ESCOLA SUPERIOR DE GESTÃO E TECNOLOGIA DE SANTARÉM

Licenciatura em Informática

Aplicações Multimédia

Elder Costa - n 180100528
180100528@esg.ipsantarem.pt
Tauã Almeida - n 180100527
180100527@esg.ipsantarem.pt
Rayssa Rosa - n 180100523
180100523@esg.ipsantarem.pt

Trabalho Prático 2

Jogo Web em CANVAS

Resumo: *Este documento visa apresentar um relatório acerca do desenvolvimento do segundo trabalho prático proposto da matéria de Aplicações Multimédia. Nele serão apresentadas as soluções encontradas pelos alunos, bem como as funções e o resultado final do Jogo desenvolvido para Web utilizando canvas.*

Palavras-chave: *Canvas; Desenvolvimento Web; Javascript*

Santarém, Portugal,

21 de Junho de 2019.

1 - Introdução

O Jogo foi desenvolvido para web em CANVAS HTML5 e se denomina “Estanislau e o campo de armadilhas". Ele possui background com scrolling e físicas implementadas como aceleração, gravidade, bouncing e salto e foi inspirado no famoso jogo *I Wanna Be The Boshy* de Jesper Solgryn Erlandsen, que consiste em um jogo de plataforma 2D com várias armadilhas pelo caminho. O jogo se passa em um campo cheio de artifícios mortais, em que o personagem principal precisa passar por todas as adversidades encontradas sem ser ferido até chegar ao fim desse território.

2 - Descrição do Jogo

O jogo possui dois níveis onde o objetivo final é sair do campo armadilhas para conseguir vencer o jogo. Para isso ele deve escapar de todos os contratempos encontrados dentro do jogo.

No primeiro nível do jogo, ainda está de dia e o personagem pode encontrar armadilhas como barris que explodem, serras que sobem e descem e espinhos de ferro que podem se mover de um lado para o outro. Ele também pode encontrar cogumelos que lhe dão 10 pontos ao serem pegos.

Para passar para o segundo nível o personagem deve chegar na placa de madeira que se encontra no final do campo, onde ele vai se teletransportar para um mundo invertido . Ao para esse mundo, ele encontrará uma área onde costumava ser um cemitério, onde as armadilhas são mais perigosas e mais velozes.

Ao chegar ao fim da zona do cemitério, o personagem irá encontrar uma placa com uma seta indicando o fim do território inimigo e, portanto, o fim do jogo.

3 - Arquivos e Funções Utilizadas

O jogo possui 12 arquivos javaScript que serão brevemente explicadas a seguir.

Entity.js - Arquivo da classe Entity que é responsável por todas as entidades do jogo. Nela se encontram funções de colisão que fazem o teste de *hit,* que diz se ocorreu uma colisão ou não e funções de *Block* que bloqueiam uma entidade passar por cima de outra, caso haja um colisão.

GameWorld.js, Camera.js e Background.js - São as classes responsáveis por fazer a rolagem do background (Background Scrolling).

SpriteSheet.js - Classe que indexa a URL dos assets e faz o carregamento deles no mapa.

Objetos.js - Classe responsável pelos objetos do mapa como arbustos, caixotes e cogumelos.

Traps.js e Ground.js - Responsáveis pelas plataformas, armadilhas e chão do mapa, respectivamente.

groundMap.js - Arquivo que possui matrizes com objetos do jogo que formam o mapa.

Hero.js - Classe responsável pelo personagem principal a qual possui todos os seus atributos que serão utilizados durante o jogo.

animatedGame.js - Arquivo com todas as funções de interação, onde é feita toda a lógica do jogo.

3.1 - Principais Funções de AnimatedGame.js

init() - Cria um canvas 2D e chama a função load().

load() - Faz o carregamento de todas as SpriteSheets que serão necessárias durante todo o jogo, assim como todos os sons que serão usados.

loaded() - Função ativada quando todos as SpriteSheets do jogo tiverem sido carregadas e fica a espera do jogador pressionar alguma tecla para o jogo ser iniciado.

setupGame() - Cria um objeto do tipo Heroi, chama as funções que colocam os objetos no jogo (generateGround(), generateTrap(), generateObject()), ativa o *EventListener* para *keyUp* e *KeyDown* e chama a função playGame() ao final.

playGame() - Função que inicia o jogo e mudando do ambiente *espera* para o ambiente *jogando* e chama a função update() para começar o ciclo do jogo.

generateGround() - Cria um objeto Ground para cada tipo de plataforma encontrada na matriz groundMap, colocando esse novo objeto no vetor de entidades (entities) e no vetor de entidades do Ground (grEntities).

generateTrap() - Varre a matriz trapMap criando um objeto Trap para cada armadilha encontrada e inicializa as variáveis necessárias como na criação de *SPIKE,* onde é inicializado a velocidade em que o objeto irá se mover e as coordenadas iniciais de x e y. Após essa etapa cada objeto será adicionado nos vetores *entities* e *trapEntities.*

GenerateObject() - Gera os objetos do jogo passando por cada posição da matriz objectMap ao criar um objeto da classe Objetos e o adicionar nos vetores *entities* e *objectEntities*.

keyDownHandler() - Função ativada ao pressionar uma tecla onde cada tecla possui uma posição no vetor *teclas* que passa para *true* ao ser acionada.

keyUpHandler() - Função ativada ao soltar uma tecla onde sua posição no vetor *teclas* passa para *false*.

moveHero() - Responsável por fazer a movimentação do personagem, assim como aplicar as físicas presentes no jogo como a gravidade, aceleração e velocidade máxima. Ele checa o vetor *teclas* para saber qual a movimentação deve ser feita e ativa a sua respectiva animação.

moveTrap() - Movimenta todas as armadilhas fazendo a checagem de direção de cada uma delas para que não haja colisão.

takeObject() - É chamada quando o personagem coleta um objeto que vale pontos. Quando essa função é acionada o atributo *pontos* encontrada no objeto *umHero* recebe um acréscimo de 10 valores.

checkColisions() - Função que verifica se existe colisões entre o personagem principal (var *umHero*) e os vetores trapEntities, grEntities, objectEntities. Também é feito a ativação dos sons de colisão caso seja necessário.

update() - Responsável pelo ciclo do jogo e é chamado a cada refrescamento do browser. Ela chama as funções *moveTrap()* e *moveHero()*. Também é responsável por manter a câmera e o personagem principal dentro dos limites do mundo e mover a câmera de acordo com a movimentação do objeto *umHero.* Após essa etapa é feita a chamada para as funções *checkColisions(), render()* e *requestAnimationFrame().*

clearArrays() - Função cujo objetivo é fazer a limpeza das entidades desativadas.

clearGeneralArrays() - Assim como *clearArrays(),* tem o objetivo de limpar as entidades desativadas porem para apenas um array especifico que é passado por parâmetro.

filterByActiveProp() - Função que recebe um objeto e devolve esse mesmo objeto apenas se ele estiver ativo dentro do mapa.

changeLevel() - Função chamada quando ocorre a mudança de nível, limpando todos os vetores de entidade utilizados. Ele também chama as funções *cancelAnimationFrame*(), *setupGame*().

render() - Responsável pela limpeza do canvas e por salvar o contexto. Ele renderiza todas as entidades do jogo por meio do vetor *entities* e depois restaura o contexto.

endGame() - Função para tratar o fim do jogo. Ela chama a função *stopGame*(), verifica se o usuário perdeu ou ganhou e projeta um modal na tela com uma mensagem, seja de vitória ou derrota, e os pontos do personagem.

4 - Conclusão Screenshots

O trabalho tinha como objetivo inicial ter uma maior complexidade e uma maior diversidade de armadilhas, entretanto devido à má organização do tempo dos participantes do projeto o trabalho não ficou como desejado.

Ocorreu, também, um atraso no andamento do código devido á um erro encontrado na classe Entity, que demorou demasiadamente para ser percebido e afetou a programação do segundo nível do jogo, fazendo com que ele não tenha nem o número nem a diversidade de armadilhas planejadas.

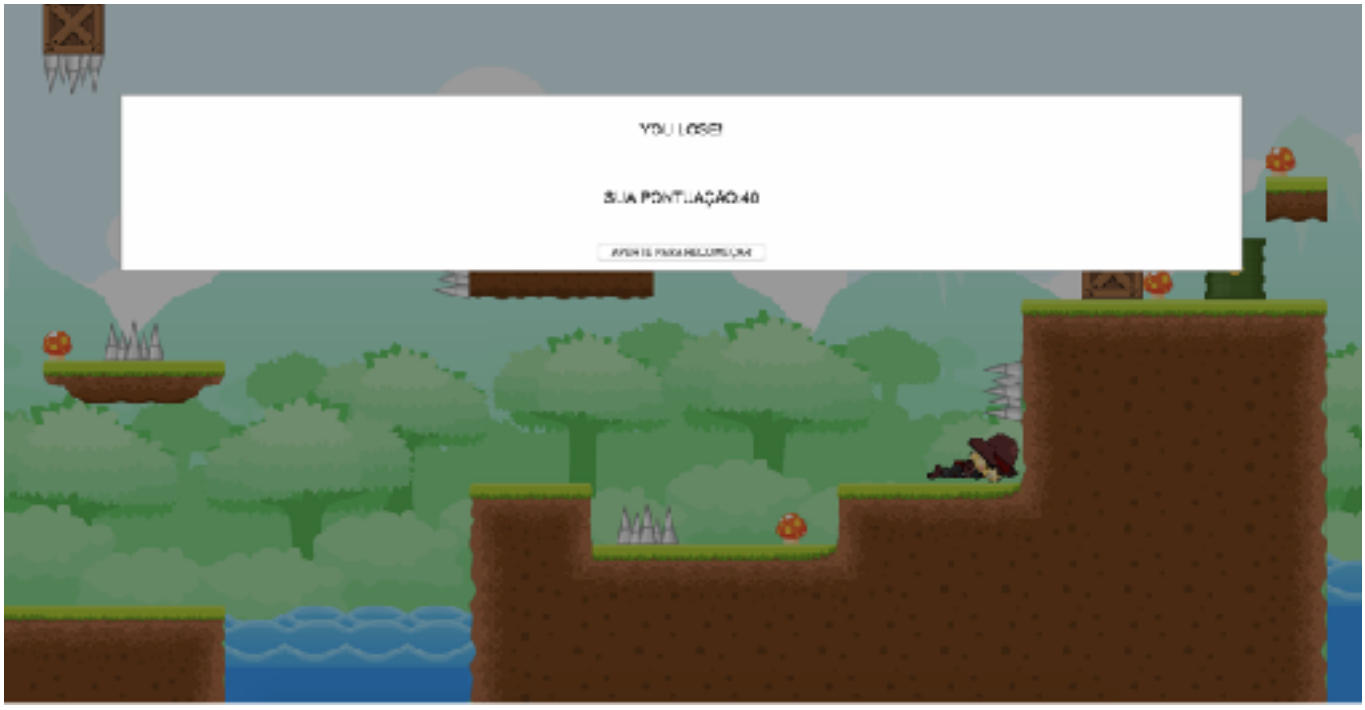
Apesar de todas as adversidades encontradas, a criação do jogo teve suma importância para o aprendizado e fixação das técnicas aprendidas em sala.

5 - Screenshots

MAPA - NÍVEL 1

MAPA - NÍVEL 1

MAPA - NÍVEL 2



MODAL DE FIM DE JOGO