1η εργασία για το εργαστήριο Τεχνητής Νοημοσύνης

Εφαρμογή αλγορίθμων αναζήτησης στο πρόβλημα του Pac-man

Από τον Άγγελο Νιάφα
Α.Μ. 151037 πτυχίο ΠΑ.Δ.Α.

Εισαγωγή

Για την εργασία αυτή χρησιμοποίησα την εφαρμογή PyCharm Community Edition 2022.3 καθώς η απλοποιημένη του μορφή με βοήθησαν στην κατανόηση της εργασίας και στην καλύτερη υλοποίησή της.

Ανάλυση του προγράμματος

Η ανάλυση του κώδικα γραμμή προς γραμμή έγινε με μορφή σχολίων στον κώδικα που έχει αναρτηθεί και δεν θα αναλυθεί περαιτέρω στο pdf αυτό. Εδώ θα αναλύσουμε την λειτουργικότητα του κώδικα καθώς και τα αποτελέσματα του.

Αρχικά για την επιλυση της ερασίας αυτής ξεκίνησα με το αρχείο pacman_operators που δόθηκε από το εργαστήριο. Από εκεί μορφοποίησα τον κώδικα έτσι ώστε το grid να είναι 2 διαστάσεων και όχι μονοδιάστατος. Όρισα initial state και goal της δικής μου αρέσκειας όπως είχε ζητηθεί από τους καθηγητές.

Επιπλεόν πρόσθεσα φυσικά τις κινήσεις πανω και κάτω καθώς και άλλαξα τις κινήσεις αριστερα και δεξία με τα κατάλληλα όρια.

```
def can_move_up(state):
    for i in range(len(state)):
        for k in range(2):
            for j in range(1, 4):
                return not state[j][i][k] == 'p'


def can_move_down(state):
    for i in range(len(state)):
        for k in range(2):
            for j in range(3):
                if j == 3:
                    return 0
                else:
                    return not state[j][i][k] == 'p'

def move_up(state):
    if can_move_up(state):
        for j in range(1, 4):
            for i in range(len(state)):
                if state[j][i][0] == 'p':
                    state[j][i][0] = ''
                    state[j - 1][i][0] = 'p'
                    return state
    else:
        return state


def move_down(state):
    if can_move_down(state):
        for j in range(0, 3):
            for i in range(len(state)):
                if state[j][i][0] == 'p':
                    state[j][i][0] = ''
                    state[j + 1][i][0] = 'p'
                    return state
```

Στη συνέχεια εμπνεύστηκα από τα αρχεία search_algorithm_front_queue_with_operators pacman και Search_algorithm_front_with_operators pacman που μου δόθηκαν για να υλοποιήσω τον αλγόριθμο πρώτα σε βαθος αναζήτησης (DFS) με παρακολούθηση μετώπου και έπειτα τον αλγόριθμο της πρώτα σε πλάτος αναζήτησης (BFS) με παρακολούθηση της ουράς των μονοπατιών.

```
def extend_queue(queue, method):
    if method == 'DFS':
        print("Queue:")
        print(queue)
        node = queue.pop(0)
        queue_copy = copy.deepcopy(queue)
        children = find_children(node[-1])
        for child in children:
            path = copy.deepcopy(node)
            path.append(child)
            queue_copy.insert(0, path)

    elif method == 'BFS':
        print("Queue:")
        print(queue)
        node = queue.pop(0)
        queue_copy = copy.deepcopy(queue)
        children = find_children(node[-1])
        for child in children:
            path = copy.deepcopy(node)
            path.append(child)
    #    queue_copy.........
    # elif method=='BestFS':
    # else: "other methods to be added"

    return queue_copy
```

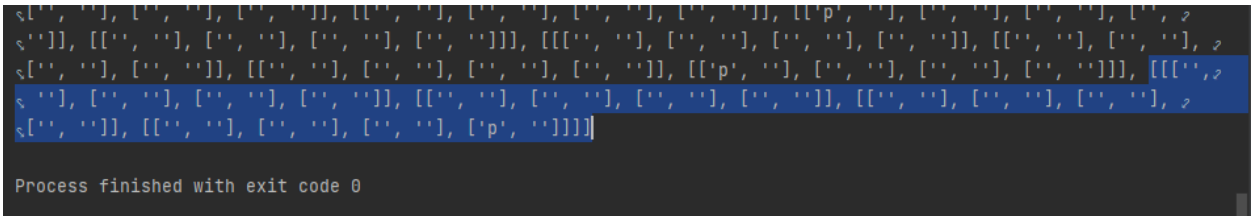
Φυσικά οι μέθοδοι αυτοί αρχικοποιούνται από τη συνάρτηση εύρεσης απογόνων που έχουμε δημιουργήσει πιο πάνω.

```
def find_children(state):
    children=[]
    right_state=copy.deepcopy(state)
    child_right=move_right(right_state)
    left_state=copy.deepcopy(state)
    child_left=move_left(left_state)
    up_state=copy.deepcopy(state)
    child_up=move_up(up_state)
    down_state=copy.deepcopy(state)
    child_down=move_down(down_state)
    eat_state=copy.deepcopy(state)
    child_eat=eat(eat_state)

    if not child_right==None:
        children.append(child_right)

    if not child_left==None:
        children.append(child_left)

    if not child_up==None:
        children.append(child_up)

    if not child_down==None:
        children.append(child_down)

    if not child_eat==None:
        children.append(child_eat)

    return children
```

Κάνοντας αυτές τις αλλαγές στους δοσμένους κώδικες θα αναλύσουμε τα αποτελέσματα που έβγαλα και πως έφτασα στο επιθυμητό αποτέλεσμα ή αλλιως :

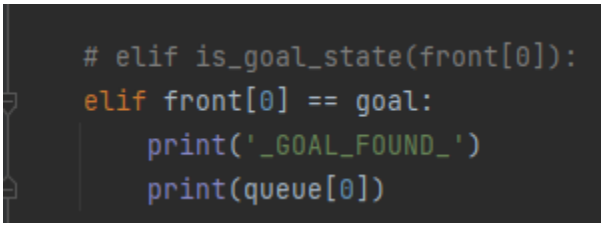
```
_GOAL_FOUND_
```

Άμα τρέξουμε το πρόγραμμα θα παρατηρήσουμε ότι κάνει πολλές κινήσεις αλλά εν τέλει φτάνει εκεί που θέλουμε να φτάσει. Για παράδειγμα αν παρατηρήσουμε στο τέλος του αποτελέσματος θα παρατηρήσουμε το εξής:



Παρατηρούμε λοιπόν ότι όλα τα φρούτα είναι φαγωμένα και τον pac man στην τελική θέση που έχω ορίσει εγω, δηλαδή τέρμα κάτω δεξιά του grid.

Πιο πριν όμως παρατηρούμε ένα goal found στη μέση του αποτελέσματος. Αυτό συμβαίνει διότι το πρόγραμμα βρήκε λύση. Και αυτό το βρίσκουμε χάρη στις εξής γραμμές κώδικα:



Κατακλείδα

Εν κατακλείδι μέσω των δεδομένων αρχείων που δόθηκαν και τις κατάληλλες παραμετροποιήσεις που δέχθηκαν για να τρέχει ο pacman πρώτα με DFS και έπειτα με BFS για την κατανάλωση των φρούτων και στην οδήγηση του στην τελική θέση οδηγηθήκαμε στο τελικό αποτέλεσμα. Που αυτό είναι η καλύτερη κατανόηση αφενός της γλώσσας python και αφετέρου των αλγορίθμων BFS και DFS που αποτελούν τα βασικά εργαλεία στην εύρεση αποτελέσματος στο φάσμα της τεχνητής νοημοσύνης.