Αναπαράσταση του κόσμου του προβλήματος του pacman με Python

Περίληψη Προβλήματος

Το συγκεκριμένο Πρόβλημα είναι η αναπαράσταση του εσωτερικού κόσμου του Pacman. Όπου ο Pacman πρέπει να φάει όλα τα επιτρεπτά φρούτα και να καταλήξει στο τελευταίο κελί .Η αναπαράσταση του κόσμου έχει ως εξής:

• Τα κελιά οπού τα εμφανίζουμε ως λίστες [[“ ”,” ”] [“ ”,” ”] [“ ”,” ”] [“ ”,” ”]]

• Τα φρούτα οπού τα εμφανίζουμε με γράμματα F,O,D

• Και ο Pacman που εμφανίζεται με P

Έπειτα έχουμε της επιτρεπόμενες κινήσεις οπού είναι :

Ο Pacman μπορεί να κάνει τίς εξής κινήσεις :

Να πάει

• δεξιά

• Αριστερά

Και το να τρώει τα φρούτα

Επίσης κάποια απο τα φρούτα παρασιάζουν κάποιες ιδιότητες :

- Το O (orange) πηγαίνει τον pacman ένα βήμα πίσω
- Το D (double) εμφανίζει ενα F(fig) ένα κελι μπροστά

1

Τελεστές μετάβασεις

Οι τελεστές μετάβασεις είναι η αναπαράσταση των κινήσεων που μπορεί να κάνει ο pacman.Οπως είδαμε πιο πάνω οι κινήσεις είναι ως εξής:

- Κίνηση δεξία οπου υλοποιήτε με την συνάρτηση move_right

```
def move_right(state):
 if can_move_right(state):       #ελεγχος αν μπορεί να κινηθεί δεξιά
     for i in range(len(state)):
         if state[i][0]=='p':          #αν μπορεί τότε ο pacman
             state[i][0]=''            #πηγένει μια θεση δεξία
             state[i+1][0]='p'
              return state
   else:                                    #αλλιώς μενει εκεί που είναι
     return state
```

- Κίνηση αριστερά οπου υλοποιήτε με την συνάρτηση move_left

```
def move_left(state):
  if can_move_left(state):       #ελεγχος αν μπορεί να κινηθεί  αριστερά
     for i in range(len(state)):
         if state[i][0]=='p':      #αν μπορεί τότε ο pacman
               state[i][0]=''     #πηγένει μια θεση αριστερά
                 state[i-1][0]='p'
                  return state
         else:                        #αν δεν μπορεί μένει εκει που είναι
             return state
```

- Φάγωμα φρούτου οπου υλοποιήτε με την συνάτηση eat

```
def eat(state):
    if can_eat(state):                          #ελεγχος αν μπορεί να φαει
        for i in range(len(state)):
            if state[i][0]=='p' and state[i][1]=='f':        # ελεγχος για ποιο φρουτο ειναι
                state[i][1]=''                              # το τρώει και επιστρέφη
                return state                                # την κατασταση
        for i in range(len(state)):
            if state[i][0]=='p' and state[i][1]=='o':        #ελεγχος για ποιο φρουτο ειναι
                state[i][1]=''                              # το τρώει
                for i in range(len(state)):
                    if state[i][0]=='p':                    #και πηγένει μια θεση πίσω
                        state[i][0]=''
                        state[i-1][0]='p'
                        return state
                return state
        for i in range(len(state)):
            if state[i][0]=='p' and state[i][1]=='d':        #ελεγχος για ποιο φρουτο ειναι
                state[i][1]=''                              #το τρωει
                state[i+1][1]='f'                           #και βαζει ενα φρούτο μια θεση
                return state                                #μπορστά
        else:
            return state
```

• Και τελος εχουμε την find_children οπου ειναι η συναρταση δημιουργειας απογονων οπου δημιουργουνται απο τον συνδιασμο τον τελεστων μετάβασεις

```
def find_children(state):
    children=[] #αρχικοποίηση άδειας λίστας για την αποθήκευση των
                                        #καταστάσεων παιδιών
    right_state=copy.deepcopy(state)    #αντιγραφή τρέχουσας κατάστασης
    child_right=move_right(right_state) #εφαρμογή τελεστή μετακίνησης δεξιά στην τρέχουσα
                                        #κατάσταση και δημιουργία της νέας κατάστασης
    left_state=copy.deepcopy(state)
    child_left=move_left(left_state)  #αντιγραφή τρέχουσας κατάστασης
                                        #εφαρμογή τελεστή μετακίνησης αριστερά στην τρέχουσα
                                        #κατάσταση και δημιουργία της νέας κατάστασης
    eat_state=copy.deepcopy(state)
    child_eat=eat(eat_state) #αντιγραφή τρέχουσας κατάστασης
                                        #εφαρμογή τελεστή φαγώματος στην τρέχουσα
                                        #κατάσταση και δημιουργία της νέας κατάστασης
    if not child_eat==state:
        children.append(child_eat) #αν ο pacman βρίσκεται σε κελί με φρούτο
                                        #προσθέτουμε τη νέα κατάσταση, αφού έχει φάει το φρούτο,
    if not child_left==state:
        children.append(child_left)         #αν ο pacman μπορεί να κινηθεί αριστερά
                                        #προσθέτουμε τη νέα κατάσταση, στην οποία έχει
                                        #μετακινηθεί στο αριστερό κελί, στη λίστα
```

```
    if not child_right==state:
        children.append(child_right)        #αν ο pacman μπορεί να κινηθεί δεξιά
                                                #προσθέτουμε τη νέα κατάσταση, στην οποία έχει
                                                #μετακινηθεί στο δεξί κελί, στη λίστα
    return children                             #επιστρέφουμε σε λίστα τις νέες καταστάσεις παιδιά
```

Κανόνες κινήσεων

Για να εξασφαλίσουμε οτι οι τελεστές των κινήσεων είναι έγκυρες και οτι ακουλουθούν τους κανόνες του κόσμου του προβλήματος εχουμε δημιουργήσει τις εξής μεθόδους:

- Can_move_left :

  Η μέθοδος can_move_left ελέγχει αν ο pacman μπορεί να κινηθεί αριστερά και το αν βρίσκεται στο πρώτο κελί

- Can_move_right :

  Η μέθοδος can_move_right ελέγχει αν ο pacman μπορεί να κινηθεί δεξιά και το αν βρίσκεται στο τελευταίο κελί

- Can _eat :

  Η μέθοδος can_eat ελέγχει αν ο pacman βρίσκετε στο ίδιο κελί με ένα απο τα φρούτα που επιτρέπεται να φάει

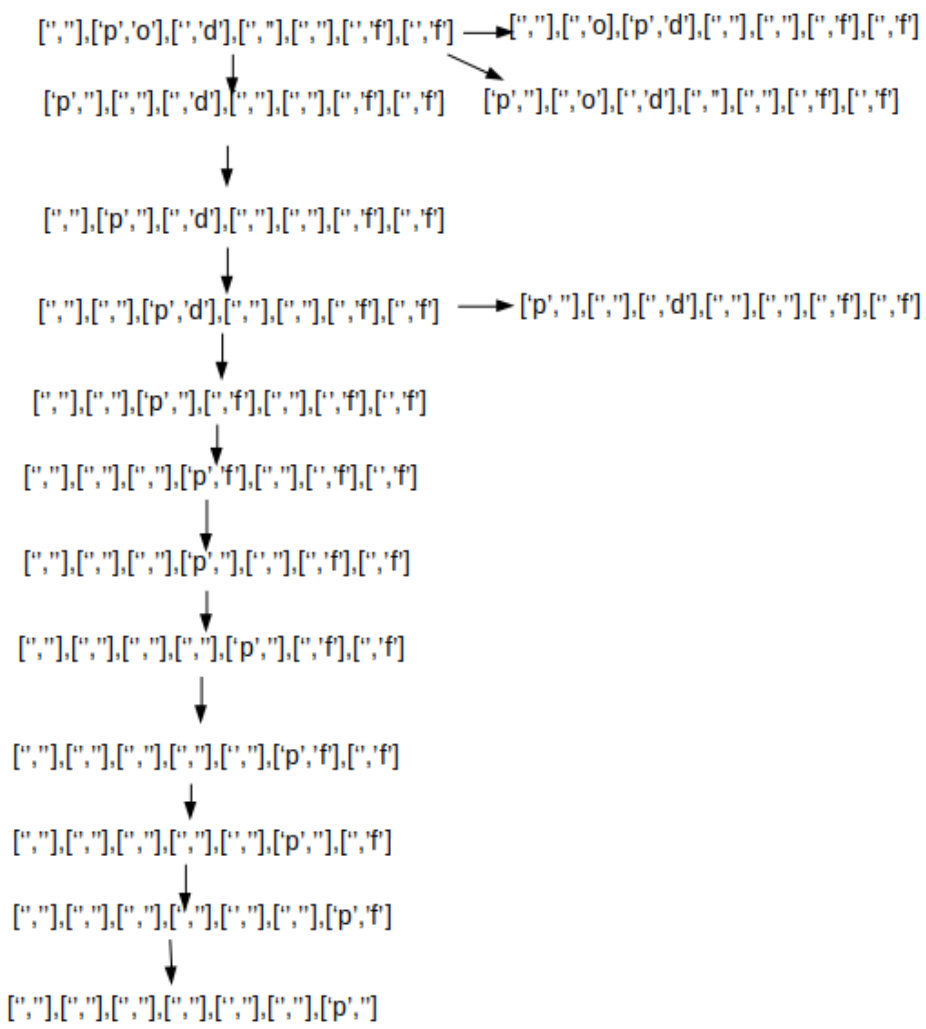
Αλγόριθμος αναζήτησης

Για την επίλυση του προβλήματος χρησιμοποίηθηκε ο αλγόριθμος search_front_queue οπού είναι αναδρομικός και υλοποιεί τον κορμό του κώδικα της αναζήτησης με παρακολούθηση του μετώπου αλλά και της ουράς των μονοπατιών που σχηματίζει η εκάστοτε μέθοδος αναζήτησης. Ο κώδικας επιστρέφει τη λύση υπό μορφή ακολουθίας καταστάσεων που οδηγούν από την αρχική κατάσταση στον επιδιωκόμενο στόχο.Και με την χρήση της expand_front και exttend_queue δημιουργουντε οι λίστες οπου έχουν κάθε φορά την το μέτωπο αναζητήσεις (front) και την ουρά(queue) .Επίσης έχουμε την λίστα closed οπου έχουμε τίς καταστάσεις απο τα προηγούμενα τρεξίματα όπου αποτελείται απο τους προηγούμενους γονικούς κόμβους

Μεθοδος αναζήτησης

Η μέθοδος αναζήτησης που χρησιμοποιήθηκε είναι ο DFS όπου παίρνει τον αριστερότερο κόμβο child ενός γονικού κόμβου και επαναλαμβάνει αυτή τη διαδικασία μέχρι να βρει αδιέξοδο ή να βρει τον στόχο. Κάθε φορά που ένας κόμβος γίνεται γονικός αφαιρείται από το μέτωπο και οι κόμβοι child του μπαίνουν στην αρχή του μετώπου. Ταυτόχρονα ο γονικός κόμβος προστίθεται στο σύνολο closed.

Δένδρο αναζήτησης



Παρατηρήσεις

Κατα την διάρκεια τον δοκιμών παρατηρήθηκε ότι ο pacman δεν τρώει τα φρούτα με την σειρα που τα βλέπει οταν αυτα βρισκονται αρκετα κοντά μεταξύ τους γαι αυτο τον λόγο εχουμε αφησει αρκετές αποστάσεις μεταξη τους.

ΚΩΔΙΚΑΣ

```
import copy
 def can_move_left(state):
    return not state[0][0]=='p’
def can_move_right(state):
    return not state[len(state)-1][0]=='p'
 def can_eat(state):
    for i in range(len(state)):
        if state[i][0]=='p' and state[i][1]=='f':
            return 1
        if state[i][0]=='p' and state[i][1]=='o':
            return 1
        if state[i][0]=='p'and state[i][1]=='d':
            return 1
```

```
def move_left(state):
    if can_move_left(state):
        for i in range(len(state)):
            if state[i][0]=='p':
                state[i][0]=''
                state[i-1][0]='p'
                return state
            else:
                return state

def move_right(state):
    if can_move_right(state):
        for i in range(len(state)):
            if state[i][0]=='p':
                state[i][0]=''
                state[i+1][0]='p'
                return state
    else:
        return state
```

```
def eat(state):
    if can_eat(state):
        for i in range(len(state)):
            if state[i][0]=='p' and state[i][1]=='f':
                state[i][1]=''
                return state
        for i in range(len(state)):
            if state[i][0]=='p' and state[i][1]=='o':
                state[i][1]=''
                for i in range(len(state)):
                    if state[i][0]=='p':
                        state[i][0]=''
                        state[i-1][0]='p'
                        return state
                return state
        for i in range(len(state)):
            if state[i][0]=='p' and state[i][1]=='d':
                state[i][1]=''
                state[i+1][1]='f'
                return state
    else:
        return state
```

```
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)
    eat_state=copy.deepcopy(state)
    child_eat=eat(eat_state)
      if not child_eat==state:
        children.append(child_eat)
    if not child_left==state:
        children.append(child_left)
    if not child_right==state:
        children.append(child_right)
 return children
 def make_front(state):
    return [state]
```

```
def expand_front(front, method):
    if method=='DFS':
```

```
        if front:
            print("Front:")
            print(front)
            node=front.pop(0)
            for child in find_children(node):
                front.insert(0,child)
        return front
 def make_queue(state):
    return [[state]]


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]
    return queue_copy
```

```
def find_solution(front, queue, closed, goal, method)
    if not front:
        print('_NO_SOLUTION_FOUND')

    elif front[0] in closed:
        new_front=copy.deepcopy(front)
        new_front.pop(0)
        new_queue=copy.deepcopy(queue)
        new_queue.pop(0)

        find_solution(new_front, new_queue, closed, goal, method)
    elif front[0]==goal:
        print('_GOAL_FOUND_')
        print(queue[0])
       else:
        closed.append(front[0])
        front_copy=copy.deepcopy(front)
        front_children=expand_front(front_copy, method)
        queue_copy=copy.deepcopy(queue)


        queue_children=extend_queue(queue_copy, method)
        closed_copy=copy.deepcopy(closed)
        find_solution(front_children, queue_children, closed_copy, goal, method)
```

```
def main():

    initial_state=[['',''],['p','o'],['','d'],['',''],['',''],['','f'],['','f']]
    goal=[['',''],['',''],['',''],['',''],['',''],['',''],['p','']]
    method='DFS'
    print('____BEGIN__SEARCHING____')
    find_solution(make_front(initial_state), make_queue(initial_state), [], goal, method)
     if __name__ == "__main__":
              main()
```