

Dragon Souls of Senac: Touhou Edition

Danilo Makoto Ikuta¹, Gabriel Nopper Silva Roddrigues¹

¹Centro Universitário Senacário Senac
04.696-000 – São Paulo – SP – Brasil

danilo0195@hotmail.com, ganopper@gmail.com

Resumo. Este artigo explica como foi feito o jogo Dragon Souls of Senac: Touhou Edition – A estrutura de código, alguns algoritmos e resultados. Este jogo foi produzido para o Projeto Interativo III (PI 3), cujo objetivo foi desenvolver um jogo que explorasse a visão computacional como forma de controle e utilizando a linguagem C com auxílio da biblioteca Allegro 5. As funções (previamente disponibilizadas) que realizam a captura de imagens da câmera utilizam a biblioteca OpenCV.

Dragon Souls of Senac: Touhou Edition é um jogo de escudo e espada, estes, que devem ser utilizados para atacar e se defender de inimigos. Ao derrotar uma certa quantidade deles, o chefe da fase aparece, e ao derrotá-lo, o jogador avança para a próxima fase.

Abstract. This article explains how the game Dragon Souls of Senac: Touhou Edition was made – It's code structure, some of it's algorithms and results. This game was developed for Interactive Project III class, whose objective is to develop a game that explored computer vision as the only form of controller, using C language with Allegro5 library extension. The functions (Previously given to us) that perform the image capture with the camera make use of the OpenCV library.

Dragon Souls of Senac: Touhou Edition is a sword & shield game, those which must be used to block and attack foes. At defeating a certain amount of them, the stage boss will appear, and by defeating him, the player may proceed to the next levels.

1. Introdução

Este relatório traz os resultados do Projeto Interativo III(PI III), de Bacharelado em Ciencia da Computação do Centro Universitário Senac, 2014. O PI III traz como proposta a construção de um jogo em linguagem C com auxilio das bibliotecas Allegro, para construção grafica, e OpenCV para captura de video. Este jogo, deve ter apenas uma camera como meio de interação jogador-programa. Cabe aos alunos esta interação jogador-programa via algoritmos de tratamento de visão computacional, e os relacionar com a interface também gerada via Allegro.

2. Revisão de Literatura

2.1. A idéia

O próprio nome dado ao jogo indica uma composição de bases para a idéia. A idéia do jogo efetivamente surgiu com Dragon Quest, para plataforma Wii. Neste jogo, utiliza-se o

WiiMote (O controle específico do console) como a espada, e o nun-chuck(A contraparte esquerda do controle) como um escudo. No jogo, o jogador é colocado e um caminho no qual se anda constantemente. Ao longo deste caminho, entretanto, surgem diversos monstros que ele deve derrotar, e no final de cada caminho, um chefe. A idéia é similar - é dada uma fase, monstros nascem, mas ao invés de completar tempo, um numero específico de monstros derrotados é necessário. Ainda sim, no final, um chefe também será encontrado.

O nome DSoSTE (Dragon Souls of Senac: Touhou edition) é dado baseado nos 3 jogos inspiração para o projeto. Destaquemos os outros 2:

Dark Souls: Dark Souls é um jogo recente (2011, 2014 o segundo) que ficou rapidamente famoso deviado a alguns fatores. O primeiro destes, e potencialmente o que faz a fama do jogo, é uma extrema dificuldade. A jogabilidade "sem perdão" e a falta de recursos para cura ou instante seguros tornou o jogo famoso em curto espaço de tempo. O outro fator é uma combinação de graficos e musica que traz um tema sombrio ao jogo. Tentamos trazer ambos fatores ao jogo: Um clima ora sombrio ora de tensão, com uma dificuldade grande e escassez intencional de recursos.

Touhou: Touhou é um jogo não-comercial feito por programadores apenas por diversão. Trata-se de um jogo plataforma de tiro, similar ao clássico Space Invaders, com extremas diferenças. Primeiro, o movimento é livre para qualquer lado em direções no eixo X e Y. Além disso, os inimigos se movem independentemente, os tiros são mirados no jogador, e com frequencia eles são tiros multiplos, lotando a tela de tiros. tenta-se trazer a quantidade enorme de tiros em tela do Touhou, para este jogo.

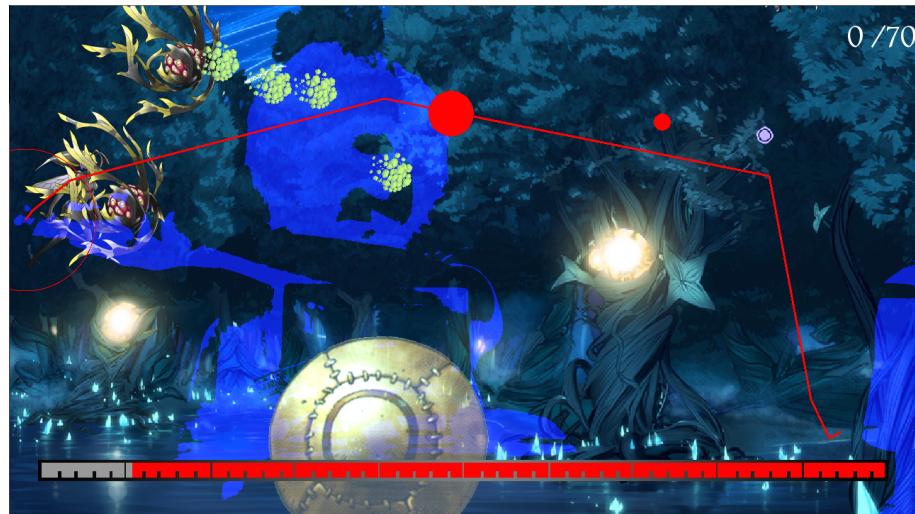


Figura 1. Ataques de monstro na tela. Cada circulo vermelho é um alvo sendo gerado, e o ataque pode ser bloqueado se o escudo estiver sobre a posição do circulo no instante de acerto do ataque

2.2. Adaptação

Em se tratando de visão computacional, é necessário diferenciar o sistema em relação ao Dragon Quest.

Primeiro, observa-se que no jogo original, utiliza-se como controle o Wii Mote e o Nun-chuck, controles centrados por infravermelho em um aparelho captor de alta performance. isto permite além de uma precisão extrema, a habilidade de, apenas com

botões pressionados, acionar a espada ou escudo. Com a visão computacional, entretanto não há controles - isto demanda certa adaptação.

Primeiro, foi personalizado a lógica de acerto. Com uma subtração de fundo, cria-se um hitbox igual ao corpo da pessoa. Maioria dos ataques acerta o jogador apenas se desferido sobre aquela hitbox. caso contrário, o ataque "erra", permitindo que o jogador desvie.

Além disso, não há uma forma adequada de controle para balancear escudo e espada, originalmente restrito a apenas um ligado em tela. Por isso, o tamanho do escudo é reelevantemente menor que em Dragon Quest, e o corte da espada também não pode acertar sobre o escudo.

Também não há uma forma de pause ou inventário por limitações de controle. Portanto, substitui-se magias de cura ou uso de itens, por algo diferente. Em geral a escolha obvia para visão computacional é a de monstros derrubarem "itens" de vida adquiríveis ao toque. Entretanto, esta escolha complicaria muito a usabilidade, gerando três focos distintos em uma mesma tela ao jogador.

Por isso, optou-se pela estratégia de ganho de vida no acerto. A espada do jogador seria uma espada mágica, que drena a força vital dos inimigos acertados, lentamente recuperando vida baseado em dano total.

3. Desenvolvimento

Para fazer o uso da visão computacional, para controlar a espada e o escudo, foi escolhido o método de identificação por cor e quanto à identificação e segmentação entre jogador e fundo(background), foi utilizado a técnica de subtração de fundo.

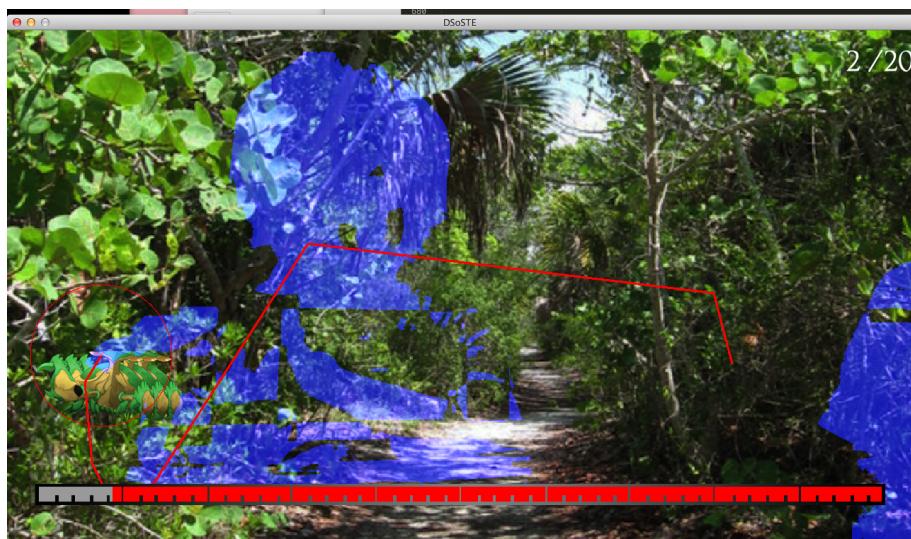


Figura 2. Capturas da Camera. A imagem azul é a subtração de fundo do jogador, e a linha vermelha foi desenhada pela detecção da espada (Cor vermelha)

3.1. Identificação de Espada e Escudo

Ao utilizar a separação por cor com o sistema RGB(padrão da câmera), onde a cor de um pixel é representada pela quantidade de vermelho, verde e azul que ela possui, sendo que

se todos os valores forem mínimos, a cor será preta e se todos os valores forem máximos, a cor do pixel será branca, por exemplo. Para tentar identificar a espada pela cor vermelha, notou-se que nem sempre os resultados saiam como o esperado, pois quando havia muita variação de luz, os resultados saiam com alguma distorção, já que essa diferença de iluminação afeta todos os parâmetros de cor de dado pixel. Era preciso parametrizar a quantidade de luz. Ao testar a identificação por cor, antes convertendo os dados de entrada da câmera para o sistema HSV(Hue, Saturation, Value), onde hue é o "tom" de cor propriamente dito(vermelho, azul, amarelo, verde, etc.), saturation corresponde à intensidade da cor(quanto menor a saturação, mais próximo de tons acinzentados) e value, que mede o brilho dessa cor(quanto maior, mais clara a cor e quanto menor, mais escura). Através da função rgbToHsv(abaixo), foi possível utilizar uma fonte de luz vermelha como espada com precisão satisfatória. Aproveitando a conversão, são checados os parâmetros para a identificação do escudo, de cor verde.

```

1 void rgbToHsv(int r, int g, int b, int *h, int *s, int *v){
2     Divide r, g e b por 255(em variáveis float), afim de representá-los
3         entre o intervalo [0, 1];
4
5     Identifica os valores máximo(max) e mínimo(min) dentre os três;
6     Calcula a diferença(delta) entre os valores máximo e mínimo;
7
8     Se (delta > 0){
9         Se(max == r){
10             Se(g > b){
11                 *h = 60 * ((g - b) / delta);
12             }
13
14             Senão{
15                 *h = 60 * ((g - b) / delta) + 360;
16             }
17
18         Se(max == g){
19             *h = 60 * ((b - r) / delta) + 120;
20         }
21
22         Senão{
23             *h = 60 * ((r - g) / delta) + 240;
24         }
25     }
26
27     Senão{
28         *h = 0;
29     }
30
31     *s = delta / max * 100;
32
33     *v = max;
34 }
```

Tanto para espada quanto escudo, calcula-se o ponto médio dos pixels identificados de cada um, e a partir de uma quantidade mínima de pixels identificados no frame, esses dados são utilizados para calcular a posição dos centros de massa atuais da espada

e do escudo.

Para identificar e guardar o rastro dos ataques, a cada execução do ciclo de câmera, a posição da espada é armazenada numa fila, cujos pontos são processados em pares pela função drawAtk para determinar as posições inicial e final de cada execução da função al_draw_line. A cada vez que a função drawAtk é chamada toda a fila é percorrida, pois para os pares de pontos que foram adicionados antes, aplica-se um efeito de *fade-out* ao reduzir o valor de alpha da cor da linha desenhada por al_draw_line. O ponto mais antigo da fila sempre é removido toda vez que a contagem de pontos for maior que o limite determinado (no caso, 5 pontos).

```
1 drawAtk( fila *f){  
2     Se(existirem pelo menos dois elementos em f){  
3         Cria dois apontadores de elemento(e1 , e2) e faz com que apontem  
4             para os dois primeiros elementos da fila ;  
5         Enquanto(e1 e e2 não apontarem para NULL){  
6             Desenha uma linha entre os pontos e1 e e2;  
7  
8             Faz com que e1 aponte para o elemento apontado para e2 e e2  
9                 para o próximo elemento de e2;  
10            }  
11        }  
12    }  
13 }
```

O caso do escudo é mais simples: a posição determina o ponto central do escudo, que é utilizado para desenhar a imagem deste.

3.2. Separação de fundo

Toda vez que o jogo é executado, alguns segundos são reservados à obtenção do fundo(de preferência estático), sem o jogador (através da função getBackground). Algumas imagens do fundo são convertidas para HSV. A média para cada pixel dessas imagens são armazenadas numa matriz, a qual será comparado cada frame para determinar se dado pixel pertence ao fundo ou ao jogador. Os pixels identificados como sendo do jogador são armazenados em uma matriz com uma cor azulada, da qual é obtida um ALLEGRO_IMAGE, as partes que não forem dessa cor são transformadas em transparência através da função al_convert_mask_to_alpha e desenhadas na tela. O parâmetro utilizado atualmente no jogo para diferenciar jogador e fundo é verificar se a diferença de luminosidade(value) entre pixel verificado no momento e o pixel de mesma posição na matriz de background é maior que 25%.

```
1 getBackground(camera *cam, int ***background){  
2     Carrega a imagem de título do jogo e a exibe enquanto se obtém o  
3         fundo ;  
4  
5     Para um certo número(i) de vezes{  
6         Para cada pixel do frame atual da câmera (x, y){  
7             Converte o pixel de RGB para HSV;  
8  
9             Soma os atributos de cada valor ao array background[y][x];  
10            //background[y][x][0] = h, background[y][x][1] = s,  
11                background[y][x][2] = v  
12        }  
13 }
```

```

11    }
12
13    Para cada pixel do frame atual da câmera{
14        Divide os valores de background[y][x][0] ,background[y][x][1] e
15        background[y][x][2] pela quantidade de amostras(i);
16    }

```

3.3. Ciclo de câmera

A cada ciclo do jogo, a função que encapsula o processamento da câmera(exceto a obtenção de fundo) é chamado, e para cada pixel da tela(cuja resolução é a mesma que a da câmera), é verificado se esse pixel é de espada, escudo, jogador ou fundo. Após a varredura, os pixels identificados como sendo do jogador são desenhados na tela(display) e as coordenadas do centro de massa da espada e do escudo, além de gerenciar a fila dos pontos da espada e desenhar o ”rastro do ataque”através da função drawAtk.

3.4. Código do Jogo

O fluxo de jogo ocorre dentro de um looping geral, descrito abaixo, onde a cada loop, os monstros e ataques se movem ou realizam algo, enquanto a camera atualiza as ações e movimentos do jogador, e a interface. O algoritmo a seguir é uma versão completa do encontrado no poster referente a esse artigo, e explica detalhadamente o funcionamento do fluxo de jogo.

```

1 Enquanto(Ultimo chefe não derrotado){
2
3     // Ciclo de nascimento
4     Se (Tempo para criar monstro = tempo da fase){
5         Se(Monstro deve ser Invasor){
6             Cria invasor;
7             Luta contra invasor = Verdadeiro;
8             Desenhar Animação de invasor = Verdadeiro;
9         }
10        Se não
11            Cria monstro baseado na fase;
12        }
13
14
15     // Ciclo dos monstros
16     Para (cada monstro na lista){
17         Realiza Movimento;
18         Avança contador de timer p/ ataque;
19         Se (contador de ataque pronto){
20             Gera ataque;
21             Reseta contador de ataque;
22         }
23         Desenha bitmap do monstro;
24     }
25
26     Verifica e limpa monstros mortos;
27
28     // Ciclo do chefe
29     Se (Há um chefe na lista){

```

```

30     Se (Vida abaixo de 0){
31         Limpa chefe ;
32         Se(luta contra invasor){
33             Desenha animação de morte de invasor ;
34         }
35         Se não{
36             Deve Mudar de fase = Verdadeiro ;
37         }
38     }
39     Se não{
40         Executa movimento ;
41         Avança cont. de timer p/ ataque ;
42         Se (Contador de ataque pronto){
43             Gera ataque ;
44             Reseta contador de ataque ;
45         }
46         Desenha bitmap do chefe ;
47     }
48 }
49
50 // Ciclo de câmera
51 Chama captura da câmera ;
52
53 // Ciclo de ataques
54 Para(Cada ataque na lista){
55     Move ataque ;
56     Se(Posição do Ataque for a Pos. alvo){
57         Se(acertou escudo){
58             Executa audio ;
59             Deleta ataque ;
60         }
61         Se não{
62             Subtrai dano da defesa ;
63             Se(Jogador possui barreira){
64                 Causa dano sobre barreira ;
65                 Causa dano restante sobre vida ;
66             }
67             Se Não
68                 Causa Dano sobre vida ;
69         }
70     }
71 }
72
73 // Ciclo de interface
74
75 Se(Há animações para fazer)
76     Executa animações ;
77
78 Desenha escudo ;
79 Desenha Elementos de interface ;
80
81 Joga elementos desenhados na tela ;
82
83 // Ciclo do jogador
84 Para (Cada monstro na lista){
85     Verifica se foi acertado ;

```

```

86      // Dentro desta verificação ocorre dano
87  }
88
89      // Ciclo de mudança de fase
90  Se(Deve mudar de fase){
91      Executa animação de transição;
92  }
93
94      // ciclo de fim de jogo
95  Se(Vida do jogador inferior a 1){
96      Encerra jogo por função de morte;
97  }
98
99
100 }

```

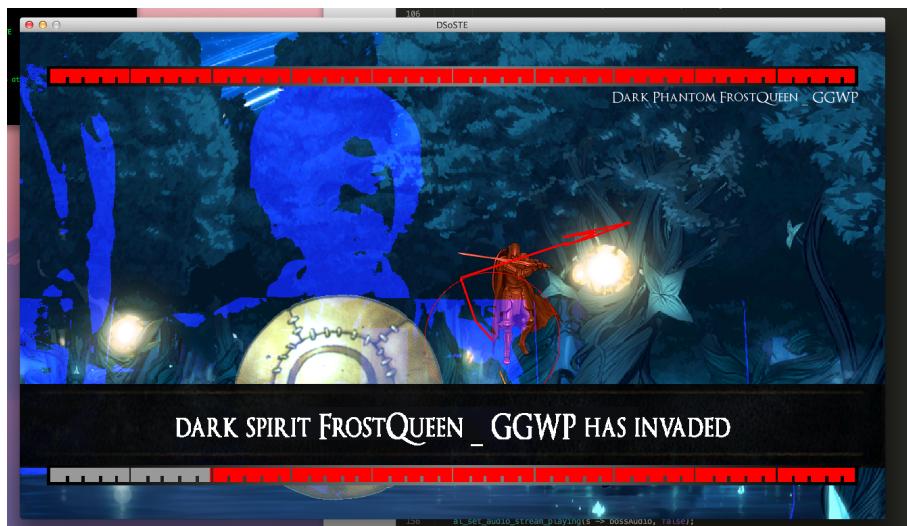


Figura 3. Nascimento de um invasor. Note que surge uma barra de vida referente a este, no topo da tela, e ocorre uma animação informando a invasão.

4. Resultados

O resultado foi bastante próximo do desejado. Embora haja um atraso na detecção natural de qualquer WebCam, é possível jogar o jogo mesmo este sendo rápido. A detecção do escudo ficou extremamente precisa, com uma detecção rápida que permite movimentações rápidas. A detecção da espada, similarmente possui uma detecção rápida e precisa, embora devido a nosso hardware, o usuário deva tomar mais cuidado com apontar o led frontalmente.

Inicialmente, foi escolhido este projeto, não apenas pela detecção de objetos diversos, mas pela construção de um jogo elaborado e bem construído, algo que foi atingido.

O único fator não atingido nos resultados esperados foi o hitbox do jogador, gerado pela subtração de fundo. A subtração de fundo do corpo do jogador deveria ser a única região onde ele fosse acertado, permitindo esquivas no jogo. Entretanto, subtração de fundo não é um método perfeito, deixando vãos e "Buracos" no hitbox do jogador, especialmente em regiões onde a cor do jogador e do fundo são bem similares.

Mesmo assim, manteve-se a subtração de fundo, não como hitbox, mas apenas como representação gráfica do jogador no jogo.

5. Considerações Finais

Chegamos onde queríamos. Isto pode ser dito com certeza. O conhecimento prévio de allegro foi relevante, entretanto nós dois trabalhamos no BEPiD, e lá realizamos vários pequenos projetos de lógica similar, o que nos permitiu realizar este projeto maior facilmente. A mesma lógica usada lá, gerou os ciclos de ataque e monstros.

O grande desafio foi o reconhecimento da Camera, e concluímos que a camera pode ser um instrumento extremamente eficiente, apesar de custoso para a memória. Foi interessante também aprender a composição visual do computador e o funcionamento por trás da WebCam, lições que definitivamente nos serão úteis no futuro.



Figura 4. Morte do jogador. Fim de jogo.

Referências

Allegro 5 Development Team(2013).Allegro 5.0 reference manual, <https://www.allegro.cc/manual/5/>, ultimo acesso em junho de 2014.

Site de Rafael Toledo, <http://www.rafaeltoledo.net>, ultimo acesso em em junho de 2014

Heredian, PI6 de 2013 do BCC-Senac <https://github.com/celsovlpss/BCC-2s13-PI6-Heredian>