Webvaders GL - Implementação em WebGL de Space Invaders

Nuno Miguel Soares Silva [72708] Pedro Marques Ferreira da Silva [72645]

***Resumo* –O trabalho proposto para o projeto da unidade curricular de Computação Visual é o clássico jogo de arcada Space Invaders desenvolvido em WebGL. Para o efeito foi necessário implementar uma página com HTML, Javascript e WebGL.**

Pensou-se em criar uma plataforma user-friendly, tanto a nível de opções como da implementação usando WebGL. Este relatório vai se concentrar na descrição da arquitetura desenvolvida e do código que foi escrito.

O relatório apresenta passo-a-passo as decisões tomadas no desenvolvimento do jogo, assim como um esclarecimento do que foi usado.

I Implementação Inicial

O código base usado foi o exemplo 25 das aulas de WebGL pois era o exemplo em que eram usados models em formato .OBJ.
Inicialmente criaram-se os próprios models que iriam fazer parte do jogo no 3D Builder (programa da Microsoft que está incluído no Windows 10). Foi usado node.js em conjunto com jQuery para ultrapassar desafios que foram aparecendo durante a implementação.
Esta foi a primeira abordagem que houve na implementação do jogo.

I-A 3D Builder

Pensou-se em usar o Blender mas devido à complexidade do programa cedeu-se facilmente à desistência de continuar utilizar essa ferramenta. O 3D Builder da Microsoft foi escolhido por ser simples de usar e poder importar/exportar models em .OBJ (formato que iria ser usado para carregar os models posteriormente). Foram criados os seguintes models em 3D para serem carregados no jogo:

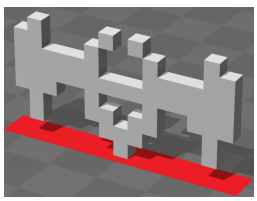

Fig. 1: Invader Boss

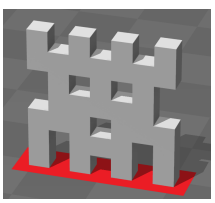

Fig. 2: Invader 3

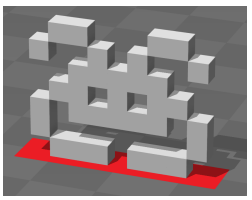

Fig. 3: Invader 2

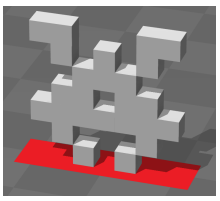

Fig. 4: Invader 1

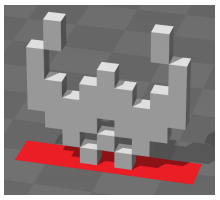

Fig. 5: Nave do jogador

Devido a problemas de exportação do 3D Builder para os ficheiros .OBJ serem lidos em WebGL, a construção destes models foram feitos cubo a cubo a partir do ficheiro *cubo.obj* fornecido nas aulas práticas.

I-B Node.js e jQuery

Um dos primeiros obstáculos com que se deparou foi idealizar uma forma de carregar a partir do sistema de ficheiros os models em .OBJ apresentados previamente. Em Javascript puro é simplesmente impossível implementar essa solução.
Inicialmente, pensou-se em desenvolver uma aplicação com node.js que permitisse alojar o ficheiro HTML num servidor (server.js). Este servidor era necessário pois juntamente com jQuery seria possível carregar os ficheiros .OBJ automaticamente sem ser necessário input do utilizador através de funções *app.get()* e exportando-as para o WebGL com uma função *parseOBJ()*.

I-C WebGL

Ao abrir o browser para visualizar o ficheiro HTML é arrancada uma função *runWebGL()* que inicializa o *canvas* com a função *initWebGL()*. De seguida arranca os *event listeners* com a função *setEventListeners()* e inicializa os buffers usando a função *initBuffers()*. Cada ficheiro .OBJ seria carregado e transformado para um array de vértices. Para este efeito foi desenvolvida uma função *getObjFile()*

que fazia uso da função *parsingobj()* tornando o conteúdo do .OBJ interpretável pelo WebGL através de um array. Quando fosse necessário desenhar um model, o programa iria arrancar uma função *DrawScene()* que inclui funções *drawModel()* para cada model a ser desenhado, e inicializa um buffer *triangleVertexPositionBuffer* e outro buffer *triangleVertexColorBuffer* para ligar os vértices e cores dos models aos buffers respetivos usando a função *initBuffers()*. Foi utilizada, a seguir, a função *computeIllumination()* para tratar do array de cores correspondente ao model a desenhar. Esta função criava um foco de luz que iluminava os models e lhes proporcionava cor. Inicialmente optou-se por um tom de roxo nessa luz.
O jogo terá sempre uma projeção em perspetiva para dar uma noção de profundidade. Todos os invaders inicialmente rodavam no eixo YY alternando a direção cada vez que atingiam um valor absoluto de rotação superior a 30º. Para esse efeito existiu uma função *animate()* que usou uma variável *angleYY* que animava os invaders no eixo YY.
O invader boss e o jogador tinham translações horizontais calculadas com a variável *tboss* e *tplayer* respetivamente. A *tboss* era calculada automaticamente pela função *animate()*. Por outro lado era necessário input do teclado para interagir com a variável *tplayer*.
Ainda existe uma variável *tdown* que indica a translação vertical dos invaders. Numa fase inicial o jogo teria este aspeto:

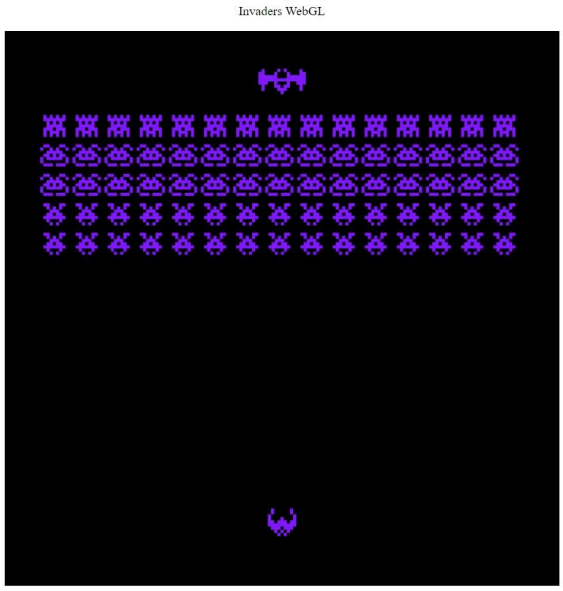

Fig. 6: Interface numa fase inicial

Nesta fase ainda não estariam implementadas todas as funcionalidades do jogo e por isso o programa ainda não era jogável nesta altura.

I-D Arquitetura inicial

Explicadas todas as componentes que constituem a plataforma do jogo, segue-se abaixo uma representação da arquitetura nesta fase:

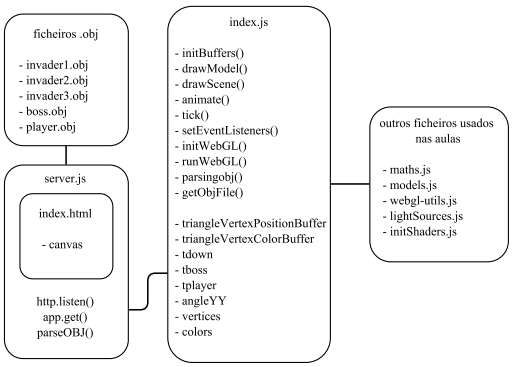

Fig. 7: Arquitetura na fase inicial de implementação

Os resultados desta implementação não foram de encontro com as expectativas. Vários problemas surgiram principalmente sobre o desempenho da interface. Os models eram redesenhados sensivelmente a cada segundo o que era uma taxa extremamente baixa. Ao usar a ferramenta de *profiling* do Chrome analisou-se a seguinte situação:

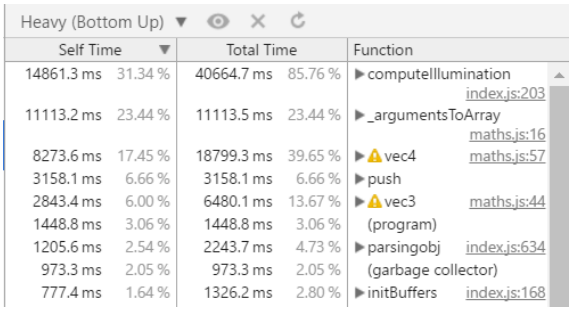

Fig. 8: Resultado do desempenho pelo profiler

É possível ver que o programa consumia 85.76% do seu tempo na função *computeIllumination()* pelo que esta teria de ser removida e algumas outras alterações teriam de ser feitas.
Os ficheiros .OBJ não tem dados sobre as cores, apenas da posição dos vértices. A cor dos models era calculada por esta função, portanto na nova implementação teria de existir outra maneira de obter um array *colors* sem o uso da função.

II Implementação Final

Para simplificar a execução do programa foram feitas algumas mudanças. Esta secção vai-se focar em como foram feitas essas alterações e que novas funcionalidades foram adicionadas.

II-A Remoção de node.js, jQuery e ficheiros .OBJ

Usar ficheiros .OBJ para serem lidos e transformados num array de vértices era inútil quando poderia haver um array previamente definido no código com a mesma informação que é obtida após a leitura do .OBJ.

Esta alteração iria promover o desempenho do programa e excluir o uso de node.js, jQuery e ficheiros OBJ simplificando a implementação.

II-B WebGL - alterações e adições

Ao nível do WebGL várias mudanças foram feitas.

Em primeiro lugar, foram criados os arrays chamados *invader1, invaders2, invaders3, boss, player* e *cube* cujo conteúdo é a posição dos vértices de cada model e onde o *cube* é um cubo que vai ser usado como o projétil que o jogador dispara sobre os invaders.

O problema que surgiu de seguida foi como obter um array de cores para cada array de vértices. Experimentou-se igualar o array de cores ao array de vértices no momento anterior ao desenho do model. O resultado foi surpreendente.

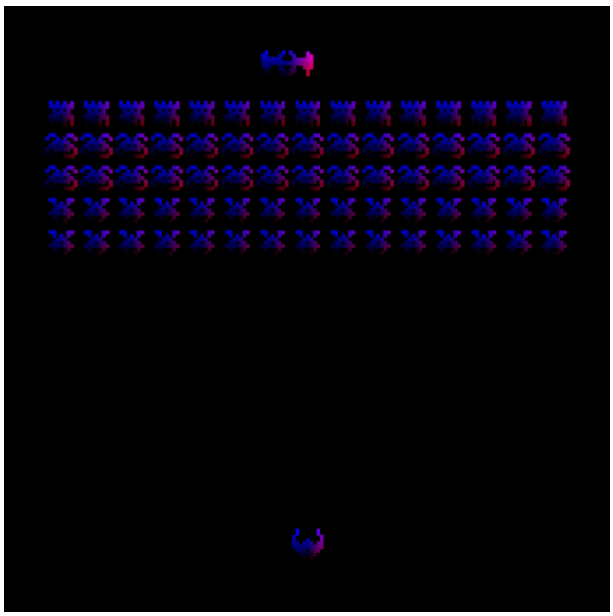

Fig. 9: Resultado de *colours=vertices*

Os models foram desenhados com tons de azul escuro com um *degrade* para rosa o que imitava o efeito de uma luz e proporciona uma noção de três dimensões o que não aconteceria se os vértices tivessem todos a mesma cor.

Neste momento é possível perceber que o desempenho do jogo teria melhorado consideravelmente dado que a taxa de renderização estava muito mais elevada superando o que era expectável.

Inicialmente tinha-se pensado em ter vários focos de luz para diferentes models. Neste caso seria fácil agora manipular o array *colours* antes de executar o desenho 3D. Após adicionar esses blocos de código o aspeto do jogo seria este:

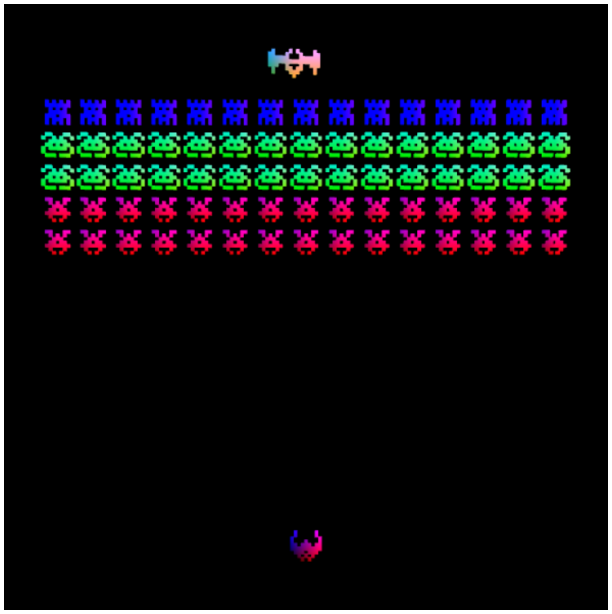

Fig. 10: Resultado da manipulação das cores

II-C Funcionalidades do jogo

Agora com um jogo esteticamente melhorado seguimos para a implementação do jogo propriamente dito.
Em primeiro lugar foi criada uma variável *angleplayer* que altera a rotação no eixo YY do jogado quando este interage com a sua variável de translação *tplayer*.
De seguida incluiu-se o projétil no jogo. Este é desenhado em função da variável*txbullet*, *tplayer* e duma variável de translação vertical *tybullet* que é inicializada com o valor *1.3*. Isto permite que o projétil esteja acima e fora do canvas dando a ilusão de que não existe. Foi implementada a opção de premir a barra de espaço ou o W para disparar o projétil. O que acontece neste momento é que as variáveis *tybullet* e *txbullet* ficam ao nível da nave do jogador e a função *animate()* trata de incrementar o valor *tybullet* para lançar o projétil.
Como só existe um projétil (senão teria de haver mais do que uma variável para controlar a posição de cada projétil o que assim simplifica a implementação do jogo) houve necessidade de criar uma variável booleana *bulletready* que é *true* se o projétil estiver fora do canvas e é *false* se o projétil se encontrar dentro do canvas. Isto impede que o jogador lance o projétil quando este ainda se encontra a meio do canvas.
A fase seguinte seria arranjar forma de fazer "desaparecer" os invaders quando o projétil os atingisse. O problema que surgiu aqui foi que o WebGL não deteta colisões nem consegue "eliminar" completamente um model. Para este efeito foi criado uma matriz 5x15 *invpos* cujo conteúdo era inicializado a 0. Esta matriz foi útil pois permitiu adicionar translação horizontal para fora do canvas a cada invader

que tenha sido atingido. Quando era detetada colisão com o projétil a matriz passava a ter valor 3 na posição do invader em questão. Para detetar colisão foi feita uma função *killinvaders()* que, resumindo, deteta quando o projétil se encontra num intervalo definido pela sua posição e pela posição dos invaders. Para o invader boss criou-se uma variável *tdboss* cujo efeito seria o mesmo. Podemos ver o que se passa por detrás do canvas na seguinte imagem.

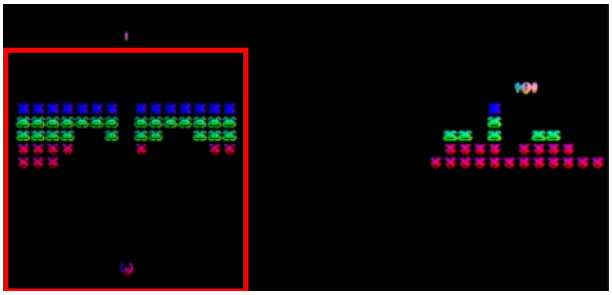

Fig. 11: Acontecimentos dentro e fora do canvas

Podemos visualizar na imagem acima um quadrado vermelho que representa os limites do canvas. Fora deste à direita temos os invaders que já foram derrotados com uma translação incrementada de 3, e acima do canvas podemos ainda ver o projétil no estado *bulletready=true*.

Seguidamente criou-se uma variável booleana *gameover* que verificava se o *tdown* em função de outra variável *rowskilled* inicializada na função *animate()* ultrapassava um certo valor. Por outras palavras, indicava se os invaders teriam atingido a posição do jogador, o que terminaria o jogo.

Foi implementada uma função *resetlevel()* que repunha os valores iniciais dos invaders após estes serem todos atingidos, recomeçando assim o nivel.

II-D Funcionalidades adicionais

Após esta implementação anterior temos as funcionalidades mínimas do jogo. Seguem-se agora extras que foram adicionados ao jogo.
Foi implementada uma opção de pausar o jogo. Para isto foi adicionada à função *setEventListeners()* a hipótese de ao premir a tecla P e usando uma variável booleana *pause* ser possível ativar e desativar esta variável. Quando *pause = true* as animações e controlos que interagem com a nave do jogador são desativadas, proporcionando assim um estado de pausa.
Foi determinado que ao abrir a página HTML o jogo estaria no estado *gameover = true* e foi definida na função *setEventListeners()* a opção de premir a letra R para reiniciar ou começar o jogo sempre que esta variável se encontra ativa.
Foram adicionados também sons ao jogo. Os sons foram obtidos de sites que se podem encontrar nas referências. Escolheu-se o tema *1741 (The Battle of Cartagena)* de *Alestorm* numa versão *8-bit* para obter o ambiente clássico dos jogos antigos. Outros efeitos de sons foram também escolhidos de forma a serem integrados no ambiente do jogo. Por senso comum, é sabido que nem a toda a gente poderia agradar os sons dentro do jogo e por isso foi implementada uma variável *muted* que é interagida com o rato a partir de um ícone de som da interface que está ligada a uma função *mute()*. Resumindo, esta variável impossibilita a reprodução de sons caso esteja ativa.
Foi concebida uma variável *level* e uma variável *points*. A variável *level* é incrementada quando todos os invaders forem derrotados (com exceção ao boss que optou-se por ser um bónus). A variável *points* é incrementada de 10 por cada invader normal atingido e 100 pelo invader boss. Isto permite ao jogador ter uma pontuação final quando terminar o jogo.
Para visualização destes dados foi concebida uma interface com um tipo de letra ao estilo dos antigos jogos de arcada e incluiu-se um conjunto de instruções para o jogador saber quais as teclas premir. Por uma questão estética decidiu-se fazer um wallpaper para o site. O aspeto final é o seguinte.

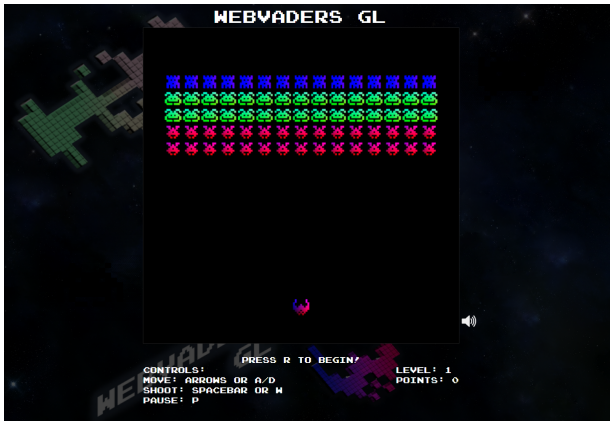

Fig. 12: Aspeto final da página

II-E Arquitetura final

Após implementar as melhorias e otimizações acima referidas podemos obter uma arquitetura mais simplificada. A seguinte imagem mostra uma representação da arquitetura final:

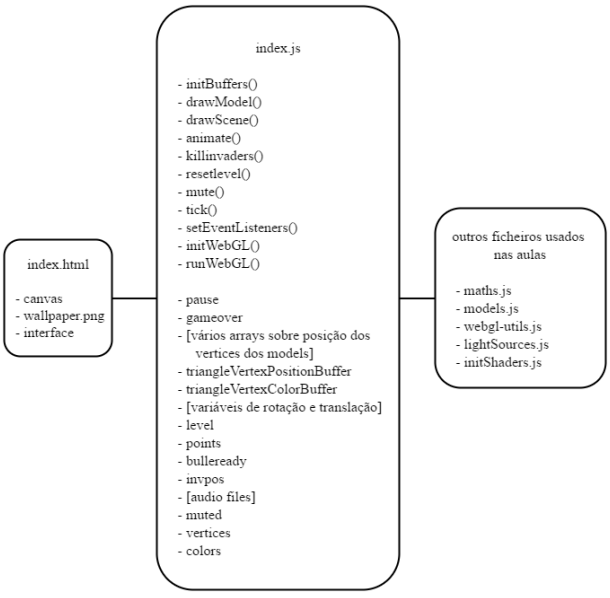

Fig. 13: Arquitetura final

III Conclusão

O principal objetivo foi conseguido, inicialmente teve-se um pouco com receio sobre o que iria ser possível ser feito e quanto tempo iria demorar a implementação do projeto. Optou-se por usar o código disponibilizado pelo professor nas aulas práticas e fazer uma reformulação eliminando algumas componentes que nos foram desnecessárias e acrescentando outras quando fosse necessário.

Os principais problemas prenderam-se em desenvolver soluções que implementassem as funcionalidades mínimas do jogo, tal como deteção de colisões e "remoção" de models. Outro grande problema foi ao nível da otimização do desempenho do programa que numa fase inicial impossibilitava a jogabilidade.

De um modo geral e como estes problemas foram superados, tendo contornado estes obstáculos que de certa forma nos eram pouco familiares ajudou ao nosso crescimento como futuros engenheiros. Foi uma tema interessante e divertido dado a nossa paixão por vídeo-jogos e foi uma oportunidade que nos permitiu desenvolver um jogo e ultrapassar as nossas expectativas e capacidades.

References

http://stackoverflow.com/
http://youtube.com/
https://www.freesound.org/