Programação Avançada

2018/19

Trabalho Prático

Pretende-se implementar uma aplicação que permita ao utilizador jogar o jogo individual *Destination Earth*. As regras originais bem como material gráfico diverso pronto a imprimir (tabuleiro, cartas, etc.) encontram-se disponíveis nos ficheiros anexos e em https://boardgamegeek.com/boardgame/255468/destination-earth .

Regras gerais

O trabalho deve ser realizado por grupos de dois alunos.

A elaboração do trabalho está dividida em duas fases separadas.

As datas de entrega do trabalho nas duas fases são as seguintes:

1. Primeira fase: **28 de Abril**;
2. Segunda fase: **2 de Junho**.

As entregas correspondentes às duas fases do trabalho devem ser feitas através do *moodle* num ficheiro compactado. O nome deste ficheiro deve incluir o primeiro nome, o último nome e o número de estudante de cada um dos elementos do grupo, bem como a indicação da turma prática a que pertencem. O ficheiro deve conter, pelo menos:

- O ficheiro jar executável;
- O projecto NetBeans, incluindo todo o código fonte;
- Eventuais ficheiros de dados e recursos auxiliares necessários à execução do programa;
- O relatório em formato pdf.

O relatório deve incluir em ambas as fases:

1 Uma descrição sintética acerca das opções e decisões tomadas na implementação (máximo uma página).

2 O diagrama explicado da máquina de estados que controla o jogo e os diagramas de outros padrões de desenho que tenham eventualmente sido aplicados no trabalho.

3 A descrição das classes utilizadas no programa (o que representam e os objectivos) (máximo de 60 palavras / 4 linhas por classe).

4 A descrição do relacionamento entre as classes (podem ser usados diagramas).

5 Para cada funcionalidade ou regra do enunciado, a indicação de cumprido totalmente / cumprido parcialmente (indicar o que foi cumprido), não cumprido (indicar a razão). O uso de uma tabela pode simplificar a elaboração desta parte.

Ambas as fases estão sujeitas a defesas que incluem a apresentação, explicação e discussão do trabalho. Estas podem incluir a realização de alterações ao trabalho entregue, feitas individualmente por cada um dos elementos do grupo.

A cada uma das fases do trabalho corresponde a cotação **4 valores**.

Funcionalidade e requisitos

As funcionalidades requeridas nas duas fases do trabalho prático são as seguintes:

1. **Primeira Fase**: Implementação do jogo *Destination Earth* com interface de utilizador em consola / modo texto;
2. **Segunda Fase**: Implementação do jogo *Destination Earth* com interface de utilizador em modo gráfico e em consola / modo texto.

Em **ambas as fases**, deve ser possível gravar em ficheiro o jogo que se encontra a decorrer, bem como carregar e continuar um jogo previamente guardado.

A implementação do trabalho deve **obrigatoriamente** obedecer aos requisitos seguintes:

1. Deve ser seguida a arquitectura indicada na secção seguinte deste enunciado.
2. Deve ser utilizada, de forma adequada, uma máquina de estados para realizar a lógica do jogo (a Figura 1 ilustra, de forma incompleta e a título de exemplo, uma possível abordagem com identificação de eventos e acções associados às transições entre estados).
3. Ao terminar um jogo, a máquina de estados deverá permitir jogar de novo ou terminar a execução da aplicação.
4. Relativamente ao lançamento de um dado previsto na acção *Attack* da *Crew Phase*, deve ser dada a possibilidade ao jogador de usar um número aleatório entre 1 e 6 gerado pelo computador ou, então, de especificar explicitamente o número que deseja que seja usado (para efeitos de *debugging* e avaliação). Não se trata de uma escolha feita uma única vez no início do jogo – pelo contrário, o utilizador tem esta escolha constantemente ao longo do jogo.
5. Na *Alien Phase*, as 4 acções devem ser automaticamente desencadeadas sem qualquer intervenção do jogador, incluindo a geração de um valor aleatório entre 1 e 6 destinado a reflectir o Lançamento de um dado (acção IV).
6. A aplicação deve apresentar a informação necessária ao acompanhamento e verificação do bom funcionamento do jogo. Isto significa que o jogo deverá gerar e disponibilizar informação detalhada sobre o estado interno e o resultado das várias acções. Sem exclusão de outros, isto inclui: (1) o jogador deve poder saber qual foi o resultado dos lançamentos dos dados que foram feitos automaticamente para perceber as suas consequências; (2) devem ser devidamente apresentados todos os dados relevantes que caracterizam a informação dos elementos *Ship Game Board* e *Player Board*, bem como os dados dos *crew members* seleccionados; (3) deve ser apresentado o turno e a fase em actuais; e (4) devem ser identificadas as situações de vitória e derrota (e a sua causa).
7. Na segunda fase do trabalho, deve ser utilizado o padrão de separação entre Modelo e Vista baseado na classe *java.util.Observable* e na interface *java.util.Observer*, conforme estudado nas aulas. Para além das vistas gráficas a aplicação deve integrar uma vista *consola / modo texto* de acordo com este padrão.

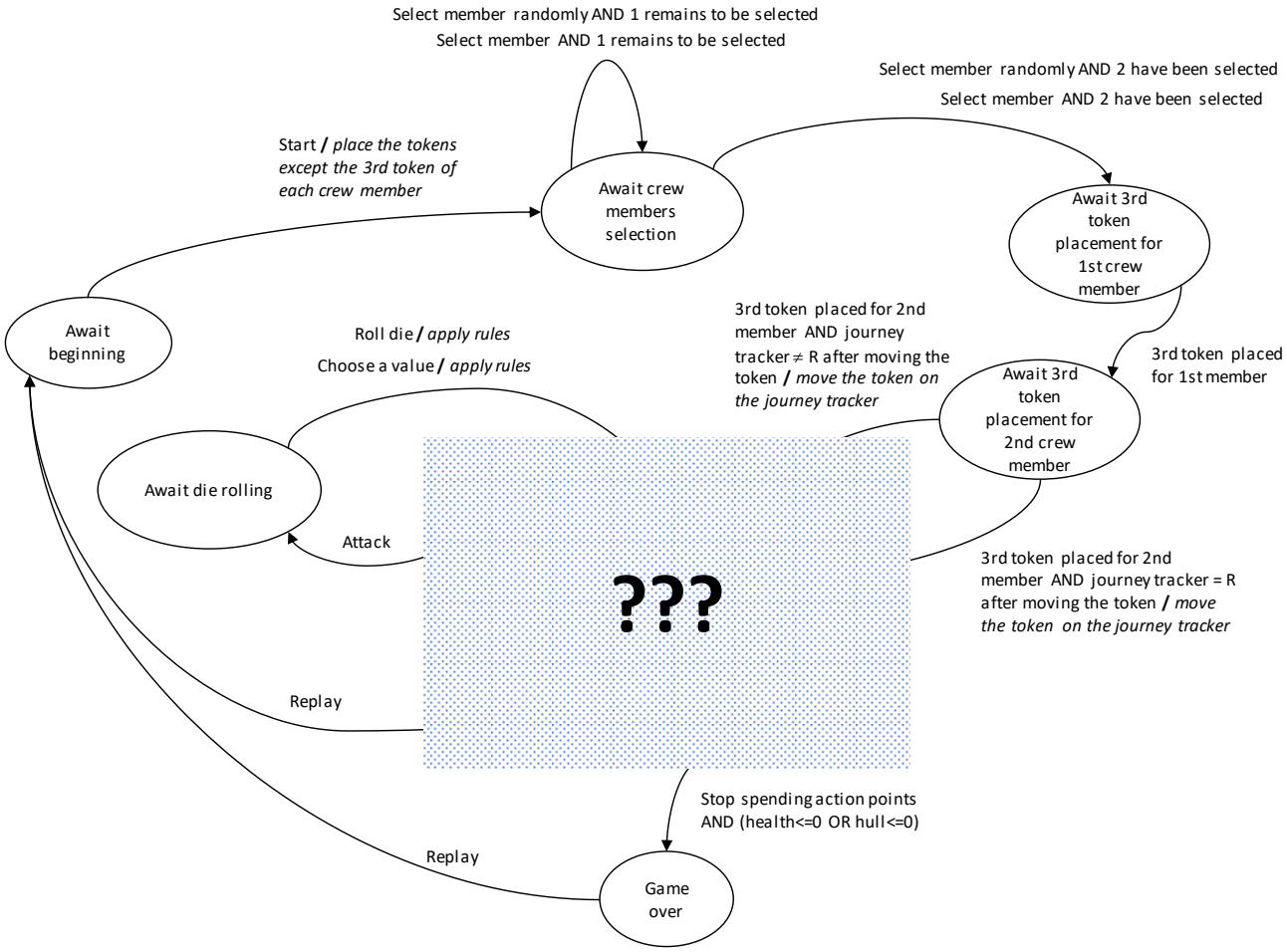

Figura 1 – Versão incompleta de uma possível máquina de estados

Arquitectura (fase 1)

A arquitectura da aplicação, na fase 1, deve estar condicionada de acordo com o apresentado na Figura 2.

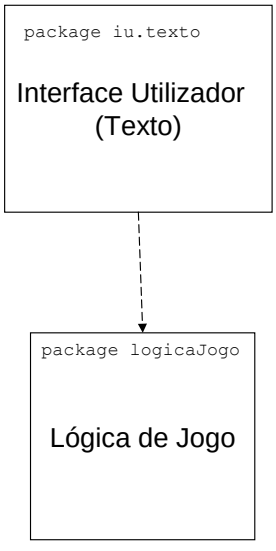

Figura 2 – Arquitectura (fase 1)

A aplicação deve estar organizada em (pelo menos) dois *packages*, conforme apresentado na Figura 2. Pode existir um número qualquer de classes em cada *package*. Conforme indicado pela seta na Figura 2, o *package iu.texto* (e as classes que este inclui) é dependente de classes definidas no *package logicaJogo*. No entanto, o inverso não é verdade. Isto significa que, em qualquer classe definida no *package iu.Texto,* podemos encontrar:

```
package iu.texto;
import logicaJogo.*;
...
```

No entanto, NÃO será necessário nem permitido fazer um *import* no sentido inverso:

```
package logicaJogo;
import iu.texto.*;  //ERRADO: NÃO CUMPRE A ESPECIFICAÇÃO.
...
```

Adicionalmente, cada um dos packages deve cumprir as seguintes restrições:

- *Package iu.texto* – Não deve ser encontrada neste *package* qualquer lógica relacionada com o jogo e as suas regras. Por exemplo, aqui é possível imprimir o estado do jogo ou perguntar ao utilizador qual a sua jogada. No entanto, não existirá aqui nenhum código que determina aquilo que acontece em resposta a uma jogada;
- *Package logicaJogo* – A lógica do jogo deve estar aqui. Não pode, no entanto, existir aqui código que realize operações de leitura do teclado ou de escrita no ecran (nem em código chamado por código incluído neste *package* – recordar o sentido da seta na figura 2).

Porque é que é necessário seguir esta arquitectura? Aparentemente, estas restrições só tornam as coisas mais difíceis… Na fase 2 vamos ver que estas restrições fazem sentido e ajudam muito na obtenção do programa.

Arquitectura (fase 2)

Conforme apresentado na Figura 3, na fase 2 é criado um *package* adicional que implementa uma interface gráfica. O que é que aconteceria se as indicações anteriores não tivessem sido cumpridas?

- Se existir lógica do jogo no *package iu.texto*, o *package logicaJogo* não inclui tudo o que é necessário. É, então, preciso refazê-la (a solução menos má) ou reimplementar a lógica que falta novamente no outro *package* (repetindo código, eventualmente inconsistente com o já existente). É preferível construir de raiz o *package logicaJogo* para que esta inclua tudo o que é necessário…
- Se o *package logicaJogo* incluir código que pede ao utilizador para introduzir informação (através do teclado) e/ou imprimir informação no écran (consola?), então este código não pode ser usado por um componente que pretenda implementar uma interface gráfica (ou que que automatize as escolhas do jogo através de um módulo que implemente um jogador virtual, nem …. etc./muitos outros exemplos possíveis).
- Resumindo, as restrições arquiteturais aqui descritas, que constituem um exemplo de boas práticas de programação orientada a objetos, destinam-se a poupar trabalho e

maximizar as possibilidades de reaproveitar, na segunda fase, aquilo que é feito na primeira.

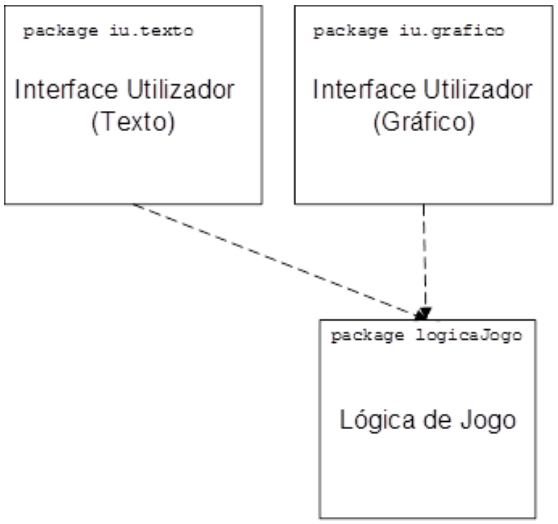

Figura 3 - Arquitectura (fase 2)