Technická dokumentácia
k programu Piškvorky s metódou Minimax a Alfa–Beta orezávaním

Alžbeta Valachová

Vysoké učení technické v Brne
Fakulta strojního inženýrství
Ústav automatizace a informatiky
10. 5. 2022

1 Minimax

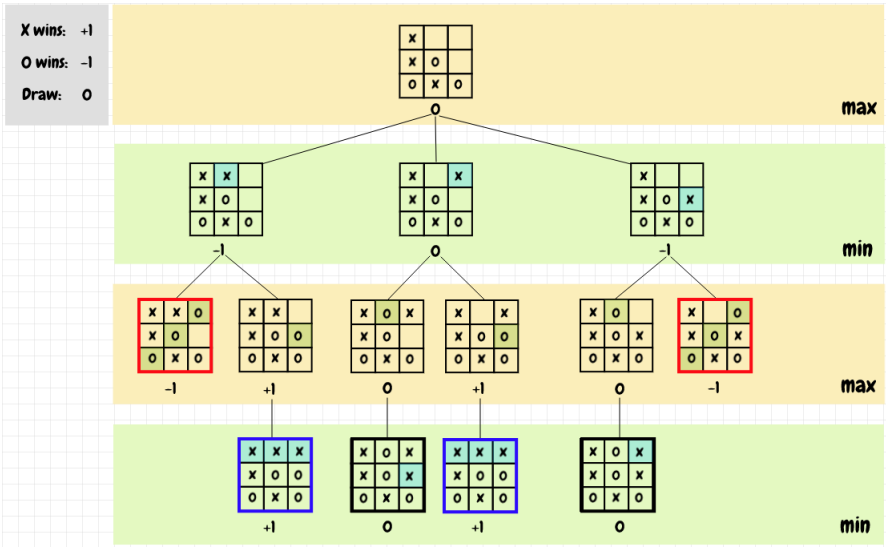
Minimax je *backtracking* algoritmus, ktorý sa používa v teórii hier na rozhodovanie optimálneho ťahu pre hráča za predpokladu, že protihráč hrá optimálne. Dá sa použiť v hrách pre dvoch hráčov ako piškvorky, backgammon aj šach. [2]

V Minimaxe sa hráči nazývajú *maximizer* a *minimizer*. *Maximizer* sa snaží získať čo najvyššie skóre, zatiaľ čo *minimizer* naopak najnižšie. Každý stav hracej plochy má príslušné skóre. Pokiaľ má navrch *maximizer*, skóre bude kladné a pokiaľ *minimizer*, tak skóre bude záporné. Skóre je počítané pomocou heurestickej funkcie, ktorej tvar závisí od konkrétnej hry.

Na obrázku Obr. 1 sa nachádza stav hracej plochy piškvoriek vo veľkosti 3x3. Algoritmus zistí ohodnotenie koncových listových uzlov. V tomto názornom prípade veľmi jednoducho:

Výhra **X** : 1
Výhra **O** : -1
Remíza: 0

Z nich vyberie najprv minimum a posunie sa v hĺbke stromu o jedno vyššie. Tam vyberie maximum a opäť sa posunie. Potom minimum až nakoniec znova maximum. Vďaka poslednému výberu vie, ktorou cestou kombinácii rozhodnutí sa má vybrať, aby vyhral pri optimálnom hraní druhého hráča (O).



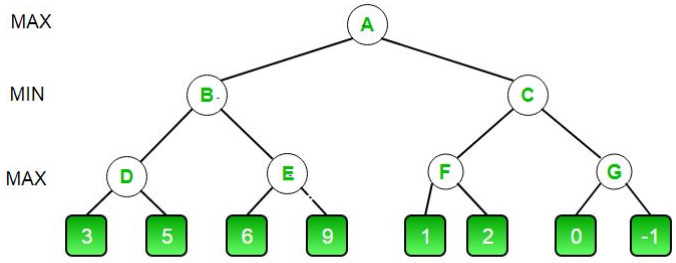
Obr. 1: *Piškvorky 3x3* [1]

1.1 Alfa-Beta orezávanie

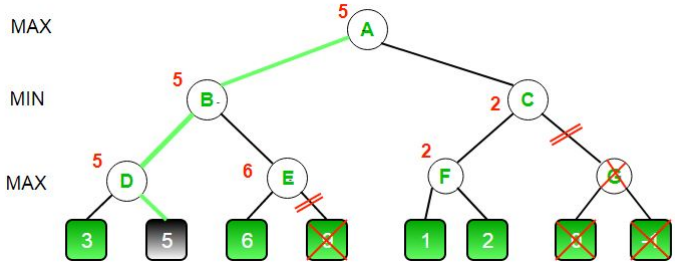
Pokiaľ je hra zložitejšia, Minimax nie je najideálnejší algoritmus na použitie. Ide o prehľadávanie do hĺbky, ktoré môže byť v niektorých prípadoch náročné na čas aj pamäť. Preto sa vyvinula akási optimalizácia Minimaxu – Alfa-Beta orezávanie. Výrazne redukuje čas výpočtu. Vďaka tomu umožňuje ísť hlbšie v stavovom priestore a neberie do úvahy vetvy stromu, ktorú sú neperspektívne. Oproti Minimaxu naviac obsahuje parametre Alfa

a Beta. Alfa je najlepšia hodnota, ktorú *maximizer* môže zaručiť na aktuálnej hĺbke alebo vyššej a Beta je najlepšia hodnota, ktorú *minimizer* môže zaručiť na aktuálnej hĺbke alebo vyššej. [3]

Na Obr.2 sa nachádza aktuálny stav hry. Na Obr.3 sa aplikovala procedúra Alfa-Beta. Pre koncové listové uzly sa najprv zistili ich heurestické hodnoty. V prvom kroku sme vybrali najvyššie skóre a presunuli sme sa v hĺbke o jednu vyššie. Vybrali sme teraz najnižšie skóre a znova sme sa presunuli a takto to pokračovalo až do koreňového uzlu. Uzly, ktoré sa nevybrali, boli označené za neperspektívne, a vyškrtli sa.



Obr. 2: *Strom reprezentujúci stav hry* [3]



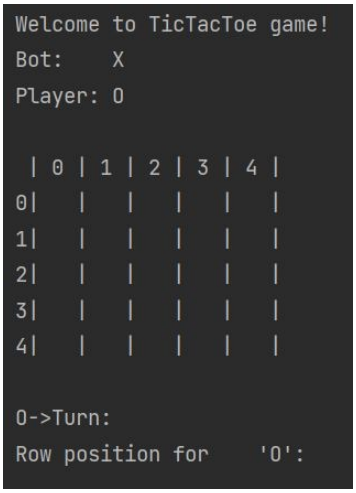
Obr. 3: *Alfa-Beta orezanie neperspektívnych uzlov* [3]

1.2 Programová implementácia

V tejto kapitole sa pozrieme na programovú implemetáciu metódy Minimax Alfa-Beta na piškvorkách s hracou plochou veľkosti 5x5 v Pythone. Uvažujeme prípad, keď sú na výhru potrebné 4 znaky v riadku, stĺpci alebo diagonále. Pokiaľ by sa na implentáciu využil iba Minimax a v prvom kroku by začínal počítač (X), mohol by vyberať z 15×10^{24} možností. V tomto prípade je teda Alfa-Beta skoro nevyhnutná.

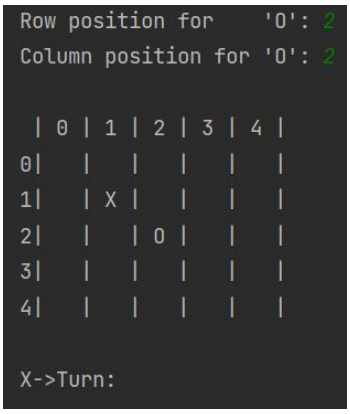
1.2.1 Spustenie

Prvá vec, ktorú program vypíše po spustení, je prázdna hracia plocha. Riadky a stĺpce sú indexované príslušnými číslami, ktoré sa nachádzajú po bokoch plochy. Pomocou funkcie `random.choice([1, 2])` sa vyberie, či má prvý ťah bot alebo človek.



Obr. 4: *Začiatok hry*

Pokiaľ je na ťahu človek, vstupnú pozíciu zadáva z klávesnice ako index pre riadky a index pre stĺpce. Potom sa vykreslí hracia plocha s týmto ťahom a čaká sa na odpoveď Bota. Hra skončí, pokiaľ vyhral jeden z hráčov alebo nastala remíza.



Obr. 5: *Výber políčka*

1.2.2 Použité triedy

`Board` je trieda, ktorá definuje hraciu plochu. Obsahuje nasledujúci konštruktor:

```
class Board:
 def __init__(self):
   self.size = 5 #board is size 5x5
   self.line = [Empty for _ in range(self.size**2)]#1D
   self.square = [list(Empty*self.size) for _ in range(self.size)]#2D
```

Konštruktor generuje 1D a 2D pole, do ktorého sa ukladajú hodnoty políčok v hracej ploche. Políčko môže obsahovať hodnoty ' ','X','O'.

Trieda obsahuje množstvo ďalších procedúr, ktoré sú potrebné pre výpočty:

`isEmptyLine` – kontrola, či políčko je prázdne pre 1D
`isEmptySquare` – kontrola, či políčko je prázdne pre 2D

`printBoard` – vykreslí aktuálnu hraciu plochu
`convert1d_2d` – prepočíta 1D hodnotu na 2D
`convert2d_1d` prepočíta 2D hodnotu na 1D
`Board2D` – prepočíta 1D pole políčok na hraciu plochu v 2D
`getRow` – vráti riadkový index z 2D poľa
`getColumn` – vráti stĺpcový index z 2D poľa
`getMainDiagonal` – vráti dve hlavné diagonály hracej plochy
`getOtherDiagonal`– vráti štyri vedľajšie diagonály hracej plochy
`calculateLine` – spočíta koľko políčok má bot/človek alebo sú prázdne v riadku/stĺpci/diagonále

Veľmi dôležité sú procedúry, ktoré počítajú skóre pre aktuálne stavy hracích ploch. Prvá počíta skóre v riadkoch, stĺpcoch a hlavných diagonálach podľa toho, aké elementy sa v nich nachádzajú:

```
def getScoreLine4_5imp(self, line): # heurestic function
  score = 0
  botSum, humSum, emptySum = self.calculateLine(line)
  #Score for Bot
  if botSum >= 4 and (line[0] == Bot or line[self.size-1] == Bot): # Bot
                                        wins
    score += 20 ** botSum
  elif humSum == 0 and botSum != 0: # Human "almost" wins
    score += 20 ** botSum
  elif humSum == 1 and (line[0] == Human or line[self.size-1] == Human):
                                        # Bot near to win
    score += 8 ** botSum
  else: # Bot is in line without being near to winning
    score += 2 ** botSum

  # Score for Human
  if humSum >= 4 and (line[0] == Human or line[self.size - 1] == Human):
                                         # Human wins
    score -= 20 ** humSum
  elif botSum == 0 and humSum != 0:  # Human "almost" wins
    score -= 20 ** humSum
  elif botSum == 1 and (line[0] == Bot or line[self.size - 1] == Bot):
                                       # Human near to win
    score -= 8 ** humSum
  else:  # Human is in line without being near to winning
    score -= 2 ** humSum
  return score
```

Ďalšia procedúra, ktorá počíta skóre podľa elementov pre vedľajšie diagonály.

```
def getScoreLine4_4imp(self, line): # heurestic function
  score = 0
  botSum, humSum, emptySum = self.calculateLine(line)
  if humSum == 0 and botSum == self.size-1: # Bot wins
    score += 20 ** botSum
  elif humSum == 0: # Bot wins
    score += 10 ** botSum
  else:
    pass
  if botSum == 0 and humSum == self.size - 1:  # Human wins
    score -= 20 ** humSum
  else:
```

```
    pass
  return score
```

Nakoniec funkcia `evaluate` spočíta skóre pre všetky riadky, stĺpce a diagonály.

Druhá trieda je trieda `SmartPlayer`. Definuje pohyb inteligentného bota. Obsahuje tieto procedúry:

`__init__` – koštruktor, nastaví maximálnu hĺbku pre Minimax 5
`turn` – ťah bota, používa Minimax s Alfa-Beta orezávaním

Procedúra `minimax_ab` používa Minimax algoritmus s Alfa-Beta orezávaním do hĺbky 5. Pre ťah bota je hodnota Alfy nastavená na -1000000 a Bety na 1000000. Na začiatku Minimaxu sa získa skóre aktuálnej hracej plochy. Bot bude vždy *maximazer* a človek *minimizer*. Kontroluje sa, či niekto vyhral alebo nastala remíza. Pokiaľ sa Minimax dostane na hĺbku 0, vyskočí, aj keď nikto nevyhral, respektívne nenastala remíza. Ďalej sa vyhodnocujú ťahy. Pokiaľ ide o bota, začnú sa prechádzať voľné políčka rekurzívnym Minimaxom. Potom sa kontroluje, či je skóre hracej plochy väčšie ako Alfa. Ak áno, tak sa Alfa nastaví na aktuálne skóre a aktuálna hracia pozícia sa uloží. Ťah aj možný víťaz sa potom vymaže. Pokiaľ je Beta väčšie ako Alfa, Minimax ďalej neprehľadáva ďalšie voľné políčka. Pokiaľ ide o človeka, postup je veľmi podobný s malými zmenami. Pre pozíciu získanú Minimaxom sa kontroluje, či je skóre menšie ako Beta. Ak áno, Beta sa nastaví na toto skóre a pozícia sa uloží. Z procedúry sa vyskočí, ak je Alfa väčšia ako Beta.

```
def minimax_ab(self, gamestate, depth, player, alpha, beta):

  score = gamestate.evaluate() # gets the score from the actual board
  position = None

  max_player = Bot  # yourself
  if player == Bot:
    other_player = Human
  else:
    other_player = Bot

  gameResult = game.CheckWin4() #checks if there is a winner
  if gameResult == Human:
    return -20**(gamestate.size-1)-gamestate.numberEmptySpaces()*
                                    gamestate.size, position
  elif gameResult == Bot:
    return 20**(gamestate.size-1)+gamestate.numberEmptySpaces()*
                                    gamestate.size, position
  elif gamestate.checkFullBoard():
    return 0, position

  if depth == 0: # if minimax reaches depth limit -> returns the score
    return score, position

  if player == max_player:
    for i in gamestate.availableMoves():
      gamestate.makeMove(i, player)
      score, dummy = self.minimax_ab(gamestate, depth-1, other_player,
                                    alpha, beta)
```

```
      if score > alpha:
        alpha = score
        position = i

      gamestate.line[i] = Empty # undo move
      gamestate.current_winner = None # undo winner

      if beta <= alpha:
        break

    return alpha, position

  else:
    for i in gamestate.availableMoves():
      gamestate.makeMove(i, player)
      score, dummy = self.minimax_ab(gamestate, depth-1, other_player,
                                     alpha, beta)
      if score < beta:
        beta = score
        position = i

      gamestate.line[i] = Empty
      gamestate.current_winner = None
      if alpha >= beta:
        break

    return beta, position
```

Tretia trieda je trieda `Player`. Definuje ťahy človeka. Obsahuje konštruktor a procedúru `turn`, ktorá sa zavolá, pokiaľ je na ťahu človek a vypýta si vstupné indexy do hracej plochy, ktoré človek zadá z klávesnice.

Posledná je trieda `TicTacToe`. Táto trieda je odvodená od triedy `Board`. Obsahuje nasledujúce procedúry:

`__init__` – konštruktor
`setPlayer`
`startGame` – výpis začiatku hry a náhodný výber prvého hráča
`checkFullBoard` – kontroluje, či je hracia plocha plná
`CheckWin4` kontrola výhry – 4 elementy pri sebe
`availableMoves`
`makeMove`
`gameOver` kontroluje, či nastala vyhra/remíza
`changePlayer` – zmena hráča

Hlavný program obsahuje cyklus `while`, ktorý skončí, pokiaľ nastala výhra nejakého hráča alebo remíza. Ak je na ťahu človek, zavolá sa príslušná procedúra. Ak je na rade bot, optimálny ťah sa spočíta pomocou Minimaxu s Alfa-Beta. Po každom ťahu sa vykreslí aktuálna hracia plocha a zmení sa hráč, ktorý je na ťahu.

Literatúra

[1] *A simple Tic Tac Toe* [online]. [cit. 2022-05-22]. Dostupné z: https://github.com/jason-shepherd/tictactoe

[2] *Minimax Algorithm in Game Theory — Set 1 (Introduction)* [online]. [cit. 2022-05-22]. Dostupné z: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

[3] *Minimax Algorithm in Game Theory — Set 4 (Alpha-Beta Pruning)* [online]. [cit. 2022-05-22]. Dostupné z: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/