The documentation for Fyrox is rather inextensive, which resulted in a lot of trial and error regarding combining writing code with adding objects via the editor since it was a new concept for us to add scripts to entities (not to mention the fact that there are 3 main resources which we researched since there's little to no information about this engine and AIs are not trained for our purpose).

Used Perplexity, ChatGPT, DeepSeek and GitHub Copilot as prompt engineering tools assisting with code writing, concept explanation, and bug resolution.

ChatGPT was a bit useless at first, when designing the main logic for the game, especially for configuring the game context, along with the player and bot `*on_start*`, `*on_update*` and `*on_os_event*` overriden methods. For this reason, as prompt engineers we consulted the provided Fyrox tutorials (https://fyrox-book.github.io/tutorials/platformer/intro.html). Then, GPT alongside Copilot greatly improved our efficiency by helping with logic and code for item spawning (heart, bomb, fire). Interesting enough, GPT also created the background image called `BG.png` located in /data/background folder.

Perplexity used tuned GPT-4.1, o3 and Claude 4.0 Sonnet Thinking models for helping diagnose and solve a performance issue where target sprites were accumulating as invisible nodes in the scene graph, causing lag, by identifying that setting visibility to false did not remove nodes, and recommending node removal to prevent buildup.* Perplexity's model Sonar helped in identifying problems like unused nodes and inefficient patterns, assuring quality standards across methods.

DeepSeek provided us with an interesting approach for creating the enemies via code, including most of the needed children nodes. However, this approach caused the sharing resources issue, which as prompt users we came to the conclusion that it's resource safe to create 5 skeleton nodes containing different children nodes within `*scene.rgs*` which are set from invisible to visible via code after the game starts, rather than creating these nodes from code and not knowing how to handle the children nodes not intervening with each other.

* Perplexity found a decent solution using *ctx.scene.graph.remove_node(prev_target)*, however the problem with imports, which orchestrated the overall node removal process, was impossible to be solved by LLMs (couldn't even be solved by using GPT's Think for longer option). As prompt engineers we had to step-in and had to navigate through the official Rust documentation https://docs.rs/fyrox/latest/fyrox/index.html to find the solution for imports.