Grupo 3:

Trabalho realizado por:

Bruno Freitas Ferreira a13226 | Carlos Costa a13673 | Márcio Rocha a13218

Índice

Introdução 3

Descrição e mecânicas 3

Controlos 4

Mapas e sprites 5

Sons 9

Problemas e mudanças 9

Código 11

Máquinas de estado 16

Conclusão 18

Introdução

Neste trabalho, foi-nos dada a liberdade de fazer o nosso jogo sem qualquer tipo de limitações quanto à criatividade, portanto, nós decidimos fazer um jogo de luta.

Nós sabíamos que não ia ser fácil pois deixámos a barra muito alta para nós logo no início ao desafiar-nos a fazer um jogo deste tipo, mas isso era parte do que nós queríamos, desafiar as nossas capacidades ao nosso limite neste trabalho.

Descrição e mecânicas

O nosso jogo é um jogo de luta 1v1 num ambiente 2D em que os jogadores lutam em plataformas suspensas no ar e podem cair abaixo, perdendo a ronda se caírem fora do mapa. Cada jogador tem a sua vida e não há limite de tempo por ronda.

Numa luta no jogo, o jogador vencedor é aquele que ganha a melhor de 3 rondas, sendo todas as rondas no mesmo mapa e sempre que uma ronda começa, os jogadores são colocados nos seus cantos do mapa com a vida a 100%.

O jogador dispõe de dois saltos, um no chão e outro no ar, e três tipos de ataques para dois ataques diferentes em três diferentes situações que dá um total de 18 ataques diferentes. Ou seja, tem ataques leves, médios e pesados e estes podem ser socos ou pontapés, mas, também dependem se estão no ar, no chão de pé, ou aninhado. Cada um tem o seu dano mas quanto maior este é, mais lento é o ataque.

Também há certos ataques que são perfeitos para certas situações, como quando o inimigo está no ar e o jogador está no chão, há um soco que cobre a parte acima da sua cabeça, interrompendo o ataque do inimigo e etc.

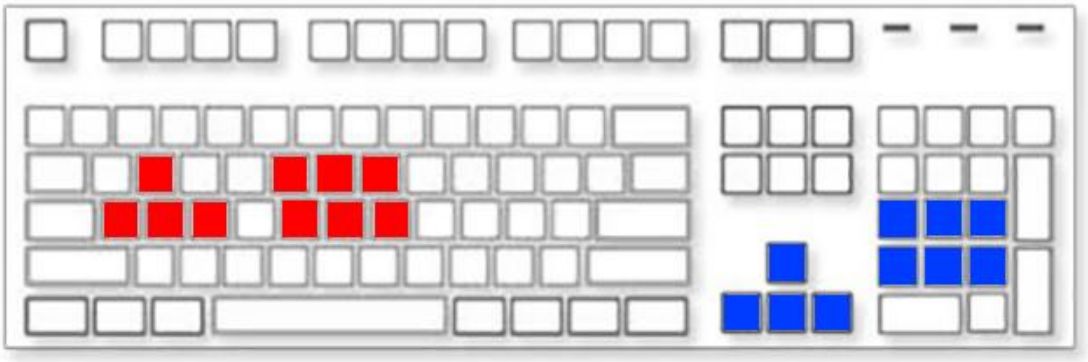
Outro pormenor que o jogo tem são combos. Estes, para além dos ataques normais referidos acima, são feitos através de uma combinação de teclas do movimento e depois com o pressionar de um ataque normal, este transforma-se no ataque especial.

O jogo é multiplayer local com o máximo de dois jogadores, ou seja, duas pessoas no mesmo computador.

Para fazer este jogo, nós inspirámo-nos no género de jogos de luta de *arcades* como o Street Fighter e jogos tipo Super Smash Bros e Brawlhalla, que fizeram parte da nossa infância e, no caso do Brawlhalla, era um jogo que na altura jogávamos muito.

Com este jogo, pretendemos atingir uma audiência dos 16 anos até os 25, pois é um jogo que requer habilidade.

Controlos



Vermelho – Jogador 1

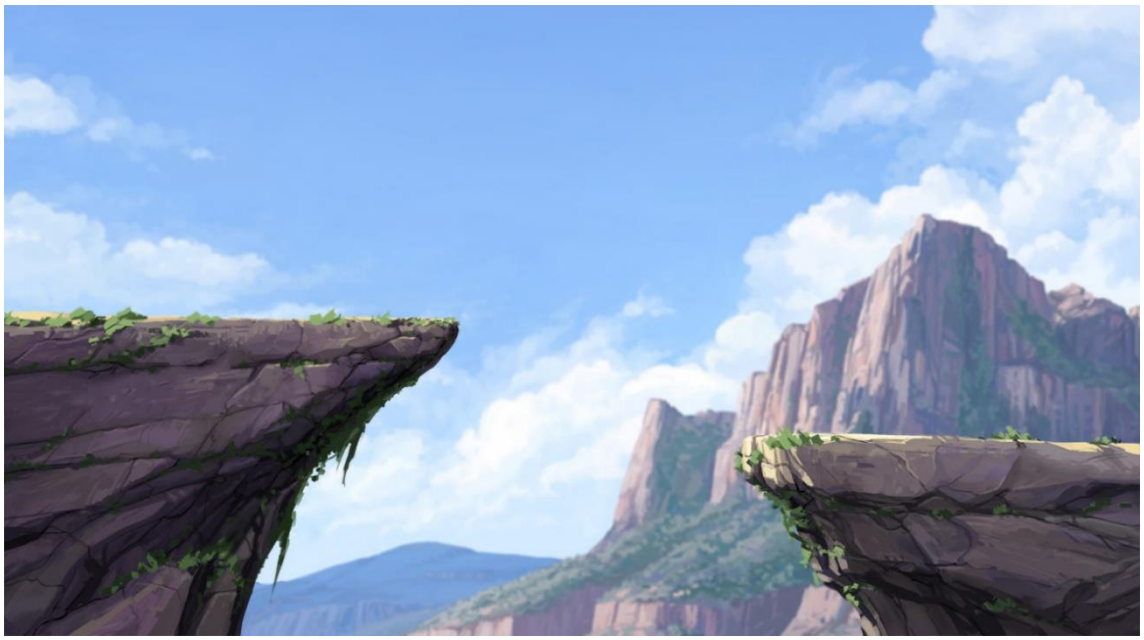
Azul – Jogador 2

T / 4 – Soco leve
Y / 5 – Soco médio
U / 6 – Soco pesado
G / 1 – Pontapé leve
H / 2 – Pontapé médio
J / 3 – Pontapé pesado
A / ← - Movimento para a esquerda
D / → - Movimento para a direita
W / ↑ - Salto
S / ↓ - Aninhar
S S D (ou A) T /↓ ↓ →(ou ←) 4 – Hadouken

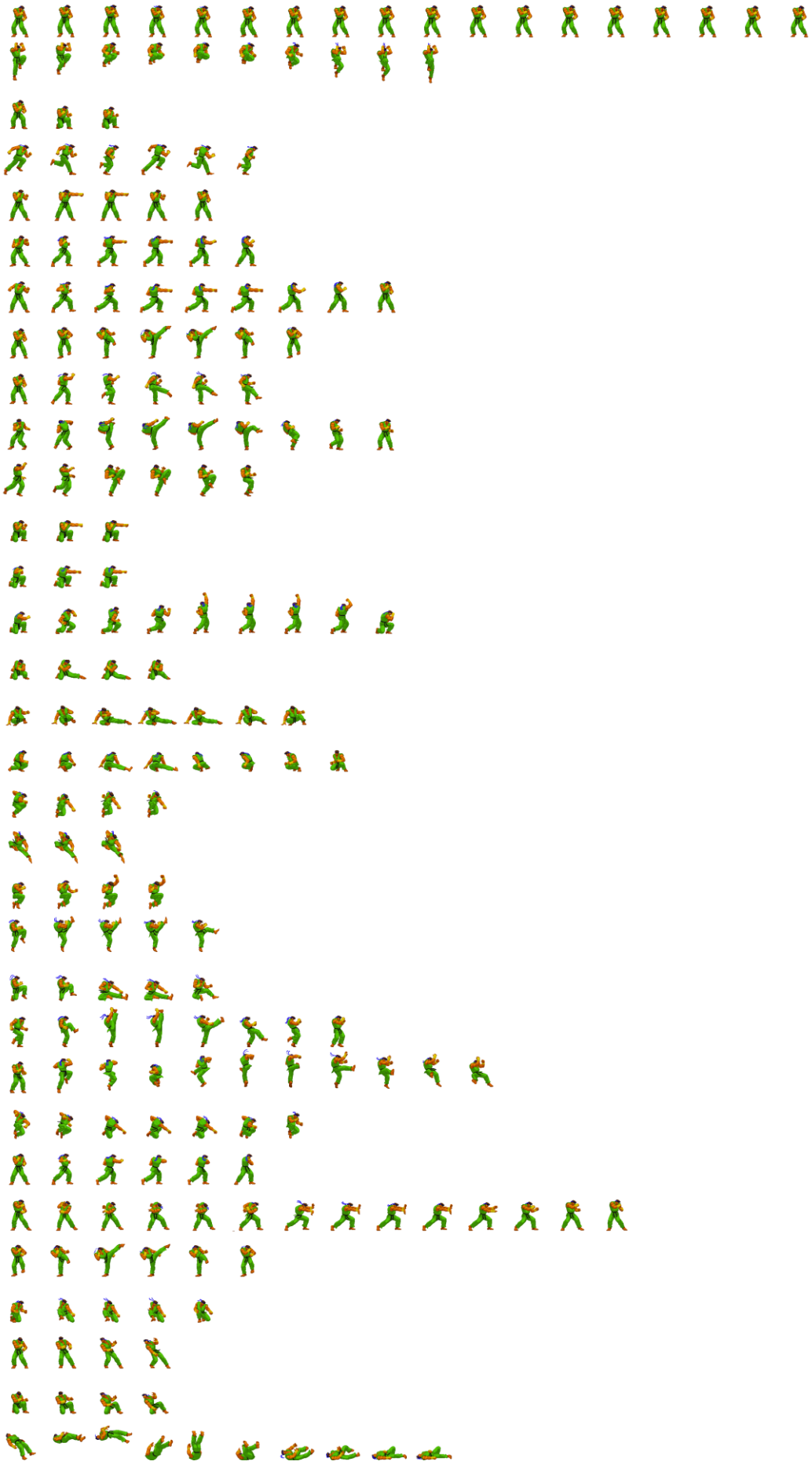
Mapas e sprites

O jogador, na seleção de mapas, pode escolher entre dois:

Mapa 1:



Mapa 2:



Hadouken

Ícone do jogo

Sons

Nós experimentamos pôr música e efeitos sonoros no jogo e tivemos vários problemas pois, no Monogame, ficheiros de som não é tão simples como arrastar o ficheiro para o Content como nas imagens, pois é necessário converter para xnb com um programa do Monogame.

Depois de termos descoberto isto, adicionámos algumas músicas de rock para experimentar e acabámos por as deixar pois gostamos do ambiente que dá ao jogo.

Também fizemos alguns ficheiros de som com a nossa boca, tipo "beatbox" para simular os sons de quando os ataques acertam ou falham, mas só implementamos quando acerta.

Problemas e mudanças

Menu

No início queríamos fazer um menu simples de estilo arcade em que o jogador apenas utilizaria as teclas baixo e cima para escolher a opção e depois clicava no enter, mas tivemos alguns problemas em tentar realizar isso. Então, tivemos a ideia de que no menu haveria uma personagem que pode atacar, e, para escolher a opção, atacava no ecrã a opção (como por exemplo uma porta que dizia start), visto que todo o que código de atacar e detetar os ataques já estava feito e então foi só reutilizá-lo.

Timer

No início pensámos em pôr um contador em todas as rondas que quando acabasse, a ronda acabava e ganhava o jogador com a maior vida ou algo do género mas tivemos alguns problemas com por isso a funcionar, pois por alguma razão, tanto o GameTime e o TimeSpan do Monogame, inicializavam-se a zero e não alteravam ao longo do tempo, portanto acabámos por desistir e portanto as rondas só acabam quando alguém morre ou cai.

Spritesheet

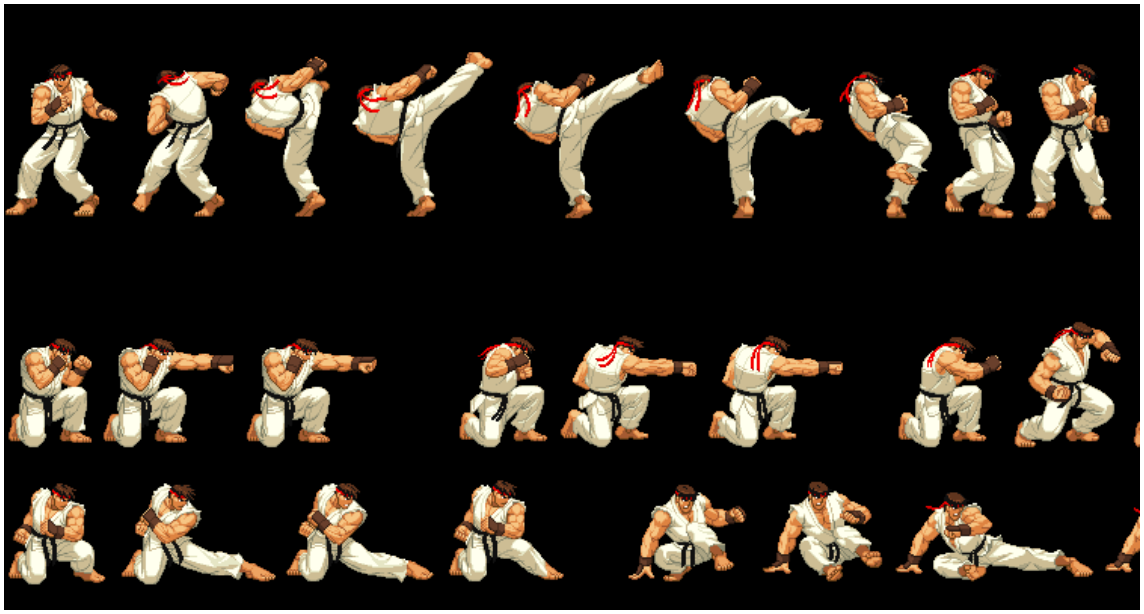
Outro problema, foi em utilizar a spritesheet do jogo Street Fighter, pois tivemos que a alterar em grande escala.

Primeiro, não fazia sentido em ter todas as animações na spritesheet e então cortámos o que não queríamos.

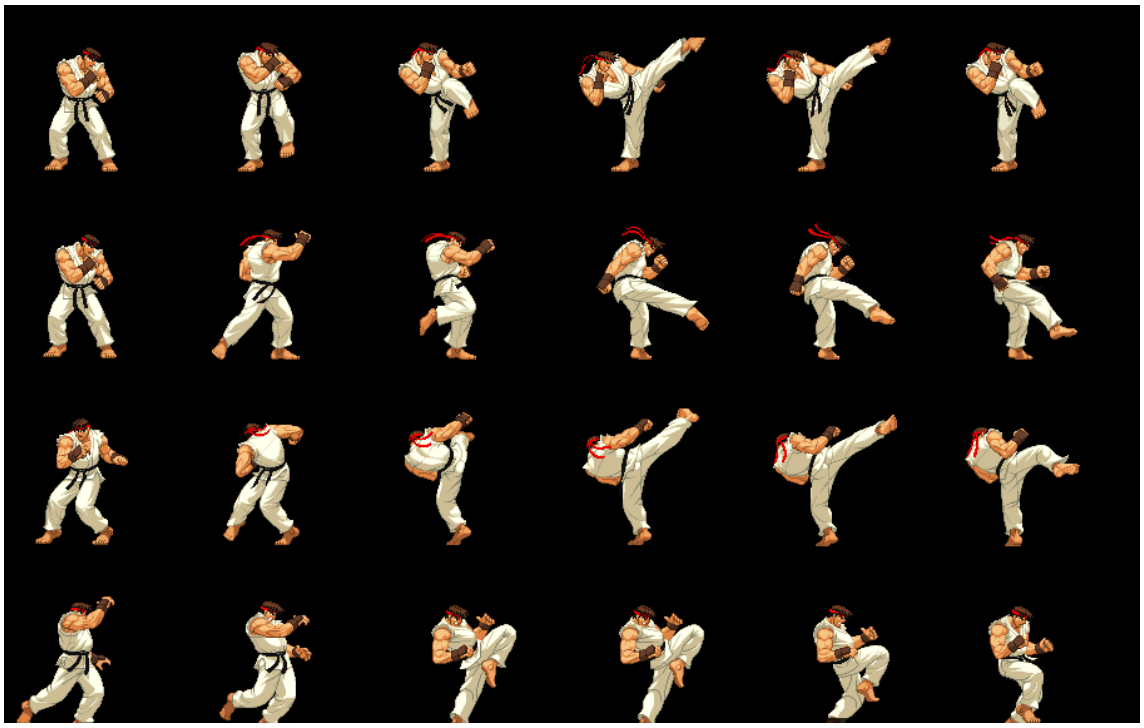
Segundo, a spritesheet original era muito complexa em termos de programação, pois ela foi compactada o máximo possível para ocupar o menos espaço possível na memória. Isto para nós complicava muito o código, pois teríamos que estar a alterar o tamanho do retângulo para desenhar em cada animação diferente. Por isso, tivemos a

ordená-la com um espaçamento igual para todas as animações, o que foi também esgotante e demorou bastante e não nos permitiu incluir outras personagens no jogo.

Spritesheet original:



Spritesheet modificada por nós:



Como se pode ver, esta está ordenada numa grelha, o que facilita bastante o código.

Colisões

Outro problema que tivemos, foi nós no início termos utilizado as funções de teste de colisão já presentes no Monogame, mas como este fazia testes de colisão pixel

a pixel em todos os *frames*, era muito intenso e baixava imensamente os *frames* por segundo do jogo, tornando o jogo não jogável. Então, para contornar este problema, nós decidimos fazer nós os nossos testes de colisões à mão utilizando retângulos, o que foi um trabalho enorme mas acabou por funcionar muito melhor.

Código

Som

Fizemos uma função para alterar o *pitch* dos efeitos sonoros, como, por exemplo, o som dos socos a acertar é apenas um único som, mas através do *pitch*, é dado um valor aleatório ao parâmetro de frequência do som já existente no Monogame.

```
private float GiveSoundEffectRandomPitch()
{
    float randomPitch = (float)(rnd.NextDouble() * (0.5 - (-0.5)) + ( -0.5));
    return randomPitch;
}
```

O cálculo pode-se fazer assim: rnd.NextDouble() * ((máximo - mínimo) + mínimo).

Esta função é depois chamada aqui e é atribuído o *pitch*.

```
public void PlaySoundEffectRandomPitch(SoundEffect effect, float volume)
{
    float pitch = GiveSoundEffectRandomPitch();
    effect.Play(volume, pitch, 0f);
}
```

Em termos de músicas, estas são todas carregadas para uma lista no início do jogo, e é depois escolhida aleatoriamente uma música da lista para ser reproduzida.

```
public void PlayRandomSong(List<Song> lista)
{
    int randomSong = rnd.Next(0, 2); //min <= rnd < max
    MediaPlayer.Volume = songVolume;
    song = lista[randomSong];
    MediaPlayer.Play(song);
}
```

Movimento

O movimento no início tentámos fazê-lo ao alterar um único vetor mas isso estava a criar muitos bugs no jogo. Portanto, decidimos criar as variáveis *valorX* e *valorY* e estas guardarem os valores do movimento e só no final do Update, é que o Vetor2D *Velocity* é alterado com estes valores, sendo esta a variável do movimento das personagens.

```
Velocity = (Vector2.UnitX * valorX) + (Vector2.UnitY * valorY);
```

Para simular a gravidade, nós temos função que decrementa o *valorY* todos os frames para que este.

```
//gravidade
if (!isGrounded)
{
    valorY -= gravity;
    if (valorY < fallingSpeedLimiter) valorY = fallingSpeedLimiter;
}
```

Também pusemos um fallingSpeedLimiter que, no fundo, serve para a velocidade do jogador, a cair, não aumente infinitamente, pois, acontecia um problema em que o jogador caía tão rápido que não detetava a colisão com a plataforma e então este limite resolve esse problema.

Combos

Para fazer os combos, tivemos alguns problemas pois as funções de Monogame de detetar as teclas pressionadas como, por exemplo, a função Keyboard.GetState().IsKeyDown(), pois nós tentámos pôr as teclas pressionadas numa queue, mas com esta função o que acontecia é que em todos os frames adicionava a mesma tecla na queue enquanto o jogador segurasse a tecla, ou seja, com um simples clique, registava várias vezes a mesma tecla como se tivesse sido clicada 50 vezes. Isto, obviamente, não podia ser utilizado para fazer combos.

Então criámos uma função que quando é pressionada a tecla, esta testa nos frames seguintes se ela foi largada e, só assim, é que insere na queue a tecla.

```
if (Keyboard.GetState().IsKeyDown(right))
    isRightDown = true;
if (isRightDown == true && Keyboard.GetState().IsKeyUp(right))
{
    isRightDown = false;
    movementKeyHistory.Enqueue(right);
}


if (Keyboard.GetState().IsKeyDown(left))
    isLeftDown = true;
if (isLeftDown == true && Keyboard.GetState().IsKeyUp(left))
{
    isLeftDown = false;
    movementKeyHistory.Enqueue(left);
}


if (Keyboard.GetState().IsKeyDown(down))
    isDownDown = true;
if (isDownDown == true && Keyboard.GetState().IsKeyUp(down))
{
    isDownDown = false;
    movementKeyHistory.Enqueue(down);
}
```

Depois de ter as teclas na queue, precisávamos de as testar, então retiramos a primeira tecla inserida e testámos se é a correta do combo. Se for, retirámos a segunda e assim até ao fim.

```
public bool CheckHadouken()
{
    if (movementKeyHistory.Dequeue() == down)
    {
        if (movementKeyHistory.Dequeue() == left || movementKeyHistory.Dequeue() == right)
        {
            return true;
        }
        return false;
    }
    return false;
}
```

Isto dava um problema que não nos apercebemos ao primeiro, mas o que acontecia é que no segundo if, em vez de retirar uma Key da queue e ver se era left ou right, o que tava a acontecer era que retirava uma Key e via se era left e depois retirava uma outra Key e via se era right. Para resolver isto pusemos o Dequeue numa variável e comparámos essa variável.

```
public bool CheckHadouken()
{
    if (movementKeyHistory.Dequeue() == down)
    {
        if (movementKeyHistory.Dequeue() == down)
        {
            Keys a = movementKeyHistory.Dequeue();
            if (a == left || a == right)
            {
                return true;
            }
            return false;
        }
        return false;
    }
    return false;
}
```

Animações

Estas são todas as animações que temos:

```
private enum CharState
{
    Idle,
    WalkingRight,
    WalkingLeft,
    Dead,
    Air,
    Stunned,
    MKick,
    MPunch,
    LPunch,
    LKick,
    HPunch,
    HKick,
    MKickAir,
    MPunchAir,
    LPunchAir,
    LKickAir,
    HPunchAir,
    HKickAir,
    Hadouken,
    Crouch,
    cLPunch,
    cMPunch,
    cHPunch,
    cLKick,
    cMKick,
    cHKick
}
```

Por estas animações todas a funcionarem através da spritesheet modificada por nós foi simples. No fundo testámos se as condições de ativar as animações estão a acontecer no Update e depois atualizamos a animação numa função à parte, onde tem as "coordenadas" da grelha onde a animação está e a sua duração.

Exemplo de um teste no Update:

```
if (Keyboard.GetState().IsKeyDown(mKick))
{
    valorX = 0;
    isAttacking = true;
    mCurrentCharState = CharState.MKick;
}
```

A sua posição na spritesheet:

```
//MEDIUM-----------------------------------------
case CharState.MKick:
    SetSpriteAnimation(7, 0, 7, 6, 3);
    break;
```

Ou seja, está na linha 7, começando no 0, na coluna 0 e acaba na linha 7 coluna 6, com um intervalo de 3 ticks entre cada frame da animação.

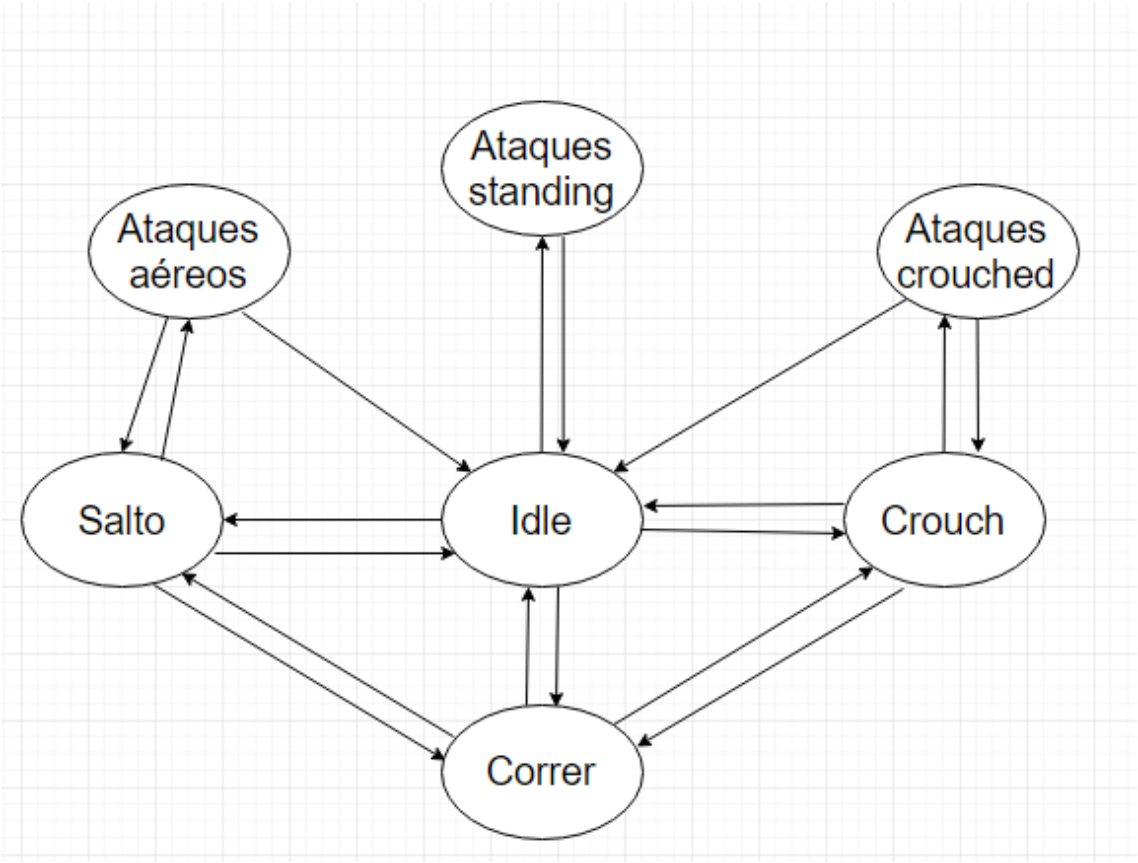
Depois temos esta função para todos os ataques para criar o retângulo da hitbox apenas por um frame quando chega a um ponto da animação.

```
//MEDIUM-----------------------------------------------------
case CharState.MKick:
    if (SpriteCurrentColumn == 3)
        attacks.MediumKick(mPosition, SpriteEffects);
    else
    {
        attacks.hitbox.Location = new Point(-100, -100);
    }
    break;
```

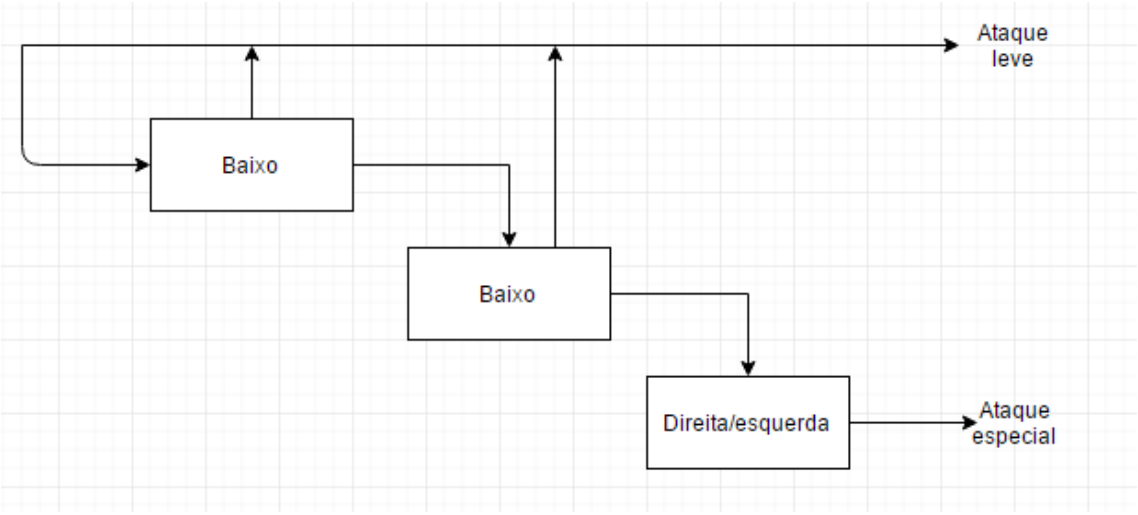
Outra coisa que fizemos foi alterar a classe SpritePrimitive para que as sprites pudessem parar de ser alteradas quando quiséssemos, para dar a ideia de que o tempo para quando alguém morre.

Máquinas de estados

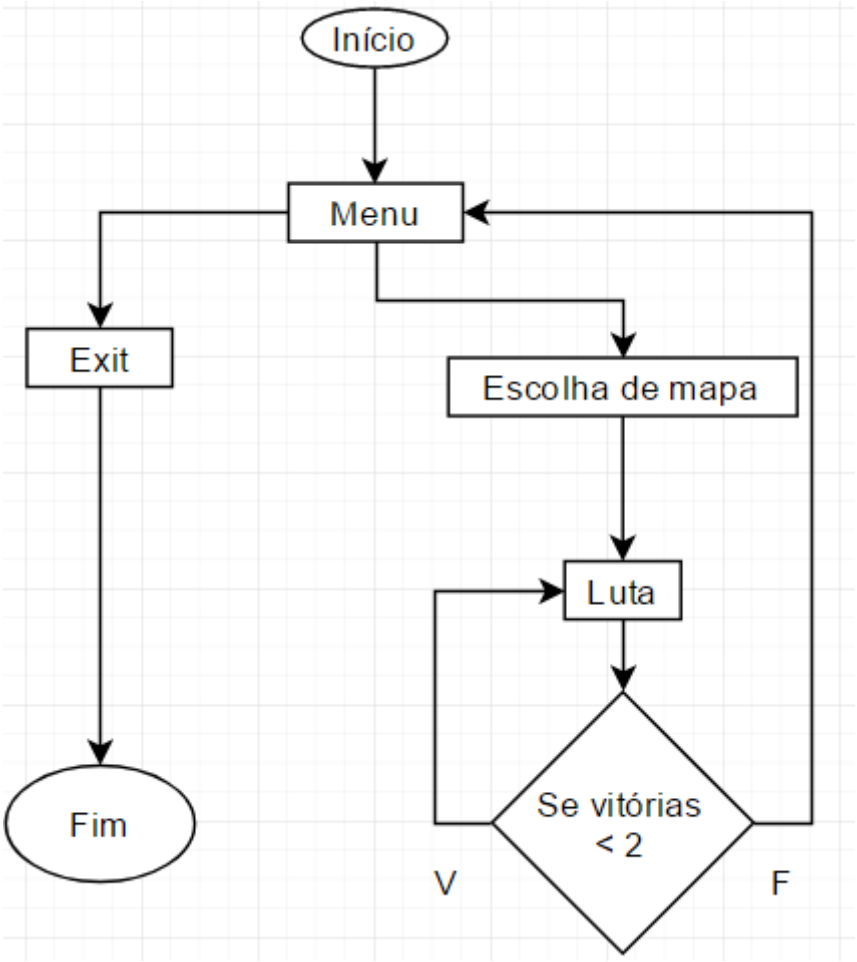
Movimento



Combos



Menu



Conclusão

Concluindo, nós estamos bastante contentes com o nosso trabalho e foi uma experiência espetacular em termos de desenvolvimento das nossas capacidades individuais e em equipa. Também, o facto de termos trabalhado em Monogame, foi uma experiência boa, pois, quase tudo, tem que ser feito por nós.

Nós tínhamos definido estes objetivos:

- 14 de abril - Físicas, movimentos
- 28 de abril - Ataques e hitboxes.
- 12 de maio - Sons e mapas.
- 28 de maio - Menu, personagens.
- Se houver tempo: AI, score, torneio

Conseguimos cumprir a primeira meta de 14 de abril, mas subestimamos as hitboxes e isto demorou até os últimos dias, mas à medida que fossemos trabalhando nas hitboxes fomos trabalhando em tudo o resto portanto acabámos por conseguir implementar tudo.

Uma coisa que gostávamos de implementar era mais personagens mas não conseguimos por causa de tempo, pois adequar a spritesheet e alterar as hitboxes é um processo muito exaustivo. Ainda melhor, gostávamos de ter feito as nossas próprias personagens, mas isso certamente é que não haveria tempo.

Outra coisa seria ter implementado uma espécie de AI, mas nós não sabemos nada sobre AI e também, novamente, por causa do tempo não conseguiríamos.

Também queríamos por uma espécie de torneio no jogo onde o jogador teria de luta contra AI para tentar ganhar um torneio, mas, como não temos AI nem várias personagens diferentes, não faria sentido implementar isto.

Link do github:
https://github.com/washinima/Street-Halla-Kombat