Relatório do Exercício Multimédia Interativo - Desporto Olímpico P5.js - (Processing)

Engenharia da Computação Gráfica e Multimédia

Sistemas Multimédia

2023-2024

Exercício Realizado por: Patrícia Pereira, nº 22304

Objetivo e Descrição Sumária do Exercício

O objetivo deste exercício multimédia interativo é criar uma experiência de jogo interativo baseada num desporto olímpico à escolha do aluno. Nesse jogo, os jogadores têm como objetivo atingir uma pontuação máxima através da interação com os objetos visuais no ecrã. O jogo requer habilidade e precisão, pois os jogadores devem acertar alvos específicos para marcar pontos. Além disso, o jogo apresenta um desafio de tempo, uma vez que os jogadores têm um número limitado de tentativas para marcar pontos. Este exercício faz uso de recursos multimédia, como áudio e elementos visuais, para proporcionar uma experiência de jogo envolvente.

Funcionalidades Implementadas/Desenvolvidas

1. Escolha do Desporto Olímpico: O exercício é baseado num desporto olímpico escolhido pelo aluno. O desporto escolhido foi o tiro com arco.

2. Controlo do Jogo: O jogador controla a interação com o jogo por meio de áudio (microfone) e o rato. O uso do microfone permite que o jogador dispare as flechas no jogo com base nos níveis de som detetados e o cursor é usado para apontar a mira nos alvos. Ao atingir o 'volumeLimite' a flecha é lançada. O limite está definido em 0,5 (escala que vai de 0 a 1, onde 0 é a ausência de som e 1 é o volume máximo possível).

3. Pontuação e Desafio de Tempo: O jogador é desafiado a marcar pontos atingindo alvos. Cada alvo atingido adiciona pontos à pontuação do jogador. No entanto, o jogador tem um número limitado de 10

tentativas (flechas) antes de o jogo terminar. O tempo desempenha um papel importante na experiência do jogador, já que ele precisa tomar decisões rápidas para maximizar sua pontuação.

A pontuação atribuída difere dependendo da zona do alvo acertada:

- Parte externa- 1 ponto;
- parte intermédia-5 pontos;
- Parte interna-10 pontos;

Se o jogador não acertar no alvo perde 2 pontos.

```
getPoints(x, y) {
    let d = dist(x, y, this.x, this.y);
    if (d < this.raioTotal - 60) {
      return 10;
    } else if (d < this.raioTotal - 30) {
      return 5;
    } else {
      return 1;
    }
  }
}
```

```
if (volume > volumeLimite && currentTime - lastShotTime >= cooldown) {

      if (target.checkCollision(mouseX, mouseY)) {

        let points = target.getPoints(mouseX, mouseY);
        score += points;
        ...
      } else {
        score -= penalizacao;
        ...
      }
     ...
    }
```

4. Elementos Aleatórios: O jogo inclui elementos de aleatoriedade controlados programaticamente, como a posição dos alvos no ecrã. Isso garante que cada sessão de jogo seja única e desafiadora.

```
function newTarget() {
  // Gera coordenadas x e y aleatórias dentro das áreas específicas
  let x = random(200, 700);
  let y = random(200, 400);
```

```
  ...
  }
class Alvo {
  constructor(x, y) {
   ...
    this.raioTotal = random(20, 100); // Gera um raio aleatório entre 50 e 100
  }
```

5. Colisões entre Objetos: O jogo implementa colisões entre os objetos no ecrã. Isso permite que o jogador determine se atingiu ou não um alvo, afetando a pontuação.

```
checkCollision(x, y) {
    let d = dist(x, y, this.x, this.y);
    return d < this.raioTotal;
  }
```

6. Reprodução de Sons: Sons, como efeitos sonoros e música de fundo, são incorporados ao jogo para melhorar a experiência do jogador. Os sons são reproduzidos em resposta a ações do jogador, como acertar ou errar um alvo.

```
function preload() {
  soundAlvo = loadSound('seta2.mp3'); // som quando se acerta o alvo
  soundErro = loadSound('erro.mp3'); // som quando se falha o alvo
 ...

}
```

7. Ecrãs de Jogo: O exercício possui 3 ecrãs principais: o ecrã inicial, que representa o ponto de entrada para inicia o jogo. O ecrã de jogo, onde o jogador interage com os elementos do jogo, e o ecrã de "Repetir", que permite ao jogador reiniciar o jogo após uma rodada e que mostra a pontuação obtida na rodada atual.

Detalhes Relevantes da Implementação

Estrutura do Código

Claro, a estrutura do código é uma parte fundamental de qualquer projeto de programação. Vou explicar a estrutura do código em termos de classes e sua organização geral.

Classes:

1. Classe Alvo:

- Esta classe é responsável pela criação e renderização dos alvos no jogo.

- Ela possui um construtor que define as propriedades de um alvo, como posição e tamanho (raios).

- O método 'display' é usado para desenhar os alvos no ecrã com diferentes cores.

- Os métodos 'checkCollision' e 'getPoints' são usados para verificar se um tiro acertou o alvo e calcular os pontos a serem atribuídos com base na distância entre o tiro e o alvo.

2. Funções Globais:

- 'preload': Carrega os recursos do jogo, como sons e imagens, para uso posterior.

- 'setup': Define as configurações iniciais, como o tamanho do ecrã e inicializa o microfone.

- 'draw': É o ciclo principal do jogo, onde os diferentes estados (ecrã inicial, ecrã de jogo, ecrã de reinício) são controlados.

- 'touchStarted': Lida com o toque no ecrã, reiniciando o jogo.

- 'newTarget': Gera um novo alvo aleatório.

Estrutura Geral:

O código segue um modelo baseado em estados, onde há três estados principais:

1. Ecrã Inicial (currentScreen === 0):

- No ecrã inicial, o jogador vê uma imagem de fundo, que indica como começar o jogo.

```
image(fundoInicial, 0, 0, 1030, 518); // exibe a imagem de fundo
```

- Quando o jogador pressiona a tecla "Enter", o estado muda para 1, que é o ecrã do jogo.

```
if (keyIsPressed && key === 'Enter') {
      currentScreen = 1;
      cursor(); // mostra o cursor padrão
      newTarget(); // cria um novo alvo
    }
```

2. Ecrã do Jogo (currentScreen === 1):

- O ecrã do jogo apresenta um ambiente onde o jogador pode interagir com alvos.

- A interação é baseada no volume capturado pelo microfone.

```
let volume = mic.getLevel();
...
if (volume > volumeLimite && currentTime - lastShotTime >= cooldown) {
   …
}
```

- A pontuação é exibida no canto superior esquerdo, o número de flechas restantes no canto superior direito e informações de autoria no canto inferior direito.

```
// Mostra a pontuação no canto superior esquerdo
    textSize(textSizeOriginal);
    textAlign(textAlignOriginal);
    fill(0);
    text("Pontuação: " + score, 20, 30);
    // Mostra o número de flechas restantes no canto superior direito
    text("Flechas Restantes: " + flechasRestantes, width - 260, 30);
    // Texto no canto inferior direito
    textSize(16);
    text("Patrícia Pereira, 22304, ECGM", width - 250, height - 20);
```

- Os tiros do jogador são representados como círculos.

```
// Desenha os tiros no alvo
    fill(0);
    for (let hit of hits) {
      ellipse(hit.x, hit.y, 10, 10);
    }
```

- Se a pontuação atingir zero ou o número de flechas restantes se esgotar, o estado muda para 2, o ecrã de reinício.

```
if (flechasRestantes <= 0 || score < 0) {
      currentScreen = 2;
}
```

- O rato é representado pela imagem de um arco.

```
cursor('none');
image(arcoImg, mouseX - 50, mouseY - 50, 100, 100); // O arco é centralizado no
                                                    cursor
```

3. Ecrã de Reinício (currentScreen === 2):

- O ecrã de reinício pode exibir diferentes imagens com base na pontuação do jogador. Se a pontuação for maior que zero, o jogador vê uma imagem de vitória; caso contrário, uma imagem de derrota.

```
if (score > 0) {
      // Caso a pontuação seja maior que 0
      image(imagemVencedor, 0,0, width, height);
      text("Pontuação: " + score, width/2, height/2 + 40)

    } else {
      // Caso a pontuação seja menor ou igual a 0
      image(imagemPerdedor,  0,0, width, height);
      text("Pontuação: 0", width/2, height/2 + 40)

    }
```

- A pontuação é exibida no centro do ecrã.

```
text("Pontuação: " + score, width/2, height/2 + 40)
```

- O jogador pode clicar para reiniciar o jogo.

```
function touchStarted() {
  // Verifica se o ecrã atual é a de reinício
  if (currentScreen === 2) {
    // Redefine a pontuação, flechas restantes e muda para o ecrã de jogo
    score = 0;
    flechasRestantes = 10;
    currentScreen = 1;
    redraw(); // Redesenha o ecrã para atualizar as alterações
  }
}
```

Exemplo de Funcionamento das Colisões

A deteção de colisões entre o cursor do rato (ou posição do som) e os alvos é essencial para o funcionamento do jogo. Isso é feito comparando a distância entre o ponto de colisão potencial e o centro do alvo. Se a distância for menor que o raio do alvo, consideramos uma colisão.

```
checkCollision(x, y) {
    let d = dist(x, y, this.x, this.y);
    return d < this.raioTotal;
}
```

Exemplo do funcionamento dos alvos

Os alvos são os elementos que os jogadores tentam acertar para ganhar pontos. Aqui está um exemplo de como os alvos são criados e exibidos:

```
function newTarget() {
  // Gera coordenadas x e y aleatórias dentro das áreas específicas
  let x = random(200, 700);
  let y = random(200, 400);

  // Cria um novo objeto 'Alvo' com as coordenadas geradas
  target = new Alvo(x, y);

  // Regista o tempo atual (em milissegundos) para controlar o intervalo desde o
último alvo
  lastAlvoTime = millis();
}
```

Neste trecho de código, criamos um novo alvo com posições (x, y) aleatórias dentro de uma área específica no ecrã. Isso adiciona um elemento de aleatoriedade ao jogo, tornando cada tentativa única.

Os alvos são exibidos no ecrã através do método display() da classe Alvo.

```
function draw() {
...
   target.display();
...
}
```

Dentro do método display(), os alvos são desenhados no ecrã com diferentes cores e tamanhos, criando uma aparência visual distintiva para cada alvo.

```
display() {
    noStroke();
    let raioTotal = this.raioTotal;
    let raio1 = raioTotal;
    let raio2 = raioTotal - raioTotal * 0.3; // 30% do raioTotal
    let raio3 = raioTotal - raioTotal * 0.6; // 60% do raioTotal
    fill(color(255, 0, 0));
    ellipse(this.x, this.y, raio1 * 2);
    fill(color(255, 255, 255));
    ellipse(this.x, this.y, raio2 * 2);
    fill(color(0, 0, 255));
    ellipse(this.x, this.y, raio3 * 2);
```

Exemplo de funcionamento dos hits(tiros)

Os hits representam os tiros disparados pelos jogadores quando tentam atingir os alvos. Aqui está um exemplo de como os hits funcionam:

```
if (volume > volumeLimite && currentTime - lastShotTime >= cooldown) {

      if (target.checkCollision(mouseX, mouseY)) {
         let points = target.getPoints(mouseX, mouseY); // calcula os pontos obtidos com base na
                                                posição do cursor em relação ao alvo.            score +=
points;
        hits.push({ x: mouseX, y: mouseY, timer: millis() }); // Regista o hit (tiro bem-sucedido)
                                                                atual, armazenando a posição e
                                                      tempo do hit.
        soundAlvo.play();
      } else {
        score -= penalizacao;
        soundErro.play();
      }
      flechasRestantes--;  // Decrementa o número de flechas restantes.
      lastShotTime = currentTime;    // Atualiza o tempo do último disparo para o tempo atual.
    }
```

Neste excerto, estamos a verificar se o volume do microfone é superior a um limite específico e se já passou tempo suficiente desde o último tiro (para evitar tiros muito rápidos). Se essas condições forem atendidas, verificamos se o cursor do rato (ou posição do som) colide com o alvo atual. Se houver uma colisão, o jogador ganha pontos com base na proximidade do tiro ao centro do alvo. Se não colidir é penalizado.

Os hits são representados por objetos que incluem as coordenadas (x, y) do tiro e um temporizador para rastrear quando o tiro ocorreu. Eles são armazenados num array chamado hits, que é usado para exibir tiros anteriores no ecrã.

Aspetos que Poderiam/Deveriam ser Melhorados

1. Melhorias Gráficas: O jogo pode ser aprimorado com gráficos mais elaborados, animações e uma interface mais atraente.

2. Variedade de Níveis: Adicionar níveis progressivamente mais desafiadores ao jogo para manter o interesse do jogador.

3. Instruções do Jogo: Incluir um tutorial ou instruções claras no início do jogo para orientar novos jogadores.

4. Efeitos de Som Avançados: Adicionar mais efeitos sonoros, música de fundo e áudio direcional para aprimorar a imersão do jogador.

5. Personalização Avançada: Permitir que o jogador personalize elementos do jogo, como a escolha do desporto olímpico ou a aparência dos alvos.

6. Estatísticas e Registos: Implementar um sistema de classificação e registo das pontuações mais altas dos jogadores.

7. Testes e Otimização: Realizar testes extensivos para garantir que o jogo funcione sem problemas em diferentes plataformas e dispositivos.

Conclusão

O exercício multimédia interativo baseado num desporto olímpico é uma demonstração de como a tecnologia pode ser usada para criar experiências de jogo envolventes. A interação entre elementos visuais, áudio e jogabilidade cria uma experiência desafiadora e divertida para os jogadores. Embora o exercício atual tenha atingido os objetivos principais, há oportunidades para melhorias e expansões futuras, tornando-o ainda mais envolvente e cativante para o público. O exercício atende aos requisitos obrigatórios estabelecidos e fornece uma base sólida para a exploração contínua de desenvolvimento multimédia interativo.