Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
Prof. Igor Knop
Seminário em Computação – Desenvolvimento de Jogos com HTML5

Relatório de Desenvolvimento de Jogos com HTML5

Paulo Vitor Freitas da Silva

1 – Introdução

O jogo desenvolvido possui mecânica de *sidescrolling*, em que o cenário se move o tempo todo, com velocidade constante. Nele o jogador controla um carro, inicialmente centrado na tela. Esse carro possui um canhão que aponta na mesma direção do ponteiro do mouse. O objetivo é derrotar carros inimigos que chegam por trás, mirando e atirando neles, e desviar de pedras que aparecem aleatoriamente.

2 – Gênero

O jogo se apresenta no estilo *Arcade*, devido à sua mecânica simples e a presença de um único nível, cujo objetivo é apenas sobreviver o maior tempo possível. Além disso os controles são simples: teclas W, A, S e D para movimentar o carro, ponteiro do mouse para mirar o canhão, e botão esquerdo do mouse para atirar.

3 – Experiência e divertimento

A experiência a ser transmitida para o jogador é simples: ele precisa ter raciocínio rápido e estar atento a todo o cenário para desviar das balas inimigas e das rochas adiante para garantir a sobrevivência. Muitos jogos atualmente seguem a dinâmica da quebra de recordes e da sobrevivência, como é o caso de títulos como Flappy Bird, Subway Surfers, entre outros. Nesses jogos o objetivo é chegar cada vez mais longe.

Pretende-se garantir a diversão do usuário ao mantê-lo entretido enquanto ele tenta elaborar estratégias para sobreviver e aprende a mecânica do jogo.

4 – Mecânica

A mecânica do jogo é simples: com as teclas W, A, S e D o jogador pode mover o carro pelo cenário. O canhão do jogador aponta sempre para a posição do ponteiro do mouse. Inicialmente são inseridos três carros inimigos em posições aleatórias, começando sempre no lado esquerdo da tela e fora dela. Ao aparecer na tela, os carros inimigos desaceleram e começam a atirar no jogador. Para cada inimigo, os tiros são feitos num intervalo entre 1,5 e 3 segundos. O jogador pode atirar à vontade, mas sempre que ele ou um inimigo é atingido, há um tempo de 1 segundo em que eles ficam inatingíveis até poderem ser acertados novamente. Cada tiro recebido retira 10% da vida. Quando todos os três inimigos são derrotados, são criados três novos, e o processo se repete.

Além das balas inimigas, o jogador deve desviar o carro de pedras que são dispostas aleatoriamente no eixo Y e inicialmente fora da tela, no canto direito. O intervalo de inserção

entre as pedras é aleatório: entre 1,5 e 3 segundos. Quando uma perda é criada no cenário, um aviso aparece para o jogador no canto direito: uma pedra menor fica piscando na mesma posição Y da pedra que está para chegar.

O jogo computa o tempo que o jogador conseguiu manter-se vivo e o número de inimigos derrotados. O jogador tem três vidas no início do jogo. A tabela a seguir traz a lista de comandos via teclado e mouse.

Ação	Efeito
Tecla W	Tecla direcional para cima
Tecla A	Tecla direcional para esquerda
Tecla S	Tecla direcional para baixa
Tecla D	Tecla direcional para direita
Mover mouse	Mira canhão
Botão esquerdo do mouse	Atira

Tabela 1: Lista de comandos do jogo

5 – Elementos de design

Para compor os *sprites* e cenário do jogo, alguns desenhos tiveram que ser criados. Usou-se o programa para edição de imagens vetoriais CorelDRAW para criação desses *sprites*. Os carros (tanto do jogador quanto dos inimigos) foram desenhados com 10 *frames* diferentes: 5 para luz de freio apagado, e 5 para freio aceso, e em cada *frame* as rodas estão numa posição diferente para dar a sensação de rotação delas. Os *frames* com luz de freio apagada foram agrupados na primeira linha do arquivo de imagem, e o restante na segunda linha. As imagens a seguir mostram esses arquivos.

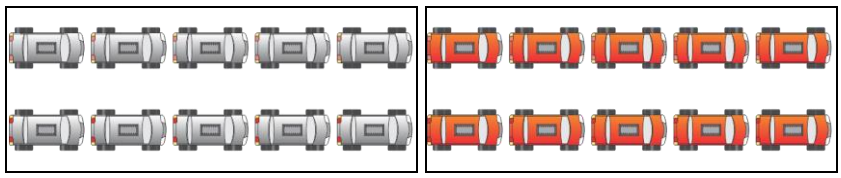

Figura 1: Textura para os carros do jogador e dos inimigos, respectivamente

O canhão foi desenhado com apenas dois *frames*: No primeiro o canhão está ocioso, apenas mirando, e no segundo ele está atirando, possuindo uma pequena representação de chamas em sua ponta. O último elemento que foi criado especialmente para o jogo foi a bala do canhão, esta em *frame* único.

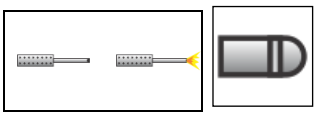

Figura 2: Textura do canhão e da bala de canhão

As outras imagens foram retiradas de fontes abertas na internet. O cenário é um chão de terra, cuja textura se move constantemente e é repetida lado a lado ao fundo, dando a sensação de *loop*. As pedras também possuem *frame* único, e buscou-se uma textura que combinasse com o chão de terra escolhido. Por último, é usado um arquivo com 16 *frames* para efeito de explosão, usados quando um inimigo ou o jogador é derrotado.

Figura 3: Texturas para o cenário [1], pedras [2] e frames de explosão [3], respectivamente

6 – Efeitos sonoros

Alguns efeitos sonoros foram inseridos no jogo, todos retirados do repositório público da OpenGameArt.org. O efeito de tiro [4] é inspirado nos jogos antigos de 8 bits. Os outros efeitos inseridos foram o de explosão [5] e o de motor de carro [6] que fica em *loop*, sendo reproduzido o tempo todo.

7 – Implementação

O código foi organizado em vários arquivos de *JavaScript,* listados a seguir:

- *AudioResources.js:* responsável pelo carregamento e gerenciamento da biblioteca de áudio;
- *Cenario.js:* responsável pelo gerenciamento e desenho do cenário e das pedras;
- *Constantes.js*: contém as constantes usadas no jogo, entre elas o *fps* usado, a fração de tempo *dt*, os tempos mínimos entre a criação de pedras e dos disparos dos inimigos, entre outras;
- *Informacoes.js:* contém funções para desenho de informações na tela, incluindo tempo de jogo, número de inimigos derrotados e número de vidas restantes;
- *Inimigos.js:* contém funções ara criação e gerenciamento dos inimigos;
- *Minimapa.js:* específico para desenho do mini mapa;
- *Mouse.js*: contém funções para tratar eventos de mouse;
- *Teclado.js:* contém funções para tratar eventos de teclado;

O tamanho da tela de jogo foi fixado em 800x600, podendo ser facilmente alterada dentro do código, exigindo poucas adaptações nas funções de desenho, uma vez que buscou-se implementar as disposições dos elementos em função do tamanho do *canvas*. No entanto deve-se observar que o redimensionamento do *canvas* não altera a escala dos *sprites* no jogo.

7.1 – Execução principal e passo

O arquivo HTML principal, que contém o *script* inicial, começa definindo três conjuntos vazios, a saber: um para manter e outro para excluir tiros, e um para manter e outro para excluir inimigos. Isso permite que sejam carregados tiros e inimigos nesses conjuntos enquanto houver memória disponível. Da forma como foi definido, o jogo mantém no máximo três inimigos no conjunto, embora seu tamanho não tenha sido fixado. Os conjuntos usados para exclusão mantêm aqueles objetos que devem ser apagados da memória: no caso dos tiros, quando um deles sai da dela ou atinge um inimigo ou o jogador, e no caso dos inimigos quando um deles é derrotado. No mesmo arquivo, o personagem (carro) do jogador é criado com três vidas e 100% de saúde.

O jogo foi programado para executar a função *Passo* a cada intervalo de *frame*. Essa função é a responsável por disparar os métodos para redesenho, para verificação das restrições (abordadas mais adiante), para movimentação dos elementos do jogo, e para verificação de colisões dos inimigos e do jogador com as balas disparadas. Caso seja detectada colisão, a própria função de *Passo* computa o dano causado e trata a exclusão dos objetos envolvidos da memória. Na mesma função são criados inimigos novos caso todos tenham sido derrotados.

7.2 – Constantes

A maioria das constantes foi separada no arquivo *Constantes.js*, facilitando assim a manutenção e teste do jogo. É o segundo arquivo a ser carregado pelo HTML principal. Entre as constantes definidas estão: o *fps* (*frames* por segundo), fixado em 60; a fração de tempo *dt*; os tempos mínimos entre a criação de pedras e dos disparos dos inimigos (ambos definidos em 1,5 segundos); o intervalo entre os *frames* de animação (0,025 segundos), a velocidade do cenário (600 pixels por segundo); a aceleração do personagem quando acionada uma das teclas de movimento (1000 pixels/seg^2 em todas as direções); o raio dos *sprites* dos carros (60 pixels), do canhão (45 pixels), e das pedras (60 pixels); as posições iniciais do jogador (x = -100, y = centro na vertical) e dos inimigos (x = -500, y é aleatório no momento da criação); os caminhos dos arquivos de imagem e de som; entre outras.

7.3 – Sprites

As principais funções relacionadas aos *sprites* do jogador, dos inimigos e das balas foram agrupadas no arquivo *Sprites.js.* Um *Sprite* é um objeto, que possui as seguintes informações essenciais: posição x e y, raio, velocidade no eixo x, velocidade no eixo y, aceleração no eixo x, aceleração no eixo y, rotação, saúde, e as variáveis de controle de desenho que indicam a coluna e a linha do arquivo de animação de explosão, e a coluna do arquivo de animação dos carros. Além disso há a variável que indica quando o jogador ou o inimigo foi atingido. Essa variável, denominada *danificado*, é temporal, sendo decrementada enquanto o jogo estiver animando o efeito de dano recebido. Por fim, tem-se as variáveis para animação do canhão: rotação do canhão, a variável booleana que diz se o jogo deve mostrar o *frame* de "atirando" do canhão e a variável de tempo que diz a quanto tempo o *frame* de "atirando" está sendo mostrado, a fim de interromper a animação quando tempo limite definido nas constantes for atingido.

Os protótipos de funções usados no objeto *Sprite* são listados a seguir:

- **Sprite.prototype.restaurar**: usada quando o jogador perde uma vida e ainda tem vidas para gastar. Restaura o valor de saúde e as variáveis de controle de animação, sem mudar a posição e a aceleração atuais.
- **Sprite.prototype.atirar**: usada tanto pelo jogador, quando ele clica com o botão esquerdo do mouse, quanto pelos inimigos, quando sorteado para atirar dentro do intervalo pré-definido. Essa função verifica se a saúde do atirador está positiva e se o jogador ainda possui vidas, tornando-o apto a atirar. Assim é criado um novo tiro e inserido no conjunto de tiros mencionado anteriormente. Ao criar-se um novo tiro com essa função, a função desenho dele é redefinida, sendo diferente do desenho usado nos outros *Sprites*: ela é mais simples, apenas desenhando na tela o arquivo de imagem do tiro, respeitando a sua rotação. A função de movimento também é redefinida: nela, não existe o fator de tração que limita a velocidade do *Sprite*. É criada também a função *foraDaTela*, que auxilia o teste de restrição, retornando *true* caso a bala tenha ultrapassado os limites da tela, a tornando candidata a ter seu objeto excluído da

memória, e *false* caso contrário. Por último, o raio do tiro é definido em 10, suas posições em x e y (iguais ao do atirador), sua rotação (igual a rotação atual do canhão), sua força (quantidade de saúde que retira do alvo atingido), e por fim é definida uma variável booleana que diz se o tiro saiu de um inimigo ou do jogador (isso é importante para fazer teste de colisão). A animação de tiro então é habilitada para o *Sprite*, e toca-se o som de tiro no final do processo.

- **Sprite.prototype.mover**: recalcula as posições x e y do *Sprite*, em função da sua velocidade e aceleração. Aqui, a velocidade é limitada por um valor de tração, definido em 0.07. Além do movimento nos eixos x e y, a rotação do *Sprite* também é recalculada em função da posição atual do ponteiro do mouse.
- **Sprite.prototype.desenhar**: função de desenho do *Sprite*, faz a verificação de alguns valores para decidir o que desenhar ou animar. Primeiro, se a aceleração no eixo x for negativa (o carro está freando), a função deve desenhar a segunda linha do arquivo de imagens do carro. Depois, se a saúde do indivíduo for zero, significa que a animação de explosão deve ser executada, assim como o efeito sonoro. Se não for o caso, mas o indivíduo tiver sofrido dano, ele deve piscar por algum tempo antes de ser atingível novamente. Uma barra de saúde é desenhada logo acima do carro, indo do verde ao vermelho dependendo do nível dela. Por fim, o canhão é desenhado com um ângulo de acordo com a posição do mouse. Sempre que essa função faz um desenho, ela executa translação e rotação para o objeto a ser desenhado. A rotação do carro está em função da sua aceleração no eixo y.
- **Sprite.prototype.restricoes**: verifica se o objeto está atingindo os limites da tela, não permitindo que ele as ultrapasse, exceto quando o objeto já começa fora da tela e acelera até aparecer.
- **Sprite.prototype.colidiuComCircular**: verifica colisão circular simples entre o objeto e um alvo passado por parâmetro.

7.4 – Cenário

O cenário, gerenciado pelo *script* do arquivo *Cenario.js,* é um objeto com funções de desenho simples: ele simplesmente desenha a imagem de fundo repetidamente, em *loop*. Seu destaque está na criação de pedras das quais o jogador deve desviar para não sofrer danos. A cada criação de pedra, o cenário sorteia um tempo entre 1,5 e 3 segundos para criar a próxima. Um objeto de pedra possui apenas suas posições em x e y, seu raio, e sua velocidade em x, que é a mesma velocidade do plano de fundo. Ao desenha uma pedra, o jogo desenha apenas um pequeno objeto dela no canto direito, fazendo-o piscar, caso a pedra esteja fora dos limites da tela, na direita. Esse efeito serve como aviso ao jogador do obstáculo se aproximando. O movimento da pedra é apenas no eixo x, portanto sua função de movimento considera apenas esse eixo.

O mesmo esquema de exclusão de inimigos e tiros é adotado no cenário para limpeza de memória: é mantido um conjunto de pedras a serem apagadas quando já passaram pelo cenário. Essa exclusão é tratada na função de restrições do cenário, assim como a criação das pedras. As pedras são criadas em posições aleatórias no eixo y.

7.5 – Mini mapa

Para expandir a visibilidade do jogador, foi inserido um mini mapa no canto inferior direito do jogo. Esse mini mapa, gerenciado pelas funções do arquivo *Minimapa.js*, desenha objetos primitivos, como retângulos e círculos, para ilustrar a posição do jogador, dos inimigos,

e das pedras. O mini mapa mostra objetos numa área maior da mostrada pelo cenário: 400 pixels para trás e 400 pixels para frente. Isso ajuda o jogador a visualizar a posição dos inimigos e obstáculos se aproximando. Para desenhar o mini mapa, é aplicada uma operação de *scale* sobre o *canvas*, e então os objetos são desenhados com as mesmas posições x e y originais, sem necessidade de conversão.

7.6 – Inimigos

Durante o jogo, a função *insereInimigos* é chamada sempre que não há mais inimigos para o jogador enfrentar. Essa função cria três novos inimigos em posições aleatórias na posição y, e fora da tela. A criação de um inimigo é feita pela função *criaInimigo*, presente no arquivo *Inimigos.js*. Um inimigo é um *Sprite*, definido anteriormente. A diferença está no tempo aleatório definido entre um disparo e outro (entre 1,5 e 3 segundos), na definição aleatória de sua posição em y. Além disso, os métodos de restrições e de movimento são redefinidos. As restrições dos inimigos atualizam sua aceleração e velocidade quando entram na tela, e também efetuam o disparo quando o tempo anteriormente definido é atingido e então define o tempo para o próximo disparo. Já a função de movimento possui uma tração de 0,05, diferente da 0,07 usada para o jogador.

7.7 – Mouse e teclado

Os arquivos *Mouse.js* e *Teclado.js* tratam os eventos de mouse e teclado respectivamente. A função *atualizaPosicaoMouse* é dispara quando o jogador move o ponteiro, atualizando então a posição x e y dele, para que o canhão do jogador seja desenhado num ângulo tal que o canhão aponte para ponteiro. O cálculo desse ângulo é feito da seguinte forma:

$$rotacao = \begin{cases} atan\left(\frac{pc.y - mouse.y}{pc.x - mouse.x}\right), se\ mouse.x \geq pc.x \\ atan\left(\frac{pc.y - mouse.y}{pc.x - mouse.x}\right) + \pi, c.c. \end{cases}$$

A segunda condição garante que a rotação fará o canhão sempre apontar para o sentido certo caso o ponteiro do mouse esteja antes da posição do personagem. O botão esquerdo do mouse faz o personagem atirar.

Em relação ao teclado existem dois eventos sendo tratados: tecla pressionada e tecla solta. Ao pressionar uma das teclas direcionais (W, A, S, D) ao jogo atualiza a aceleração no sentido correspondente, de acordo com as constantes definidas no arquivo *Constantes.js*. Ao soltar uma das teclas, o jogo zera a aceleração.

7.8 – Scores e outras informações

O arquivo *Informacoes.js* mantem o tempo de jogo e o número de inimigos derrotados. Nesse arquivo está presente uma única função, *desenhaInfos*, que mostra no canto superior esquerdo o tempo de jogo (e também atualiza ele), o número de inimigos derrotados e o número de vidas restantes, esse representado através de pequenos carros desenhados iguais ao carro do jogador. Quando o jogador não possui mais vidas, essa função para a contagem de tempo e desenha “Game over” na tela.

7.9 – Áudio

Os efeitos sonoros são gerenciados pelas funções presentes no arquivo *AudioResources.js*. Ao criar um *AudioResource* (ou uma biblioteca de sons), define-se a quantidade de canais a serem usadas no jogo. Essa quantidade foi fixada em 5. Nessa biblioteca são carregados os sons de explosão, de tiro e de motor ligado. O som de motor é um som de fundo, portanto foi necessária a criação de uma função especial para reproduzi-lo: *playBackground*, que toca o som em *loop*. Já a função de *play*, usada para tocar os outros efeitos, verifica se o som a ser tocado já não está sendo reproduzido, e procura um canal disponível para reproduzi-lo.

8 – Conclusões e Melhorias futuras

Foi apresentada a descrição da implementação do jogo proposto para a disciplina de seminário em desenvolvimento de jogos em HTML5. Aqui propõe-se algumas alterações e melhorias que podem ser implementadas no jogo:

- Itens: alguns podem ser dispostos pelo cenário, incluindo itens para aumento de vida e saúde do jogador;
- Armas diferentes: alguns tiros diferentes podem ser adquiríveis através da coleta de itens, por exemplo. Entre alguns tiros possíveis estão tiros duplos, triplos, *lasers*, bombas, entre outros;
- Ataques inteligentes dos inimigos: os inimigos poderiam ter algumas frentes de ataque diferentes, em que eles são dispostos em posições e quantidades diferentes, além da aplicação de inteligência artificial neles para desviar das pedras, por exemplo;
- Níveis de dificuldade: o jogo apresentado possui dificuldade constante. Uma versão futura poderia inserir elementos que aumentem a dificuldade, à medida que o jogador também consiga aumentar seu poder de fogo;
- Outros tipos de inimigos: helicópteros, aviões, bombas sendo lançadas, etc;

Referências

Arquivos de textura:

[1] http://2.bp.blogspot.com/-UTV6xMDLlzk/TtENL3YbYZI/AAAAAAAAAFE/wzG3veFt2EI/s1600/Dirt+cracked+00+seamless.jpg

[2] https://s-media-cache-ak0.pinimg.com/236x/91/05/af/9105af1e9bab92e1c176553e8a65f767.jpg

[3] http://opengameart.org/content/explosion

Arquivos de efeitos sonoros:

[4] http://opengameart.org/content/gunloop-8bit

[5] http://opengameart.org/content/explosion-0

[6] http://opengameart.org/content/car-engine-loop-96khz-4s