# Projeto Intercalar - Rogue

## 1 Introdução

O projeto consiste em criar um jogo similar ao Rogue (http://en.wikipedia.org/wiki/Roguelike). As principais características deste tipo de jogo são as seguintes:

- É um jogo de estratégia, baseado normalmente em cenários de fantasia;
- O personagem controlado pelo utilizador (herói) faz um movimento de cada vez;
- Para cada movimento do herói as criaturas (adversários) têm a possibilidade de fazer também um movimento;
- Normalmente envolve a passagem do herói por várias de salas em cada nível e por vários níveis até chegar ao último, onde se encontra o objetivo final do jogo;
- A "morte" do personagem implica voltar ao início, ou à última posição gravada. A gravação pode ser permitida apenas em certos pontos do jogo.

## 2 Objetivos

Usando a interface gráfica fornecida pelos formadores em anexo ao enunciado, deve criar o "motor de jogo" para um jogo deste tipo. O motor de jogo deve permitir jogar um jogo em que o herói percorre algumas salas, com adversários diferentes (com imagens, comportamentos e forças diferentes). Devem existir também objetos que possam ser apanhados pelo herói e que alterem o modo como este interage com os adversários ou a

sala (por exemplo uma espada que ajuda a dar mais dano aos adversários ou uma chave que permite abrir uma porta).

O herói apanha automaticamente todos os objetos pelos quais passa até um máximo de três e pode largar objetos (por exemplo usando as teclas de números 1, 2, 3). O herói não deverá poder atravessar alguns objetos (como paredes ou adversários). Mover-se para a mesma posição de um adversário é considerado um ataque e é dado ao adversário dano correspondente à capacidade do herói. Caso um adversário se mova para a posição do herói este deve sofrer o dano correspondente.

Sempre que o herói apanha um objeto ou sofre o ataque de um adversário, deve ser apresentada na barra informativa o valor da pontuação associada a essa ação.

Os adversários movimentam-se aleatoriamente quando estão "longe" do herói, e convergem para o herói quando estão "perto". Uma das personagens existentes é o "Thief". Esta personagem deve ter uma movimentação especifica sendo semelhante à peça de xadrez do bispo, ou seja, a sua movimentação deve ser feita apenas na diagonal.

O herói deve poder atacar adversários à distância desde que estejam na mesma linha ou coluna (por exemplo com uma bola de fogo, como as apresentadas na barra de estado na Figura 1). A bola de fogo deve ter efeito imediatamente, antes mesmo do movimento dos adversários.

Ao chegar ao fim do jogo é apresentado ao jogador uma pontuação, calculada com base em vários fatores. Por cada movimento, a pontuação desce em $1$, por cada item apanhado recebe $X$ pontos (definidos pelo tipo de item) e por cada inimigo destruído recebe $Y$ pontos (definidos pelo nível do inimigo). A pontuação nunca poderá ser menor que $0$. Ao chegar ao fim do jogo, deve ser mostrada essa pontuação (com uma *leaderboard* por exemplo) e deve ser mantido um registo das pontuações num ficheiro.

A barra de estado pode ser alterada desde que represente a vida e os três objetos que o herói tem na sua posse.

## 3 Requisitos

Os mapas das salas devem ser lidos de um ficheiro com o formato indicado na secção "Ficheiros de salas". Para facilitar os testes, não se recomenda que a primeira sala tenha adversários.

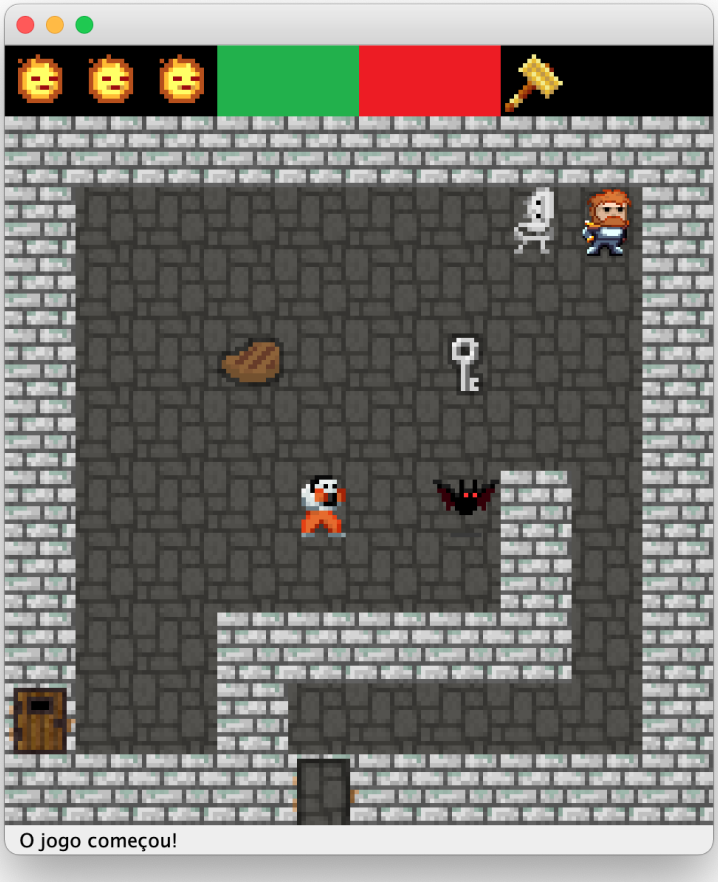


Figura 1: Exemplo da aplicação em funcionamento, com uma sala com uma porta, um "herói" (a personagem principal deste jogo), duas criaturas (morcego e lutador) que tentarão impedir o herói de atingir a porta de saída. Na barra de estado é indicado o número de bolas-de-fogo disponíveis, o nível de saúde do herói (vermelho é a saúde perdida e verde a que resta) e os objetos que tem na sua posse. As imagens usadas neste jogo são do jogo Pixel Dungeon (http://pixeldungeon.watabou.ru/) e encontram-se online em http://pixeldungeon.wikia.com/.

O motor de jogo recebe da interface gráfica informações sobre as teclas pressionadas. As teclas das setas devem controlar a posição do jogador, o espaço deve ser usado para disparar e as teclas 1 a 3 devem ser usadas para largar objetos na posição respetiva.

O motor de jogo pode enviar imagens para a janela usando implementações de `ImageTile` (como as classes `Hero` e `Floor`, do exemplo fornecido). De cada vez que há

uma alteração (por exemplo, mudança das posições das imagens) deve ser chamado o `update()` da interface gráfica.

A interacção entre objectos deve ser tão autónoma quanto possível (passando o mínimo possível pelo motor de jogo). O formato dos ficheiros onde são guardadas as melhores pontuações é livre.

As classes fornecidas não devem ser alteradas e são fundamentais para a resolução do trabalho.

É obrigatória a utilização de:

- herança de propriedades;
- classe(s) abstracta(s);
- implementação de interface(s);
- listas;
- leitura e escrita de ficheiros;
- excepções (lançamento e tratamento).

## 4 Ficheiro de salas

Nas Figuras 2 e 3 pode ver os ficheiros de configuração de duas salas, que deverão ser lidos pela aplicação no momento de inicialização do jogo. A primeira sala tem apenas paredes ('W') e três portas (indicadas no mapa com 0, 1 e 2). Repare que o ficheiro tem $10 \times 10$ caracteres (excluindo os comentários iniciais), exatamente o número de posições que é possível mostrar na interface gráfica. Os cardinais '#' são usados para sinalizar linhas que não fazem parte do mapa, neste caso contêm a definição das portas. A configuração da porta 0 está na linha `"# 0 D room1.txt 1"` onde é indicado que a porta 0 é do tipo 'D' neste caso quer dizer que tem mesmo uma porta (o tipo 'E' indica uma passagem sem porta) e vai dar à sala descrita no ficheiro `room1.txt`, à porta `1`. Ou seja quando o herói desaparecer por esta porta vai sair à porta `1` da sala descrita no ficheiro `room1.txt`. A porta `1`, descrita na segunda linha, indica a chave que é preciso obter para passar aquela porta, a `key1`.

```
#
# 0 D room1.txt 1
# 1 D room1.txt 0 key1
# 2 E room2.txt 0
#
WWWW0WWWWW
W         W
W         W
W         W
W         W
W       W W
W       W W
W       W W
W       W 1
WWWW2WWWWW
```

Figura 2: Exemplo de ficheiro de sala `1`

No exemplo seguinte, temos uma sala com vários adversários (S = Skeleton, B = Bat , G = BadGuy) e um objeto (s = Sword), além de duas portas.

## 5 Interface Gráfica

A interface gráfica fornecida (que consiste na classe `ImageMatrixGUI` e na interface `ImageTile`) permite abrir uma janela com três áreas: uma barra de estado no topo e no centro a janela de jogo e por baixo uma pequena barra informativa. Na Figura 1 pode ver a re-presantação de duas dessas áreas: a barra de estado e a janela de jogo.

A janela de jogo tem um tamanho de $480 \times 480$ pixels que pode ser vista como uma grelha de $10 \times 10$ posições para imagens de $48 \times 48$ pixels.

A barra de estado tem um tamanho de $(480 \times 48)$ e pode ser vista com uma barra onde pode mostrar $10$ imagens de $48 \times 48$. As imagens que são usadas fazem parte de uma "biblioteca" que se encontra na pasta "imagens" dentro do projeto.

A barra informativa apresenta o tamanho necessário para apresentar uma linha de informação.

A interface `ImageTile` é usada para seleccionar uma imagem e indicar a sua posição na matriz de imagens onde que se irá inserir.

```
#
# 0 D room0.txt 1 key1
# 1 D room3.txt 0
#
WWWWWWWWWW
WS   G   W
W        W
W       sW
W        W
W      W W
W    B W W
W  WWWWW W
0  W     W
WWWW1WWWWW
```

Figura 3: Exemplo de ficheiro de sala 2

Ao implementar o interface `ImageTile` é obrigatório indicar o nome da imagem (sem extensão) e a sua posição na grelha de $10 \times 10$ que constitui a janela visível (ou na grelha de $10 \times 1$ que constitui a barra de estados).

São usadas as funções `public void newImages(List<ImageTile>...)` e `public void newStatusImages(List<ImageTile>...)` para adicionar imagens à janela de jogo e à janela de estado (respectivamente).

Pode, se quiser, usar outras imagens embora esse não seja um dos objetivos do trabalho, por isso não é contabilizado, nem serão prioritárias as dúvidas ou erros gerados pela utilização de outras imagens. Todas as imagens usadas devem ter $48 \times 48$ pixels.

## 6 Lançamento de Bolas de Fogo

Interface `FireTile` descreve dois métodos: o método `setPosition()` e o `validateImpact()` que é chamado cada vez que a bola de fogo se move. Este método deve ser responsável por ver se a bola de fogo atingiu algum objeto e fazê-la "explodir". Também deve ser responsável por fazer o inimigo perder a vida, caso a bola de fogo o atinja.

`FireBallThread` recebe uma direção e uma `FireTile` e tem como objetivo mudar a posição da bola de fogo de $300$ ms em $300$ ms. Cada vez que a bola de fogo se move, o método `validateImpact()` da `FireTile` é invocado. Caso o método `validateImpact()` devolva `true`, então o processo continua normalmente. Caso devolva `false`, o ciclo termina e a bola de fogo é eliminada da Gui.

## 7 Execução do Trabalho

É encorajada a discussão entre colegas sobre o trabalho, mas estritamente proibida a troca de código. A partilha de código em trabalhos diferentes será penalizada. Serão usados meios de verificação de plágio e os trabalhos plagiados serão comunicados à organização do curso.

Peça regularmente ao formador para que reveja consigo o trabalho, quer durante as aulas, quer no decorrer do trabalho autónomo. Desse modo, pode evitar algumas más opções iniciais que podem aumentar muito a quantidade de trabalho.

## 8 Entrega

O projeto deverá ser entregue até às 23h59 do dia da Aula 23. No dia do primeiro teste, os projetos serão vistos individualmente em aula e serão classificados de A a C consoante as funcionalidades implementadas e a funcionar corretamente.

**As funcionalidades necessárias para alcançar cada um dos patamares da nota será partilhada posteriormente.**

Depois da avaliação, todos os projetos e respetivo código serão analisados individualmente pelos formadores e passados por um detetor de plágio, estando a nota atribuida na avaliação inicial sujeita a alteração.

## 9 Avaliação

O trabalho será classificado com A, B ou C. A avaliação será feita do seguinte modo:

Serão **excluídos** (não aceites para discussão e classificados com zero valores) os trabalhos que:

- Apresentem erros de sintaxe;
- Apresentem um funcionamento muito limitado, sem que componentes essenciais do projeto estejam implementadas. Por exemplo, se a gestão de colisões dos objetos não está implementada é considerado que o trabalho não está utilizável;
- Não é utilizada herança (por exemplo no comportamento e características das criaturas) e coleções (por exemplo para guardar os vários objetos presentes numa cena);
- O comportamento de todo do programa é concentrado num método, numa classe ou não demonstrem que sabem usar corretamente classes e objetos para modularizar o código.

Serão classificados com **C**, os trabalhos que:

- Forem jogáveis numa sala, com uma personagem que se move de acordo com as decisões do utilizador e (pelo menos) dois tipos diferentes de criaturas que exibem algum tipo de comportamento diferente;
- Usem corretamente classes, herança e coleções;
- O comportamento do programa não está maioritariamente concentrado num ou dois métodos / classes.

Serão classificados com **B**, os trabalhos com uma estrutura interna razoável, nomeadamente:

- Classes desenhadas de acordo com as boas práticas;
  - utilização das coleções adequadas para cada situação;
  - boa proteção dos atributos de cada classe;
  - métodos de tamanho razoável com um único objetivo;
- Que permitam, em geral, jogar um jogo até ao fim, em que a personagem tenha meios de causar dano às criaturas à distância, com pelo mais de duas salas (idealmente deveria ter cerca de seis salas em dois níveis diferentes).

Serão classificados com **A** os trabalhos que:

- Tenham uma estrutura interna que, além dos pontos anteriores, permita a implementação do jogo de modo realista (i.e. que resista bem ao escalamento para muitas salas e níveis distintos). A demonstração não precisa de ter mais de seis salas em dois níveis;
- Gravem e recuperem todas as salas por onde o personagem passou até ao momento, bem como o estado do jogo em cada sala, nomeadamente a saúde das criaturas em jogo;
- Gravem e carreguem de um ficheiro as pontuações máximas alcançadas no jogo.
- Devem saber usar adequadamente exceções para tratar erros de programação e situações de exceção;
- Elementos extra.

**Bom trabalho!**

## Anexos

| Modifier and Type | Method and Description |
|---|---|
| ImageMatrixGUI | getInstance()<br>Access to the Singleton instance of ImageMatrixGUI. |
| void | setEngine(Engine engine)<br>Set the game engine that will feed the GUI. |
| void | setStatus(String text)<br>Display a text on the status portion of the game window. |
| void | go()<br>Launch the game window. |
| void | newImages(List<? extends ImageTile> newImages)<br>Add a new set of images to the main window. |
| void | removeImage(ImageTile image)<br>Removes the image given as a parameter. |

| Modifier and Type | Method and Description |
| --- | --- |
| void | addImage(ImageTile image)<br>Add image to main window. |
| void | clearImages()<br>Clear all images displayed in main window. |
| void | newStatusImages(List<ImageTile> newImages)<br>Add a new set of images to the status window. |
| void | addStatusImage(ImageTile image)<br>Removes the image given as a parameter from the status bar. |
| void | removeStatusImage(ImageTile image)<br>Adds image to status window. |
| void | clearStatus()<br>Clear all images displayed in status window. |
| void | showMessage(String title, String message)<br>Displays a window with message information to the user. |
| String | showInputDialog(String title, String message)<br>Displays a window with a question for the user and returns their answer. |
| void | update()<br>Force scheduling of a new window paint this may take a while, it does not necessarily happen immediately after this instruction is issued) |