

Chess Milestone 7



Equipo



Iker Aramuburu
Futuro ingeniero



Daniel Cobos
Ingeniero



Alicia del Amo
Ingeniera



Javier Ruiz
Futuro ingeniero

Índice

- + Introducción
- + Nomenclatura
- + Árbol de funciones
- + Ventajas de OOP
- + Métricas
- + Cambios aplicados
- + Aprendizajes
- + Conclusiones




Introducción

Este trabajo ha consistido en la actualización y optimización de un repositorio de Python sobre el juego del ajedrez.



Nomenclatura



```
'camelCase',  
'lowercase',  
'lower-kebab-case',  
'lower_snake_case',  
'pascalCase',  
'Sentencecase',  
'UPPERCASE',  
'UPPER-KEBAB-CASE',  
'UPPER_SNAKE_CASE'
```

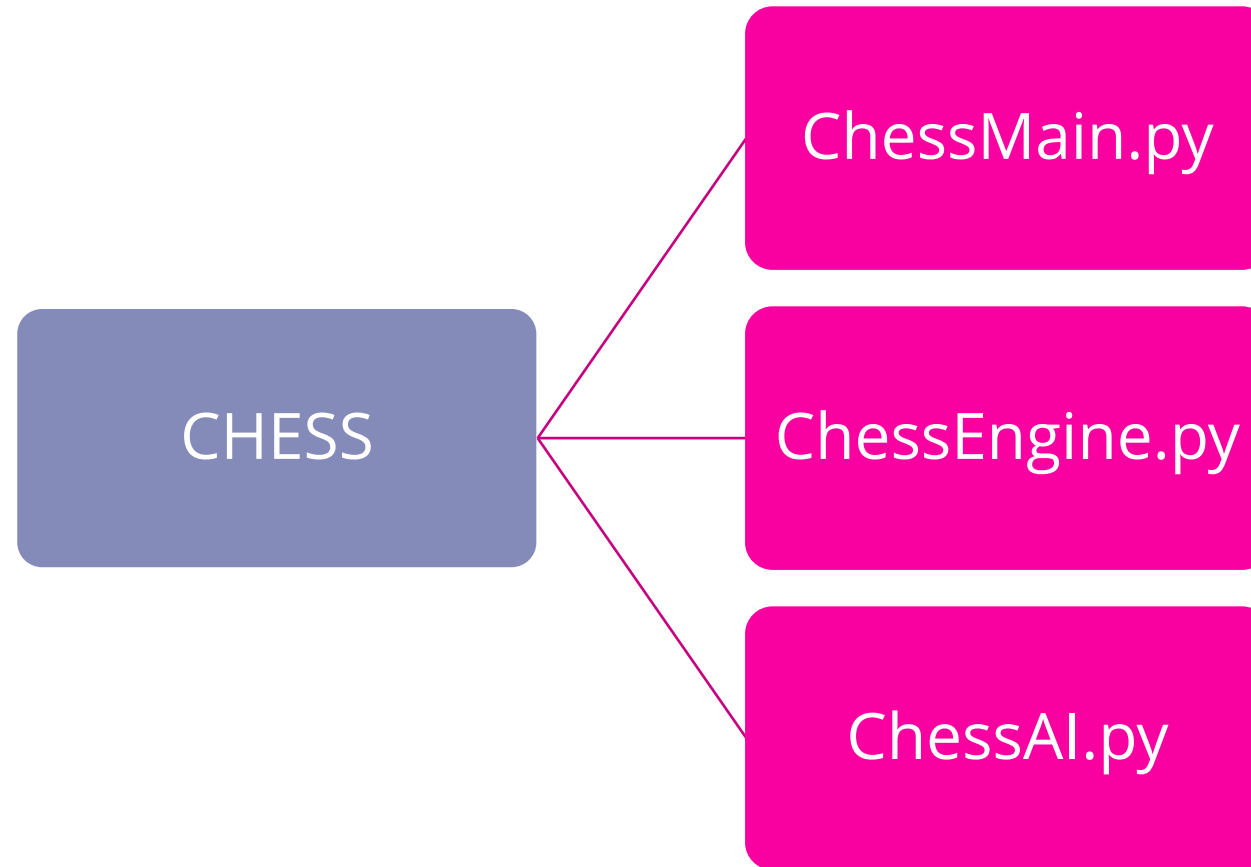
- ❑ Atributos, variables y objetos → lower_snake_case
- ❑ Constantes → UPPER_CASE_WITH_UNDERSCORES
- ❑ Archivos .py, clases, métodos y funciones → lowerCamelCase



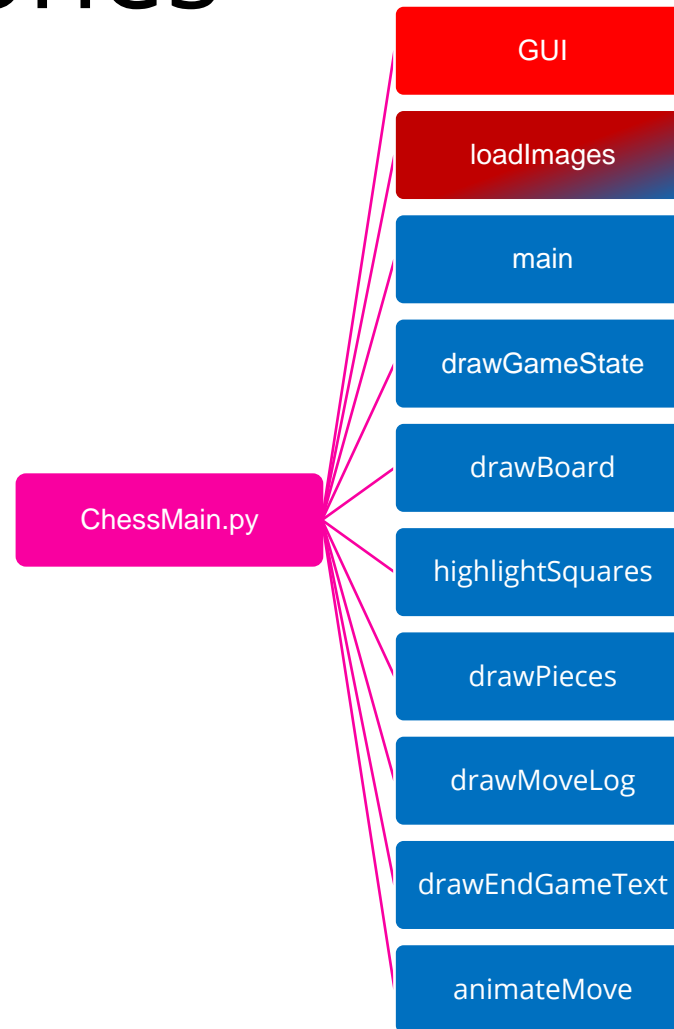
Árbol de funciones

Árbol de funciones

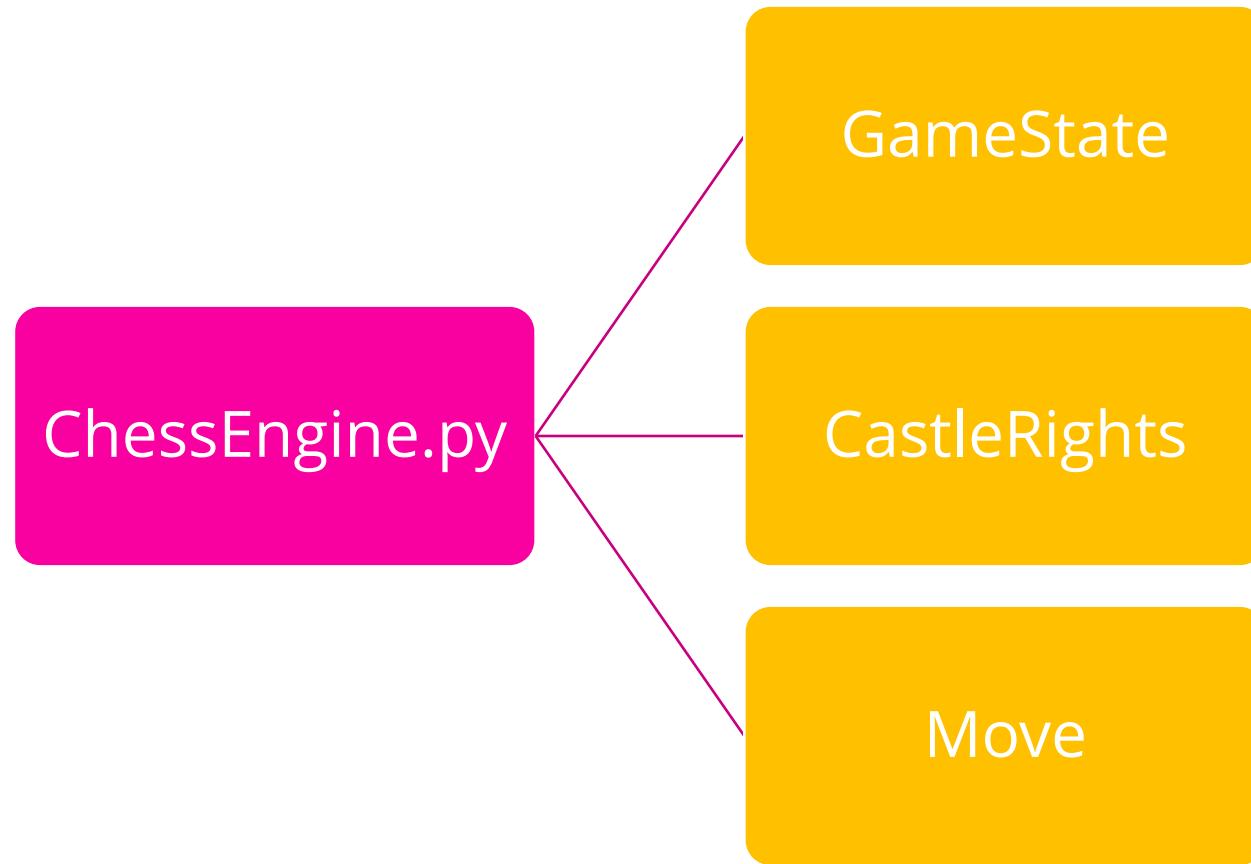
ARCHIVO PYTHON
CLASE
MÉTODO
FUNCIÓN
MEJORA



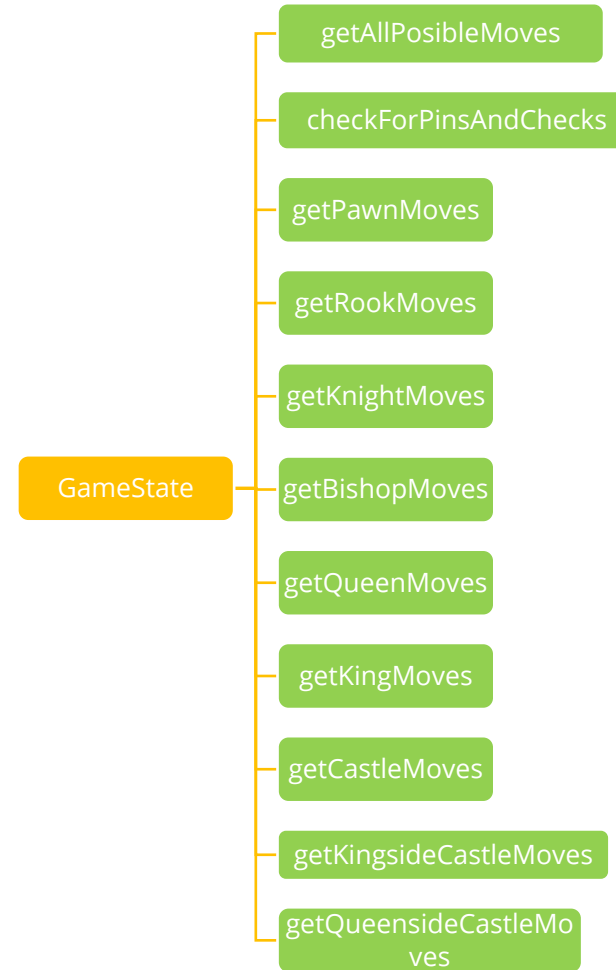
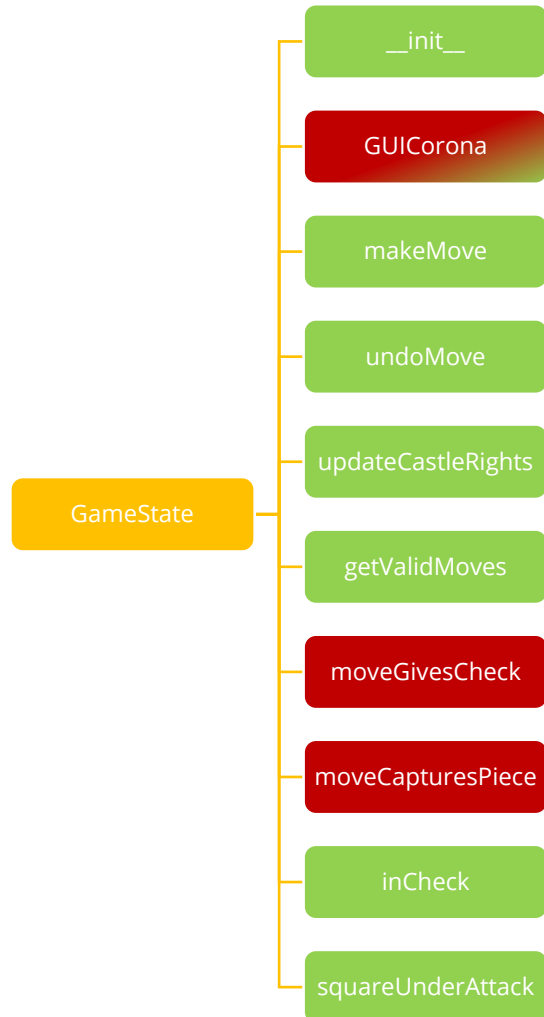
Árbol de funciones



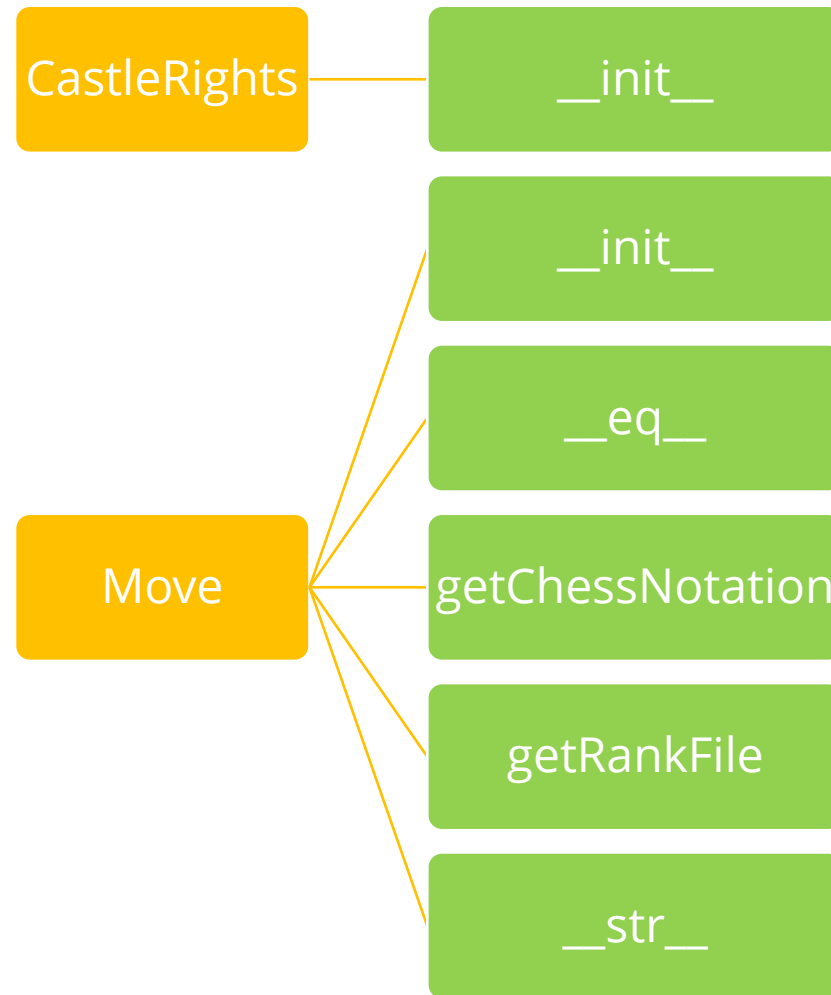
Árbol de funciones



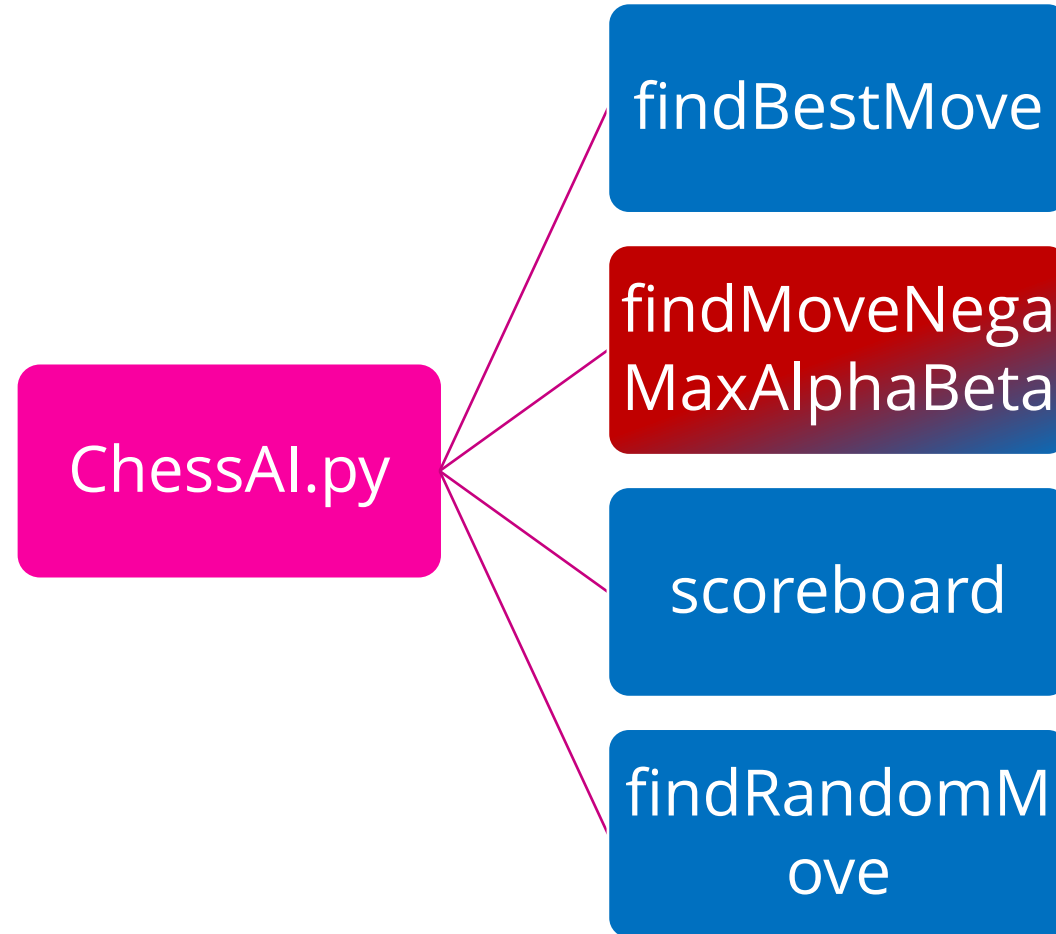
Árbol de funciones



Árbol de funciones

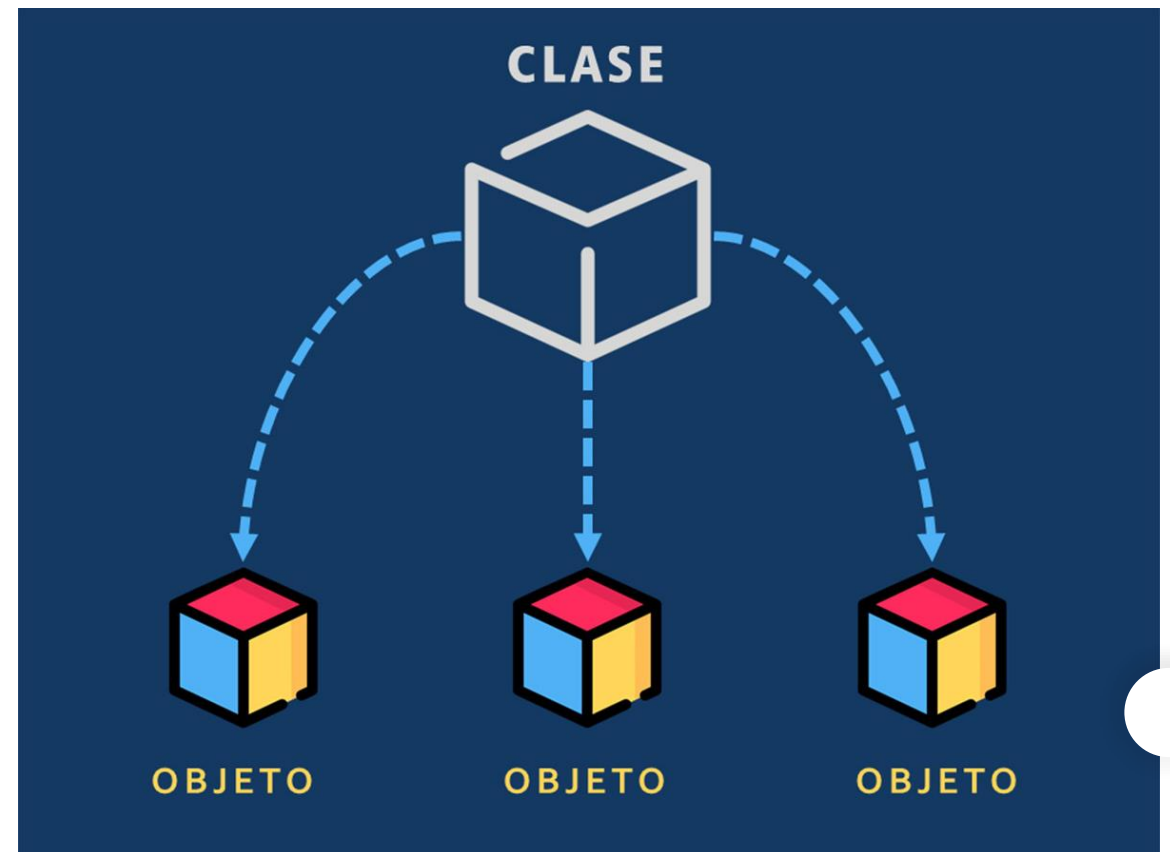


Árbol de funciones



Ventajas de la programación orientada a objetos

1. Modularidad y reutilización
2. Abstracción
3. Herencia
4. Mantenimiento y extensibilidad
5. Colaboración en equipos






Métricas

Análisis pylint

	Original	Modificado
ChessMain.py	6,21	9,25
ChessEngine.py	7,6	8,00
ChessAI.py	7,41	8,18



```
31
32 self.file = None
33 self.fingerprints = set()
34 self.logdupes = True
35 self.debug = debug
36 self.logger = logging.getLogger(__name__)
37 if path:
38     self.file = open(os.path.join(path, 'requests.json'),
39                     'a')
40     self.fingerprints.update(set(self.fingerprints))
41
42 @classmethod
43 def from_settings(cls, settings):
44     debug = settings.getbool('DEBUG', False)
45     return cls(job_dir(settings), debug)
46
47 def request_seen(self, request):
48     fp = self.request_fingerprint(request)
49     if fp in self.fingerprints:
50         return True
51     self.fingerprints.add(fp)
52     if self.file:
53         self.file.write(fp + os.linesep)
54
55 def request_fingerprint(self, request):
56     return request_fingerprint(request)
```



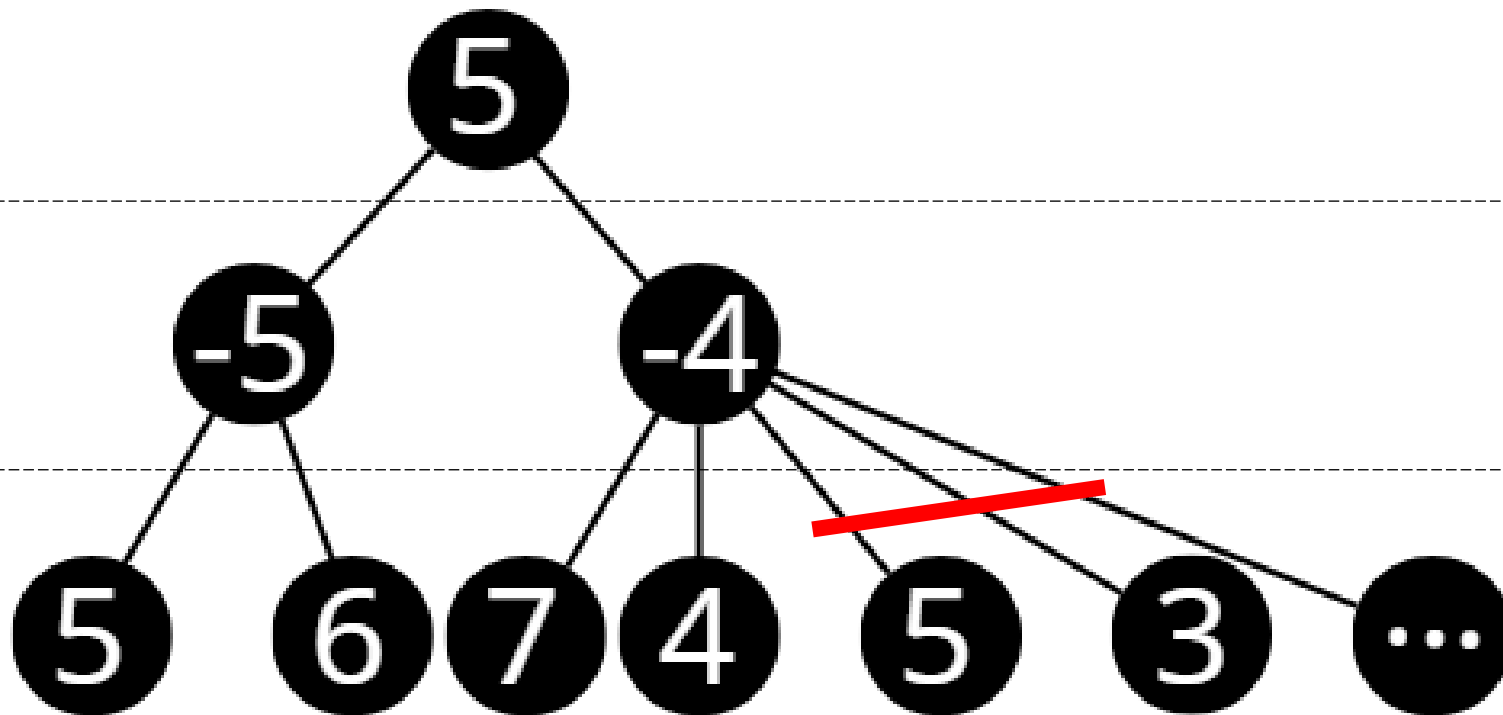
Cambios aplicados

Algoritmo NegaMax + Poda Alpha-beta

Turno blancas

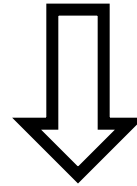
Turno negras

Turno blancas



Optimización algoritmo

```
1 def findBestMove(game_state, valid_moves, return_queue):
2     global next_move
3     next_move = None #Almacena el mejor movimiento encontrado
4     random.shuffle(valid_moves) #Mezcla todos los movimientos válidos para introducir algo de aleatoriedad en la selección
5     findMoveNegaMaxAlphaBeta(game_state, valid_moves, DEPTH, -CHECKMATE, CHECKMATE, 1 if game_state.white_to_move else -1) #Búsqueda del mejor movimiento con poda alfa-beta
6     return_queue.put(next_move) #añade a la lista de movimientos el movimiento elegido
```



```
1 def moveGivesCheck(self, move)
```

```
1 def moveCapturesPiece(self, move)
```

Tablas por 50 movimientos

En ChessEngine.py,
en GameState:

```
1 self.contador = 0
```

En ChessEngine.py,
en MakeMove:

```
1 if move.is_capture == True:  
2     self.contador = 0  
3     self.contador += 1
```

En ChessEngine.py,
en UndoMove:

```
1 self.contador -= 1
```

Posibles elecciones (1)



Configuración del Juego

Blancas:
Jugador ▾

Negras:
CPU ▾

Nivel:
0 ▾

Tiempo (mins):
3

Iniciar Juego

Posibles elecciones (2)

Jugadores

```
1 p_1 = bool(p1_e == "Jugador")
2
3 p_2 = bool(p2_e == "Jugador")
```

```
1 if __name__ == "__main__":
2     i =+1
3     if i == 1:
4         p1_e, p2_e, nivel, tiempo = GUI()
5
6     main()
```

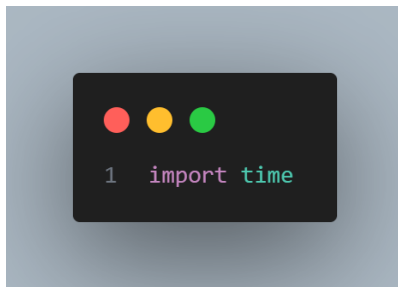
Nivel

```
1 def findBestMove(game_state, valid_moves, return_queue, depthAux):
```

```
1 def findMoveNegaMaxAlphaBeta(game_state, valid_moves, depth, alpha, beta, turn_multiplier, depthaux):
```

Posibles elecciones (3)

Tiempo

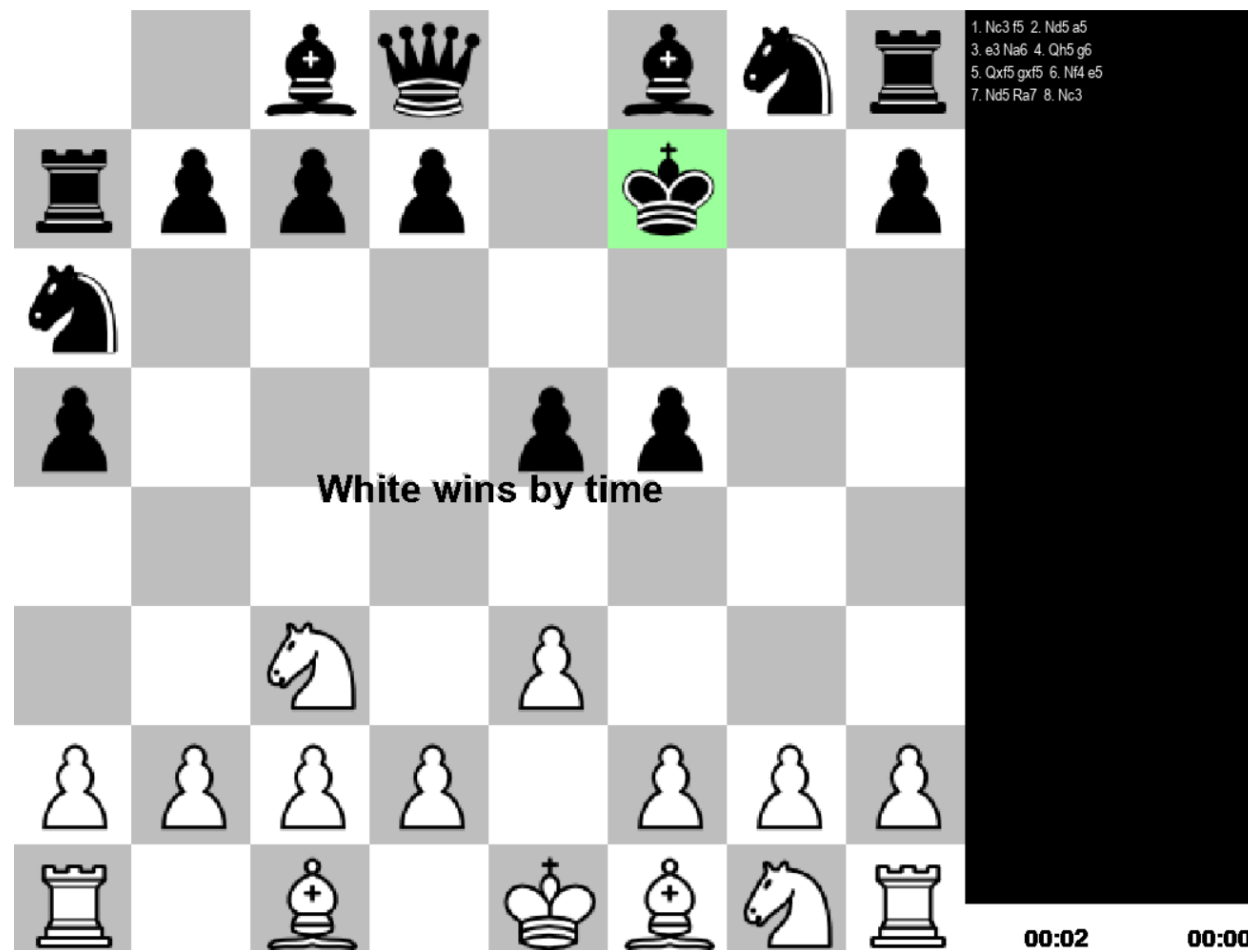


```
1 # Variables del temporizador (AHORA CON TIEMPO DE INICIO INDIVIDUAL)
2 time_limit = float(tiempo) * 60
3
4 # Variables del temporizador 1 (Jugador Blanco)
5 time_left1 = time_limit
6 timer_active1 = False
7
8 # Variables del temporizador 2 (Jugador Negro)
9 time_left2 = time_limit
10 timer_active2 = False
```

```
1 # --- Lógica CRUCIAL para el manejo de turnos y temporizadores ---
2 if not game_over:
3     if game_state.white_to_move: # Turno del blanco
4         if not timer_active1: # Si no estaba activo, se inicia el tiempo
5             start_time1 = time.time() # Se guarda el tiempo de inicio del turno
6             timer_active1 = True
7             timer_active2 = False
8         else: # Turno del negro
9             if not timer_active2: # Si no estaba activo, se inicia el tiempo
10                 start_time2 = time.time() # Se guarda el tiempo de inicio del turno
11                 timer_active2 = True
12                 timer_active1 = False
```

```
1 # Actualizar los temporizadores (AHORA CON LA LÓGICA CORRECTA)
2 if timer_active1 and not game_over:
3     elapsed_time = time.time() - start_time1 # Tiempo transcurrido desde el inicio del turno
4     time_left1 -= elapsed_time # Se resta el tiempo transcurrido al tiempo restante
5     time_left1 = max(0, time_left1) # Se asegura de que no sea negativo
6     start_time1 = time.time() # Se actualiza el tiempo de inicio para la siguiente iteración
7
8     if time_left1 <= 0:
9         game_over = True
10         winner = "Black"
11
12 if timer_active2 and not game_over:
13     elapsed_time = time.time() - start_time2
14     time_left2 -= elapsed_time
15     time_left2 = max(0, time_left2)
16     start_time2 = time.time()
17
18     if time_left2 <= 0:
19         game_over = True
20         winner = "White"
```

Posibles elecciones (4)



Elección coronación

En ChessEngine.py:

```
1 # pawn promotion
2 if move.is_pawn_promotion:
3     if j == 1:
4         promoted_piece = GameState.GUICorona()
5         self.board[move.end_row][move.end_col] = move.piece_moved[0] + promoted_piece
6     else:
7         self.board[move.end_row][move.end_col] = move.piece_moved[0] + "Q"
```

j = 1 (humano); j = 0 (CPU)



Aprendizajes

Algoritmos de búsqueda

- + Eficientes, precisos y flexibles.
- + Limitados, no aprenden.

Redes neuronales

- + Aprendizaje de patrones complejos, generalización.
- + Datos, interpretabilidad, tiempo de aprendizaje.

Stockfish



Conclusiones

- Mejora sustancial de los conocimientos en Python
- Optimización y depuración del código
- Margen de mejora



Gracias

+ ¿Momento de probar el código
Juan Antonio?

