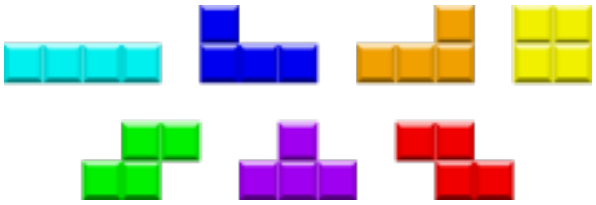
1 O projeto.

A indústria dos jogos é hoje em dia um ramo importante da engenharia de *software* (e da economia em geral). Com orçamentos ao nível dos filmes mais caros de *Hollywood*, os jogos apresentam requisitos extremos que introduzem desafios sem precedentes à construção de *software*, em particular, de sistemas distribuídos. Além do grafismo, que está ao nível da produção cinematográfica, e do desempenho, que pressiona a indústria de *software* e de *hardware* aos limites, nos tempos recentes tem-se assistido ao domínio dos jogos *online* multi-jogador, em que pessoas de todo o mundo formam equipas para se defrontarem. São exemplos desta tendência jogos como o *League of Legends*, o *Counter Strike* ou até alguns mais clássicos que não tinha enveredado inicialmente por este caminho, como o caso do *FIFA*. Estes, não só juntam gráficos de extrema qualidade a inteligência artificial, que faz com que partes do jogo ganhem "vida própria", como também nos permitem formar equipas com jogadores dos quatro cantos do globo que cooperam em tempo real para levar a sua equipa à vitória.

O que vos proponho neste projeto é a construção de um jogo, multi-jogador, distribuído, mas em termos muito mais modestos. O objetivo irá centrar-se em torno dos tópicos da disciplina, em particular, da comunicação dos jogadores com o servidor que aloja o jogo, da partilha de parte do estado global do jogo entre os seus jogadores e, por último, na atualização, em tempo real, da interface de utilizador dos diversos jogadores. Esta última funcionalidade será exigida aos alunos que decidirem apresentar o projeto em época de exame.

O jogo é em si muito simples. Consiste numa versão simplificada do bem conhecido *Tetris* lançado no anos 1980. Muito resumidamente, o jogo consiste em arrumar peças, chamadas *tetriminos*, num tabuleiro vertical. As peças surgem do topo de forma aleatória e caem até tocarem na base do tabuleiro ou em alguma peça já arrumada previamente. À media que as peças caem podem ser rodadas para a direita ou para a esquerda. Ao tocarem na base vão preenchendo linhas, que ao estarem preenchidas são eliminadas. Existem sete tipos de peças como se ilustra a seguir.



A primeira simplificação consiste em considerar somente três peças: o L (direito e esquerdo) e uma outra peça à vossa escolha. Claro, podem sempre optar por utilizar todas as peças. No decorrer do jogo, a única coisa que se pode fazer é rodar a peça para a esquerda e para a direita.

A parte gráfica também pode ser mantida o mais simples possível. A linguagem *Python* tem diferentes pacotes que os podem auxiliar na interface com o utilizador. Como referido, o foco será na comunicação entre os jogadores (clientes) e o jogo (servidor) e na partilha do estado do jogo, neste caso: o tabuleiro em que o jogo se joga; na informação sobre onde se encontram as peças em movimento; e na pontuação individual de cada jogador, ou seja, número de linhas que cada jogador finalizou.

O projeto será dividido em três entregas, em que cada uma delas irá focar-se num propósito específico, a saber:

- a primeira iteração irá focar-se na construção da versão inicial do jogo. Esta versão é mono-jogador e descentralizada. O objetivo é construir a lógica nuclear do jogo e uma interface para a interação do jogador com o jogo. A camada nuclear do jogo (com as regras) será alojada num servidor com o qual o cliente (UI) interage, ou seja, a lógica do jogo encontra-se remota em relação à camada de interação com o utilizador;

- a segunda iteração centra-se em produzir uma versão multi-jogador. Aqui o foco será em construir um servidor que permita múltiplas ligações referentes aos vários jogadores e que forneça a parte relevante do estado do jogo a cada jogador;

- por último, a entrega em exame, adiciona a vertente de atualização da interface de utilizador de acordo com as alterações ocorridas no servidor (produzidas pelos outros utilizadores).

2 Que devemos fazer na iteração 1?

A primeira iteração do jogo `Distributed Tetris` (DT) vai disponibilizar dois casos de uso: **entrada de novo jogador** e **jogar**. Os casos de uso a implementar têm a seguinte descrição breve:

- caso de uso **entrada de novo jogador**—Um jogador indica ao jogo que pretende entrar no jogo. O sistema pede o seu nome (uma *string*). O sistema (liga-se ao servidor e) verifica que não existe outro jogador com o mesmo nome e adiciona-o (e mantém-se ligado).

- caso de uso **jogar**—Este é o caso de uso central do jogo. Nesta primeira iteração, quando o jogador indica que pretende jogar o jogo é lhe apresentada o tabuleiro vazio e uma peça no topo que foi escolhida pelo jogo de forma aleatória. O jogador pode indicar ao jogo que pretende rodar a peça para a esquerda ou para a direita. Passado um

determinado intervalo de tempo (cuja a escolha fica ao vosso critério) a peça desce uma posição. Geralmente nas diferentes versões existentes do jogo este intervalo de tempo vai diminuindo à medida que vão sendo completadas linhas, mas, para simplificar podem utilizar sempre o mesmo intervalo de tempo.

Sempre uma peça atinge a base do tabuleiro ou uma posição ocupada numa linha (acima da base) a peça deixa de descer, assentando nesta posição. Uma linha que ficar completa de uma ponta à outra é eliminada e o (último) jogador que completou a linha ganha um ponto.

O jogo termina assim que as linhas não preenchidas atingirem o topo. Neste momento ganhará o jogador que tiver completado mais linhas.

Nesta primeira fase do projeto devem desenvolver os casos de uso indicados acima implementados na linguagem *Python*, recorrendo ao paradigma orientado a objetos. O sistema deve estar organizado camadas: a *camada de negócio* que implementa as regras do jogo; a camada de *apresentação*, que oferece as funcionalidades do jogo recorrendo a um ambiente gráfico simplificado; e a camada de *middleware* que abstrai a distribuição.

Notem que seguindo as boas práticas de separação de responsabilidades e coesão no desenvolvimento de *software*, a camada de negócio não deve tratar de nenhum aspeto da apresentação nem de distribuição e as camadas de negócio e de *middleware* não devem conter as regras do jogo.

Em concreto, devem:

- elaborar os SSDs e o modelo de domínio de forma a identificar as operações do sistema, os conceitos relevantes e associações entre estes;
- pensar no desenho do sistema e proceder à sua implementação. Para tal devem utilizar um pacote *Python* para cada camada e utilizar classes e módulos de forma conveniente. A organização da vossa aplicação será objeto de estudo nas aulas laboratoriais da disciplina;
- criar dois projetos: um com a parte cliente e outro para a parte servidor;
- a comunicação entre o cliente e o servidor deve ser feita utilizando um dos pacotes *socket* ou *zmq* da linguagem *Python*;
- ambos os projetos devem ser organizados em camadas, contendo uma camada responsável pela comunicação entre o cliente e o servidor. Esta camada terá um papel semelhante a um *middleware* (simplificado) que deve abstrair a interação entre as componentes cliente e servidor utilizando *stubs* e *skeletons*;
- devem estabelecer um protocolo adequado entre o cliente e o servidor de forma a passar a informação dos casos de uso **entrada de novo jogador** e **jogar**;

No que se refere à organização dos projetos, estes devem obedecer à seguinte estrutura.

Cliente:

- Um pacote `dt_ui` que contém o código cliente do jogo que oferece uma interface de texto e a comunicação com o servidor do jogo;
- Este pacote deve conter um ficheiro `__main__.py` que lança cliente.
- Sub-pacote `ui` que contém a parte de interação com o cliente;
- Sub-pacote `stubs` que contém as classes que permitem ao cliente interagir com os objetos respetivos no servidor.

Servidor:

- Um pacote `dt_server` que contém o código com as regras do jogo e a comunicação com o cliente de texto;
- Este pacote deve conter um ficheiro `__main__.py` que lança o servidor.
- Sub-pacote `game` que contém a lógica de jogo;
- Sub-pacote `skeletons` que contém as classes que permitem ao servidor receber e responder a pedidos dos clientes e fazer a interligação destes pedidos e respostas com os objetos na camada `game`.

Podem adicionar mais algum projeto que ajude a reduzir o código eventualmente duplicado entre o cliente e o servidor.

3 Que devemos fazer na iteração 2?

Na segunda iteração pretende-se adaptar o jogo a uma versão cliente-servidor multi-utilizador. Assim, passa a ser possível que vários clientes se liguem ao servidor do jogo *DT* e jogar em simultâneo. A versão do cliente será essencialmente a mesma da iteração anterior, bem como o protocolo de comunicação entre o cliente e o servidor que estabeleceram. A ênfase desta fase será na modificação do comportamento do servidor para acomodar a interação feita por vários jogadores e na consequente atualização do estado do jogo que será partilhado por todos. Para tal, devem:

- Alterar o projeto servidor de modo a atender vários clientes. Para tal devem utilizar *threads* ou *processos*, aquele que melhor servir as vossas necessidades;
- Deverá ser lançada uma nova *thread* ou *processo* sempre que um novo jogador entrar no jogo de forma a atender de forma exclusiva os seus pedidos;

- A comunicação com cada cliente deve ocorrer de forma separada dos outros recorrendo à camada de comunicação da iteração 1;
- Tenham em atenção que o **estado do jogo** passa a ser partilhado entre os diversos clientes, o que significa que podem ocorrer *condições de corrida*, pois poderão existir alguns clientes a ler o estado do jogo, por exemplo, para mostrar a visão do tabuleiro no ecrã, enquanto outros estão a alterar este estado em simultâneo, por exemplo, deslocando peças no tabuleiro;
- Para evitar condições de corrida devem utilizar um mecanismo de exclusão mútua, por exemplo, *locks*;
- A parte cliente do jogo deve ser alterada para permitir um jogador *entrar no jogo* (o que antes era a *criação de um jogador*) e *sair do jogo*. Agora, quando o primeiro jogador entrar no jogo é criado um novo jogo no servidor. Este jogo manter-se-á ativo até acontecer uma das seguintes situações: o tabuleiro fica completamente preenchido com linhas não completas e, neste caso, ganha o jogador com mais linhas eliminadas ou é declarado um empate, se ambos tiverem eliminado o mesmo número de linhas; todos os jogadores exceto um abandonam o jogo e neste caso este jogador é declarado o vencedor e o jogo termina. O restante funcionamento do jogo é essencialmente o descrito nas fases anteriores do projeto e o código já escrito deve ser reutilizado o mais possível. A única alteração é a seguinte: sempre que surge uma nova peça no topo do ecrã, o primeiro dos jogadores (que não estiver a controlar uma peça) a fazer uma ação de rotação ficará a controlar a peça até ao fim do tabuleiro.

  Sempre que se faz uma atualização do tabuleiro este deve informar a pontuação do jogador que está a jogar naquele cliente e a pontuação do jogador mais alto, bem como as linhas que estão semipreenchidas. Não deve mostrar as outras peças para não ficar muitos elementos no ecrã e confundir o jogador.
- A *thread* ou *processo* que atende um cliente deve terminar quando o jogador sair do jogo ou quando o jogo terminar.

No que se refere à organização dos projetos cliente e servidor, estes devem obedecer à estrutura definida na fase 2 do projeto.

4 O que é comum às várias iterações?

O código deve ser desenvolvido em equipa e utilizar-se o sistema de controlo de versões *git* via *bitbucket.org*. Cada grupo deve ter o seu repositório no servidor *git* com o nome no formato *sd2021xx*, em que *xx* é o número

do grupo. O repositório pode pertencer a qualquer um dos elementos do grupo, deve ser privado, e só os elementos do grupo e o utilizador *francisco.cc.martins@uac.pt* devem ter acesso à informação do repositório. O utilizador *francisco.cc.martins@uac.pt* só necessita de permissões de leitura dos dados.

Devem privilegiar as boas práticas de programação. Para tal, entre outras coisas, devem:

- Evitar repetição de código, tirando partido dos mecanismos oferecidos pela programação orientada a objetos, em particular, herança, composição e ligação dinâmica a métodos;
- Não utilizar literais (constantes) ao longo do código. Para tal devem definir constantes ou enumerados;
- As constantes de cada pacote devem estar definidas nos `__init__.py` respetivos de cada pacote;
- Devem tirar partido do *lint* que é fornecido pelo *PyCharm* e entregar os vossos módulos sem erros nem avisos identificados por este *software*. São permitidos *typos*;
- As classes e métodos devem estar comentadas corretamente. Ou seja, devem conter informação sobre cada atributo, sobre cada parâmetro e resultado dos métodos e ter uma breve descrição sobre o propósito de cada classe, atributo e método;
- Deve tirar partido das facilidades fornecidas pelo *Python 3*, em particular, a utilização de propriedades sempre que adequado, a anotação de métodos com os tipos dos parâmetros e o tipo do valor devolvido.

5 Como e quando entregamos?

Simplesmente vou considerar o **último *commit* no vosso repositório de grupo** até ao fim do dia da entrega. A primeira entrega até ao fim do dia **25 de maio de 2021**; A segunda entrega até ao fim do dia **14 de junho de 2021**.

O repositório deve conter:

1. o código fonte do vosso projeto;
2. um documento *PDF* com as decisões (mais importantes) que tiveram de tomar em termos de desenho da aplicação.

Podem esclarecer qualquer questão sobre o projeto no final de cada aula (sempre que possível) ou durante o horário de atendimento semanal. Notem que o trabalho é para ser elaborado em grupo com dois ou três elementos e que o grupo deve ser constituído no *moodle*.