



UNIVERSIDADE DE AVEIRO

DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICAÇÕES E  
INFORMÁTICA

47053- COMPUTAÇÃO VISUAL

---

# Puzzle

Implementação em WebGL de um Puzzle

---

8240 - MESTRADO INTEGRADO EM ENGENHARIA DE  
COMPUTADORES E TELEMÁTICA

António Rafael da  
Costa Ferreira  
NMec: 67405

Rodrigo Lopes  
da Cunha  
NMec: 67800

Docente: Joaquim João Estrela Ribeiro Silvestre Madeira

Dezembro de 2015  
2015-2016

# Conteúdos

1	Introdução . . . . .	2
2	JavaScript e WebGL . . . . .	3
	2.1 Arquitectura da implementação . . . . .	3
3	Organização da implementação . . . . .	4
	3.1 Modelos . . . . .	4
	3.2 Javascript . . . . .	4
	3.3 JSON . . . . .	5
4	Interface do utilizador . . . . .	7
	4.1 Controlos do utilizador . . . . .	7
	4.2 Movimento do rato . . . . .	8
	4.3 Mesa de jogo . . . . .	8
	4.4 Help . . . . .	9
5	Instalação . . . . .	9

# 1 Introdução

O trabalho proposto para o projeto da unidade curricular de Computação Visual é um Puzzle desenvolvido em WebGL. Para o efeito foi necessário implementar um site com Bootstrap, JQuery, Json e WebGL.

Foi pensada uma implementação baseada na experiência do utilizador ao utilizar o nosso jogo, para isso desenvolveu-se uma interface atraente e com bastantes funcionalidades, tanto a nível de opções como da implementação usando WebGL. Neste relatório procurou-se focar na explicação da arquitectura desenvolvida que permite uma melhor reutilização de modelos, código e independência entre modelos desenvolvidos.

O relatório reflete todos os passos e decisões tomadas na criação da implementação, assim como uma explicação do que foi usado e da interface do utilizador.

## 2 Instalação

Para a instalar o que é necessário para o puzzle funcionar sem problemas foi usado um Simple HTTP Server que é disponibilizado pelo Python. Apenas é preciso ter o Python instalado no seu computador.

Para iniciar o servidor:

```
# ./run.sh
```

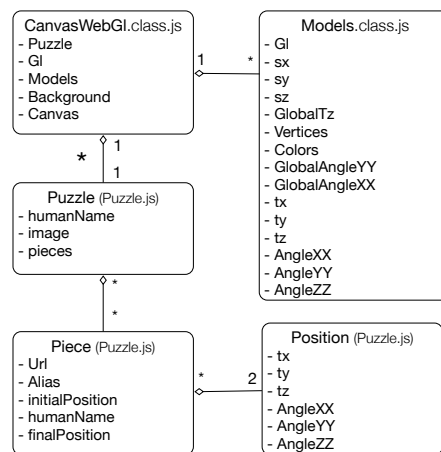
ou

```
# python -m SimpleHTTPServer 8000
```

e depois aceder a: <http://localhost:8000>

## 3 JavaScript e WebGL

### 3.1 Arquitectura da implementação



\* O CanvasWebGl apenas tem um puzzle de cada vez, mas eles são guardados num array no PuzzleGame.js o que permite que depois seja trocado o puzzle facilmente.

Figura 1: Diagrama da implementação desenvolvida

Na implementação desenvolvida procurou-se uma solução que permitisse reutilizar o código e instanciar quantas peças e puzzles fosse preciso. Para isso criou-se uma class Models, que tem como atributos os que estão descritos em cima, que instância um modelo, independente dos outros que irá fazer uso das translações, rotações e outros métodos usados durante as aulas práticas. Para isso, esta class irá, no seu construtor instanciar uma única vez dois buffers, um chamado *triangleVertexPositionBuffer* e outro *triangleVertexColorBuffer*, de resto, o *initBuffers* é sempre chamado sempre que for feito um *drawScene* para os arrays de buffers serem atualizados.

Já na class *CanvasWebGl*, é onde o puzzle instância todos os modelos (peças), aplica as translações globais e independentes, é desenhada a cena, inicializado o modelo de fundo e inicializado o WebGL.

Para alimentar a class *CanvasWebGl*, foi criada uma class *Puzzle* que tem todos os atributos necessários para instanciar um puzzle. O puzzle irá ter o seu *humanName* que será apresentado ao utilizador, a *image* que é a imagem final da solução do puzzle, e as *pieces* que são as várias peças do puzzle.

A peça, terá o URL sendo este onde será obtido a lista de vértices e de cores para a peça. O alias é o usado para identificar a peça, tem de ser único para todas as peças existentes no Puzzle, o *initialPosition* que identifica a posição inicial da peça e o *finalPosition* que identifica a posição final da figura.

## 4 Organização da implementação

### 4.1 Modelos

Foram criados vários modelos para cada puzzle, estes depois são usados para peças e estão localizados na pasta "modelos". Ests são ficheiros .txt como usados nas aulas onde armazenam a lista de vértices e de cores.

### 4.2 Javascript

Os ficheiros javascript onde está a maior parte da implementação, desde "listeners" a código de WebGL está guardado na pasta "js".

- *bootstrap.js* contem os scripts do bootstrap que são responsáveis por exemplo pelas modalbox, tooltips entre outros.
- *CanvasWebGl.class.js* como já foi explicado, contem a automatização necessária para instanciar um puzzle em WebGL.
- *confetti.js* contem o script necessário para a animação de confettis quando um utilizador termina o puzzle.
- *document.jquery.js* contem os scripts jquery necessários para a lupa quando o utilizador passa o rato por cima da pré-visualização do puzzle quando é terminado e a animação necessária para a pontuação do utilizador.
- *initShaders.js* contem os scripts necessários para inicializar cada shader program e para inicializar o shader. Este foi um aspeto importante no nosso desenvolvimento porque a percepção do que estes elementos faziam, além de outras coisas, permitiu-nos instanciar os modelos de forma independente.

- *jquery.js* é uma framework javascript que simplifica a vida ao programador, permite criar listeners de forma mais fácil e dinâmica com seletores, modificar atributos de objetos do documento HTML, fazer animações com esses objetos entre outras coisas.
- *maths.js* contem os scripts auxiliares usados nas aulas.
- *microscope.js* contem o script para instanciar a lupa na imagem.
- *Models.class.js* tal como descrito em cima, contem toda a informação necessária para instanciar uma peça de um puzzle e fazer as translações, rotações entre outras coisas do puzzle.
- *models.js* não necessitámos de olhar para este ficheiro, mas, contem as funções necessárias para processar as "triangle mesh models".
- *parseFiles.js* adaptando o código fornecido nas aulas práticas para fazer o processamento de ficheiros txt e obj, contudo a adaptação prendeu-se em fazer download via ajax de forma assíncrona dos ficheiros.
- *puzzle.js* contem as classes que permitem fazer programação ao estilo orientada aos objetos para definir o puzzle, as peças e as posições iniciais e finais.
- *PuzzleGame.js* é responsável por carregar o ficheiro puzzles.json que tem a definição dos puzzles em JSON e disponibiliza o runWebGL que irá instanciar o CanvasWebGl e irá chamar o setScreenPuzzle e o setEventListeners.
- *setEventListeners.js* contem os event listeners necessários para os controlos do puzzle e da página.
- *setScreenPuzzle.js* contem os scripts necessários para popular a página que é apresentada ao utilizador com toda a informação necessária.
- *webgl-utils.js* Copyright 2010, Google Inc, All rights reserved.

### 4.3 JSON

Para uma melhor definição de todos os puzzles usados no jogo foi criado um ficheiro JSON onde é possível criar todas as peças de cada puzzle e definir todos os atributos anteriormente detalhados. Este ficheiro é carregado inicialmente sempre que o jogo é iniciado, apresentando assim ao utilizador a lista de puzzles e inicializando o jogo com o puzzle inicial.

```
1 {
2   "puzzles" : [
3     {
4       "humanName": "Level 1",
5       "image" : "img/puzzles/puzzle1.png",
6       "pieces" : [
7         {
8           "alias": "triangulo",
9           "url": "modelos/trianguloBlue.txt",
10          "humanName": "Triangulo Blue",
11          "initialPosition" : {
12            "tx": 0.2,
13            "ty": 0.4,
14            "tz": 0.5,
15            "angleXX": 225,
16            "angleYY": 45,
17            "angleZZ": 45
18          },
19          "finalPosition" : {
20            "tx": 0,
21            "ty": 0,
22            "tz": 0,
23            "angleXX": 0,
24            "angleYY": 0,
25            "angleZZ": 0
26          }
27        }
28      ]
29    }
30  ]
31 }
```

## 5 Interface do utilizador

### 5.1 Controlos do utilizador

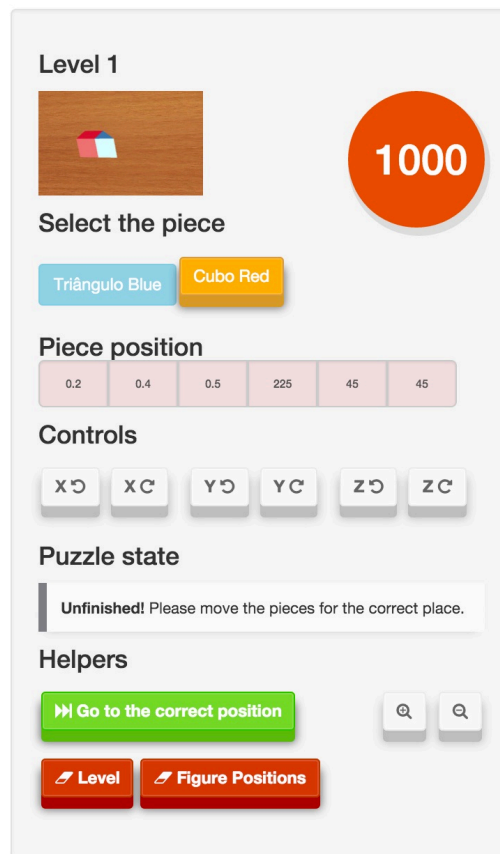


Figura 2: Web interface

A interface de utilizador dispõe de comandos que permite ao utilizador, controlar a rotação e translação em X, Y e Z de cada peça do puzzle. O "piece position" é a posição da peça no puzzle, quando está vermelho é porque a posição está errada, quando fica a verde a posição fica correta. O "puzzle state" dá um feedback sobre o estado do estado do puzzle, se está concluído ou não, também tem um "Helper" que permite ao utilizador ir para a posição correta no puzzle. Para fazer translação em Z é necessário percecionar a tecla "Z" e depois usar as arrow keys para controlar a translação.

Também tem controlos para fazer zoom in, zoom out e reset do nível e da posição da figura.

A pontuação é disponibilizada ao utilizador no círculo, que fica verde



quando o utilizador termina. Ao clicar nessa bola também é aberta uma janela de facebook para o utilizador conseguir fazer a partilha da sua pontuação. Também quando termina todos os níveis, ao fechar a animação de conclusão é aberto uma janela de facebook para partilhar no seu facebook.

Como referido anteriormente, o utilizador quando termina um nível é mostrada uma animação com confettis e uma janela de conclusão do nível.

## 5.2 Movimento do rato

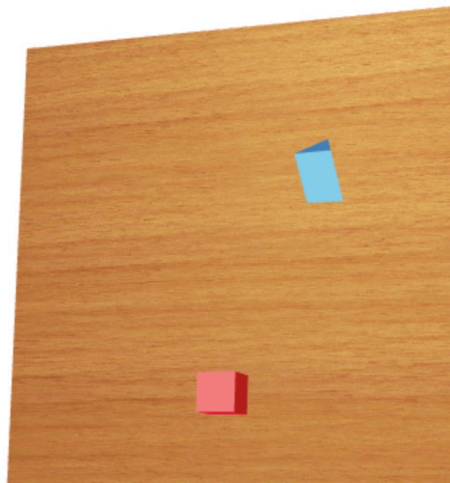


Figura 3: 3D movimento do rato

Foi implementado uma translação global em Y e X controlada pelo evento do movimento do rato quando clica no canvas.

## 5.3 Mesa de jogo

Como mostrado na figura anterior, foi implementado um modelo especial para dar o efeito de uma mesa de jogo, usando uma textura. Não é permitido ao utilizador fazer translação em Z para detrás da mesa, nem translações das peças para fora do puzzle.

## 5.4 Help

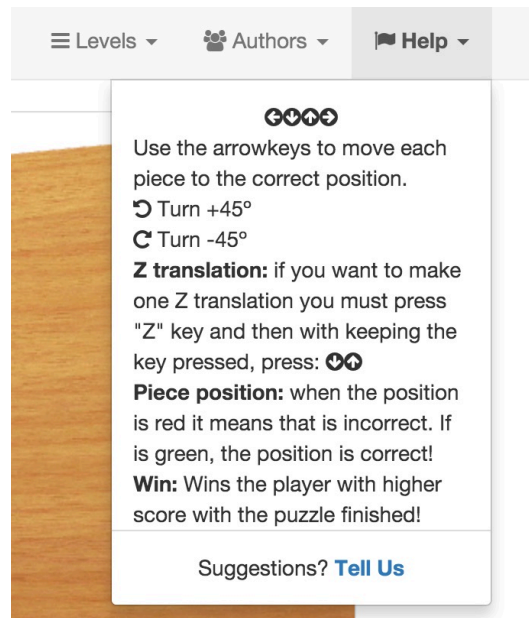


Figura 4: Help

Fizemos também um menu superior onde é dado ao utilizador todo o apoio para usar o puzzle, assim como é dada a ligação para os autores e para outros níveis.

## 6 Conclusão

O principal objetivo foi conseguido, inicialmente esteve-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 tendo como objetivo instanciar objetos independentes, com atributos diferentes, translações e rotações. Assim seria possível desenvolver um puzzle com as peças que quiséssemos e de forma simples. Esse objetivo foi conseguido, acabou-se por fazer duas class bastante gerais que poderão ser usadas para qualquer tipo de aplicação que pretenda usar WebGL e que apenas necessite de translações, rotações e outro tipo de funcionalidades.

Os principais problemas prendeu-se em desenvolver esta aproximação de modelos independentes e na criação de um modelo com textura, que serviu para fundo, contudo, o projeto acabou por exceder as nossas expetativas e foi divertido e interessante de ser realizado.

# Referências

- [1] <http://stackoverflow.com/>
- [2] <http://sweet.ua.pt/jmadeira/CV/index.html>
- [3] <http://book.mixu.net/node/ch6.html>
- [4] [http://chimera.labs.oreilly.com/books/1234000000802/ch02.html#the\\_shader](http://chimera.labs.oreilly.com/books/1234000000802/ch02.html#the_shader)
- [5] <http://bootsnipp.com/>
- [6] <http://getbootstrap.com/getting-started/>