Relatório – EP de COO 1º semestre/2025

Nome: Kauã Lima de Souza | **Nº USP:** 15674702

---

1. Críticas ao Código Original

O código original do jogo escrito de uma maneira procedural, embora funcional, apresentava diversos problemas de um estilo procedural e sem modularização:

- **Falta de encapsulamento:** Toda a lógica do jogo estava concentrada em uma única classe Main, o que tornava a documentação muito difícil.
- **Repetição de código:** Muitos pedaços de código eram duplicados para tratar entidades semelhantes (inimigos, projéteis etc.).
- **Dificuldade de adições de mais features:** A adição de novas funcionalidades (features), como power ups ou chefes, é difícil.
- **Uso de arrays fixos:** O gerenciamento das entidades no código procedural é feito por arrays de tamanho fixo, levando à reutilização manual de posições e dificultando a remoção de entidades inativas.
- **Difícil entendimento:** A mistura de lógica de jogo, controle de fluxo e manipulação gráfica em métodos longos tornava o código difícil de entender e modificar.

2. Justificativa da Nova Estrutura de Classes/Interfaces

A reescrita do projeto seguiu princípios fundamentais da orientação a objetos, tendo encapsulamento, heranças, polimorfismo, abstração, modularidade, reutilização e facilidade de manutenção (com possibilidade de identificar facilmente onde acontecem erros). As principais decisões de projeto foram:

- **Separação de entidades:** Criei uma classe abstrata Entidade, da qual herdam as classes Player, EnemyBase, Projetil, PowerUp1, PowerUp2, etc. Isso permite tratar diferentes tipos de entidades de forma polimórfica, já que tudo no jogo que se move, atira, muda de estado é uma entidade e possui alguns atributos em comum.
- **Heranças:** Inimigos (Enemy1, Enemy2), chefes (Boss1, Boss2) e power-ups são subclasses especializadas ou de Entidade ou de alguma classe que herda de entidade, cada uma com seu comportamento próprio, sobrescrevendo métodos conforme necessário.

- **Interfaces para contratos:** Interfaces
  como BossInterface e PowerupInterface definem contratos para entidades especiais, facilitando a extensão e a implementação de novos tipos.
- **Encapsulamento:** Cada classe é responsável por seu próprio estado e comportamento, expondo apenas métodos necessários.
- **Separação de responsabilidades:** A lógica do jogo foi dividida em controladores Game e Phase, de modo que um jogo instancia/gerencia as fases e com determinada condição (Eliminação dos chefes e fim da ultima fase) o jogo termina, entidades do jogo e utilitários (Classe Utils), promovendo coesão.
- **Pacotes:** Usei pacotes para a separação devida de cada tipo de classe

3. Uso das Coleções Java no Lugar de Arrays

Substituí arrays fixos por listas dinâmicas (ArrayList) do collections do Java para gerenciar entidades como inimigos, projéteis e power-ups. Isso permite adicionar e remover elementos de forma eficiente e segura.

- **Remoção de entidades inativas:** Entidades que deixam de ser relevantes (ex: projéteis fora da tela, inimigos destruídos) são removidas ou marcadas como inativas (Existe um ENUM que mantem quais estados são possíveis), evitando vazamento de memória e melhorando a performance.
    - **OBS:** Poderíamos apenas ter substituído essa funcionalidade de manter as entidades Inativas ou ativas por simplesmente inserir novas entidades na lista ou removê-las com “pop”, porém o mecanismo de estados faz com que se tenha um número máximo de entidade de um tipo no jogo e assim a memória não é extrapolada, nos causando menos problemas.
- **Iteração segura:** O uso de coleções permite iteração direta sobre os elementos ativos, simplificando a lógica de atualização e colisão.
- **Exemplo:**
  No método de atualização de projéteis, percorremos a lista de projéteis e removemos aqueles que saíram da tela ou colidiram, sem a necessidade de gerenciar índices manualmente.

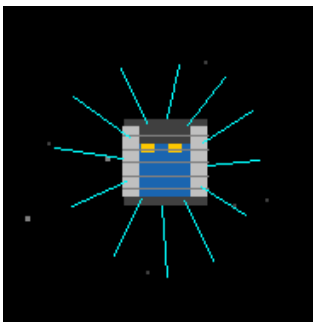
4. Implementação das Novas Funcionalidades com Apoio do Código Orientado a Objetos

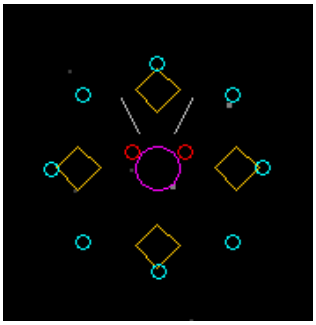
Power-ups

- **Dois tipos implementados:** PowerUp1 e PowerUp2, ambos derivados de uma interface comum, com efeitos distintos sobre o jogador (o poweup1 concede aumento de velocidade, powerup2 dobra a quantidade de tiros).
    - **PowerUp1:** Faz com o player ganhe aumento de velocidade por 30 segundos.
        - ➔ **Icone do poder a ser pego:** 
        - ➔ **Design do Player quando obtém o poder:** 
    - **PowerUp2:** Faz com que o player ganhe o dobro de tiros do que o estado anterior por 30 segundo, seu design muda também**.**
        - ➔ **Icone do poder a ser pego:** 
        - ➔ **Design do Player quando obtém o poder:** 
- **Aparição configurável:** Os power-ups são definidos nos arquivos de configuração das fases, podendo ser facilmente ajustados ou expandidos.
    - **Ex:** POWERUP 1 174000 316 -10
        - ➔ **Aqui ele aparece quase no fim do jogo com as devidas posições**
- **Coleta e efeito:** O player verifica colisão com power-ups em métodos específicos na classe Player, aplicando o efeito correspondente de forma encapsulada, o próprio player gerencia os status que ele tem, então se o efeito passa ele para de ter esses atributos melhorados.
- **Design próprio:** Os powerups tem um desenho próprio e mudam o design do player de acordo com qual power up é selecionado

Chefes de Fase

- **Dois chefes distintos:** Boss1 e Boss2 ambos implementando a interface BossInterface e herdando de EnemyBase, já que de certa forma também são inimigos. (Boss1 e Boss2 nas respectivas imagens)





- **Comportamentos diferenciados:** Cada chefe possui padrões de movimento e ataque próprios, definidos em seus métodos de atualização.
- **Barra de vida:** A barra de vida do chefe é desenhada na tela enquanto ele está ativo, utilizando métodos utilitários, método que desenha a vida do boss implementada na classe Utils.
- **Avanço de fase:** A derrota do chefe é detectada e aciona a transição para a próxima fase, de acordo com a classe Game.

Outras Melhorias

- **Configuração externa:** O jogo lê arquivos de configuração para definir fases, inimigos, chefes e power-ups, facilitando a customização.
- **Facilidade de extensão:** Novos tipos de inimigos, power-ups ou fases podem ser adicionados com mínimo impacto no restante do código, agora a fase não gera os inimigos aleatoriamente.

Conclusão

A reestruturação do código utilizando princípios de orientação a objetos resultou em um projeto mais modular, legível e fácil de manter. O uso de coleções dinâmicas e a separação clara de responsabilidades permitiram a implementação eficiente de novas funcionalidades, como power-ups e chefes, além de facilitar futuras expansões do jogo.

Além disso, futuras implementações de mais fases, chefes, inimigos e powerups é mais fácil agora de forma que as classes e as regras do jogo já estão estabelecidas.

Pensei em implementar menus ou algum esquema de pausa para o jogo, porém achei que não contasse para o escopo do projeto, porém para algo assim seria necessário implementações na GameLib, de forma que pudéssemos renderizar palavras e frases dentro do Jogo.