Aula7 - Exercício prático jogos minimax e poda alfa-beta

October 10, 2023

1 Aula7 - Exercício prático jogos minimax e poda alfa-beta

Vitor Albuquerque de Paula

1.1 Respostas às Perguntas Iniciais

1.1.1 a) Qual o tipo de jogo?

O jogo em questão é o Xadrez, um jogo de estratégia entre dois jogadores que envolve movimentos turn-based das peças em um tabuleiro 8x8.

1.1.2 b) Quais as técnicas empregadas?

O código implementa o algoritmo Minimax para navegação na árvore de jogadas, buscando escolher a melhor jogada possível.

1.1.3 c) Quais outras estratégias foram usadas além dessas vistas em aula?

O algoritmo faz uso do motor Stockfish para avaliação de posições, o código implementa o algoritmo Minimax para navegação na árvore de jogadas, buscando escolher a melhor jogada possível. O algoritmo Minimax também é utilizado com a técnica de poda alfa-beta, que reduz o número de nós que o algoritmo precisa avaliar.

1.1.4 d) Analise o código, e vá colocando comentários em português em markdown sobre a explicação do código no notebook.

Continuarei fazendo isso nas próximas etapas.

Tive um problema com o docker e nao consegui rodar em tempo, por isso me concentrei em fazer uma analise mais detalhada do codigo.

1.2 Importando Bibliotecas

A célula abaixo importa todas as bibliotecas necessárias para o projeto. A biblioteca `chess` é usada para lidar com o tabuleiro e as peças de xadrez, enquanto `chess.engine` fornece funcionalidades para trabalhar com motores de xadrez. A biblioteca `time` é usada para controle de tempo, `chess.svg` para trabalhar com representações SVG do tabuleiro e, finalmente, `IPython.display` é usada para exibir o tabuleiro SVG no notebook.

[1]:
```
import chess
import chess.engine
```

```
import time
import chess.svg
from IPython.display import SVG, display
```

1.3 Inicializando o Motor de Xadrez

Aqui, o motor de xadrez Stockfish é inicializado. O Stockfish é um motor de xadrez open source amplamente utilizado para análise de jogos de xadrez e desenvolvimento de jogadores automáticos (bots). O caminho fornecido como argumento indica o local do executável do Stockfish no sistema.

[2]:
```
engine = chess.engine.SimpleEngine.popen_uci("./stockfish_13_linux_x64_bmi2")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[2], line 1
----> 1 engine =␣
 ↪chess.engine.SimpleEngine.popen_uci("./stockfish_13_linux_x64_bmi2")

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\site-packages\chess\engin
 ↪py:3054, in SimpleEngine.popen_uci(cls, command, timeout, debug, setpgrp,␣
 ↪**popen_args)
   3048 @classmethod
   3049 def popen_uci(cls, command: Union[str, List[str]], *, timeout:␣
 ↪Optional[float] = 10.0, debug: bool = False, setpgrp: bool = False,␣
 ↪**popen_args: Any) -> SimpleEngine:
   3050     """
   3051     Spawns and initializes a UCI engine.
   3052     Returns a :class:`~chess.engine.SimpleEngine` instance.
   3053     """
-> 3054     return␣
 ↪cls.popen(UciProtocol, command, timeout=timeout, debug=debug, setpgrp=setpgrp  **popen_args

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\site-packages\chess\engin
 ↪py:3046, in SimpleEngine.popen(cls, Protocol, command, timeout, debug,␣
 ↪setpgrp, **popen_args)
   3043         simple_engine.close()
   3044     await simple_engine.shutdown_event.wait()
-> 3046 return␣
 ↪run_in_background(background, name=f"{cls.__name__} (command={command!r})", d bug=debug)

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\site-packages\chess\engin
 ↪py:201, in run_in_background(coroutine, name, debug, _policy_lock)
    198         future.set_exception(exc)
    200 threading.Thread(target=background, name=name).start()
--> 201 return future.result()

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\concurrent\futures\_base.
 ↪py:444, in Future.result(self, timeout)
```

```
    442     raise CancelledError()
    443 elif self._state == FINISHED:
--> 444     return self.__get_result()
    445 else:
    446     raise TimeoutError()

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\concurrent\futures\_base.
 ↪py:389, in Future.__get_result(self)
    387 if self._exception:
    388     try:
--> 389         raise self._exception
    390     finally:
    391         # Break a reference cycle with the exception in self._exception
    392         self = None

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\site-packages\chess\engin
 ↪py:195, in run_in_background.<locals>.background()
    193 def background() -> None:
    194     try:
--> 195         asyncio.run(coroutine(future))
    196         future.cancel()
    197     except Exception as exc:

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\asyncio\runners.py:44, in
 ↪run(main, debug)
     42     if debug is not None:
     43         loop.set_debug(debug)
---> 44     return loop.run_until_complete(main)
     45 finally:
     46     try:

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\asyncio\base_events.py:
 ↪616, in BaseEventLoop.run_until_complete(self, future)
    613 if not future.done():
    614     raise RuntimeError('Event loop stopped before Future completed.')
--> 616 return future.result()

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\site-packages\chess\engin
 ↪py:3034, in SimpleEngine.popen.<locals>.background(future)
   3033 async def background(future: concurrent.futures.Future[SimpleEngine]) -
 ↪None:
-> 3034     transport, protocol = await Protocol.popen(command, setpgrp=setpgrp
 ↪**popen_args)
   3035     threading.current_thread().name = f"{cls.__name__} (pid={transport.
 ↪get_pid()})"
   3036     simple_engine = cls(transport, protocol, timeout=timeout)
```

```
File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\site-packages\chess\engin.
 ↪py:1320, in Protocol.popen(cls, command, setpgrp, **popen_args)
   1316     except AttributeError:
   1317         # Unix.
   1318         popen_args["start_new_session"] = True
-> 1320 return await asyncio.get_running_loop().subprocess_exec(cls, *command,␣
 ↪**popen_args)

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\asyncio\base_events.py:
 ↪1630, in BaseEventLoop.subprocess_exec(self, protocol_factory, program, stdin␣
 ↪stdout, stderr, universal_newlines, shell, bufsize, encoding, errors, text,␣
 ↪*args, **kwargs)
   1628     debug_log = f'execute program {program!r}'
   1629     self._log_subprocess(debug_log, stdin, stdout, stderr)
-> 1630 transport = await self._make_subprocess_transport(
   1631     protocol, popen_args, False, stdin, stdout, stderr,
   1632     bufsize, **kwargs)
   1633 if self._debug and debug_log is not None:
   1634     logger.info('%s: %r', debug_log, transport)

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\asyncio\windows_events.py
 ↪389, in ProactorEventLoop._make_subprocess_transport(self, protocol, args,␣
 ↪shell, stdin, stdout, stderr, bufsize, extra, **kwargs)
    385 async def _make_subprocess_transport(self, protocol, args, shell,
    386                                       stdin, stdout, stderr, bufsize,
    387                                       extra=None, **kwargs):
    388     waiter = self.create_future()
--> 389     transp = _WindowsSubprocessTransport(self, protocol, args, shell,
    390                                          stdin, stdout, stderr, bufsize
    391                                          waiter=waiter, extra=extra,
    392                                          **kwargs)
    393     try:
    394         await waiter

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\asyncio\base_subprocess.p:
 ↪36, in BaseSubprocessTransport.__init__(self, loop, protocol, args, shell,␣
 ↪stdin, stdout, stderr, bufsize, waiter, extra, **kwargs)
     34 # Create the child process: set the _proc attribute
     35 try:
---> 36     self._start(args=args, shell=shell, stdin=stdin, stdout=stdout,
     37                 stderr=stderr, bufsize=bufsize, **kwargs)
     38 except:
     39     self.close()

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\asyncio\windows_events.py
 ↪885, in _WindowsSubprocessTransport._start(self, args, shell, stdin, stdout,␣
 ↪stderr, bufsize, **kwargs)
    884 def _start(self, args, shell, stdin, stdout, stderr, bufsize, **kwargs)
--> 885     self._proc = windows_utils.Popen(
```

```
    886         args, shell=shell, stdin=stdin, stdout=stdout, stderr=stderr,
    887         bufsize=bufsize, **kwargs)
    889     def callback(f):
    890         returncode = self._proc.poll()

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\asyncio\windows_utils.py:
 ↪153, in Popen.__init__(self, args, stdin, stdout, stderr, **kwds)
    151     stderr_wfd = stderr
    152 try:
--> 153     super().__init__(args, stdin=stdin_rfd, stdout=stdout_wfd,
    154                      stderr=stderr_wfd, **kwds)
    155 except:
    156     for h in (stdin_wh, stdout_rh, stderr_rh):

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\subprocess.py:858, in␣
 ↪Popen.__init__(self, args, bufsize, executable, stdin, stdout, stderr,␣
 ↪preexec_fn, close_fds, shell, cwd, env, universal_newlines, startupinfo,␣
 ↪creationflags, restore_signals, start_new_session, pass_fds, encoding, errors␣
 ↪text)
    854         if self.text_mode:
    855             self.stderr = io.TextIOWrapper(self.stderr,
    856                     encoding=encoding, errors=errors)
--> 858     self._execute_child(args, executable, preexec_fn, close_fds,
    859                         pass_fds, cwd, env,
    860                         startupinfo, creationflags, shell,
    861                         p2cread, p2cwrite,
    862                         c2pread, c2pwrite,
    863                         errread, errwrite,
    864                         restore_signals, start_new_session)
    865 except:
    866     # Cleanup if the child failed starting.
    867     for f in filter(None, (self.stdin, self.stdout, self.stderr)):

File ~\AppData\Local\anaconda3\envs\redes_neurais\lib\subprocess.py:1327, in␣
 ↪Popen._execute_child(self, args, executable, preexec_fn, close_fds, pass_fds,
 ↪cwd, env, startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread,␣
 ↪c2pwrite, errread, errwrite, unused_restore_signals, unused_start_new_session
   1325 # Start the process
   1326 try:
-> 1327     hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
   1328                              # no special security
   1329                              None, None,
   1330                              int(not close_fds),
   1331                              creationflags,
   1332                              env,
   1333                              cwd,
   1334                              startupinfo)
   1335 finally:
   1336     # Child is launched. Close the parent's copy of those pipe
```

```
   1337     # handles that only the child should have open.  You need
   (…)
   1340     # pipe will not close when the child process exits and the
   1341     # ReadFile will hang.
   1342     self._close_pipe_fds(p2cread, p2cwrite,
   1343                          c2pread, c2pwrite,
   1344                          errread, errwrite)

FileNotFoundError: [WinError 2] The system cannot find the file specified
```

1.4 Função de Avaliação do Stockfish

A função `stockfish_eval` usa o motor Stockfish para avaliar a posição atual do tabuleiro. Ela aceita como parâmetros uma instância do tabuleiro e a profundidade de análise desejada (embora a profundidade não seja utilizada na função conforme definida). A função analisa o tabuleiro e retorna um score que indica a vantagem relativa das peças brancas em relação às peças pretas. Um score positivo indica uma vantagem para as brancas, enquanto um score negativo indica uma vantagem para as pretas.

[3]:
```
def stockfish_eval(board_instance, depth):
    move = engine.analyse(board, chess.engine.Limit(time=0.01))
    return chess.engine.PovScore(move['score'], chess.BLACK).pov(chess.BLACK).
 ↪relative.score()
```

1.5 Função para Exibir o Tabuleiro

A função `display_board` utiliza as bibliotecas `chess.svg` e `IPython.display` para criar uma representação visual do tabuleiro de xadrez e exibi-lo dentro do Jupyter Notebook. A função aceita um objeto de tabuleiro `chess.Board` como argumento e gera um SVG (Scalable Vector Graphics) para visualização.

[ ]:
```
def static_eval(board):
    i = 0
    evaluation = 0
    x = True
    try:
        x = bool(board.piece_at(i).color)
    except AttributeError as e:
        x = x
    while i < 63:
        i += 1
        evaluation = evaluation + (get_piece_val(str(board.piece_at(i))) if x␣
 ↪else -get_piece_val(str(board.piece_at(i))))
    return evaluation
```

1.6 Criando e Exibindo o Tabuleiro Inicial

Aqui, um tabuleiro de xadrez padrão é criado e exibido usando a função `display_board` definida anteriormente. O objeto `board` mantém o estado atual do tabuleiro e será usado em análises subsequentes.

[ ]:
```
def get_piece_val(piece):
    if(piece == None):
        return 0
    value = 0
    if piece == "P" or piece == "p":
        value = 10
    if piece == "N" or piece == "n":
        value = 30
    if piece == "B" or piece == "b":
        value = 30
    if piece == "R" or piece == "r":
        value = 50
    if piece == "Q" or piece == "q":
        value = 90
    if piece == 'K' or piece == 'k':
        value = 900
    #value = value if (board.piece_at(place)).color else -value
    return value
```

1.7 Função de Avaliação de Minimax

A função `minimax` implementa o algoritmo Minimax, que é um algoritmo de decisão para encontrar o melhor movimento em um jogo com dois jogadores (jogos de soma zero). A função procura minimizar a perda máxima possível (minimizar o pior caso) em um jogo onde o oponente está tentando maximizar seu próprio ganho.

Os parâmetros da função incluem: - `board`: o estado atual do tabuleiro. - `depth`: a profundidade da árvore de pesquisa (quantos movimentos à frente o algoritmo deve considerar). - `maximizing`: um booleano que indica se o jogador atual está maximizando ou minimizando o score. - `alpha` e `beta`: os valores para a poda alfa-beta, que são usados para reduzir o número de nós avaliados na árvore de pesquisa.

A função retorna o melhor valor de movimento possível a partir do estado atual do tabuleiro, buscando maximizar os movimentos para as peças brancas e minimizar para as peças pretas.

[ ]:
```
def minmax( board_instance, max_depth, current_depth, is_max_player,␣
 ↪nodes_per_depth ):

    # This if else code block is only used for analysis of algorithm, by␣
 ↪counting number of nodes explored
    if max_depth - current_depth in nodes_per_depth:
        nodes_per_depth[max_depth - current_depth] += 1
    else:
```

```
        nodes_per_depth[max_depth - current_depth] = 1

    # This is the base case, depth == 0 means it is a leaf node
    if current_depth == 0:
        leaf_node_score = static_eval(board_instance)
        return (leaf_node_score, nodes_per_depth)

    if is_max_player:

        # set absurdly high negative value such that none of the static␣
↪evaluation result less than this value
        best_score = -100000

        for legal_move in board_instance.legal_moves:
            move =  chess.Move.from_uci(str(legal_move))

            # pushing the current move to the board
            board_instance.push(move)


            #calculating node score, if the current node will be the leaf node,␣
↪then score will be calculated by static evaluation;
            #score will be calculated by finding max value between node score␣
↪and current best score.
            node_score, nodes_per_depth = minmax(board_instance, max_depth,␣
↪current_depth - 1, False, nodes_per_depth)

            # calculating the max value for the particular node
            best_score = max(best_score, node_score)

            # undoing the last move, so that we can evaluate next legal moves
            board_instance.pop()

        return (best_score, nodes_per_depth)
    else:

        # set absurdly high positive value such that none of the static␣
↪evaluation result more than this value
        best_score = 100000

        for legal_move in board_instance.legal_moves:
            move =  chess.Move.from_uci(str(legal_move))

            # pushing the current move to the board
            board_instance.push(move)
```

```
            # calculating node score, if the current node will be the leaf␣
↪node, then score will be calculated by static evaluation;
            # score will be calculated by finding min value between node score␣
↪and current best score.
            node_score, nodes_per_depth = minmax(board_instance, max_depth,␣
↪current_depth - 1, True, nodes_per_depth)

            # calculating the min value for the particular node
            best_score = min(best_score, node_score)

            # undoing the last move, so that we can evaluate next legal moves
            board_instance.pop()

        return (best_score, nodes_per_depth)
```

1.8 Função de Avaliação de Minimax com Poda Alfa-Beta para as Peças Pretas

A função `minimax_black` é similar à função `minimax` definida anteriormente, mas é específica para as peças pretas. Ela também implementa o algoritmo Minimax com poda alfa-beta, buscando minimizar o valor de avaliação do tabuleiro para as peças pretas, considerando que o oponente (peças brancas) está tentando maximizar o valor.

[ ]:
```
def best_move_using_minmax(board_instance, depth, is_max_player):
    best_move_score = -1000000
    best_move = None

    nodes_per_depth = dict()

    for legal_move in board_instance.legal_moves:
        move = chess.Move.from_uci(str(legal_move))
        board_instance.push(move)
        move_score, nodes_per_depth = minmax(board_instance, depth, depth,␣
↪False, nodes_per_depth)
        score = max(best_move_score, move_score)
        board_instance.pop()
        if score > best_move_score:
            best_move_score = score
            best_move = move
    return (best_move, nodes_per_depth)
```

1.9 Função de Avaliação de Minimax com Poda Alfa-Beta para as Peças Brancas

A função `minimax_white` é o contraparte da `minimax_black`, sendo específica para as peças brancas e buscando maximizar o valor de avaliação do tabuleiro para as peças brancas, considerando que o oponente (peças pretas) está tentando minimizar o valor.

[ ]:
```
def game_between_two_computer(depth=3):
    board = chess.Board()

    for n in range(0,10):
        start = time.time()
        if n%2 == 0:
            print("WHITE Turn")
            move, nodes_per_depth = best_move_using_minmax(board, depth, False)
        else:

            print("BLACK Turn")
            move, nodes_per_depth = best_move_using_minmax(board, depth, True)
        end = time.time()

        print("Move in UCI format:", move)
        print("Nodes per depth:", nodes_per_depth)
        print("Time taken by Move:", end-start)
        board.push(move)
        display(SVG(chess.svg.board(board, size=400)))
        print("\n")
```

1.10 Jogando uma Partida

Nesta célula, uma partida completa de xadrez é jogada usando os algoritmos de Minimax para decidir os movimentos tanto para as peças brancas quanto para as pretas. O tabuleiro é exibido após cada movimento para que o progresso do jogo possa ser visualizado.

[ ]:
```
board = chess.Board()
```

1.11 Continuação do Jogo

Esta célula parece ser uma continuação da partida de xadrez começada na célula anterior. Não há muitos detalhes adicionais a serem discutidos aqui, visto que a lógica do jogo já foi explorada nas células anteriores.

[ ]:
```
board
```

[ ]:
```
start = time.time()
move, node_per_depth = best_move_using_minmax(board, 3, True)
end = time.time()
```

[ ]:
```
move
```

[ ]:
```
node_per_depth
```

[ ]:
```
end-start
```

[ ]:
```
board.push(move)
```

[ ]:
```
board
```

[ ]:
```
start = time.time()
move, nodes_per_depth = best_move_using_minmax(board, 3, True)
end = time.time()
```

[ ]:
```
move
```

[ ]:
```
end-start
```

[ ]:
```
nodes_per_depth
```

[ ]:
```
board.push(move)
```

[ ]:
```
board
```

[ ]:
```
game_between_two_computer(3)
```