Programação Avançada

2019/20

Trabalho Prático

Pretende-se implementar uma aplicação que permita ao utilizador jogar o jogo individual *Planet Bound*. As regras originais bem como material gráfico diverso, pronto a imprimir e a usar para a elaboração da interface de utilizador em modo gráfico (GUI) na 2ª fase, encontram-se disponíveis nos ficheiros anexos e em:

https://boardgamegeek.com/boardgame/298332/planet-bound

Regras gerais

O trabalho deve ser realizado de forma **individual**.

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 (modelo do jogo utilizando os padrões apresentados nas aulas): **10 de maio**;
2. Segunda fase (jogo funcional com GUI): **12 de junho**.

As entregas correspondentes às duas fases do trabalho devem ser feitas através do *inforestudante* num ficheiro compactado em formato ZIP. O nome deste ficheiro deve incluir o primeiro nome, o último nome e o (novo) número de estudante do aluno, bem como a indicação da turma prática a que pertence (ex: P3-201906104-Antonio-Ferreira.zip).

Na primeira fase, o ficheiro deve conter, pelo menos:

- O projeto NetBeans, incluindo todo o código fonte produzido;
- O relatório em formato pdf.

Na segunda fase, o ficheiro deve conter, pelo menos:

- O ficheiro *jar* executável;

- O projeto NetBeans, incluindo todo o código fonte produzido;
- 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 da máquina de estados que controla o jogo, devidamente explicado, e os diagramas de outros padrões de programação que tenham eventualmente sido aplicados no trabalho;
3. A descrição das classes utilizadas no programa (o que representam e os objetivos);
4. A descrição do relacionamento entre as classes (podem ser usados diagramas);
5. Em ambas as fases, para cada funcionalidade ou regra do jogo, a indicação de cumprido/implementado totalmente ou parcialmente (especificar o que foi efetivamente cumprido neste caso) ou não cumprido/implementado (especificar 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 apresentado. Estas podem incluir a realização de alterações ao que foi entregue.

À primeira e segunda fases do trabalho correspondem, respetivamente, as cotações de **5** e **9** valores**.**

Objetivos e requisitos

Os objetivos das duas fases do trabalho prático são os seguintes:

1. **Primeira Fase**: Definição e implementação do modelo de dados e da máquina de estados que representa o jogo de acordo com os padrões dados nas aulas como, por exemplo, estados polimórficos e fábrica de objetos; **Não faz parte da avaliação desta fase a interação com o utilizador feita para testar a lógica realizada, no entanto é aconselhada a introdução de mecanismos mínimos de interação que permitam o teste de todas as classes desenvolvidas**;
2. **Segunda Fase**: Implementação completa e funcional do jogo *Planet Bound* com interface de utilizador em modo gráfico (GUI), tendo por base os elementos apresentados e o resultado (*feedback*) da discussão/defesa na primeira fase. O início da implementação da 2ª fase não está condicionado pela prévia defesa da primeira fase. O código correspondente à primeira fase pode ser modificado com vista à obtenção dos objetivos da segunda meta, em particular se este

apresentava deficiências, não significando isso que a meta 1 seja reavaliada ou a sua cotação recuperada.

Na segunda fase, deve ser possível gravar em ficheiro de objetos o jogo que se encontra a decorrer, bem como carregar e continuar um jogo previamente guardado.

Por uma questão de **simplificação**, devem **obrigatoriamente** ser considerados os seguintes esclarecimentos e **alterações às regras originais**:

1. O jogo inicia-se pela escolha do tipo de nave a usar: militar ou exploratória (*mining*);
2. O jogo deve terminar no final do turno em que o último dos cinco artefatos alienígenas é encontrado (i.e., **a fase *Path to the Home World* não deve ser implementada**);
3. Nas cartas *Terrain*:
    a. Não existem obstáculos nem para os drones nem para os alienígenas (estes têm capacidade de voar durante distâncias curtas), pelo que qualquer movimentação entre células adjacentes é permitida (na vertical e na horizontal);
    b. É gerada uma carta representativa da superfície com dimensão 6 x 6, em que as posições onde a nave aterra e onde o recurso se encontra são geradas aleatoriamente, garantindo que não se sobrepõem;
    c. Quando o drone se encontra numa célula adjacente ao alienígena é obrigatório iniciar um ataque, não sendo possível fugir dessa situação.
4. Movimentações da nave pelo espaço:
    a. A fase *Scan* e as cartas S*pace Travel* deixam de existir;
    b. Na fase *Space Travel*, a movimentação da nave obedece às seguintes regras: (1) entre dois sectores do tipo círculo (branco ou vermelho) existe sempre uma passagem por um ponto vermelho; (2) o tipo de círculo atingido (branco ou vermelho) é determinado de forma aleatória com a seguinte distribuição: círculos vermelhos = 3/10 e círculos brancos = 7/10.
    c. Em cada sector planetário o jogador poderá optar por avançar ou explorar os recursos do planeta que ainda não tenham sido explorados. Apenas poderá optar por explorar caso tenha o Oficial com essa competência e tenha o drone de exploração;
    d. O facto de uma movimentação ter sido efetuada através de um buraco negro é determinada de forma aleatória com probabilidade de 1/8;
    e. O tipo de planeta (negro, vermelho, azul e verde) presente num setor do tipo círculo vermelho ou branco é determinado de forma aleatória com probabilidades iguais (distribuição uniforme) para as quatro cores possíveis;

f. São considerados apenas os eventos 3, 4, 5, 6, 7 e 9 do jogo original, passando a ser numerados com 1, 2, 3, 4, 5, 6, respetivamente.

Quando a nave atinge um ponto vermelho, além do evento poder ser gerado de forma aleatória (regras do jogo), deve, igualmente, ser possível aplicar um evento específico (e.g., poderão existir algures no código do modelo os métodos *aplicaEvento()* e *aplicaEvento(int idEvento)*). O objetivo principal deste acrescento **obrigatório** às regras do jogo é facilitar o *debugging* durante o desenvolvimento e defesa.

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

1. Devem ser seguidos os padrões apresentados nas aulas;

2. Deve ser utilizada, de forma adequada, uma máquina de estados para concretizar a lógica do jogo (a Figura 1 ilustra, de forma incompleta e apenas a título de exemplo, uma possível abordagem com identificação de eventos e ações associados às transições entre estados, sendo consideradas as alterações introduzidas nas regras originais);

3. Ao terminar um jogo, a máquina de estados deve permitir ambas as hipóteses de jogar de novo ou de terminar a aplicação;

4. Na fase *Alien Attacks*, a sequência de ações correspondentes ao ataque do alienígena e do *drone*, que não dependem de qualquer decisão por parte do jogador, devem ser automaticamente desencadeadas sem qualquer intervenção do jogador. Os momentos do jogo em que o jogador tem que tomar decisões devem corresponder a estados. Podem existir estados em que o jogador é informado acerca do ponto da situação e apenas decide quando prosseguir. A existência de estados nestas situações depende apenas da dinâmica que pretendemos dar à interação com o utilizador;

5. A aplicação deve apresentar a informação necessária ao acompanhamento e verificação do bom funcionamento do jogo (dados relevantes que caracterizam as diversas cartas usadas no jogo, o turno e a fase atual, as situações de vitória e derrota, etc.). Relativamente ao assunto da apresentação de informação (interface com o utilizador), chama-se a atenção para o fato do ecrã não ter as mesmas limitações de um tabuleiro físico. Significa isto que a interface pode apresentar a mesma informação de forma mais compacta ou inteligente. Por exemplo, em vez de uma série de quadrados para representar combustível ou ocupação do porão, um indicador numérico ou um ponteiro como o do combustível nos automóveis cumprirão o mesmo objetivo de uma forma mais interessante e visualmente mais agradável;

6. O modelo do jogo (máquina de estados) deve, através de um *log* ("registo") interno (não é permitida qualquer tipo de interação com o utilizador), gerar e disponibilizar (note que "disponibilizar" não significa "mostrar" mas sim "tornar acessível") informação detalhada sobre o estado interno e o resultado das várias ações. Por exemplo, deve ser possível a interface do utilizador obter os detalhes do que ocorreu automaticamente (e.g., durante a fase *Alien Attacks*) e apresentá-los ao utilizador (e.g., resultado dos lançamentos dos dados e as suas consequências);

7. Em ambas as fases do trabalho, deve ser utilizado o padrão de separação entre Modelo e Vista estudado nas aulas.

Arquitetura

A figura 1 apresenta uma possível definição de alguns estados, faltando os restantes que deverão ser propostos (os estados já apresentados podem ser substituídos por outros, desde que façam sentido). Os nomes das ações identificadas nas transições da máquina de estados devem ser iguais aos métodos correspondentes no modelo. Por exemplo, tendo por base o diagrama da Figura 1, deverão existir os métodos *start*, *selectShip*, *move* e *nextTurn*. Tal como neste exemplo, estes nomes devem ser sugestivos das ações que representam.

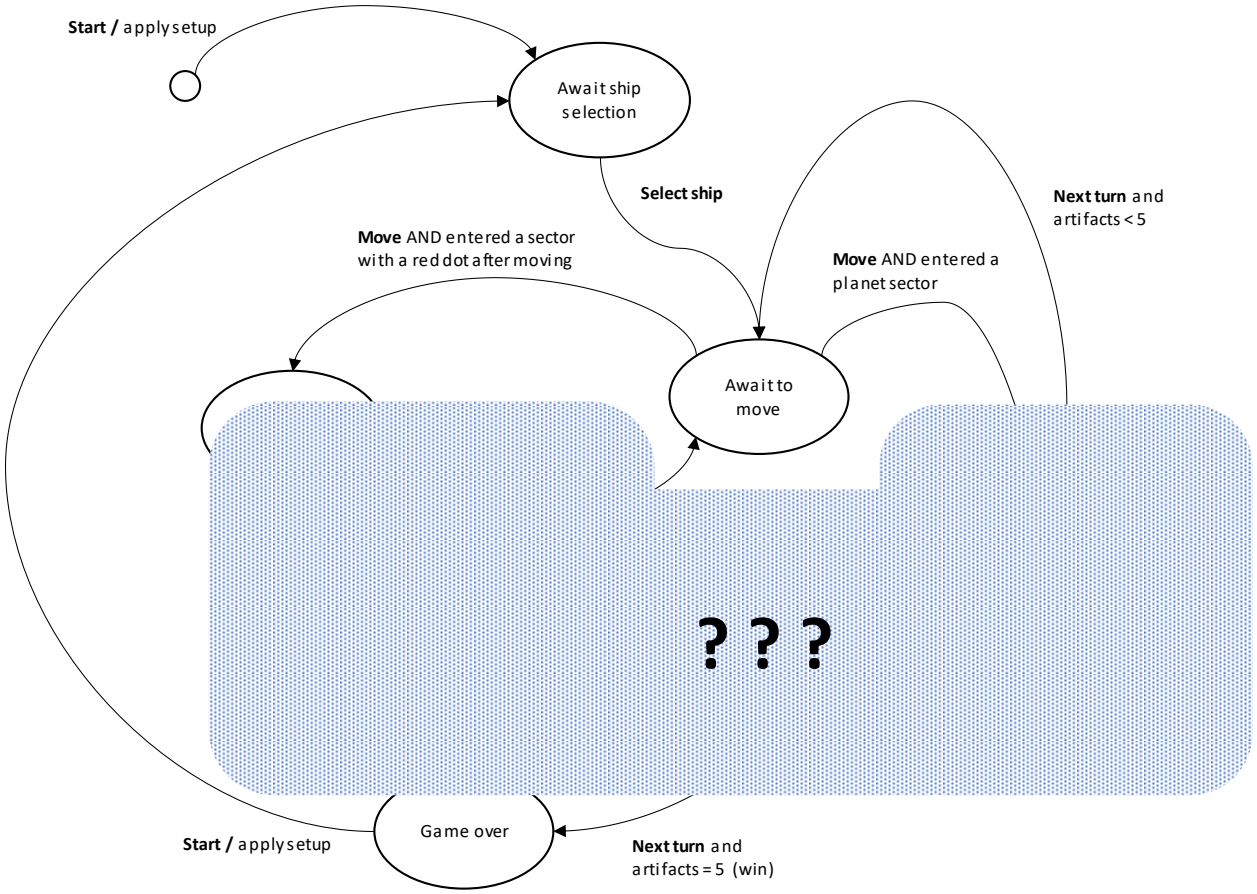

Figura 1 – Versão incompleta de uma possível máquina de estados

A aplicação deve estar organizada em (pelo menos) quatro *packages*: ***logica***, ***logica.estados***, ***logica.dados*** e ***iu.gui***. O *package* ***logica*** corresponde à implementação do modelo/lógica. Os *packages* ***logica.estados*** e ***logica.dados*** conterão, respetivamente, a hierarquia de estados de acordo com a abordagem estudada nas aulas e os dados dos jogo. Os packages ***logica***, ***logica.estados*** e ***logica.dados*** são os elementos a apresentar na primeira fase. O *package* ***iu.gui*** corresponde à implementação da interface do utilizador em JavaFx, sendo este apresentado na segunda fase do trabalho prático. Nas classes do package *iu.gui*, *n*ão deve existir qualquer tipo de lógica associada ao jogo e às suas regras. Por outro lado, a lógica do jogo não pode incluir código que realize, por exemplo, operações de leitura do teclado ou de escrita no écran.