Projeto 2 de Linguagens de Programação I 2023/2024

Versão 1.0, 13 Maio 2024, 12h15

Introdução

Os grupos devem implementar o jogo *Yet Another Dungeon Crawler*, na forma de uma aplicação de consola em C# 8.0 compatível com .NET Standard 2.1. O jogo deve usar apenas a linha de comandos e descrições de texto.

Leiam o enunciado com atenção!

Objetivo

Yet Another Dungeon Crawler é um explorador de masmorras em que o jogador explora um conjunto de áreas de uma masmorra à procura de tesouros. Em cada uma das áreas da masmorra, o jogador pode encontrar inimigos que o atacam, ou items que ajudam na sua demanda. O jogo acaba quando o jogador morrer ou sair da masmorra, regressando à sua área inicial.

No jogo devem existir, pelo menos, os seguintes tipos ou conceitos:

- *Player* - Representa o jogador e deve conter características tais como `Health` e `AttackPower`, e várias ações possíveis, nomeadamente `Move`, `Attack`, `PickUpItem` e `Heal`.
- *Enemy* - Representa os inimigos. Tem características tais como `Health` e `AttackPower` e pode realizar a ação `Attack`. Deve existir pelo menos um tipo de inimigo.
- *Item* - Representa os items. Deve existir pelo menos o tipo de item, `HealthPotion`, que restora *x* pontos de `Health` ao jogador.
- *Room* - Representa cada uma das áreas da masmorra. Cada um pode conter um inimigo, um item ou ambos.

Outros aspetos importantes do jogo:

- Cada área da masmorra tem de estar ligada a pelo menos uma outra área.
- Quando o jogador entra numa área ocupada por um inimigo, só pode progredir depois de o derrotar.
- O mapa da masmorra é fixo e a disposição das áreas é decidida pelo grupo. **O mapa deve ser incluido no relatório**.

Requisitos do código

O código do projeto deve estar implementado da seguinte forma:

- O código deve estar organizado usando uma abordagem MVC (**obrigatório**).

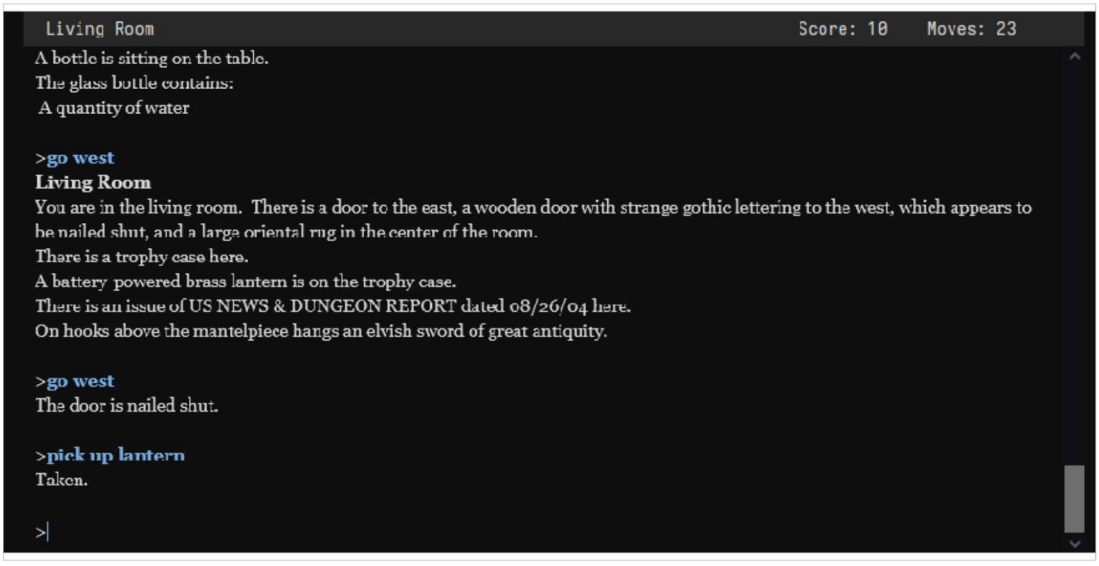
- As classes devem representar os conceitos do jogo, seguindo boas práticas de encapsulação.
- As classes, `structs` e enumerações devem ser colocadas num ficheiro com o mesmo nome (e.g. `Player` deve estar no ficheiro `Player.cs`).
- Cada classe deve ter uma responsabilidade específica e bem definida.
- O código deve ser bem estruturado para que seja simples adicionar mais inimigos ou itens.
- O código deve fazer uso de bons princípios de programação orientada por objetos, tais como **polimorfismo** e **interfaces**.

Interface de utilizador e visualização

A interface de utilizador (UI) é um dos aspetos mais importantes de qualquer jogo, permitindo ao jogador jogar sem obstruções, sabendo a cada momento o que se passa e o que aconteceu.

Durante o jogo é importante realçar informações chave para saber o que se passa e oferecer ao jogador informação suficiente para planear a sua próxima jogada. Considerando que este projeto irá usar a linha de comandos é importante considerar este aspeto antes de começar a fazer a implementação!

Exemplo



Funcionamento da Aplicação

O funcionamento da aplicação é da responsabilidade de cada grupo. No entanto, quando a aplicação inicia, **deve ser claro como o jogador joga o jogo**. Ou seja, o jogo deve ter

instruções muito claras sobre que comandos fazem o quê. Devem ter em conta as regras importantes de *game design*, pois serão tidas em conta na avaliação do projeto.

Podem e devem fazer uso de cores na consola, bem como de carateres Unicode para clarificar as diferenças entre peças e tornar o jogo mais apelativo.

A aplicação deve funcionar em Windows, macOS e Linux. Podem testar o jogo em WSL (Linux em Windows) ou em Linux numa máquina virtual para garantir essa compatibilidade. Algumas instruções incompatíveis com macOS e Linux são, por exemplo:

- Console.Beep()
- Console.SetBufferSize()
- Console.SetWindowPosition()
- Console.SetWindowSize()
- Entre outras.

As instruções que só funcionam em Windows têm a seguinte indicação na sua documentação:

> PlatformNotSupportedException
>
> The current operating system is not Windows.

Not Supported

Objetivos e critério de avaliação

Este projeto tem os seguintes objetivos:

- **O1** - Programa deve funcionar como especificado e deve ter em conta as regras básicas do *game design*. Atenção aos detalhes, pois é fácil desviarem-se das especificações caso não leiam o enunciado com atenção.
- **O2** - Projeto e código bem organizados, nomeadamente:
  - Código organizado segundo o indicado na secção Requisitos do código.
  - Código devidamente comentado e indentado.
  - Inexistência de código "morto", que não faz nada, como por exemplo variáveis, propriedades ou métodos nunca usados.
  - Projeto compila e executa sem erros e/ou *warnings*
- **O3** - Projeto adequadamente documentado com comentários de documentação XML.
- **O4** - Repositório Git deve refletir boa utilização do mesmo, nomeadamente:
  - Devem existir *commits* de todos os elementos do grupo, *commits* esses com mensagens que sigam as melhores práticas para o efeito (como indicado aqui, aqui, aqui e aqui).
  - Ficheiros binários não necessários, como por exemplo todos os que são criados nas pastas `bin` e `obj`, bem como os ficheiros de configuração do Visual Studio

(na pasta `.vs` ou `.vscode`), não devem estar no repositório. Ou seja, devem ser ignorados ao nível do ficheiro `.gitignore`.
  - Eventuais *assets* binários necessários, como por exemplo o **mapa do das masmorras** ou outras imagens ilustrativas do projeto (opcionais), devem ser integrados no repositório em modo Git LFS.
- **O5** - Relatório em formato Markdown (ficheiro `README.md`), organizado da seguinte forma:
  - Título do projeto.
  - Autoria:
    - Nome dos autores (primeiro e último) e respetivos números de aluno.
    - Informação de quem fez o quê no projeto. Esta informação é **obrigatória** e deve refletir os *commits* feitos no Git.
    - Indicação do repositório Git utilizado. Esta indicação é opcional, pois podem preferir manter o repositório privado após a entrega.
  - Arquitetura da solução:
    - Descrição da solução, com breve explicação de como o código foi organizado, bem como dos algoritmos não triviais que tenham sido implementados.
    - Um diagrama UML de classes simples (i.e., sem indicação dos membros da classe) descrevendo a estrutura de classes. Para o efeito devem usar o Mermaid diretamente no vosso `README.md`.
  - Referências, incluindo trocas de ideias com colegas, código fornecido por IAs generativas (indiquem também os *prompts* utilizados para obter esse mesmo código, bem como o sistema utilizado, i.e. ChatGPT, GPT4, etc.), código aberto reutilizado (e.g., do StackOverflow) e bibliotecas de terceiros utilizadas. Devem ser o mais detalhados possível.
  - **Nota:** o relatório deve ser simples e breve, com informação mínima e suficiente para que seja possível ter uma boa ideia do que foi feito. Atenção aos erros ortográficos e à correta formatação Markdown, pois ambos serão tidos em conta na nota final.

O projeto tem um peso de 3 valores na nota final da disciplina e será avaliado de forma qualitativa. Isto significa que todos os objetivos têm de ser parcialmente ou totalmente cumpridos. A cada objetivo, O_1 a O_5, será atribuída uma nota entre 0 e 1. A nota do projeto, N, será dada pela seguinte fórmula:

$$N = 3 \times O_1 \times O_2 \times O_3 \times O_4 \times O_5 \times D$$

Em que D corresponde à nota da discussão e percentagem equitativa de realização do projeto, também entre 0 e 1. Isto significa que se os alunos ignorarem completamente um dos objetivos, não tenham feito nada no projeto ou não comparecerem na discussão, a nota final será zero.

Entrega

O projeto deve ser entregue por **grupos de 2 a 3 alunos** via Moodle até às **23h00 de 26 de Maio de 2024**. Um (e apenas um) dos elementos do grupo deve submeter um ficheiro `zip` com a solução completa, nomeadamente:

- Pasta escondida `.git` com o repositório Git local do projeto.
- Ficheiro da solução (`.sln`).
- Pasta do projeto, contendo os ficheiros `.cs` e o ficheiro do projeto (`.csproj`).
- Ficheiro `README.md` que consiste no relatório do projeto em formato Markdown, que deve incluir o diagrama UML de classes (embebido no `README.md` em código Mermaid), bem como o **mapa das masmorras**.
- Outros ficheiros de configuração, como por exemplo `.gitignore` e `.gitattributes`.

Muito importante: Antes de criarem o ficheiro `zip` devem limpar o repositório de ficheiros temporários com o comando `git clean -dfx` (atenção que este comando limpa tudo o que não estiver no repositório, portanto verifiquem se não existem *commits* por fazer; podem correr um *preview* não destrutivo deste comando com `git clean -ndfx`).

Não serão avaliados projetos sem estes elementos e que não sejam entregues através do Moodle.

Honestidade académica

Nesta disciplina, espera-se que cada aluno siga os mais altos padrões de honestidade académica. Isto significa que cada ideia que não seja do aluno deve ser claramente indicada, com devida referência ao respetivo autor. O não cumprimento desta regra constitui plágio.

O plágio inclui a utilização de ideias, código ou conjuntos de soluções de outros alunos ou indivíduos, ou quaisquer outras fontes para além dos textos de apoio à disciplina, sem dar o respetivo crédito a essas fontes. Os alunos são encorajados a discutir os problemas com outros alunos e devem mencionar essa discussão quando submetem os projetos. Essa menção **não** influenciará a nota. Os alunos não deverão, no entanto, copiar códigos, documentação e relatórios de outros alunos, ou dar os seus próprios códigos, documentação e relatórios a outros em qualquer circunstância. De facto, não devem sequer deixar códigos, documentação e relatórios em computadores de uso partilhado, e muito menos usar repositórios Git públicos (embora os mesmos possam ser tornados públicos 12h após a data limite de submissão).

Sobre IAs generativas (e.g. ChatGPT): podem usar este tipo de ferramentas para esclarecer dúvidas, ou até para obterem uma ou outra sugestão de código, desde de que as ideias e organização geral do projeto sejam originais. Código completamente fornecido por uma IA generativa será facilmente detetável pelas ferramentas de deteção de plágio, pelo que sugerimos muito cuidado no uso deste tipo de ferramentas. De qualquer forma, todo o código gerado através de IA (parcialmente ou totalmente) deve ser indicado em forma de comentário e nas referências do relatório, e sempre com indicação do *prompt* e do sistema de IA utilizados.

Nesta disciplina, a desonestidade académica é considerada fraude, com todas as consequências legais que daí advêm. Qualquer fraude terá como consequência imediata a anulação dos projetos de todos os alunos envolvidos (incluindo os que possibilitaram a ocorrência). Qualquer suspeita de desonestidade académica será relatada aos órgãos superiores da escola para possível instauração de um processo disciplinar. Este poderá resultar em reprovação à disciplina, reprovação de ano ou mesmo suspensão temporária ou definitiva da Universidade Lusófona.

Texto adaptado da disciplina de Algoritmos e Estruturas de Dados do Instituto Superior Técnico.

Referências

- [1] Whitaker, R. B. (2022). **The C# Player's Guide** (5th Edition). Starbound Software.
- [2] Albahari, J., & Johannsen, E. (2020). **C# 8.0 in a Nutshell**. O'Reilly Media.
- [3] Dorsey, T. (2017). **Doing Visual Studio and .NET Code Documentation Right**. Visual Studio Magazine. Retrieved from https://visualstudiomagazine.com/articles/2017/02/21/vs-dotnet-code-documentation-tools-roundup.aspx.

Licenças

Metadados

- Autores: Ana Pinha, Nuno Fachada, Afonso Oliveira
- Curso: Licenciatura em Videojogos
- Instituição: Universidade Lusófona - Centro Universitário de Lisboa