



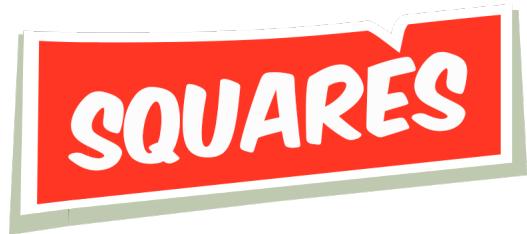
**Universidade do Minho**  
Licenciatura em Ciências da Computação

## **Unidade Curricular Projeto**

Ano Letivo de 2020/2021

### **Projeto**

**Hypatiamat –Departamento de Informática UM**  
**Jogo “Pontos”**



**André Araújo - a87987**  
**Carlos Ferreira - a87953**  
**Daniel Ribeiro - a87994**  
**Paulo Costa - a87986**

Junho, 2021

# **Projeto**

Data de Receção	
Responsável	
Avaliação	
Observações	

**André Araújo - a87987  
Carlos Ferreira - a87953  
Daniel Ribeiro - a87994  
Paulo Costa - a87986**

Junho, 2021

## **Resumo**

O seguinte relatório documenta, justifica, analisa e expõe todas as decisões tomadas ao longo do projeto "Pontos" realizado no âmbito da Unidade Curricular denominada Projeto no contexto do 3º ano do curso universitário Licenciatura em Ciências da Computação.

Ao longo deste relatório vamos apresentar as nossas ideias de forma clara e objetiva, apoiando a sua explicação através de esquemas e tabelas para uma melhor compreensão por parte do leitor.

Ao desenrolar do relatório, encontrar-se-á diversos pontos, relativos ao jogo desenvolvido, os quais tentamos que não fossem “chatos” de se ler e que abordassem as partes mais cruciais no desenvolvimento do mesmo.

Por fim, concluímos o trabalho realizado, com uma apreciação crítica do mesmo, especificando os seus pontos fortes e fracos analisando também a escalabilidade e extensão do mesmo.

# **Índice**

Resumo	1
Índice	2
1. Introdução	3
2. Menu	4
3. Variação do tamanho dos tabuleiros	10
4. Desenho de tabuleiro	12
5. FullScreen	14
6. Jogo	15
7. Bot	27
Conclusão	40
Referências	41

## 1. Introdução

Achamos por bem, inicialmente, falar do que é o Hypatiamat.

“O Hypatiamat, nomeadamente através da sua Associação Hypatiamat (AHM) tem, entre outros, o objetivo de contribuir para despertar junto dos alunos dos vários graus de ensino o gosto pela matemática e uma melhor compreensão da sua natureza; de promover o desenvolvimento do ensino da Matemática a todos os níveis; de promover a qualidade do ensino/aprendizagem da matemática mediante a utilização e integração das novas tecnologias em sala de aula, através dos recursos disponíveis na Plataforma Hypatiamat, capitalizando-se, assim, a familiaridade e o gosto dos alunos por ambientes mais tecnológicos.”

Logo desde o início, o tema em si chamou logo à atenção do grupo, não só devido ao tema que se iria desenvolver, mas também do contexto em que este jogo iria estar envolvido.

O jogo escolhido para desenvolver foi o jogo “**Pontos**”, jogo este que acompanha muitos de nós em vários momentos, seja desde distrações em aulas, papéis em restaurantes, etc...

O objetivo do jogo, para aqueles que não estão familiarizados, e de uma forma muito compacta é tentar capturar o maior número de quadrados em tabuleiros de diferentes tamanhos (5x5, 6x6, 8x8 são os abordados no desenvolvido deste projeto).

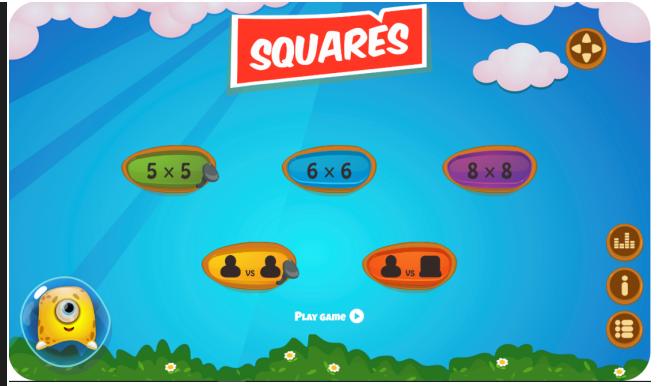
As regras também de uma forma resumida são:

- Cada jogador, simultaneamente, traça um lado de um quadrado à escolha;
- O Jogador que fechar o quadrado ganha esse quadrado;
- Ganha, quem no final, tiver mais quadrados.

## 2. Menu

Na classe menu, inicializamos com o construtor básico desta parte, onde de realçar que, começamos com o tamanho do tabuleiro de valor 5, com o modo inicial o pvp e a dificuldade inicial, como sendo o nível “fácil”. Isto porque, depois quando se inicializa o jogo, vemos que a imagem de nome “assinala” irá estar nestas posições.

```
class Menu extends Phaser.Scene {  
    constructor() {  
        super('Menu');  
        this.menu = new Map();  
        this.tamanhoTabuleiro = 5;  
        this.modo = "pvp";  
        this.dificuldade = "1";  
        this.primeiroMenu = 1;  
    }  
}
```



Depois, procedemos ao carregamento das imagens, com determinados nomes e colocamos estas nos sítios, onde era necessário que elas aparecessem na cena.

Em seguida, temos os diversos casos de interação com estas imagens, onde clicando em determinada imagem, teremos uma determinada funcionalidade, as quais passaremos a mostrar.

```
switch (gameObject.name) {  
  
    case '5x5':  
        this.menu.get('assinala_5x5').visible = true;  
        this.menu.get('assinala_6x6').visible = false;  
        this.menu.get('assinala_8x8').visible = false;  
        this.tamanhoTabuleiro = 5;  
        break;  
    case '6x6':  
        this.menu.get('assinala_5x5').visible = false;  
        this.menu.get('assinala_6x6').visible = true;  
        this.menu.get('assinala_8x8').visible = false;  
        this.tamanhoTabuleiro = 6;  
        break;  
    case '8x8':  
        this.menu.get('assinala_5x5').visible = false;  
        this.menu.get('assinala_6x6').visible = false;  
        this.menu.get('assinala_8x8').visible = true;  
        this.tamanhoTabuleiro = 8;  
        break;  
}
```

Aqui, podemos verificar que “brincamos” um pouco com a visibilidade das imagens, isto porque clicando na imagem “5x5”, iremos colocar a visibilidade dos outros tamanhos de tabuleiro a falso, de modo que fique apenas assinalado o tamanho pretendido pelo utilizador.

Isto acontece também para os casos de “6x6” e “8x8”, não colocaremos aqui no relatório para que não fique ainda mais massudo.

Outro caso que temos, é o caso de clicar num modo de jogo. No caso de ser o modo jogador contra jogador, basicamente colocamos o assinalador de que temos esse modo selecionado e colocamos tudo relacionado com o bot, com o booleano de visibilidade a falso.

```

case 'plvspl':
    this.menu.get('assinala_2pl').visible = true;
    this.menu.get('assinala_plvscp').visible = false;

    this.menu.get('level1').visible = false;
    this.menu.get('level2').visible = false;
    this.menu.get('level3').visible = false;
    this.menu.get('assinala_level1').visible = false;
    this.menu.get('assinala_level2').visible = false;
    this.menu.get('assinala_level3').visible = false;
    this.modo = "pvp";

    break;

```

Já no caso de assinalarmos o modo de jogo contra o computador, aqui já abordamos a situação de outra maneira. Aqui basicamente, fazemos com que a visibilidade das imagens que contêm os níveis de dificuldade existentes apareçam e passamos a assinalar primeiramente, o nível 1.

```

case 'plvscp':
    this.menu.get('assinala_2pl').visible = false;
    this.menu.get('assinala_plvscp').visible = true;

    this.menu.get('level1').visible = true;
    this.menu.get('level2').visible = true;
    this.menu.get('level3').visible = true;

    this.menu.get('assinala_level1').visible = true;
    this.menu.get('assinala_level2').visible = false;
    this.menu.get('assinala_level3').visible = false;
    this.modo = "pvcp";
    this.dificuldade = 1;
    t1.visible = false;
    t2.visible = false;
    t3.visible = false;
    t4.visible = false;
    break;

```

Depois, como já era de esperar temos os casos para os diversos níveis. Explicaremos como se procedeu para o caso do nível 1, pois os restantes são bastante similares.

Basicamente, assinalamos então o nível em que clicamos e colocamos a dificuldade com o valor do nível que se pretende enfrentar.

```
case 'level1':  
    this.menu.get('assinala_level1').visible = true;  
    this.menu.get('assinala_level2').visible = false;  
    this.menu.get('assinala_level3').visible = false;  
    this.dificuldade = 1;  
    t1.visible = false;  
    t2.visible = false;  
    t3.visible = false;  
    t4.visible = false;  
    break;
```

Depois, temos os 3 casos dos botões que aparecem do lado direito da cena, estes são o botão de informação (“info”), o botão das estatísticas (“stats”) e o botão dos créditos (“creditos”).

Relativamente ao de informação, o que fazemos aqui é fazer aparecer a imagem que contém as informações do jogo e o botão de fechar este quadro que foi aberto.

```
case 'info':  
    this.menu.get('texto_info').visible = true;  
    this.menu.get('texto_trofeus').visible = false;  
    this.menu.get('texto_creditos').visible = false;  
    this.menu.get('fechar').visible = true;  
    this.menu.get('fechar').setInteractive();  
    t1.visible = false;  
    t2.visible = false;  
    t3.visible = false;  
    t4.visible = false;  
    break;
```

Se quisermos abrir o quadro dos créditos, o processo é exatamente o mesmo, mas em vez de se colocar visível o quadro das informações, faz-se aparecer na cena o quadro relativamente a esta opção, que é o quadro onde aparecem os créditos pretendidos.

Mas, na parte das estatísticas abordou-se totalmente de forma diferente, isto porque foi necessário utilizar a local storage.

### O que é então o HTML Web Storage?

Com o armazenamento na web, os aplicativos da web podem armazenar dados localmente no navegador do usuário.

Antes do HTML5, os dados do aplicativo tinham que ser armazenados em cookies, incluídos em todas as solicitações do servidor. O armazenamento na Web é mais seguro e grandes quantidades de dados podem ser armazenadas localmente, sem afetar o desempenho do site.

Ao contrário dos cookies, o limite de armazenamento é muito maior (pelo menos 5 MB) e as informações nunca são transferidas para o servidor.

O armazenamento da Web é por origem (por domínio e protocolo). Todas as páginas, de uma origem, podem armazenar e acessar os mesmos dados.

Então, para usar localStorage apenas referenciamos o nome e usamos como se fosse um atributo desse objeto com localStorage.variavel.

Se não tivermos a certeza que ela existe, isto é, já está guardada (undefined), podemos testar com um if(localStorage.variavel).

Para fazer as estatísticas ao final de cada jogo, tivemos de incrementar o número de jogos totais e o número de jogo contra um determinado nível do bot.

Caso seja uma vitória também incrementamos o número de vitorias, respetivas contra o nível em concreto.

Por fim, calculamos a percentagem de vitória contra cada nível com a seguinte fórmula:  $\text{floor}((G/J) * 100)$ , onde G são os jogos ganhos contra aquela dificuldade e J os jogos totais contra essa mesma dificuldade.

```

var jogosTotais = 0;
if (localStorage.jogosTotais) {
    jogosTotais = Number(localStorage.jogosTotais);
}

var jogosVencidosN1 = 0;
var jogosVencidosN2 = 0;
var jogosVencidosN3 = 0;

if (localStorage.jogosVencidosN1) {
    jogosVencidosN1 = Math.trunc((Number(localStorage.jogosVencidosN1)/Number(localStorage.jogosTotaisN1))*100);
}

if (localStorage.jogosVencidosN2) {
    jogosVencidosN2 = Math.trunc((Number(localStorage.jogosVencidosN2)/Number(localStorage.jogosTotaisN2))*100);
}

if (localStorage.jogosVencidosN3) {
    jogosVencidosN3 = Math.trunc((Number(localStorage.jogosVencidosN3)/Number(localStorage.jogosTotaisN3))*100);
}

var t1 = this.add.text(0.475 * game.config.width, 0.54 * game.config.height, jogosTotais.toString(), style);
var t2 = this.add.text(0.575 * game.config.width, 0.64 * game.config.height, jogosVencidosN1.toString()+"%", style);
var t3 = this.add.text(0.575 * game.config.width, 0.785 * game.config.height, jogosVencidosN2.toString()+"%", style);
var t4 = this.add.text(0.575 * game.config.width, 0.77 * game.config.height, jogosVencidosN3.toString()+"%", style);

this.menu.set('t1', t1);
this.menu.set('t2', t2);
this.menu.set('t3', t3);
this.menu.set('t4', t4);

this.menu.get('t1').visible = false;
this.menu.get('t2').visible = false;
this.menu.get('t3').visible = false;
this.menu.get('t4').visible = false;

```

```

case 'stats':
    this.menu.get('texto_info').visible = false;
    this.menu.get('texto_trofeus').visible = true;
    this.menu.get('texto_creditos').visible = false;
    this.menu.get('fechar').visible = true;
    this.menu.get('fechar').setInteractive();
    this.menu.get('t1').visible = true;
    this.menu.get('t2').visible = true;
    this.menu.get('t3').visible = true;
    this.menu.get('t4').visible = true;
    break;

```

Temos ainda, o caso “fechar”, basicamente aqui colocamos todos os quadros dos casos anteriores a falso, pois este caso é abordado pelo botão que fecha tanto as opções das informações, como dos créditos e mesmo das estatísticas.

```
case 'fechar':
    this.menu.get('texto_info').visible = false;
    this.menu.get('texto_trofeus').visible = false;
    this.menu.get('texto_creditos').visible = false;
    this.menu.get('fechar').visible = false;
    this.menu.get('fechar').setInteractive(false);
    t1.visible = false;
    t2.visible = false;
    t3.visible = false;
    t4.visible = false;
    break;
```

Agora, temos o último caso, mas não menos importante, que é o botão que tem como finalidade, iniciar o jogo com o modo pretendido, isto é, com o modo que o utilizador pretende.

Aqui, utilizamos a transição (explicada mais à frente no relatório), que tem como *target*, o ecrã de jogo e por fim damos *stop*, no ecrã atual.

```
case 'play':
    this.scene.transition({
        target: 'Jogo',
        duration: 1000,
        moveBelow: true,
        onUpdate: this.transitionOut,
        data: {
            t: this.tamanhoTabuleiro,
            m: this.modo,
            d: this.dificuldade,
            j1: 0,
            j2: 0
        }
    });
    this.scene.stop('Menu');
//this.scene.start('Main' , {t : this.tamanhoTabuleiro});
    break;
```

### 3. Variação de tamanho dos tabuleiros

Bem, nesta parte tivemos de ter em conta todas as variáveis que iríamos usar e trabalhar com as escalas para que, de facto, pudéssemos continuar a ter o tabuleiro centrado com o menu de jogo e que continuasse tudo funcional, como é pretendido.

Primeiramente, tínhamos bem desenvolvido e bem enquadrado o tabuleiro de tamanho 5x5, como podemos verificar no seguinte *print*:

```
if (tamanhoTabuleiro == 5) {  
    scaleBox = 0.85;  
    imagemColuna = 0.90;  
    imagemLinha = 0.90;  
    tamanho = 2.0;  
    cheirinho = 20.0;  
    cheirinho2 = 0.0;  
    imagem = this.add.image(0.14 * game.config.width, 0.16 * game.config.height, '5x5').setScale(0.50);  
    this.imagens.push(imagem);  
    probRandom = 0.025;  
}
```

Mas, criámos as variáveis “ScaleBox”, “imagemColuna”, “imagemlinha”, “tamanho” e, finalmente, “cheirinho” para de facto as podermos utilizar, de forma que estas variassem da forma que nos fosse útil.

Sendo que, definimos utilizar estas variáveis com as seguintes variações, tanto nos tabuleiros 6x6 e 8x8:

```
if(tamanhoTabuleiro==6){  
    scaleBox = 0.75;  
    imagemColuna = 0.75;  
    imagemLinha = 0.75;  
    tamanho = 2.5;  
    cheirinho = 20.0;  
    imagem = this.add.image(0.14*game.config.width, 0.16*game.config.height, '6x6').setScale(0.50);  
    imagem = this.imagens.push(imagem);  
}  
if(tamanhoTabuleiro==8){  
    scaleBox = 0.60;  
    imagemColuna = 0.60;  
    imagemLinha = 0.60;  
    tamanho = 3.5;  
    cheirinho = 0.0;  
    imagem = this.add.image(0.14*game.config.width, 0.16*game.config.height, '8x8').setScale(0.50);  
    this.imagens.push(imagem);  
}
```

Depois de termos definido estes valores, que se foram alterando com certas tentativas e ajustes, bastava utilizá-los na criação do tabuleiro.

Nesta parte, utilizamos um ciclo for dentro de outro, para que corrésssemos todo o tamanho do tabuleiro, fazendo a criação, primeiramente, dos quadrados (imageQuadrado) com o tamanho correto, para o tamanho de tabuleiro que se quer utilizar.

```

stepBox = scaleBox*100;
for (var i = 0; i < tamanhoTabuleiro + 1; i++){
    let linhaBox = [];
    let linhaLinhas = [];
    let colunaLinhas = [];

    for (var j = 0; j < tamanhoTabuleiro + 1; j++){

        if ( j < tamanhoTabuleiro && i < tamanhoTabuleiro){
            //4.5 para 10    2.5 para 6     3.5 para 8
            var imageQuadrado = this.add.sprite(0.5*game.config.width-tamanho*stepBox + stepBox*j, yAux+stepBox*i + cheirinho,
                'square_2').setScale(scaleBox)
            linhaBox[j] = imageQuadrado;
            this.imagens.push(imageQuadrado);
        }
    }
}

```

Em seguida, após a criação dos quadrados, fazemos com que as linhas e colunas apareçam também de forma alterada, para cada um dos tamanhos de tabuleiro.

```

if (j < tamanhoTabuleiro){
    var imageLinha = this.add.sprite(0.5*game.config.width-tamanho*stepBox + stepBox*j + cheirinho2, yAux+(stepBox*i)-(stepBox/2) + cheirinho,
        'linha_horizontal_verde').setScale(imagemLinha);
    var hitbox = new Phaser.Geom.Rectangle(0,-5,100, 25);
    this.imagens.push(imageLinha);
    imageLinha.setInteractive(hitbox, Phaser.Geom.Rectangle.Contains);
    imageLinha.on('clicked', this.clickHandlerLinha, this);
    linhaLinhas[j] = imageLinha;
}

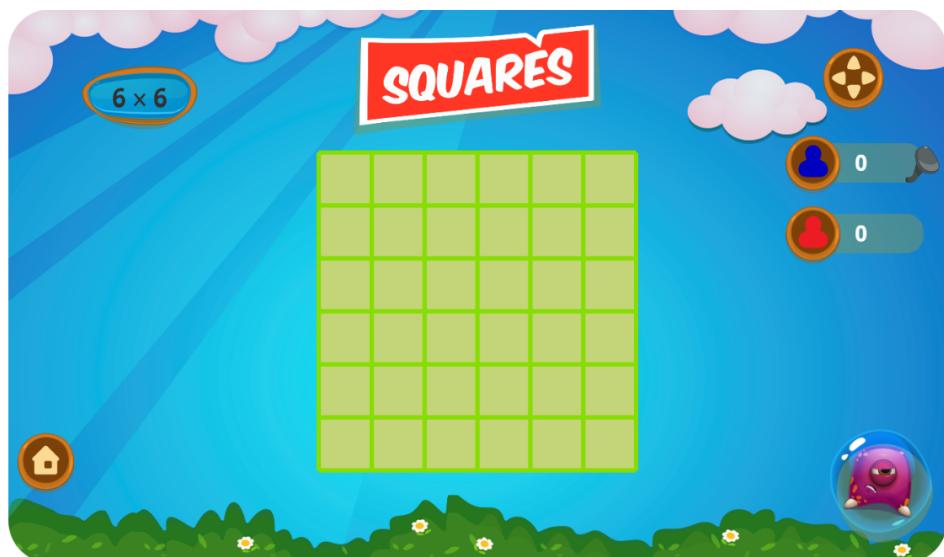
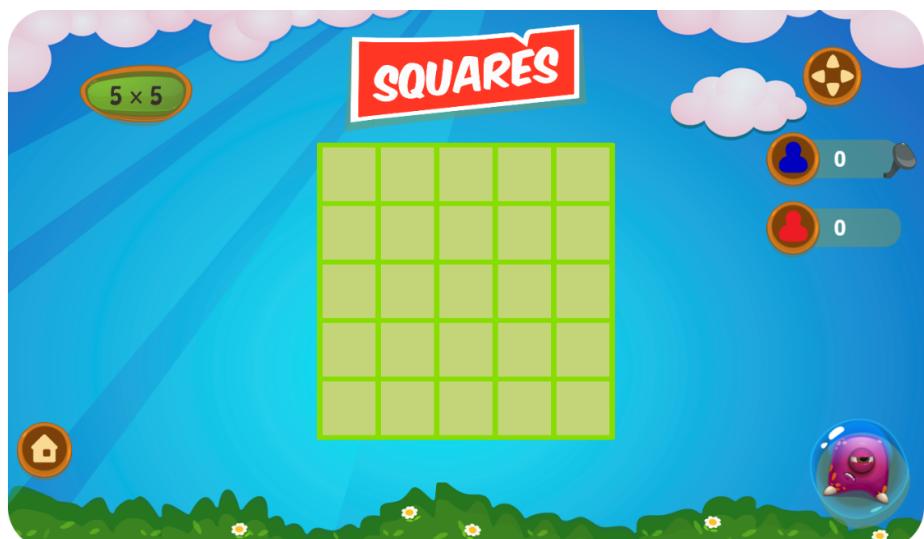
if (i < tamanhoTabuleiro){
    var imageColuna = this.add.sprite(0.5*game.config.width-tamanho*stepBox + stepBox*j-(stepBox/2), yAux+(stepBox*i) + cheirinho,
        'linha_vertical_verde').setScale(imagemColuna);
    var hitbox = new Phaser.Geom.Rectangle(-5,0,25,100);
    this.imagens.push(imageColuna);
    imageColuna.setInteractive(hitbox, Phaser.Geom.Rectangle.Contains);
    imageColuna.on('clicked', this.clickHandlerColuna, this);
    colunaLinhas[j] = imageColuna;
}

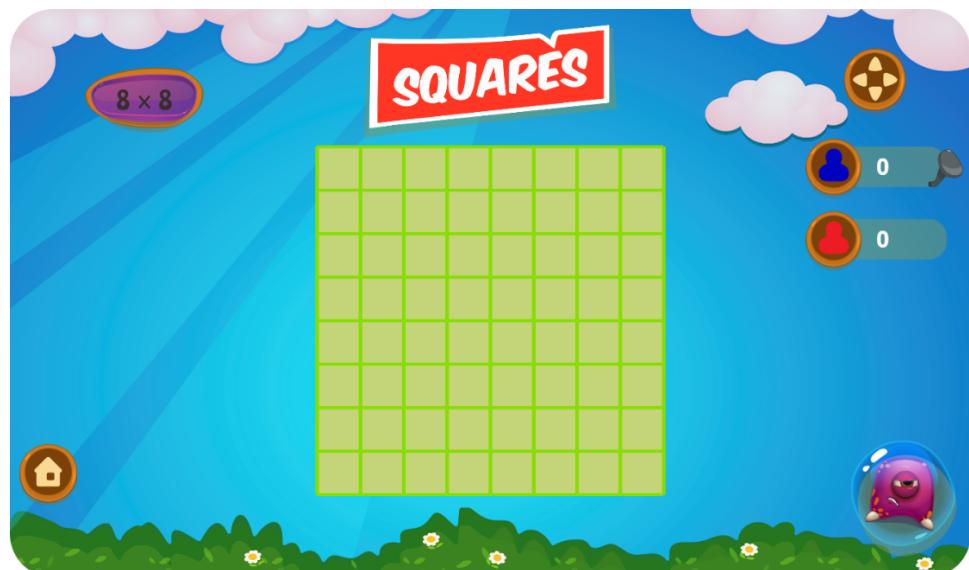
```

#### 4. Desenho do tabuleiro

Após a definição das medidas de escala procedemos ao desenho dos quadrados, das linhas e colunas do tabuleiro. Onde, dentro de dois ciclos um para percorrer as linhas e outro para as colunas, começamos por definir os quadrados e guardamos numa matriz adicionando a imagem, depois começamos a definir as linhas e colunas onde novamente adicionamos a imagem e guardamo-la numa matriz (*linhasMatrix* no caso das linhas e *colunasMatrix* no caso das colunas), por fim definimos uma *hitbox*, usada para aumentar a área iterativa da linha ou coluna, isto para que fosse mais fácil acertar na linha ou coluna.

Mostramos agora, o resultado final obtido por cada tabuleiro, tanto no tabuleiro 5x5, 6x6 e 8x8, respetivamente:





## 5. FullScreen

Colocamos esta *feature* porque era necessário que pudéssemos colocar em modo ecrã cheio o jogo, isto porque, mesmo nós preferimos quando o jogo é, de facto, jogado assim.

Começamos por verificar se estamos, na verdade, em fullscreen e dependendo iremos colocar então ou o botão relativo a tirar ou a colocar o jogo em ecrã cheio.

Usamos ainda o método que o phaser3 nos disponibiliza que é o “useHandCursor”, este faz com que o cursor se altere se estivermos com o rato em cima do botão.

```
setFullScreenButton() {
    if (!this.scale.isFullscreen) {
        this.fullScreen = this.add.image(0.90 * game.config.width, 100, 'fullScreen');
    } else {
        this.fullScreen = this.add.image(0.90 * game.config.width, 100, 'noFullScreen');
    }

    this.menu.set('fullScreen', this.fullScreen);
    this.fullScreen.setScale(0.5);

    this.fullScreen.setInteractive({
        useHandCursor: true
    });
}
```

Depois, temos a “pointerover” e a “pointerout”, que aumentam em 5 pixéis a imagem quando estamos por cima do botão, e retira a mesma quantidade quando já não estamos nessa zona, respetivamente.

E por fim, mas não menos importante, temos a “pointerup” que verificando se nos encontramos ou não no modo tela cheia, coloca-nos no modo contrário isto porque o botão foi clicado e, portanto, queremos inverter a situação onde nos encontramos.

```
this.fullScreen.on('pointerover', () => {
    this.fullScreen.displayHeight += 5;
    this.fullScreen.displayWidth += 5;
});

this.fullScreen.on('pointerout', () => {
    this.fullScreen.displayHeight -= 5;
    this.fullScreen.displayWidth -= 5;
});

this.fullScreen.on('pointerup', function() {
    if (!this.scale.isFullscreen) {
        this.fullScreen.setTexture('noFullScreen');
        this.scale.startFullscreen();
    } else {
        this.scale.stopFullscreen();
        this.fullScreen.setTexture('fullScreen');
    }
}, this);
}
```

## 6. Jogo

Agora, falando da parte que suporta o jogo em si.

```
class Jogo extends Phaser.Scene {
    constructor() {
        super('Jogo');
        this.imagens = [];
        this.pontosJogador1 = 0;
        this.pontosJogador2 = 0;
    }

    init(data) {

        console.log('init', data);
        tamanhoTabuleiro = data.t;
        modo = data.m;
        dificuldade = data.d;
        jogosGanhos1 = data.j1;
        jogosGanhos2 = data.j2;
        console.log(data.t);
        this.pontosJogador1 = 0;
        this.pontosJogador2 = 0;
        jogadorAtual = 1;

    }
}
```

Como podemos ver, primeiramente esta classe é inicializada com todas as variáveis que nos irão ser úteis ao longo do jogo e sobretudo de cada tipo de jogo que está a ser realizado, pois pode ser modo “pvp” como contra o próprio computador.

Em seguida, obviamente carregamos todas as imagens que irão ser utilizadas nas diversas cenas, que irão ter as suas próprias finalidades.

Em seguida, e com o que já foi dito anteriormente, passamos à construção do tabuleiro.

```
create() {
    jogoAcabou = false;

    imagemUltimaJogada = null;
    this.add.image(0.5 * game.config.width, 0.5 * game.config.height, 'background');

    const style = {
        font: "bold 32px Arial",
        fill: "#fff"
    };

    for (var i = 0; i < tamanhoTabuleiro; i++) {
        boxesMatrixJogadas[i] = new Array(tamanhoTabuleiro);
    }
    for (var i = 0; i < tamanhoTabuleiro + 1; i++) {
        linhasMatrixJogadas[i] = new Array(tamanhoTabuleiro);
        colunasMatrixJogadas[i] = new Array(tamanhoTabuleiro);
    }
}
```

De realçar que aqui, temos a variável “jogoAcabou” que é um booleano inicialmente falso, isto porque nos será útil na constante confirmação se o jogo terminou ou não.

Em seguida, colocamos na cena as imagens relativamente ao modo de jogo em que nos encontrámos, isto é, se estamos em “pvp” o placard de resultado com 2 jogadores, enquanto se for contra o bot, aparecerá um jogador contra um computador.

```
if (modo == "pvp") {
    imagem = this.add.image(1200, 320, 'player_2').setScale(0.45);
    this.imagens.push(imagem);
} else {
    imagem = this.add.image(1200, 320, 'computer').setScale(0.45);
    this.imagens.push(imagem);
}

var imagem = this.add.image(1200, 220, 'player_1').setScale(0.45);
this.imagens.push(imagem);
```

Depois, temos a parte em que criamos um novo placard de resultados, mas este aparece apenas quando o jogador pretende jogar de novo a partida, isto é, quando há uma acumulação de jogos contra o mesmo adversário, de modo que se possa por exemplo, realizar um “à melhor de 3”.

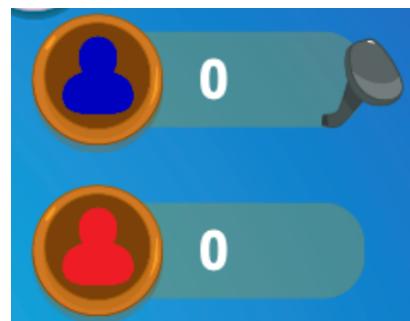
Aqui utilizámos as variáveis que contêm os jogos ganhos de cada jogador “jogosGanhos1” e “jogosGanhos2”.

```
if (jogosGanhos1 != 0 || jogosGanhos2 != 0) {  
  
    const style2 = {  
        font: "bold 24px Arial",  
        fill: "#fff"  
    };  
  
    if (modo == "pvp") {  
        imagem = this.add.image(1215, 540, 'player_2').setScale(0.35);  
        this.imagens.push(imagem);  
    } else {  
        imagem = this.add.image(1215, 540, 'computer').setScale(0.35);  
        this.imagens.push(imagem);  
    }  
  
    var imagem = this.add.image(1215, 470, 'player_1').setScale(0.35);  
    this.imagens.push(imagem);  
  
    textJogosGanhos1 = this.add.text(1215, 455, jogosGanhos1, style2);  
    textJogosGanhos2 = this.add.text(1215, 525, jogosGanhos2, style2);  
    this.imagens.push(textJogosGanhos1);  
    this.imagens.push(textJogosGanhos2);  
}
```

Decidimos ainda, colocar a pequena *feature*, em que basicamente aparece uma imagem que assinala qual o próximo jogador a jogar.

```
atualJogador = this.add.sprite(1296, 220, 'assinala').setScale(0.55);
this.imagens.push(atualJogador);
atualJogador.visible = true;

atualJogador2 = this.add.sprite(1296, 320, 'assinala').setScale(0.55);
this.imagens.push(atualJogador2);
atualJogador2.visible = false;
```



Depois, temos os diferentes casos e finalidades de cada botão presente na cena de jogo, os quais passaremos a mostrar individualmente.

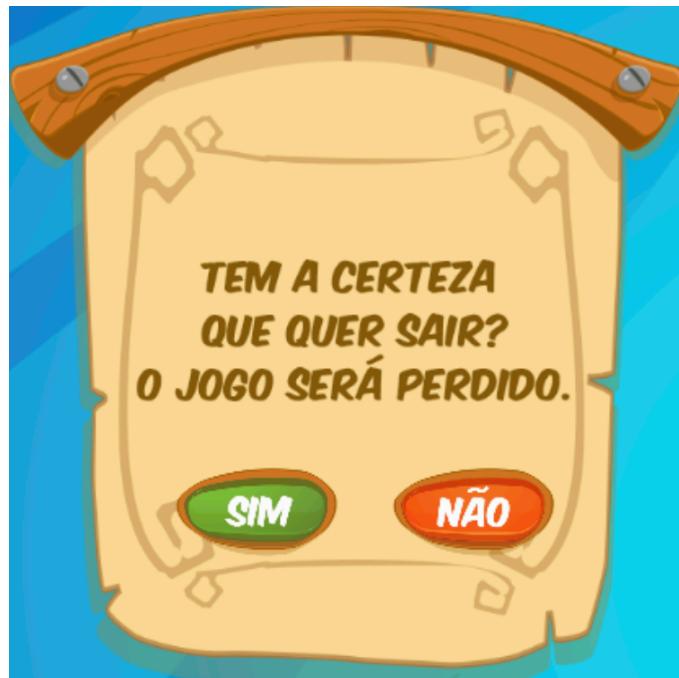
```
switch (gameObject.name) {

    case 'casinha':
        this.mudaInteractiveFalse();
        imagemSair.visible = true;
        imagemSim.visible = true;
        imagemSim.setInteractive();
        imagemNao.visible = true;
        imagemNao.setInteractive();

    break;
```



Onde, este caso acaba por abrir no seguinte quadro:



E aqui, temos a abertura para mais dois casos, com diferentes finalidades que são o “SIM” e o “NÃO”.

```
case 'Sim':
    imagemSair.visible = false;
    imagemSim.visible = false;
    imagemNao.visible = false;
    this.scene.transition({
        target: 'Menu',
        duration: 1000,
        moveBelow: true,
        onUpdate: this.transitionOut,
        data: {
            t: tamanhoTabuleiro,
            m: modo,
            d: dificuldade
        }
    });
    this.scene.stop('Jogo');
    //this.scene.start('Main' , {t : this.tamanhoTabuleiro});
    break;

case 'Nao':
    imagemSair.visible = false;
    imagemSim.visible = false;
    imagemSim.setInteractive(false);
    imagemNao.visible = false;
    imagemNao.setInteractive(false);
    mudaInteractiveTrue();
    jogoAcabou = false;
    this.fimDeJogo();
    break;
```

Basicamente, o primeiro faz com que se passe para a cena “Menu” com uma determinada transição, transição esta que vamos explicar em frente. Já a do lado direto, mantém a situação de jogo em que nos encontrávamos e coloca as imagens como não visíveis.

Depois temos ainda o caso de jogar de novo, este aparece quando acontece o final de um jogo, isto é, quando a função “fimDeJogo”, a qual explicaremos em seguida também, deteta que o jogo já terminou e então faz com que seja possível voltar ao menu, ou então, jogar de novo.

```

    case 'PlayAgain':
        this.registry.destroy();
        this.events.off();
        //this.scene.restart();
        this.scene.start('Jogo', {
            t: tamanhoTabuleiro,
            m: modo,
            d: dificuldade,
            j1: jogosGanhos1,
            j2: jogosGanhos2
        });
        break;

```

Explicaremos agora então, as **transições** que são utilizadas.

De forma a colocar o jogo mais “polido” e bonito adicionamos transições entre o menu e o jogo.

As transições implementadas são de um estilo “deslizante”, quando passamos do menu para o jogo, todo o nosso ecrã desliza da direita para a esquerda e se estivermos a passar do jogo para o menu o deslize é contrário.

Para a implementação de uma transição deste tipo, precisamos de considerar no Phaser a transição de saída e a transição de entrada em ambas as “Scene’s”, começando pela transição do menu para o jogo, na qual desliza da direita para a esquerda.

Começamos por criar uma “transitionOut” no menu e uma “transitionstart” no jogo.

Sendo a “transitionOut” chamada quando saímos do menu e a “transitionstart” quando entramos no jogo.

```

transitionOut(progress) {
    progress = progress / 9;

    this.menu.forEach((values, keys) => {
        values.x = values.x - progress * (game.config.width / 4);
    })
}

```

```

this.events.on('transitionstart', function(fromScene, duration) {
    var targetsX = []
    this.imagens.forEach((item, index) => {
        item.x += game.config.width;
        targetsX.push(item)
    });

    this.tweens.add({
        delay: 1000,
        targets: targetsX,
        duration: 1000,
        x: '-=' + game.config.width,
        ease: 'Power2'
    });

}, this);

```

A “transitionOut” vai arrastar todas as imagens do ecrã para a esquerda até as mesmas saírem do ecrã, para isso diminuímos a coordenada X das imagens e textos enquanto a animação decorre.

Na “transitionstart” começamos por colocar todas as imagens e textos para a direita um valor fixo, para elas começarem fora do ecrã sem serem vistas, após isto vamos então começar a puxar as mesmas para a esquerda, diminuindo a coordenada X ao longo do tempo até as imagens chegarem á posição original.

Para a transição contrária a ideia é a mesma, mas envez de diminuir o X ao longo do tempo, nós vamos aumentar-lo para dar a sensação de arraste para a direita, agora também vamos ter a “transitionOut” no jogo e a “transitionstart” no menu.

```
transitionOut(progress) {  
    progress = progress / 9;  
  
    this.imagens.forEach((values, keys) => {  
        values.x = values.x + progress * (game.config.width / 4);  
    })  
  
    if (jogoAcabou) {  
        t1.x = t1.x + progress * (game.config.width / 4);  
        t2.x = t2.x + progress * (game.config.width / 4);  
        t3.x = t3.x + progress * (game.config.width / 4);  
        t4.x = t4.x + progress * (game.config.width / 4);  
    }  
}
```

```
this.events.on('transitionstart', function(fromScene, duration) {  
  
    var targetsX = []  
    this.menu.forEach((item, index) => {  
        item.x -= game.config.width;  
        targetsX.push(item)  
    });  
  
    this.tweens.add({  
        delay: 1000,  
        targets: targetsX,  
        durantion: 1000,  
        x: '+=' + game.config.width,  
        ease: 'Power2'  
    });  
}, this);
```

Em relação à função que nos permite verificar se no final de uma jogada o jogo terminou, a qual denominamos de “**fimDeJogo**”, esta foi implementada da seguinte maneira.

Na função fim de jogo começamos por ver se temos jogadas livres e verificamos o caso em que estas estejam todas ocupadas. Se as jogadas estiverem ocupadas, então o jogo termina e apresentamos um quadro com dois botões, um para voltar a jogar onde clicamos para voltar a jogar e outro botão para voltar para o menu principal, onde também apresentamos o vencedor e o resultado do jogo bem como aumentamos internamente o número de jogos ganhos pelo jogador. Como podemos verificar nos seguintes prints:

```
fimDeJogo() {
    if (!jogoAcabou) {

        var f = 1;
        var fim= Math.floor((tamanhoTabuleiro*tamanhoTabuleiro)/2);
        if (PontosJogador1>fim || PontosJogador2>fim){
            f=0;
        }

        if (f == 0){
            imagemAmpulheta.visible = false;
            mudaInteractiveFalse();
            imagemCasinha.setInteractive(true);
            imagemCasinha.input.enabled = true;
            jogoAcabou = true;
            imagemFimJogo.visible = true;
            imagemPlayAgain.visible = true;
            imagemPlayAgain.setInteractive();
```

```
if(modos == "pvcp"){
    if (localStorage.jogosTotais) {
        localStorage.jogosTotais = Number(localStorage.jogosTotais) + 1;
    } else {
        localStorage.jogosTotais = 1;      }

    if(dificuldade == 1){
        if (localStorage.jogosTotaisN1) {
            localStorage.jogosTotaisN1 = Number(localStorage.jogosTotaisN1) + 1;
        } else {
            localStorage.jogosTotaisN1 = 1;      }
    }
    if (dificuldade == 2){
        if (localStorage.jogosTotaisN2) {
            localStorage.jogosTotaisN2 = Number(localStorage.jogosTotaisN2) + 1;
        } else {
            localStorage.jogosTotaisN2 = 1;      }
    }
    if (dificuldade == 3){
        if (localStorage.jogosTotaisN3) {
            localStorage.jogosTotaisN3 = Number(localStorage.jogosTotaisN3) + 1;
        } else {
            localStorage.jogosTotaisN3 = 1;      }
    }
}
```

```

    if (pontosJogador1 > pontosJogador2) {
        if(dificuldade == 1){
            if (localStorage.jogosVencidosN1) {
                localStorage.jogosVencidosN1 = Number(localStorage.jogosVencidosN1) + 1;
            } else {
                localStorage.jogosVencidosN1 = 1;
            }
        }
        if (dificuldade == 2){
            if (localStorage.jogosVencidosN2) {
                localStorage.jogosVencidosN2 = Number(localStorage.jogosVencidosN2) + 1;
            } else {
                localStorage.jogosVencidosN2 = 1;
            }
        }
        if (dificuldade == 3){
            if (localStorage.jogosVencidosN3) {
                localStorage.jogosVencidosN3 = Number(localStorage.jogosVencidosN3) + 1;
            } else {
                localStorage.jogosVencidosN3 = 1;
            }
        }
    }
}

```

Nestes dois últimos, verificamos que se efetua uma salvaguarda interna, tanto do número de jogos efetuados, bem como do número de jogos vencidos, isto para que depois na cena “Menu”, nas estatísticas se possa aceder a estes valores e, posteriormente, mostrar ao utilizador como se encontra o seu estado de jogo, isto é, para ver se tem uma boa percentagem de vitória ou não.

Agora, achamos por bem mostrar a função que nos permite somar e guardar os pontos de cada jogador durante o jogo, esta é chamada na função “jogada” a qual explicaremos em frente.

```

aumentaPontos() {
    if (jogadorAtual == 1) {
        this.pontosJogador1++;
        text1.setText(this.pontosJogador1.toString());
    } else {
        this.pontosJogador2++;
        text2.setText(this.pontosJogador2.toString());
    }
}

```

Como podemos ver, esta sabendo que o jogador atual é 1 ou 2, incrementa o seu número de pontos e altera o resultado do mesmo, isto porque esta função é apenas chamada em caso de pontuação, ou seja, no caso em que se fecha um quadrado.

Passaremos então a mostrar agora as funções que tornam possível efetuar, de facto, uma jogada neste jogo.

Temos o ponto de partida nas funções “clickHandlerLinha” e “clickHandlerColuna”, onde tal como o nome indica, assumem o ponto da cena onde se efetuou a jogada e verificando a localização do clique, chama ou a função “jogarLinha” ou “jogarColuna”, caso se tenha jogado numa linha ou numa coluna, respetivamente.

```
jogarLinha(i, j, linha) {
    if (jogadorAtual == 1) {
        if (modo == "pvcp" && imagemUltimaJogada != null) {
            if (orientacaoUltimaJogada == 'c') {
                imagemUltimaJogada.setTexture('linha_vertical_vermelha');
            } else {
                imagemUltimaJogada.setTexture('linha_horizontal_vermelha');
            }
            linha.setTexture('linha_horizontal_azul');
        }
    } else {
        orientacaoUltimaJogada = 'l';
        if (modo == "pvcp") {
            linha.setTexture('linha_ultima_jogada_h');
            imagemUltimaJogada = linha;
        } else {
            linha.setTexture('linha_horizontal_vermelha');
        }
    }

    linhasMatrixJogadas[i][j] = jogadorAtual;
    this.jogada('l', i, j);
    if (Pontuou == 0) {
        if (jogadorAtual == 1) {
            jogadorAtual = 2;
            atualJogador.visible = false;
            atualJogador2.visible = true;
        } else {
            jogadorAtual = 1;
            atualJogador2.visible = false;
            atualJogador.visible = true;
        }
    }
    else {
        Pontuou = 0;
    }
}
```

Basicamente e como se pode verificar, apenas efetuamos a colocação das linhas de cor vermelha ou azul, isto dependendo do jogador em questão, seguido de uma chamada a outra função “jogada”, a qual nos dirá se houve o fecho de um quadrado e se estamos perante um final de jogo, com a jogada efetuada.

## A função jogada:

```
jogada(t, i, j) {
    if (t == 'L') {
        if (i > 0 && linhasMatrixJogadas[i - 1][j] != null && colunasMatrixJogadas[i - 1][j] != null && colunasMatrixJogadas[i - 1][j + 1] != null) {
            console.log("fechou quadrado")
            imagemUltimaJogada = null
            boxesMatrixJogadas[i - 1][j] = jogadorAtual;
            //jogadorAtual = jogadorAtual==1 ? 2 : 1;
            if (jogadorAtual == 1) {
                boxesMatrix[i - 1][j].setTexture('square_2_blue')
            } else {
                boxesMatrix[i - 1][j].setTexture('square_2_red')
            }

            linhasMatrix[i][j].setTexture('linha_horizontal_preta')
            linhasMatrix[i - 1][j].setTexture('linha_horizontal_preta')
            colunasMatrix[i - 1][j].setTexture('linha_vertical_preta')
            colunasMatrix[i - 1][j + 1].setTexture('linha_vertical_preta')
            pontuou = 1;
            aumentaPontos();
            this.fimDeJogo();
        }
        if (i < tamanhoTabuleiro && linhasMatrixJogadas[i + 1][j] != null && colunasMatrixJogadas[i][j] != null && colunasMatrixJogadas[i][j + 1] != null) {
            console.log("fechou quadrado")
            imagemUltimaJogada = null
            boxesMatrixJogadas[i][j] = jogadorAtual;
            //jogadorAtual = jogadorAtual==1 ? 2 : 1;
            if (jogadorAtual == 1) {
                boxesMatrix[i][j].setTexture('square_2_blue')
            } else {
                boxesMatrix[i][j].setTexture('square_2_red')
            }

            else {
                if (j > 0 && colunasMatrixJogadas[i][j - 1] != null && linhasMatrixJogadas[i][j - 1] != null && linhasMatrixJogadas[i + 1][j - 1] != null) {
                    console.log("fechou quadrado")
                    imagemUltimaJogada = null
                    boxesMatrixJogadas[i][j - 1] = jogadorAtual;
                    //jogadorAtual = jogadorAtual==1 ? 2 : 1;
                    if (jogadorAtual == 1) {
                        boxesMatrix[i][j - 1].setTexture('square_2_blue')
                    } else {
                        boxesMatrix[i][j - 1].setTexture('square_2_red')
                    }

                    linhasMatrix[i][j - 1].setTexture('linha_horizontal_preta')
                    linhasMatrix[i + 1][j - 1].setTexture('linha_horizontal_preta')
                    colunasMatrix[i][j - 1].setTexture('linha_vertical_preta')
                    colunasMatrix[i][j].setTexture('linha_vertical_preta')
                    pontuou = 1;
                    aumentaPontos();
                    this.fimDeJogo();
                }
                if (j < tamanhoTabuleiro && colunasMatrixJogadas[i][j + 1] != null && linhasMatrixJogadas[i][j] != null && linhasMatrixJogadas[i + 1][j] != null) {
                    console.log("fechou quadrado")
                    imagemUltimaJogada = null
                    boxesMatrixJogadas[i][j] = jogadorAtual;
                    //jogadorAtual = jogadorAtual==1 ? 2 : 1;
                    if (jogadorAtual == 1) {
                        boxesMatrix[i][j].setTexture('square_2_blue')
                    } else {
                        boxesMatrix[i][j].setTexture('square_2_red')
                    }

                    linhasMatrix[i][j].setTexture('linha_horizontal_preta')
                    linhasMatrix[i + 1][j].setTexture('linha_horizontal_preta')
                    colunasMatrix[i][j].setTexture('linha_vertical_preta')
                    colunasMatrix[i][j + 1].setTexture('linha_vertical_preta')
                    pontuou = 1;
                    aumentaPontos();
                    this.fimDeJogo();
                }
            }
        }
    }
}
```

Aqui, podemos então verificar que a forma como se abordou esta função é bastante simples, basicamente começa-se por conformar se a jogada é numa linha ou numa coluna, utilizando para isto a variável *t*, que pode ter valores ‘l’ e ‘c’ que corresponde a linha e coluna, respetivamente.

Depois, verifica-se se estamos perante uma jogada que fecha um quadrado ou não, isto para que se altere as linhas e colunas, que envolvem esse quadrado com a cor preta, para que se aumente os pontos do jogador atual, para que se teste se estamos perante o final de jogo e ainda, importante de destacar, alterar a variável “pontuou” para 1, para que depois na função “jogarColuna” ou “jogarLinha”, se possa manter o mesmo jogador com a posse da jogada, pois este fechou um quadrado, ao invés de trocar de jogador, como habitual.

## 7. Bot

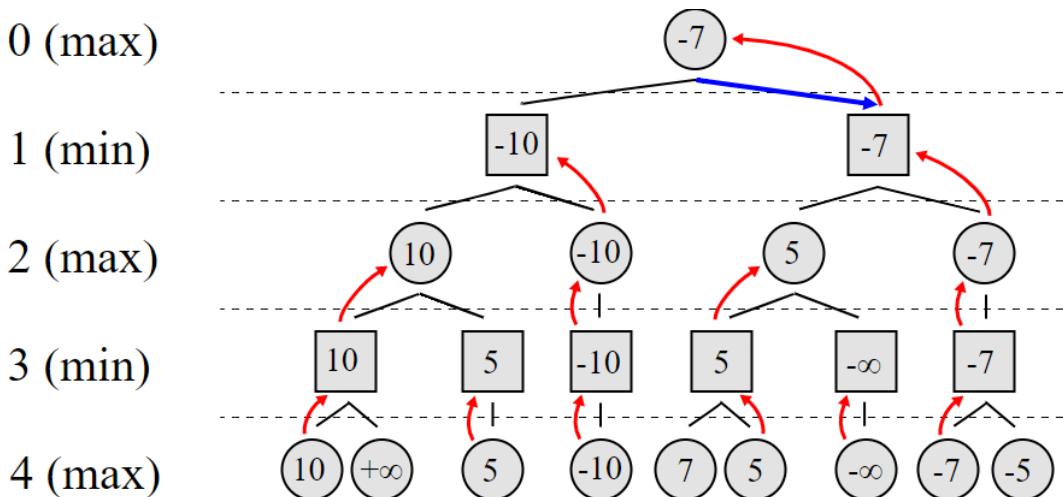
O Bot foi sem dúvida a parte mais complexa e desafiadora do Projeto e também a que nos deu mais problemas no caminho tendo exigido de nós vários testes, debugging e soluções inteligentes para colocar o mesmo mais eficiente e inteligente.

A base para o nosso Bot é o algoritmo **MinMax**, este é um algoritmo de força-bruta recursivo que procura minimizar as perdas (Min) e maximizar os ganhos (Max).

Este algoritmo vai percorrer todas as alternativas de jogada possíveis, simulando as mesmas num novo “clone” do estado de jogo, e em cada uma destas simulações volta a simular novamente, recursivamente até a uma certa profundidade.

Quanto maior a profundidade mais à frente o Bot vai conseguir prever e uma melhor decisão será tomada por parte do mesmo. Mas esta profundidade não pode ser exagerada visto que a cada vez que é aumentada, o número de simulações cresce exponencialmente.

Vamos tomar como exemplo a seguinte árvore:



Esta árvore foi feita imaginando uma situação onde cada jogador tem só 2 jogadas possíveis, onde os números positivos representam vantagem para o jogador maximizante e os negativos para o minimizante, desta forma o infinito é quando o maximizante ganha e acaba o jogo e o -infinito quando o minimizante ganha e acaba o jogo.

Para construir a árvore recursiva começamos por partir do estado original e vamos testar todas as jogadas válidas e para essas jogadas válidas vamos testar outra vez todas as jogadas válidas, quando fizermos isso 4 vezes vamos obter a linha número 4 onde cada número representa a avaliação dada aquela linha.

Agora para determinar a melhor jogada, vamos no nível 3 escolher os mínimos do nível 4 visto que é o minimizante a jogar, na 2 vamos escolher os máximos, na 1 os mínimos e raiz os máximos.

Seguindo esta sequência conseguimos saber o movimento que o jogador deve fazer para minimizar as perdas e maximizar os ganhos.

Com a lógica explicada passamos então á implementação do mesmo.

Começando com um pequeno pseudocódigo:

```
function minimax(node, depth, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the utility of the node

    if maximizingPlayer
        bestValue := ???
        for each child of node
            v := minimax(child, depth ? 1, FALSE)
            bestValue := max(bestValue, v)
        return bestValue

    else (* minimizing player *)
        bestValue := +?
        for each child of node
            v := minimax(child, depth ? 1, TRUE)
            bestValue := min(bestValue, v)
        return bestValue
```

Usando este pseudocódigo como base vamos construir o nosso algoritmo para este jogo em concreto.

Precisamos de uma estrutura para guardar os dados do jogo atual para isso usamos a variável “estado” onde é guardado as informações do jogo sobre quem têm quais linhas, colunas ou quadrados e quem é o jogador atual.

Também temos a “jogadaBotStruct” com informações relevantes ao bot que serão abordadas mais á frente

```
var estado = {
    jogadorAtual: 0,
    linhasMatrixJogadas: [],
    colunasMatrixJogadas: [],
    boxesMatrixJogadas: []
}
```

```
var jogadaBotStruct = {
    modo: '',
    iMelhor: 0,
    jMelhor: 0,
    caminho:[],
    valor:-2000
}
```

Começando pelas funções auxiliares precisamos de principalmente:

- Detetar fim de jogo
- Avaliar o estado de jogo
- Validar jogada
- Simular jogada
- Clonar o estado
- ...

Para detetar o fim de jogo apenas temos de verificar se todos os quadrados estão capturados

```
function testeFimDeJogo() {
    var f = 1;
    var fim= Math.floor((tamanhoTabuleiro*tamanhoTabuleiro)/2);
    if (pontosJogador1>fim || pontosJogador2>fim){
        f=0;
    }
    return !f;
}
```

A função para avaliar o estado do jogo, tem como objetivo dar uma avaliação ao tabuleiro dizendo se este é um estado favorável para o Bot(avaliação positiva) ou para o Jogador(avaliação negativa) para isto contamos o número de quadrados do bot subtraindo o número de quadrados do jogador.

```
function avaliacaoTabuleiro(e) {
    var p1 = 0
    var p2 = 0
    for (let i = 0; i < tamanhoTabuleiro; i++) {
        for (let j = 0; j < tamanhoTabuleiro; j++) {
            if (e.boxesMatrixJogadas[i][j] == 1) {
                p1++;
            } else if (e.boxesMatrixJogadas[i][j] == 2) {
                p2++;
            }
        }
    }
    return p2 - p1
}
```

Para validar uma jogada verificamos se aquela linha não está preenchida

```
function validaJogadaColuna(e,i,j){  
    if (i == tamanhoTabuleiro){  
        return false  
    }  
    return (e.colunasMatrixJogadas[i][j] == null)  
}  
  
function validaJogadaLinha(e,i,j){  
    if (j == tamanhoTabuleiro){  
        return false  
    }  
    return (e.linhasMatrixJogadas[i][j] == null)  
}
```

Para simular uma jogada é semelhante á jogar normal mas apenas altera o estado recebido e não faz nenhuma alteração gráfica.

```
function simulaJogadaLinha(e,i,j){  
  
    var f1 = 0;  
    e.linhasMatrixJogadas[i][j] = e.jogadorAtual  
  
    if(i>0 && e.linhasMatrixJogadas[i-1][j]!=null && e.colunasMatrixJogadas[i-1][j]!=null&& e.colunasMatrixJogadas[i-1][j+1]!=null){  
        e.boxesMatrixJogadas[i-1][j]=e.jogadorAtual;  
        f1 += 1;  
    }  
    if(i<tamanhoTabuleiro && e.linhasMatrixJogadas[i+1][j]!=null && e.colunasMatrixJogadas[i][j]!=null && e.colunasMatrixJogadas[i][j+1]!=null){  
        e.boxesMatrixJogadas[i][j]=e.jogadorAtual;  
        f1 += 1;  
    }  
    else if(f1 == 0){  
        e.jogadorAtual = e.jogadorAtual == 1 ? 2 : 1  
    }  
  
    return e  
}  
  
function simulaJogadaColuna(e,i,j){  
  
    var f1 = 0;  
    e.colunasMatrixJogadas[i][j] = e.jogadorAtual  
  
    if(j>0 && e.colunasMatrixJogadas[i][j-1]!=null && e.linhasMatrixJogadas[i][j-1]!=null && e.linhasMatrixJogadas[i+1][j-1]!=null){  
        e.boxesMatrixJogadas[i][j-1]=e.jogadorAtual;  
        f1 += 1;  
    }  
    if(j<tamanhoTabuleiro && e.colunasMatrixJogadas[i][j+1]!=null && e.linhasMatrixJogadas[i][j]!=null && e.linhasMatrixJogadas[i+1][j]!=null){  
        e.boxesMatrixJogadas[i][j]=e.jogadorAtual;  
        f1 +=1;  
    }  
    else if (f1 == 0 ){  
        e.jogadorAtual = e.jogadorAtual == 1 ? 2 : 1  
    }  
  
    return e  
}
```

Para clonar o estado primeiro começamos por tentar usar um `JSON.stringify()` após um `JSON.parse()` mas como o clone era feito muitas vezes esta é uma solução lenta em termos de performance apesar de ser a mais fácil de usar, visto isto tivemos de implementar uma função para o fazer o clone do estado.

```
function deepClone(estado){

    var len1 = estado.linhasMatrixJogadas.length, copylinhasMatrixJogadas = new Array(len1);
    for (let i=0; i<len1; ++i)
        copylinhasMatrixJogadas[i] = estado.linhasMatrixJogadas[i].slice(0);

    var len2 = estado.colunasMatrixJogadas.length, copycolunasMatrixJogadas = new Array(len2);
    for (let i=0; i<len2; ++i)
        copycolunasMatrixJogadas[i] = estado.colunasMatrixJogadas[i].slice(0);

    var len3 = estado.boxesMatrixJogadas.length, copyboxesMatrixJogadas = new Array(len3);
    for (let i=0; i<len3; ++i)
        copyboxesMatrixJogadas[i] = estado.boxesMatrixJogadas[i].slice(0);

    var r = {
        jogadorAtual: estado.jogadorAtual,
        linhasMatrixJogadas: copylinhasMatrixJogadas,
        colunasMatrixJogadas: copycolunasMatrixJogadas,
        boxesMatrixJogadas: copyboxesMatrixJogadas
    };
    return r;
}
```

Com todos estes ingredientes, podemos começar a montar a nossa função minimax a partir do pseudocódigo.

O cabeçalho da nossa função é o seguinte:

```
function minimax(nodo, profundidade, alpha, beta, mudarMelhorJogada , caminho , iteracoes)
```

Sendo os argumentos:

- O nodo atual ou estado atual que estamos a analizar;
- A profundidade da pesquisa;
- O alpha e o beta é para um “pruning” que fazemos na função para melhorar a performance serão explicados mais á frente;
- Uma flag mudarMelhorJogada;
- O caminho que percorremos atualmente útil para as cadeias;
- E as iterações que funciona quase como a profundidade mas no nosso caso uma iteração é um turno completo de um jogador enquanto que a profundidade é só uma jogada.

Inicialmente vamos avaliar os casos onde termina a função e damos return à avaliação daquele nodo e do caminho.

```
if ( this.nodoTerminal(nodo) && (profundidade == profundidadeMaxima)){
    fimEncontradoaux = 1;
    chain = caminho
    return [this.avaliacaoTabuleiro(nodo) , caminho];
}

if (profundidade == 0 || this.nodoTerminal(nodo) || (this.jogadasValidasColunas(nodo) == 0 && this.jogadasValidasLinhas(nodo) == 0)){
    chain = caminho
    return [this.avaliacaoTabuleiro(nodo) , caminho];
}
```

Após os casos base, passamos para o do caso de se for o jogador maximizante a jogar de seguida o jogador maximizante é o jogador 2 (Bot).

Iniciamos o valorMax com o pior valor para o maximizante, assim escolhe logo uma jogada, escolhemos á sorte se vamos começar a avaliar linhas ou colunas para as jogadas se valerem o mesmo serem balanceadas e colocamos a lista de jogadas possíveis para linhas e colunas no prioL e no prioC.

Esta lista de jogadas é ordenada pela sua prioridade que é quanto maior se for uma jogada que capture e tanto menor se perde quadrados.

Ao ordenarmos desta forma podemos otimizar a velocidade do algoritmo quando formos dar pruning na árvore.

```
var jogadorMaximizante = nodo.jogadorAtual == 2 ? true : false
if (jogadorMaximizante){

    var valorMax = Number.NEGATIVE_INFINITY

    if(Math.random() < 0.5{
        var aux = ['l','c']
    }
    else{
        var aux = ['c','l']
    }

    var prioL = this.prioridadeLinha(nodo);
    var prioC = this.prioridadeColuna(nodo);
```

```

function capturasJogadaLinha(e,i,j){

    var r = 0
    if( i != tamanhoTabuleiro &&
        e.colunasMatrixJogadas[i][j] != null &&
        e.colunasMatrixJogadas[i][j+1] != null &&
        e.linhasMatrixJogadas[i+1][j] != null){
        r+=1
    }
    else if (i != tamanhoTabuleiro &&
            ((e.colunasMatrixJogadas[i][j] != null && e.colunasMatrixJogadas[i][j+1] != null) ||
            (e.colunasMatrixJogadas[i][j] != null && e.linhasMatrixJogadas[i+1][j] != null) ||
            (e.colunasMatrixJogadas[i][j+1] != null && e.linhasMatrixJogadas[i+1][j] != null))){
        r-=1
    }

    if( i != 0 &&
        e.colunasMatrixJogadas[i-1][j] != null &&
        e.colunasMatrixJogadas[i-1][j+1] != null &&
        e.linhasMatrixJogadas[i-1][j] != null){
        r+=1
    }
    else if(i != 0 &&
            ((e.colunasMatrixJogadas[i-1][j] != null && e.colunasMatrixJogadas[i-1][j+1] != null) ||
            (e.colunasMatrixJogadas[i-1][j] != null && e.linhasMatrixJogadas[i-1][j] != null) ||
            (e.colunasMatrixJogadas[i-1][j+1] != null && e.linhasMatrixJogadas[i-1][j] != null))){
        r-=1
    }
    return r
}

```

```

function prioridadeLinha(e){

    var lista = []
    var listasorted = []

    for (var i = 0; i < tamanhoTabuleiro + 1; i++){
        for (var j = 0; j < tamanhoTabuleiro; j++){
            if(validaJogadaLinha(e,i,j)){
                var aux = [i,j,capturasJogadaLinha(e,i,j)]
                lista.push(aux)
            }
        }
    }

    lista.sort(function(a,b){
        return b[2] - a[2]
    })

    for (i = 0 ; i<lista.length ; i++){
        var aux = lista[i]
        listasorted.push(aux[0])
        listasorted.push(aux[1])
        listasorted.push(aux[2])
    }
    return listasorted.reverse()
}

```

Após isto vamos percorrer todas as hipóteses de jogada para o maximizante, utilizando um ciclo nessas listas de prioridades, depois de validar se a jogada pode ser feita nesse sitio é feito um clone do estado atual e simulamos a jogada no clone.

Após a jogada ser simulada no clone voltamos a executar recursivamente a função minimax mas mandamos como argumento este clone com a jogada simulada e diminuímos a profundidade

```

for (var tipo = 0 ; tipo < 2 ; tipo += 1){
    var m = aux[tipo];
    var zz;
    if (m == 'l')(zz = prioL.length);
    else(zz = prioC.length);
    for (var auxx = 0 ; auxx < zz; auxx = auxx + 3){
        if (m == 'l'){
            var i = prioL.pop();
            var j = prioL.pop();
            var prio = prioL.pop();

            if(this.validaJogadaLinha(nodo,i,j)){
                let clone = deepClone(nodo);
                jogadorAnterior = clone.jogadorAtual;
                clone = this.simulaJogadaLinha(clone,i,j);

                if (jogadorAnterior == clone.jogadorAtual){
                    p+=1;
                    var auxilio = caminho
                    if ( profundidade == profundidadeMaxima && iteracoes<5 && dificuldade==3 ){
                        auxilio.push(['l',i,j])
                    }
                    var auxL = this.minimax(clone,profundidade - 1 ,alpha,beta,0,auxilio,iteracoes+1);
                    auxilio = [];
                    var valorTab = auxL[0];
                    var lastChain = auxL[1];
                }
                else{
                    var auxL = this.minimax(clone,profundidade - 1 ,alpha,beta,0,caminho,iteracoes+1);
                    var valorTab = auxL[0];
                    var lastChain = auxL[1];
                }
                if(profundidade == profundidadeMaxima && fimEncontradoaux && mudarMelhorJogada ||
                    profundidade == profundidadeMaxima && (valorTab > tamanhoTabuleiro/2) && mudarMelhorJogada ){
                    fimEncontrado = 1;
                }
            }
        }
    }
}

```

No final deste caso se encontrarmos uma jogada melhor, ou seja, que o máximo dela é maior que o máximo atual, vamos trocar o máximo atual e guardar esta jogada na estrutura.

```

let a = valorMax;
valorMax = Math.max(valorMax, valorTab);

if (profundidade == profundidadeMaxima && mudarMelhorJogada == 1 && cond){           //so muda jogadas no primeiro minimax
    if (a == valorTab){          //melhor jogada tambem
        if (Math.random() < 0.25){ //para ser mais aleatório a jogada, quando as jogadas são valorizadas de igual forma
            jogadaBotStruct.modo = 'l'
            jogadaBotStruct.iMelhor = i
            jogadaBotStruct.jMelhor = j
            jogadaBotStruct.caminho = lastChain
            jogadaBotStruct.valor = valorTab
            prioridadeJogada = Math.max(prio , prioridadeJogada)
        }
    }
    else {
        if (valorMax == valorTab){ //melhor jogada agora
            jogadaBotStruct.modo = 'l'
            jogadaBotStruct.iMelhor = i
            jogadaBotStruct.jMelhor = j
            jogadaBotStruct.caminho = lastChain
            jogadaBotStruct.valor = valorTab
            prioridadeJogada = Math.max(prio , prioridadeJogada)
        }
    }
}
}
}

```

Para o minimizante a estratégia é a mesma mas envez de procurar-mos um valor maior que o atual registado, procuramos um menor.

No final do ciclo do maximizante temos o pruning em alpha:

```

alpha = Math.max(alpha,valorMax);
if (beta <= alpha || fimEncontrado)           // pruning
|   break;

```

E no final do ciclo do minimizante temos o pruning em beta semelhante ao em alpha:

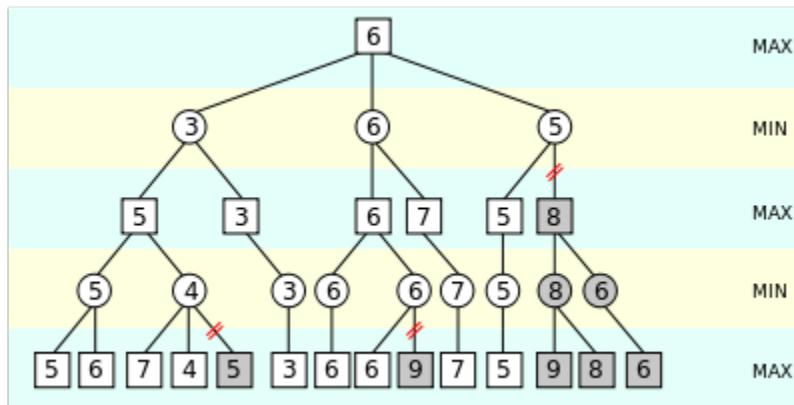
```

beta = Math.min(beta,valorMin);
if (beta <= alpha)                         // pruning
|   break;

```

Estes prunings são uma extensão de performance ao algoritmo Minimax chamado de “alpha-beta pruning”, não influenciando o resultado final, apenas a velocidade de execução pois diminui as subárvores que temos de pesquisar.

Tomando a seguinte árvore como exemplo



Analizando a mesma da esquerda para a direita, o alpha-beta pruning vai cortar os ramos a vermelho pois já encontrou um valor melhor é escusado procurar nas subárvores cortadas.

Com isto a função principal minimax está acabada, falta agora ver como a função é chamada e usada.

Quando queremos que o bot jogue chamamos a função botMain, que por sua vez chama a jogadaBot que chama a Minimax que vimos anteriormente.

No final da jogadaBot vamos ter então as coordenadas da melhor jogada na jogadaBotStruct, por sua vez no final da botMain vamos ter a lista das melhores jogadas feitas pelo bot.

A botMain é chamada recursivamente dentro dela até que o jogo acabe ou seja a vez do outro jogador.

Caso seja a vez do bot a função vai chamar a jogadaBot.

Também temos nesta parte implementada uma tática para o nível 3 de dificuldade, quando temos uma cadeia grande tentamos capturar tudo menos os 2 últimos quadrados assim o oponente vai capturar esses e fazer uma jogada má de seguida, com grande chance de nos oferecer outra cadeia grande.

```
function botMain() {
    chain = []
    ischain = 0
    estado = deepClone(atualizaEstado(estado, jogadorAtual, linhasMatrixJogadas, colunasMatrixJogadas, boxesMatrixJogadas));
    lateGame = isLateGame(estado)

    if (testeFimDeJogoBot(estado)) {
        return
    }
    if (jogadorAtual == 2) { //bot

        fimEncontrado = 0;
        fimEncontradoaux = 0;
        jogadaBot();
        var sitio = jogadaBotStruct.modo
        var i = jogadaBotStruct.iMelhor
        var j = jogadaBotStruct.jMelhor
        var lista = jogadaBotStruct.caminho
        var valor = jogadaBotStruct.valor
        lista = removeSame(lista)

        if (lista.length == 0){
            lista.push([sitio,i,j])
        }
        else if (lista.length >= 1){
            isChain = 1
            if(fimEncontrado == 0 && lateGame && dificuldade==3){
                var last = lista.pop()
                lista.pop()
                lista.push(last)
            }
        }
        for (var a = 0 ; a < lista.length && jogadorAtual == 2 ; a++){
            var aux = lista[a]
            sitio = aux[0]
            i = aux[1]
            j = aux[2]
            if (sitio == 'l') {
                jogadasBot.push([sitio,i,j])
                JogadaLinha(i,j)
            } else if (sitio == 'c') {
                jogadasBot.push([sitio,i,j])
                JogadaColuna(i,j)
            }
        }
        if (jogadorAtual == 2) {
            botMain()
        }
    }
}
```

É na jogadaBot onde o minimax é chamado, os minimax variam na profundidade dependendo da dificuldade escolhida pelo jogador, quando mais profunda a pesquisa melhor a previsão da melhor jogada.  
As profundidades são iguais ao nível.

É de notar que se aumentássemos mais do que 3 a profundidade máxima o bot vai demorar um tempo considerável a jogar, principalmente nos tabuleiros maiores.

```
function jogadaBot() {  
  
    if (dificuldade == 1) {  
        profundidadeMaxima = 1;  
        minimax(estado, profundidadeMaxima, Number.NEGATIVE_INFINITY, Number.POSITIVE_INFINITY, 1, [], 0)  
    } else if (dificuldade == 2) {  
        profundidadeMaxima = 2;  
        minimax(estado, profundidadeMaxima, Number.NEGATIVE_INFINITY, Number.POSITIVE_INFINITY, 1, [], 0)  
    } else if (dificuldade == 3) {  
        profundidadeMaxima = 3;  
        minimax(estado, profundidadeMaxima, Number.NEGATIVE_INFINITY, Number.POSITIVE_INFINITY, 1, [], 0);  
    }  
}
```

Após estas funções executarem temos então a nossa lista de jogadas feitas pelo bot, agora o que resta é ler e apresentar as mesmas.

No ínicio fizemos tudo no mesmo processo, mas tivemos um problema pois enquanto o processo estava a executar a função do bot, ele não podia atualizar a vista e o jogo parecia “empancado” quando o bot jogava, não respondendo aos cliques nos botões e também não apresentava a ampulheta.

A solução para isto passou por dividir a vista em um processo e o bot em outro processo.

Em javascript podemos fazer isto usando “Workers”, estes Workers correm código em outro processo e comunicam com o processo principal através de mensagens.

Assim temos todo o código do bot em um ficheiro chamado “Worker.js” que comunica com o código do jogo no “Jogo.js”.

Dito isto quando for a vez do bot jogar vamos chamar-lo da seguinte forma:

```
var myWorker = new Worker('worker.js');  
  
estado = new auxiliares().atualizaEstado(estado, jogadorAtual, linhasMatrixJogadas, colunasMatrixJogadas, boxesMatrixJogadas);  
  
var objetoEnviar =  
{  
    e : estado,  
    d : dificuldade,  
    t : tamanhoTabuleiro,  
    pr: probRandom  
}  
  
var object = JSON.stringify(objetoEnviar);  
myWorker.postMessage(object)
```

Ou seja, definimos o ficheiro onde está o worker e criamos o objeto que queremos lhe enviar que é composto pelo estado atual do jogo, a dificuldade, tamanho do tabuleiro e um valor gerado para colocar as jogadas mais aleatórias.

E enviamos isso ao worker em formato JSON.

Agora no worker recebemos esse JSON e damos parse do mesmo guardando os dados variáveis locais ao mesmo.

Após isso é só executar a botMain, receber os resultados da mesma e enviar ao processo principal.

```
onmessage = function (message) {
    dados = JSON.parse(message.data);

    linhasMatrixJogadas = dados.e.linhasMatrixJogadas;
    colunasMatrixJogadas = dados.e.colunasMatrixJogadas;
    boxesMatrixJogadas= dados.e.boxesMatrixJogadas;
    dificuldade = dados.d;
    jogadorAtual = dados.e.jogadorAtual;
    tamanhoTabuleiro = dados.t;
    probRandom = dados.pr;

    botMain();
    var r = {
        c : colunasMatrixJogadas,
        l : linhasMatrixJogadas,
        b : boxesMatrixJogadas,
        j : jogadasBot
    }

    postMessage(r);
}
```

Que vai receber a informação da seguinte forma:

```
myWorker.onmessage = function(oEvent) {
    console.log('Worker said : ' + oEvent.data);
    ...

    var estadoRecebido = oEvent.data;
    var colunasRecebidas = estadoRecebido.c;
    var linhasRecebidas = estadoRecebido.l;
    var boxesRecebidas = estadoRecebido.b;
    var listaJogadasBot = estadoRecebido.j;
    console.log(colunasRecebidas,linhasRecebidas,boxesRecebidas,listaJogadasBot)

    listaJogadasAnteriores = listaJogadasBot;      Dimit3, 3 weeks ago + estatisticas, informacao e bot com worker

    for(i = 0; i < listaJogadasBot.length; i++ ){
        if(!jogoAcabou){
            if (listaJogadasAnteriores[i][0] == "1"){
                context.jogarLinha(listaJogadasAnteriores[i][1],listaJogadasAnteriores[i][2],linhasMatrix[listaJogadasAnteriores[i][1]][listaJogadasAnteriores[i][2]]);
            }
            else {
                context.jogarColuna(listaJogadasAnteriores[i][1],listaJogadasAnteriores[i][2],colunasMatrix[listaJogadasAnteriores[i][1]][listaJogadasAnteriores[i][2]]);
            }
        }
    }
}
```

Percorre a lista e faz a jogadas recebidas pelo bot, após isto termina o processo do Worker e continua o jogo para a vez do jogador.

## Conclusão

Este projeto permitiu-nos aprofundar e colocar e praticar diversos conhecimentos adquiridos ao longo deste curso.

Tivemos dificuldade na implementação do bot, pois na implementação deste foram precisos vários ajustes de forma a chegarmos a uma implementação que tenha um tempo de espera pela resposta do bot razoável e dificuldade ajustada para o nível do bot

Estamos assim satisfeitos com o resultado deste projeto pois conseguimos chegar a implementação do jogo “Pontos” que cumpra todos os requisitos pedidos.

Posto isto, e em jeito de conclusão achamos que o projeto foi bem desenvolvido e que se obteve o resultado pretendido. Esperemos que esteja do agrado dos professores também.

*André Araújo*

*Henrique Ferreira*

*Daniel Ribeiro*

*Paulo Costa*

## Referências

- [1] <http://phaser.io/examples>
- [2] <https://github.com/photonstorm/phaser3-examples>
- [3] <https://www.youtube.com/watch?v=KU9Ch59-4vw>
- [4] <https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/>
- [5] <https://en.wikipedia.org/wiki/Minimax>