Enunciado do projeto de POO 2022/23 (v1.0)

Rogue

Introdução

Neste projeto pretende-se criar um jogo do género conhecido como *Roguelike*[1], utilizando os principais conceitos de POO lecionados ao longo do semestre.

As principais características de um jogo deste tipo são:

- É um jogo de estratégia *turn-based* (*i.e.*, uma jogada em cada ciclo), tipicamente com cenários de fantasia.
- A personagem controlada pelo utilizador – o herói – faz um movimento em cada jogada. Por cada movimento do herói os seus adversários têm a possibilidade de fazer também um movimento.
- Normalmente o herói atravessa várias salas/cenários em cada nível, e percorre 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.

No final do semestre, o jogo desenvolvido deverá permitir que o herói percorra várias salas, onde existem adversários de diferentes tipos, com características e comportamentos distintos. Deverão também existir objetos que possam ser apanhados pelo herói e que alterem o modo como este interage com os adversários ou com outros objetos da sala (como por exemplo uma espada que ajuda a causar maior dano aos adversários ou uma chave que possa abrir uma porta).

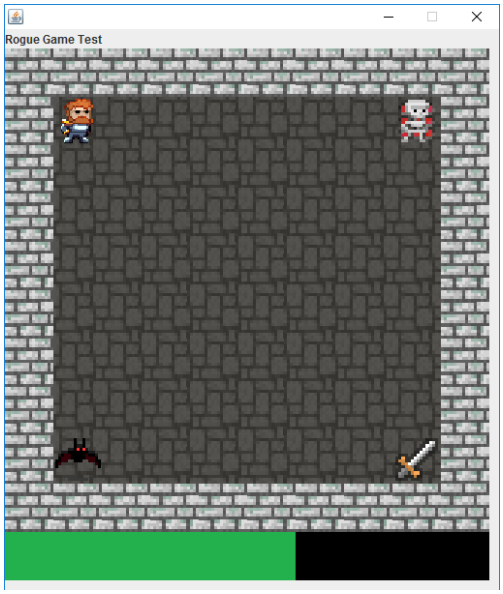

Fig. 1 - Exemplo do interface gráfico do jogo. As imagens usadas são provenientes do jogo Pixel Dungeon[2,3] e estão disponíveis no pacote fornecido com a interface gráfica.

[1] https://en.wikipedia.org/wiki/Roguelike
[2] http://pixeldungeon.watabou.ru/
[3] http://pixeldungeon.wikia.com/

Descrição do jogo e implementação

Funcionamento do jogo e requisitos funcionais

Como o jogo é *turn-based*, cada jogada (ciclo de execução) é iniciada de cada vez que o utilizador prime uma tecla. Assim, em cada jogada o herói deverá mover-se segundo a tecla direcional que tiver sido pressionada, ou interagir com adversários ou outros objetos caso algum destes ocupe a posição de destino.

Sugere-se que cada jogada comece pela ação do herói. Seguem-se as ações dos adversários, de acordo com os seus comportamentos específicos. Devem ser executados de imediato os efeitos resultantes dessas ações, quer estas sejam movimentos ou interações com o herói. Para algumas interações, a ordem de execução é fundamental para que o efeito seja o esperado. A interação entre os vários objetos do jogo deve ser tão autónoma quanto possível, distribuindo-se o código pelas várias classes correspondentes e minimizando-se a quantidade de código presente na classe principal do jogo (`GameEngine`).

O funcionamento do jogo deverá cumprir as seguintes regras:

- Tanto o herói como os adversários não podem atravessar paredes ou ocupar posições onde esteja alguma personagem do jogo. Podem, no entanto, passar por cima de objetos "apanháveis", como a espada ou as chaves.
- O herói apanha automaticamente objetos "apanháveis" caso passe por cima deles e tenha espaço disponível para os guardar (pode guardar até três objetos). Usando as teclas numéricas 1, 2 e 3 o herói poderá largar equipamento ou consumir poções.
- Quando o herói se tenta mover para a mesma posição de um adversário, essa ação é considerada um ataque e é aplicado um valor de dano ao adversário. De modo análogo, quando um adversário se move para a posição do herói, este deve sofrer o dano correspondente.
- Caso os *hitpoints* de um adversário cheguem a zero, este desaparece. Caso os *hitpoints* do herói cheguem a zero, o jogo termina.
- O herói muda de sala quando se move para uma porta que esteja destrancada. Quando se move para uma porta trancada tendo a chave correspondente na sua posse, a porta é destrancada (caso não tenha a chave, a porta comporta-se como uma parede).

Tenha em atenção que o herói poderá voltar atrás a salas por onde já passou e que estas devem estar no estado em que o herói as deixou e não no estado inicial.

A título de exemplo, a Fig. 2 representa um possível estado do jogo após algumas jogadas, tendo partido do estado inicial representado na Fig. 1. Nesta fase do jogo tanto o herói como os adversários já se movimentaram, o herói perdeu alguns *hitpoints* por contacto com uma das criaturas (parte vermelha da barra de saúde) e apanhou a espada (esta deixou de estar no chão e aparece agora na barra de estado)

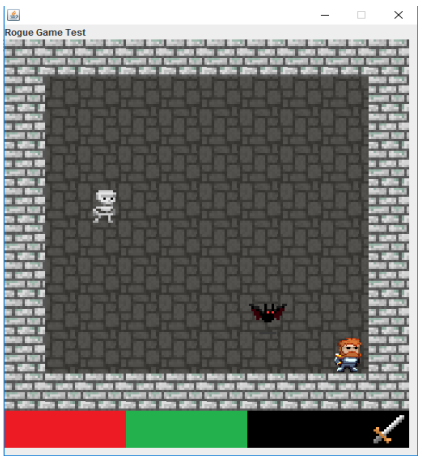

Fig. 2 – Exemplo do estado do jogo após algumas jogadas.

O final do jogo pode dar-se por morte do herói, ou chegando à posição onde está um objeto especial – o tesouro. No final do jogo deve ser atribuída uma pontuação ao jogador, baseada por exemplo no número de adversários que eliminou ou nos danos que lhes infligiu. Deve ser mantido um ficheiro de pontuações onde consta um Top-5 das melhores pontuações, associadas aos nomes dos jogadores que as obtiveram.

```
##########
#        #
#        #
####   ###
#        #
#        #
#        #
#  #     #
#  #
#### #####

Bat,8,1
Skeleton,1,8
Sword,8,8
Key,4,4,KEY1
Door,4,9,room1,4,0
Door,9,8,room2,0,8,KEY1
```

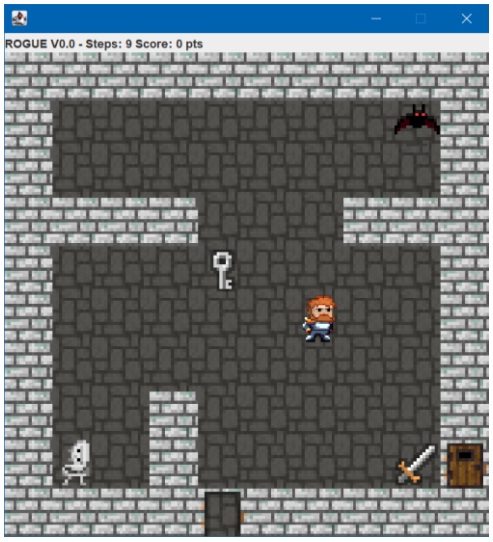

Fig. 3 – Exemplo de um ficheiro de configuração da sala inicial e correspondente representação gráfica.

Leitura dos elementos de jogo

O estado inicial de cada sala é determinado por um ficheiro de configuração de sala que segue o formato ilustrado na Fig. 3.

As primeiras linhas do ficheiro contêm um "mapa" que corresponde à configuração das paredes da sala, onde os cardinais (#) representam paredes, e os espaços em branco representam zonas vazias (chão). Assume-se que todas as salas têm um tamanho fixo de 10 x 10 células.

Depois do mapa segue-se uma descrição dos objetos presentes na sala. Exceto nos casos mais específicos de portas e chaves todos os elementos seguem o seguinte formato:

```
<Nome_do_elemento>,<x>,<y>
```

Onde x e y são as coordenadas da posição inicial dos elementos.

No caso das chaves (`Key`), a descrição inclui também um identificador (`ID`) para a chave em questão:

```
Key,<x>,<y>,<ID>
```

O identificador é alfanumérico e serve para se poder associar uma chave a uma porta específica. Para as portas, a descrição inclui mais campos de informação, pois é necessário indicar qual a sala e posição aonde vai dar a porta e, nos casos de portas trancadas, qual o ID da chave que a abre:

```
Door,<x>,<y>,<Sala_de_destino>,<x_dest>,<y_dest>[,Key_ID]
```

A `Sala_de_destino` terá o nome do ficheiro que descreve a sala de destino (sem a extensão "`.txt`"), `x_dest` e `y_dest` são as coordenadas da posição aonde a porta vai dar na sala de destino. Adicionalmente, e apenas para as portas trancadas, é fornecido o ID da chave que a abre através do campo de informação `Key_ID`.

Cada sala é descrita por um ficheiro com o nome "`roomN.txt`" em que `N` é o número da sala. É aconselhada a implementação de uma "fábrica de objetos" para criar os elementos de jogo. A sala onde o jogo começa é sempre a "`room0.txt`" e o herói começa sempre numa posição fixa (1, 1). Ao mudar de sala, o herói deve começar na posição da porta por onde entra na nova sala.

Características e comportamento dos elementos de jogo

As principais características dos elementos jogo são as seguintes:

- **Herói (Hero)**
  - começa o jogo com 10 *hitpoints* (barra verde cheia);
  - ao atacar um adversário dá 1 ponto de dano. Se tiver equipamento pode causar maior dano.
- **Esqueleto (Skeleton)**
  - começa com 5 *hitpoints*;
  - ao atacar o herói dá 1 ponto de dano;
  - só se movimenta uma vez por cada duas jogadas e sempre na direção do herói.
- **Morcego (Bat)**
  - começa com 3 *hitpoints*;
  - dá 1 ponto de dano ao herói 50% das vezes que o ataca. Sempre que o morcego ataca com sucesso restaura um 1 *hitpoint* (até ao máximo de 3);
  - movimenta-se em todas as jogadas, 50% das vezes na direção do herói e 50% das vezes numa direção aleatória. Se o movimento for na direção de um obstáculo intransponível, não se mexe.
- **Brutamontes (Thug)**
  - começa com 10 *hitpoints*;
  - em todas as jogadas tenta mover-se em direção ao herói;
  - dá 3 pontos de dano ao herói em 30% das vezes em que o ataca.
- **Espada (Sword)**
  - dano dado pelo herói aos seus adversários é duplicado enquanto tiver a espada.
- **Armadura (Armor)**
  - quando o herói tem a armadura, 50% dos ataques que lhe são dirigidos com sucesso não causam qualquer dano.
- **Poção da cura (HealingPotion)**
  - ao usar uma poção de cura, o herói recupera 5 *hitpoints* (até ao máximo de 10).
- **Chave (Key)**
  - cada chave tem um identificador associado (ID);
  - algumas portas só abrem com uma chave específica.
- **Porta (Door)**
  - faz o herói passar para outra sala;
  - pode ter dois estados: trancada ou destrancada – quando está trancada será necessária uma chave específica para abrir a porta.
- **Tesouro (Treasure)**
  - Se o herói chegar à posição onde está o tesouro, o jogo termina com sucesso.

Interface gráfico

Neste projeto, a interface com o utilizador é composta pela classe `ImageMatrixGUI` e pelo interface `ImageTile`, que estão incluídos no pacote `pt.iscte.poo.gui`. do projeto `GraphPack` fornecido.

A classe `ImageMatrixGUI` permite abrir uma janela como a representada na Fig. 1. A área de jogo na janela pode ser vista como uma grelha bidimensional de 10x10 posições onde se podem desenhar imagens de 50x50 pixels em cada posição. Além da área de jogo, poderá existir uma barra de estado onde se podem mostrar informações sobre o jogo, nomeadamente as pontuações.

As imagens que são usadas para representar os elementos de jogo encontram-se na pasta "imagens", dentro do projeto. Para se desenhar a imagem de um objeto, este deverá implementar `ImageTile`:

```
public interface ImageTile {
	String getName();		// nome da imagem
	Point2D getPosition();	// posicao de desenho
	int getLayer();			// camada de desenho
}
```

As classes de objetos que implementarem `ImageTile` terão que indicar o nome da imagem a usar (sem extensão), a posição da grelha de jogo onde é para desenhar, e a camada de desenho (*layer*). Esta última determina a ordem pela qual as imagens são desenhadas, nos casos em que há várias sobrepostas na mesma posição (quanto maior o *layer*, "mais em cima" será desenhada a imagem).

Na classe `ImageMatrixGUI` estão disponíveis os seguintes métodos:

- `public void update()` – redesenha os objetos ImageTile associados à janela de desenho – deve ser invocado no final de cada jogada, para atualizar a representação dos elementos de jogo.
- `public void addImages(final List<ImageTile>)` – envia uma lista de objetos `ImageTile` para a janela de desenho. Note que não é necessário voltar a enviar objetos `ImageTile` quando há alterações nos seus atributos – isso apenas contribuiria para tornar o jogo mais lento.
- `public void addImage(final ImageTile)` – envia um objeto `ImageTile` à janela de desenho;
- `public void removeImage(final ImageTile)` – remove um `ImageTile` da área de desenho;
- `public void clearImages()` – remove todos os `ImageTile` da área de desenho;
- `public void setStatusMessage(final String message)` – modifica a mensagem que aparece na barra de estado.

O código fornecido inclui também o enumerado `Direction` e as classes `Point2D` e `Vector2D` (pacote `pt.iscte.poo.utils`). `Direction` representa as direções (UP, DOWN, LEFT, RIGHT) e inclui métodos úteis relacionados com as teclas direcionais do teclado. A classe `Point2D`, representa pontos no espaço 2D e deverá ser utilizada para representar os pontos da grelha de jogo. Inclui métodos para obter os pontos vizinhos e para obter o resultado da soma de um ponto com um vetor. Finalmente, a classe `Vector2D` representa vetores no espaço 2D.

A utilização dos pacotes fornecidos será explicada com maior detalhe nas aulas práticas.

Poderão vir a ser publicadas atualizações aos pacotes fornecidos caso sejam detetados *bugs* ou caso se introduzam funcionalidades adicionais que façam sentido no contexto do jogo. É possível utilizar outras imagens, diferentes das fornecidas, para representar os elementos do jogo, ou para criar elementos de jogo adicionais. Não são esperados problemas desde que as imagens tenham uma dimensão de 50x50 pixels.

Notas:

- O código de exemplo anexo ao pacote gráfico serve apenas para ilustrar algumas das possibilidades de utilização da biblioteca. Não foi desenvolvido para ser usado diretamente no projeto.
- O `GraphPack` não é suposto ser alterado durante o desenvolvimento do projeto.

Estruturação do código

Sugere-se que comece por seguir um diagrama de classes baseado no que se apresenta na Fig. 4.

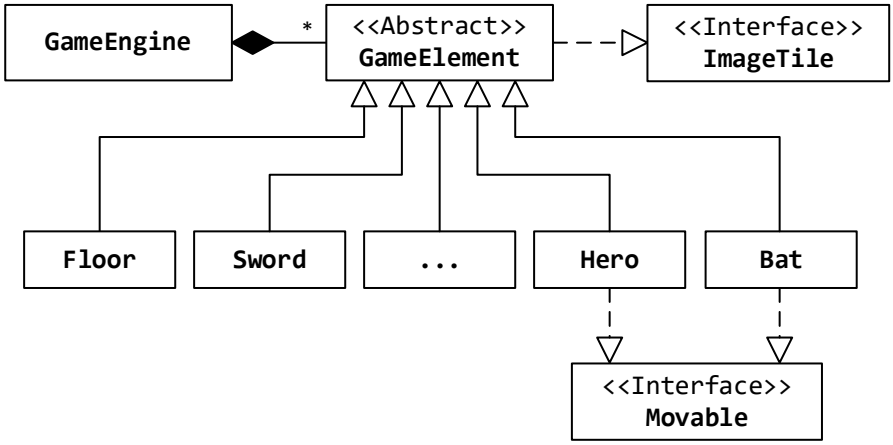

Fig. 4 – Desenho genérico, em UML, de algumas das principais classes e interfaces.

Existe uma classe central (`GameEngine`) que é responsável pela inicialização do jogo, manter as listas de objetos que correspondem aos elementos de jogo e despoletar cada jogada. Sugere-se que a classe central siga o padrão *solitão* (*singleton*) a fim de se facilitar a comunicação com as restantes classes.

Quanto aos elementos de jogo, estes devem ser hierarquizados de forma a tirar o máximo partido da herança, **sendo derivados direta ou indiretamente** de uma classe abstrata `GameElement`.

Devem também ser definidas **características dos elementos de jogo que possam ser modelizadas utilizando interfaces**. Desta forma, os métodos declarados nos interfaces poderão ser invocados de uma forma mais abstrata, sem que seja necessário saber em concreto qual o tipo específico de objeto que o invoca. A título de exemplo, na Fig. 4 sugere-se um Interface `Movable` que poderia ser usado para modelizar as operações feitas por elementos que se movem. Fica a seu cargo definir interfaces adicionais que façam sentido e tirar partido dos mesmos.

Desenvolvimento e entrega do Projeto

Regras gerais

O trabalho deve ser feito por grupos de dois alunos e espera-se que em geral demore 30 a 40 horas a desenvolver. Poderá também ser feito individualmente (nesse caso o tempo de resolução poderá ser um pouco maior). Recomenda-se que os grupos sejam constituídos por estudantes com um nível de conhecimentos semelhante, para que ambos participem de forma equilibrada na execução do projeto e para que possam discutir entre si as opções de implementação.

É encorajada a partilha de ideias sobre o trabalho, **mas é inaceitável a partilha de código**. Caso sejam detetados trechos do mesmo código em trabalhos diferentes, os estudantes envolvidos ficam sujeitos ao que está previsto no código de conduta académica do ISCTE-IUL, arriscando-se a uma reprovação imediata a POO e a que o caso seja reportado à Reitora.

Acompanhamento do projeto

Contacte regularmente o docente das aulas práticas para que este vá revendo o projeto consigo, quer durante as aulas, quer nos horários de dúvidas (com marcação). Desse modo:

- evita opções erradas na fase inicial do projeto (opções essas que em geral conduzem a grandes perdas de tempo e escrita de muito mais código do que o necessário);
- a discussão final do trabalho poderá ser dispensada, dado que tanto os estudantes como o docente foram discutindo as opções tomadas.

Faseamento do projeto

Nesta secção apresenta-se um possível faseamento do projeto, baseado num esquema de *Action Points* semanais:

- Semana 1:
  - Entender a mecânica de comunicação entre o motor de jogo e a GUI;
  - Modelizar uma primeira aproximação à hierarquia de classes do jogo, fazendo o diagrama de classe (com classes, interfaces e suas relações);
  - Ler o ficheiro de configuração e representar os elementos na GUI;
- Semana 2:
  - Implementar o movimento do herói e dos adversários;
  - Criar código para as interações entre o herói e os adversários;
  - Ponderar ajustes à estrutura do programa;
- Semana 3:
  - Apanhar e largar objetos;
  - Gerir a barra de saúde e de objetos;
- Semanas 4 e 5:
  - Implementar a mudança de sala;
  - Implementar o registo de pontuações em ficheiro;
  - Implementar os novos elementos de jogo (a divulgar mais tarde);
  - Rever/refinar opções tomadas, de modo a tornar o programa mais flexível face à introdução de novos elementos de jogo.

A conclusão dos *Action Points* dentro dos tempos indicados não conta para avaliação mas, se houver muitos em atraso, significa que o projeto não está a ser desenvolvido ao ritmo que devia.

Verificação estado de desenvolvimento e *feedback*

Haverá uma verificação ao funcionamento dos trabalhos entre 14 e 25/nov/2022. Esta verificação será feita durante as aulas e destina-se a detetar antecipadamente potenciais problemas que possam colocar em risco a aprovação no projeto. **Os alunos que por algum motivo não possam vir nessas semanas devem enviar os projetos por e-mail ao docente da aula prática em que estão inscritos e estar disponíveis para agendar um horário de dúvidas** para receberem instruções quanto a melhorias a fazer ao seu projeto.

Na semana em que é dado o *feedback* deverá estar a funcionar (pelo menos): a leitura do ficheiro e criação da sala correspondente, o movimento do herói e dos adversários sem passarem por cima de paredes (incluindo paredes que poderão estar no meio do cenário).

Entrega do projeto

O prazo para entrega dos trabalhos é **23:59 de 6ª feira, dia 9/dez de 2022**. A entrega é feita através do *moodle*.

Para a entrega final do trabalho deve proceder da seguinte forma:

1. **Garantir que o nome do seu projeto, tanto no eclipse como no ficheiro zip, contém o nome e número de aluno de cada membro do grupo** – p.e., `Rogue_TomasBrandao741_LuisNunes538` seria um nome válido. Para mudar o nome do projeto no eclipse faça *File/Refactor/Rename*.
2. **Incluir no projeto do eclipse um relatório sucinto em PDF com um diagrama UML das principais classes do trabalho** (use *drag&drop*), usando um nível de detalhe idêntico ao da Fig. 4. Sugere-se que use o plugin para eclipse UMLet[4] ou um editor freeware simples como o Violet UML Editor[5]. É opcional uma breve discussão das opções tomadas e/ou um manual de utilização do jogo.
3. Gerar o *archive file* que contém a exportação do seu projeto – **apenas o projeto onde tem o jogo**, sem o `GraphPack`. Para exportar o projeto, dentro do eclipse deverá fazer *File/Export/Archive File/*, selecionar o projeto que tem o jogo, e exportar para um ficheiro zip.
4. **Entregar via *moodle*** o zip gerado no ponto anterior, na pasta de entrega de trabalhos do docente que acompanhou o trabalho (a disponibilizar brevemente).

Tenha também atenção ao seguinte:

- O código do seu projeto não pode usar caracteres especiais (á, à, é, ç, etc.).
- Se possível, teste a importação do seu projeto num outro computador, antes de o entregar.

Caso não se cumpram os procedimentos em cima, os projetos em causa não se conseguirão importar com facilidade para o eclipse, obrigando os docentes a realizar *setups* manuais que atrasam de forma muito significativa o processo de correção. Como tal, os **projetos mal entregues poderão ser penalizados ou não ser sequer avaliados.**

Todos os projetos entregues serão submetidos a uma ferramenta antiplágio. Nos casos em que houver indícios de fraude académica serão seguidos os procedimentos previstos no código de conduta académica do ISCTE-IUL.

[4] https://www.umlet.com/
[5] https://sourceforge.net/projects/violet/

Avaliação

Objetivo

Realizar o projeto deverá ajudar a consolidar e a explorar os conceitos próprios de POO. **Por isso, para além da componente funcional do trabalho**, **é fundamental demonstrar a utilização correta da matéria e conteúdos lecionados em POO**, em particular:

- modularização e distribuição do código, explorando as relações entre classes e interfaces;
- utilização de herança e sobreposição de métodos com vista a evitar duplicação de código;
- definição, implementação e utilização de interfaces, com vista a flexibilizar o código.

Próximo da data de entrega do trabalho será publicada uma adenda ao enunciado onde se irão acrescentar alguns elementos ao jogo, com o intuito de testar se o seu desenvolvimento foi feito de modo a acomodar facilmente o crescimento do projeto. Poderá nessa altura reavaliar algumas opções e rever algum código com vista a tornar o seu projeto mais flexível, mais fácil de crescer, e com melhor nota!

Critérios de avaliação

O trabalho será classificado com A, B, C ou D, de acordo com os seguintes critérios:

- Grau de cumprimento dos requisitos funcionais;
- Utilização correta de herança;
- Definição e utilização correta de interfaces;
- Modularização, distribuição e legibilidade do código;
- Boas práticas de encapsulamento;
- Originalidade e extras (implementação de padrões, expressões lambda, etc.);
- Relatório de acordo com as instruções dadas na secção “Entrega do projeto”.

Serão imediatamente classificados com D os projetos que:

- Contenham erros de sintaxe;
- Não tenham um mínimo de requisitos funcionais implementados. No mínimo deverá ser possível jogar numa única sala contendo vários adversários;
- Apresentem uma estrutura muito desajustada no que toca a programação orientada para objetos;
- Apresentem indícios de plágio.

Note que, mesmo cumprindo estes mínimos, **também poderão acabar com D os projetos que não demonstrem saber usar/aplicar os conceitos próprios de POO** (herança, interfaces, etc.).

Discussão

A discussão, quando requerida pelo docente, é individual e os estudantes terão que demonstrar ser capazes de realizar um trabalho com qualidade igual ou superior ao que foi entregue. Os membros do grupo são implicitamente responsáveis pelo código do projeto entregue. Assume-se que os membros do grupo participaram de forma equilibrada na sua execução, tendo adquirido os conhecimentos necessários para produzir um trabalho do mesmo tipo individualmente. Caso um dos membros do grupo não consiga demonstrar esta capacidade na discussão poderá ficar com uma nota mais baixa do que a do projeto, reprovar na UC, ou mesmo ser acusado de plágio caso haja suspeita de intenção de entregar em seu nome um trabalho em que não participou.

As discussões serão realizadas de 12 a 16/dez/2022, durante os horários das aulas práticas.