Seminarska naloga 1
Iskanje šah matov pri 1-2-3 šahu

Danijel Maraž

Fakulteta za Računalništvo in Informatiko UL dm9929@student.uni-lj.si

Abstract. The article covers the work done in the scope of the first seminar assignment as part of the subject Algorithms.

Keywords: A* BFS Progressive Chess Heuristics Pathfinding

1 Uvod

Pri seminarski nalogi nam je bil dan cilj implementiranja algoritma A* (A zvezda) v namen iskanja potez, ki vodijo do šah mata pri dani postavitvi šahovskih figur. Kot jezik implementacije sem si izbral Python, ker menim, da se z njim da pisati lepšo in bolj berljivo kodo. Iz drugega vidika pa je tudi bistveno počasnejši od Jave. Če bi torej program pisal za trg bi ga zagotovo s pomočjo jezika, ki se hitreje izvaja.

2 Struktura programa

```
import chess
import sys
import random
import heapq
import time
from chess.polyglot import zobrist_hash

start = time.time()



def FCM():

        file = open(sys.argv[1], "r")
        fen0 = [line for line in file][0]
        fen0s = fen0.split(" ")
        avmoves=int(fen0.split(" ")[-1])
        startfen=fen0s[-3]+" "+fen0s[-2]+" "+" - - 0 0"
        board=chess.Board(startfen)
```

```
# Check who plays black or white
color=fen0s[1]
moveside = False
if color == "w":
    moveside = True
enemyside = not moveside

#print(startfen)

def KSCoverage(board, kingpos):
    return 100 * sum([len(board.attackers(moveside, a)) for a in
                    board.attacks(kingpos)
                    if board.is_attacked_by(moveside, a)])

def Man_dist(sq1, sq2):
    return abs(chess.square_file(sq1) - chess.square_file(sq2)) + abs(ch

def ManhattanLight(move,kingpos):
    return 100 * (Man_dist(move.to_square, kingpos)-1)

def BFS_move(brd, c):
    visited = set()
    nextq = [(0, c, id(brd),brd)]
    heapq.heapify(nextq)
    while nextq:

        if (time.time()-start) > 30:
            break
        current = heapq.heappop(nextq)
        current[3].turn = moveside

        zh = zobrist_hash(current[3])
        visited.add(zh - current[1])
       # Check if the given position is a checkmate
        if current[1] == 0:
            current[3].turn = enemyside
            if current[3].is_checkmate():
               #print(current[3].fen())
               sstr=""
               moveslist=[str(m) for m in current[3].move_stack]
               for move in moveslist:
                   sstr+=move[:2]+"-"+move[2:]+";"
               print(sstr[:-1])
               break
            current[3].turn = moveside
            continue

       # The main loop
        for move in current[3].legal_moves:

            current[3].turn = moveside
            sc = current[1]

            current[3].push(move)
            current[3].turn = moveside

            # Eliminate premature attacks on the king
            if sc > 1 and current[3].was_into_check():
               current[3].pop()
               continue
            # Eliminate last moves that don't attack the king
            if sc == 1 and not current[3].was_into_check():
               current[3].pop()
               continue
            # Keep the king from being captured
            #if current[3].king(enemyside) is None:
               #print("king capture keeping")
               #current[3].pop()
               #continue

            # Check if we already saw this board before
            zh = zobrist_hash(current[3])
            if zh - (sc-1) in visited:
               current[3].pop()
               continue

            r = 0
            # Reward promotions
            prom = move.promotion
            if prom is not None:
               r -= -200
               if prom == chess.Piece(5,moveside):
                   r -= 300
               if prom == chess.Piece(4, moveside):
                   r -= 200


            # Desperate measures
            rand = 0
```

```
                        if (time.time() - start) > 17:
                            rand = random.randint(1, 20)
                        r =+ rand

                        # Reward last turn attacks on the king
                        if sc == 1 and current[3].was_into_check():
                            r -= 1000

                        # Apply various heuristics

                        # Coverage of squares around the King
                        r -= KSCoverage(current[3],board.king(enemyside))
                        # Manhattan distance from all non pawn figures to the King
                        r += ManhattanLight(move, board.king(enemyside))

                        # Reward depth
                        r -= (current[1]-1)*100

                        ccopy = current[3].copy()
                        current[3].pop()

                        heapq.heappush(nextq, (r,current[1]-1,id(ccopy),ccopy))

        BFS_move(board, avmoves)

FCM()
```

3 Hevristike

4 Ovrednotev

References

Bonča, J. (2015a). *Sudoku 9x9-15.* Retrieved 2019-01-13, from https://i.pinimg.com/564x/4c/17/f2/4c17f2454b20fa3200882047d1722684.jpg

Bonča, J. (2015b). *Tipkopisi.* Retrieved 2019-01-13, from http://likovnodrustvo-kranj.weebly.com/gostujo269e-razstave/jaka-bonca-tipkopisi

SocialBladeLLC. (2019). *Youtube social blade stats.* Retrieved 2019-01-5, from https://socialblade.com/youtube/

Welbourne, D. J., & Grant, W. J. (2016). Science communication on youtube: Factors that affect channel and video popularity. *Public Understanding of Science*, *25*(6), 706–718.