

ANDROID DEVELOPMENT ASSESSMENT

Όνομα: Γιάννης

Επώνυμο: Φραγκούλης

Εργασία: KNIGHT TOUR PATHS

*(Σημείωση: Επειδή πολλοί κόμβοι είναι κοινοί όταν δοκίμασα να βάλω διαφορετικά χρώματα για κάθε μονοπάτι δεν ήταν ευδιάκριτα τα μονοπάτια επειδή υπάρχουν πολλά κοινά. Οπότε εμφανίζω με ίδιο χρώμα μόνο τους κόμβους με τις τελικές τιμές και απο κάτω σε textview τα ολοκληρωμένα μονοπάτια)



Γλώσσα προγραμματισμού → Java

Minimum SDK version → 21

Περιγραφή υλοποίησης

Αρχικά θα περιγράψω τον αλγόριθμο που έφτιαξα σε φυσική γλώσσα. Χρησιμοποίησα έναν αναδρομικό αλγόριθμο στην ουσία. Καλώ αρχικά μια συνάρτηση με το αρχικό σημείο που θέτει ο χρήστης. Το ένα σημείο μπορεί να έχει το πολύ οχτώ έγκυρες κινήσεις, λιγότερες αν βρίσκεται σε γωνία. Τις κινήσεις τις έχω αποθηκευμένες σε πίνακα. Η πρώτη κλήση της συνάρτησης μου επιστρέφει όλα τα πιθανά σημεία του πρώτου σημείου. Στη συνέχεια καλώ ξανά την ίδια συνάρτηση με την νέα λίστα. Στη συνέχεια ξανα το ίδιο με την νέα λίστα απο συντεταγμένες. Στο μεταξύ για κάθε σημείο κρατάω σε ένα attribute της κλάσης τον γονικό κόμβο έτσι ώστε αργότερα να μπορώ να έχω το μονοπάτι των κινήσεων. Έτσι μετά κρατάω τα μονοπάτια που έχουν τελικό αυτό που έβαλε ο χρήστης. Επίσης στο ίδιο σημείο καλώ μια άλλη συνάρτηση η οποία για κάθε σημείο μου δίνει το αναλυτικό μονοπάτι προς τον προηγούμενο κόμβο. Στο τέλος για να συνθέσω το ολοκληρωμένο μονοπάτι τρέχω μια επανάληψη για κάθε μονοπάτι τελικών κινήσεων και απο εκεί ελέγχω αν το `List[k]` και το `list[k-1]` είναι ίσο με ένα τελικό και ένα αρχικό σημείο της λίστας με τα αναλυτικά μονοπάτια, αν είναι το ενσωματώνω στο μονοπάτι που θέλω να δημιουργήσω και έτσι στο τέλος έχω όλα τα πιθανά μονοπάτια.

Παρακάτω θα κάνω ένα overview από τα σημαντικότερα σημεία του κώδικα.

Η συνάρτηση που είναι υπεύθυνη για την αναζήτηση κατά πλάτος. Την καλώ τρείς φορές.

```
9
10 public class Layers {
11     static final private int[] X = {2, 1, -1, -2, -2, -1, 1, 2};
12     static final private int[] Y = {1, 2, 2, 1, -1, -2, -2, -1};
13
14     @
15     static ArrayList<Coordinates> findLayerCoordinates(ArrayList<Coordinates> totalList) {
16         ArrayList<Coordinates> queueLevel = new ArrayList<>();
17
18         for (int i = 0; i < totalList.size(); i++) {
19
20             Coordinates parentCoord = totalList.get(i);
21
22             for (int j = 0; j < 8; j++) {
23
24                 int x = parentCoord.x + X[j];
25                 int y = parentCoord.y + Y[j];
26
27                 if (x >= 0 && y >= 0 && x < 8 && y < 8) {
28
29                     Coordinates coordinates = new Coordinates(x, y, parentCoord);
30                     addNeighbors(x,y, coordinates);
31                     queueLevel.add(coordinates);
32                 }
33             }
34         }
35     }
36     return queueLevel;
37 }
```

Γραμμη 11,12 : όλοι οι πιθανοί συνδιασμοί των κινήσεων που μπορεί να κάνει το αλογάκι.

Γραμμή 16: Η νέα λίστα που θα επιστραφεί.

Γραμμές 18-35: Για κάθε σημείο βρίσκω όλα τα έγκυρα γειτονικά καθώς και κρατάω τον γονικό κόμβο.

Γραμμή 30: Για κάθε κόμβο προσθέτω με μια συνάρτηση όλα τα αναλυτικά μονοπάτια προς όλους τους κόμβους.

Συνάρτηση που φτιάχνει το grid layout

```
51 private void setChessBoard() {
52     GridLayoutId.setColumnCount(8);
53     GridLayoutId.setRowCount(8);
54
55     for (int i = 1; i <= 8; i++) {
56         for (int j = 1; j <= 8; j++) {
57
58             final int row = i;
59             final int col = j;
60             final Button b1 = new Button( context: this);
61             b1.setId(counter++);
62             b1.setTag((i) + "," + (j));
63             b1.setOnClickListener(onClick(v) → {
64                 handleClicks(b1, row, col);
65             });
66             setChessColors(i, b1);
67
68             RelativeLayout.LayoutParams lp = new RelativeLayout.LayoutParams( w: 130, h: 130);
69             lp.addRule(RelativeLayout.RIGHT_OF, subject: b1.getId() - 1);
70             b1.setLayoutParams(lp);
71             GridLayoutId.addView(b1);
72         }
73     }
74
75     resetbtn.setOnClickListener(onClick(v) → { resetChess(); });
76     findPathsbtn.setOnClickListener(onClick(v) → {
77         if (Click_Times < 2)
78             Toast.makeText( context: MainActivity.this, text: "choose two points!", Toast.LENGTH_SHORT).show();
79         else
80             startProsess();
81     });
82     showCoorbtn.setOnClickListener((v) → { showCoordinates(); });
83 }
```

Γραμμή 61: Για να μπορέσω να προσδιορίσω σε ποια θέση του grid layout βρίσκεται το κάθε κουμπί βάζω tag τις κατάλληλες συντεταγμένες.

Γραμμή 84-90: Ελέγχω το πόσες φορές έχει κλικάρει ο χρήστης γιατί πρέπει να έχει επιλέξει δύο σημεία για να αρχίσει η διαδικασία

Συνάρτηση όπου γίνεται η βασική λειτουργία, δηλαδή η έυρεση των μονοπατιών.

```
174 for (int i = 0; i < thirdLayer.size(); i++) {
175     Coordinates coordinates = thirdLayer.get(i);
176     String initial = coordinates.toString(); //έχω όλες τις συντεταγμένες του κάθε σημείου
177     if (initial.contains(end)) { //παιρνω μόνο τις συντεταγμένες που περιέχουν το τελικό σημείο
178         int flag = 0;
179         String[] sec = initial.split( regex: "-");
180         StringBuilder newStr = new StringBuilder();
181
182         for (int h = sec.length - 2; h >= 0; h--) {
183             if (sec[h].equals(end))
184                 flag = h;
185             if (flag <= h) { //κρατώ την συμβολοσειρά συντεταγμένων μέχρι να βρω το τελικό σημείο
186                 newStr.append(sec[h]).append("-");
187             }
188         }
189
190         StringBuilder newStrTotal = new StringBuilder();
191         if (!final_list.contains(newStr.toString()) && newStr.toString().contains(end)) {
192             final_list.add(newStr.toString());
193             String[] sec1 = newStr.toString().split( regex: "-");
194             for (int h = 0; h <= sec1.length - 1; h++) {
195                 if (sec1[sec1.length - 1].equals(sec1[sec1.length - 2]) )
196                     continue;
197                 //ελέγχω τα όρια του πίνακα για να καλέσω την συνάρτηση που βρίσκει γειτονικούς κόμβους παίρνοντας της 2 ορίσματα
198                 //ένα τελικό και ένα αρχικό σημείο το αποτέλεσμα το κάνω append σε ένα string ώστε για κάθε τελικό σημείο να
199                 //έχω το πλήρες μονοπάτι
200                 if (h + 1 <= sec1.length - 1)
201                     newStrTotal.append(returnCorrectNeighbors(sec1[h + 1], sec1[h]));
202             }
203
204         }
205
206         if (!final_listWithNeighbors.contains(newStrTotal.toString()) && !newStrTotal.toString().equals("")) { //πρέπει αυτό
207             newStrTotal = new StringBuilder(removeLastCharacter(newStrTotal.toString()));
208             final_listWithNeighbors.add(newStrTotal.toString());
209             displayedPathsId.append(newStrTotal + "\n\n");
210         }
211     }
212 }
```

Γραμμή 174: η λίστα thirdLayer περιέχει την Τρίτη κλήση της ίδιας συνάρτησης

Γραμμή 175-176: Παιρνω το κάθε αντικείμενο το οποίο η override method toString έχει όλους τους γονικούς κόμβους

Γραμμή 177: Κρατώ μόνο τα μονοπάτια που περιέχουν το τελικό σημείο που όρισε ο χρήστης

Γραμμές 182-188: Κρατάω μόνο τα μονοπάτια που τελειώνουν στο τελικό σημείο

Γραμμή 201: Καλώ μια συνάρτηση με τα γειτονικά σημεία έτσι ώστε να κάνει append το ολοκληρωμένο μονοπάτι.

SCREENSHOT ΑΠΟ ΤΗΝ ΕΚΤΕΛΕΣΗ



