Universidade Federal de São Paulo

Campus São José dos Campos

Relatório 5: Enduro 3D

Computação Gráfica

Profª. Ana Luísa Lemos

Nomes:

Fernando Bandeira Soares	86281
Magno Lamas Oliveira	77619
Marcos Eduardo Lopes Honorato	86379
Rodrigo Teiji Takaki	71100
Yoji Wadatsumi Kojio	69474

1. Introdução

A produção brasileira de jogos está em ascensão. São várias empresas envolvidas nesse mercado promissor e os empreendedores que se arriscam em investir na criação de um jogo sabem que é caro e que leva muito tempo, mas que esses riscos podem gerar muito lucro.

A contratação de profissionais nessa área vem crescendo e há uma grande procura por pessoas com especialização em linguagens de jogos eletrônicos. Na maioria das desenvolvedoras, a busca é por programadores, designers, ilustradores e produtores executivos.

O objetivo desse projeto é desenvolver um remake do Enduro, jogo clássico do Atari, e mostrar que o desenvolvimento de jogos está muito além de criação e programação dos mesmos.

2. Descrição do jogo

Enduro é um dos jogos clássicos de maior sucesso do console Atari 2600, originalmente criado por Larry Miller, produzido e publicado pela Activision em 1983. Sua mecânica simples e desafiadora traduz-se em manobrar um carro de corrida em um Enduro Nacional, corrida de resistência de longa distância, ultrapassando seus adversários em busca de uma posição no pódio.

Figura 1: Imagem representando o jogo durante o dia

Figura 2: Imagem representando o jogo durante a noite

A jogabilidade de Enduro consiste em uma corrida infinita que se segue por dias (em jogo). O jogador controla um carro de corrida com a câmera fixada na traseira assim como ilustrado na Figura 1, à esquerda, a uma certa distância e altura em relação ao carro e solo, e deve evitar colisões com outros pilotos, buscando a primeira posição na corrida em cada dia, superando mudanças de condições de tempo e clima: quando anoitece, o jogador deixa de enxergar os carros adversários e passa a ver apenas suas luzes traseiras, assim como ilustrado na Figura 2, à direita, quando em neblina, a visibilidade fica prejudicada etc.

O carro ganha velocidade conforme o jogador seleciona, até um limite máximo, podendo desacelerar também. Sua pontuação é baseada na distância percorrida, o número de ultrapassagens necessárias por dia aumenta em cada corrida e as colisões geram uma desaceleração brusca do carro.

2.1 Jogabilidade

O controle do Atari era provido por uma haste analógica e um botão vermelho de comando, como ilustrado na Figura 3. Só era possível a movimentação do carro na horizontal para desviar dos adversários e a aceleração. Sendo assim, no enduro, a haste movimenta o carro para a esquerda e para a direita e o botão vermelho é o responsável pela aceleração.

Figura 3: Controle típico utilizado no console Atari

3. Metodologia

A mecânica do Enduro, como todos os jogos da época, é simples e programado para um ambiente 2D. Pensando em um mundo virtual 3D o planejamento, programação e implementação são diferentes e mais difíceis.

O projeto manteve foco na principal característica dos jogos da época, diversão infinita. Ordenando as ideias, o grupo decidiu iniciar com uma pista simples, mantendo a mecânica e física como são no jogo original, utilizando e seguindo exercícios e exemplos feitos no decorrer do curso. Para tanto, era necessário criar os carros para testes visuais e iniciais da mecânica.

3.1 Divisão das Tarefas

Inicialmente foi decidido que o melhor seria uma divisão parcial das tarefas. As tarefas com maior dificuldade seriam discutidas, revisadas e divididas entre todos durante os processos de execução. No decorrer do projeto, mais precisamente na fase alpha (3º relatório), notou-se conflitos de implementação de ideias e, por isso, perdia-se muito tempo revisando os códigos em busca de erros e adaptações. Assim, viu-se necessário uma figura central para organizar as ideias e implementá-las.

Fernando:	Mecânica, programação da pista, animações dos carros e organização e implementação das ideias;
Magno:	Cenário, mecânica, física, inspeção e revisão dos códigos;
Eduardo:	Processo criativo (design da pista), tela inicial, relatório, testes;;
Rodrigo:	Processo criativo (design da pista e carros), programação do carro, mecânica, adversários, iluminação, relatórios e revisão dos códigos;
Yoji:	Colisões, sistema de pontuação e revisão dos códigos;

3.2 Criação do(s) carro(s)

Partindo de dimensões de um carro real, foram feitos rascunhos de um carro (exemplo foto 1, à esquerda), baseando-se no carro de um cartaz oficial do jogo, divulgado por sua publicadora Activision (figura 4, à direita).

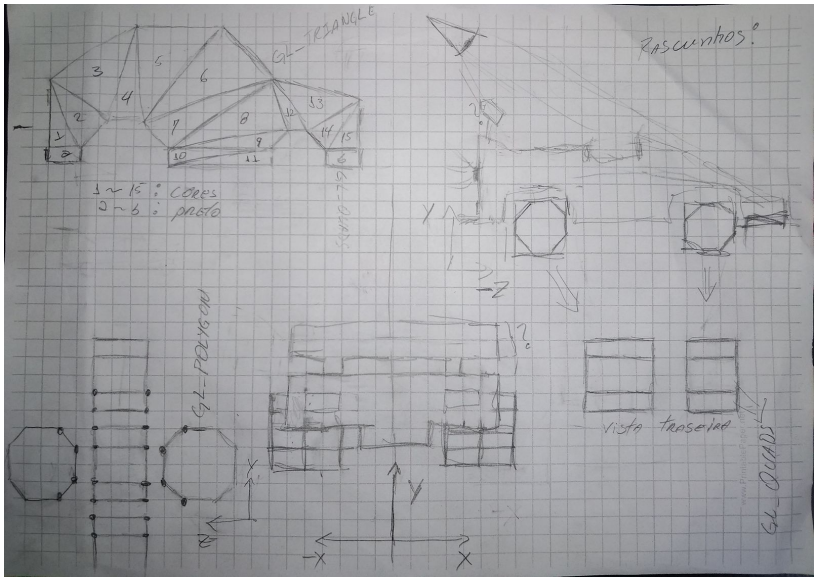

Foto 1: Um dos rascunhos para iniciar o projeto do carro

Figura 4: Cartaz oficial do jogo

Em seguida, foi utilizado papel quadriculado para desenhar cada componente, com vistas de diferentes ângulos, que formará o carro, facilitando na obtenção das coordenadas de cada ponto de cada figura geométrica que compõem cada parte do carro.

Em posse das coordenadas, foi criado cada componente em uma ordem pré-definida anteriormente, utilizando-se de uma estratégia de cor aleatória para cada figura geométrica, para verificar possíveis ajustes nas coordenadas dos objetos criados (como ajuste de sobreposição, por exemplo).

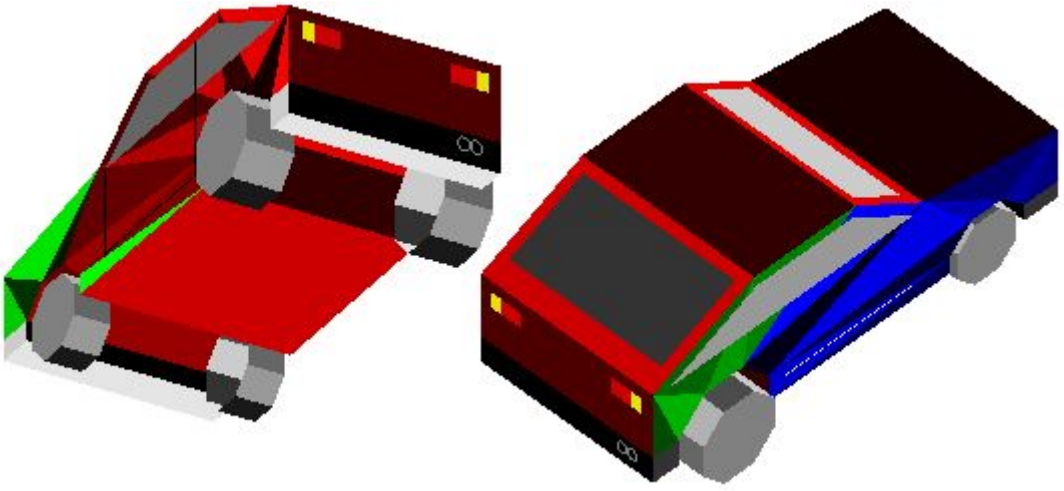

Figura 5: Técnica de cor aleatória para checagem de erros de posição

Para colorir a lateral do carro foi preciso dividir em vários triângulos, numerados no rascunho (foto 1, canto superior esquerdo). No arquivo “carro.c”, cada parte do código do carro está comentado, incluindo a numeração do rascunho, como é mostrado em negrito no quadro 1.

```
void DesenhaCarro(GLfloat *cor){
   glPushMatrix();
...
   //Laterais
   glColor3fv(cor);
   glBegin(GL_TRIANGLES);
      //Esquerda
...
      //Direita
      glNormal3f(1.0, 0.0, 0.0);
      glVertex3f(8.5, 5.4, 0.0);   glVertex3f(8.5, 5.4, -2.5);   glVertex3f(8.5, 11.6, 0.0);//01
      glVertex3f(8.5, 5.4, -2.5);  glVertex3f(8.5, 8.1, -5.1);   glVertex3f(8.5, 11.6, 0.0);//02
      glVertex3f(8.5, 8.1, -5.1);  glVertex3f(8.5, 16.2, -10.0); glVertex3f(8.5, 11.6, 0.0);//03
      glVertex3f(8.5, 8.1, -5.1);  glVertex3f(8.5, 8.1, -8.9);   glVertex3f(8.5, 16.2, -10.0);//04
      glVertex3f(8.5, 8.1, -8.9);  glVertex3f(8.5, 16.2, -20.0); glVertex3f(8.5, 16.2, -10.0);//05
      glVertex3f(8.5, 8.1, -8.9);  glVertex3f(8.5, 11.8, -26.7); glVertex3f(8.5, 16.2, -20.0);//06
      glVertex3f(8.5, 8.1, -8.9);  glVertex3f(8.5, 5.4, -11.5);  glVertex3f(8.5, 11.8, -26.7);//07
      glVertex3f(8.5, 5.4, -11.5); glVertex3f(8.5, 6.6, -31.5);  glVertex3f(8.5, 11.8, -26.7);//08
      glVertex3f(8.5, 5.4, -11.5); glVertex3f(8.5, 4.5, -29.3);  glVertex3f(8.5, 6.6, -31.5);//09
      glVertex3f(8.5, 5.4, -11.5); glVertex3f(8.5, 2.9, -11.5);  glVertex3f(8.5, 4.5, -29.3);//10
      glVertex3f(8.5, 2.9, -11.5); glVertex3f(8.5, 2.9, -29.3);  glVertex3f(8.5, 4.5, -29.3);//11
      glVertex3f(8.5, 6.6, -31.5); glVertex3f(8.5, 6.6, -34.5);  glVertex3f(8.5, 11.8, -26.7);//12
      glVertex3f(8.5, 6.6, -34.5); glVertex3f(8.5, 9.6, -40.0);  glVertex3f(8.5, 11.8, -26.7);//13
      glVertex3f(8.5, 6.6, -34.5); glVertex3f(8.5, 4.5, -36.7);  glVertex3f(8.5, 9.6, -40.0);//14
      glVertex3f(8.5, 4.5, -36.7); glVertex3f(8.5, 4.5, -40.0);  glVertex3f(8.5, 9.6, -40.0);//15
   glEnd();
...
   glPopMatrix();
   glNormal3f(0.0, 1.0, 0.0);
}
```

Quadro 1: Trecho do código do carro

Após a verificação, as cores foram trocadas por uma variável para poder ser determinada em cada situação, vermelho para o principal e outras cores para os adversários, destacando-os do principal.

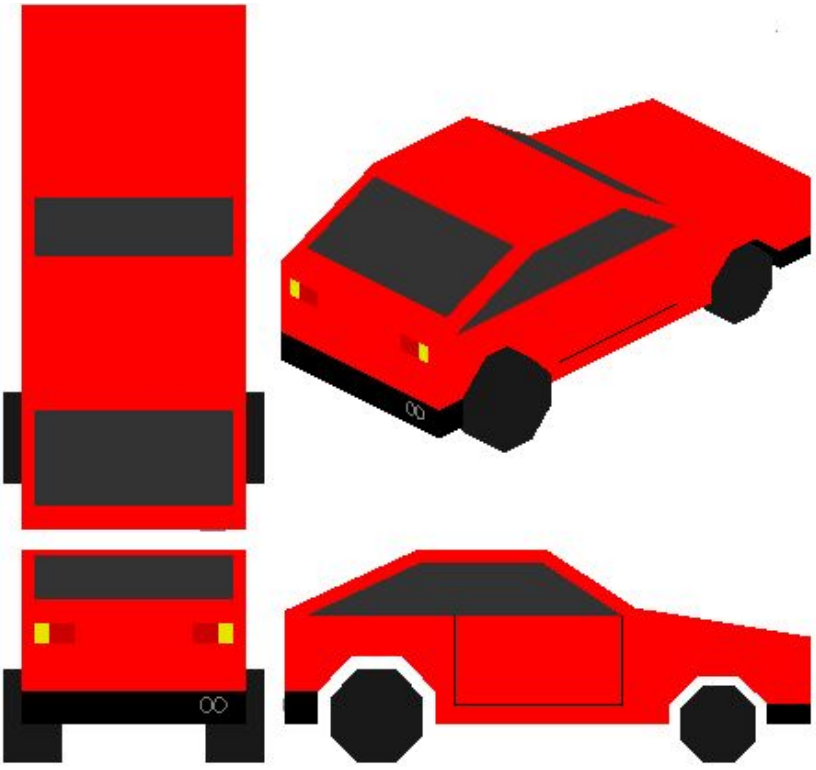

Figura 6: Carro do jogador, vista superior, traseira e lateral direita

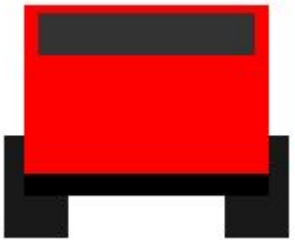

Figura 7: Carro do jogador, vista frontal

Apesar de a câmera ser posicionada na traseira do carro (figura 8), a uma dada distância em relação ao mesmo e o solo, foi criado a parte dianteira sem as lanternas (figura 7), e a base (figura 5 à esquerda) para evitar possíveis problemas durante sua animação.

Figura 8: Carro do jogador, 3 vistas possíveis no jogo, incluindo a animação de manobra

Após o término do carro, foi adicionado o vetor normal para cada face do carro:

-1 em x para a face esquerda;
+1 em x para a face direita;
+1 em y para as partes superioras;
-1 em z para a traseira;
+1 em z para a dianteira;

3.3 Animação do carro principal

Para a animação de deslocamento do carro, foram criadas duas animações, em destaque no quadro 2. Ambas, definidas nas teclas A (esquerda) e D (direita), na função Desenha() em “main.c”:

A variável “carPosX”, utilizada na função “glTranslatef()”, translada o carro no eixo x para criar a animação de deslocamento na horizontal;

A variável “viraCarro”, utilizada nas funções “glRotatef()”, rotaciona o carro para criar a animação de inclinação do carro na curva;

```
void Desenha(){
...
   // Player
   glPushMatrix();
      glTranslatef(carPosX, 0,  -180);
      glTranslatef(0,0,-5);
      glRotatef(viraCarro - Pontos.ponto[pos].curve * 2000, 0, 1, 0);
      glRotatef(-0.5*viraCarro + Pontos.ponto[pos].curve * 2000, 0, 0, 1);
      glTranslatef(0,0, 5);
      glScalef(s_car, s_car, s_car);
      DesenhaCarro(vermelho);
   glPopMatrix();
...
   glFlush();
   glutSwapBuffers();
}
```

Quadro 2: Trecho do código de animação

A animação suave de retorno após a animação de inclinação do carro e a animação de deslocamento do carro estão definidas na função Desenha() em destaque no quadro 3, e suas teclas configuradas como:

- A (esquerda, “botoes[2]”);
- D (direita, “botoes[3]”);
- W (aceleração, “botoes[0]”);
- S (desaceleração, “botoes[1]”);

```
void Desenha(){
...
   // Verifica Teclas:
   if(!colidiu){
      if(botoes[0] && anima){
         pos += (0.12 * speed);
         posBot += 0.05 * speed;
      }
      if((botoes[1] ) && anima){
         pos -= (0.12 * speed);
         posBot += 0.15 * speed;
      }
      if(botoes[2] && !isTouchingLeft()){ //impedir virar pra esquerda quando estiver fora da
pista
         carPosX = carPosX >= -(larPista/2+30)? carPosX - 1.5 * speed/(15+(volta*2)): carPosX;
         if(viraCarro < 0)
            viraCarro = viraCarro > 25 ? viraCarro : viraCarro + 1.5;
         viraCarro = viraCarro > 25 ? viraCarro : viraCarro + 0.8;
         if(anima) speed = speed > 1 ? speed + abs(carPosX) * 0.0001 * speed/(15+(volta*2)) : 1;
      }
      if(botoes[3] && !isTouchingRight()){ //impedir virar pra direita quando estiver fora da
pista
         carPosX = carPosX <= larPista/2+30? carPosX + 1.5 * speed/(15+(volta*2)): carPosX;
         if(viraCarro > 0)
            viraCarro = viraCarro < -25 ? viraCarro : viraCarro - 1.5;
         viraCarro = viraCarro < -25 ? viraCarro : viraCarro - 0.8;
         if(anima) speed = speed > 1 ? speed + abs(carPosX) * 0.0001 * speed/(15+(volta*2)): 1;
      }

      if(!botoes[2] && !botoes[3]){
         if(viraCarro > 0){
```

```
            viraCarro = viraCarro * 0.92;
         }else if (viraCarro < 0){
            viraCarro = viraCarro * 0.92;
         }
      }
   }
...
   glFlush();
   glutSwapBuffers();
}
```

Quadro 3: Trecho de código da função Desenha(), onde é definido as posições e cores de cada adversário

3.4 Criação da(s) pista(s)

As ideias para a pista eram duas: Pista Pseudo 3D e Pista Real 3D. A Pista Pseudo 3D é um triângulo isóscele 2D (foto 2, à esquerda), dando a sensação de profundidade em uma estrada no mundo real ao jogador. Neste caso, deve-se usar a função glOrtho().

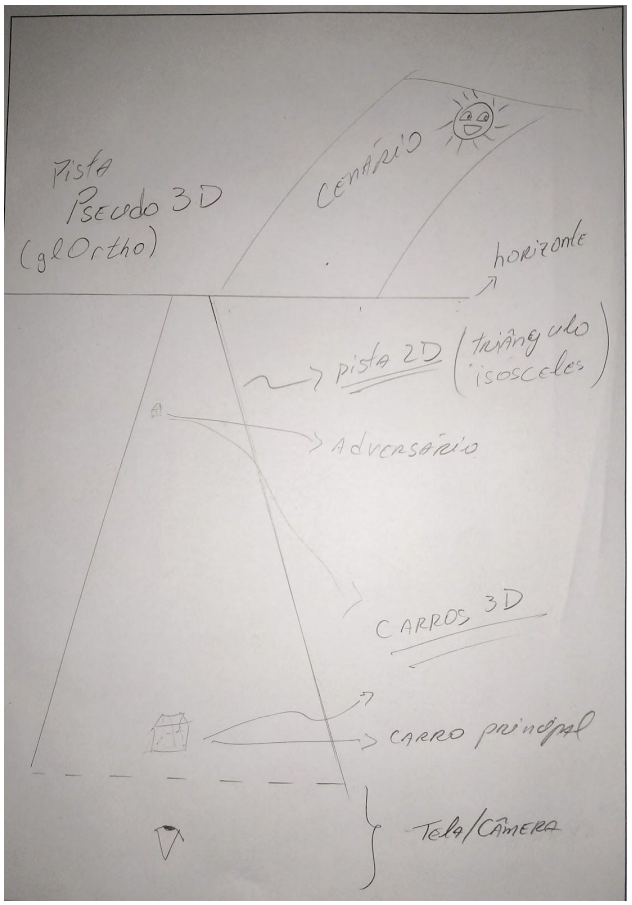

Foto 2: Uma das dezenas de rascunhos, Pista Pseudo 3D

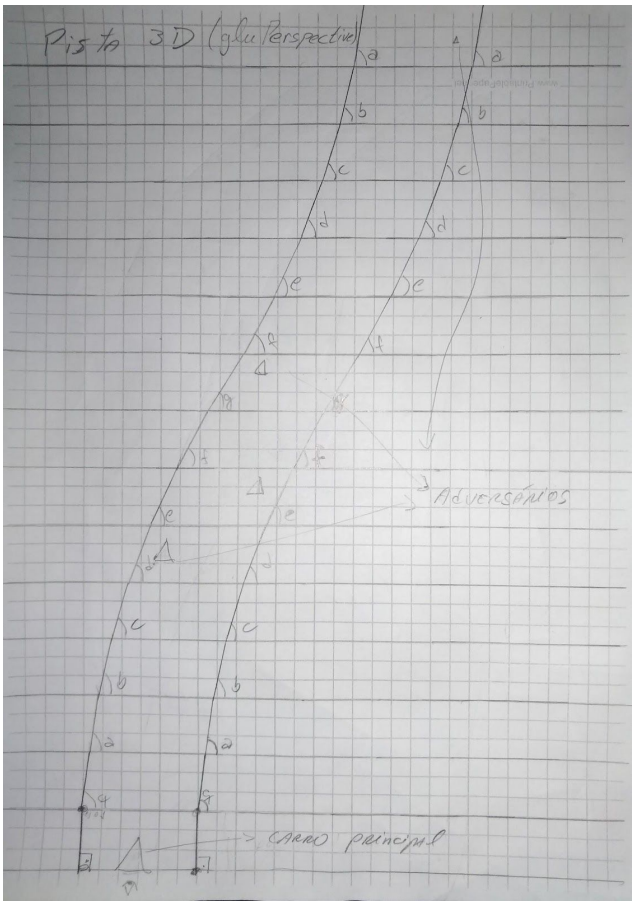

Foto 3: Pista Real 3D

A Pista Real 3D é formada por vários retângulos (foto 2, à direita) construídos conforme o avanço e, portanto, deve-se usar a função gluPerspective() para dar a sensação de profundidade.

Para este projeto foi utilizado a Pista Real 3D com modificações e a visão em perspectiva. Esta é construída em um vetor na função main() e desenhada na função DesenhaPista(), ambas em "main.c". É formada por vários retângulos com duas cores alternadas para dar melhor sensação de movimento, pois se fossem de apenas uma cor, seria difícil notar que o carro está percorrendo a pista. O início de cada curva é pré-determinada em pontos da pista conforme o trecho "//Curvas",

em destaque no quadro 4. O término de um “if” e o início do seguinte é uma reta e cada if indica o início e término da curva, sendo:

0.001 para curvas a direita;
-0.001 para curvas a esquerda;

```
int main(int argc, char *argv[]){
...
   initArray(&Pontos, tamPista + 2);
      for(int i = 0; i < tamPista; i++){
         Ponto_t ponto;
         ponto.x = 0;
         ponto.y = 0;
         ponto.z = -i;
         ponto.curve = 0;

         //Cor
         if(c == 50){
            c = 0;
            flagCor = flagCor? false : true;
         }else
            c++;
         ponto.cor = flagCor;

         // Curvas
         if(i >  1800 && i <  2800) ponto.curve =  0.001;
         if(i >  3800 && i <  4800) ponto.curve = -0.001;
         if(i >  5800 && i <  8800) ponto.curve =  0.001;
         if(i > 12800 && i < 14800) ponto.curve = -0.001;
         if(i > 10000 && i < 12000) ponto.curve =  0.001;
         if(i > 15000 && i < 17000) ponto.curve = -0.001;

         insertArray(&Pontos, ponto);
      }
...
   glutMainLoop();
}
```

Quadro 4: Pista Real 3D, Vetor onde os pontos da pista e suas curvas são definidos

3.5 Adversários

Os adversários são criados com o mesmo modelo do carro principal, apenas muda-se a cor. No trecho em destaque no quadro 5, na função DesenhaBots(), os adversários são criados ao longo da pista, alternando cores e posições em relação a horizontal (dxBot) e ao longo da pista (i).

```
void Desenha(){
...
   for(int i = 0; dzBot < tamPista; i += 991){
      glPushMatrix();
         if(contador == 0){
            contador++;
            dxBot = 0;
            DesenhaBots(corBot[contador], i, dxBot);
         }else if(contador == 1){
            contador++;
            dxBot = 35;
            DesenhaBots(corBot[contador], i, dxBot);
         }else if(contador == 2){
            contador++;
            dxBot = 0;
            DesenhaBots(corBot[contador], i, dxBot);
         }else if(contador == 3){
            contador = 0;
            dxBot = -35;
            DesenhaBots(corBot[contador], i, dxBot);
```

```
        }
      glPopMatrix();
   }
...
   glFlush();
   glutSwapBuffers();
}
```

Quadro 5: Trecho de código da função Desenha(), onde é definido as posições e cores de cada adversário

A função DesenhaBots(), que desenha os adversários, está no arquivo “global.c”, onde é calculado a distância deles e o carro principal. Além de pequenos ajustes para a animação deles com as curvas.

3.6 Colisões

Para que o carro não saia da tela ao mover-se na horizontal, foi colocado o limite da zebra, onde o carro desacelera até um valor mínimo e também deixe de sair do campo de visão, destacado no seguinte trecho de código da função Desenha() no quadro 6:

```
void Desenha(){
...
   if(!colidiu){
       if(botoes[0] && anima){
           pos += (0.12 * speed);
           posBot += 0.01 * speed;
       }
       if((botoes[1] ) && anima){
           pos -= (0.12 * speed);
           posBot += 0.15 * speed;
           speed -= 0.02;
       }
       if(botoes[2] && !isTouchingLeft()){ // impedir virar pra esquerda quando estiver fora da
pista
           carPosX = carPosX >= -(larPista/2+30)? carPosX - 1.5 * speed/(15+(volta*2)): carPosX;
           if(viraCarro < 0)
               viraCarro = viraCarro > 25 ? viraCarro : viraCarro + 1.5;
           viraCarro = viraCarro > 25 ? viraCarro : viraCarro + 0.8;
           if(anima) speed = speed > 1 ? speed + abs(carPosX) * 0.0001 * speed/(15+(volta*2)) :
1;
       }
       if(botoes[3] && !isTouchingRight()){ // impedir virar pra direita quando estiver fora da
pista
           carPosX = carPosX <= larPista/2+30? carPosX + 1.5 * speed/(15+(volta*2)): carPosX;
           if(viraCarro > 0)
               viraCarro = viraCarro < -25 ? viraCarro : viraCarro - 1.5;
           viraCarro = viraCarro < -25 ? viraCarro : viraCarro - 0.8;
           if(anima) speed = speed > 1 ? speed + abs(carPosX) * 0.0001 * speed/(15+(volta*2)):
1;
       }

       if(!botoes[2] && !botoes[3]){
           if(viraCarro > 0){
               viraCarro = viraCarro * 0.92;
           }else if (viraCarro < 0){
               viraCarro = viraCarro * 0.92;
           }
       }
   }
...
}
```

Quadro 6: Trecho de código da função Desenha(), onde foi estipulado o limite na horizontal

A colisão do carro principal com os adversários está definido na função DesenhaBots() no trecho em destaque no quadro 7.

```
void DesenhaBots(GLfloat *cor, GLint dzBot, GLint dx){
...
   if(Pontos.ponto[pos].curve == 0.0){
      if( (pos > (posBot+dzBot-220) && pos < posBot+dzBot-140) && ( // estao na mesma posicao
em z
         (carPosX - 18 <= dx && carPosX >= dx) || // player do lado direito do bot
         (carPosX + 18 >= dx && carPosX <= dx) )  // player do lado esquerdo do bot
      ){
         colidiu = true;
         posQndoBateu = 0;
       }
   }else if(Pontos.ponto[pos].curve > 0.0){
      if( (pos > (posBot+dzBot-220) && pos < posBot+dzBot-140) && ( // estao na mesma posicao
em z
         (carPosX - 26 <= dx && carPosX >= dx) || // player do lado direito do bot
         (carPosX + 10 >= dx && carPosX <= dx) )  // player do lado esquerdo do bot
      ){
         colidiu = true;
         posQndoBateu = 0;
       }
   }else if(Pontos.ponto[pos].curve < 0.0){
      if( (pos > (posBot+dzBot-220) && pos < posBot+dzBot-140) && ( // estao na mesma posicao
em z
         (carPosX - 10 <= dx && carPosX >= dx) || // player do lado direito do bot
         (carPosX + 26 >= dx && carPosX <= dx) )  // player do lado esquerdo do bot
      ){
         colidiu = true;
         posQndoBateu = 0;
       }
   }
...
}
```

Quadro 7: Trecho de código da função DesenhaBots(), onde é definido as colisões com os adversários

3.7 Física

Para aumentar o desafio foram adicionados o peso da aceleração do carro, a desaceleração do carro ao andar sobre a zebra e inércia nas curvas, para dar sensação de dificuldade ao realizá-las. Todas estão definidas na função TimerFunc() em “main.c” no trecho em destaque no quadro 8.

```
void TimerFunc(int valor){
...
   //Controle de velocidade
   // if((int)pos%tamPista == 0) volta++; //Cada volta no mapa tem tamPista posições.
   if(speed<15+(volta*2)) speed += 0.005; //Aceleracao maxima 35, aumenta em 2 para cada volta.
   if(speed<10) speed += 0.005;           //Aceleracao 0.2 quando abaixo de speed 20.
   if(speed<5) speed += 0.005;            //Aceleracao 0.3 quando abaixo de speed 05.

   //Inércia nas curvas
   if(Pontos.ponto[pos].curve > 0 ){ //Curva para a direita.
      if(carPosX >= -(larPista/2+25))
         carPosX = carPosX-1.3*speed/(15+(volta*2));
   }
   if(Pontos.ponto[pos].curve < 0 ){ //Curva para a esquerda.
      if(carPosX <= larPista/2+25)
         carPosX = carPosX+1.3*speed/(15+(volta*2));
   }

   // verifica se o carro está tocando alguma das bordas e desacelera
   if(isTouchingRight() || isTouchingLeft())
      speed = speed >= 3 ? speed - 0.08 : speed ;
...
   glutPostRedisplay();
```

```
}
```

Quadro 8: Trecho de código da função TimerFunc(), onde é definido a inércia nas curvas

3.8 Iluminação

Para o efeito de iluminação foi usado iluminação global, definido as normais nas faces dos carros, como mencionado anteriormente, em destaque no quadro 9. Além disso, foi criado um efeito de transição do céu, em destaque no quadro 10, e iluminação para criar um efeito de tempo avançando conforme o avanço na pista.

```
void InitScreen(){
...
   //Iluminacao
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_COLOR_MATERIAL);

   float difusao[]={1.0, 1.0, 1.0, 1.0};
   glLightfv(GL_LIGHT0, GL_DIFFUSE, difusao);
   //Controle do ambiente
   if(volta != voltaAnt2){
      voltaAnt2 = volta;
      contaCor2 = (contaCor2+1) % 4;
   }
   if(contaCor2 == 3){
     ambiente[0] = 1.0-((float)pos/(float)tamPista) > 0.3 ? 1.0-((float)pos/(float)tamPista) :
0.3;
     ambiente[1] = 1.0-((float)pos/(float)tamPista) > 0.3 ? 1.0-((float)pos/(float)tamPista) :
0.3;
     ambiente[2] = 1.0-((float)pos/(float)tamPista) > 0.3 ? 1.0-((float)pos/(float)tamPista) :
0.3;
      ambiente[4] = 1.0;
   }else if(contaCor2 == 2){
      ambiente[0] = 1.0;
      ambiente[1] = 1.0;
      ambiente[2] = 1.0;
      ambiente[4] = 1.0;
   }else if(contaCor2 == 1){
      ambiente[0] = ((float)pos/(float)tamPista) > 0.3 ? ((float)pos/(float)tamPista) : 0.3;
      ambiente[1] = ((float)pos/(float)tamPista) > 0.3 ? ((float)pos/(float)tamPista) : 0.3;
      ambiente[2] = ((float)pos/(float)tamPista) > 0.3 ? ((float)pos/(float)tamPista) : 0.3;
      ambiente[4] = 1.0;
   }else{
      ambiente[0] = 0.3;
      ambiente[1] = 0.3;
      ambiente[2] = 0.3;
      ambiente[4] = 1.0;
   }
   glLightfv(GL_LIGHT0, GL_AMBIENT, ambiente);
}
```

Quadro 9: Trecho de código da função InitScreen(), onde ocorre a transição de iluminação de dia para noite

```
void TimerFunc(int valor){
...
   //Controle do céu
   if(volta != voltaAnt){
      voltaAnt = volta;
      contaCor = (contaCor+1) % 4;
   }
   if(contaCor == 3){
      glClearColor(.0f, .0f, 1.0 - ((float)pos/(float)tamPista), .0f);
   }
   else if(contaCor == 2){
      glClearColor(.0f, .0f, 1.0, .0f);
   }
   else if(contaCor == 1){
```

```
      glClearColor(.0f, .0f, ((float)pos/(float)tamPista), .0f);
   }
   else{
      glClearColor(.0f, .0f, 0, .0f);
   }
...
   glutPostRedisplay();
}
```

Quadro 10: Trecho de código da função TimerFunc(), onde ocorre a transição do céu de dia para noite

3.9 Sistema de Pontuação

O Enduro, por se tratar de um jogo de resistência, a pontuação é incrementada constantemente ao longo do tempo com penalizações referentes a colisões, contra carros ou bordas da pista.

```
void TimerFunc(int valor){
...
   if(pontuacao < 0){
       pontuacao = 0;
   }else{
       pontuacao += 10;
   }
...
}
```

Quadro 11: Pontuação é incrementada constantemente na TimerFunc()

Em cada um dos eventos de colisão, tanto contra carros quanto as bordas da pista, foi feito uma penalização na variável de pontuação. Colisões na borda penalizam de acordo com a duração da colisão, sendo necessário se mover para dentro da pista para encerrar a penalização. Colisões contra carros causam uma grande penalização na pontuação (-2000 pontos), além do efeito de colisão.

```
void TimerFunc(int valor){
...
   if(isTouchingRight() || isTouchingLeft()){
       pos -= (0.12 * speed);
       posBot += 0.15 * speed;
       speed -= 0.02;
       pontuacao -= 30;
   }
...
}
```

Quadro 12: Pontuação é penalizada constantemente durante colisão com a borda na TimerFunc()

4. Conclusão

Esse projeto foi muito além da criação e programação do jogo. Deu uma amostra aos envolvidos sobre os processos de desenvolvimento de um jogo e parte do trabalho envolvido em uma produção do mesmo. É apenas parte de algo muito complexo, mas que tem um mercado muito amplo e promissor.