**Projeto Prático Integrador**

**Processamento Paralelo para Jogos Digitais**

1. **Identificação dos Alunos**

Fatec Carapicuíba;

Jogos Digitais – Manhã;

Caíque Anastácio - RA: 1430961611006;

Felipe Goes - RA: 1430961611012;

Rodrigo Zeferino - RA: 1430961611015.

1. **Função de cada Integrante**

Caíque Anastácio – Artista Conceitual, Designer Gráfico, Produtor, Designer de fases;

Felipe Goes – Roteirista, Programador, Designer de áudio;

Rodrigo Zeferino – Programador, Testador de jogos.

1. **Nome do Jogo**

B.I.A.L. – Basic Inception Adventure Life.

1. **Plataforma**

PC – Windows.

1. **Gráficos**

2D – *Top Down View* (Câmera acima do jogador)

1. **Estilo**

Action RPG.

1. **Número de Jogadores**

1.

1. **Suporte a LAN**

Não.

1. **Linguagem de Programação**

Engine - Unity 5.

Linguagem – C#.

1. **Descrição geral do jogo | Mecânica**

O jogo possui característica do gênero RPG (*Role-playing game*), no que o jogador controla o personagem numa visão 2D “*top down view*” (perspectiva vista de cima). Utilizam-se de comandos do teclado, as SETAS para movimentação (cima, esquerda, baixo, direita), SHIFT ESQUERDO para correr, BARRA DE ESPAÇO para confirmação e ESC para cancelamento.

A luta é baseada no sistema de *Space Invaders*. O inimigo no topo da tela disparando projéteis contra o jogador localizado na parte inferior, ao mesmo tempo em que o jogador dispara projéteis contra o inimigo. Ao derrota-lo, o jogador é levado para a tela de vitória que ao pressionar ESC o jogador é levado de volta para o início.

Game Over – Quando o jogador perde ele é levado à tela de fim de jogo, ao pressionar ESC ele volta para o início do jogo.

1. **Objetivo do jogo**

Entreter com ambiente virtual similar ao da FATEC Carapicuíba, utilizando os conhecimentos adquiridos através das aulas.

1. **Recursos do jogo**

Há dois tipos de controle, dentro e fora da luta.

**Dentro da luta:**

O jogador se movimenta para os lados esquerdo e direito utilizando as SETAS do teclado e disparar com a BARRA DE ESPAÇO.

**Fora da luta:**

O jogador pode se movimentar para cima, baixo, direita e esquerda (utilizando as SETAS do teclado para a respectiva direção), não podendo se movimentar para as diagonais. Ao manter pressionado SHIFT ESQUERDO o jogador aumenta sua velocidade e começa a correr. Com a BARRA DE ESPAÇO o jogador pode concluir diálogos e pular a cena de introdução.

1. **Processamento Paralelo**

Por conta de ter utilizado a Engine Unity 5, todos métodos relacionados à Unity 5 API não são *Thread-Safe* (só podem ser executados na *main thread)*, limitando o uso de threads para processos de cálculo e lógica.

* 1. **Disparo de projeteis (Jogador e Inimigo):**

Para o jogador e o inimigo poderem atirar numa frequência desejada é necessário que ele tenha um tempo de *cooldown* (um tempo sem poder atirar) e para isso foi feito um método para que apenas quando a variável “cooldown\_Tiro” ser menor ou igual à zero seja possível disparar, após isso a variável volta a ter o valor inicial, 0.5 para o jogador e 0.3 para o inimigo (número equivalente a segundos). E para que essa variável continue descendo para 0, uma thread separada é responsável por ficar reduzindo-a em relação ao tempo passado (em segundos) desde que o ultimo frame tenha sido concluído.

(Sendo possível ver nos códigos PlayerBattleMov.cs e NPCBattleMov.cs).

* 1. **Cálculo de dano (Jogador e Inimigo):**

Ao receber um disparo, tanto o jogador quanto o inimigo têm em seu código uma thread exclusiva para calcular o dano e reduzir sua vida.

(Sendo possível ver no código CausarDano.cs).

* 1. **Invulnerabilidade (Jogador e Inimigo):**

Ao tomar dano o jogador possuí 1,5 segundos de invulnerabilidade (não podendo ser atingido novamente por disparos) e o inimigo com 1 segundo, ao longo da duração as animações dos personagens devem “piscar” para mostrar que estão invulneráveis. Para isso acontecer há um sistema de *cooldown* semelhante aos disparos, porém além de descer a variável “tempoinvul” (feita para calcular o tempo em que o personagem fica invulnerável em segundos) em uma thread própria nela também é realizado a verificação da variável “check” a cada 0.1 segundos (feita para que quando seja “*true*” o personagem fique invisível e quando seja “*false”* o personagem volte a ser visivel, dando a ilusão de que eles estão piscando).

(Sendo possível ver no código CausarDano.cs).

* 1. **Diálogos (NPCs):**

Para se concluir um diálogo é necessário checar se o diálogo chegou ao seu fim quando o jogador se distanciar do *NPC* (Personagens não jogáveis, do inglês *Non-playable Characters*). Para isso, há threads para verificar se a fala chegou ao fim, mudando a variável “emDialogo” para “*false”* e resetando “dialogoConcluido” para seu estado original “*false*”.

(Sendo possível ver nos códigos Dialogo.cs e DialogoBomDiaMestre.cs).

* 1. ***Fader* (Transição entre partes da fase):**

Durante a transição entre as partes da fase, há uma animação da tela escurecendo até ficar totalmente preta e após o personagem transitar a tela volta a clarear para o estado normal. Para checar o estado da animação se está completa ou não, uma thread é responsável por mudar o estado da variável “Fading” de “*true”* para “*false”*.

(Sendo possível ver no código Fader.cs).

* 1. **Projeção de *Gizmos* (Utilizado para verificar os *raycastings*):**

Para os *NPCs* detectarem de onde o jogador está vindo eles projetam raycastings para as quatro direções (cima, baixo, esquerda e direita) e para poder vê-los dentro do projeto é necessário “desenhar” as projeções, para isso há 2 threads separadas, uma para desenhá-las e mantê-las renderizadas em todos os frames e outra para desenhar os raycastings somente quando o objeto os contendo é selecionado.

(Sendo possível ver no código ShowGizmo.cs).