**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 όσον αναφορά τις συναρτήσεις που δινόντουσαν έτοιμες στο template.

Αλλάξαμε την isBoardFull και ο νέος της κώδικας είναι όπως φαίνεται από κάτω.

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

Επιστρέφει δηλαδή True άμα όλα τα tiles του πίνακα έχουν γραφτεί από πάνω με το γράμμα G η R. Αυτό δείχνει ότι κάποιος από τους δυο παίκτες το έχει κρατήσει και αρά άμα είναι όλα κρατάμε το ταμπλό είναι γεμάτο.

Δεν χρησιμοποιήσαμε την getRowAndColumn μιας και είχαμε φτάσει σε σημείο στο πρόγραμμα μας, όταν δόθηκε αυτή η συνάρτηση, που την είχαμε υλοποιήσει αυτή την λειτουργία με άλλες συναρτήσεις.

Τώρα θα αρχίσει η ανάλυση του κώδικα με την σειρά που εκτελείται ο κώδικας όταν τρέχουμε το πρόγραμμα.

Στην αρχή του προγράμματος αφού εκτυπωθούν οι κανόνες και ο τρόπος που παίζεται το παιχνίδι στην οθόνη το πρόγραμμα μας τρέχει τις παρακάτω γραμμές του κώδικα πριν αρχίσουν τα input του χρήστη.

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()

Αρχικά δηλώνουμε τον μέγιστο αριθμό κινήσεων (πόσα κουτιά μπορεί να παίξει ο υπολογιστής η ο παίκτης σε μια σειρά) και ένα flag που χρησιμοποιείται για να ξέρουμε ποτέ ο παίκτης επιλέξει να αρχίσει το παιχνίδι από την αρχή. Στη συνέχεια το Ν που είναι το μέγεθος του board αλλά και το ίδιο το board oπως θα το τυπώνουμε αρχικοποιείται. Έχουμε ορίσει όπως φαίνεται άλλο ένα board που το ονομάσαμε initialNimBoard το οποίο θα έχει την μορφή του αρχικού board καθ’ όλη την διάρκεια του παιχνιδιού (μιας και ο πίνακας nimBoard αλλάζει κατά την εκτέλεση του προγράμματος). Υστέρα οι μεταβλητές που με βάση τις εισαγωγές του χρήστη στα αρχικά μηνύματα κρατάνε ποιο γράμμα θα έχει ο παίκτης και ποιο ο υπολογιστής καθώς και ποιος θα παίξει πρώτος που με βάση την whoGoesFirst είναι 50-50. Όσον αναφορά τώρα τον πίνακα diagonalCells και την computerStrategy. Ο diagonalCells είναι ένας πίνακας που με την βοήθεια της συνάρτησης getDiagonalCells ανεξαρτήτως του μεγέθους του board βρίσκει και αποθηκεύει τα κελιά που είναι στην διαγώνιο του board, όπως φαίνεται παρακάτω και στην getDiagonalCells. Τέλος η μεταβλητή computerStrategy επιστρέφει το όνομα της στρατηγικής που θα χρησιμοποιήσει ο υπολογιστής που το παίρνει από την εισαγωγή του χρήστη στην αρχή του προγράμματος.

*# 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

Συνεχίζουμε τώρα αφού ο χρήστης έχει υποβάλει τις προτιμήσεις του (μέγεθος του board, στρατηγική υπολογιστή, και χρώμα που προτιμάει να έχει).

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'

Αρχικά έχουμε μια λίστα που αρχικά είναι κενή (playermoves). Αυτή η λίστα θα χρησιμοποιηθεί για την στρατηγική copycat οπού εκεί μέσα θα αποθηκεύεται η κίνηση που έκανε ο παίκτης κάθε φορά στην σειρά του.

Έστω ότι είναι η σειρά του παίκτη τότε καλείται αρχικά η συνάρτηση getPlayerMove όπως φαίνεται ο κώδικας της παρακάτω.

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(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(choice):

print(bcolors.GREEN +

'Move executed succesfully!' + bcolors.ENDC)

return choice

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 συνάρτηση αυτή μαζί και με κάποιες άλλες που καλούνται μέσα από αυτή είναι υπεύθυνη για να παίρνει τις κινήσεις του παίκτη και να τις ελέγχει άμα είναι νόμιμες. Άμα δεν είναι τότε τυπώνονται τα αντίστοιχα λάθη και είτε του δίνεται η εύκαιρα να προσπαθήσει ξανά είτε του ολοκληρώνεται η σειρά και κρατιούνται οι επιλογές που έχει κάνει μέχρι τότε (εξαρτάται το λάθος). Ας αρχίσουμε την ανάλυση της συνάρτησης αυτής. Αρχικά καλούμε την συνάρτηση getAvailableCells που παίρνει σαν όρισμα το board κάθε φορά. Αυτή η συνάρτηση ουσιαστικά επιστρέφει κάθε φορά που την καλείς μια λίστα με όλα τα tiles του board που είναι ελεύθερα και μπορεί να γράψει πάνω είτε ο παίκτης είτε ο υπολογιστής. Στην συνέχεια δημιουργείται μια λίστα που ονομάσαμε choice που θα είναι ουσιαστικά η λίστα που επιστρέφει η συνάρτηση getPlayerMove και θα περιέχει τα tiles της κίνησης του παίκτη. Τώρα όπως βλέπουμε στην συνέχεια όλη η υπόλοιπη συνάρτηση είναι μέσα σε μια while και μέσα της υπάρχουν και άλλες παρόμοιες while. Αυτό γίνεται γιατί έχουμε φτιάξει το πρόγραμμα έτσι ώστε αναλόγως το θέμα που παρουσιάζεται με τις επιλογές του χρήστη να του δίνεται σε κάποιες περιπτώσεις η επιλογή να επαναλάβει όλο την κίνηση η κάποια μέρη της, έτσι θα πρέπει να είναι μέσα σε μια while loop. Έτσι μέσα σε αυτή την πρώτη while έχουμε την επιλογή του πρώτου tile της κίνηση του παίκτη. Αναγκάζουμε τον παίκτη να δώσει τουλάχιστον ένα tile για την κίνηση του οπότε αυτό το κομμάτι είναι απαραίτητο. Άμα ο χρήστης βάλει κάποιο μη αποδεκτό input (κάτι διαφορετικό από έναν ακέραιο που αντιπροσωπεύει τον αριθμό του tile από το board η εάν tile που δεν μπορεί να επιλέξει γιατί είναι κατειλημμένο, δεν είναι δηλαδή στην λίστα availableMoves) τότε τυπώνεται πάλι το μήνυμα να εισάγει το πρώτο tile. Επίσης άμα η επιλογή του είναι κάποια tile της διαγωνίου τότε ολοκληρώνεται η σειρά του κρατώντας μόνο αυτό το tile. Τώρα για το δεύτερο tile υπάρχουν και άλλες προϋποθέσεις που πρέπει να τηρούνται. Αρχικά πρέπει πάλι να είναι κάποιος ακέραιος από την λίστα availableMoves αλλά επίσης πρέπει να είναι νόμιμη η κίνηση που αποτελεί το πρώτο ταle και το δεύτερο μαζί. Αυτό τον έλεγχο τον κάνει η συνάρτηση checkValidMove που φαίνεται από κάτω.

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 είδαμε πριν η συνάρτηση καλείται με όρισμα το αρχικό board μιας και θέλουμε να δούμε άμα είναι νόμιμη η κίνηση από άποψη θέσης των tile (έχουμε ήδη ελέγξει άμα απασχολείται ήδη το tile) και η λίστα choice που περιέχει τα ήδη περασμένα tiles της κίνησης του παίκτη μαζί με το πιο πρόσφατο tile που θέλει να χρησιμοποιήσει ο χρήστης. Τώρα μέσα στην συνάρτηση checkValidMove έχουμε δυο περιπτώσεις αναλόγως άμα ελέγχουμε το δεύτερο η το τρίτο tile της κίνησης του παίκτη. Άμα είναι το δεύτερο τα πράγματα είναι απλά κοιτάμε άμα το προηγούμενο tile (στην περίπτωση αυτή το πρώτο) είναι στην ιδιά γραμμή η στην ιδιά στήλη (με τις συναρτήσεις getRowCells, getColumnCells που επιστρέφουν σαν λίστα τα tiles που είναι στην γραμμή η στήλη με το αρχικό tile) με αυτό που θέλουμε να επιλέξουμε. Άμα είναι η συνάρτηση επιστρέφει αληθής και το tile προστίθεται στην λίστα με τα tiles της κίνησης του παίκτη, άμα όχι τότε τυπώνεται ένα μήνυμα που του λέει ότι το tile αυτό είναι λάθος και να προσπαθήσει ξανά. Πριν πάμε να δούμε πως ελέγχεται άμα δώσουμε το τρίτο tile ας σημειώσουμε πως έχουμε φτιάξει το πρόγραμμα μας έτσι ώστε να τερματίζει ο παίκτης την σειρά του, κρατώντας τις επιλογές των tiles που έχει κάνει μέχρι τώρα, με παραπάνω από έναν τρόπους. Ο ένας τρόπος είναι να δώσει κάτι άλλο από αριθμό (κάποιο γράμμα) και ο άλλος είναι να δώσει δυο tiles συνεχόμενα ίδια . Πχ με την εισαγωγή 2,3,3 ο παίκτης τερματίζει την σειρά του κρατώντας σαν κίνηση το 2,3. Αξίζει να σημειωθεί πως άμα ο παίκτης δώσει σαν είσοδο το 2,2,3 τότε το πρόγραμμα μας θα καταλάβει ότι ο χρήστης έχει δώσει δυο ιδιά στην και θα κρατήσει μόνο το 1 κάνοντας την κίνηση να είναι το 2,3. Αφού το ξεκαθαρίσαμε πάμε να δούμε τον τρόπο ελέγχου του τρίτου tile. Βλέπουμε ότι ελέγχουμε άμα το τρίτο tile είναι ίδιο με το δεύτερο (άμα είναι τότε ο έλεγχος είναι ιδίου στυλ με τον έλεγχο που περιγράψαμε πριν). Άμα δεν είναι ίδια λοιπόν πάμε να δούμε τις προϋποθέσεις. Άμα θέλουμε να είναι κατά γραμμή θα πρέπει να είναι στην ίδια γραμμή τόσο με το πρώτο όσο και με το δεύτερο tile. Για αυτό κάνουμε έλεγχο με την getRowCells και για τα δυο αυτά tile. Ενώ άμα πάμε κατά στήλη θα πρέπει πάλι να είναι στην ίδια στήλη τόσο με το πρώτο όσο και με το δεύτερο tile. Άμα δεν υπήρχαν αυτές οι προϋποθέσεις και ελέγχαμε ας πούμε μόνο με σχέση το δεύτερο tile τότε ο παίκτης θα μπορούσε να ξεγελάσει το σύστημα και να είχε την δυνατότητα να σχεδιάσει ένα Γ με την κίνηση του, το οποίο απαγορεύεται. Ας επιστρέψουμε τώρα στην getPlayerMove. Άμα περάσει λοιπόν τον έλεγχο το δεύτερο tile τότε το βάζουμε στην λίστα με τα tiles της κίνησης του παίκτη. Προχωράμε τώρα στο τρίτο tile. Γίνονται πάλι οι ίδιοι έλεγχοι (άμα είναι ακέραιος και είναι available άμα επιστρέφει true η checkValidMove) και άμα γίνουν επιτυχώς πάλι βάζουμε το τρίτο tile sτην λίστα. Τώρα όμως πρέπει κάπως να γίνει ο έλεγχος για αυτό που περιγράψαμε πάνω (άμα δυο συνεχόμενα ταle είναι ίδια να κρατάει μόνο το κομμάτι που είναι διαφορετικά και να ελέγχει άμα είναι νόμιμη πάλι). Αυτό γίνεται με την checkValidMove2 όπου φαίνεται παρακάτω.

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

Έτσι αφού τελειώσαμε με την getPlayerMove (δηλαδή όλους τους ελέγχους για την εγκυρότητα των tiles που θέλει ο παίκτης να έχει στην κίνηση του) προχωράμε παρακάτω στο main πρόγραμμα 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'

Αφού λοιπόν καλέσαμε και εκτελέσαμε την getPlayerMove βλέπουμε ότι υπάρχει μια while από κάτω η οποία εκτελεί την ίδια εντολή που εξηγήσαμε συνεχόμενα μέχρι να μην βγάλει error. Αυτό το κάνουμε για να μπορούμε να ελέγχουμε για άμα είναι συνεχόμεναί τα tiles που έδωσε ο χρήστης συνεχόμενα και όχι μόνο μια φορά (άμα κάνει λάθος το πρόγραμμα του ζητάει να επαναλάβει όλη την κίνηση από την αρχή αρά πρέπει κάπως να ελέγχουμε και μετα την πρώτη εισαγωγή του χρήστη άμα είναι συνεχόμενα τα tiles). Μετα από αυτό τον έλεγχο (όταν ο χρήστης τελειώσει με το input της κίνησης του και είναι νόμιμη) βάζουμε αυτή την κίνηση στον πίνακα που περιγράψαμε στην αρχή, που κρατάει την τελευταία κίνηση του παίκτη (για την copycat). Στην συνέχεια καλούμε την συνάρτηση makeMove η οποία είναι υπεύθυνη για την εισαγωγή της κίνησης στο ταμπλό, όπως φαίνεται στον κώδικα της από κάτω.

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 σημειωθεί ότι εδώ γίνεται και η αύξηση της μεταβλητής counter όπου μας δείχνει κάθε φορά πόσα tiles του ταμπλό είναι πιασμένα.

Αφού λοιπόν εκτυπώσουμε την κίνηση στο ταμπλό προχωράμε στην εκτύπωση του ταμπλό στην οθόνη με την drawNimPalette. Τέλος έχουμε μια συνθήκη η οποία άμα δει ότι μετα από την ολοκλήρωση της κίνησης το ταμπλό είναι γεμάτο, τυπώνει ένα μήνυμα νίκης για τον παίκτη.

Στην περίπτωση που είναι η σειρά του υπολογιστή εκτελείτε το κομμάτι κώδικα που φαίνεται παρακάτω.

        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 βλέπουμε το κομμάτι του υπολογιστή είναι αρκετά παρόμοιο με το τι γίνεται όταν παίζει ο παίκτης. Η μονή διαφορά είναι το μήνυμα νίκης που τώρα έγινε ήττας για τον παίκτη, το else στο τέλος ,που είναι για την αναδρομική εκτέλεση του κώδικα και την εναλλαγή σειράς συνέχεια, και το πιο βασικό η κλήση της getComputerMove. Αυτό λοιπόν είναι το κομμάτι που πρέπει να αναλύσουμε.

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 παραπάνω κώδικα της getComputerMove έχουμε 3 υποπεριπτώσεις (μια για κάθε στρατηγική του υπολογιστή). Αναλόγως λοιπόν με ποια στρατηγική έχει επιλεχτεί στην αρχή της εκτέλεσης του προγράμματος καλούνται διαφορετικές συναρτήσεις.

Με την κλήση της getComputerMove εκτελείται παρόμοιος κώδικας για όλες τις στρατηγικές:

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 μόνη διαφορά μεταξύ των συναρτήσεων αυτών είναι ότι καλούν αναλόγως μια άλλη στρατηγική και ότι στην copycat έχουμε και την λίστα με την προηγουμένη κίνηση του παίκτη.

Στην αρχή έχουμε την λίστα choice όπου είναι η λίστα που έχουμε να επιστρέφει την κίνηση του υπολογιστή. Το σημαντικό κομμάτι είναι όμως οι έλεγχοι που γίνονται για το πόσα ελευθέρα tile απομένουν στο ταμπλό. Άμα αυτά τα tiles είναι παραπάνω από 5 ο υπολογιστής παίζει κανονικά την επιλεγμένη στρατηγική (random). Άμα όμως τα ελευθέρα tiles είναι λιγότερα από 5 τότε ο υπολογιστής εκτός από την στρατηγική του παίζει για να νικήσει άμα είναι εφικτό. Αυτό γίνεται με την τις συναρτήσεις 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 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*)

            elif *mode* == 'first free':

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

            elif *mode* == 'copycat':

                t\_choice = copycatMoreThanFiveLeft(

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

            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.extend(*possibleMoves*)

        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':

                        choice4.extend(t\_posmoves)

                        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':

                                choice5.extend(t\_posmoves)

                                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':

            choice5.append(t\_posmoves[0])

            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

    neighbours = getCellNeighbours(*cell*, *initialNimBoard*)

    for i in range(0, len(neighbours)):

        if neighbours[i] 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

Αφού λοιπόν αναλύσαμε τι γίνεται στην περίπτωση που ο υπολογιστής δει ότι υπάρχουν 5 η λιγότερα tiles ελευθέρα στο ταμπλό, μπορούμε να αναλύσουμε τώρα τι γίνεται για κάθε στρατηγική του υπολογιστή.

* 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

Αρχίζουμε δημιουργώντας μια λίστα choice όπου στο τέλος θα περιέχει τα κελιά που επέλεξε ο υπολογιστής να παίξει. Συνεχίζουμε επιλέγοντας ισοπίθανα πόσα κελιά θα γράψει ο υπολογιστής, άμα θα είναι ανά στήλη η ανά γραμμή και την κατεύθυνση (από δεξιά προς αριστερά η το αντίθετο για την γραμμή και από πάνω προς κάτω η το αντίθετο για την στήλη). Αφού έχουν γίνει όλες αυτές οι τυχαίες επιλογές και τις έχουμε κρατήσει πάμε στην main λειτουργία της συνάρτησης. Άμα το τυχαίο νούμερο των tiles είναι παραπάνω από 1 (άμα είναι 1 απλά το γραφεί, και στην λίστα είναι μόνο αυτό το tile) και το πρώτο tile δεν είναι στην διαγώνιο (από την εκφώνηση άμα γράψει κάποιος το πρώτο tile στην διαγώνιο δεν τον αφήνει να γράψει κάποιο άλλο tile) τότε μπαίνουμε στην if. Εκεί υπάρχουν 4 περιπτώσεις όπου γίνονται τα ίδια πράγματα σχεδόν. Για να μπει σε κάθε περίπτωση είναι αναλόγως με τις τυχαίες τιμές που πήραν οι μεταβλητές στην αρχή της συναρτήσεις (για κατεύθυνση και στήλη η γραμμή). Έχουμε φτιάξει την συνάρτηση έτσι ώστε να προσπαθεί να κάνει την κίνηση που να έχει μέγεθος όσο πιο κοντά στην τυχαία τιμή της μεταβλητής num. Δηλαδή άμα έχει επιλέξει να πάει ανά γραμμή και έχει επιλέξει να γράψει 3 tiles και την κατεύθυνση να είναι προς τα αριστερά το πρόγραμμα μας είναι ικανό άμα συναντήσει για οποιοδήποτε λόγο εμπόδιο να αλλάξει την κατεύθυνση (να την κάνει προς τα δεξιά) για να δει άμα μπορεί να γράψει τα tile που λείπουν. Ένα παράδειγμα φαίνεται από κάτω όπου είναι η σειρά του υπολογιστή τώρα (το πράσινο χρώμα). Άμα το τυχαίο tile που επιλεχτεί στην αρχή είναι το 4, να θέλει τυχαία να πάει ανά γραμμή, προς αριστερά και να γράψει 3 tiles τότε δεν θα τα καταφέρει. Αυτό θα γίνει γιατί γράφοντας στο 4 και μετα στο 3 όπως βλέπουμε το 2 δεν είναι διαθέσιμο. Αρά κανονικά θα τέλειωνε εκεί. Με την αναδρομή που έχουμε κάνει και θα εξηγήσουμε στον κώδικα ο υπολογιστής αφού βρει εμπόδιο στο tile 2 θα γυρίσει να κοιτάξει και από την άλλη μεριά (δεξιά εδώ) του αρχικού tile (4) στην προσπάθεια να γράψει 3 tile. Τώρα αρά αντί να έγραφε 4,3 και να ολοκλήρωνε την σειρά του, θα γράψει 4,3,5 και θα ολοκληρώσει την σειρά του έχοντας τώρα μια νόμιμη κίνηση μεγέθους num με το αρχικό tile και ανά γραμμή.

A picture containing text, screenshot, font, number

Description automatically generated

Αν επιστρέψουμε στον κώδικα θα δούμε υπάρχει μια μορφή αναδρομικότητας. Για να γίνεται αυτό που περιγράψαμε παραπάνω άμα έχουμε πχ ανά γραμμή, στον κώδικα θα κοιτάει από την κατεύθυνση που έχει επιλεχτεί αρχικά και άμα δεν είναι επιτρεπτή κίνηση αυτό το tile τότε θα έχουμε δυο σενάρια. Άμα είναι το δεύτερο tile που είμαστε τότε απλά θα αλλάζει η κατεύθυνση και θα κοιτά από την άλλη (αφού πρώτα αυξήσει ένα flag για να ξέρουμε ότι κάναμε ήδη μια αλλαγή και μην κολλήσουμε σε loop). Άμα είναι το τρίτο tile θα πρέπει να αλλάξουμε εκτός από την κατεύθυνση και το tile το οποίο κοιτάμε από διπλά. Πχ άμα όπως είπα είχαμε το 4,3 και θέλουμε να αλλάξουμε κατεύθυνση δεν αρκεί μόνο αυτό γιατί μετα θα κοιτάμε από το 3 δεξιά και δεν θα έχει νόημα. Θα πρέπει να κοιτάμε από το αρχικό tile από την άλλη. Για αυτό και έχουμε υπό συνθήκη το tile = choice[0] όπου αλλάζει το tile που κοιτάει δεξιά η αριστερά ο υπολογιστής. Άμα όμως είμαστε στην περίπτωση που του to\_where= down τότε δεν μπορούμε απλά να αλλάξουμε την μεταβλητή αφού όλος ο κώδικας για to\_where= up έχει προσπεραστεί ήδη. Για αυτό μειώνουμε το i που είναι ο iterator της while για να ξανατρέξει την while με ίδιο iterator (έχοντας και εδώ την flag και αλλάζοντας την με την αλλαγή του iterator). Έτσι τώρα έχουμε φτιάξει όλη την συνάρτηση για την επιλογή tile από τον υπολογιστή με την στρατηγική 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 βλέπουμε πάλι γίνεται τυχαία επιλογή για τον αριθμό των tiles που θα περιέχει η κίνηση καθώς και για το άμα η κίνηση θα είναι ανά στήλη η ανά γραμμή. Αυτό που αλλάζει με σχέση με πριν είναι ότι αυτή την φορά το πρώτο tile δεν είναι τυχαίο αλλά το πρώτο ελεύθερο και η κατεύθυνση είναι προς τα δεξιά ανά γραμμή και προς τα κάτω ανά στήλη (αλλιώς θα έπεφτε πάνω σε πιασμένα tile). Ο κώδικας είναι πιο απλός από της random μιας και είναι σταθερό το to\_where και αρά υπάρχουν 2 περιπτώσεις. Άμα τύχη να θέλει να πάει ανά γραμμή τότε ελέγχει άμα το στοιχείο διπλά είναι ελεύθερο και νόμιμη κίνηση και άμα είναι τότε το βάζει στην λίστα με τα tiles, άμα δεν είναι τότε ολοκληρώνει την σειρά έχοντας επιλέξει μόνο το πρώτο κενό tile που βρήκε στο ταμπλό. Το ίδιο συμβαίνει και άμα πάει ανά στήλη.

* 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

Αρχικά όπως και στις προηγούμενες συναρτήσεις ορίζουμε στην αρχή την λίστα που θα έχει τα tiles της κίνησης και βρίσκουμε πόσες ελεύθερες θέσεις υπάρχουν στο ταμπλό τώρα. Άμα οι ελεύθερες θέσεις τώρα είναι ίσες με αυτές του αρχικού ταμπλό (είναι η πρώτη κίνηση του παιχνιδιού) τότε ο υπολογιστής διαλέγει τυχαία μια από τις υπόλοιπες δυο στρατηγικές για να εκτελέσει για την πρώτη κίνηση του. Αυτό γίνεται γιατί δεν έχει να αντιγράψει κάποια κίνηση άμα δεν έχει παίξει καμία ακόμη ο παίκτης. Άμα έχει παίξει ο παίκτης έστω και μια φορά τότε πάλι έχουμε υποπεριπτώσεις. Άμα ο παίκτης έχει παίξει κελί διαγωνίου τότε ο υπολογιστής προσπαθεί να παίξει το συμμετρικό κελί της διαγωνίου από το κέντρο του ταμπλό. Αυτό γίνεται με την choice.append(diagon[len(diagon)-1-i]). Οπού diagon είναι η λίστα με όλα τα tiles της διαγωνίου. Άμα όμως αυτό το tile της διαγωνίου δεν είναι διαθέσιμο τότε βάζει σε ένα άλλο τυχαίο tile της διαγωνίου. Άμα δεν έχει άλλη tile ελεύθερο η διαγώνιος τότε επιλέγεται το πρώτο μη-κενό κελί στο ταμπλό. Αυτό γίνεται με την if not any(x in possibleMoves for x in diagon) και μετα την αντίστοιχη else (άμα δεν ύπαρχε άλλο tile στην διαγώνιο). Τώρα για το κύριο κομμάτι της συνάρτησης άμα ο παίκτης κάνει μια κίνηση ο υπολογιστής θα πρέπει να την αντιγράψει και να κάνει την συμμετρική αυτής με βάση την διαγώνιο. Αφού έχουμε την playermoves (την κίνηση του παίκτη που θέλουμε να αντιγράψει ο υπολογιστής) θα πρέπει όλα τα συμμετρικά tile των αρχικών να είναι ελευθέρα. Άμα είναι απλά τα προσθέτει στην λίστα choice και τα επιλεγεί. Άμα έστω και ένα δεν είναι τότε η μεταβλητή flag γίνεται 1 και τότε ο υπολογιστής επιλεγεί ισοπίθανα άμα θα εκτελέσει την κίνηση του με την random η την first fit. Και στις δυο περιπτώσεις θα πρέπει το μέγεθος της κίνησης να είναι το πολύ ίσο με το μέγεθος της κίνησης του παίκτη που προσπαθούμε να αντιγράψουμε. Άρα για οποία από τις δυο στρατηγικές πάει να κάνει, θα πρέπει να κοιτάει να είναι μικρότερο το αποτέλεσμα από την κίνηση του παίκτη. Όταν εν τέλη βρει μια τέτοια κίνηση τότε την περνάει στη choice και την εκτελεί.

Αφού εξηγήσαμε όλες τις συναρτήσεις και όλο το πρόγραμμα ας πάμε να δούμε πως δουλεύεις την πράξη. Να επισημάνουμε ότι για κάθε κίνηση του υπολογιστή πριν τυπωθεί το ταμπλό με την κίνηση του τυπώνεται η κίνηση σαν λίστα έτσι ώστε να μην χάνουμε τι έκανε ο υπολογιστής στην σειρά 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