ANDROID DEVELOPMENT ASSESSMENT

Όνομα: Γιάννης

Επώνυμο: Φραγκούλης

Εργασια: 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 ολα τα πιθανά σημεία του πρώτου σημείου.Στη συνέχεια καλώ ξανά την ίδια συνάρτηση με την νέα λιστα.Στη συνέχεια ξανα το ίδιο με την νέα λίστα απο συντεταγμένες.Στο μεταξύ για κάθε σημείο κρατάω σε ενα 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 αυτο 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 εκει ελέγχω αν το List[k] και το list[k-1] είναι ίσο με ένα τελικό και ένα αρχικο 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.

```
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  @   static ArrayList<Coordinates> findLayerCoordinates(ArrayList<Coordinates> totalList) {
15
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                      addNeighboors(x,y, coordinates);
31                      queueLevel.add(coordinates);
32                  }
33              }
34
35          }
36          return queueLevel;
37      }
```

Γραμμές 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) → {
66                          handleClicks(b1, row, col);
67                  });
69                  setChessColors(i, b1);
70
71                  RelativeLayout.LayoutParams lp = new RelativeLayout.LayoutParams( w: 130,  h: 130);
72                  lp.addRule(RelativeLayout.RIGHT_OF,  subject: b1.getId() - 1);
73                  b1.setLayoutParams(lp);
74                  gridLayoutId.addView(b1);
75              }
76          }
77
78          resetbtn.setOnClickListener(onClick(v) → { resetChess(); });
84          findPathsbtn.setOnClickListener(onClick(v) → {
87                  if (Click_Times < 2)
88                      Toast.makeText( context: MainActivity.this,  text: "choose two points!", Toast.LENGTH_SHORT).show();
89                  else
90                      startProsess();
91          });
93          showCoorbtn.setOnClickListener((v) → { showCoordinates(); });
99      }
100
```

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();          //εχω ολες τις συντεταγμενες του καθε σημείου
177         if (initial.contains(end)) {                      //παιρνω μονο τις συντεταγμενες που περιεχουν το τελικο σημειο
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) {                           //κραταω την συμβολοσειρα συντεταγμενων μεχρι να βρω το τελικο σημειο
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                     //ελεγχω τα ορια του πινακα για να καλεσω την συναρτηση που βρισκει γειτονικους κομβους παιρνωντας της 2 ορισματα
198                     //ενα τελικο και ενα αρχικο σημειο το αποτελεσμα το κανω append σε ενα string ωστε για καθε τελικο σημειο να
199                     //εχω το πλήρες μονοπατι
200                     if (h + 1 <= sec1.length - 1)
201                         newStrTotal.append(returnCorrectNeighboors(sec1[h + 1], sec1[h]));
202                 }
203
204             }
205
206             if (!final_listWithNeighboors.contains(newStrTotal.toString()) && !newStrTotal.toString().equals("")) {       //πρεπει αυτ
207                 newStrTotal = new StringBuilder(removeLastCharacter(newStrTotal.toString()));
208                 final_listWithNeighboors.add(newStrTotal.toString());
209                 displayedPathsId.append(newStrTotal + "\n\n");
210             }
211         }
212     }
```

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

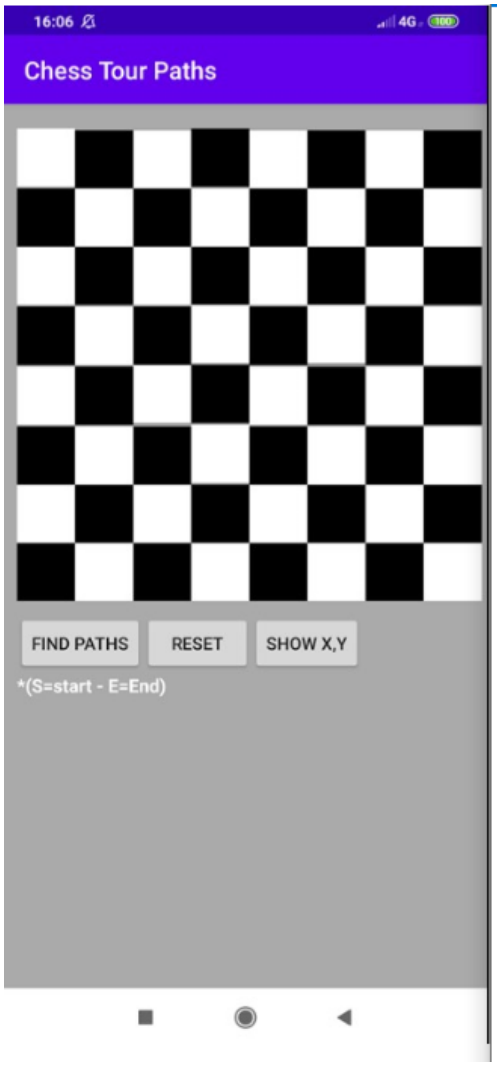
Γραμμές 175-176: Παιρνω το κάθε αντικείμενο το οποίο η override method toString έχει όλους τους γονικούς κόμβους

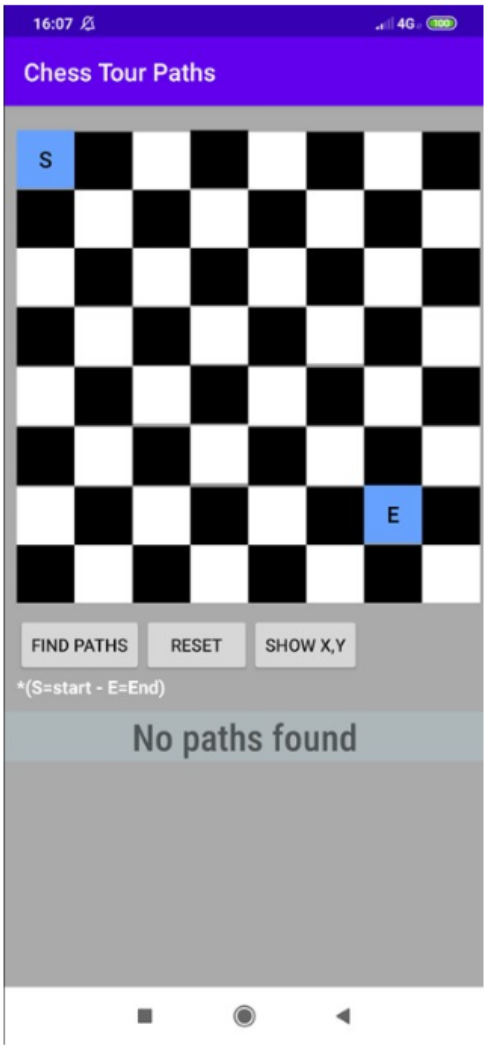
Γραμμή 177: Κρατώ μόνο τα μονοπάτια που περιέχουν το τελικό σημείο που όρισε ο χρήστης

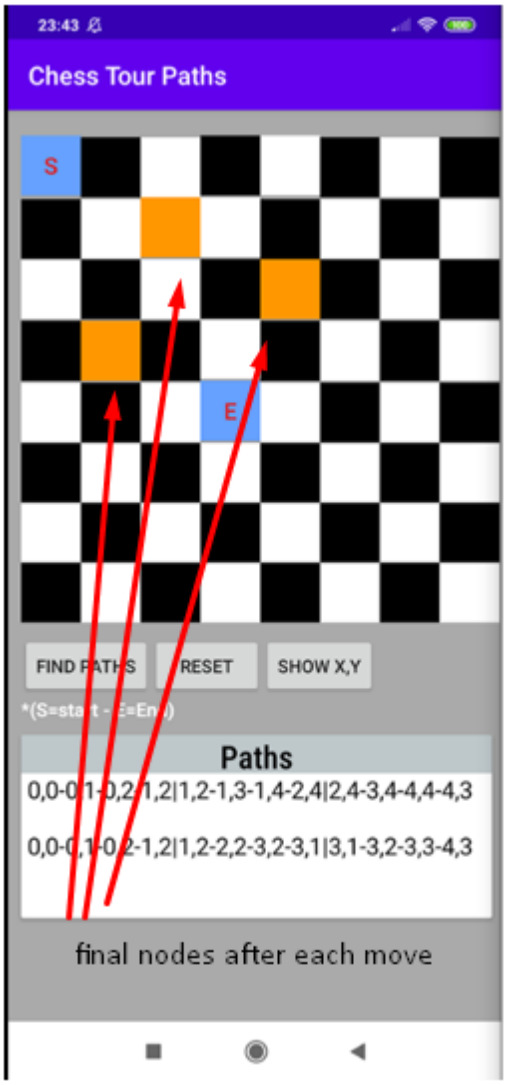
Γραμμές 182-188: Κρατάω μόνο τα μονοπάτια που τελειώνουν στο τελικό σημείο

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

SCHREENSHOT ΑΠΟ ΤΗΝ ΕΚΤΕΛΕΣΗ






23:43
Chess Tour Paths
S
E
FIND PATHS
RESET
SHOW X,Y
*(S=start - E=End)
Paths
0,0-0,1-0,2-1,2|1,2-1,3-1,4-2,4|2,4-3,4-4,4-4,3
0,0-0,1-0,2-1,2|1,2-2,2-3,2-3,1|3,1-3,2-3,3-4,3
final nodes after each move


23:42
Chess Tour Paths
S
0,1
0,2
0,3
0,4
0,5
0,6
0,7
1,0
1,1
1,2
1,3
1,4
1,5
1,6
1,7
2,0
2,1
2,2
2,3
2,4
2,5
2,6
2,7
3,0
3,1
3,2
3,3
3,4
3,5
3,6
3,7
4,0
4,1
4,2
E
4,4
4,5
4,6
4,7
5,0
5,1
5,2
5,3
5,4
5,5
5,6
5,7
6,0
6,1
6,2
6,3
6,4
6,5
6,6
6,7
7,0
7,1
7,2
7,3
7,4
7,5
7,6
7,7
FIND PATHS
RESET
SHOW X,Y
*(S=start - E=End)
Paths
0,0-0,1-0,2-1,2|1,2-1,3-1,4-2,4|2,4-3,4-4,4-4,3
0,0-0,1-0,2-1,2|1,2-2,2-3,2-3,1|3,1-3,2-3,3-4,3
after pressing this button