O nosso objetivo foi tentar implementar um código que leve a que o nosso jogo se ajuste à resolução da página, cada vez que esta seja redimensionada (ideia inspirada no jogo slither.io). Numa pesquisa, na internet, o nosso grupo encontrou vários sites, mas o que melhor correspondeu ao nosso interesse foi o que podemos consultar através do link: <https://stackoverflow.com/questions/33515707/scaling-a-javascript-canvas-game-properly>.

**TAMANHO**

Com base no que lemos, aprendemos que deveríamos começar por indicar os valores das medidas de dimensão originais (base) do jogo, a saber: a altura, *originalH*, e a largura, *originalW*. Quando a página fosse redimensionada, os elementos do canvas seriam também redimensionados através do método scale().

Nos primeiros testes realizados, consideramos a altura da página e a altura base, importantes para a determinação do valor y da escala. O valor x era calculado com a ajuda da largura da página e da largura base. O primeiro parâmetro de scale() seria a variável *sW,* que armazenava o valor da razão entre as duas larguras. No lugar do segundo parâmetro estaria a variável *sH*, que contêm o valor da razão entre as duas alturas

ctx.scale(sW,sH)

Embora houvesse a vantagem de que em qualquer resolução, que não fosse 1920x1080, o utilizador ter o mesmo campo de visão do que quem joga a 1920x1080, existia o problema de que os elementos do canvas perdiam a sua forma característica quando apenas a largura ou a altura da página fosse alterada. Por exemplo, se somente a altura/largura da página diminuísse, os asteroides já não teriam o formato de um círculo (que é o pretendido), mas sim a forma oval. Como o jogo é offline, não há necessidade de indicar a escala com precisão. Portanto, escolhemos só uma das medidas de distância como fator que influencia a escala. A escolhida foi a largura, ficando *sW* responsável por definir o valor dos dois parâmetros do scale().

ctx.scale(sW,sW)

A escala do elemento aumenta caso a largura da página seja maior do que a largura base. Diminui se a largura da página for menor do que a largura base. Na resolução original do jogo (que é 1920 de largura por 1080 de altura, uma vez que é a mais popular), os elementos do canvas não sofrem nenhuma transformação à escala, dado que ctx.scale(sW,sW) ⬄ ctx.scale(1,1). Como recorremos ao método scale(), tivemos que substituir o valor dos parâmetros relativos às coordenadas x e y, que se encontram nos métodos que desenham o objeto, para zero. De seguida, atribuimos aos parâmetros do método translate() esses valores que foram substituídos. Por exemplo, na função draw() da classe Ball, antes de ter aplicado o método scale(), esta função abrangia o seguinte código:

ctx.beginPath()

ctx.arc(**this**.x, **this**.y, **this**.R, **0**, **2** \* Math.PI)

(…)

ctx.stroke()

Para aplicar a escala, fizemos as seguintes alterações e acréscimos ao código:

ctx.translate(**this**.x, **this**.y);

ctx.scale(**this**.sW,**this**.sW)

ctx.beginPath()

ctx.arc(**0**, **0**, **this**.R, **0**, **2** \* Math.PI) // altero os valores dos dois

primeiros parâmetros para 0

(…)

ctx.stroke()

O método *save()* e o método *restore()* foram chamados dentro da função *render().*

Também tentamos ajustar o tamanho da fonte à resolução. A desvantagem é que o tamanho da letra diminui quando a resolução diminui. Por isso é difícil conseguir ler o que está escrito. Em contrapartida, podemos observar, no jogo slither.io, que o valor do tamanho da letra nunca muda. Quando a resolução é aumentada, é criada a ilusão de que o tamanho da letra diminuiu e quando diminuímos a resolução o tamanho parece aumentar. Segue-se algumas ilustrações que podem ser elucidativas.

**Antes:**

Quando a página inteira carrega, incluindo o seu conteúdo (imagens, CSS, scripts, etc.), eis o código que usamos para ajustar o texto da landing page:

window.onload = () => {

menuInicial.children[**0**].style.fontSize= (W \* **40**)/originalW + "px"

menuInicial.children[**1**].style.fontSize= (W \* **17**)/originalW + "px"

(…)

}

Este foi o código implementado, dentro da função makeItResize() (falaremos desta função mais adiante) para ajustar a fonte à resolução, todas as vezes que a página é redimensionada:

**function** makeItResize() { //função que redimensiona todo conteúdo do

canvas

(…)

//O nome do jogo que se encontra posicionado no rodapé da página

menuInicial.style.width=W + "px"

menuInicial.style.height=H + "px"

menuInicial.children[**1**].style.fontSize= (W \* **17**)/originalW + "px"

menuInicial.children[**0**].style.fontSize= (W \* **40**)/originalW + "px"

//O texto que se encontra no canto superior esquerdo

playerPoints.font = `${**20**\*W/originalW}px myFont`

playerLives.font = `${**20**\*W/originalW}px myFont`

playerLives.xText= W\***40**/originalW

playerLives.yText= H\***90**/originalH

playerPoints.xText=W\***40**/originalW,

playerPoints.yText= H\***60**/originalH,

(…)

}

Imagem que demonstra como fica a landing page no telemóvel:

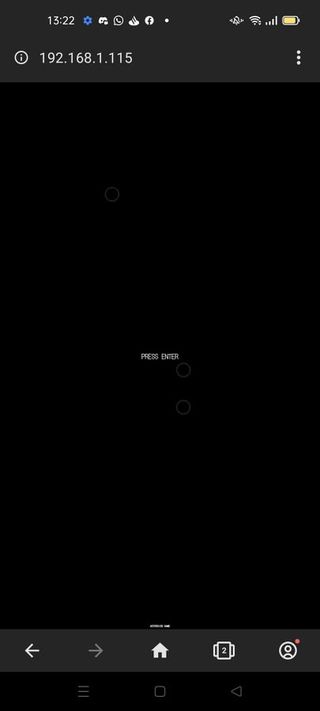
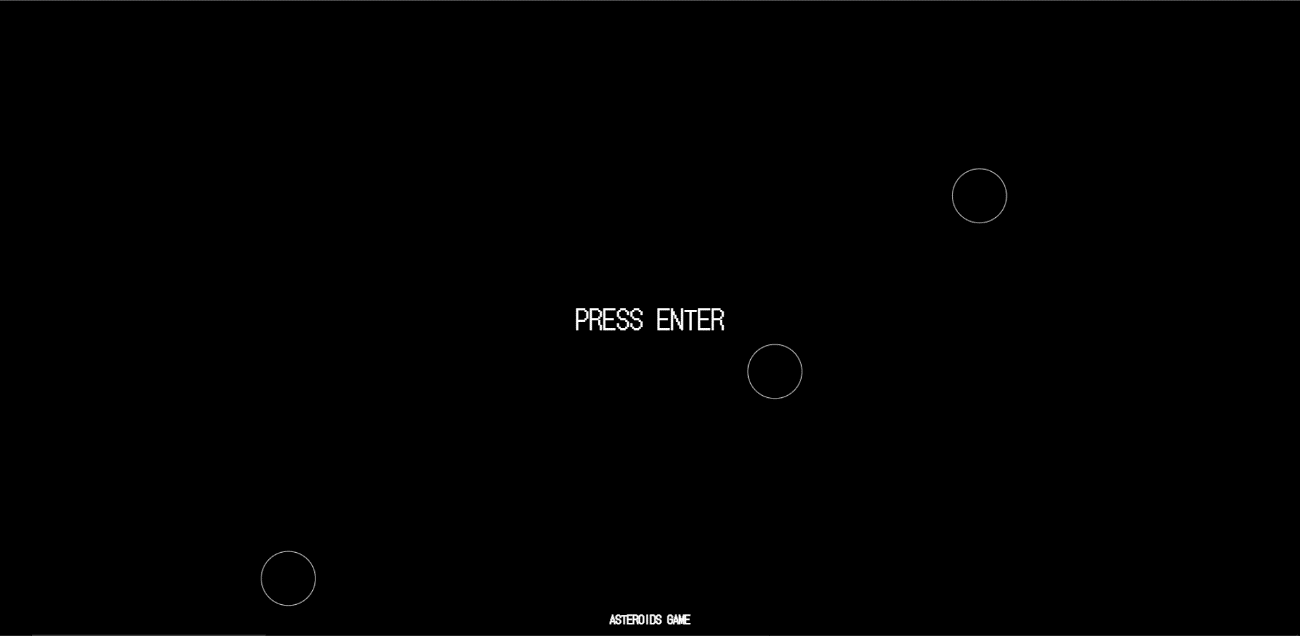


Imagem que demonstra como fica a landing page no computador com uma resolução 1920x1080.



**Depois:**

Comentamos todo o código que ajusta o tamanho da fonte à resolução. Obtivemos o seguinte resultado:





Concluímos que não tem lógica ajustar o tamanho da fonte à resolução.

**VELOCIDADE**

Durante os testes, verificamos que quando se aumentava a largura da página, a velocidade de um elemento, apesar de ser constante (=1), parecia diminuir. Isto porque, quanto maior o espaço do canvas, maior será o tempo que um objeto demora a deslocar-se do ponto A ao ponto B. Logo, definimos que o valor da velocidade seria o resultado da multiplicação entre o valor da velocidade na resolução original (=1) e *sW*,

*ball.v* = *ball.v* \* *sW*

Esta expressão permite que o objeto se movimente mais rápido numa página com uma largura maior e mais lento numa página com uma largura menor, dando a sensação de que os elementos se movimentam sempre à mesma velocidade, independentemente da largura da página.

**“DEBOUNCE”**

O site, que foi referido pela professora no moodle e cujo link deixamos aqui: <https://flaviocopes.com/canvas/>, ensina-nos como podemos fazer com que o jogo se adapte à resolução, não quando a página é redimensionada, mas sim, quando passou um x tempo desde que a página foi redimensionada. Se, nesse espaço de tempo, a página for novamente redimensionada, então, o programa recomeça a contagem do tempo.

O seguinte código faz com que a variável *resize* receba o valor true 600 milissegundos depois da página ter sido redimensionada:

**function** debounce(func) {

**let** timer;

**return** **function** () {

**if** (timer) { clearTimeout(timer) }

timer = window.setTimeout(func, **600**)

};

};

window.addEventListener('resize', debounce((**function** () {

resize = true

})))

A função *render()* está estruturada da seguinte forma:

**function** render(){

**if**(resize){

resize = false

//codigo para redimensionar os elementos do canvas

}**else**{

//codigo para desenhar o jogo

}

window.requestAnimationFrame(render)

}

No frame em que *resize* fosse igual a true, o programa mudava o valor desta variável para false e também o valor das dimensões do canvas e do seu conteúdo. Nos frames seguintes, o programa voltaria a desenhar o jogo.

No entanto, concluimos que ao usar o *setTimeout()* e o *requestAnimationFrame()* ao mesmo tempo pode baixar a performance do jogo. Também quando começamos a codificar a landing page, na função *render()*, o código estava a depender de muitos ifs e elses. Portanto, surgiu-nos a ideia de acrescentar algo na função *debounce()* que permitisse que *H1* e *W1* (variáveis abordadas no próximo capítulo) guardassem, respetivamente, o valor da altura e da largura da página, antes de ser redimensionada e que *H* e *W* armazenassem os novos valores da altura e da largura.

**function** debounce(func) {

**let** timer;

**return** **function** () {

**if**(!resize){

H1 = H // o programa vai guardar a altura da página

antes de ser redimensionada

W1 = W // o programa vai guardar a largura da página

antes de ser redimensionada

resize = **true** // significa que a página foi

redimensionada

}

H = window.innerHeight // atualiza o valor da altura

W = document.body.offsetWidth // atualiza o valor da largura

**if** (timer) { clearTimeout(timer) }

timer = window.setTimeout(func, **600**)

};

};

Não usamos window.innerWidth para fornecer à variável *W* os novos valores da largura, porque nem sempre estavam corretos. Com o document.body.offsetWidth, os valores de largura da página estão sempre corretos. Para a variável *H*, das propriedades existentes - innerHeight, clientHeight e offsetHeight - a vencedora foi innerHeight por ter dado mais valores corretos, mas não significa que nunca dê valores incorretos.

Neste caso, *resize* fica com o valor true mal a página seja redimensionada.

window.addEventListener('resize', debounce((**function** () {

makeItResize()

})))

(…)

**function** render() {

//codigo para desenhar o jogo

**if** (!resize) { // enquanto a página não for redimensionada

window.requestAnimationFrame(render)

}

}

A função render() para de ser chamada no frame em que *resize* é igual a true

A função makeItResize() é chamada 600 milisegundos depois da página ter sido redimensionada. Esta função vai ajustar o canvas e o seu conteúdo à página e depois chamará novamente a função *render()* para voltar a desenhar o jogo.

**function** makeItResize() {

//codigo para redimensionar os elementos do canvas

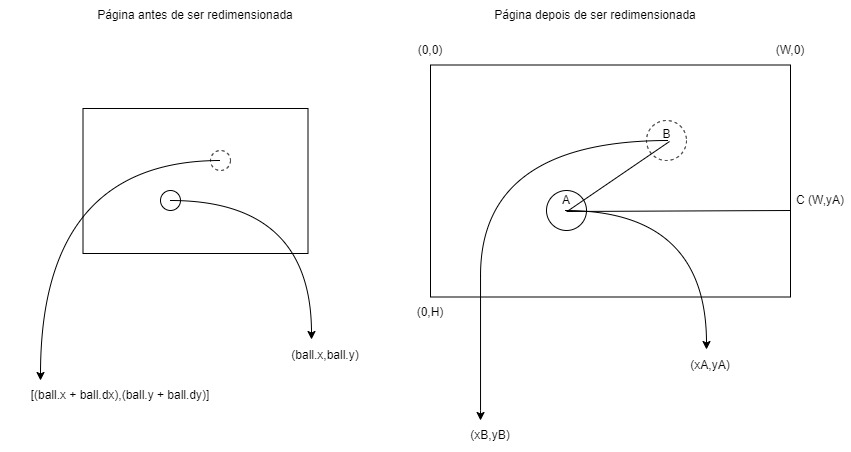
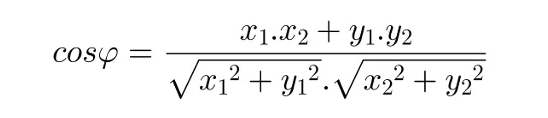
window.requestAnimationFrame(render) // ou render()

}

Resumindo, quando a página for redimensionada, o canvas deixará de ser desenhado, como se tivéssemos dado pausa ao jogo. O jogo reajustado voltará a funcionar, 0.6 segundos depois da página ter sido redimensionada

**POSIÇÃO**

Por fim, precisamos de garantir que os objetos se encontrem na mesma posição em que se encontravam, antes da página ter sido redimensionada. À primeira vista, constatamos que, devido ao redimensionamento, não só as coordenadas do elemento mudam como também a sua direção. Por isso, recorremos à expressão matemática que nos fornece o cosseno do ângulo entre o vetor do ponto A, relativo à posição atual do elemento e do ponto B, correspondente à posição seguinte do mesmo, e o vetor do ponto A e ponto C. Para se compreender melhor, desenhamos o seguinte esboço:



****

****

****Porém,

****Logo,

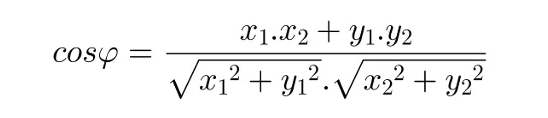
**let** xB = ((ball.x + ball.dx)\* W) / W1

**let** xA = ball.x = (ball.x \* W) / W1

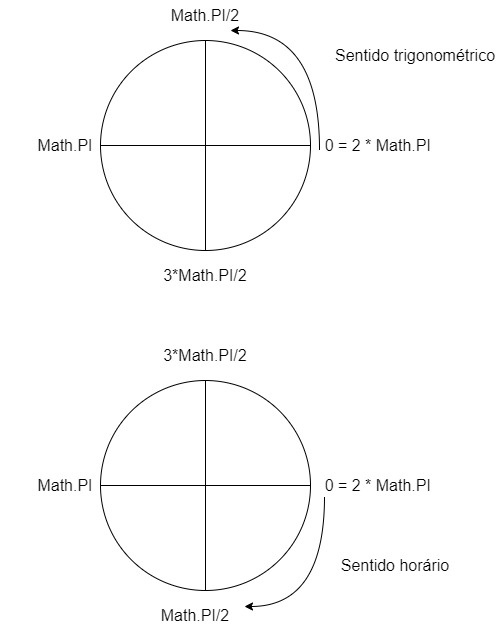
**let** yB = ((ball.y + ball.dy)\* H) / H1

**let** yA = ball.y = (ball.y \* H) / H1

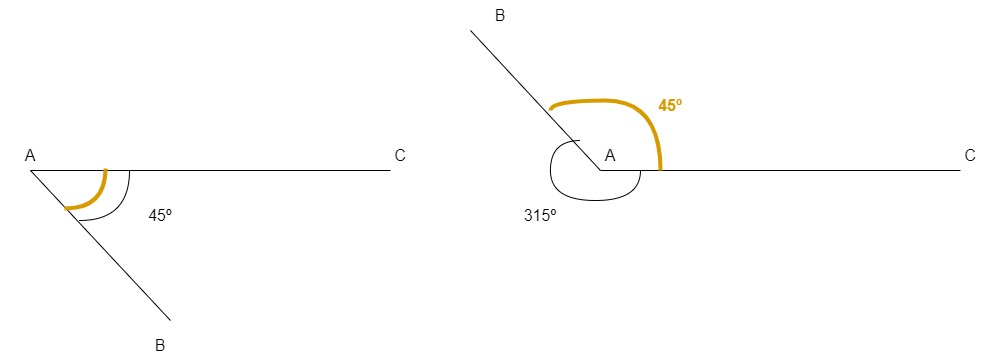
Usamos a regra de três simples para determinar o valor das novas coordenadas, como o caso do x do ponto A (se o x do asteroide tinha esse respetivo valor quando a largura da resolução era *W1*, então para *W*, o novo valor de x será *xA*).

A expressão ´

devolve-nos apenas o valor absoluto do cosseno do ângulo no sentido anti-horário (ou trigonométrico) entre os vetores e , no canvas, o sentido dos ângulos é horário.



Se o valor original da direção (o valor de *ball.d*) fosse maior ou igual do que Math.PI (180 graus) e menor do que 2\*Math.PI (360 graus), então o valor da variável *angulo* teria que ser o valor2\*Math.PI (360) - *angulo.* Por exemplo, se *ball.d* tiver o valor Math.PI/4 (45 graus) e a página for redimensionada, a variável *angulo* terá o valor Math.PI/4+x porque é o menor ângulo entre os dois vetores. Contudo, se o valor de *ball.d* for maior do que Math.PI (180 graus) e menor do que 2\*Math.PI (360 graus), como 7/(4\*Math.PI) (315 graus), significa que, caso a página seja redimensionada, o valor do menor ângulo será 2\*Math.PI (360 graus) - *ball.d,* que é 45+x graus.



Por isso, temos que fazer2\*Math.PI (360) – *angulo* para obtermos outra vez 7/(4\*Math.PI)+x.

A variável *angulo* representa a nova direção do objeto enquanto que *ball.d* representa a direção inicial. O valor de *angulo* pode ser diferente do valor de *ball.d,* portanto*,* somou-se x aos valores da variável *angulo.*

Segue aqui o código final:

H1 = H

W1 = W

canvas.width = window.innerWidth;

canvas.height = window.innerHeight;

H = window.innerHeight;

W = window.innerWidth;

b.forEach(**function** (ball) {

**let** xB = ((ball.x + ball.dx)\* W) / W1

**let** xA = ball.x = (ball.x \* W) / W1

**let** yB = ((ball.y + ball.dy)\* H) / H1

**let** yA = ball.y = (ball.y \* H) / H1

**let** xVetorAB = xB - xA

**let** yVetorAB = yB - yA

**let** xVetorAC = W - xA

**let** denominador = xVetorAB \* xVetorAC

**let** normaAB = Math.sqrt(xVetorAB\*\***2** + yVetorAB\*\***2**)

**let** normaAC = Math.sqrt(xVetorAC\*\***2**)

**let** numerador = Math.abs(normaAB \* normaAC)

**let** cos = denominador / numerador

**let** angulo = Math.acos(cos)

**if**(Math.PI <= ball.d && ball.d < **2**\*Math.PI ){

angulo = **2**\*Math.PI - angulo

}

ball.dx = ball.v \* Math.cos(angulo)

ball.dy = ball.v \* Math.sin(angulo)

})

Concluindo, decorridas as quatro fases, conseguimos deixar o nosso jogo offline *Asteróides* adaptável a qualquer tipo de resolução.