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

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

            #Pop best rated board
        current = heapq.heappop(nextq)
        current[3].turn = moveside

            #Add board to visited
        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():
                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
```

Funkcija *BFS_move* prejme kot argumenta *brd* (board razred Python šahovkse knjižnice inicializiran iz fen notacije posamezne testne datoteke) in *c* (koliko potez ima program na voljo). Takoj za tem ustvarimo množico *visited*, ki hrani zgoščene slike že videnih šahovnic ter min kopico *nextq*. Ta hrani terke oblike *(ocena, št. potez, identifikator, šahovnica)* in jih med seboj razporeja glede na prvi element. Nato se začne prva while zanka, ki se izvaja vse dokler ne najdemo šah mata ali dokler ni kopica *nextq* prazna. Nadalje iz kopice pridobimo najnižje ocenjeno terko, zgoščeno sliko njene šahovnice dodamo k množici *visited* in ob pogoju, da ta nima več na voljo potez preverimo ali smo prišli do šah mata.

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

    sc = current[1]

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

    # Eliminate premature attacks on the king
    # Eliminate last moves that don't attack the king
    wic = current[3].was_into_check()
    if (sc > 1 and wic) or (sc == 1 and not wic):
        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
```

Z zgornjo zanko nato pregledamo zalogo vseh dovoljenih potez in marsikatero že na osnovi osnovnih pravil 1-2-3 šaha izločimo. Potezo porinemo na sklad potez izbrane šahovnice in s pomočjo funkcije *was_into_check()* preverimo ali je bil kralj napaden pred zadnjo potezo in ali smo prišli do zadnje poteze brez, da bi napadli kralja. V obeh primerih se poteza odstrani iz sklada ter zanka vrne na začetek. Potezo prav tako preskočimo, če s pomočjo množice *visited* ugotovimo, da smo šahovnico že obiskali pri danem številu potez. Bistvene izboljšave na tem področju:

– Najprej smo za vsako novo potezo naredili kopijo šahovnice in na njej izvajali preverjanja kar se je izkazalo kot zelo potratno, saj je kopiranje celotne instance razreda počasna operacija. Boljša alternativa je bila posamezne poteze preprosto poriniti na sklad že obstoječe instance.
– Smiselno bi bilo, da bi takoj preverili ali smo šahovnico že videli, ampak je v praksi operacija zgoščevanja izjemno počasna. Izkazalo se je da je veliko hitrejše poteze najprej izločiti glede na pravila igre kot pa za vsako potezo računati zgoščevalno funkcijo.

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

# Reward last turn attacks on the king
if sc == 1:
    r -= 1000

# Apply various heuristics

# Coverage of squares around the King
r -= KSCoverage(current[3], board.king(enemyside))
# Manhattan distance from the new position of the piece that moved
r += ManhattanLight(move, board.king(enemyside))

# Reward depth
r -= (sc - 1) * 100

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

heapq.heappush(nextq, (r, sc - 1, id(ccopy), ccopy))
```

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.