Enunciado do Trabalho Final de POO 2017/18

Simulador da Quinta, V1.0

Introdução

O trabalho final consiste em criar um simulador de uma plantação. As principais características deste simulador são: tem dois tipos de culturas (tomate e couve), um agricultor e um tipo de animal (ovelhas). O objetivo do agricultor é amealhar pontos plantando, cuidando e colhendo das culturas e simultaneamente não deixar as ovelhas ficar famintas. Colher plantas dá pontos (tomate 3 pontos, couve 2 pontos), e são também atribuídos pontos em cada ciclo de jogo (ver abaixo) em que as ovelhas estejam bem alimentadas (1 ponto por ciclo).
O agricultor é controlado pelas teclas de movimento (setas) e o pressionar do espaço indica que o próximo "movimento" é uma acção sobre os objetos que estão nessa direcção. O agricultor não "colide" com nenhum dos objetos do jogo.

Fig. 1 Exemplo. Na imagem podemos ver o agricultor, culturas de dois tipos (tomate dos lados e couve ao centro) bem como as ovelhas.
(Imagens processadas com base em originais publicados em: https://opengameart).

Funcionamento do simulador

O simulador funciona do seguinte modo:

- Começa com uma configuração fixa (por exemplo, terreno de 7 x 5, sem plantas e com o agricultor no canto superior esquerdo)
- O pressionar de uma tecla (qualquer tecla) conta como um ciclo de jogo para todos os objetos da quinta que dependem do "tempo de simulação"
- Tem dois tipos de cultura: tomate e couve
- Pode ter ovelhas
- Tem um agricultor, que é movido pelo utilizador usando as teclas de direcção, e que faz uma acção sempre que a tecla de direcção é precedida da tecla de espaço, por exemplo, para andar para a direita usa-se a tecla da seta para a direita, mas para lavrar o campo à direita deve pressionar espaço e depois a tecla da seta para a direita
- Os campos podem estar: por lavrar ou lavrados
- Em campos não lavrados não pode haver culturas
- Em campos lavrados pode haver: nada; vegetais pequenos (em crescimento), vegetais prontos a colher (plantas grandes), ou vegetais prontos a limpar (plantas estragadas)
- Cada vez que o agricultor interage com um campo e com os objetos que estão nele todos os objetos podem sofrer alterações
- A interações são as seguintes:
    - *Agricultor -> Campo por lavrar*: o campo passa a estar lavrado
    - *Agricultor -> Campo lavrado*: o campo passa a estar plantado com um vegetal aleatório: tomate ou couve
    - *Agricultor -> Campo com plantas pequenas*: o agricultor cuida das plantas, no caso das couves isso acelera o seu crescimento, no caso do tomate só poderá ficar maduro se o agricultor já tiver cuidado da planta pelo menos uma vez entre o momento em que plantou e o momento de ficar maduro
    - *Agricultor -> Campo com plantas maduras*: depende do vegetal (ver abaixo)
    - *Agricultor -> Campo com plantas estragadas*: limpar as plantas
    - *Agricultor -> Ovelha*: alimenta a ovelha
    - *Ovelhas -> Vegetal*: se a ovelha não é alimentada há mais de 10 ciclos, come o vegetal (a menos que este esteja estragado)
- O tomate tem as seguinte propriedades: amadurece ao fim de 15 ciclos, apodrece ao fim de 25 ciclos, pára de amadurecer se não for cuidado por mais de 10 ciclos (volta a amadurecer a partir do momento em que for cuidado)
- A couve tem as seguinte propriedades: amadurece ao fim de 10 ciclos, apodrece ao fim de 30 ciclos, quando é cuidada amadurece mais rápido
- A ovelha tem as seguintes propriedades: se não for alimentada há mais de 10 ciclos começa a movimentar-se aleatoriamente; e, se tentar mover-se para uma posição que tenha plantas, come-as (e não executa o movimento); passa ao estado faminto se não comer nada há mais de 20 ciclos
- Os vegetais mudam de imagem consoante o seu estado (jovem, maduro, estragado) e as ovelhas mudam de imagem (para a imagem de famintas) se estiverem mais de 20 ciclos sem ser alimentadas.
- Deve ser possível ler (tecla ´l´) e escrever (tecla ´s´) a informação para um ficheiro de modo a repôr / gravar o estado de uma simulação

Objectivos

Usando o interface gráfico fornecido com o enunciado e **respeitando o desenho parcial das classes do sistema apresentado na figura 2** deve criar um conjunto de classes que permita realizar as simulações e observar os resultados.

Requisitos

Atenção: o código de exemplo serve para ilustrar algumas das possibilidades de utilização da biblioteca. Não é suposto o seu projeto ser uma mera continuação do código de exemplo.

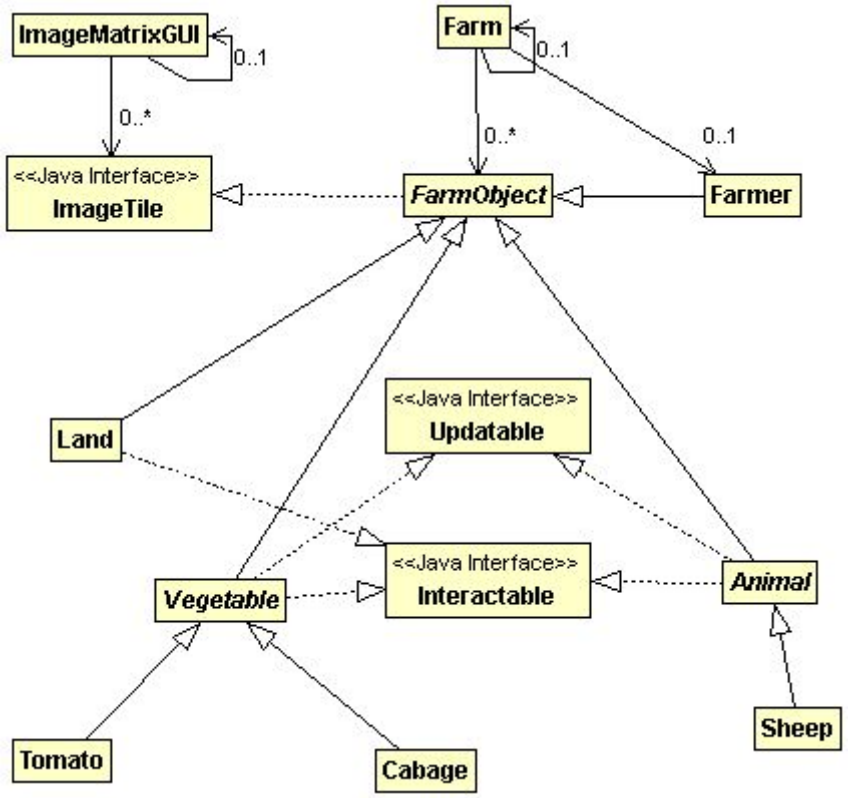

Fig. 2 Desenho em UML genérico de algumas das principais classes e interfaces

O simulador pode enviar imagens para a janela usando implementações de `ImageTile` (como algumas classes do exemplo fornecido). De cada vez que há uma alteração (por exemplo, mudança das posições das imagens) deve ser invocado o método update() do interface gráfico. **Não é necessário re-enviar as imagens para o interface após cada alteração da posição, isso iria apenas sobrecarregar o interface gráfico e tornar o jogo mais lento à medida que a lista de imagens vai crescendo.**

A interação entre os vários objectos do simulador deve ser tão autónoma quanto o possível, distribuindo-se o código pelas várias classes correspondentes e minimizando-se a quantidade de código presente na classe principal do simulador.

As classes fornecidas no projeto GraphPack **não devem ser alteradas** e são fundamentais para a

resolução (para garantir isso os docentes poderão substituir esse package pela versão "oficial" na apresentação / correcção do trabalho).

O objetivo geral é que, ao realizar o trabalho, consolide e explore os conceitos próprios de POO. Por isso, para além da componente funcional do trabalho, é fundamental que demonstre a utilização correta da matéria dada em POO. Assim, é **obrigatória** a utilização de:

- Bom **encapsulamento** (boa utilização do **private e dos inspectores**)
- Boa **modularização** ao nível das funções e das classes
- **herança de propriedades e sobreposição de métodos** na interação entre objetos do jogo
- implementação e uso de **interface(s)**
- utilização de listas (use a interface List em vez das classes específicas na declaração da referência)
- Leitura / escrita de ficheiros

Para demonstrar conhecimentos sobre todas as partes da matéria, é importante que abordem também os seguintes tópicos:

- Excepções - tratamento de problemas que possam acontecer na leitura dos ficheiros, ou na passagem de informação entre as suas classes (argumentos nulos ou outros problemas). É necessário demonstrar que sabe usar excepções (lançar e tratar adequadamente), não é obrigatório usá-las em todos os casos em que possam surgir problemas, mas deve ter pelo menos um exemplo de uma boa utilização de exceções para detecção de uma inicialização inconsistente e fazer algum tipo de recuperação, se possível.
- JUnit - realizar um teste de funcionamento para uma das classes derivadas de `Vegetable`, simulando a interação com esta e as mudanças de estado.
- Javadoc - documentar a mesma classe para a qual fez a JUnit.

É ainda aconselhada a experimentação com estruturas de dados auxiliares diferentes das listas (mapas, conjuntos ou filas) caso encontre situações em que estas sejam adequadas.

O enunciado poderá ser alterado com vista a acrescentar mais conteúdos quer a nível de classes, quer a nível de interações entre os mesmos - tenha isso em mente ao planear e programar o seu trabalho. Uma das características importantes da Programação Orientada para Objetos é a sua flexibilidade. Para testar se o seu desenvolvimento foi feito de modo flexível deve perceber que alterações teria que fazer para incorporar novas classes e funcionalidades que façam sentido no contexto do trabalho. Assim, após a primeira entrega poderá ser publicada uma nova versão do enunciado com alguns requisitos adicionais.

Ficheiro de Configuração / Gravação do estado da Simulação
(exemplo de formato aconselhado)

Neste trabalho o formato de escrita e leitura do ficheiro é livre, mas tendo em consideração a possibilidade de haver mais do que um objeto em cada ponto do terreno aconselha-se o seguinte formato (apenas parcialmente descrito):

```
5 7 // tamanho da quinta
0   // pontos acumulados até ao momento
Land (0, 0)        // indicação do tipo de objeto e da sua posição
Land (0, 1) plowed // tipo de objeto, sua posição e uma característica (lavrado)
...
Land (4, 5) plowed
Land (4, 6)
Tomato (4, 1) ... // tipo de objeto, sua posição e outras características
Tomato (4, 2) ...
...
Tomato (0, 5) ...
Cabage (2, 1) ...
...
Cabage (2, 5) ...
Sheep (3, 3) ...
```

Fig. 3 Exemplo **parcial** de um ficheiro de estado simples (o mesmo usado para gerar a Figura 1).

O trabalho final deve poder ser gravado e carregado de novo e deve ter um ficheiro de configuração que resulte numa imagem igual à da figura 1, com zero pontos e todos os temporizadores dos vegetais e das ovelhas com o valor inicial.

Interface Gráfico

O interface gráfico fornecido (classes `ImageMatrixGUI` e interface `ImageTile`, no package `pt.iul.ista.poo.gui` permite abrir uma janela como a apresentada na figura 1), tem dentro uma área que é vista como uma grelha bidimensional de tamanho variável contendo cada célula da grelha uma imagem de 50x50 píxeis. Além dessa zona, o utilizador desta biblioteca dispõe de uma barra de mensagens, acima, onde se podem mostrar mensagens simples. As imagens que são usadas fazem parte de uma "biblioteca" que se encontra na pasta "imagens" dentro do seu projeto. A *interface* `ImageTile` é usada para indicar qual a imagem a desenhar e qual a posição em que esta será desenhada dentro da grelha.

O nome da imagem, por omissão, corresponde ao nome da classe nas implementações de `ImageTile` que são classes derivadas de `FarmObject` embora o nome da imagem seja composto apenas por minúsculas. Por exemplo, se quer uma imagem para a classe `Farmer` terá que se chamar `farmer.png`, ou `farmer.jpg`, ou `farmer.gif`. Uma `ImageTile` precisa também de definir a posição de desenho na grelha que constitui a janela visível e a sua camada (`layer`) que serve para determinar quais as imagens que ficam por baixo ou por cima.

Por exemplo, para dar uma imagem à classe `Sheep`, deve ter na pasta de imagens uma imagem com o nome `sheep.png`. Na classe `Sheep` precisa de ter um modo de indicar qual a posição de cada objeto desse tipo (usando um `Point2D`) e responder ao método `getLayer` com um número inteiro que representa a camada de desenho. Quanto maior for este número mais "acima" será desenhada a imagem (e.g., uma imagem na camada 2 irá ser desenhada por cima de uma imagem na camada 1 que esteja na mesma posição).

A classe `ImageMatrixGUI` contém os seguintes métodos que poderão ser utilizados no trabalho:

```
public void addImages(final List<ImageTile> ...)
public void addImage(final ImageTile ...)
public void removeImage(final ImageTile ...)
public void removeImages(final List<ImageTile> ...)
Public void clearImages()
public void newStatusMessage(final String message)
```

Caso prefira, pode usar outras imagens para representar os elementos do jogo mas, como esse não é um dos objetivos do trabalho, não é contabilizado, nem serão prioritárias as dúvidas ou erros relacionados com a utilização de outras imagens. À partida não são esperados problemas desde que todas as imagens usadas tenham uma dimensão de 50x50 píxeis. Caso use imagens que não sejam da sua autoria, no trabalho devem ser dados os devidos créditos aos autores.
Este *interface* será distribuído e explicado numa aula prática. Poderão vir a ser publicadas atualizações ao pacote de classes fornecido caso sejam detectadas falhas de funcionamento. Mantenha-se atento aos avisos.

Execução do Trabalho

O trabalho deve ser feito por grupos de dois alunos e espera-se que demore cerca de 40h a executar. Poderá também ser feito individualmente - nesse caso estima-se que o tempo de resolução seja um pouco maior, mas não substancialmente. Recomenda-se que seja feito por grupos de dois alunos com um nível de conhecimentos semelhante porque a discussão das opções de implementação é em geral muito benéfica para a aprendizagem.

É encorajada a discussão entre colegas sobre o trabalho, mas é estritamente proibida a troca de código. Atenção que a partilha de código em trabalhos diferentes será seriamente penalizada. Serão usados os meios habituais de verificação de plágio e os trabalhos plagiados serão comunicados ao Conselho Pedagógico para procedimento disciplinar. Serão ainda feitas discussões do trabalho, das quais poderão ser dispensados os trabalhos que foram acompanhados pelo docente que está a avaliar. Obviamente que o código entregue deve ser integralmente da autoria dos membros do grupo.

Peça regularmente ao docente para que reveja o trabalho consigo, quer durante as aulas, quer em horário de dúvidas (com marcação). Desse modo:

1. pode evitar algumas más opções iniciais que normalmente conduzem a um grande aumento da quantidade de trabalho;
2. a discussão final do trabalho é menos crítica e poderá ser dispensada, dado que tanto o aluno como o docente foram discutindo o trabalho e as opções tomadas.

Entrega Intercalar

A entrega intercalar será até às **23:59 do dia 8 de Abril de 2018**.

Nesta entrega, deve estar a funcionar todo o "ciclo de vida" da couve e do tomate, i.e. o lavrador deverá conseguir realizar as seguintes interações com um pedaço de terra: 1) lavrar, 2) plantar couve ou tomate aleatoriamente, e 3) cuidar do vegetal (com efeitos diferentes em cada um).

Para entregar o trabalho deve proceder da seguinte forma:

1. **Garantir que o nome do seu projeto no eclipse contém o nome e número de aluno de cada membro do grupo**
2. Incluir na pasta do projeto, no eclipse todos os ficheiros que forem necessários para a entrega (inclusive imagens e, no caso da entrega final, o relatório - pode fazer *drag and drop* do relatório diretamente para a pasta do projeto no eclipse)
3. Usar *File/Export/Archive File/*, selecionar os projetos onde tem o trabalho final e o pacote gráfico e **entregar no *e-learning*** na página de Entrega de Trabalhos a disponibilizar brevemente. Caso não esteja associado à UC de POO no *e-learning* contacte o coordenador da UC previamente

Tenha em atenção que:

- Após exportar o seu projeto, importá-lo e executá-lo noutro computador não pode dar origem a problemas imprevistos. Teste por isso a exportação/importação antes de entregar.
- Não use caracteres acentuados no código do projeto.

Trabalhos que não sejam corretamente entregues poderão nem sequer ser vistos e caso sejam avaliados serão penalizados com uma redução no nível da nota.

Entrega Final

A entrega final será até às **23:59 do dia 27 de Maio de 2018**.

Nesta entrega, além do código deverá também ter um relatório sucinto **em PDF dentro do projeto** em que é obrigatório o desenho UML das principais classes do trabalho (no mesmo nível de detalhe apresentado na Fig. 3), uma declaração dos membros do grupo que atesta que o código entregue neste trabalho é integralmente da sua autoria. É opcional ainda uma breve discussão das opções tomadas e/ou um manual de utilização do simulador.

O procedimento de entrega da versão final do trabalho é o mesmo que foi feito para a Entrega Intercalar (descrita no ponto anterior).

Avaliação

O trabalho será classificado com A, B, C ou D. Os critérios de avaliação são os que se apresentam em seguida.

Serão excluídos e classificados com D (sem que sejam aceites para discussão) os trabalhos que:

- apresentem erros de sintaxe;
- tenham um funcionamento muito limitado, sem que componentes essenciais do projeto estejam implementados. Por exemplo, se o ciclo de crescimento dos vegetais não estiver bem implementado é considerado que o trabalho não cumpre os requisitos mínimos;
- não usem herança;
- não usem coleções (e.g., listas) para agregar os vários objectos;
- não demonstrem que sabem usar corretamente classes e objetos para modularizar o código (e.g., concentrem o código numa só classe);
- apresentem por norma métodos extensos mal modularizados e/ou código repetitivo.

A exclusão do trabalho final implica reprovação e obriga à repetição da UC no semestre seguinte.

As notas de A a C serão atribuídas com base na qualidade do trabalho relativamente aos seguintes critérios:

- Classes bem modularizadas e bem encapsuladas (métodos curtos que realizam ações bem definidas, boa escolha de atributos, boa utilização do private e de inspectores);
- Boa distribuição do código pelas várias classes de objetos;
- Utilização correta de classes e interfaces, herança, sobreposição de métodos e coleções;
- Boa escalabilidade do código (i.e., flexibilidade e facilidade em acrescentar novos elementos ao simulador);
- Cumprimento dos requisitos funcionais;
- Cumprimento de requisitos adicionais sem impacto a nível funcional (teste JUnit, documentação e utilização de excepções para tratamento de possíveis erros);
- Tenham um relatório de acordo com as indicações na secção Entrega Final;

A discussão é individual e todos os alunos terão que demonstrar ser capazes de fazer um trabalho com o nível de qualidade igual ao que assinaram. Ao entregar o trabalho, os alunos implicitamente afirmam que são os únicos responsáveis pelo código entregue e que todos os membros do grupo participaram de forma equilibrada na sua execução, tendo todos adquirido os conhecimentos necessários para produzir um trabalho do mesmo tipo individualmente. Não demonstrar essa capacidade pode implicar a não aceitação do trabalho para um dos membros do grupo ou mesmo a acusação de plágio e processo disciplinar, caso haja suspeita de intenção de entregar como seu um trabalho em que não participaram.

Os trabalhos serão também submetidos, como é já habitual, a um detetor automático de plágio.