2017

Relatório do Projeto Final

Relatório do jogo

"Escape from war"

Fernando Nogueira Camilo Nº13233

André Moutinho Rodrigues Nº13231

6/6/2017

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

Introdução

No seguimento da avaliação da disciplina de técnicas de desenvolvimento de jogos, em que nos foi confiada a tarefa de desenvolver um jogo com base no livro "Learn 2D Game Development with C#", dos autores Kelvin Sung, Jebediah Pavleas, Rob Zhu, Jack Keng-Wei Chang, que foi a referência bibliográfica base para a disciplina , ou fazer um jogo completamente de raiz. Após a leitura cuidada do livro em questão o grupo de trabalho decidiu usar algumas das componentes do livro para servirem de base para o jogo que nos propusemos a fazer. Conceitos de como efetuar colisões pixel-a-pixel, animações de objetos de jogo e também simples inteligências artificias para esses mesmo objetos de jogo. Usamos também conceitos já explorados no relatório entregue para o trabalho anterior como a manipulação de "assests" visuais e como colocar texto na janela de jogo (visto que o monogame não o suporta diretamente), sendo que nestes conceitos o relatório irá apenas referir modificações efetuadas a essas classes e no conteúdo repetido fará referência ao relatório anterior.

Portanto este trabalho teve o objetivo de testar a capacidade dos alunos em conceber um jogo de raiz, dentro de um tempo limite reduzido de modo a colocar uma pressão adicional, funcionado como um emulador, à escala, de como seria trabalhar na indústria, a que esta licenciatura aspira a abrir portas para os seus alunos, a dos videojogos. Sendo esta uma indústria de alta pressão e alto investimento, em que o nosso país verifica uma grande dificuldade em acompanhar relativamente ao resto da Europa e com especial enfase para os Estados Unidos da América e o Japão que foram os pioneiros da área e ainda se encontram numa posição cimeira na hierarquia da indústria.

Visto isto então com o desenvolvimento deste projeto tivemos a possibilidade de ter um primeiro impacto, ainda que seja em ambiente académico, sobre como será a nossa vida profissional.

Decidimos então criar um jogo na categoria de "*shooter*" 2D, "*pixel-art*", baseado em "*stealth*" como uma homenagem aos originais do criador **Hideo Kojima** que nos finais da década de 80 foi encarregue pela sua empresa empregadora de fazer um jogo de ação mas como na altura os computadores não conseguiam ter mais de 4 texturas em movimento (jogador + três inimigos) ele idealizou um conceito para se adaptar as circunstâncias ("*thinking outside of the box*") que passou a ser designado de "*stealth action*" que virava o conceito de ação do avesso tornando o conceito habitual, de eliminar vários inimigos como um herói, num mais realista inspirado nos espiões da CIA.

Dai advém o nome do nosso jogo "Escape from War" em que o jogador tenta escapar dos seus inimigos sem ser detetado, e em que o protagonista "*Plinskin*" um espião de origem Russa que desertou o seu país, tenta de acordo com as informações; que conseguiu adquirir, tenta destruir um armazém que contém armamento nuclear e assim impedir uma terceira e final guerra mundial.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

Estrutura base do jogo

Este jogo foi feito usando o "*MonoGame*" em linguagem C# e toda a sua estrutura vai ser descrita por classes e a influência dessas classes no resultado final.

Como já foi referido na introdução ao relatório questões como o carregamento de imagens organização do projeto por pastas, entre outros, que já foram dissecadas no relatório anterior serão referenciadas, mas não ao pormenor limitando este relatório a conteúdo novo.

Começando então pela classe que permite colisões de pixéis (TexturasPixel), sendo que o objetivo desta classe é colocar todos os pixéis de uma imagem num array de tipo Color e assim verificar se existe sobreposição de pixéis e consequentemente uma colisão, no início da classe é definido um array só para a classe (private) que irá então guardar as cores dos pixéis, depois é criado um dicionário que guarda então esse array de cores dos pixéis de acordo com o nome da imagem correspondente para cores iguais em imagens diferentes não serem interpretadas como do mesmo pixel, sendo que depois essas cores da imagem são obtidos através da propriedade "*GetData*" das texturas 2D do *monogame*, o método LeDadosCor permite adicionar um novo array de pixéis ao dicionário caso o nome da imagem ainda não esteja presente e o método BuscaCores recebe um índice permitindo assim obter a cor de um pixel específico do array de cores. O método PixeisTocaram é que permite saber os objetos de jogo se tocaram e assim efetuar a colisão, sendo que este recebe a textura e devolve um bool que define se houve ou não colisão, para além de também retornar através de um "out" como parâmetro o ponto onde houve colisão.

Este método começa por definir uma variável designada de toca que é o bool, que usa o método ToqueTexturas para verificar se os limites das imagens se sobre puseram e se sim então houve toque, depois é usada uma condição em que caso tenha havido um toque então vai ser verificado se esse toque é de pixel colocando a variável (toquepixel) inicialmente como false, depois são adquiridas as direções de cada objeto para depois ser possível dar o ponto de colisão, são então usados dois ciclos while para verificar todos os possíveis pixéis, então enquanto não existe colisão e a procura esta dentro do limite da imagem, são adquiridos todos os pixéis da imagem e o índice desses no espaço do jogador, caso esses pixéis não sejam transparentes (CorI.A > 0), são adquiridos os índices da outra textura e caso nesse índice as cores também não sejam transparentes quer dizer então que houve um toque entre pixéis de cores não transparentes em imagens diferentes.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
public bool PixeisTocaram(Texturas outraTextura, out Vector2 pontoColisao)
{
    bool toca = ToqueTextures(outraTextura);
    pontoColisao = Posicao;
    if (toca)
    {
        bool toquePixel = false;

        Vector2 direccaoIX = MostraVector.ModificarPeloAng(Vector2.UnitX, Rotacao);
        Vector2 direccaoIY = MostraVector.ModificarPeloAng(Vector2.UnitY, Rotacao);

        Vector2 outraDirX = MostraVector.ModificarPeloAng(Vector2.UnitX, outraTextura.Rotacao);
        Vector2 outraDirY = MostraVector.ModificarPeloAng(Vector2.UnitY, outraTextura.Rotacao);

        int i = 0;
        while ((!toquePixel) && (i < SpriteLargura))

        {
            int j = 0;
            while ((!toquePixel) && (j < SpriteAltura))
            {
                pontoColisao = IndiceDaPosicaoCamera(i, j, direccaoIX, direccaoIY);
                Color corI = BuscaCores(i, j);
```

```
                if (corI.A > 0)
                {
                    Vector2 outroIndex = outraTextura.PosicaoCameraIndex(pontoColisao, outraDirX, outraDirY);
                    int xMin = (int)outroIndex.X;
                    int yMin = (int)outroIndex.Y;

                    if ((xMin >= 0) && (xMin < outraTextura.SpriteLargura) && (yMin >= 0) && (yMin < outraTextura.SpriteAltura))
                    {
                        toquePixel = (outraTextura.BuscaCores(xMin, yMin).A > 0);
                    }
                }
                j++;
            }
            i++;

        }
        toca = toquePixel;
    }
    return toca;
```

André Rodrigues Nº13231
Fernando Camilo Nº13233

Por último na classe temos dois métodos o primeiro devolve um vetor com nova direção de acordo com um índice na janela de jogo e o outro faz o mesmo, mas calculando a nova direção de acordo com outro vetor recebido com argumento.

```
private Vector2 IndiceDaPosicaoCamera(int i, int j, Vector2 direcaoX, Vector2 direcaoY)
{
    float x = i * Largura / (float)(SpriteLargura - 1);
    float y = j * Altura / (float)(SpriteAltura - 1);

    Vector2 novoVector = Posicao + (x - (tamanhoI.X * 0.5f)) * direcaoX - (y - (tamanhoI.Y * 0.5f)) * direcaoY;

    return novoVector;
}

private Vector2 PosicaoCameraIndex(Vector2 p, Vector2 DireccaoX, Vector2 DireccaoY)
{
    Vector2 delta = p - Posicao;
    float xOffset = Vector2.Dot(delta, DireccaoX);
    float yOffset = Vector2.Dot(delta, DireccaoY);
    float i = SpriteLargura * (xOffset / Largura);
    float j = SpriteAltura * (yOffset / Altura);
    i += SpriteLargura / 2;
    j = (SpriteAltura / 2) - j;
    return new Vector2(i, j);
}
```

Quanto à classe camera, foram adicionados métodos que permitem zoom desta ao aumentar/reduzir a largura da mesma através da variável delta (alterada através das teclas L e K para zoom in e out por esta ordem), um outro método que permite verificar as colisões de um objeto de jogo com os limites da camera (verificando se colide com qualquer um dos quatro lados possíveis) e caso não colida devolve o estado que indica que esta dentro da área de jogo, sendo que este método vai permitir depois na classe de estado de jogo que a camera se mova de acordo com o jogador e por último o método PosicaoAleatoria que define uma área possível onde colocar um objeto aleatório dentro dos limites da janela de jogo.

```
static public void ZoomCamera(float deltaX)
{
    float ogLargura = largura; // Guarda altura original da janela.
    float ogAltura = altura; // Guarda altura original da janela.

    largura =largura + deltaX; // Modifica o valor da largura.
    ratio = -1f;
    RazaoAreaJogo(); // Chamar o ratio da area do jogo.

    float dX = 0.5f * (largura - ogLargura);
    float dY = 0.5f * (altura - ogAltura);
    origem -= new Vector2(dX, dY);
}
```

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
static public EstadoColisaoCamera ColisaoComJanela(Texturas prim)
{
    Vector2 min = posicaoCantoInferiorEsq;
    Vector2 max = posicaoCantoSuperiorDir;

    if (prim.LimiteMaxx.Y > max.Y)
        return EstadoColisaoCamera.ColisaoSuperior;
    if (prim.LimiteMin.X < min.X)
        return EstadoColisaoCamera.ColisaoEsq;
    if (prim.LimiteMaxx.X > max.X)
        return EstadoColisaoCamera.ColisaoDir;
    if (prim.LimiteMin.Y < min.Y)
        return EstadoColisaoCamera.ColisaoInferior;

    return EstadoColisaoCamera.DentroJanela;


}
```

```
static public Vector2 PosicaoAleatoria()
{

    float rangeX = 0.8f * largura;
    float offSetX = 0.1f * largura;
    float rangeY = 0.8f * altura;
    float offSetY = 0.1f * altura;

    float x = (float)(Game1.numAleatorios.NextDouble()) * rangeX + offSetX + origem.X;
    float y = (float)(Game1.numAleatorios.NextDouble()) * rangeY + offSetY + origem.Y;

    return new Vector2(x, y);

}
```

Passando agora para a classe SpriteTexturas que herda todas as características da classe ObjectoDeJogo que existia no projeto anterior, em que esta nova classe vai permitir animar os objetos de jogo através de "*SpriteSheets*" , usando variáveis para o tamanho da imagem que se pretende usar, três variáveis para obter o número de colunas, linhas e espaços (entre animações) existentes na "*SpriteSheet*" , depois dentro da animação utiliza-se duas variáveis para definir o período para passar para a próxima animação e o tempo que se fica em cada imagem, também para animação são usadas seis variáveis para definir os limites que se pretende para cada animação, assim sendo duas variáveis são para as linhas e colunas onde se pretende iniciar a animação, outras duas para as colunas e linhas de final de animação e as últimas para reter a animação em que se está. Depois usa-se um construtor da classe para iniciar as variáveis em que a largura da sprite se obtém através da razão da largura da imagem pelo número de colunas e a altura se obtém pela razão entra a altura da imagem pelo seu número de linhas assim temos o tamanho de cada imagem usada para animação de forma individual. Todas as variáveis relativas às animações são inicializadas a zero com exceção para o timer para a próxima que é inicializado a 1 depois são criados get/set

para cada variável de forma a puderem ser usadas ao longo do programa. No que a métodos diz respeito a classe possui um que permite definir como pretendemos as animações chamado de DefineAnimaçao, um método que efetua a animação chamada Update que verifica se o tempo que está na animação é superior ao definido para avançar na animação e se for avança para a próxima coluna e caso a próxima coluna seja superior à última então volta à inicial fazendo "*reset*" à animação e entrando num "*loop*" infinito que é a animação da personagem (o mesmo se verifica se a linha atual for superior à final). Por último o método Draw permite passar este objeto animado para a janela de jogo, havendo no fim um "*Override*" da classe para esta permitir usar as funcionalidades da classe que permite a colisão entre pixéis.

```
public void DefineAnimacao(int linhainicial, int colunainicial, int linhafinal, int colunafinal, int timer)
{
    timerProximaAnimacao = timer;
    linhaInicial = linhainicial;
    colunaInicial = colunainicial;
    linhaFinal = linhafinal;
    colunaFinal = colunafinal;

    //Inicializar animacao
    linhaActual = linhaInicial;
    colunaActual = colunaInicial;
    timerActualAnimacao = 0;


}
```

```
public override void Update()
{
    base.Update();

    //Update estado Sprite
    timerActualAnimacao++;

    if (timerActualAnimacao > timerProximaAnimacao)
    {
        timerActualAnimacao = 0;//Comeca a proxima imagem de sprite

        colunaActual++;//Coluna a seguir
        if (colunaActual > colunaFinal)
        {
            //Reinicia a animation
            colunaActual = colunaInicial;
            colunaActual++;

            if (linhaActual > linhaFinal)
                linhaActual = linhaInicial;
        }

    }
}
```

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
public override void Draw()
{
    //localizacao e tamanho da area a sert desenhada em pixel space
    Rectangle areaTextura = Camera.ConvercaoParaPixeisRectangulo(Posicao, Tamanho);

    int topoImagem = linhaActual * SpriteILargura;
    int esqImagem = colunaActual * SpriteIAltura;
    //define a area e a localizacao a printar da SpriteSheet para a area de jogoo
    Rectangle areaSprite = new Rectangle(esqImagem + numeroEspacos, topoImagem + numeroEspacos, SpriteILargura
        , SpriteIAltura);

    //Origem da rotacao (ferencia de rotacao)
    Vector2 origem = new Vector2(SpriteILargura / 2, SpriteIAltura / 2);

    //Print das texturas
    Game1.spriteBatch.Draw(assetVisual, areaTextura, areaSprite, Color.White, rotacaoI, origem, SpriteEffects.None, 0f);

    if (null != Texto)
        TextoCust.PrintTextoLocalizacao(Posicao, Texto
            , CorTexto);
}
```

Começando agora pela primeira classe que suporta os inimigos do jogo chamada de InimigoPatrulha que possui quatro estados possíveis (em patrulha, em perseguição, "*stunned*" ou morto) que são definidos por um enumerador designado por EstadoPatrulha e outro chamado de TipoPatrulha que define os três possíveis tipos de patrulha (Aleatória, no eixo horizontal ou vertical).

```
protected enum EstadoPatrulha
{
    Patrulha, Perseguicao, Stunned, Morto
}
protected enum TipoPatrulha
{
    Aleatoria, MovHorizontal, MovVertical
}
```

Quanto a variáveis duas definem a velocidade de perseguição e patrulha, outras duas definem o raio de perceção por parte do inimigo, ou seja, a variável raioPersepçao assume o valor de 20 que define a área em que o inimigo deve procurar e a raioParaPerseguir define o raio para quando o inimigo começa a perseguir o herói, o timer de estado por definição é de cerca de 5 segundos que é o tempo que demora a mudar de estado, e o tempo stunned corresponde a metade desse tempo que demora a mudar de estado. Duas para o tamanho do inimigo e uma para a borda do inimigo que vai ser usada para quando aparecerem inimigos aleatórios fora da janela em estado de perseguição.

```
private const float velociadePatrulha = 0.2f;
private const float raioPersecpcao = 20f;//Para outros inimigos
private const float raioParaPerseguir = 40f;//Para iniciar o estado de pereguicao
private const int timerEstado = 60 * 5;//Sendo que os Updates sao de 60 por segundo demora 5 segundos a mudar de estado
private const int tempoStunned = timerEstado / 2;
private const float velocidadePerseguicao = 0.3f;
protected const float larguraInimigo = 10f;
protected const int tamanhoIinimigo = 1;
private const float BordaRaio = 0.55f;
```

André Rodrigues Nº13231
Fernando Camilo Nº13233

Depois temos três variáveis que suportam as cores para que mudam os inimigos quando mudam de estado, uma variável que suporta a posição do alvo a perseguir, três instâncias de enumerados para suportarem o tipo e estado atual da patrulha e também o tipo atual de inimigo, dois bool para verificar se os inimigos podem rodar e uma para a destroiFlag que define se o herói foi morto, por último são definidos dois inteiros um para o tempo que está preso num estado e outro para o tamanho inicial e get/set para este último e o destroiFlag. Depois definimos um construtor que coloca a posição do alvo a zero, o vetor velocidade com o valor 1 no eixo dos yy, a cor que tinge a imagem para o padrão definido anteriormente para patrulha, o tipo de patrulha aleatória, posição da patrulha para um local aleatório, o bool de morte do inimigo para falso, definição de animação para as linhas inicias e o tipo de inimigo para soldado interno.

```
private Color corTinge = Color.Black;
private Color corTingePerseguicao = Color.OrangeRed;
private Color corTingeStunned = Color.LightCyan;

private Vector2 posicaoAlvo;
private EstadoPatrulha estadoActual;
protected TipoPatrulha tipoPatrulha;
protected TipoEnimigo tipoActual;
protected bool permiteRotacao;

private int timerEmEstado;

private bool destroiFlag;

public bool DestroiFlag
{ get { return destroiFlag; } }

protected int tamanhoIni;
public int TamanhoIni
{
    get { return tamanhoIni; }
    set
    {
        tamanhoIni = value;
        this.Tamanho = new Vector2(tamanhoIni * larguraInimigo + larguraInimigo,
            tamanhoIni * larguraInimigo + larguraInimigo);

    }
```

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
public InimigoPatrulha(String imagem, Vector2 posicao, Vector2 tamanho, int contagemLinha, int contagemColuna, int espacosImagens) :
    base(imagem, posicao, tamanho, contagemLinha, contagemColuna, espacosImagens)
{
    //causa um update no estado
    posicaoAlvo = Posicao = Vector2.Zero;
    VelocidadeVector = Vector2.UnitY;
    tintImage = corTinge;
    tipoPatrulha = TipoPatrulha.Aleatoria;
    Posicao = PosicaoAleatoria(true);
    destroiFlag = false;
    DefineAnimacao(0, 0, 1, 1, 10);
    TamanhoIni = tamanhoIinimigo;
    tipoActual = TipoEnimigo.SoldadoExterno;
    //DEPOIS adicionaar o externo



}
```

Quanto a métodos definimos um método que faz o Update do estado do inimigo que verifica se o estado não é se stunned caso não seja calcula a direção para o alvo (método que permite ajustar a direção para o alvo caso seja permitida a rotação e caso não seja verifica se está na direção correta do eixo dos xx) depois verifica o estado atual, em que caso seja de patrulha verifica se o timer de estado é inferior a 0 ou se a distância para o alvo está dentro do raio de perceção e caso não esteja continua em patrulha em que se for aleatória procura próximo alvo, caso horizontal , gera uma patrulha de movimento no eixo dos xx e caso vertical gera uma patrulha no eixo dos yy.

```
public bool UpdatePatrulha(Heroi hero, out Vector2 posicaoApanhado)
{

    bool apanhado = false;
    posicaoApanhado = Vector2.Zero;

    timerEmEstado--;

    //Comum a todos os estados
    if(estadoActual!=EstadoPatrulha.Stunned)
    {
        base.Update();
        Vector2 paraHeroi = hero.Posicao - Posicao;
        paraHeroi.Normalize();
        Vector2 paraAlvo = posicaoAlvo - Posicao;
        float distanciaAlvo = paraAlvo.Length();
        paraAlvo.Normalize();//Diferente do codigo
        CalculoNovaDireccao(paraAlvo, paraHeroi);
```

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
        switch(estadoActual)
        {
            case EstadoPatrulha.Patrulha:
                UpdatePatrulhaEstado(hero, distanciaAlvo);
                break;
            case EstadoPatrulha.Perseguicao:
                apanhado = UpdateEstadoPerseguicao(hero, distanciaAlvo, out posicaoApanhado);
                break;
        }

    }
    else
    {
        UpdateEstadoStunned(hero);
    }
    return apanhado;
}
```

Por último se o estado for de perseguição verifica se foi apanhado com o método UpdateEstadoPerseguicao que verifica se houve toque de pixéis e a posição do alvo, caso tenham tocado quer dizer que o herói foi apanhado e o tipo de inimigo retira um ponto de vida ao herói matando-o (destroiFlag = true) caso não tenha sido apanhado e o timer de estado seja inferior a zero então passa para o próximo alvo, no caso de nenhum dos estados se verificar o inimigo tem que estar num estado de stunned. No fim os métodos devolvem sempre um bool para verificar se foi apanhado. No que ao resto dos métodos diz respeito criamos uma que verifica se pode detetar o herói ao ver se a distância para a posição do herói entra no raio de perceção do inimigo e se assim for passa para estado de perseguição, o método que permite passar para esse estado coloca o timer de estado a 1,5 vezes o timerEstado, muda o enum para perseguição, define a posição alvo para a posição do herói e tinge a imagem da cor de perseguição. O mesmo faz o DefineParaEstadoStunned, mas para o efeito de stunned e o método de UpdadeEstadoStunned apenas verifica se o timer de estado já permite sair do estado stunned. Agora os métodos de geração de inimigos, primeiro o que gera novas posições de procura (ProximoAlvo) que gera um número aleatório para o tempo do início de estado mediante o que gerar define a posição da janela onde vai procurar fazendo sempre um reset ao timer para ter posições de procura aleatórias, o GeraInimigosForaX verifica a posição no eixo dos x max e min para gerar um novo inimigo fora da janela e os métodos de posição aleatórias definem a região da janela onde vão aparecer, o de PosicaoAleatoria define uma posição completamente aleatória para aparecer os dois últimos métodos geram os inimigos de movimento vertical e horizontal.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
private bool UpdateEstadoPerseguicao(Heroi hero, float distaciaHeroi, out Vector2 posicao)
{
    bool apanhado = false;
    apanhado = PixeisTocaram(hero, out posicao);//Podese usar aqui o metode de deteteccao do triangulo
    posicaoAlvo = hero.Posicao;

    if (apanhado)
    {
        switch (tipoActual)
        {
            case TipoEnimigo.SoldadoInterno:
                hero.AjustaVida(-1);
                this.TamanhoIni--;
                this.destroiFlag = true;
                break;
            case TipoEnimigo.SoldadoExterno:
                hero.AjustaVida(-1);
                this.TamanhoIni--;
                this.destroiFlag = true;
                break;
                //codigo alterado da fonte
        }
    }
    else if (timerEmEstado < 0)
        ProximoAlvo();//quando acaba o tempo passa para o proximo

    return apanhado;
}
```

```
public void  DefineParaEstadoPerseguicao(ObjectoDeJogo heroi)
 {
     timerEmEstado = (int)(timerEstado * 1.5f);
     Velocidade = velociadePatrulha;
     estadoActual = EstadoPatrulha.Perseguicao;
     posicaoAlvo = heroi.Posicao;
     tintImage = corTingePerseguicao;
 }
     private void CalculaNovoTimerVelocidade_Reset()
 {
     Velocidade = velociadePatrulha * (0.8f + (float)(0.4 * Game1.numAleatorios.NextDouble()));
     timerEmEstado=(int)(timerEstado*(0.8f+(float)(0.6*Game1.numAleatorios.NextDouble())));
 }
```

```
public void DefineParaEstadoStunned()
{
    tintImage = corTingeStunned;
    timerEmEstado = tempoStunned;
    estadoActual = EstadoPatrulha.Stunned;
    AudioSupport.PlaySom("Stunned");
}
private void UpdateEstadoStunned(Heroi hero)
{
    if (timerEmEstado < 0)
        DefineParaEstadoPerseguicao(hero);
}
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
private void ProximoAlvo()
{
    timerEmEstado = timerEstado;
    estadoActual = EstadoPatrulha.Patrulha;
    tintImage = corTinge;
    double InicioEstado = Game1.numAleatorios.NextDouble();
    if (InicioEstado < 0.25)
        posicaoAlvo = PosiçaoAleatoriaInfDir();
    else if (InicioEstado < 0.5)
        posicaoAlvo = PosiçaoAleatoriaSupDir();
    else if (InicioEstado < 0.75)
        posicaoAlvo = PosiçaoAleatoriaSupEsq();
    else
        posicaoAlvo = PosiçaoAleatoriaInfEsq();

    CalculaNovoTimerVelocidade_Reset();
}
```

```
private Vector2 GeraInimigosForaX(double xFora, double yFora)
{
    Vector2 max = new Vector2(PosicaoX + Camera.Largura / 2, Camera.posicaoCantoSuperiorEsq.Y);
    Vector2 min = new Vector2(PosicaoX - Camera.Largura / 2, Camera.posicaoCantoInferiorEsq.Y);
    float x = min.X + tamanhoI.X * (float)(xFora+(BordaRaio* Game1.numAleatorios.NextDouble()));
    float y = max.Y + tamanhoI.Y * (float)(yFora+ (BordaRaio * Game1.numAleatorios.NextDouble()));
    return new Vector2(x, y);

}
```

```
public Vector2 PosicaoAleatoria(bool foraCamera)
{
    Vector2 posicao;
    float posX = (float)Game1.numAleatorios.NextDouble() * Camera.Largura * 0.80f + Camera.Largura * 0.10f;
    float posY = (float)Game1.numAleatorios.NextDouble() * Camera.Altura* 0.80f + Camera.Altura * 0.10f;

    if (foraCamera)
        posX += Camera.posicaoCantoSuperiorDir.X;

    posicao = new Vector2(posX, posY);
    return posicao;
}
```

```
private void GeraInimigo_UpDown()
{
    timerEmEstado = timerEstado;
    estadoActual = EstadoPatrulha.Patrulha;
    tintImage = corTinge;
    float posY;
    //Usa os limites da camera
    float distanciaLIMTopo = Camera.posicaoCantoSuperiorEsq.Y - PosicaoY;
    float distanciaLIMInf = PosicaoY - Camera.posicaoCantoInferiorEsq.Y;
    if(distanciaLIMTopo>distanciaLIMInf)//Caso a distancia ate ao limite superior fosse maior entao produz um movimwnto superior caso contrario um movimento inferior
    {
        posY = (float)Game1.numAleatorios.NextDouble() * distanciaLIMTopo / 2 * 0.8f + PosicaoY + distanciaLIMTopo / 2;
    }
    else
    {
        posY = (float)Game1.numAleatorios.NextDouble() * -distanciaLIMInf / 2 * 0.8f + PosicaoY + distanciaLIMInf / 2;
    }
    posicaoAlvo = new Vector2(PosicaoX, posY);
    CalculaNovoTimerVelocidade_Reset();
}
```

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Licenciatura em Engenharia em desenvolvimento de jogos de digitais
2016/2017

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
private void GeraInimigo_LefRight()
{
    timerEmEstado = timerEstado;
    estadoActual = EstadoPatrulha.Patrulha;
    tintImage = corTinge;
    float posX;

    if(VelocidadeVector.X<=0)
    {
        posX = (float)Game1.numAleatorios.NextDouble() * Camera.Largura / 2 + PosicaoX;
    }
    else
    {
        posX = (float)Game1.numAleatorios.NextDouble() * -Camera.Largura / 2 + PosicaoX;
    }
    CalculaNovoTimerVelocidade_Reset();
}
```

A classe DefineInimigo cria uma lista de inimigos, faz Spawn de inimigos e adiciona à lista, o método SpawnInimigos usa um gerador de números aleatórios para puder ter um número sempre diferente de cada tipo de inimigos caso seja um 0 é criada uma instância da classe cão e caso seja um 1 é criada uma instância da classe soldadoInterno, o método UpdateDefiniçao verifica a que distância colocamos os inimigos à medida que o herói avança no nível e caso seja destruído então esse é removido da lista e adicionado um novo inimigo, caso seja por disparos esses são contabilizados e o inimigo fica stunned no fim é percorrida a lista e é feito um recolocar dos inimigos através do método RespawnInimigos e por último o método Drawset para fazer print dos inimigos na tela de jogo.

```
public InimigoPatrulha SpawnInimigos()
{
    int numAleatorio = (int)(Game1.numAleatorios.NextDouble() * 3);
    InimigoPatrulha inimigo = null;
    switch(numAleatorio)
    {
        //case (int)TipoEnimigo.SoldadoInterno:
        //    inimigo = new SoldadoInterno();
        //    break;
        //case (int)TipoEnimigo.SoldadoExterno:
        //    inimigo = new SoldadoExterno();
        default:
            break;
    }
    return inimigo;
}
```

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
public int UpdateDefinicao(Heroi hero)
{
    int count = 0;
    Vector2 posicaoToque;

    //Add an enemy at 100m and every 50 after
    //Should an additional enemy be added?
    if(hero.PosicaoX/20>adicionaDistanciaInimigo)
    {
        InimigoPatrulha inimigo = SpawnInimigos();
        listaInimigos.Add(inimigo);
        adicionaDistanciaInimigo += 50;
    }

    // destroy and respawn, update and collide with bubbles
    for(int i=listaInimigos.Count-1;i>=0;i--)
    {
        if(listaInimigos[i].DestroiFlag)
        {
            listaInimigos.Remove(listaInimigos[i]);
            listaInimigos.Add(SpawnInimigos());
            continue;
        }
```

```
        //if(listaInimigos[i].UpdatePatrulha(hero, out posicaoToque))
        //{
        //    efeitoColisao.AdiconaAt(CriaParticulaVermelha, posicaoToque);
        //    count++;
        //}

        //List<Shooter> todosDiparos = hero.TodosDisparos();
        //int numDisparos = todosDiparos.Count;
        //for(int j=numDisparos-1;j>=0;j--)
        //{
        //    if(todosDiparos[j].ToquePixeis(listaInimigos[i], out posicaoToque))
        //    {
        //        listaInimigos[i].DefineParaEstadoStunned();
        //        todosDiparos.RemoveAt(j);
        //        //efeitoColisao.AddEmissor(criaParticulavermelha, posicaoToque);
        //    }

        //}
    }
    //efeitoColisao.UpdateParticulas();
    RespawnInimigos();
    return count;
}
```

```
public void DrawSet()
{
    foreach (var inimiigo in listaInimigos)
        inimiigo.Draw();
   // efeitoColissao.DrawSistemaParticulas();
}
```

Quanto ao áudio foi criada a classe audioSupport que usa um dicionário para associar os sons a um nome, e uma instância da framework áudio para a música. Quanto a métodos o Playsom usa o

método EncontraClipSom, que apenas verifica se este se encontra no dicionário e caso não esteja vai à pasta content para o adicionar ao dicionário, então depois desta verificação o método PlaySom usa a propriedade play da framework para tocar o som. O StartBackground encontra a música coloca em loop e usa a propriedade play e o background pára a música que estiver a dar e toca uma nova, por último a stop simplesmente para a música e coloca-a a null.

```
static public void PlaySom(String nomeSom)
{
    SoundEffect som = EncontraClipSom(nomeSom);
    if (som != null)
        som.Play();
}

private static SoundEffect EncontraClipSom(String nomeSom)
{
    SoundEffect som = null;
    if (efeitosAudio.ContainsKey(nomeSom))
        som = efeitosAudio[nomeSom];
    else
    {
        som = Game1.content.Load<SoundEffect>(nomeSom);
        if (som != null)
            efeitosAudio.Add(nomeSom, som);
    }
    return som;
}
```

```
private static void StartBackGround(String musica, float nivle)
{
    SoundEffect backGround = EncontraClipSom(musica);
    backgoundSound = backGround.CreateInstance();
    backgoundSound.IsLooped = true;
    backgoundSound.Volume = nivle;
    backgoundSound.Play();
}
```

A classe herói possui todas as características do herói enum para os seus estados (andar, stealth, morto, stunned), uma instância deste para reservar o estado atual, um construtor para a classe que define a lista de balas, a sprite e o estado inicial para andar, um método update para usar a animação e usar a mecânica de “shooting”, outro para o estado de stunned e um para quando não pode ser stunned, o método draw para os disparos que é um override da spritetexturas, um método que ajusta a vida, outro para colocar em estado stealth, um para aumentar a vida na apanha de uma ração militar e um última para o estado de morto.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
public Heroi(Vector2 posicao) : base("St", posicao, new Vector2(kLarguraHeroi, kLarguraHeroi), 4, 2, 0)
{
    //vidaHeroiActual = 1;
    tempostunned = 0;
    estadoActual = EstadoHeroi.Andar;
    disparos = new List<Shooting>();

    DefineAnimacao(0, 0, 1, 3, 5);
    SpriteLinhaActual = 0;
}
```

```
public void Update(GameTime gametime, Vector2 delta, bool houveDisparo)
{
    switch(estadoActual)
    {
        case EstadoHeroi.Andar:
            UpdateEstadoHeroi(gametime, delta, houveDisparo);
            break;
    }
}
public void UpdateEstadoHeroi(GameTime gametime, Vector2 delta, bool houveDisparo)
{
    base.Update();
    boundToCamera();
    //Controlo jogador
    Posicao += delta;

    //direcao do sprite
    if (delta.X > 0)
        SpriteLinhaActual = 1;
    else if(delta.X<0)
        SpriteLinhaActual = 0;

    //Direcao do disparo
    int direcaoDisparo = 1;
    if (SpriteLinhaActual == 0)
        direcaoDisparo = -1;
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
    //efeitoColisao.UpdateParticulas();

    float deltaTime = gametime.ElapsedGameTime.Milliseconds;
    tempoUltimoDisparo += deltaTime / 1000;

    //verifica se pode fazer disparo
    if (tempoUltimoDisparo >= ktempoEntreDisparos)
    {
        if (houveDisparo)
        {
            Shooting j = new Shooting(new Vector2(Posicao.X + kDisparoFora * direcaoDisparo, Posicao.Y), direcaoDisparo);
            disparos.Add(j);
            tempoUltimoDisparo = 0;
            //SuporteAudio.EmiteSom("bala");
        }
    }
    //update de todos os disparos
    int count = disparos.Count;
    for (int i = count - 1; i >= 0; i--)
    {
        if (!disparos[i].BalaEstaNoEcra())
        {
            disparos.RemoveAt(i);
        }
        else
            disparos[i].Update();
    }
}
```

```
public void UpdateEstadoStunned(GameTime gametime)
{
    float deltaTime = gametime.ElapsedGameTime.Milliseconds;
    tempostunned += deltaTime / 1000;
    if(tempostunned>=tempoStunnrd)
    {
        tempostunned = 0;
        estadoActual = EstadoHeroi.Morto;//Unnstunable
    }
}
public void UpdateUnnstunable(GameTime gametime)
{
    float deltaTime = gametime.ElapsedGameTime.Milliseconds;
    tempostunned += deltaTime / 1000;
    if(tempostunned>=tempoStunnrd)
    {
        tempostunned = 0;
        estadoActual = EstadoHeroi.Andar;
    }
}
public override void Draw()
{
    base.Draw();
    foreach (var j in disparos)
        j.Draw();
   // efeitoColisao.DrawSistemaParticulas();
}
```

INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Licenciatura em Engenharia em desenvolvimento de jogos de digitais
2016/2017

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
public void AjustaVida(int ajuste)
{
    if (ajuste + VidaHeroi > kVidaHeroi)
        return;
    VidaHeroi += ajuste;
    MathHelper.Clamp(VidaHeroi, 0, 3);
        if(vidaHeroiActual<=0)
    {
        estadoActual = EstadoHeroi.Morto;
    }
}
public void StealthHeroi()//Era o Unstunnable
{
    if(estadoActual!=EstadoHeroi.Morto && estadoActual!=EstadoHeroi.Stunned)
    {
        estadoActual = EstadoHeroi.Stunned;
        AudioSupport.PlaySom("Stun");
        AjustaVida(-1);
    }
}
```

```
public bool Morto()
{
    if (estadoActual == EstadoHeroi.Morto)
        return true;
    else
        return false;
}

public void AumentoVida()
    {
    AjustaVida(2);
    AudioSupport.PlaySom("Ration");
}
```

E por último temos a classe Estado de Jogo que basicamente funciona como a central do programa, pois é esta a classe que chamada todos os métodos das outras classes.

É aqui que é definido os ecrãs de jogo, ou seja, quando no ecrã está representado o menu, ou o jogo em si, ou a tela de "Game Over". Inicialmente temos um construtor que define o ecrã de jogo atual para o menu e inicia a música de fundo, logo a seguir temos métodos que inicializam o Menu, o Jogo e a tela de "GameOver" e depois temos um método que define qual método inicializar através de um switch case que recebe o ecrã de jogo atual, outro método para ir atualizando o jogo, saber se houve colisões, o jogador disparou a arma ou se morreu e por último o método de dar draw.

```
public EstadodeJogo()
{
    EcraAtual = Ecras.Menu;
    //AudioSupport.Background("menusound", 0.5f);
    InicializarMenu();

}
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
//Inicializar Menu
public void InicializarMenu()
{
    //Calcula o centro do ecrã
    float centroX = (Camera.posicaoCantoSuperiorDir.X - Camera.Largura) / 2;
    float centroY = (Camera.posicaoCantoSuperiorDir.Y - Camera.Altura) / 2;

    //Definições do ecrã
    //EcraMenu = new Texturas("menu", new Vector2(centroX, centroY), new Vector2(Camera.Largura, Camera.Altura), null);
    EcraMenu = new Texturas("menu", new Vector2(50, 30), new Vector2(100, 80), null);
    //Mensagem a aparecer
    String msg = "";
    EcraMenu.Texto = msg;
    EcraMenu.CorTexto = Color.White;
}
```

```
public void InicializarJogar()
{
    //Cenário
    background = new Texturas("ColisoesF", new Vector2(50, 30), new Vector2(490, 490));
    chao = new Texturas("ChaoF", new Vector2(50, 30), new Vector2(490, 490));
    condutas = new Texturas("Condutas", new Vector2(50, 30), new Vector2(490, 490));
    redes = new Texturas("RedesAndExtras", new Vector2(50, 30), new Vector2(490, 490));
    teto = new Texturas("Teto", new Vector2(50, 30), new Vector2(490, 490));

    //Inimigos
    inimigos = new Texturas[numEnimigos];
    inimigos[0] = new Texturas("e", new Vector2(10, -20), new Vector2(20, 20));
    inimigos[1] = new Texturas("e", new Vector2(10, 20), new Vector2(10, 10));

    tocaPlayer = new Texturas("a", new Vector2(0, 0), new Vector2(20, 20));
    colisaoPixel = false;
    colisaoJanela = false;

    player = new TexturasSprite("St", new Vector2(7, -185), new Vector2(3, 3), 4, 2, 0);

    player.DefineAnimacao(0, 0, 1, 3, 5);
}
```

```
//Incializar GameOver
public void InicializarEcraGameOver()
{
    //calcular centro do ecrã
    float centroX = Camera.posicaoCantoSuperiorDir.X - Camera.Largura / 2;
    float centroY = Camera.posicaoCantoSuperiorDir.Y - Camera.Altura / 2;

    EcraGameOver = new Texturas("GameOver", new Vector2(50, 30), new Vector2(100, 80), null);
    String msg = "";
    EcraGameOver.Texto = msg;
    EcraGameOver.CorTexto = Color.Red;
}
```

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Licenciatura em Engenharia em desenvolvimento de jogos de digitais
2016/2017

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
public void UpdateJogo(GameTime gameTime)
{

    switch (EcraAtual)
    {
        case Ecras.Menu:
            UpdateMenu();
            break;
        case Ecras.Jogar:
            UpdateGamePlay(gameTime);
            break;
        case Ecras.GameOver:
            UpdateEcraGameOver();
            break;
    }
}
```

```
public void UpdateGamePlay(GameTime gameTime)
{

    //Utiliza o input
    Vector2 playerMoveDelta = InputWrapper.ThumbSticks.Left;
    player.Posicao += playerMoveDelta;
    player.Update();

    UpdateColisoes();

    //Bounce back
    if (colisaoPixel)
        player.Posicao -= playerMoveDelta;
    PlayerMoveCamera();
    UserControlUpdate();

    //if (heroiT.Morto())
    //{
    //    EcraAtual = Ecras.GameOver;
    //    //AudioSupport.PlayACue("Break");
    //    InicializarEcraGameOver();
    //    return;
    //}

    bool hdisparar = (InputWrapper.Buttons.Y == ButtonState.Pressed);
    //h.Update(gameTime, playerMoveDelta, hdisparar);

}
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

```
public void DrawJogo()
{
    switch(EcraAtual)
    {
        case Ecras.Menu:

            if (EcraMenu != null)
                EcraMenu.Draw();
            break;
        case Ecras.Jogar:
            chao.Draw();
            redes.Draw();
            background.Draw();
            condutas.Draw();
            foreach (var p in inimigos)
                p.Draw();
            player.Draw();
            teto.Draw();
            TextoCust.PrintTexto("", null);

            break;
        case Ecras.GameOver:
            if (EcraGameOver != null)
                EcraGameOver.Draw();
            break;
    }
}
```

INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Licenciatura em Engenharia em desenvolvimento de jogos de digitais
2016/2017

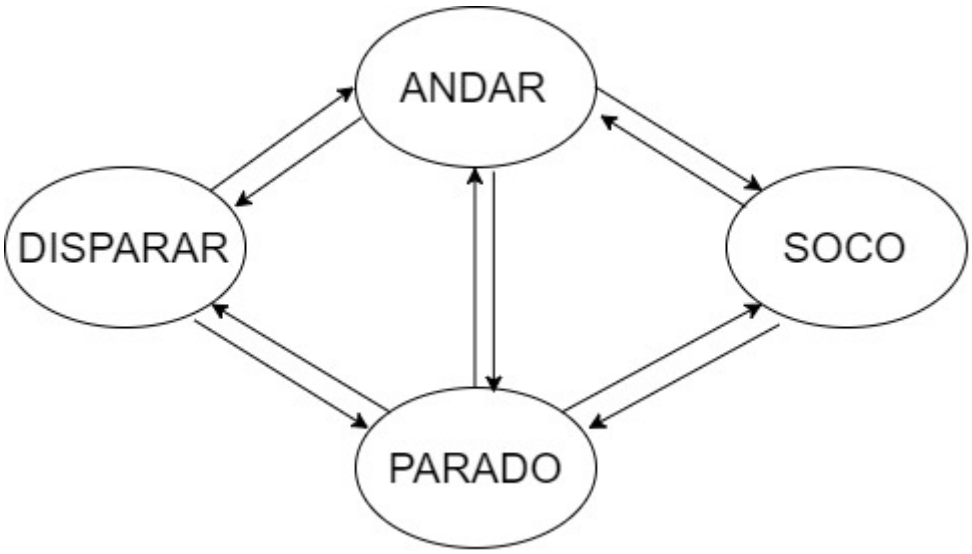
ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

Máquinas de Estado e Fluxogramas

Máquina de estado do personagem:



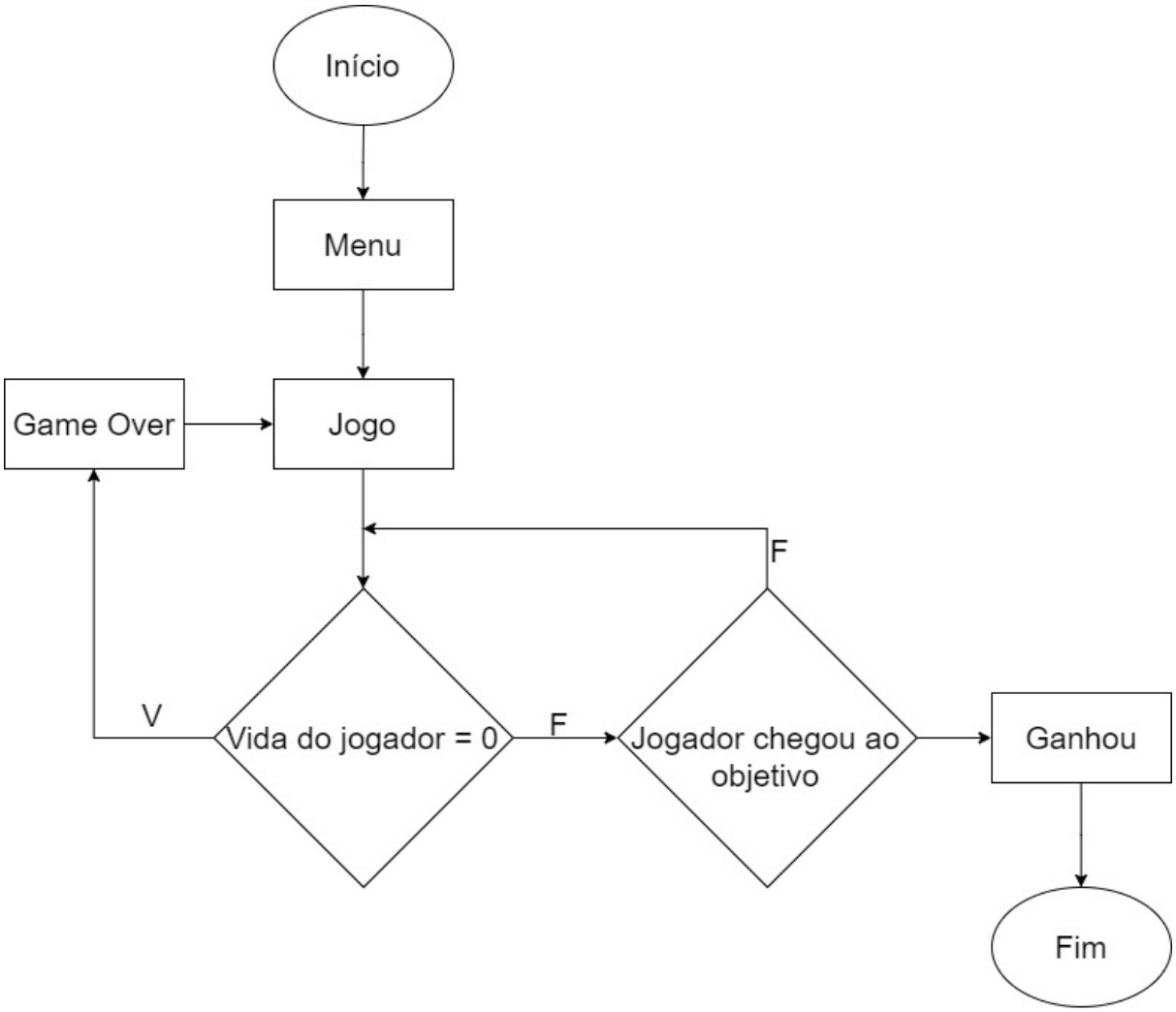
IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
Licenciatura em Engenharia em desenvolvimento de jogos de digitais
2016/2017

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

Fluxograma do programa no geral:



ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

André Rodrigues Nº13231
Fernando Camilo Nº13233

Conclusão

A jeito de conclusão e de reflexão final sobre o projeto, o grupo ficou aquém do que foi proposto sendo que ao usarmos arte não original tivemos muita dificuldade em usar as ferramentas de animação, nunca tendo conseguido uma animação coesa em todas as partes do jogo, ficando também por implementar mecânicas como a apanha de itens, inimigos mais diversificados como os cães de patrulha e até uma patrulha mais eficiente. Caso tenhamos essa oportunidade com algum tempo será possível implementar essas características e tornar o jogo mais próximo do que tínhamos planeado inicialmente. Estas dificuldades advertiram-se também da passagem de um jogo "top-down" para um com um ângulo de 45º graus para dar mais vivacidade aos cenários, apesar de todas as dificuldades a parte sonora ficou fantástica e com qualidade para um produto de retalho.

Concluindo então o grupo apercebe-se que tentou fazer um trabalho demasiado complexo para apenas duas pessoas que se encontram no primeiro ano do curso e onde as dificuldades sentidas deveram-se à falta de entrosamento do grupo e também de traquejo por parte dos elementos do grupo e por isto pedimos que se for possível efetuar uma melhoria do trabalho que nos seja concedida essa oportunidade.