Projeto de COO - 1ºsemestre/2025

Aplicação de princípios de POO a um código procedural

Descrição

Neste projeto, a ser feito em grupo, vocês devem reescrever o o código de um jogo, disponibilizado no arquivo **Projeto_COO.zip**, de modo a aplicar diversos conceitos estudados na disciplina *Computação Orientada a Objetos*. O jogo, trata-se de um *shoot 'em up* (http://en.wikipedia.org/wiki/Shoot_'em_up) vertical bastante simples, sem acabamento (não possui tela de título, placar, vidas, fases, chefes, *power-ups*, etc) e que roda de forma indefinida (até que o jogador feche a janela do jogo).

Embora funcione, seu código **não** foi elaborado seguindo bons princípios de orientação a objetos. Apesar de escrito em Java, o código foi elaborado seguindo um estilo de programação procedural e, mesmo considerando este estilo, não muito bem feito uma vez que há muito código redundante. Existem portanto inúmeras oportunidades de melhoria do código. Os dois principais aspectos que devem ser trabalhados no desenvolvimento deste projeto são:

- A aplicação de **princípios de orientação a objetos**, através da criação de uma boa estrutura de interfaces e classes bem encapsuladas, definição de uma hierarquia de classes/interfaces adequada, e uso de recursos como herança/composição.

- O uso das **coleções Java** ao invés de *arrays* para manter/gerenciar as entidades do jogo que aparecem em multiplicidade (inimigos, projéteis, etc).

O código do jogo é composto por dois arquivos fonte: **Main.java** e **GameLib.java**. No primeiro arquivo está implementada toda a lógica do jogo, enquanto o segundo implementa uma mini biblioteca com recursos úteis no desenvolvimento de jogos: inicialização da interface gráfica, desenho de figuras geométricas simples e verificação de entrada através do teclado.

O foco da reescrita de código deve ser a classe **Main**. Pode-se assumir que a classe **GameLib** é uma biblioteca fechada à qual não se tem acesso ao código-fonte (como se realmente fosse uma biblioteca feita por terceiros) e portanto ela não precisa ser modificada neste trabalho, apenas utilizada.

Além da reescrita do código, também deverão ser implementadas algumas funcionalidades extras no jogo (se a reescrita do código for bem feita, a implementação destas funcionalidades extras deve acontecer de forma relativamente simples):

Power-ups

Itens que aparecem na tela de jogo e quando coletados pela nave do jogador melhoram algum aspecto do comportamento da nave. Devem ser implementados dois tipos de *power-ups* diferentes cujos efeitos ficam a cargo do grupo. O aparecimento dos *power-ups* deve ser definidos nos arquivos de configuração das fases, como se fosse um inimigo simples, usando-se a palavra-chave **POWERUP** (mais detalhes sobre o funcionamento do mecanismo para configuração das fases são apresentados a adiante).

Chefes de fase

Devem ser implementados 2 chefes de fase distintos. Os chefes de fase podem ser considerados inimigos especiais, que devem possuir as seguintes características particulares:

- Os chefes não podem sair da tela uma vez que tenham entrado na área de jogo (diferentemente dos inimigos comuns que em algum momento acabam saindo da tela caso não sejam abatidos pelo jogador).

- Os chefes deve apresentar comportamentos de ataque e movimento diferentes daqueles apresentados pelos inimigos comuns (e diferentes entre si também). Devem apresentar visual próprio também.

- Os chefes também devem possuir pontos de vida. Quando entrar na área de jogo uma barra de vida também deve ser exibida para o chefe.

- Deve-se assumir que um chefe aparece uma única vez durante uma fase. Além disso, a derrota de um chefe implica no avanço para a próxima fase (ver mais detalhes sobre as fases adiante).

Fases (e arquivos de configuração do jogo/fases)

Outra funcionalidade a ser implementada consiste na criação de um mecanismo para a configuração do jogo e definição de fases. Uma fase nada mais é do que um conjunto de entradas que definem **quando** e **onde** os inimigos devem aparecer, além do **tipo**. Na definição de cada fase deve-se **obrigatoriamente** incluir uma entrada que irá definir a aparição de **um** chefe. Uma fase é considerada finalizada quando o chefe é abatido pelo jogador, e deve-se passar para a próxima fase (caso não seja a última). O jogo deve ser configurado através de um arquivo (formato texto) estruturado da seguinte forma:

```
<PONTOS DE VIDA DO JOGADOR>
<NUMERO DE FASES>
<ARQUIVO DE CONFIGURAÇÃO DA FASE 1>
<ARQUIVO DE CONFIGURAÇÃO DA FASE 2>
…
<ARQUIVO DE CONFIGURAÇÃO DA FASE N>
```

onde: `PONTOS DE VIDA DO JOGADOR` é um valor inteiro que define a quantidade de pontos de vida com o qual o jogador começa o jogo; `NUMERO DE FASES` um valor inteiro que define quantas fases o jogo terá; e as demais linhas definem os nomes de arquivos de configuração para cada uma das fases. Cada arquivo configuração de fase deve estar estruturado da seguinte forma:

```
INIMIGO <TIPO> <QUANDO> <POSIÇÃO INICIAL X> <POSIÇÃO INICIAL Y>
INIMIGO <TIPO> <QUANDO> <POSIÇÃO INICIAL X> <POSIÇÃO INICIAL Y>
…
INIMIGO <TIPO> <QUANDO> <POSIÇÃO INICIAL X> <POSIÇÃO INICIAL Y>
CHEFE <TIPO> <PONTOS DE VIDA> <QUANDO> <POSIÇÃO INICIAL X> <POSIÇÃO INICIAL Y>
```

onde: cada linha determina a aparição de um inimigo simples (definida pela palavra-chave **INIMIGO**) ou chefe (definida pela palavra-chave **CHEFE**. Note também que a linha que define a aparição do chefe deve ocorrer apenas uma vez no arquivo, mas esta linha não precisa ser, obrigatoriamente, a última); `TIPO` é um valor inteiro que define o tipo do inimigo ou chefe, e este deve assumir valor 1 ou 2 (afinal, tem-se dois inimigos comuns e deverão ser implementados dois chefes); `QUANDO` é um valor inteiro que define o instante em que um inimigo ou chefe deve aparecer na fase (este valor deve ser definido em milissegundos e relativo ao instante de início da fase); `POSIÇÃO X` e `POSIÇÃO Y` são valores inteiros que definem as coordenadas do ponto no qual o inimigo ou chefe deve aparecer; e finalmente `PONTOS DE VIDA` é um valor inteiro que define a quantidade de pontos de vida que o chefe da fase deve possuir.

Relatório

Também faz parte do trabalho a elaboração de um relatório que deve documentar:

- Críticas ao código original do jogo.
- Descrição e justificativa para a nova estrutura de classes/interfaces adotada.
- Descrição de como as coleções Java foram utilizadas para substituir o uso de *arrays*.
- Descrição de como as novas funcionalidades foram implementadas e como o código orientado a objetos ajudou neste sentido.

Algumas observações sobre o uso das coleções Java

Na versão original do código, é feito um uso extensivo de *arrays* para gerenciar entidades do jogo que ocorrem em multiplicidade (inimigos, projéteis, etc). Devido ao fato de *arrays* serem estruturas de armazenamento estáticas, todos os *arrays* são alocados com tamanhos fixos e suas posições são **reutilizadas** sempre que uma entidade associada a um determinado índice torna-se inativa (quanto sai da tela, no caso dos inimigos e projéteis, ou quando é abatida pelo jogador, no caso dos inimigos).

Fazendo-se bom uso da das coleções Java, para armazenar e gerenciar esses conjuntos de entidades, a

reutilização de posições deixa de ser necessária pois todas as coleções implementadas pelo Java são dinâmicas. Contudo é importante não esquecer de remover as entidades que se tornam inativas da coleção que as armazena a fim de evitar vazamentos de memória durante a execução do jogo.

Entrega

Este projeto pode ser feito (e recomenda-se que seja feito) em grupos de até 4 pessoas. Deve ser entregue:

- Código fonte reescrito.
- Relatório (em formato **PDF**).

A entrega deverá ser feita pelo eDisciplinas, na atividade aberta para este fim, até o final do dia 30/06/2025. Entregue um único arquivo **ZIP** contendo tanto o código reescrito quanto o relatório.

Boa diversão!