζι με το πιο προσφατο tile που θελει να χρησημοποιησει ο χρηστης. Τωρα μεσα στην συναρτηση checkValidMove εχουμε δυο περιπτωσεις αναλογως αμα ελεγχουμε το δευτερο η το τριτο tile της κινησης του παικτη. Αμα είναι το δευτερο τα πραγματα είναι απλα κοιταμε αμα το προηγουμενο tile (στην περιπτωση αυτή το πρωτο) είναι στην ιδια γραμμη η στην ιδια στήλη (με τις συναρτησεις getRowCells, getColumnCells που επιστρεφουν σαν λιστα τα tiles που είναι στην γραμμη η στηλη με το αρχικο tile) με αυτό που θελουμε να επιλεξουμε. Αμα είναι η συναρτηση επιστρεφει αληθης και το tile προστίθεται στην λιστα με τα tiles της κινησης του παικτη, αμα όχι τοτε τυπωνεται ένα μηνυμα που του λεει ότι το tile αυτό είναι λαθος και να ξαναπροσπαθησει. Πριν παμε να δουμε πως ελεγχεται αμα δωσουμε το τριτο tile ας σημειωσουμε πως εχουμε φτιαξει το προγραμμα μας ετσι ώστε να τερματιζει ο παικτης την σιερα του, κρατωντας τις επιλογες των tiles που εχει κανει μεχρι τωρα, με παραπανω από έναν τροπους. Ο ενας τροπος είναι να δωσει κατι άλλο από αριθμο (καποιο γραμμα) και ο άλλος είναι να δωσει δυο tiles συνεχομενα ιδια. Πχ με την εισαγωγη 2,3,3 ο παικτης τερματιζει την σειρα του κρατωντας σαν κινηση το 2,3. Αξιζει να σημειωθει πως αμα ο παικτης δωσει σαν εισοδο το 2,2,3 τοτε το προγραμμα μας θα καταλαβει ότι ο χρηστης εχει δωσει δυο ιδια στην και θα κρατησει μονο το 1 κανοντας την κινηση να είναι το 2,3. Αφου το ξεκαθαρισαμε παμε να δουμε τον τροπο ελεγχου του τριτου tile. Βλεπουμε ότι ελεγχουμε αμα το τριτο tile είναι ιδιο με το δευτερο (αμα είναι τοτε ο ελεγχος είναι ιδιου στυλ με τον ελεγχο που περιγραψαμε πριν). Αμα δεν είναι ιδια λοιπον παμε να δουμε τις προυποθεσεις. Αμα θελουμε να είναι κατά γραμμη θα πρεπει να είναι στην ιδια γραμμη τοσο με το πρωτο οσο και με το δευτερο tile. Για αυτό κανουμε ελεγχο με την getRowCells και για τα δυο αυτά tile. Ενώ αμα παμε κατά στηλη θα πρεπει παλι να είναι στην ιδια στηλη τοσο με το πρωτο οσο και με το δευτερο tile. Αμα δεν υπηρχαν αυτές οι προυποθεσεις και ελεγχαμε ας πουμε μονο με σχεση το δευτερο tile τοτε ο παικτης θα μπορουσε να ξεγελασει το συστημα και να ειχε την δυνατοτητα να σχεδιασει ένα Γ με την κινηση του, το οποιο απαγορευεται. Ας επιστρεψουμε τωρα στην getPlayerMove. Αμα περασει λοιπον τον ελεγχο το δευτερο tile τοτε το βαζουμε στην λιστα με τα tiles της κινησης του παικτη. Προχωραμε τωρα στο τριτο tile. Γινονται παλι οι ιδιοι ελεγχοι (αμα είναι ακεραιος και είναι available αμα επιστρεφει true η checkValidMove) και αμα γινουν επιτυχως παλι βαζουμε το τριτο tile sτην λιστα. Τωρα όμως πρεπει καπως να γινει ο ελεγχος για αυτό που περιγραψαμε πανω (αμα δυο συνεχομενα τile είναι ιδια να κραταει μονο το κομματι που είναι διαφορετικα και να ελγχει αμα είναι νομιμη παλι). Αυτό γινεται με την checkValidMove2 οπου φαινεται παρακατω.

def checkValidMove2(*nimBoard*, *move*):

length = len(*move*)

*move*.sort(*reverse*=False) *# arranges the move list into ascending order*

*# check if the move is valid*

*# check if the move is consecutive*

if length == 3:

*# if some of the tiles have been chosen more than once:*

if *move*[0] == *move*[1] == *move*[2]:

return True

elif (*move*[0] == *move*[1]) and (*move*[0] == *move*[2] - 1 or *move*[0] == *move*[2] - N):

return True

elif (*move*[1] == *move*[2]) and (*move*[0] == *move*[1] - 1 or *move*[0] == *move*[1] - N):

return True

*# if all the tiles have been chosen exactly once:*

*# check if the tiles are consecutive in the same row*

elif (*move*[0] == *move*[1] - 1) and (*move*[0] == *move*[2] - 2):

return True

*# check if the tiles are consecutive in the same column*

elif (*move*[0] == *move*[1] - N) and (*move*[0] == *move*[2] - 2\*N):

return True

else:

return False

elif length == 2:

*# if the player chose the same tile twice*

if *move*[0] == *move*[1]:

return True

*# if the two chosen tiles are different*

else:

*# check if the tiles are consecutive in the same row*

if *move*[0] == *move*[1] - 1:

return True

*# check if the tiles are consecutive in the same column*

elif *move*[0] == *move*[1] - N:

return True

else:

return False

elif length == 1:

return True

ΕΞΗΓΗΣΗ ΚΩΔΙΚΑ