**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**

Παραδοχές οσον αναφορουν τις συναρτήσεις που δινόντουσαν ετοιμες στο template.

Αλλάξαμε 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).

playermoves = []

*# provided this code make the computer play with the random strategy*

    while not isBoardFull(nimBoard, N):

        if turn == 'player':

            playermoves = []

*# Player's turn.*

            move = getPlayerMove(nimBoard, diagonalCells, initialNimBoard)

            while (move == False):

                move = getPlayerMove(

                    nimBoard, diagonalCells, initialNimBoard)

            playermoves.extend(move)

            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'

Αρχικα εχουμε μια λιστα που αρχικα είναι κενη (playermoves). Αυτή η λιστα θα χρησιμοποιηθεί για την στρατηγική copycat οπου εκει μεσα θα αποθηκευεται η κινηση που εκενε ο παικτης κάθε φορα στην σειρα του.

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 υπήρχαν αυτές οι προυποθεσεις και ελέγχαμε ας πουμε μονο με σχέση το δευτερο tile τότε ο παικτης θα μπορούσε 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 ελέγχει 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

Ετσι αφου τελειωσαμε με την getPlayerMove (δηλαδη ολους τους ελεγχους για την εγκυροτητα των tiles που θελει ο παικτης να εχει στην κικνηση του) προχωραμε παρακατω στο main προγραμμα μας.

*# Player's turn.*

            move = getPlayerMove(nimBoard, diagonalCells, initialNimBoard)

            while (move == False):

                move = getPlayerMove(

                    nimBoard, diagonalCells, initialNimBoard)

            playermoves.extend(move)

            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'

Αφου λοιπον καλεσαμε και εκτελεσαμε την getPlayerMove βλεπουμε ότι υπαρχει μια while από κατω η οποια εκτελει την ιδια εντολη που εξηγησαμε συνωεχομενα μεχρι να μην βγαλει error. Αυτό το κανουμε για να μπορουμε να ελεγχουμε για αμα είναι συνεχομενια τα tiles που εδωσε ο χρηστης συνεχομενα και όχι μονο μια φορα (αμα κανει λαθος το προγραμμα του ζηταει να επαναλαβει ολη την κινηση από την αρχη αρα πρεπει καπως να ελεγχουμε και μετα την πρωτη εισαγωγη του χρηστη αμα είναι συνεχομενα τα tiles). Μετα από αυτό τον ελεγγο (όταν ο χρηστης τελειωσει με το input της κινησης του και είναι νομιμη) βαζουμε αυτή την κινηση στον πινακα που περιγραψαμε στην αρχη, που κραταει την τελευταια κινηση του παικτη (για την copycat). Στην συνεχεια καλουμε την συναρτηση makeMove η οποια είναι υπευθυνη για την ειαγωγη της κινησης στο ταμπλο, όπως φαινεται στον κωδικα της από κατω.

def makeMove(*nimBoard*, *letter*, *move*):

*# This function places the letter on the board at the move*

*# (i.e. it changes the board).*

    if isinstance(*move*, list):

        for i in range(len(*move*)):

*nimBoard*[*move*[i]] = *letter*

*nimBoard*[0] = *nimBoard*[0] + 1

Να σημειωθει ότι εδώ γινεται και η αυξηση της μεταβλητης counter οπου μας δειχνει κάθε φορα ποσα tiles του ταμπλο είναι πιασμενα.

Αφου λοιπον εκτυπωσουμε την κινηση στο ταμπλο προχωραμε στην εκτυπωση του ταμπλο στην οθονη με την drawNimPalette. Τελος εχουμε μια συνθηκη η οποια αμα δει ότι μετα από την ολοκληρωση της κινησης το ταμπλο είναι γεματο, τυπωνει ένα μηνυμα νικης για τον παικτη.

Στην περιπτωση που είναι η σιερα του υπολογιστη εκτελειτε το κομματι κωδικα που φαινεται παρακατω.

        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'

Όπως βλεπουμε το κομματι του υπολογιστη είναι αρκετα παρομοιο με το τι γινεται όταν παιζει ο παικτης. Η μονη διαφορα είναι το μήνυμα νικης που τωρα εγινε ηττας για τον παικτη, το else στο τελος ,που είναι για την αναδρομικη εκτελεση του κωδικα και την ενναλαγη σειρας συνεχεια, και το πιο βασικο η κληση της getComputerMove. Αυτό λοιπον είναι το κομματι που πρεπει να αναλυσουμε.

def getComputerMove(*nimBoard*, *computerStrategy*, *initialNimBoard*, *playermoves*):

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

*# The computer will choose a random move from the list of*

*# available moves.*

    if *computerStrategy* == 'random':

        return getComputerMove\_random(*nimBoard*, *initialNimBoard*)

    else:

        if *computerStrategy* == 'first free':

            return getComputerMove\_firstfit(*nimBoard*, *initialNimBoard*)

        else:

            if *computerStrategy* == 'copycat':

                return getComputerMove\_copycat(*nimBoard*, *initialNimBoard*, *playermoves*)

            else:

                print(

                    bcolors.ERROR + 'ERROR 2: Incomprehensible strategy was provided. Try again...' + bcolors.ENDC)

Στον παραπανω κωδικα της getComputerMove εχουμε 3 υποπεριπτωσεις (μια για κάθε στρατιγικη του υπολογιστη). Αναλογως λοιπον με ποια στρατηγική εχει επιλεχτεί στην αρχη της εκτελεσης του προγραμματος καλουνται διαφορετικες συναρτησεις.

Με την κληση της getComputerMove εκτελειτε παρομοιος κωδικας για ολες τις στρατιγικες:

def getComputerMove\_random(*nimBoard*, *initialNimBoard*):

*# This function returns a random move from the list of*

*# available moves.*

*# return a random available tile to start the move*

    possibleMoves = getAvailableCells(*nimBoard*)

    choice = []

*# if there's more than five available cells left (case of 0 or 1 cell left are considered trivial and also included)*

    if len(possibleMoves) > 5 or len(possibleMoves) < 2:

        choice = randomMoreThanFiveLeft(*nimBoard*, *initialNimBoard*)

*# if there's five or less available cells left (not including 0 or 1 cell left)*

    elif len(possibleMoves) == 2:

        choice = twoCellsLeft(

            'random', *nimBoard*, *initialNimBoard*, possibleMoves, [])

    elif len(possibleMoves) == 3:

        choice = threeCellsLeft(

            'random', *nimBoard*, *initialNimBoard*, possibleMoves, [])

    elif len(possibleMoves) == 4:

        choice = fourCellsLeft(

            'random', *nimBoard*, *initialNimBoard*, possibleMoves, [])

    elif len(possibleMoves) == 5:

        choice = fiveCellsLeft(

            'random', *nimBoard*, *initialNimBoard*, possibleMoves, [])

    return choice

def getComputerMove\_firstfit(*nimBoard*, *initialNimBoard*):

*# This function returns the first free move from the list of*

*# available moves.*

*# return the first available tile to start the move*

    possibleMoves = getAvailableCells(*nimBoard*)

    choice = []

*# if there's more than five available cells left (case of 0 or 1 cell left are considered trivial and also included)*

    if len(possibleMoves) > 5 or len(possibleMoves) < 2:

        choice = firstfreeMoreThanFiveLeft(*nimBoard*, *initialNimBoard*)

*# if there's five or less available cells left (not including 0 or 1 cell left)*

    elif len(possibleMoves) == 2:

        choice = twoCellsLeft('first free', *nimBoard*,

*initialNimBoard*, possibleMoves, [])

    elif len(possibleMoves) == 3:

        choice = threeCellsLeft(

            'first free', *nimBoard*, *initialNimBoard*, possibleMoves, [])

    elif len(possibleMoves) == 4:

        choice = fourCellsLeft(

            'first free', *nimBoard*, *initialNimBoard*, possibleMoves, [])

    elif len(possibleMoves) == 5:

        choice = fiveCellsLeft(

            'first free', *nimBoard*, *initialNimBoard*, possibleMoves, [])

    return choice

def getComputerMove\_copycat(*nimBoard*, *initialNimBoard*, *playermoves*):

*# This function returns the copycat move from the list of*

*# available moves.*

*# return the first available tile to start the move*

    possibleMoves = getAvailableCells(*nimBoard*)

    choice = []

*# if there's more than five available cells left (case of 0 or 1 cell left are considered trivial and also included)*

    if len(possibleMoves) > 5 or len(possibleMoves) < 2:

        choice = copycatMoreThanFiveLeft(

*nimBoard*, *initialNimBoard*, *playermoves*)

*# if there's five or less available cells left (not including 0 or 1 cell left)*

    elif len(possibleMoves) == 2:

        choice = twoCellsLeft(

            'copycat', *nimBoard*, *initialNimBoard*, possibleMoves, *playermoves*)

    elif len(possibleMoves) == 3:

        choice = threeCellsLeft(

            'copycat', *nimBoard*, *initialNimBoard*, possibleMoves, *playermoves*)

    elif len(possibleMoves) == 4:

        choice = fourCellsLeft(

            'copycat', *nimBoard*, *initialNimBoard*, possibleMoves, *playermoves*)

    elif len(possibleMoves) == 5:

        choice = fiveCellsLeft(

            'copycat', *nimBoard*, *initialNimBoard*, possibleMoves, *playermoves*)

    return choice

Η μονη διαφορα μεταξυ των συναρτησεων αυτων είναι ότι καλουν αναλογως μια άλλη σταρτιγικη και ότι στην copycat εχουμε και την λιστα με την προηγουμενη κινηση του παικτη.

Στην αρχη εχουμε την λιστα choice οπου είναι η λιστα που εχουμε να επιστρεφει την κινηση του υπολογιστη. Το σημαντικο κομματι είναι όμως οι ελεγχοι που γινονται για το ποσα ελευθερα tile απομενουν στο ταμπλο. Αμα αυτά τα tiles είναι παραπανω από 5 ο υπολογιστης παιζει κανονικα την επιλεγμενη στρατιγικη (random). Αμα όμως τα ελευθερα tiles είναι λιγοτερα από 5 τοτε ο υπολογιστης εκτος από την στρατιγικη του παιζει για να νικησει αμα είναι εφικτο. Αυτό γινεται με την τις συναρτησεις twoCellsLeft, threeCellsLeft, fourCellsLeft , fiveCellsLeft.

def twoCellsLeft(*mode*, *nimBoard*, *initialNimBoard*, *possibleMoves*, *playermoves*):

    global result\_type

    result\_type = 'undefined'

    choice2 = []

*# if the two cells are consecutive and none of them is on the diagonal then the computer chooses these cells and wins*

    if *possibleMoves*[1] in getCellNeighbours(*possibleMoves*[0], *initialNimBoard*) and *possibleMoves*[0] not in getDiagonalCells(*initialNimBoard*) and *possibleMoves*[1] not in getDiagonalCells(*initialNimBoard*):

        choice2.extend([*possibleMoves*[0], *possibleMoves*[1]])

        result\_type = 'specific'

*# else if the cells can't both be chosen (because at least one of them is on the diagonal or they are not consecutive)*

*# then the computer continues playing according to its previous strategy*

    else:

        if *mode* == 'random':

            t\_choice = randomMoreThanFiveLeft(*nimBoard*, *initialNimBoard*)

            result\_type = 'random'

        elif *mode* == 'first free':

            t\_choice = firstfreeMoreThanFiveLeft(*nimBoard*, *initialNimBoard*)

            result\_type = 'first free'

        elif *mode* == 'copycat':

            t\_choice = copycatMoreThanFiveLeft(

*nimBoard*, *initialNimBoard*, *playermoves*)

            result\_type = 'copycat'

        choice2.extend(t\_choice)

    return choice2

ΕΞΗΓΗΣΗ ΚΩΔΙΚΑ

def threeCellsLeft(*mode*, *nimBoard*, *initialNimBoard*, *possibleMoves*, *playermoves*):

    global result\_type

    result\_type = 'undefined'

    choice3 = []

*# if the three cells are not consecutive and one of them has the other 2 as adjacent (with none them on the diagonal) then the computer picks the one in the middle*

    if getCellwithNumNeighbours(2, *initialNimBoard*, *possibleMoves*) == False:

*# if thats not the case the computer plays the selected strategy*

        if getCellwithNumNeighbours(1, *initialNimBoard*, *possibleMoves*) == False:

            if *mode* == 'random':

                t\_choice = randomMoreThanFiveLeft(*nimBoard*, *initialNimBoard*)

                result\_type = 'random'

            elif *mode* == 'first free':

                t\_choice = firstfreeMoreThanFiveLeft(*nimBoard*, *initialNimBoard*)

                result\_type = 'first free'

            elif *mode* == 'copycat':

                t\_choice = copycatMoreThanFiveLeft(

*nimBoard*, *initialNimBoard*, *playermoves*)

                result\_type = 'copycat'

            choice3.extend(t\_choice)

*# if there is a cell with 1 neighbour (neither of them in the diagonal), then that's the computer's next move*

        else:

            t\_choice = getCellwithNumNeighbours(

                1, *initialNimBoard*, *possibleMoves*)

            choice3.append(t\_choice)

*# if there is a cell with 2 neighbours*

    else:

*# if the three cells are consecutive and none of them is on the diagonal then the computer chooses these cells and wins*

        t\_choice = getCellwithNumNeighbours(2, *initialNimBoard*, *possibleMoves*)

*# check if the three cells are consecutive and the move is legal and not on the diagonal if yes then the computer chooses the three cells and wins*

        temp = []

        temp.append(*possibleMoves*[0])

        temp.append(*possibleMoves*[1])

        temp.append(*possibleMoves*[2])

        if (abs(*possibleMoves*[0] - *possibleMoves*[1]) == 1 or abs(*possibleMoves*[0] - *possibleMoves*[1]) == N) and (abs(*possibleMoves*[1] - *possibleMoves*[2]) == 1 or abs(*possibleMoves*[1] - *possibleMoves*[2]) == N) and *possibleMoves*[0] not in getDiagonalCells(*initialNimBoard*) and *possibleMoves*[1] not in getDiagonalCells(*initialNimBoard*) and *possibleMoves*[2] not in getDiagonalCells(*initialNimBoard*) and checkValidMove(*initialNimBoard*, temp):

            choice3.extend(temp)

            result\_type = 'consecutive'

        else:

            choice3.append(t\_choice)

    return choice3

ΕΞΗΓΗΣΗ ΚΩΔΙΚΑ

def fourCellsLeft(*mode*, *nimBoard*, *initialNimBoard*, *possibleMoves*, *playermoves*):

    choice4 = []

    global result\_type4

    result\_type4 = 'undefined'

*# picks every combination of 2 available cells and, if they are consecutive and non-diagonal,*

*# checks if they give a winning move*

    for i in range(0, 4):

        t\_posmoves = []

        t\_posmoves.append(*possibleMoves*[i])

        for j in range(0, 4):

            checklist = []

            if i != j:

                t\_posmoves.append(*possibleMoves*[j])

                twoCellsLeft(*mode*, *nimBoard*, *initialNimBoard*,

                             t\_posmoves, *playermoves*)

*# checks if the 2 cells are consecutive and not on the diagonal*

                if result\_type == 'specific':

                    for k in range(0, 4):

                        if *possibleMoves*[k] not in t\_posmoves:

                            checklist.append(*possibleMoves*[k])

                    twoCellsLeft(*mode*, *nimBoard*,

*initialNimBoard*, checklist, *playermoves*)

*# checks if the 2 cells that were previously chosen give a winning move*

                    if result\_type == 'random' or result\_type == 'first free' or result\_type == 'copycat':

                        choice = twoCellsLeft(

*mode*, *nimBoard*, *initialNimBoard*, t\_posmoves, *playermoves*)

                        choice4.extend(choice)

                        return choice4

                del t\_posmoves[1]

*# if no winning move found, then computer plays according to its previous strategy*

    if *mode* == 'random':

        t\_choice = randomMoreThanFiveLeft(*nimBoard*, *initialNimBoard*)

        result\_type4 = 'random'

    elif *mode* == 'first free':

        t\_choice = firstfreeMoreThanFiveLeft(*nimBoard*, *initialNimBoard*)

        result\_type4 = 'first free'

    elif *mode* == 'copycat':

        t\_choice = copycatMoreThanFiveLeft(

*nimBoard*, *initialNimBoard*, *playermoves*)

        result\_type4 = 'copycat'

    choice4.extend(t\_choice)

    return choice4

ΕΞΗΓΗΣΗ ΚΩΔΙΚΑ

def fiveCellsLeft(*mode*, *nimBoard*, *initialNimBoard*, *possibleMoves*, *playermoves*):

    choice5 = []

*# picks every combination of 3 available cells and, if they are consecutive and non-diagonal,*

*# checks if they give a winning move*

    for i in range(0, 5):

        t\_posmoves = []

        t\_posmoves.append(*possibleMoves*[i])

        for j in range(0, 5):

            if i != j:

                t\_posmoves.append(*possibleMoves*[j])

                for m in range(0, 5):

                    checklist = []

                    if m != j and m != i:

                        t\_posmoves.append(*possibleMoves*[m])

                        threeCellsLeft(

*mode*, *nimBoard*, *initialNimBoard*, t\_posmoves, *playermoves*)

*# checks if the 3 cells are consecutive and not on the diagonal*

                        if result\_type == 'consecutive':

                            for k in range(0, 5):

                                if *possibleMoves*[k] not in t\_posmoves:

                                    checklist.append(*possibleMoves*[k])

                            twoCellsLeft(*mode*, *nimBoard*,

*initialNimBoard*, checklist, *playermoves*)

*# checks if the 3 cells that were previously chosen give a winning move*

                            if result\_type == 'random' or result\_type == 'first free' or result\_type == 'copycat':

                                choice = threeCellsLeft(

*mode*, *nimBoard*, *initialNimBoard*, t\_posmoves, *playermoves*)

                                choice5.extend(choice)

                                return choice5

                        del t\_posmoves[2]

                del t\_posmoves[1]

*# picks every cell individually, and examines if playing it will give a winning move. If yes, the selected cell is played*

    for i in range(0, 5):

        t\_posmoves = []

        t\_posmoves.append(*possibleMoves*[i])

        checklist = []

        for j in range(0, 5):

            if *possibleMoves*[j] not in t\_posmoves:

                checklist.append(*possibleMoves*[j])

        fourCellsLeft(*mode*, *nimBoard*, *initialNimBoard*, checklist, *playermoves*)

        if result\_type4 == 'random' or result\_type4 == 'first free' or result\_type4 == 'copycat':

            choice = *possibleMoves*[i]

            choice5.append(choice)

            return choice5

    return choice5

ΕΞΗΓΗΣΗ ΚΩΔΙΚΑ

def getCellwithNumNeighbours(*num*, *initialNimBoard*, *possibleMoves*):

    for i in range(len(*possibleMoves*)):

        count = getNumberofNeighbours(

*possibleMoves*[i], *initialNimBoard*, *possibleMoves*)

        if count == *num*:

            return *possibleMoves*[i]

    return False

ΕΞΗΓΗΣΗ ΚΩΔΙΚΑ

def getNumberofNeighbours(*cell*, *initialNimBoard*, *possibleMoves*):

    count = 0

    row1 = getRowCells(*initialNimBoard*, 1)

    col1 = getColumnCells(*initialNimBoard*, 1)

    rowN = getRowCells(*initialNimBoard*, N\*N - N + 1)

    colN = getColumnCells(*initialNimBoard*, N\*N)

*# cells of first row*

    if (*cell* in row1) and *cell* != N:

        if *cell*+1 in *possibleMoves*:

            count = count + 1

        if *cell*+N in *possibleMoves*:

            count = count + 1

        if *cell* != 1:

            if *cell*-1 in *possibleMoves*:

                count = count + 1

*# cells of first column*

    elif *cell* in col1:

        if *cell*+1 in *possibleMoves*:

            count = count + 1

        if *cell*-N in *possibleMoves*:

            count = count + 1

        if *cell* != (N\*N - N + 1):

            if *cell*+N in *possibleMoves*:

                count = count + 1

*# cells of last row*

    elif *cell* in rowN:

        if *cell*-1 in *possibleMoves*:

            count = count + 1

        if *cell*-N in *possibleMoves*:

            count = count + 1

        if *cell* != N\*N:

            if *cell*+1 in *possibleMoves*:

                count = count + 1

*# cells of last column*

    elif *cell* in colN:

        if *cell*-1 in *possibleMoves*:

            count = count + 1

        if *cell*+N in *possibleMoves*:

            count = count + 1

        if *cell* != N:

            if *cell*-N in *possibleMoves*:

                count = count + 1

*# other cells*

    else:

        if *cell*-1 in *possibleMoves*:

            count = count + 1

        if *cell*+1 in *possibleMoves*:

            count = count + 1

        if *cell*-N in *possibleMoves*:

            count = count + 1

        if *cell*+N in *possibleMoves*:

            count = count + 1

    return count

ΕΞΗΓΗΣΗ ΚΩΔΙΚΑ

def getCellNeighbours(*cell*, *initialNimBoard*):

    neighbours = []

    row1 = getRowCells(*initialNimBoard*, 1)

    col1 = getColumnCells(*initialNimBoard*, 1)

    rowN = getRowCells(*initialNimBoard*, N\*N - N + 1)

    colN = getColumnCells(*initialNimBoard*, N\*N)

*# cells of first row*

    if (*cell* in row1) and *cell* != N:

        neighbours.append(*cell*+1)

        neighbours.append(*cell*+N)

        if *cell* != 1:

            neighbours.append(*cell*-1)

*# cells of first column*

    elif *cell* in col1:

        neighbours.append(*cell*+1)

        neighbours.append(*cell*-N)

        if *cell* != (N\*N - N + 1):

            neighbours.append(*cell*+N)

*# cells of last row*

    elif *cell* in rowN:

        neighbours.append(*cell*-1)

        neighbours.append(*cell*-N)

        if *cell* != N\*N:

            neighbours.append(*cell*+1)

*# cells of last column*

    elif *cell* in colN:

        neighbours.append(*cell*-1)

        neighbours.append(*cell*+N)

        if *cell* != N:

            neighbours.append(*cell*-N)

*# other cells*

    else:

        neighbours.append(*cell*-1)

        neighbours.append(*cell*+1)

        neighbours.append(*cell*-N)

        neighbours.append(*cell*+N)

    return neighbours

ΕΞΗΓΗΣΗ ΚΩΔΙΚΑ

Αφου λοιπον αναλυσαμε τι γινεται στην περιπτωση που ο υπολογιστης δει ότι υπαρχουν 5 η λιγοτερα tiles ελευθερα στο ταμπλο, μπορουμε να αναλυσουμε τωρα τι γινεται για κάθε στρατιγικη του υπολογιστη.

* Για την Random

Καλείται η randomMoreThanFiveLeft :

def randomMoreThanFiveLeft(*nimBoard*, *initialNimBoard*):

    possibleMoves = getAvailableCells(*nimBoard*)

    choice = []

    tile = random.choice(possibleMoves)

    choice.append(tile)

*# randomly select a number between 1 and 3*

    num = random.randint(1, 3)

*#  select if the next tile will be in the same row or column*

    if random.randint(0, 1) == 0:

        where = 'row'

    else:

        where = 'column'

    if random.randint(0, 1) == 0:

        to\_where = 'up'

    else:

        to\_where = 'down'

    i = 1

    if (num > 1 and tile not in getDiagonalCells(*initialNimBoard*)):

        while i < num:

            flag = 0

*# check the where and to\_where and then check the board if the next tile move is valid.*

*# If it is valid then append it to the choice list if not then break*

            if where == 'row' and to\_where == 'up':

                next\_tile = tile - 1

                if next\_tile in possibleMoves and checkValidMove(*initialNimBoard*, choice + [next\_tile]):

                    tile = next\_tile

                    choice.append(tile)

                else:

*# if the move is not valid, change the to\_where variable and try again*

                    to\_where = 'down'

                    if num == 3 and i == 2:

                        tile = choice[0]

                    flag = 1

            if where == 'row' and to\_where == 'down':

                next\_tile = tile + 1

                if next\_tile in possibleMoves and checkValidMove(*initialNimBoard*, choice + [next\_tile]):

                    tile = next\_tile

                    choice.append(tile)

                else:

*# if the move is not valid, change the to\_where variable and try again*

                    to\_where = 'up'

                    i = i-1

                    if num == 3 and i == 2:

                        tile = choice[0]

*# if the value of flag is 1 then break the loop (to\_where is already changed)*

                    if flag == 1:

                        break

            if where == 'column' and to\_where == 'up':

                next\_tile = tile - int(math.sqrt(len(*nimBoard*)))

                if next\_tile in possibleMoves and checkValidMove(*initialNimBoard*, choice + [next\_tile]):

                    tile = next\_tile

                    choice.append(tile)

                else:

*# if the move is not valid, change the to\_where variable and try again*

                    to\_where = 'down'

                    if num == 3 and i == 2:

                        tile = choice[0]

                    flag = 1

            if where == 'column' and to\_where == 'down':

                next\_tile = tile + int(math.sqrt(len(*nimBoard*)))

                if next\_tile in possibleMoves and checkValidMove(*initialNimBoard*, choice + [next\_tile]):

                    tile = next\_tile

                    choice.append(tile)

                else:

*# if the move is not valid, change the to\_where variable and try again*

                    to\_where = 'up'

                    i = i-1

                    if num == 3 and i == 2:

                        tile = choice[0]

*# if the value of flag is 1 then break the loop (to\_where is already changed)*

                    if flag == 1:

                        break

            i = i+1

    return choice

Αρχιζουμε δημιουργοντας μια λιστα choice οπου στο τελος θα περιεχει τα κελια που επελεξε ο υπολογιστης να παιξει. Συνεχιζουμε επιλεγοντας ισοπιθανα ποσα κελια θα γραψει ο υπολογιστης, αμα θα είναι ανα στηλη η ανα γραμμη και την κατευθυνση (από δεξια προς αριστερα η το αντιθετο για την γραμμη και από πανω προς κατω η το αντιθετο για την στηλη). Αφου εχουν γινει ολες αυτές οι τυχαιες επιλογες και τις εχουμε κρατησει παμε στην main λειτουργια της συναρτησης. Αμα το τυχαιο νουμερο των tiles είναι παραπανω από 1 (αμα είναι 1 απλα το γραφει, και στην λιστα είναι μονο αυτό το tile) και το πρωτο tile δεν είναι στην διαγωνιο (από την εκφωνηση αμα γραψει καποιος το πρωτο tile στην διαγωνιο δεν τον αφηνει να γραψει καποιο άλλο tile) τοτε μπαινουμε στην if. Εκει υπαρχουν 4 περιπτωσεις οπου γινονται τα ιδια πραγματα σχεδον. Για να μπει σε κάθε περιπτωση είναι αναλογως με τις τυχαιες τιμες που πηραν οι μεταβλητες στην αρχη της συναρτησεις (για κατεύθυνση και στηλη η γραμμη). Εχουμε φτιαξει την συναρτηση ετσι ώστε να προσπαθει να κανει την κινηση που να εχει μεγεθος οσο πιο κοντα στην τυχαια τιμη της μεταβλητης num. Δηλαδη αμα εχει επιλεξει να παει ανα γραμμη και εχει επιλεξει να γραψει 3 tiles και την κατευθηνση να είναι προς τα αριστερα το προγραμμα μας είναι ικανο αμα συναντησει για οποιοδιποτε λογο εμποδιο να αλλαξει την κατευθηνση (να την κανει προς τα δεξια) για να δει αμα μπορει να γραψει τα tile που λειπουν. Ένα παραδειγμα φαινεται από κατω οπου είναι η σειρα του υπολογιστη τωρα (το πρασινο χρώμα). Αμα το τυχαιο tile που επιλεχτει στην αρχη είναι το 4, να θελει τυχαια να παει ανα γραμμη, προς αριστερα και να γραψει 3 tiles τοτε δεν θα τα καταφερει. Αυτό θα γινει γιατι γραφοντας στο 4 και μετα στο 3 όπως βλεπουμε το 2 δεν είναι διαθεσιμο. Αρα κανονικα θα τελειωνε εκει. Με την αναδρομη που εχουμε κανει και θα εξηγησουμε στον κωδικα ο υπολογιστης αφου βρει εμποδιο στο tile 2 θα γυρισει να κοιταξει και από την άλλη μερια (δεξια εδώ) του αρχικου tile (4) στην προσπαθεια να γραψει 3 tile. Τωρα αρα εντι να εγραφε 4,3 και να ολοκληρωνε την σειρα του, θα γραψει 4,3,5 και θα ολοκληρωσει την σιερα του εχοντας τωρα μια νομιμη κινηση μεγεθους num με το αρχικο tile και ανα γραμμη.

A picture containing text, screenshot, font, number

Description automatically generated

Αν επιστρεψουμε στον κωδικα θα δουμε υπαρχει μια μορφη αναδρομικότητας. Για να γινεται αυτό που περιγραψαμε παραπανω αμα εχουμε πχ ανα γραμμη, στον κωδικα θα κοιταει από την κατευθυνση που εχει επιλεχτεί αρχικα και αμα δεν είναι επιτρεπτη κινηση αυτό το tile τοτε θα εχουμε δυο σεναρια. Αμα είναι το το δευτερο tile που ειμαστε τοτε απλα θα αλλαζει η κατευθυνση και θα κοιτα από την άλλη (αφου πρωτα αυξησει ένα flag για να ξερουμε ότι καναμε ηδη μια αλλαγη και μην κολλησουμε σε loop). Αμα είναι το τριτο tile θα πρεπει να αλλαξουμε εκτος από την κατευθυνση και το tile το οποιο κοιταμε από διπλα. Πχ αμα όπως ειπα ειχαμε το 4,3 και θελουμε να αλλαξουμε κατευθυνση δεν αρκει μονο αυτό γιατι μετα θα κοιταμε από το 3 δεξια και δεν θα εχει νοημα. Θα πρεπει να κοιταμε από το αρχικο tile από την άλλη. Για αυτό και εχουμε υπο συνθηκη το tile = choice[0] οπου αλλαζει το tile που κοιταει δεξια η αριστερα ο υπολογιστης. Αμα όμως ειμαστε στην περιπτωση που του to\_where= down τοτε δεν μπορουμε απλα να αλλαξουμε την μεταβλητη αφου ολος ο κωδικας για to\_where= up εχει προσπεραστει ηδη. Για αυτό μειωνουμε το i που είναι ο iterator της while για να ξανατρεξει την while με ιδιο iterator (εχοντας και εδώ την flag και αλλαζοντας την με την αλλαγη του iterator). Ετσι τωρα εχουμε φτιαξει ολη την συναρτηση για την επιλογη tile από τον υπολογιστη με την στρατιγικη random.

* Για την First Fit

Καλείται η firstfreeMoreThanFiveLeft:

def firstfreeMoreThanFiveLeft(*nimBoard*, *initialNimBoard*):

*# This function returns the first free move from the list of*

*# available moves.*

*# return the first available tile to start the move*

    possibleMoves = getAvailableCells(*nimBoard*)

    choice = []

    tile = possibleMoves[0]

    choice.append(tile)

*# randomly select a number between 1 and 3*

    num = random.randint(1, 3)

*#  select if the next tile will be in the same row or column*

    if random.randint(0, 1) == 0:

        where = 'row'

    else:

        where = 'column'

    to\_where = 'down'

    if (num > 1 and tile not in getDiagonalCells(*initialNimBoard*)):

        for i in range(1, num):

*# check if the next move is valid*

            if where == 'row' and to\_where == 'down':

                next\_tile = tile + 1

                if next\_tile in possibleMoves and checkValidMove(*initialNimBoard*, choice + [next\_tile]):

                    tile = next\_tile

                    choice.append(tile)

                else:

                    break

            elif where == 'column' and to\_where == 'down':

                next\_tile = tile + int(math.sqrt(len(*nimBoard*)))

                if next\_tile in possibleMoves and checkValidMove(*initialNimBoard*, choice + [next\_tile]):

                    tile = next\_tile

                    choice.append(tile)

                else:

                    break

    return choice

Όπως βλεπουμε παλι γινεται τυχαια επιλογη για τον αριθμο των tiles που θα περιεχει η κινηση καθως και για το αμα η κινηση θα είναι ανα στηλη η ανα γραμμη. Αυτό που αλλαζει με σχεση με πριν είναι ότι αυτή την φορα το πρωτο tile δεν είναι τυχαιο αλλα το πρωτο ελευθερο και η κατευθυνση είναι προς τα δεξια ανα γραμμη και προς τα κατω ανα στηλη (αλλιως θα επεφτε πανω σε πιασμενα tile). Ο κωδικας είναι πιο απλος από της random μιας και είναι σταθερο το to\_where και αρα υπαρχουν 2 περιπτωσεις. Αμα τυχη να θελει να παει ανα γραμμη τοτε ελεγχει αμα το στοιχειο διπλα είναι ελευθερο και νομιμη κινηση και αμα είναι τοτε το βαζει στην λιστα με τα tiles, αμα δεν είναι τοτε ολοκληρωνει την σειρα εχοντας επιλεξει μονο το πρωτο κενο tile που βρηκε στο ταμπλο. Το ιδιο συμβαινει και αμα παει ανα στηλη.

* Για την Copycat

def copycatMoreThanFiveLeft(*nimBoard*, *initialNimBoard*, *playermoves*):

*# This function returns the copycat move from what the player played. If the computer goes first or the player move can be copied then the computer chooses either a random move with getComputerMove\_random or a first free move with getComputerMove\_firstfit.*

*# If the players move cant be copied then the move randomly chosen must contain as many tiles as the player move or less, but not more.*

*# return the first available tile to start the move*

    possibleMoves = getAvailableCells(*nimBoard*)

    init\_possibleMoves = getAvailableCells(*initialNimBoard*)

    choice = []

*# if the computer goes first*

    if len(possibleMoves) == len(init\_possibleMoves):

        if random.randint(0, 1) == 0:

            choice = getComputerMove\_random(*nimBoard*, *initialNimBoard*)

        else:

            choice = getComputerMove\_firstfit(*nimBoard*, *initialNimBoard*)

*# if the player goes first*

    else:

*# find the move the player performed last and copy it by doing the same move symmetrically by the diagonal, The players last move is in the playermoves list*

*# find the tile that is symmetric to the player move*

*# if the players move is on the diagonal then the computer chooses the tile in the diagonal that is symmetric in the center of the diagonal.*

*# If the cell is not available then the computer chooses a random cell in the diagonal.*

        if *playermoves*[0] in getDiagonalCells(*initialNimBoard*):

            diagon = getDiagonalCells(*initialNimBoard*)

            for i in range(len(diagon)):

                if *playermoves*[0] == diagon[i]:

                    if diagon[len(diagon)-1-i] in possibleMoves:

choice.append(diagon[len(diagon)-1-i])

                    else:

*# check first if any of the elements of the diagonal are possible moves*

*# if not then choose a random move*

                        if not any(x in possibleMoves for x in diagon):

                            choice.append(possibleMoves[0])

                        else:

*# choice is equal to a random empty cell in the diagonal*

                            while True:

                                choice = []

                                choice.append(random.choice(diagon))

                                if choice[0] in possibleMoves:

                                    break

*# if choice is not empty then break*

                if choice:

                    break

*# if the players move is not on the diagonal then the computer chooses the tile that is symmetric to the player move,the symmetry is determined by the diagonal*

        else:

            flag = 0

            for i in range(len(*playermoves*)):

                row = (*playermoves*[i] - 1) // N  *# 0-indexed row*

                col = (*playermoves*[i] - 1) % N   *# 0-indexed column*

*# find the tile that is symmetric to the player move*

                symmetric = col \* N + row + 1

                choice.append(symmetric)

                if choice[i] not in possibleMoves:

                    flag = 1

            if flag == 1:

*# even if one tile of move is not valid, choose a random move*

                if random.randint(0, 1) == 0:

                    choice = getComputerMove\_random(*nimBoard*, *initialNimBoard*)

                    while len(choice) > len(*playermoves*):

                        choice = getComputerMove\_random(

*nimBoard*, *initialNimBoard*)

                else:

                    choice = getComputerMove\_firstfit(

*nimBoard*, *initialNimBoard*)

                    while len(choice) > len(*playermoves*):

                        choice = getComputerMove\_firstfit(

*nimBoard*, *initialNimBoard*)

    return choice

Αρχικα όπως και στις προηγουμενες συναρτησεις οριζουμε στην αρχη την λιστα που θα εχει τα tiles της κινησης και βρισκουμε ποσες ελευθερες θεσεις υπαρχουν στο ταμπλο τωρα. Αμα οι ελευθερες θεσεις τωρα είναι ισες με αυτές του αρχικου ταμπλο (είναι η πρωτη κινηση του παιχνιδιου) τοτε ο υπολογιστης διαλεγει τυχαια μια από τις υπολοιπες δυο σταρτιγικες για να εκτελεσει για την πρώτη κινηση του. Αυτό γινεται γιατι δεν εχει να αντιγραψει καποια κινηση αμα δεν εχει παιξει καμια ακομη ο παικτης. Αμα εχει παιξει ο παικτης εστω και μια φορα τοτε παλι εχουμε υποπεριτπωσεις. Αμα ο παικτης εχει παιξει κελι διαγωνιου τοτε ο υπολογιστης προσπαθει να παιξει το συμμετρικο κελι της διαγωνιου από το κεντρο του ταμπλο. Αυτό γινεται με την choice.append(diagon[len(diagon)-1-i]). Οπου diagon είναι η λιστα με όλα τα tiles της διαγωνιου. Αμα όμως αυτό το tile της διαγωνιου δεν είναι διαθεσιμο τοτε βαζει σε ένα άλλο τυχαιο tile της διαγωνιου. Αμα δεν εχει άλλη tile ελευθερο η διαγωνιος τοτε τότε επιλέγεται το πρώτο μη-κενό κελί στο ταμπλο. Αυτό γινεται με την if not any(x in possibleMoves for x in diagon) και μετα την αντιστοιχη else (αμα δεν υπαρχε άλλο tile στην διαγωνιο). Τωρα για το κυριο κομματι της συναρτησης αμα ο παικτης κανει μια κινηση ο υπολογιστης θα πρεπει να την αντιγραψει και να κανει την συμμετρικη αυτης με βαση την διαγωνιο. Αφου εχουμε την playermoves (την κινηση του παικτη που θελουμε να αντιγραψει ο υπολογιστης) θα πρεπει όλα τα συμμετρικα tile των αρχικων να είναι ελευθερα. Αμα είναι απλα τα προσθετει στην λιστα choice και τα επιλεγει. Αμα εστω και ένα δεν είναι τοτε η μεταβλητη flag γινεται 1 και τοτε ο υπολογιστης επιλεγει ισοπιθανα αμα θα εκτελεσει την κινηση του με την random η την first fit. Και στις δυο περιπτωσεις θα πρεπει το μεγεθος της κινησης να είναι το πολύ ισο με το μεγεθος της κινησης του παικτη που προσπαθουμε να αντιγραψουμε. Αρα για οποια από τις δυο στρατιγικες παει να κανει θυα πρεπει να κοιταει να είναι μικροτερο το αποτελεσμα από την κινηση του παικτη. Όταν εν τελη βρει μια τετοια κινηση τοτε την περναει στη choice και την εκτελει.

Αφου εξηγησαμε ολες τις συναρτησεις και ολο το προγραμμα ας παμε να δουμε πως δουλευεις την πραξη. Να επισημανουμε ότι για κάθε κινηση του υπολογιστη πριν τυπωθει το ταμπλο με την κινηση του τυπωνεται η κινηση σαν λιστα ετσι ώστε να μην χανουμε τι εκανε ο υπολογιστης στην σειρα του.

* Για την Random

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Στο παρακάτω προσπαθήσαμε όπως βλέπουμε να δώσουμε tile της διαγωνίου σε κάποια άλλη από την πρώτη θέση και σωστά το πρόγραμμα μας σταμάτησε. Επίσης είδαμε πως γράφοντας το αρχικό tile δυο φορές και μετα το από διπλά τερματίσαμε την σειρά μας έχοντας διαλέξει αυτά τα δυο tile.

A screenshot of a computer program

Description automatically generated with medium confidence

Εδώ βλέπουμε πως δεν έχει σχέση με ποια σειρά δίνουμε τα tiles αρκεί να είναι στο τέλος σε σειρά

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

Εδώ βλέπουμε ότι αν δώσουμε κάτι εκτός από ακέραιο τότε κρατάει τα tile που έχουμε δώσει μέχρι στιγμής και λήγει την σειρά μας.

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with low confidence

* Για την First Fit

A screenshot of a computer game

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

* Για την Copycat

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

Όπως βλέπουμε άμα τοποθέτηση ο παίκτης στην διαγώνιο ο υπολογιστής θα απαντήσει το αντίστοιχο συμμετρικό ως προς το κέντρο tile της διαγωνίου.

A screenshot of a computer

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with low confidence

Όπως βλέπουμε άμα τα έστω και ένα από τα συμμετρικά κελιά της κίνησης του παίκτη είναι πιασμένα τότε ο υπολογιστής κάνει μια random κίνηση.

A screenshot of a computer

Description automatically generated with low confidence

Και εδώ άμα δεν μπορεί να βρει το συμμετρικό της διαγωνίου και δεν υπάρχουν αλλά κενά στην διαγώνιο τότε ο υπολογιστής επιλεγεί το πρώτο κενό tile στο ταμπλό σαν την κίνηση του.

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Και εδώ αφού δίνουμε στον υπολογιστή παρτίδα νίκης επιλεγεί να παρατήσει την στρατηγική του και να παίξει για να νικήσει

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer game

Description automatically generated with medium confidence