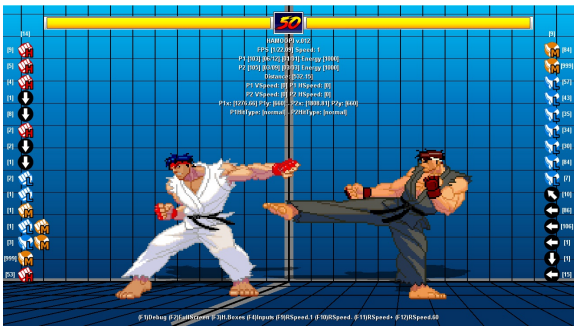
HAMOOPIG

www.youtube.com/c/GameDevBoss

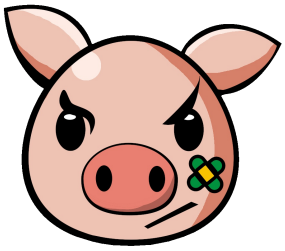
by Daniel Moura, aka GameDevBoss
2015 - 2022

HAMOOPI é um Jogo de Luta Open Source feito para rodar em PCs.
HAMOOPI é um acrônimo de "Half-Moon Punch", ou "Meia Lua Soco", uma maneira muito brasileira e particular de se referir ao movimento de soltar um hadouken, movimento clássico e emblemático dos games de luta. Esta é uma Engine feita em C++/Allegro, e que começou em Novembro/2015.
O código fonte desta engine se encontra no GitHub do autor:

https://github.com/DanielMoura79/HAMOOPI

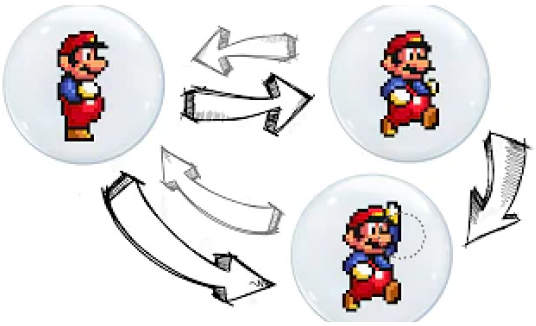


HAMOOPIG é a implementação da engine **HAMOOPI** no console SEGA Genesis. A HAMOOPIG no momento em que este texto foi escrito tem apenas 6 meses de desenvolvimento.



Como funciona a Hamoopig?

A criação de um jogo de luta é uma tarefa bastante complexa.
Mesmo em PCs, a solução mais fácil seria utilizar o MUGEN, para não ter que criar um jogo a partir do Zero. Com a **HAMOOPI** / **HAMOOPIG** você terá um ponto de partida, muitas coisas já foram feitas para você ter mais facilidade ao produzir o seu game, e se concentrar no que realmente importa, que é o conteúdo do seu jogo.
A **HAMOOPIG** foi criada para permitir que você crie jogos tão bons quanto aqueles melhores jogos de luta dos arcades dos anos 80 / 90 no auge da era 2D no Genesis! Vamos começar falando da máquina de estados.



www.youtube.com/watch?v=Y8YTk2ibPew

De maneira simplificada e resumida, a Máquina de estados (FSM) controla os estados, ou ações que o seu personagem pode fazer. Assim, o movimento parado é um estado, o movimento andando é outro, e socos e chutes também! A FSM controla a transição entre estes estados, que são finitos. A HAMOOPIG possui uma tabela numérica que determina cada um deles, e é importante você ter esta tabela por perto para criar personagens nesta engine. Vamos a ela.

Tabela de movimentos HAMOOPIG

000 - Retrato Grande do Personagem
001 - Retrato Pequeno do Personagem
100 - Personagem PARADO
101 - Soco Fraco de Longe
102 - Soco Médio de Longe
103 - Soco Forte de Longe
104 - Chute Fraco de Longe
105 - Chute Médio de Longe
106 - Chute Forte de Longe
107 - Inicio da Defesa em Pé
108 - Defendendo em Pé
109 - Final da Defesa em Pé
110 - Defesa em Pé, Aplicada *(IMG 108)

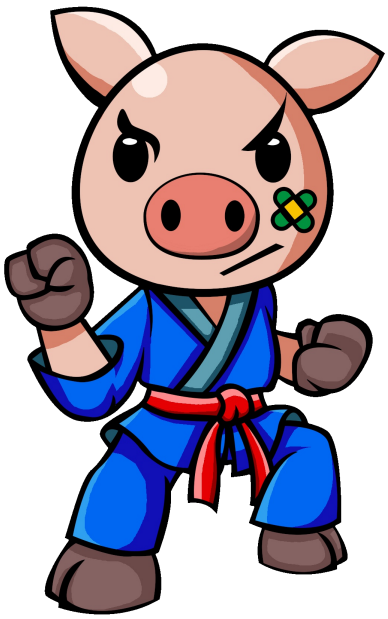


Tabela de movimentos HAMOOPIG

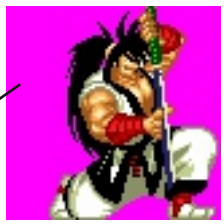
151 - Soco Fraco de Perto
152 - Soco Médio de Perto
153 - Soco Forte de Perto
154 - Chute Fraco de Perto
155 - Chute Médio de Perto
156 - Chute Forte de Perto

200 - Abaixado
201 - Soco Fraco Abaixado
202 - Soco Médio Abaixado
203 - Soco Forte Abaixado
204 - Chute Fraco Abaixado
205 - Chute Médio Abaixado
206 - Chute Forte Abaixado
207 - Inicio da Defesa Abaixado
208 - Defendendo Abaixado
209 - Final da Defesa Abaixado
210 - Defesa Abaixada, Aplicada *(IMG 208)

300 - Pulo Neutro
301 - Soco Fraco Aéreo Neutro
302 - Soco Médio Aéreo Neutro
303 - Soco Forte Aéreo Neutro
304 - Chute Fraco Aéreo Neutro
305 - Chute Médio Aéreo Neutro
306 - Chute Forte Aéreo Neutro

310 - Pulo para Trás
311 - Soco Fraco Aéreo para Trás
312 - Soco Médio Aéreo para Trás
313 - Soco Forte Aéreo para Trás
314 - Chute Fraco Aéreo para Trás
315 - Chute Médio Aéreo para Trás
316 - Chute Forte Aéreo para Trás

320 - Pulo para Frente
321 - Soco Fraco Aéreo para Frente
322 - Soco Médio Aéreo para Frente
323 - Soco Forte Aéreo para Frente
324 - Chute Fraco Aéreo para Frente
325 - Chute Médio Aéreo para Frente
326 - Chute Forte Aéreo para Frente



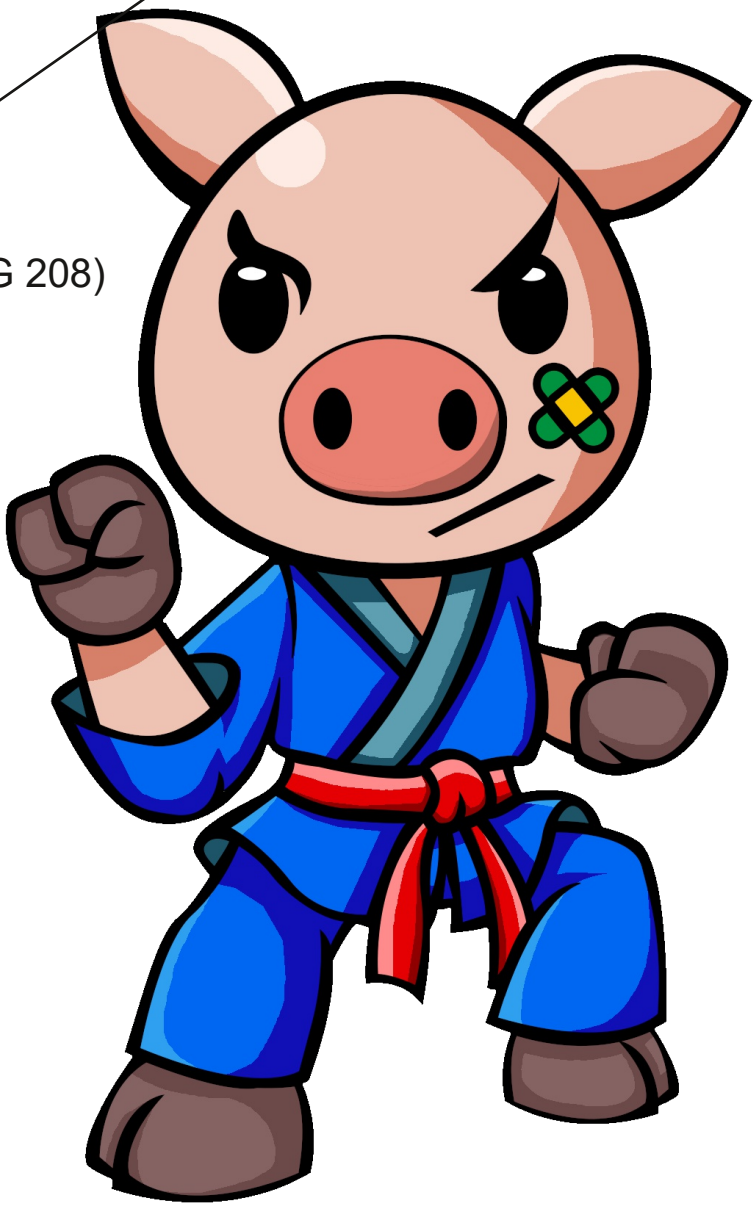
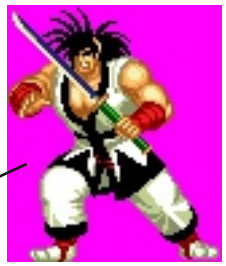


Tabela de movimentos HAMOOPIG

410 - Andando para Trás
420 - Andando para Frente
470 - Esquiva para Baixo / Esquiva para Trás
471 - Rolamento para Trás
472 - Rolamento para Frente
480 - Inicio da corrida
481 - Correndo
482 - Final da Corrida

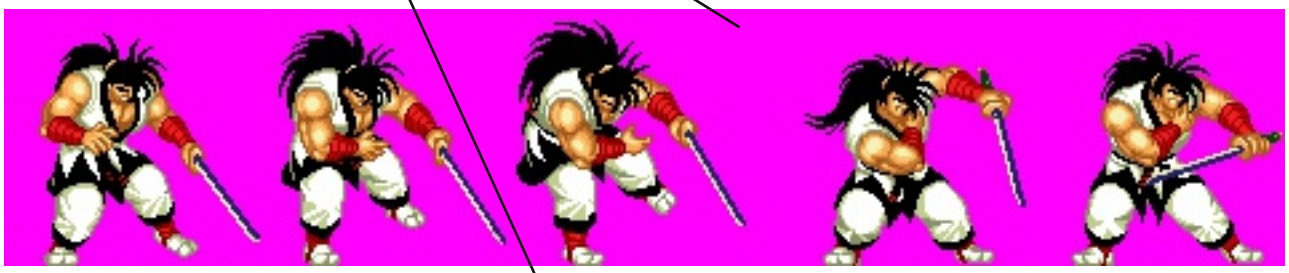


501 - Hit Tipo 1, Fraco
502 - Hit Tipo 1, Medio
503 - Hit Tipo 1, Forte
504 - disponível
505 - disponível

Hit Tipo 1 equivale a receber um golpe na altura do rosto e deslocar-se para trás com a força do impacto.

Hit Tipo 2 equivale receber um golpe na altura do estomago.

506 - Hit Tipo 2, Fraco
507 - Hit Tipo 2, Medio
508 - Hit Tipo 2, Forte
509 - disponível
510 - disponível



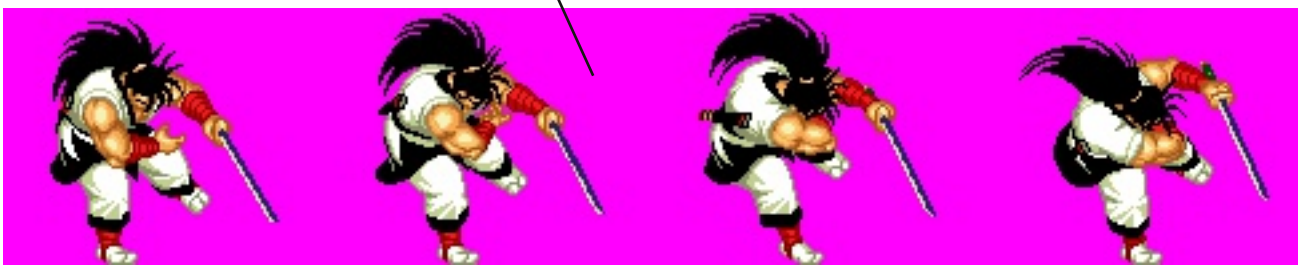


Tabela de movimentos HAMOOPIG

511 - Hit Tipo 3, Fraco
512 - Hit Tipo 3, Medio
513 - Hit Tipo 3, Forte
514 - disponível
515 - disponível
516 - User Hit
517 - disponível
518 - disponível
519 - disponível
520 - disponível

Hit Tipo 3 equivale receber um golpe estando abaixado.

550 - Caindo

551 - Quicando no chão
552 - Levantando

570 - Caido, Morto

Tabela de movimentos HAMOOPIG

601 - Abaixando
602 - Levantando
603 - Inicio Pulo para Trás *(IMG 604)
604 - Inicio Pulo Neuto
605 - Inicio Pulo para Frente *(IMG 604)
606 - Final do Pulo / Aterrisando no chão
607 - Virando, em Pé
608 - Virando Abaixado

610 - Intro
611 - Win1
612 - Win2
613 - Win3
614 - Win4
615 - Perdendo por Time Over
618 - Rage Explosion (Samurai Shodow 2)

Os numeros a partir do 700 podem ser usados para os especiais. Eu recomendo cada dezena para 1 tipo de especial, assim, do 700 até o 900 temos espaço para 20 especiais e supers.
Exemplo de como você pode trabalhar um especial (Lembre-se que você pode utilizar as numerações como melhor desejar):

700 - Especial 1
701 - Fireball 1
702 - Spark da Fireball 1
703 - -disponível-
704 - -disponível-
705 - -disponível-
706 - -disponível-
707 - -disponível-
708 - -disponível-
709 - -disponível-

Programando :)

Agora vamos analisar de forma rápida algumas funcões que fazem a engine funcionar.
A **FUNCAO_INPUT_SYSTEM()** é importantíssima, pois ela vai retornar o estado de cada botão dos controles, permitindo criar lógica de troca de estados com a FSM.

variáveis:

P[1].key_JOY_LEFT_status
P[1].key_JOY_RIGHT_status
P[1].key_JOY_UP_status
P[1].key_JOY_DOWN_status
P[1].key_JOY_A_status
P[1].key_JOY_B_status
P[1].key_JOY_C_status
P[1].key_JOY_X_status
P[1].key_JOY_Y_status
P[1].key_JOY_Z_status
P[1].key_JOY_MODE_status
P[1].key_JOY_START_status

valores:

0 => Nao apertado
1 => Acabou de apertar
2 => Mantem apertado
3 => Acabou de soltar

Assim, um comando do tipo ***if(P[1].key_JOY_START_status==1)*** está verificando se o Player 1 acabou de apertar o START, permitindo que você faça alguma coisa na sequência.

A **FUNCAO_FSM** faz a troca de estados de acordo com as condições estabelecidas. Estas condições podem ser bem simples, ou bastante complexas dependendo do caso. Um exemplo simples:

```
if(gDistancia>64)
{
	//soco fraco de pe de longe
	if( (P[i].key_JOY_X_status==1) && (P[i].state==100) )
	{
		PLAYER_STATE(i,101);
	}
}
```

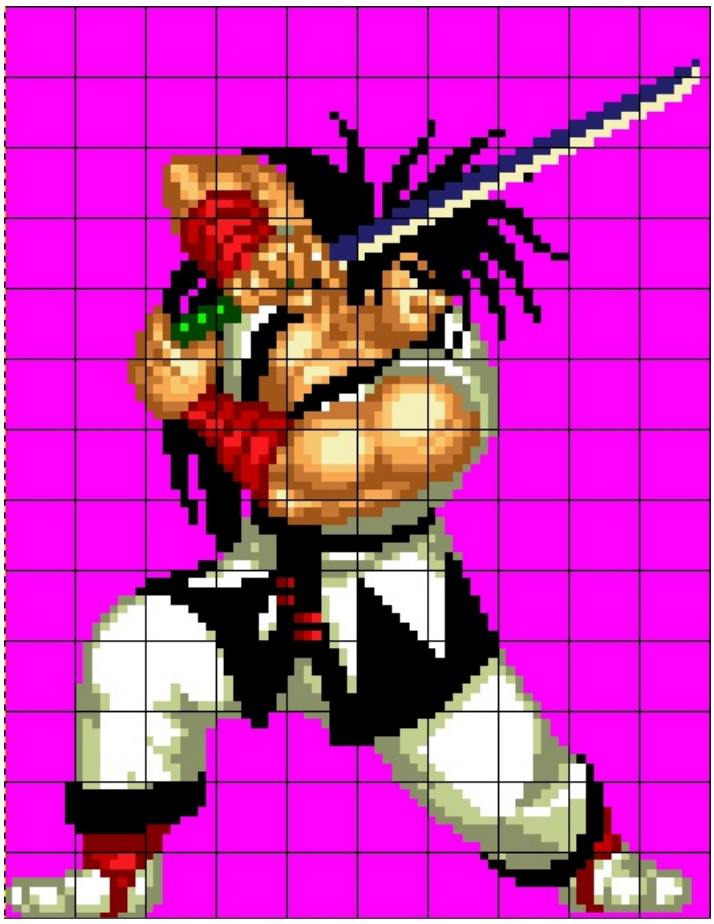
A **FUNCAO_ANIMACAO()** também faz parte da FSM, pois é ela que faz o retorno das animações de ataque ao estado parado, entre outras coisas. Assim, ao soltar um SOCO FORTE (103), ao terminar a animação, será carregado o estado PARADO (100). Quem faz isso é a FUNCAO_ANIMACAO. Outra coisa importantíssima que essa função faz é receber parâmetros para criar uma animação de tempo customizável. Resumidamente, este é um sistema de animação criado pelo GameDevBoss que permite colocar um tempo diferente para cada quadro de sua animação, tornando possível animações de nível profissional, muito melhores que as realizadas com o sistema de animação padrão da SGDK.

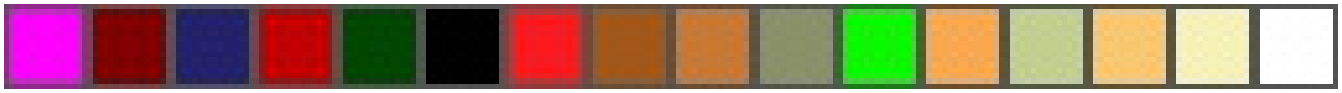
A **PLAYER_STATE** é provavelmente uma das funcoes mais importantes, ao lado da FSM. É nela que configuramos os parâmetros dos nossos personagens. Vejamos:

```
if(P[Player].id==1)
{
	if(State==100)
	{
		P[Player].y = gAlturaPiso;
		P[Player].w = 10*8;
		P[Player].h = 15*8;
		P[Player].axisX = P[Player].w/2;
		P[Player].axisY = P[Player].h;
		P[Player].dataAnim[1]  = 8;
		P[Player].dataAnim[2]  = 7;
		P[Player].dataAnim[3]  = 7;
		P[Player].dataAnim[4]  = 7;
		P[Player].dataAnim[5]  = 7;
		P[Player].dataAnim[6]  = 7;
		P[Player].animFrameTotal = 6;
		P[Player].sprite = SPR_addSpriteExSafe(...);
	}
...
}
```

P[n].id==1 significa Haohmaru, o primeiro personagem do jogo.
State é o estado do Player, neste caso, Parado (100)
P[n].x e P[n].y definem a posição do personagem na tela
P[n].w e P[n].h definem largura (width) e altura (height) do sprite (em pixels)
P[n].axisX e P[n].axisY definem a posição do ponto pivot da imagem
P[n].dataAnim[n] define o tempo que cada frame é exibido na tela; 0-255
P[n].animFrameTotal define a quantidade total de quadros desta animação, que deve ser igual a quantidade de dataAnim's definidos pelo programador.
P[n].sprite faz o carregamento da imagem / sprite deste movimento.

As imagens usadas na SGDK devem seguir o padrão do Genesis.
A partir deste momento chamaremos elas de sprites.
Esses sprites devem ter o tamanho multiplo de 8 pixels.
Cada conjunto de 8x8 pixels é chamado de TILE.
Além disso eles devem ter apenas 16 cores, sendo que a primeira cor da paleta é reservada para a cor transparente.





Cada conjunto de sprites, formando uma sequência de animação é chamada de Sprite Sheet, Folha de Sprites, ou simplesmente ESTADO, na HAMOOPIG. Este é o estado PARADO (100) do personagem Haohmaru.
Cada sprite desta sequência é chamado de FRAME.

IMPORTANTE: No canal do GameDevBoss existe um vídeo ensinando como iniciar seu primeiro projeto na SGDK! Assista ao vídeo para saber como compilar o HAMOOPIG em seu computador!

www.youtube.com/watch?v=H0XNUe4wY7E

Isso é o mais importante para começar.
Agora vamos a uma breve descrição das funções que você vai encontrar no código fonte, e o que você encontrará dentro delas.

void PLAYER_STATE(u8 Player, u16 State)
Recebe o Player(1;2) e o State(nnn). É a função que carrega os parâmetros de animacao do personagem, carrega a imagem do sprite, entre outras coisas importantes

void FUNCAO_PLAY_SND(u8 Player, u16 State)
Recebe o Player(1;2) e o State(nnn). Toca os sons do jogo, barulhos de golpes por exemplo

void FUNCAO_RELOGIO()
Atualiza o sprite do relogio, no HUD do jogo

void FUNCAO_BARRAS_DE_ENERGIA()
Atualiza as barras de energia e barras de especial

void FUNCAO_ANIMACAO()
Esta lindinha controla a espetacular animação dos personagens da HAMOOPIG, e também faz parte da FSM!

void FUNCAO_SPR_POSITION()
Ajusta a posição do sprite, depois que nós movemos ele

void FUNCAO_INPUT_SYSTEM()
Verifica os botões dos controles dos jogadores

void FUNCAO_FSM()
O Cérebro da engine. A própria engine.

void FUNCAO_FSM_HITBOXES(u8 Player)
Recebe Player (1;2). Carrega as informações sobre as caixas de colisão. É a alma de um jogo de luta consistente.

void FUNCAO_PHYSICS()
Move os objetos no jogo. Recomendo investigar como os pulos (300) e o movimento cair (550) foram programados. Um utiliza hard code, outro, um sistema de física extremamente simples e rápido criado pelo GameDevBoss para rodar especificamente no Mega Drive.

void FUNCAO_CAMERA_BGANIM()
Controla a camera e animação do background.

void FUNCAO_DEPTH(u8 Player)
Muda a ordem de empilhamento dos sprites.
Um ataque sempre deve ficar por cima, por exemplo.

void FUNCAO_SAMSHOFX()
Algumas coisas que dizem respeito ao Samurai Shodown 2, e portanto deve ser alterado ou suprimido em outros tipos de jogos.

void FUNCAO_DEBUG()
Mostra algumas informações importantes na tela, desde que você não esteja printando sprites em seus caracteres alfa numéricos rsrs

bool FUNCAO_COLISAO(s16 R1x1, s16 R1y1, s16 R1x2, s16 R1y2, s16 R2x1, s16 R2y1, s16 R2x2, s16 R2y2)
Pense numa função que faz testes de colisão extremamente rápidos.
Basta informar para ela o x1, y1, x2, y2 de um objeto e x1, y1, x2, y2 de outro, e ela vai retornar TRUE ou FALSE para você.

void FUNCAO_UPDATE_LIFESP(u8 Player, u8 EnergyType, s8 Value)
Recebe Player(1;2) EnergyType(1;2) e Value(nnn).
Permite tirar ou aumentar a energia normal ou especial dos Players.

void FUNCAO_INICIALIZACAO()
Inicializa alguns sistemas importantes

void FUNCAO_ROUND_INIT()
Inicializa o round pela primeira vez

void FUNCAO_ROUND_RESTART()
Reinicializa o round

void CLEAR_VDP()
Limpa a VDP; Video Display Processor

Me sigam lá no canal para mais informações e dicas sobre a engine.
Vamos criar juntos os melhores jogos para o Mega Drive!
O futuro é agora!

Daniel Moura, GameDevBoss

www.youtube.com/c/GameDevBoss