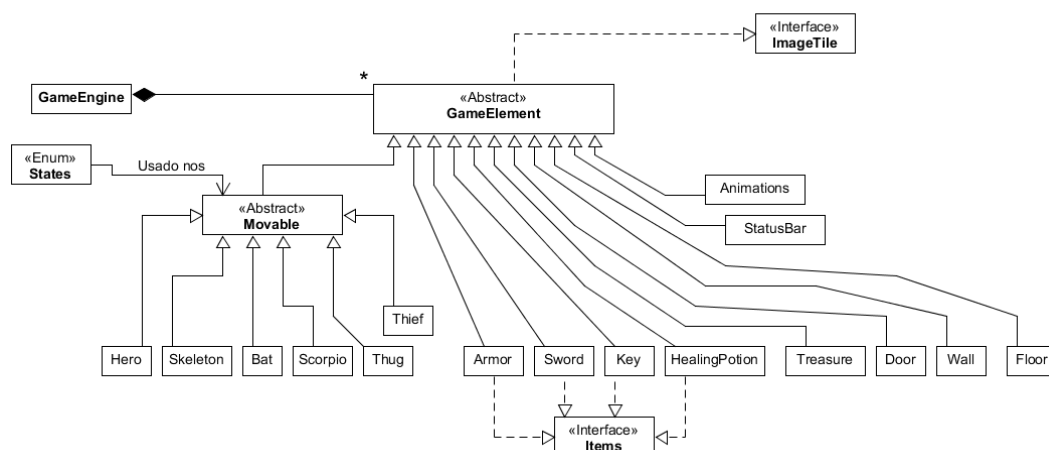


Identificação: Grupo 120-

Leonardo Andrade 99377

Durante a realização do projeto, houve decisões que cada aluno teve de tomar, visto que não havia nenhuma indicação de tal no enunciado que indicasse o contrário, e eram necessárias para o bom funcionamento do código.

Aqui em baixo deixo uma imagem UML do meu projeto e de seguida uma lista com as decisões opcionais que eu tomei ao longo deste trabalho. À partida conseguirá orientar-se bem com apenas o código e os comentários que se encontram no projeto, mas qualquer dúvida estará respondida neste relatório.



Bullet points (Por favor ler a nota importante no final do relatório)

❖ ***Métodos relevantes da classe principal GameEngine:***

- **start()** - Aparece uma mensagem para o jogador inserir o seu nome e verifica se inseriu um nome, caso tenha, o nome deve ser válido, podendo apenas conter caracteres alfanuméricos. Em caso de cancelamento terá a oportunidade de repetir o processo, mas se durante este processo todo cancelou um total de 3 vezes o jogo irá fechar.
- **getObject()** – Devolve uma lista dos objetos na posição dada. O herói aparece sempre no índice 0 devido à forma como o createLevel() está estruturado, de resto, se estiverem dois ou mais objetos por cima do chão e nenhum deles for o herói, o chão estará sempre no índice 0;
- **getAdversários()** – Devolve uma lista dos Adversários que estão presentes no nível atual;

- **getOpenDoors()** – Permite saber que portas ao longo do jogo inteiro é que estão desbloqueadas;
- createLevel().

❖ *Leitura do ficheiro (createLevel()):*

- Existem 2 exceções que são lançadas: Quando é o início do jogo e o ficheiro de mapa não é o “room0.txt”; Quando é invocado no método changeMap() e faz-se referência a um ficheiro e/ou pasta que não existe;
- É criado primeiro o herói, o terreno do mapa e por último os elementos do jogo;
- As imagens das portas são sempre as corretas para quando estão abertas, fechadas ou são corredores, em qualquer momento do jogo;
- Pode haver espaços entre cada linha de objetos nos ficheiros dos quartos.

❖ *Movimentação do Herói, Adversários e as suas interações:*

- Criação de uma classe abstrata “**Movable**”;
- Independentemente do motivo ou ação que levar as suas vidas a ficarem igual ou abaixo de 0, eles morrem sempre sem algum erro;
- Todos os Movable podem passar por cima de itens, mas apenas o Herói é que os consegue apanhar do chão;
- As percentagens de sucesso de ataque estão corretas para todos os Movable tal como o valor das suas vidas e dano dos seus ataques;
- Os morcegos podem curar-se até 3 vezes, curando-se sempre que conseguem danificar com sucesso o herói. Eles têm uma hipótese de 50% de movimentarem-se numa direção aleatória;
- Os escorpiões não danificam logo o herói na jogada em que o envenena, só apenas a partir da próxima jogada;
- Os esqueletos andam a cada 2 jogadas;
- O item que os ladrões roubam é sempre aleatório.
- Enquanto não têm pelo menos 1 item roubado, os ladrões andam sempre na direção do herói, e a partir do momento que roubam o seu primeiro item tentam ao máximo fugir sempre do herói, mas não é impossível roubarem mais do que 1 item durante a sua vida.

Quando estão a fugir do herói, a 1ª opção é movimentarem-se na direção oposta do herói (Cima). Se não poderem andar assim (Baixo), escolhem aleatoriamente uma das duas direções do eixo oposto (Esquerda e Direita). Se mesmo assim não poderem andar para nenhuma destas andam na mesma direção que o herói andou (Cima). Finalmente se depois de todas estas tentativas não encontrar um sítio por onde pudessem andar, então ficam parados;

Importante realçar que durante o processo de movimentação de fuga, em caso de os ladrões chocarem com o herói, eles vão tentar primeiro roubar o mesmo e só se verificarem que o herói não tem mais itens para serem roubados é que eles tentam movimentarem-se.

Por último, quando os ladrões morrem, todos os itens que estivessem no seu inventário são largados no mesmo sítio onde morreram.

- Esta classe contém atributos que incluem a vida, dano, direção do movimento, inventário (para movables que possam carregar itens), lista de estados (ter espada, estar envenenado, etc...) e finalmente um atributo que indica se é um ataque dos adversários ou do herói (É usado exclusivamente para as imagens extra no projeto, não tem implicações no funcionamento do ataque);
- Ainda em relação à classe Movable, o método move() diz respeito apenas aos adversários, enquanto que os outros são usados para todas as classes que a estendem;
- O método que permite ver se uma posição é válida para andar (canMove) trata de maior parte das interações.

Numa ideia geral, se poder andar, anda, caso contrário significa que atingiu uma parede/porta trancada ou então que tentou andar numa posição onde se encontra outro Movable. Se o outro em questão é o Herói, tenta atacá-lo.

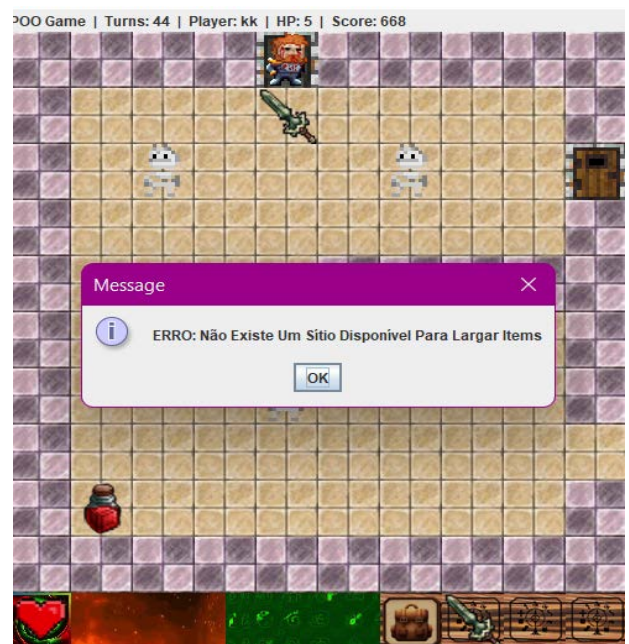
❖ *Apanhar e largar itens:*

- Quando o herói passa por cima de itens e tiver espaço no inventário, eles vão aparecer na zona inferior da tela. Quanto mais antigo for o item mais à esquerda aparece;
- Suponha-se que é removido o item que está na slot 2. Se por acaso tinha um item no slot 3 então este vai passar a ficar na posição à frente (2);
- Existem 2 métodos específicos para largar itens, removeFromInventory() e dropItem().
- O primeiro serve para remover qualquer efeito que os itens possam dar ao herói (mais dano, proteção extra, ...) e a posição onde eles são largados. Inicialmente procura uma

posição à volta do herói aleatoriamente, mas se nenhuma delas for válida larga na posição atual do herói. Em casos muito exclusivos não larga o item por não poder sequer largar na posição do nosso jogador (Ex: dentro de uma porta e a saída está ocupada).

O segundo método é responsável por selecionar corretamente qual item é que pretende remover/usar. Em casos onde tenta esvaziar incorretamente uma slot que já esteja vazia o código vai dar avisas para avisar da ocorrência e disponibiliza 2 opções ao jogador: inserir um nº de uma slot que não esteja vazia ou cancelar a jogada. Existem várias mensagens de aviso caso tente novamente remover um item de forma incorreta. No final deste método é invocado o primeiro, de forma a completar a remoção do item.

- É possível largar itens enquanto está dentro de uma porta, desde que a saída não esteja ocupada/bloqueada;
- Só é possível o dano do herói ser duplicado 1 vez, mesmo se tiver 2 ou mais espadas. Naturalmente só perde os efeitos de ter uma espada quando não tem mais nenhuma. É a mesma coisa para peças de armadura;
- As HealingPotions são guardadas no inventário e quando largadas, ou melhor dizendo, consumidas, funcionam tal como é indicado no enunciado;
- As chaves também podem ser largadas, e quando são usadas para desbloquear uma porta elas desaparecem por completo do jogo;
- A interface “**Items**” inclui um método default que examina e indica quantos items é que o herói atualmente tem de um certo tipo, que é dado como argumento.



❖ **StatusBar – Barra de vida e inventário:**

- inicialBar() - Cria a barra default de vida e inventário;
- updatedStatusBar() – Atualiza a barra de vida e inventário.

❖ *Mudança de sala:*

- **changeMap()** – Método de GameEngine. Copia tudo da tileList do nível atual, exceto os itens do inventário, e guarda num mapa essa cópia. Quando é a primeira vez a entrar no quarto que está indicado na porta, invoca o método createlevel(), caso contrário vai aceder ao mapa e vai meter na TileList a cópia mais recente do quarto pretendido;
- O método changeMap() é depois chamado dentro do método **changeRoom()** da classe “Door”. No move() do herói este segundo método é chamado caso o jogador tente andar para cima de uma porta.

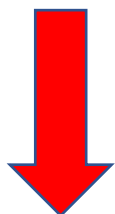
❖ *Pontuações:*

- O método **finishGame()** da classe do herói inclui o método das pontuações, **scoreboard()**. Este método vai primeiro ler o ficheiro que contem as melhores pontuações, de seguida é criado um mapa que contem os nomes dos jogadores e respetivos scores, incluindo o nome e score do jogador atual. Este mapa é organizado por ordem crescente e após isto rescreve-se o ficheiro das melhores pontuações. Pode apagar e editar linhas do ficheiro e não existe problemas, desde que para a edição apenas altere valores ou nomes e não modifique a estrutura das linhas (espaços entre cada palavra, pontos e setas).
- Se não encontrar o tesouro a pontuação do jogo é 0;
- Fórmula das pontuações:

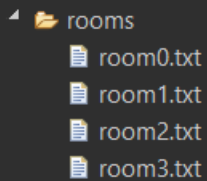
$$5 \times n^{\circ} \text{ jogadas} + 10 \times \text{dano total} + 3 \times \text{adversários mortos} + \text{vida do jogador}^{3.5}$$

❖ *Imagens novas e animações:*

- Todos os quartos têm um aspeto diferente, dando uma melhor representação da mudança de quartos;
- A classe “**Animations**” trata das animações dos: ataques do herói com e sem espada, cura da vida do herói através das HealingPotions e ataques dos adversários;
- O herói tem imagens diferentes dependendo da vida que tiver, e para além disso também pode ser diferente se estiver envenenado.
- Existem várias imagens novas por onde escolher, desde a armadura, espada, chão e paredes.



NOTA IMPORTANTE: Os ficheiros de mapas têm de estar **OBRIGATORIAMENTE** guardados no projeto dentro de pastas, e não como ficheiros separados. Usar pastas com um nome diferente de “rooms” não afeta o bom funcionamento do código, desde que altere também o nome no atributo `FIRSTROOM` de `GameEngine`.



```
rooms
├── room0.txt
├── room1.txt
├── room2.txt
└── room3.txt
```

```
public class GameEngine implements Observer {

    public static final int GRID_HEIGHT = 10;
    public static final int GRID_WIDTH = 10;
    private static final String FIRSTROOM = "rooms/room0.txt";
```