Projecto Guiado – 2ª Iteração
Estruturação em Classes

2.1 [Separação em camadas] Restruturar o programa por forma à interacção com o utilizador e a lógica do jogo serem tratadas em classes separadas. Isto é importante para facilitar posteriormente a existência em paralelo de múltiplas formas de usar/exercitar a lógica do jogo: através da interface alfanumérica (já desenvolvida na aula anterior), através de uma interface gráfica (a criar numa próxima aula), através de testes unitários automáticos (idem). Organizar o programa em *packages* distintos, como por exemplo: maze.logic e maze.cli (*command line interface*). No arranque do programa, deve ser possível começar o jogo com o exemplo pré-definido (da aula anterior) ou com um labirinto gerado automaticamente.

2.2 [Estruturação em classes] Reorganizar o *package* com a lógica do jogo (e outros *packages* se necessário) com pelo menos as seguintes classes (com todos os campos privados ou protegidos):

- **a)** Uma classe para representar o estado do jogo, agregando o labirinto em si os elementos presentes no labirinto, e disponibilizar as operações de comando do jogo pelo utilizador e consulta do estado do jogo;
- **b)** Opcionalmente, uma classe para representar o labirinto em si (terreno);
- **c)** Classes para representar os vários tipos de elementos que podem estar presentes no labirinto (espada, dragão, herói), com o respectivo estado, e uma super classe com as propriedades comuns (posição, etc.), tirando o mais possível partido de herança e polimorfismo (devendo o comportamento dos vários elementos ser distribuído o mais possível pelas classes respetivas);
- **d)** Uma classe para o gerador de labirintos (ver o padrão BUILDER).[1]

Sugestão: fazer um rascunho de diagrama de classes e discutir com o docente eventuais alternativas.

2.3 Criar uma nova estratégia em que o dragão pode adormecer por algum tempo de forma aleatória, tendo um visual diferente quando está a dormir. Quando o dragão está a dormir, o herói pode matar o dragão se estiver armado, mas não pode ser morto pelo dragão. No interface alfanumérico, representar por letra minúscula quando está a dormir. No arranque do programa ou no início do jogo, o utilizador deve poder escolher a estratégia pretendida (dragão parado, dragão com movimentação aleatória, dragão com movimentação aleatória intercalada com dormir).

2.4 Criar a possibilidade de existência de mais do que um dragão.

2.5 [Para casa] Pode existir uma águia para ajudar o herói a apanhar a espada, comandada pelo herói (isto é, pelo uilizador), com o seguinte comportamento:

- inicialmente a águia está poisada no braço do herói e acompanha-o;
- por ordem do herói, a águia pode levantar voo em direção à espada, pelo caminho mais próximo possível de uma linha reta;
- quando está a voar, a águia pode estar sobre qualquer quadrícula; ao mostrar o estado do labirinto, pode convir usar dois carateres para cada quadrícula;
- quando chega à quadrícula da espada, a águia desce para apanhar a espada (se ainda aí estiver); se um dragão estive acordado nessa posição ou adjacente, mata a águia;
- assim que pega a espada, a águia levanta voo de novo em direção à posição de partida (onde levantou voo do braço do herói);
- voltando à posição de partida, se não estiver aí o herói, a águia permanece no solo até o herói a apanhar, correndo o risco de ser morta por um dragão.

[1] Também de pode criar um interface MazeBuilder com duas classes que o implementam: uma para o gerador aleatório e outra para o labirinto por defeito.