Analysis of First Profiling Results

After First run of cProfile Optimizer

“3919289 function calls (3871468 primitive calls) in 181.505 seconds”

1. **The Biggest Time Consumer: The Game Loop's Frame Rate**

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
13775	98.166	0.007	98.166	0.007	{method 'tick' of 'pygame.time.Clock' objects}

The single most time-consuming function is **pygame.time.Clock.tick**, which takes **98.166 seconds** (over 54% of the total execution time).

Remedial Action: Lower the FPS to a more reasonable value like 30 or 60

2. **The Next Biggest Time Consumer: AI Network Latency**

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
150	58.885	0.393	58.885	0.393	{method 'read' of '_ssl._SSLSocket' objects}

The **get_ai_move** function and its underlying network calls (through **openai**, **httpx**, and **ssl**) are the next major bottleneck, consuming about **60 seconds** in total. The function {method 'read' of '_ssl._SSLSocket' objects} alone took **58.8 seconds**.

Reason: This time is spent waiting for a response from the DeepSeek API over the internet. Program is paused, waiting for an external resource, that cannot be optimized

Remedial Action: Notifying human player that AI opponent is thinking. Implement and render **thinking_text** as "AI is thinking..."
Also, will consider using function caching in the ai.py module, as Python's standard library provides a simple way to do this with the functools.lru_cache decorator.
A function decorated with @lru_cache will automatically store its recent results in a dictionary-like cache

I have also chosen to profile the AI's turn because it is the only part of the game logic that introduces a noticeable delay and is therefore a candidate for analysis and improvement.

3. **Inefficient Drawing in the Game Loop**

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
13775	4.396	0.000	4.396	0.000	{built-in method pygame.display.update}
336589	2.406	0.000	2.406	0.000	{method 'blit' of 'pygame.surface.Surface'
336589	2.401	0.000	2.401	0.000	{method 'render' of 'pygame.font.Font' objects}
578550	1.642	0.000	1.642	0.000	{built-in method pygame.draw.circle}

The functions draw_matrix_info, board.draw_board, pygame.font.Font.render,
and pygame.Surface.blit are called thousands of times and collectively consume several seconds.

Reason: draw_matrix_info is called in every single frame of the game loop. It re-renders the text for all three tracking matrices from scratch, even when the data hasn't changed. This is inefficient

Remedial Action: Refactor drawing logic to only redraw static elements like the board and matrices when the game state actually changes, not on every single frame.

a. Implement a flag in class Board's constructor (in module board.py) to indicate if the board needs to be redrawn. **Set flag to true** at object instantiation to ensure the initial rendering of the game board.
b. **Set flag to true** in **drop_piece** method as it changes board when it is called.
c. **Test flag** before calling **draw_board** and **draw_matrix_info** in main.py and **set to false** ensuring subsequent iterations of the 'while run' loop only enter this block if the **drop_piece** method is called and changes the flag. This ensures the 'while run' loop in main.py draws the board and matrix information only when the board has changed

Analysis of the Second Profile

After Optimizations and changes made to reflect recommendations from previous output

"1777137 function calls (1743005 primitive calls) in 97.454 seconds" compared to "3919289 function calls (3871468 primitive calls) in 181.505 seconds" from first profile.

Improved program performance with a total runtime of 97.4 seconds (an ~46% improvement) and far fewer function calls (1.7 million vs. 3.9 million).

1. **The Biggest Time Consumer is still the Game Loop's Frame Rate**

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
4852	77.746	0.016	77.746	0.016	{method 'tick' of 'pygame.time.Clock' objects}

The single most time-consuming function is still **pygame.time.Clock.tick**, which has been reduced from the initial profile result of **98.166 seconds** to **77.746 second**.

Remedial Action: Lower the FPS from 60 to 30

2. **AI Network Latency**

First Profiler Network Latency:

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
150	58.885	0.393	58.885	0.393	{method 'read' of '_ssl._SSLSocket' objects}

Second Network Latency after optimization:

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
28	12.711	0.454	12.711	0.454	{method 'read' of '_ssl._SSLSocket' objects}

The number of calls have reduced significantly from 150 to 28 and the total time has reduced from 58.885 to 12.711 secs. These are significant declines.

Evidence: As seen below, the cumulative time for **get_ai_move** and its children dropped from ~60 seconds to ~14 seconds. The new **_get_ai_move_cached** function is now visible in the profile.

AI Calls Optimized (lru_cache):

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
8	0.000	0.000	13.924	1.741	...\ai.py:28(get_ai_move)
8	0.002	0.000	13.924	1.740	...\ai.py:42(_get_ai_move_cached)

Result: The introduction of **@lru_cache** was highly effective. When the AI provided an invalid move, forcing a retry, the subsequent call for the same board state was served instantly from the cache instead of making another slow network request. This also made the AI more robust against temporary network errors

3. Inefficient Drawing in the Game Loop

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
13775	4.396	0.000	4.396	0.000	{built-in method pygame.display.update}
336589	2.406	0.000	2.406	0.000	{method 'blit' of 'pygame.surface.Surface'
336589	2.401	0.000	2.401	0.000	{method 'render' of 'pygame.font.Font' objects}
578550	1.642	0.000	1.642	0.000	{built-in method pygame.draw.circle}

Drawing Loop Fixed (with chk Flag):
Evidence: The drawing functions (draw_board, draw_matrix_info, blit, render) have completely disappeared from the top 20 list of time-consuming functions.
Result: The implementation of the **board.chk** flag was a massive success. The board is no longer redrawn on every frame, which slashed the total number of function calls by over 2 million and significantly reduced the time spent on rendering.