

Περιγραφή Κλάσεων:

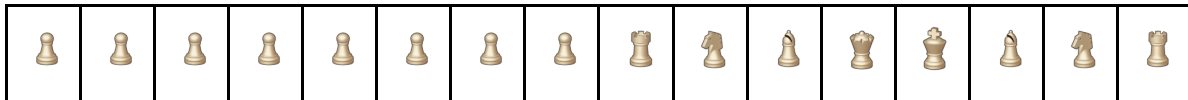
Class Spot: Περιγράφει κάθε κουτάκι της σκακιέρας
int x(0-7), y(0-7), pos(0 - 63), colour(Black or White)
boolean exist(αν βρίσκεται κάποιο πιόνι πάνω)
Piece type

Class Piece: Περιγράφει το πιόνι
int teamColour(Black or White), type(ROOKS, KNIGHT, BISHOPS, QUEEN, KING, PAWNS)
boolean isAlive
Spot position//σε ποια θέση βρίσκεται

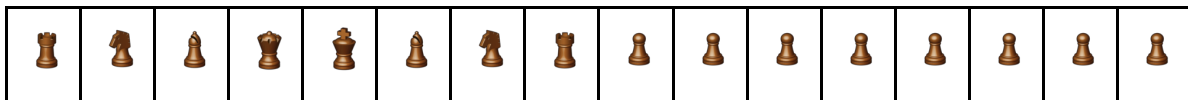
(κάποιες απο)μέθοδοι:
Kill(), Reincarnation() // χρειάζεται στην undo πράξη

Class TheBoard: Περιέχει όλη την κατάσταση του τρέχοντος παιχνιδιού και περιλαμβάνει όλες τις μεθόδους για την εξέλιξή του.
Piece[] teamWhite, teamBlack;
Spot[] board;
int maxDepth, MaxValue, MinValue;

teamWhite



teamBlack



board

κάθε θέση χαρακτηρίζεται από τους αριθμούς x,y (x= γραμμή, y=στήλη)ή από έναν αριθμό pos(όπου είναι ο αύξων αριθμός στον πίνακα ξεκινώντας μέτρημα από θέση (0,0) και πηγαίνοντας από αριστερά προς δεξιά και από πάνω προς τα κάτω)

0,0	0,1	...	0,7
1,0	1,1	...	1,7
....
7,0	7,7

μέθοδοι:

Constructor(int depth)

Γεμίζει κατάλληλα τους πίνακες για την αρχική κατάσταση

→ Κάθε κίνηση περιγράφεται από ένα String με το εξής format:

“x_source y_source x_dest y_dest | type *type f”

| **type**: αν απεσυρα κάποιο αντίπαλο πιόνι

***type**: αν το pawn έφτασε σε promotion spot

f: αν είναι η πρώτη κίνηση του rook ή του king για να ξέρω ότι πλέον το rook δεν είναι valid

“ROKE*position*LEFT/RIGHT” → περιγραφή ειδικής κίνησης ροκε, position θέση King.

MakeMove(String move)

UndoMove(String move)

list PossibleMoves(int team):

Δημιουργεί μια λίστα PossMoves

Σαρώνει τον πίνακα teamWhite/teamBlack ανάλογα με την μεταβλητή team

Για κάθε “ζωντανό” πιόνι καλώ αντίστοιχη συνάρτηση

Rook/Bishops/Pawns/King/Queen/KnightPossibleMoves για να επιστρέψουν μια λίστα με τις δυνατές τους κινήσεις.

Κάνω Merge τις λίστες και επιστρέφω την ένωσή τους.

list RookPossibleMoves(int team)

list BishopsPossibleMoves(int team)

list PawnsPossibleMoves(int team)

list KingPossibleMoves(int team)

list QueenPossibleMoves(int team)

list KnightsPossibleMoves(int team)

Επιστρέφουν μια λίστα από τις **νόμιμες** επόμενες (1 βήμα) κινήσεις. Η νομιμότητα ορίζεται ΚΑΙ από το γεγονός να μην αφήνουν τον βασιλιά τους εκτεθειμένο.

Για κάθε πιθανή κίνηση(move):

εφαρμόζεται (καλώντας την MakeMove(move))

ελέγχεται αν η KingSafe() είναι true, και μόνο τότε εισάγεται στην λίστα

καλώ την UndoMove(move) για να επανέλθω στην τωρινή κατάσταση του παιχνιδιού.

boolean check()

Ελέγχει αν οι λευκοί μπορούν να κάνουν κάποια κίνηση αν ναι επιστρέφει false αλλιώς true

String PcMove()

καλώ την CalculateNextMove(maxDepth, "", Black, MinValue, MaxValue)

Αν επιστρέψει "Lost"

επιστρέφω -1 //νίκησαν οι White

Αν επιστρέψει "Tie"

επιστρέφω -2 // οι Black είναι σε πατ, έχουμε ισοπαλία

Αλλιώς καλώ MakeMove με βάση την κίνηση που μου επέστρεψε και την επιστρέφω

Καλώ την check()

Αν επιστρέψει true:

Καλώ την KingSafe(White)

Αν επιστρέψει true:

επιστρέφω 0 // νίκησαν οι Black

Αν επιστρέψει false:

επιστρέφω -2 // οι White είναι σε πατ, έχουμε ισοπαλία

Αν επιστρέψει false:

επιστρέφω ότι επέστρεψε η calculateNextMove()

(int, String) CaculateNextMove(int depth, String move, int turn, int Calpha, int Cbeta)

//Calph κατώφλη για τους White που ενημερώνεται από Black

//Cbeta κατώφλη για τους Black που ενημερώνεται από White

String bmove=move; //best move

int local = MaxValue*turn + MinValue*(1 - turn)

list = PossibleMoves(turn)

Αν list.empty || depth == 0

Αν depth ==0 maxDepth

Αν kingSafe() επιστρέφω → (MinValue, Tie)

Αλλιώς επιστρέφω →(MinValue, Lost)

Αλλιώς επιστρέφω (raiting(list.size), depth), move) //έφτασα στο τέλος του δέντρου

Για κάθε cmove της λίστα

MakeMove(cmove)

result = CalculateNextMove(depth-1, cmove, 1 - turn, Calph, Cbeta)0

undoMove(cmove)

Αν turn == White //σκοπός MINIMIZE

Αν(local > result.score)

local = result.score

bmove = cmove

Αν local <= Calph // pruning my father wont accept any value smaller

return(local, bmove)

Cbeta = local // any other larger will be declined from my child

Αν turn == Black // σκοπός MAXIMIZE

Αν(local > result.score)

local = result.score

```

bmove = cmove
Av local >= Cbeta // pruning my father wont accept any value larger
    return(local, bmove)
Calpha = local // any other smaller will be declined from my child
return(local, bmove)

```

int Rating(int len, int depth)

Συνάρτηση τεσσάρων παραγόντων

(Η μέτρηση για κάθε παράγοντα γίνεται και για τις δύο ομάδες(εκτός από **rateMoveliy()**) ότι είναι υπέρ των Black ή εις βάρος των White προστίθεται στην σούμα, και αντίστροφα ότι είναι υπέρ των White ή εις βάρος των Black, αφαιρείται από άθροισμα)

- **rateMaterial()** βλέπω ποια είναι τα “ζωντανά” πιόνια της κάθε ομάδας και ανάλογα το πιόνι λαμβάνω πόντους (+ για Black, - για White)
- **rateMoveliy()** λαμβάνω πόντους ανάλογα με το len(πλήθος δυνατών κινήσεων, και ανάλογα με το αν depth == ζυγο αυξάνω αν μονός αριθμός είναι κατά μου αντίστροφη λογική αν len == 0, γιατί ο τρέχον παίκτης βρίσκεται σε κακή θέση)
- **rateAttack()** ελέγχω ποια πιόνια απειλούνται για κάθε Black πιόνι που απειλείται αφαιρώ, για κάθε white πιόνι που απειλείται αυξάνω.
- **ratePositional()**
το αρχείο Estimation_for_white έχει τις τιμές μιας Posional λογικής υπέρ αυτών που βρίσκονται στις τελευταίες θέσεις του πίνακα(White στην δική μας περίπτωση)
Οπότε εγώ για τους White χρησιμοποιώ τον αντίθετο του πίνακα και για τους Black τον ανάστροφο

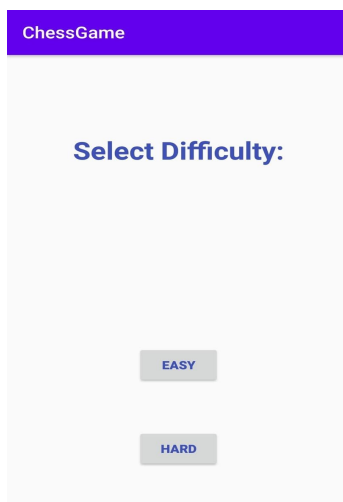
Οι μετρήσεις προσπαθούν να προσομοιώσουν μια Positonal chess τακτική

<https://chess.stackexchange.com/questions/9638/what-is-positional-play-in-chess>

--- Android ---

Class MainActivity και αντίστοιχο xml αρχείο:

Εμφανίζει την αρχική οθόνη του παιχνιδιού



Όπου ο χρήστης μπορεί να επιλέξει την δυσκολία τους παιχνιδιού.

Easy: depth = 2

Hard: depth = 4(με depth = 4 υπάρχει μια καθυστέρηση, οπότε σκεφτόμουν μήπως το βαλω με 3)



Class StartGame και αντίστοιχο xml αρχείο:



Layout:

Linear Layout



Κάθε ImageView σαν background photo έχουν  ή  ανάλογα την θέση το oneClick καλεί την Allowedmoves()

ImageView[64] interfaceBoard;

TheBoard game;

boolean firstClick;

map <int,string> prevClick

map<int,<int, string>> oldprom

μέθοδοι:

Allowedmoves()

Αν το firstClick== false

- τοτε κάνω με highlight όλες τις δυνατες κινήσεις που έχει το πioni που πάτησε ο χρήστη, τα κρατάω στο prevClick(κουτακι → κίνηση)

- Αν μέσα στις πιθανες κινήσεις έχει και την επιλογή του promotion τότε για κάθε κίνηση promotion που μπορεί να γίνει την τοποθετώ στο oldpro με κάθε δυνατή επιλογή

- εμφανίζω με highlight τις πιθανές κινήσεις.

Αν το firstClick== true

- Βλέπω αν το κουτακι που επέλεξε βρίσκεται στα prevClick και σε περίπτωση promotion εμφανίζω και το pop up μήνυμα για να μπορέσει ο χρήστης να επιλέξει το είδος του promotion

- καλώ την PcMove της κλάσης board και περιμένω αποτελέσματα για να ανανεώσω κατάλληλα και το interface

Class PrintBoard:

περιέχει την μέθοδο printBoard που σκανάρει τον πίνακα με τα pieces του παιχνιδιού και εμφανίζει τα αντίστοιχα πιόνια