Expected Design Patterns.

Due to the small scope in complexity of our project, many patterns are either overly complicated, or plainly impractical in this context. However, there are a few exceptions, where the implementation of such design patterns could greatly increase the effectiveness of the code.

**STRATEGY:** There are going to be multiple enemy types in this game, each with their unique behaviour. On top of that, some power ups the hero may collect can alter the AI behaviour, which makes the use of this pattern easy to understand.

Regarding the implementation, a public interface will be set containing the basic behaviour to be overwritten by each enemy class. There, the specifics of the move(), attack() (if the type of enemy actively attempts to damage the hero) are created. This way we can implement new behavioural methods without replacing nor damaging existing code.

**DECORATOR:** One of the core gameplay elements are the power ups. Randomly popping up in reachable areas, these grant either bonuses to the hero’s statistics, or harm all or random enemies on the game area. What makes this pattern so valuable in this context is the possibility of the hero collecting multiple power ups, each making him more powerful. Instead of building a complex state machine to think of possible combinations, we simply use a decorator class.

This new interface will be implemented by the power ups, overriding some of the hero’s characteristics, e.g. gaining an extra life, or become invincible for a short period of time, essentially making the hero’s die() function irrelevant.