Instituto Superior Manuel Teixeira Gomes

Licenciatura de Engenharia Informática, 3º Semestre

Programação Orientada a Objetos

Prof. Doutor António dos Anjos

Trabalho Prático I – Lunar Lander

Trabalho elaborado por:

Carlos Manuel Pacheco Soares, Nº 21103408

Índice

INTRODUÇÃO:

No âmbito da disciplina de Programação Orientada a Objetos foi-nos solicitado a implementação, utilizando a linguagem de programação Java e o paradigma de programação orientada a objetos, de uma versão bastante simplificada do jogo Lunar Lander. Este que foi lançado, no final da década 70, pela empresa Atari e que se podia encontrar frequentemente em casas de jogos eletrónicos e nos cafés.

Para efeitos de programação optei por utilizar o NetBeans IDE 7.3 Beta 2 utilizando a versão mais recente do Java (Java Standart Edition 7) e o JDK 7.

Em termos de personalização do jogo decidi misturar a temática do jogo um pouco com o tema de “Star Wars”.

FERRAMENTAS UTILIZADAS:

Como já referi na introdução optei por utilizar a ultima versão do Java (Java Standart Edition 7) e a versão mais recente do NetBeans IDE até que o mesmo é uma versão Beta, isto porque a favor de estar sempre atualizado relativamente às novas tecnologias, principalmente se desta forma se contribui para o desenvolvimento de tecnologias livres e *Open-Source*.

O NetBeans IDE 7.3 Beta 2 para Windows possui algumas melhorias em termos de *debugging*, torna-o um pouco mais fácil de compreender e corrigir mesmo antes de correr o programa, no entanto ainda precisa de melhorar na velocidade em que carrega os projetos (principalmente ao iniciar). Mas num ponto de vista geral, esta versão do NetBeans IDE encontra-se muito estável e encontro-me neste momento bastante ansioso por experimentar a versão final (não Beta).

Para ser sincero, não utilizei a nova versão do Java por alguma novidade mais marcante, como por exemplo as novidades do pacote "Swing" ou pela possibilidade de utilizar "Strings" em "Switch Statements", talvez um pouco pelo melhoramento das mensagens de aviso e de erro do compilador, mas sim, porque sou preguiçoso e assim que descobri que nesta versão do Java se podia utilizar, numa situação de um bloco de código poder ter vários tipos de exceções, somente um "catch()" com as exceções todas incluídas em vez de fazer o mesmo para cada uma delas, o que simplifica bastante o código (http://docs.oracle.com/javase/7/docs/technotes/guides/language/catch-multiple.html#multiple).

Para a criação da documentação do programa (API) foi nos proposto a utilização do Javadoc, do qual gerei a maioria do código automaticamente a partir das funcionalidades do NetBeans e o restante (classes e variáveis privadas) adicionei manualmente. Depois o IDE ainda disponibiliza a opção de gerar automaticamente a API, que caso contrário tinha de ser gerada a partir da linha de comandos

(http://docs.oracle.com/javase/1.4.2/docs/tooldocs/windows/javadoc.html#javadocoptions).

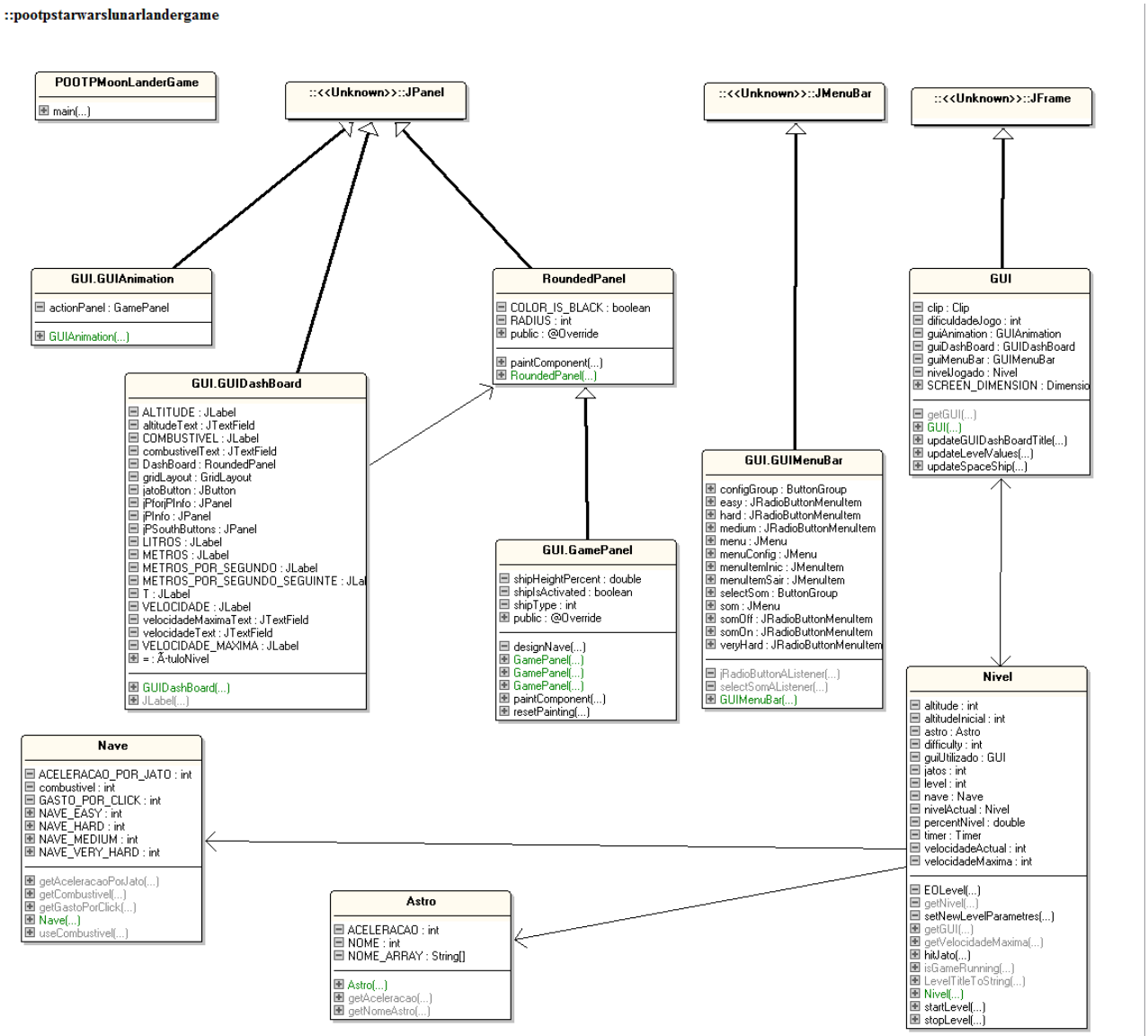
Em termos de planeamento, criei inicialmente um UML em papel em caneta, pois considero esquematizar algo em papel mais intuitivo do que num computador. A partir deste esquema passei logo a escrever o código e por fim, quando o código já estava todo acabado, decidi criar um UML informatizado do resultado final. Para tal, encontrei um programa livre e *Open-Source* bastante desconhecido chamado ESS-MODEL (http://essmodel.sourceforge.net/), que a partir do código Java cria o UML que se pode exportar para imagem ou mesmo para uma página detalhada em HTML.

DIAGRAMA EM UML:

Como referido anteriormente, criei, antes de começar a escrever o código, um esquema simples em UML para poder planificar como criar o programa. Este foi modificado imensas vezes até finalizar o código.

Ao finalizar o código utilizei um *"UML reversing tool"* chamado ESS-MODEL que a partir do código cria o UML. O programa cria um diagrama UML, automaticamente a partir dos ficheiros ".java" que selecionarmos, que depois podemos exportar para imagem PNG ou para um género de API, como o da Javadoc, em HTML.

A imagem do diagrama UML gerado a partir deste programa:



Os ficheiros originais da imagem tanto como da API em HTML, do UML gerados com este programa serão adicionados como apêndices.

Este programa é livre e também *Open-Source* mas no entanto não parece estar bem atualizado às novas versões do Java ou tem simplesmente alguns "bugs". Por exemplo, a notação "@override" utilizadas para indicar que um método foi *overriden* é reconhecido pelo programa como uma variável pública do tipo "@override" sem nome atribuído. E ao dizer que uma classe estende uma das classes *standard* do Java ele indica essa relação mas adiciona que por exemplo a classe "JFrame" é *unknown* (desconhecida). Em compensação o programa é extremamente rápido, leve e não necessita de ser instalado que é uma enorme vantagem para utilizadores de Windows.

FUNCIONAMENTO E *DESIGN* DO JOGO:

No enunciado do trabalho foi proposto que implementássemos uma versão simplificada do jogo *Lunar Lander* em que o funcionamento do mesmo deveria consistir em poder escolher a nave, o nível e o planeta e depois o jogo começar com a nave desenhado no limite superior do painel e que o mesmo depois deveria aterrar dentro da velocidade limite no limite inferior.

Como no entanto a escolha da nave, nível e o astro/planeta não era obrigatório implementar decidi, porque acho que faria um pouco mais sentido, dar só a opção de escolher a nave e que cada nave corresponde a um certo grau de dificuldade. Depois, cada nível que o utilizador passa corresponde a um astro/planeta diferente. Isto porque parece me obvio que cada veículo que nos conduzimos tem o seu grau de dificuldade em termos de condução e nas naves isso não deve ser diferente.

Em termos de definição das variáveis e inicialização da nave e astro, isto é feito na classe chamada nível que no fundo é o motor de jogo, contendo também o Timer que é utilizado para atualizar o estado do jogo de segundo em segundo.

Na criação do GUI optei por um *design* bastante simples baseando principalmente no *design* proposto pelo professor. No entanto, para marcar mais as áreas importantes do *design* criei uma classe adicional que é um “JPanel” mas com as pontas do retângulo do painel arredondadas, que chamei “RoundedPanel”. Este painel ainda tem a opção no construtor de ter o fundo preto ou com um efeito transparente relativamente a cor fundo existente. Ao criar este *design* decidi criar quatro classes internas privadas na classe GUI para separar o painel em quatro componentes, com fim de organizar melhor o código e torna-lo mais fácil de ler e compreender.

Com o objetivo de sobressair um pouco dos restantes colegas e de personalizar mais o jogo optei por adiciona-lo outro tema que achei que se enquadrasse melhor no jogo, “Star Wars”, como já foi referido na introdução. Como tal, utilizei os nomes dos Planetas dos filmes para os

nomes dos astros/planetas no jogo, por exceção da Lua e da Terra e utilizei quatro das dos filmes para as naves utilizadas no jogo tanto como no nome como no desenho. Em anexo entregarei as imagens que utilizei das naves como referência enquanto as desenhava, pelo que as desenhei todas vistas de cima e com a frente da nave virada para o limite superior do painel. Dos desenhos que criei considero que o que foi mais bem-sucedido foi o desenho do ARC-170, que o utilizador poderá conduzir no nível de dificuldade "*Very Hard*".

Para reforçar o tema e como não chegamos a explorar a utilização de ficheiros no programa decidi adicionar uma música de fundo ao jogo que é lançado no construtor do "GUI". Desta forma quando o programa iniciar a música é tocada em *loop* e se o utilizador o quer apagar tem essa opção na barra de menu. A música utilizada é o *Theme Song* dos filmes.

UTILIZAÇÃO DO PROGRAMA:

Irá ser entreguei tanto o código fonte como o ficheiro JAR.

Relativamente ao código se alguém o quiser utilizar está bem clara o funcionamento na maioria a partir de comentários utilizados para a API. Em alguns blocos de códigos em que era mais difícil de a compreender adicionei comentários soltos. Adiciono ainda que se utilizar o código deverá ter em conta as dependências, entra as diferentes classes, claramente visível no UML, ou mesmo na API.

Para utilizar o ficheiro JAR deverá estar sempre presente, na mesma pasta, o ficheiro de música com o nome de “StarWarsThem.wav”. Se não o fizerem, o jogo funcionará na mesma no entanto tudo o que estiver relacionado com a música de fundo não.

CONCLUSÃO:

Durante a execução do presente trabalho poderei dizer que aprendi muito rápido que quando estivermos com algum erro ao programar que não deveremos considerar somente uma direção, porque na grande parte dos casos existe pelo menos um caminho diferente muito mais simples. Poderei considerar que esse tipo de situações me colocaram, neste trabalho, mais entraves.

Posso acrescentar ainda que se tivesse um pouco mais tempo para finalizar o trabalho que teria adicionado, ao desenho da nave, um efeito de que o jato está ativo inclusive também o som do mesmo, como também poderia ter adicionado um fundo diferente por cada planeta e talvez mais algumas opções no *menu*. Mas no entanto considero isso pormenores sem grande importância.