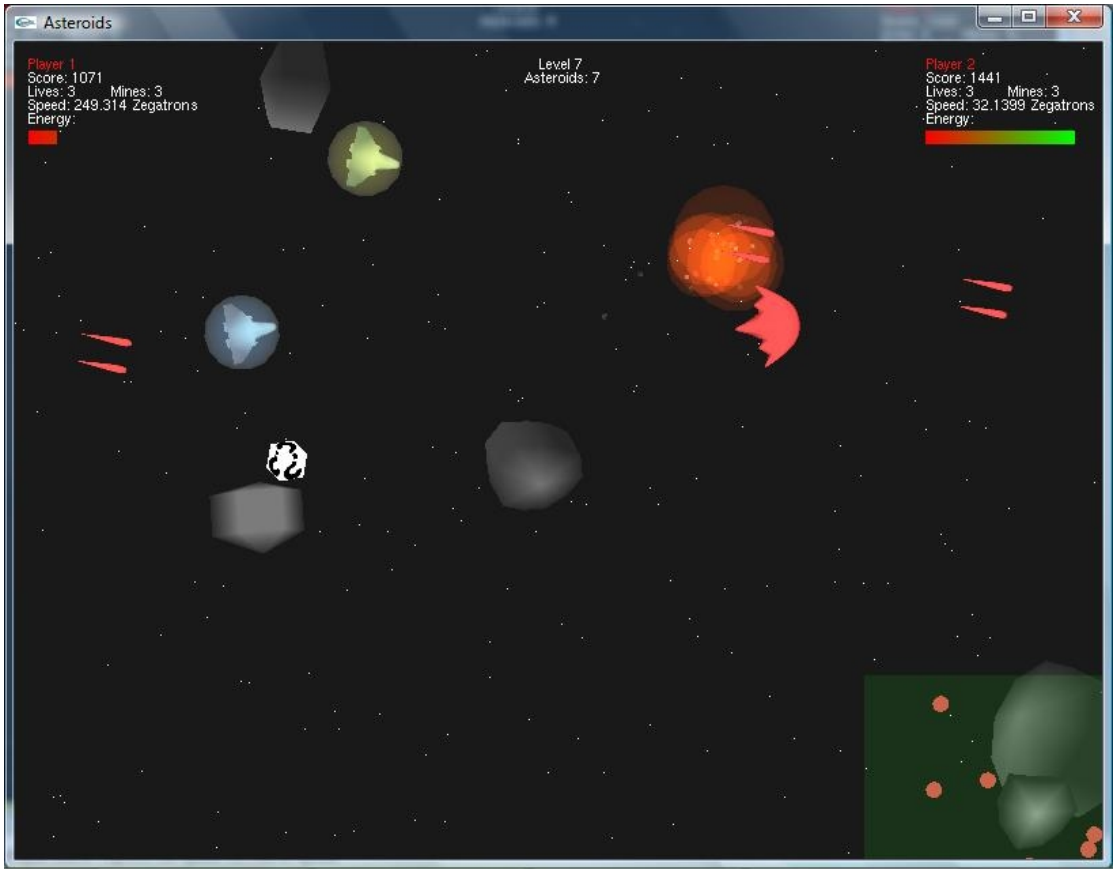
Computação Gráfica
6 de Junho 2008

# Relatório sobre o desenvolvimento da aplicação gráfica interactiva “Asteroids”


| **Miguel Branco** | **Tiago Garcia** | **João Aires** |
|---|---|---|
| 57863 | 58386 | 58537 |
| LEIC-A | LEIC-A | LEIC-A |
| miguel.branco@ist.utl.pt | tiago.garcia@ist.utl.pt | joão.aires@ist.utl.pt |


## Sumário


A aplicação desenvolvida consistiu no *remake* de um dos jogos mais populares da “idade de ouro” dos *arcade games*, *Asteroids*. O ambiente de desenvolvimento foi na linguagem C++, usando OpenGL, GLUT e a cglib fornecida pelos docentes. Tendo em conta que é um *remake*, os traços principais do jogo original mantêm-se, mas para tentar tornar o jogo mais interactivo e divertido, foi dada à aplicação um toque pessoal. Neste relatório vão ser abordados temas como o conceito subjacente à nossa aplicação, vai ser apresentado um manual para o utilizador seguir e saber como jogar, vão ser explicados a arquitectura final da aplicação, os algoritmos mais importantes desta, configurabilidade e flexibilidade do código, descrição do processo de criação do jogo e referência aos aspectos mais marcantes deste e às conclusões tiradas após o desenvolvimento da aplicação.

# 1. Conceito

Embora o conceito do nosso jogo se baseie no *Asteroids*, como já foi dito, tem vários aspectos que divergem do original. Para começar, implementa a possibilidade de *multiplayer*, quer em modo "cooperativo", onde os dois jogadores tentam cumprir os objectivos do jogo em conjunto, quer em modo de "duelo", onde o objectivo não é ganhar pontos, mas sim destruir a nave do adversário.

Outra característica especial da nossa aplicação é a inclusão de *PowerUps* e *PowerDowns* que aparecem de forma aleatória, alargando a experiência de jogo e a longevidade do título. A jogabilidade monótona do clássico está agora artilhada com armas extra (minas e tiro de energia), activação do escudo, mísseis extra, ou ainda desafios para o jogador, como ficar inibido de disparo ou aceleração. Existem outras características especiais do jogo como uso de modelos criados com programas de desenvolvimento gráfico, para as naves, os tiros, as minas e o tiro de energia, que tornam a aplicação graficamente mais agradável, a criação de um menu com imagens criadas por nós que permite o uso do rato para seleccionar as opções do utilizador, entre outras novidades em relação ao original.

# 2. Manual para Utilizador Experiente

Um dos objectivos ao criar esta aplicação foi torná-la *user-friendly* (ou "amiga do utilizador"), isto é, fácil de usar e intuitiva. Como tal, incluímos uma opção no Menu Principal, denominada *How To Play* que explica ao utilizador novato como jogar, desde as teclas/rato para comandar a nave até aos variadíssimos efeitos dos muitos *PowerUps* e *PowerDowns*.

A navegação no Menu também é simples, bastando ao utilizador *clickar* com o rato na opção que quer.

Em qualquer momento do jogo, pode usar-se a função de "pausa" recorrendo para isso à tecla "Esc", direccionando o utilizador para o Menu Principal. Quando a tecla "Esc" é pressionada novamente, o menu desaparece e o jogo continua. De salientar que o Menu no modo pausa continua totalmente interactivo, deixando o utilizador começar um jogo novo, rever o *How To Play* ou mesmo sair da aplicação.

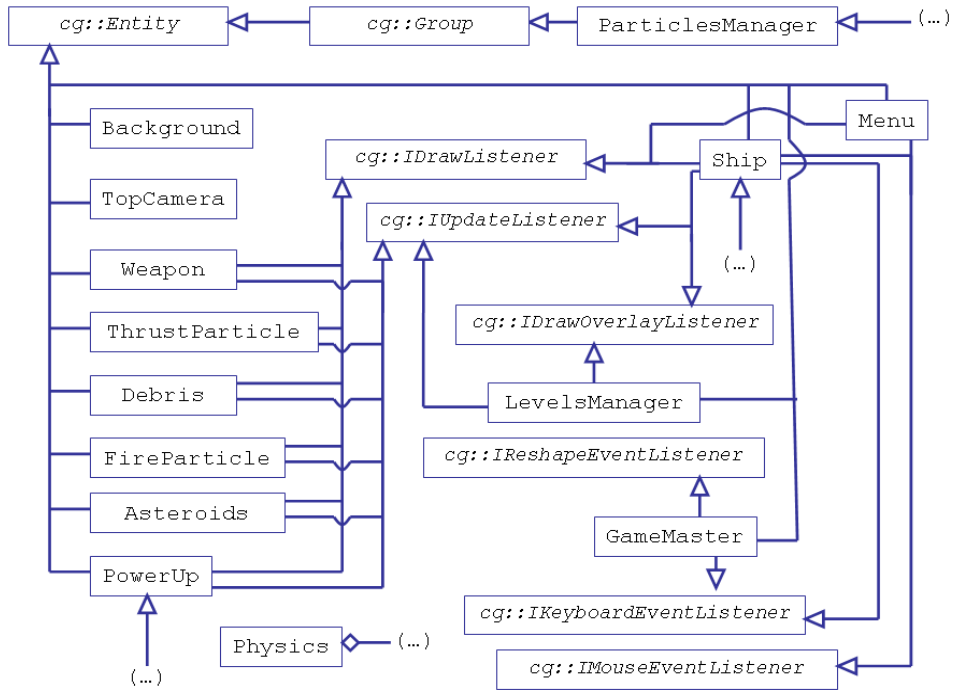
Não fazendo parte do necessário para o bom funcionamento da aplicação, é de referir que o utilizador experiente tem ainda a possibilidade de configurar a aplicação, como descrito na secção 5, Configurabilidade e Extensibilidade.

# 3. Arquitectura

## 3.1. Arquitectura geral

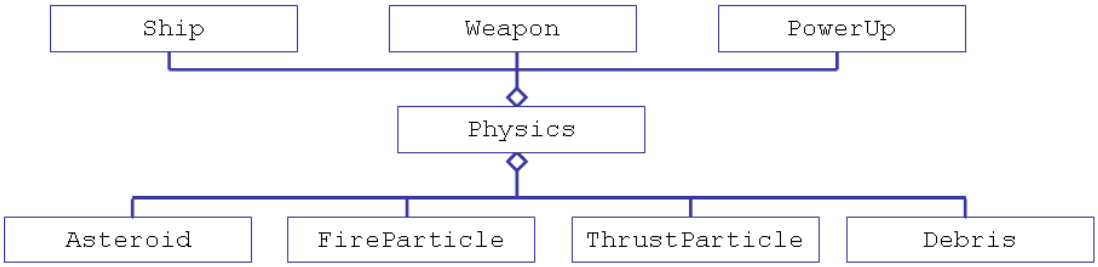
Na imagem seguinte apresentamos aquilo que nós consideramos ser a base simplificada da arquitectura por detrás do jogo desenvolvido pelo nosso grupo. O objectivo desta imagem é o de dar uma ideia muito geral da nossa aplicação aproveitando para apresentar algumas das classes mais importantes e das suas ligações com a cglib. Por questões de espaço e de simplicidade, alguns pormenores foram

propositadamente deixados de lado, sendo que alguns deles serão explicados nas secções seguintes.




3.2. *Physics*, a classe de física

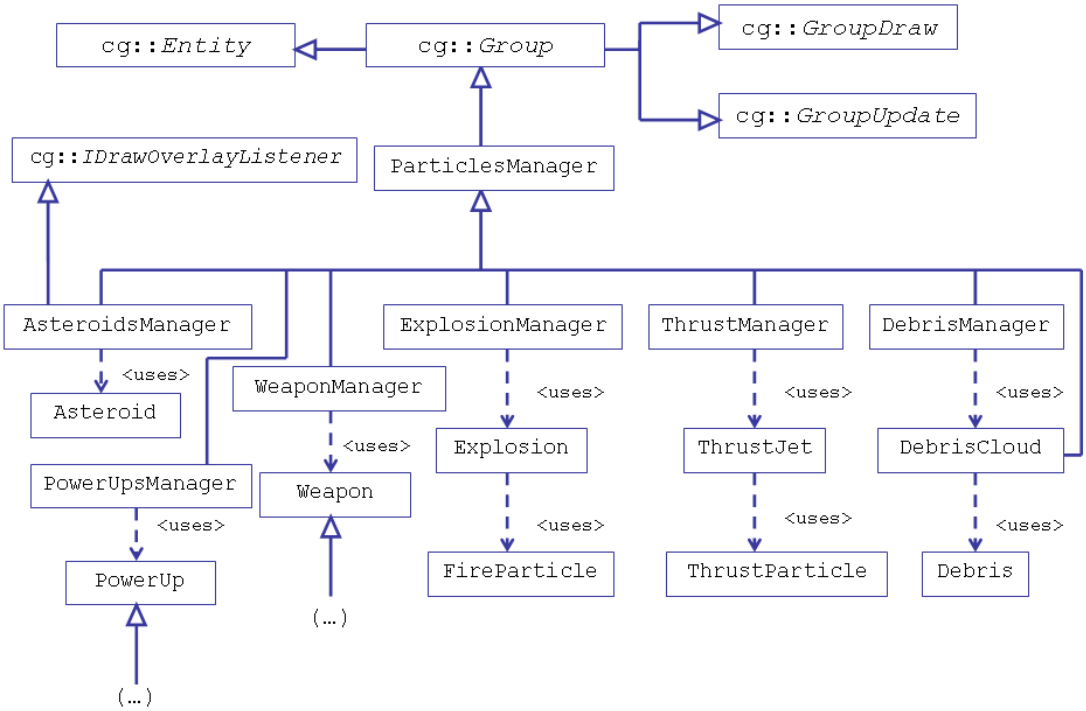
Um dos aspectos fundamentais em qualquer jogo é o movimento e dinâmica que este apresenta ao jogador dando-lhe a sensação de que existe um universo "vivo" dentro do jogo. Por isso mesmo, um aspecto muito importante durante a criação deste jogo foi a implementação da classe responsável pelo processamento em termos físicos de todos os objectos, a classe *Physics*.




A ideia é todos os objectos que se movem terem um objecto *Physics* associado que será responsável por esse mesmo comportamento. Para isso, esta classe é dotada de uma série de atributos (posição, velocidade, massa, tamanho, etc) e de métodos (actualização da posição e da velocidade, detecção de colisão, para além dos habituais *getters* e *setters* bem como de outros métodos que considerámos necessários). Esta classe é não só responsável pelo movimento do objecto mas também, por exemplo, pela detecção de colisão entre dois objectos influenciando (ou não) cada um dos deles (este algoritmo de detecção de colisão e de choque entre objectos será descrito com algum pormenor no ponto 4 deste relatório).

### 3.3. *ParticlesManager*, o gestor de partículas

Um outro conceito importante durante o desenvolvimento deste jogo foi a criação de classes que fariam a gestão de determinado tipo de objectos (asteróides, armas, etc). Ao fim de criados alguns *managers*, tornou-se obvio que seria possível (e aconselhável) generalizar todo o seu comportamento comum numa só classe, a classe *ParticlesManager*.




Tal como se pode observar na figura, todos os outros *managers*, à excepção do gestor de níveis (que difere no aspecto de não gerir nenhum objecto com movimento), herdam do gestor de partículas pois o comportamento é semelhante para cada um dos objectos que cada um dos gestores gere (asteróides, armas, explosões, *PowerUps*, etc). Para não sobrecarregar o diagrama, decidimos não colocar as classes que herdam de *Weapon* e de *PowerUp*.

É de referir que a classe ParticlesManager é uma *template class*, ou seja, é parametrizável pelos seus descendentes, o que permite o nível máximo de liberdade aos mesmos: podem ser gestores de qualquer classe, sendo ParticlesManager<*classe*>.

## 4. Algoritmos Relevantes

Um dos algoritmos que gostaríamos de descrever nesta secção é o algoritmo que efectua a detecção de colisões entre dois objectos da cena. Como o algoritmo que processa o choque de uma colisão (ou seja, o efeito que cada um dos corpos sofre após o embate) está intimamente relacionado com o algoritmo de colisão, aproveitaremos também para abordar este outro algoritmo.

Antes de mais, é apropriado referir que ambos os algoritmos estão presentes na classe de física que já foi referida anteriormente neste relatório o que de resto faz

sentido, uma vez que estamos a falar dos principais algoritmos que simulam a física mais avançada do jogo. O ponto de partida da detecção de colisões tal como a implementámos é o de aproximar o objecto em questão a uma geometria simples e de fácil processamento que, no nosso caso, será uma esfera. Essa esfera é, em termos de código, representada por dois atributos da classe física, o raio da esfera (raio de colisão) e a posição da mesma. Para verificar se existe ou não colisão entre dois objectos, o que fazemos é verificar se a distância entre os vectores posição dos dois objectos é menor do que a soma dos seus raios de colisão, se tal acontecer é porque existe sobreposição entre as esferas envolventes o que significa, segundo a nossa implementação, que existe colisão entre os objectos. Para que este algoritmo funcione e produza resultados realistas aos olhos do jogador, torna-se portanto fundamental encontrar raios de colisão adequados a cada tipo de objecto, valores esses puramente empíricos.

Depois de ser detectada uma colisão, é de esperar que algo aconteça a cada um dos corpos (ser destruído, repelido, etc). Para tal usamos o algoritmo de choque referido. Dado o ponto de colisão entre os dois objectos, o que este algoritmo faz é obter o vector de colisão normalizado (que não é mais do que a subtracção entre o vector de posição e o ponto de colisão, sendo depois normalizado). Este vector será então adicionado ao antigo vector velocidade do corpo, faltando apenas obter valores multiplicativos para este vector de forma a criar nas colisões o efeito pretendido. Para tal, serão ainda necessários outros dois parâmetros, a força do choque, bem como a massa do corpo. A equação para obter o novo vector de velocidade é a seguinte:

$$(Force * CollisionVector * VelocityVector)/mass$$

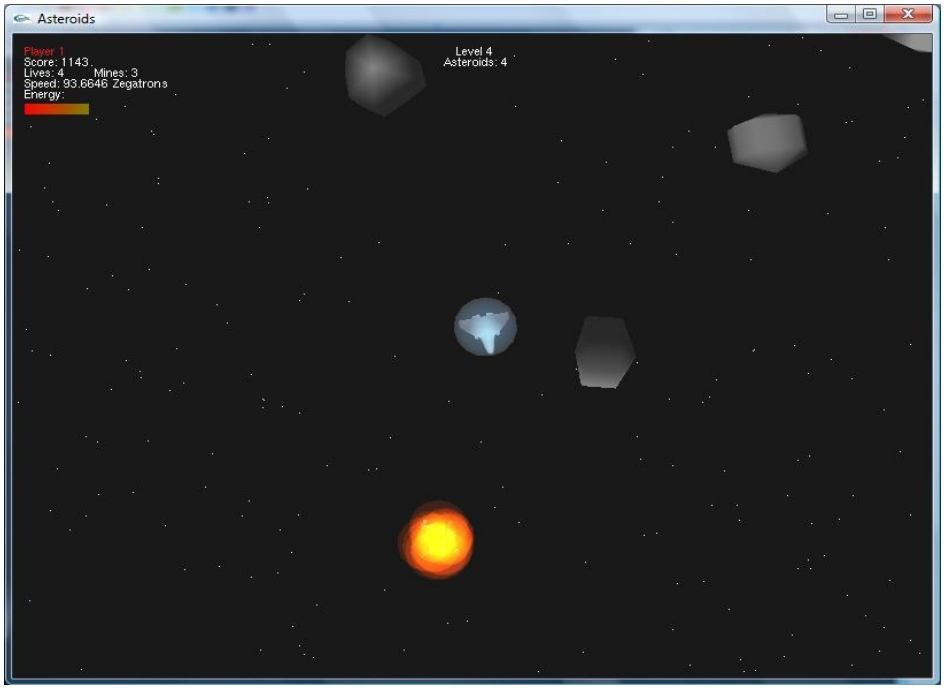
A utilização desta equação permite obter resultados bastante bons uma vez que relacionamos a força aplicada com a massa do corpo. Para corpos de igual massa, uma vez que a força é um múltiplo da massa, haverá simples troca de energia cinética, para corpos de massas diferentes o que iremos observar é que o corpo com maior massa sofrerá menos alteração no seu vector velocidade, o que de resto está de acordo com a realidade.

Outro algoritmo que gostaríamos de abordar é o algoritmo responsável pela destruição dos asteróides. Este algoritmo é chamado cada vez que ocorre uma colisão entre um asteróide e um tiro ou uma mina. Antes de mais é fundamental eliminar, do gestor de asteróides, o asteróide atingido. De seguida, a ideia é criar uma animação realista que transmita ao jogador, em termos visuais, que ocorreu uma explosão após a colisão entre aqueles dois corpos, para tal recorremos ao DebrisManager e ao ExplosionManager para criar, respectivamente uma nuvem de destroços e uma explosão para completar a animação de forma credível e agradável ao olho do jogador. A nuvem de destroços corresponde a uma série de pequenas partículas que partem do centro do asteróide com velocidade, ângulo e tempo de vida aleatórios (mas com valores compreendidos numa determinada gama adequada à simulação), criando então a ideia clara de que houve fragmentação do asteróide em várias partículas e não

só nos asteróides que irá originar (caso tenha raio suficiente para tal, caso contrário considera-se que o asteróide é pequeno e não haverá divisão deste asteróides em mais nenhum). Por outro lado, uma explosão é composta por várias FireParticles que são esferas cujo raio e transparência variam com o passar do tempo, sendo que o raio é mínimo no início de vida da explosão e a transparência crescente ao longo da mesma até que desaparece por completo. Com a combinação destes dois efeitos gráficos, cria-se então uma animação relativamente complexa e bastante realista. Por fim, falta apenas verificar se o raio médio do asteróide que explodiu é suficiente para este se dividir em novos asteróides, se tal se verificar são adicionados novos asteróides, mais pequenos, à cena. Existe ainda a possibilidade de a destruição do asteróide originar o aparecimento de um *PowerUp*, este acontecimento é no entanto aleatório tal como o tipo de *PowerUp* que poderá (ou não) aparecer após a explosão.

De realçar que os asteróides se dividem dinamicamente, isto é, não há tamanhos predefinidos como sugerido no início do projecto, mas sim cálculos simples para criar novas entidades com cerca de metade do tamanho actual. Isto assegura uma boa modularidade nos níveis do jogo, podendo cada nível ter as suas próprias dimensões para os asteróides.

Para ilustrar o efeito criado quando ocorre uma explosão, decidimos colocar um *screenshot* de uma explosão ocorrida durante o jogo. Obviamente tratando-se de um imagem fixa, muito do seu efeito é perdido, consideramos no entanto ser um bom indicador do aspecto que uma explosão terá no nosso jogo.




## 5. Configurabilidade e Extensibilidade

Na criação desta aplicação mantivemos em mente o objectivo de criar código modular e facilmente extensível, o que é bem apoiado pelo paradigma da programação por objectos. Tivemos um cuidado especial com dois tipos de expansão da aplicação: Configuração da jogabilidade para uma experiência de jogo mais

apurada e Manutenção e Desenvolvimento de Código, para novas iterações da aplicação em si.

No que diz respeito à configurabilidade, tentámos ao máximo permitir a alteração de certos aspectos usando o ficheiro config.ini (devidamente comentado) para a definição de muitos valores (essencialmente numéricos e booleanos).

Assim, é possível alterar parâmetros do jogo sem ter que o compilar de novo, o que é essencial para um jogador que não quer mexer na programação da aplicação mas que ainda assim gostaria de configurar a jogabilidade a seu gosto.

Neste ficheiro (config.ini), os valores a parametrizar seguem uma ordem comentada e perceptível, que vamos no entanto explicar detalhadamente:

O primeiro valor que o utilizador pode configurar é relativo ao pano de fundo do jogo e consiste no número de estrelas presente no ecrã.

Depois temos as constantes relacionadas com os asteróides . Começamos pela sua dimensão mínima definida pela variável MINDIMENSION, usada pelos asteróides no momento da explosão, para "saberem" se se vão dividir ou não (para mais detalhe, é descrito na secção 4, Algoritmos Relevantes, o sistema de divisão dinâmico dos asteróides).

Também é aqui definido o número mínimo e máximo de lados numa face de um asteróide, associados a MINSIDES e MAXSIDES respectivamente. A seguir, temos definido a velocidade de rotação dos asteróides na variável ASTANGSTEP e a massa destes (para efeitos físicos de colisão) na constante ASTMASS. Por fim temos a rotação aplicada aos asteróides em cada eixo definida pelas variáveis ROTX, ROTY e ROTZ.

A parte seguinte do ficheiro diz respeito a aspectos relacionados com as naves. Começamos com as vidas com que cada nave começa o jogo, que está ligado ao parâmetro LIVES. Depois, em SCALEFACTOR, configuramos o factor de escala usado para acertar o tamanho do modelo da nave. O valor seguinte, SHIPANGSTEP, diz respeito à velocidade de rotação da nave no jogo. Depois temos os valores do raio da nave e da sua massa nos campos SHIPRAD e SHIPMASS respectivamente.

A invencibilidade máxima é dada por MAX_INV.

O raio que deve ser usado para calcular colisões com o escudo da nave é SHIELDRAD e a barra do escudo no HUD deve ter um comprimento de SHIELDBAR. Já o ASTEP, ALIMIT, DSTEP e DLIMIT definem os passos de aceleração e desaceleração, o limite de velocidade e o limite de velocidade mínima para a nave parar efectivamente (conforme os nomes indicam).

Passando então aos parâmetros das armas, temos a vida do tiro normal em SHOTLIFE, a sua velocidade em SHOTSPEED, quanto ele deve avançar a princípio com SHOTINIT, o seu tamanho para as colisões em SHOTSIZE, a massa em SHOTMASS e a escala do modelo em SHOTSCALE. A mina (Mine) tem os mesmos campos à excepção da vida e da velocidade, pois não as usa, dando lugar à constante MINEAMMO, que define a quantidade de minas disponível à Nave quando esta entra no jogo. O tiro de energia tem os óbvios CHARGESHOTLIFE, CHARGEINIT,

CHARGESHOTSIZE, CHARGESHOTMASS e CHARGESCALE, aos quais se acrescentam o CHARGEMAXTIME, que dita o tempo que demora a arma a carregar, o CHARGEDISTANCE que define a distância do tiro à nave enquanto carrega, o CHARGEENERGY para a energia inicial da nave e o CHARGECONSUME para o valor de energia que consome a cada passo do carregamento.

De seguida são as duas constantes directamente ligadas à jogabilidade:

MOUSECONTROL para explicitar o uso ou não de rato e MINIMAP para ligar ou desligar o *minimap* dos asteróides.

Resta a configuração dos *PowerUps* e *PowerDowns*, que consiste no tempo de vida deles: PUPLIFE, na probabilidade de cada um, <tipo>PROB, e para os *PowerDowns* o utilizador pode também definir o tempo que a nave fica afectada, com SHOOTDISABLE e THRUSTDISABLE para os tiros e para os propulsores, respectivamente.

Além destes parâmetros de configuração, também é fácil configurar alguns aspectos visuais externos, nomeadamente os fundos e botões do Menu, o *How To Play* e os modelos das naves, tiros e minas. São coisas que achamos inofensivas o utilizador comum mudar, visto termos o efeito de escala também parametrizável pelo config.ini (como referido anteriormente).

No que toca à Manutenção e Desenvolvimento de Código, toda a aplicação foi implementada de forma a permitir uma fácil e rápida adição de novas classes, nomeadamente *PowerUps*, naves, asteróides e até mesmo sistemas de partículas para mais efeitos visuais. Novos níveis são adicionados acrescentando simplesmente um *if* com o número do nível e a lista dos asteróides que devem aparecer nesse nível (isto na classe LevelsManager). Achámos que o utilizador comum não quer e não deve ter acesso aos níveis de jogo, responsáveis pela dificuldade e não pela jogabilidade do mesmo. É uma característica que, para nós, deve ser bloqueada em *compile time*.

Como ela, também os efeitos visuais com base nos gestores de partículas são definidas no código (*hardcoded*) por fazerem parte do conjunto de informação que não deve estar presente de forma configurável, por ser inerente à aplicação. Foram decisões difíceis de tomar mas que achamos contribuir para o sucesso do projecto como jogo tendo em conta as necessidades e compreensão gerais do público-alvo da aplicação.

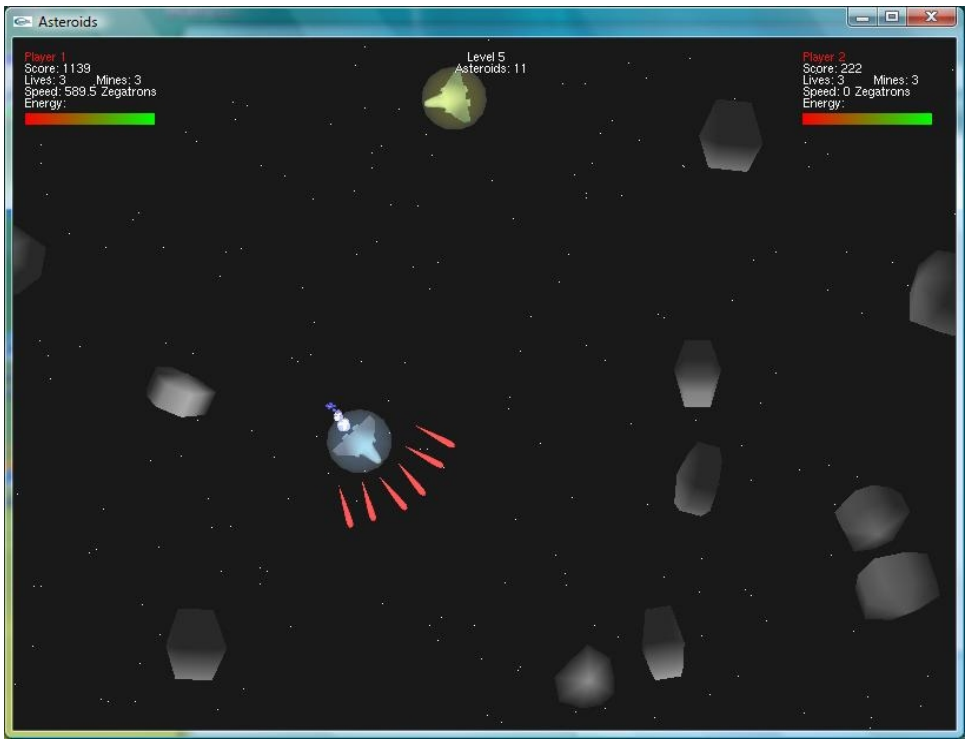
## 6. Post-Mortem

Acabado o projecto estamos orgulhosos de poder dizer que conseguimos atingir a maior parte dos objectivos a que o grupo se propôs. A criação de modos *multiplayer* foi, desde o início, um objectivo muito claro a seguir e o realismo a “bandeira” usada durante todo o desenvolvimento do projecto. No geral, ficámos orgulhosos pelos efeitos gráficos que conseguimos obter, considerando que são de uma qualidade

bastante aceitável e que simulam o pretendido de forma adequada e com alguma beleza.

A nossa inexperiência inicial para com o material de desenvolvimento deste projecto foi uma barreira que tivemos que enfrentar mas que acabou por ser ultrapassada com a prática e com a obtenção gradual de maior conhecimento e experiência na utilização destas ferramentas.

Uma outra pequena dificuldade que encontrámos prendeu-se com a simulação física do jogo que, uma vez que era nossa intenção criar um simulador realista, nos tomou algum tempo até atingirmos resultados próximos da realidade. Para ultrapassar esta barreira tivemos que recordar alguns conhecimentos de física que julgávamos perdidos. No fim, acreditamos ter conseguido obter um sistema de física que consideramos bastante aceitável e que se comporta como é esperado (tanto em termos científicos como em termos de percepção do ser humano).Um dos objectivos iniciais que ficou, com muita pena nossa, por concluir foi a texturização da nave, o que daria um maior realismo à mesma e ao jogo como um todo. Tal não foi possível por falta de tempo e por esta tarefa ser relativamente complicada uma vez que era ainda nossa intenção criar uma série de texturas dando ao jogador a possibilidade de escolher a sua *skin* favorita para a sua nave o que adicionava ao jogo alguma personalidade do jogador. Outro objectivo que ficou por cumprir, este já mais secundário, era a introdução de efeitos sonoros no jogo, nomeadamente quando era, por exemplo, disparado um míssil ou existia alguma colisão. No entanto, temos de admitir que o que nos custou mais não realizar foi a tabela de *HiScores*, que seria guardada num ficheiro à parte e carregada a cada vez que o jogo iniciava. Mais uma vez, por questões de tempo não nos foi possível adicionar essa característica.




*ScreenShot* que ilustra a utilização do *triple-shot*, um dos muitos *PowerUps* que podem ser obtidos durante o jogo e certamente, um dos mais úteis.

# 7. Impressões Finais e Conclusões

O desenvolvimento deste projecto correu bastante bem, pois cumprimos todos os objectivos que corpo docente nos impôs e ainda lhe demos o nosso toque pessoal, incluindo ideias nossas no desenvolvimento da aplicação.

Foi bastante bom serem feitas avaliações quinzenais, pois ajudaram a evitar deixar as coisas para o fim, permitindo-nos ter tempo para acabar o projecto e ainda implementar essas ideias.

Ficámos muito satisfeitos com o fruto do nosso trabalho. Achamos que o resultado final foi um bom jogo, divertido, e graficamente agradável. Orgulhamo-nos de ter implementado diferentes modos de jogo, como *Singleplayer*, e *Multiplayer*, que por sua vez se divide em *Duel* e *Cooperative*, de termos incluído *PowerUps* e *PowerDowns*, e ainda de acrescentar armas ao jogo. Por fim, gostámos do resultado final do menu, tanto das imagens usadas como dos botões e da possibilidade de navegar por ele usando apenas o rato, o que dá à aplicação um aspecto profissional.

Este foi o primeiro jogo completo que fizemos e esperamos que seja tão divertido jogar como foi criá-lo e desenvolvê-lo, e apesar de ter sido o primeiro, esperamos que não seja o último.

# Referências

Uma vez que não sentimos necessidade de referir nenhum *website* ao longo do relatório, não nos alongaremos mais nesta secção.