Pacman Project 1 Αλέξανδρος Αλατζάς **sdi1900005**

Q1, Q2, Q3 : Ακολουθώ πιστά με python τον ψευδοκώδικα που μας δίνεται στην εκφώνηση. Μεταξύ dfs και bfs αλλάζει μόνο η δομή δεδομένων από Stack σε Queue, το υπόλοιπο παραμένει ίδιο. Για τον A*, χρησιμοποιείται Priority Queue , άρα χρησιμοποιούμε την ευρετική για push. Ακολουθούμε ίδια λογική με τα προηγούμενα, μόνο που εδώ θα πρέπει να υπολογίσουμε τη συνάρτηση αξιολόγησης του A* f, η οποία θα είναι το άθροισμα των g και h, όπου g η συνάρτηση κόστους ως τον κόμβο και h η ευρετική.

Q4 : Ορίζω ως αρχική κατάσταση του προβλήματος ένα tuple(0,0,0,0) στο οποίο κάθε μηδενικό αντιπροσωπεύει μια γωνία που δεν έχουμε ακόμα επισκεφτεί. Συνεπώς κατάσταση στόχου αποτελεί ένα tuple(1,1,1,1) αφού επισκεφτούμε και τις 4 γωνίες. Στη getNextState διαθέτω μια λίστα με τις γωνίες του προβλήματος και μια λίστα για όσες συντεταγμένες έχω επισκεφτεί ήδη. Ελέγχω αν είναι γωνία την οποία δεν έχω επισκεφτεί ακόμα, και αν ναι, τότε την επισκέπτομαι και η συνάρτηση επιστρέφει ένα tuple με τις επόμενες συντεταγμένες καθώς και το tuple με τις γωνίες που έχουμε επισκεφτεί. Η expand επιστρέφει τα παιδιά ενός state υπολογίζοντας για κάθε κίνηση, ποιο είναι το επόμενο παιδί, τις κινήσεις που χρειάζονται για να το φτάσουμε και το κόστος αυτών, απλά καλώντας τις getNextState και getActionsCost.

Q5 : Από τη θέση στην οποία βρίσκεται ο pacman, υπολογίζω μέσω της Manhattan τις αποστάσεις μέχρι την κάθε γωνία που δεν έχω επισκεφτεί ακόμα, και στη συνέχεια παίρνω την μέγιστη αυτών ως ιδανικότερη απάντηση. Αυτή η ευρετική θα είναι συνεπής καθώς για κάθε ενέργεια με κόστος ‘dist’, η εκτέλεση αυτής της ενέργειας έχει σαν αποτέλεσμα την μείωση της ευρετικής συνάρτησης κατά το πολύ ‘dist’. Στην ουσία, πηγαίνοντας κάθε φορά στην γωνία με την μεγαλύτερη απόσταση(Manhattan) από το σημείο, εξασφαλίζουμε ότι δεν θα χρειαστεί ξανά στο μέλλον να κάνουμε κάποια κίνηση για μεγαλύτερη απόσταση. Δεν επιστρέφει συνέχεια μηδέν ούτε παίρνει αρνητικές τιμές και εφόσον είναι συνεπής θα είναι και παραδεκτή.

Q6 : Χρησιμοποιώ την συνάρτηση mazeDistance η οποία δίνεται από το project. Έτσι, κατά παρόμοια λογική με προηγουμένως, υπολογίζω τις αποστάσεις για κάθε τελεία, τις αποθηκεύω σε μια λίστα και κάθε φορά επιστρέφω ως λύση την μεγαλύτερη απόσταση. Οι τοίχοι του προβλήματος συμπεριλαμβάνονται ήδη από την mazeDistance, αφού μας δίνει κάθε φορά την απόσταση μέσα στον λαβύρινθο. Η λογική που ακολουθώ είναι ίδια με του Q5, θεωρώντας κάθε φορά ότι προτεραιότητά μου θα είναι η μέγιστη απόσταση πρώτα.

Q7 : Απλώς καλούμε τον bfs που έχουμε ήδη υλοποιήσει στην εργασία. Αναζητώντας κατά πλάτος, βρίσκουμε τη διαδρομή για την κοντινότερη τελεία κάθε φορά.