Veštačka inteligencija

Projekat – Dominacija (Domineering)

Faza III – Min-max algoritam i heuristika

Stefan Stojadinović 17975, Andrija Tošić 18015

Min-Max algoritam sa alfa-beta odsecanjem za zadati problem (igru)

Minmax algoritam implementiran je u narednim metodama:

```
def alfabeta(
      state: State,
      depth: int,
      alpha: float,
      beta: float,
      tt: TranspositionTable
) -> tuple[Move, int]:

def alfabeta_bt(
      state: State,
      depth: int,
      alpha: float,
      beta: float,
      tt: TranspositionTable
) -> tuple[Move, int]:
```

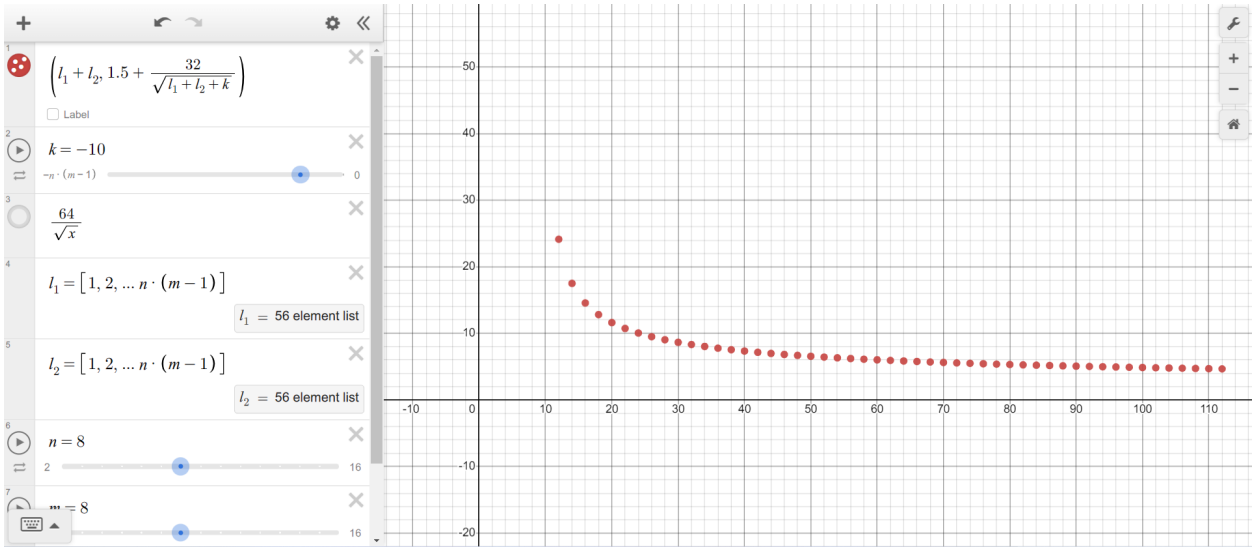
Razlika u implementaciji `alfabeta_bt` je što se umesto kreiranja novog stanja za svaki od čvorova u razvoju podstabla modifikuje postojeće stanje, a potom se ista ta instanca stanja modifikuje u suprotnom smeru pri povratku iz rekurzije (undo). Razlog postojanja `alfabeta_bt` rešenja je to što je čak nekoliko puta brže od `alfabeta`. Radi implementacije ove funkcionalosti dodate su i metode:

```
      def modify_state(state: State, move: Move) -> State:
      def undo_move(state: State, move: Move) -> State:
```

Radi ubrzanja alfabeta pretrage dodatno se koristi transpoziciona tabela koja je implementirana u klasi `TranspositionTable`.

Radi povećavanja šanse da se desi alfa-beta odsecanje, sledeća moguća stanja se pre obrade evaluiraju i sortiraju po heuristici.

Dubina min-max algoritma je dinamički određena sledećom funkcijom na osnovu zbira broja preostalih poteza za oba igrača.

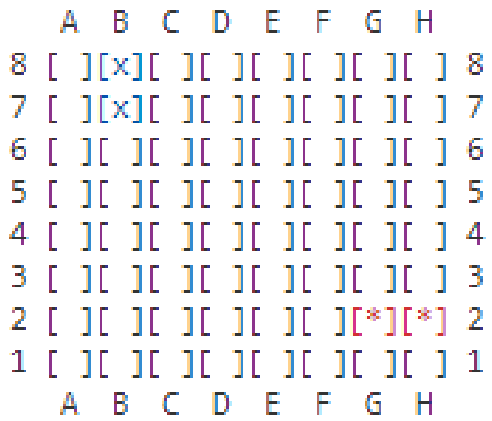



```
def dynamic_depth(state: State) -> int:

    rm = len(state.h_possible_moves) + len(state.v_possible_moves)

    return int(1.5 + 32 / math.sqrt(max(rm - 10, 10)))
```

Funkcija koja vrši procenu stanja na osnovu pravila zaključivanja

Funkcija evaluacije predstavlja linearnu kombinaciju broja preostalih poteza igrača i broja “izolovanih” poteza, odnosno poteza koje suparnički igrač ne može zablokirati.

U narednoj poziciji 8A predstavlja izolovani potez za vertikalnog, dok je 1G izolovani potez za horizontalnog igrača.




```python
def evaluate_state(state: State) -> int:
    if is_game_over(state):
        return 1000 if state.to_move is Turn.HORIZONTAL else -1000

    value = 0

    v_im_count, h_im_count = count_isolated_moves(state)

    value += 3 * (v_im_count - h_im_count)
    value += 2 * (len(state.v_possible_moves) - len(state.h_possible_moves))

    return value
```

Funkcija koja na osnovu zadatog stanja vraća broj izolovanih poteza za vertikalnog i horizontalnog igrača:

```python
def count_isolated_moves(state: State) -> tuple[int, int]:
```