LUIS FERNANDO ALVES DA SILVA
PEDRO RAMIRES DA SILVA AMALFI COSTA
REGINADO DONIZETI CÂNDIDO

**IMPLEMENTANDO COINCEITOS ORIENTADOS A
OBJETOS UTILIZANDO A FERRAMENTA DE
CRIAÇÃO DE JOGOS UNITY 2D**

Primeira Edição

MOGI GUAÇU - SP

2019

197pg

Silva, Luis Fernando Alves da Silva.
IMPLEMENTANDO COINCEITOS ORIENTADOS A
OBJETOS UTILIZANDO A FERRAMENTA DE CRIAÇÃO
DE JOGOS UNITY 2D Luis Fernando Alves da Silva,
Pedro Ramires da Silva Amalfi Costa, Reginaldo Donizeti
Cândido. -[1. ed.] - Mogi Guaçu, 2019.
28 cm.
Inclui índice
ISBN 978-85-7138-009-7

1. CRIAÇÃO DO PROJETO UNITY.
2. TRABALHANDO COM CENÁRIOS.
3. TRABALHANDO COM ARMAS NO PLAYER.
4. KNOCKBACK.
5. CAMADA HUD.
6. SISTEMA DE LOOT.
7. SISTEMA DE ALERTA DE INTERAÇÃO.
8. CORRIGINDO ALGUMAS FUNCIONALIDADES
ATUAIS.
9. MENU DE SELEÇÃO DE PERSONAGENS.

CPSCETEC / SP

CDD – 005.13 CDU – 004.43

RESUMO

Este livro tem como objetivo o desenvolvimento de um jogo de RPG Medieval com o intuito de demonstrar elementos da programação orientada a objetos de forma prática e funcional. O jogo de RPG será desenvolvido utilizando a ferramenta de desenvolvimento Unity por ser de fácil aprendizado, possuir diversos materiais e tutoriais gratuitos *on-line* e ser uma plataforma gratuita, não exigindo nenhum custo para o desenvolvimento do projeto. Para a linguagem será utilizado o C# junto com o *framework* Visual Studio.

PALAVRAS-CHAVE: Unity, Programação Orientada a Objetos, C#, Visual Studio.

LISTA DE FIGURAS

Figura 1- Nome das Sprites Iniciais.....	13
Figura 2 - Animator Player	14
Figura 3 - Lista de Animações.....	14
Figura 4 - Alguns personagens e Armas usados no jogo	15
Figura 5 - Regras de Animação.....	16
Figura 6 - Inicio do Script Player	21
Figura 7 - Speedy em JumpFall	27
Figura 8 - Código Void Update	30
Figura 9 - Condicional Abaixado	30
Figura 10 - Eventos em uma animação.....	32
Figura 11 - Personagem abaixado	33
Figura 12 - Adicionando multiplos BoxColliders no Player.....	34
Figura 13 - Mudando BoxCollider via Script.....	35
Figura 14 - Colisores.....	37
Figura 15 -Criando Sorting Layers	38
Figura 16 - Desenhando Cenario Basico	38
Figura 17 - Adicionando Itens ao Cenario	39
Figura 18 - Criando lista de Layers	40
Figura 19 - Multiplos BackGrounds	40
Figura 20 - Juntando os BackGrounds.....	40
Figura 21 - Alterando o ALFA.....	41
Figura 22 - Desafio Cenario	41
Figura 23 - Circle Colider	42
Figura 24 - Material Sem Atrito.....	43
Figura 25 - Objeto Hand.....	45
Figura 26 - LayerMask	46
Figura 27 - Camada Interação.....	46
Figura 28 - Amarrando o Script Interação	47
Figura 29 - Script Bau	49
Figura 30 - Objeto Bau	49
Figura 31 - Armas A, B e C	51
Figura 32 - Personagem com armas.....	52
Figura 33 - Arma posição C.....	52
Figura 34 - Amarrando o Script com o Objeto Armas.....	53
Figura 35 - Evento Arma	55
Figura 36 - TAG ARMA	55
Figura 37 - Circle Colider no machado	56
Figura 38 - Controle Dano Inimigo	57
Figura 39 - Mensagem de dano recebido.....	58
Figura 40 - Script mensagem de Dano.....	58
Figura 41 - knockback	59
Figura 42 - Layer Inimigo.....	60
Figura 43 - Força da explosão.....	60
Figura 44 - Prefab Knockback	61
Figura 45 - knock X	62
Figura 46 - Inimigo	66

Figura 47 - Obstaculo	68
Figura 48 - Retorno	69
Figura 49 - tempo para o retorno.....	70
Figura 50 - piscando após o HIT	72
Figura 51 - camada HUD.....	74
Figura 52 - barra de vida	74
Figura 53 - Controle dano Inimigo	75
Figura 54 - Freeze Rotation Z	77
Figura 55 - Dano TXT Prefab	78
Figura 56 - animação inimigo	79
Figura 57 - Time Line	80
Figura 58 - hit normal	81
Figura 59 - objeto Hit Normal	81
Figura 60 - Animação Hit.....	82
Figura 61 - prefab hit Normal.....	83
Figura 62 - Script Dano Inimigo.....	83
Figura 63 - Animação de Morte	84
Figura 64 - Ground Check.....	85
Figura 65 - icone coin.....	87
Figura 66 - Config Coin	87
Figura 67 - Animation Coin.....	88
Figura 68 - Matrix Camadas de Colisão	90
Figura 69 - Coin ao Loot do Baú	91
Figura 70 - Matrix de Colisão Loot	93
Figura 71 - Sorting Layer Fundo.....	93
Figura 72 - Valor do Coin	96
Figura 73 - Re Skin	99
Figura 74 - canvas.....	100
Figura 75 - Scene.....	101
Figura 76 - Order in Layer Canvas	101
Figura 77 - Script Canvas.....	101
Figura 78 - HUD Canvas	102
Figura 79 - Position	102
Figura 80 - Rect Transform	103
Figura 81 - Coração Logo	103
Figura 82 - HP BAR.....	104
Figura 83 - bar1	104
Figura 84 - Logo barra vida	105
Figura 85 - Barra de vida cheia	105
Figura 86 - Bar5	105
Figura 87 - Gold Bar.....	106
Figura 88 - Logo Barra Gold.....	107
Figura 89 - Contador Barra Gold	107
Figura 90 - Script Gold Bar.....	108
Figura 91 - Elements HP BAR.....	109
Figura 92 - Iamgens Alerta	113
Figura 93 - Balão Player.....	114
Figura 94 - Timeline AlertBallon	115
Figura 95 - Objeto Balão Alerta	115
Figura 96 - Script AlertBallon.....	116

Figura 97 - Sprite Renderer.....	117
Figura 98 - Portas na Fase.....	118
Figura 99 - Script Porta	119
Figura 100 - Objetos Portas	119
Figura 101 - Hud Transição.....	120
Figura 102 - Canvas.....	120
Figura 103 - Canvas Script.....	121
Figura 104 - Objeto Fume	121
Figura 105 - Rect Transform	121
Figura 106 - Script Raycaster.....	122
Figura 107 - Fade.....	122
Figura 108 - Iluminação cenário.....	125
Figura 109 - Fundo Caverna	125
Figura 110 - chão caverna	126
Figura 111 - 2D Padrão.....	126
Figura 112 - Luz 2D.....	126
Figura 113 - Material Luz2D	126
Figura 114 - Caverna e Tochas.....	127
Figura 115 - Luz na Tocha	127
Figura 116 - Configurações de luminosidade.....	128
Figura 117 - Ambiente Color	128
Figura 118 - tochas com luz	129
Figura 119 - cenário com tochas	129
Figura 120 - Adicionando o Script as tochas.....	130
Figura 121 - Time Line animação Tocha	132
Figura 122 - Point Light	132
Figura 123 - logo Cinemachine	135
Figura 124 - aba cinemachine	135
Figura 125 - objeto cinemachine	136
Figura 126 - propriedades do Cinemachine	136
Figura 127 - Z Damping	136
Figura 128 - Área da Câmera.....	137
Figura 129 - Dead Zone	137
Figura 130 - Cenário com dead Zone.....	138
Figura 131 - Polygon Collider 2D	138
Figura 132 - Cenário com Cinemachine	139
Figura 133 - Is Trigger no Polygon Collider	139
Figura 134 - Script Cinemachine	140
Figura 135 - Cenário com Cinemachine	140
Figura 136 - Vetor com o tamanho das armas	141
Figura 137 - Vetor com as imagens das armas.....	142
Figura 138 - Vetor com o dano da arma.....	144
Figura 139 - vetor com descrição das armas	146
Figura 140 - Portal.....	147
Figura 141 - Cena 2 Script	147
Figura 142 - Erro	148
Figura 143 - Build Settings	148
Figura 144 - Nome das Skins	151
Figura 145 - ID ARMA	152
Figura 146 - Re Skin	153

Figura 147 - Skin por Objeto	154
Figura 148 - armas a distância	155
Figura 149 - Arcos	156
Figura 150 - Pasta Armas	156
Figura 151 - Elfa com Arco	157
Figura 152 - Objeto Flecha	157
Figura 153 - Flecha Grande no Arco	157
Figura 154 - Flecha pequena com Arco	158
Figura 155 - Alinhando com a mão	158
Figura 156 - Spawn Flecha	159
Figura 157 - Flecha Prefab	159
Figura 158 Mago	160
Figura 159 - Pasta Animations	160
Figura 160 - Array com as armas	161
Figura 161 - Spawn Armas Prefab	162
Figura 162 - Código Vetores Armas	163
Figura 163 - GameController	164
Figura 164 - Vetores com valores	164
Figura 165 - Vetores com Valores	165
Figura 166 - Vetores com danos da arma	165
Figura 167 - Player Script	166
Figura 168 - Case arma	166
Figura 169 - Animação Cajado	168
Figura 170 - Regras de animação	168
Figura 171 - Speed Y	169
Figura 172 - Regras para as 3 armas	169
Figura 173 - Blend Tree	170
Figura 174 - Timeline Attack Staff	170
Figura 175 - Atacando com Arco	172
Figura 176 - Atacando com Cajado	172
Figura 177 - Função Attack Flecha	174
Figura 178 - Função Attack Staff	175
Figura 179 - Script Destroy Collisor	177
Figura 180 - Id class e arma inicial	178
Figura 181 - Menu de seleção	180
Figura 182 - Canvas Scaler	181
Figura 183 - Barbaro	181
Figura 184 - 3 Personagens	181
Figura 185 - Button	182
Figura 186 - On Click	182
Figura 187 - Condicional on Click	183
Figura 188 - Build Settings Titulo	183
Figura 189 - InfoGamePlay	185
Figura 190 - Image Script	185
Figura 191 - Rect Tranform	186
Figura 192 - Hud_0	186
Figura 193 - Posição do Menu	186
Figura 194 - Painel de Pause	187
Figura 195 - Objeto PauseGame	187
Figura 196 - Script Game Controller	189

Figura 197 - Hud_18	190
Figura 198 - textMash	190
Figura 199 - Menu Completo.....	191
Figura 200 - Sub Menu.....	192
Figura 201 - Itens	192
Figura 202 - Objeto Itens.....	193
Figura 203 - PauseGame	194
Figura 204 - Event On Click	194
Figura 205 - Image Script.....	195
Figura 206 - Rect Transform	195
Figura 207 - Botão X	195
Figura 208 - Event Click	196
Figura 209 - Event Click	197
Figura 210 - Cor de Fundo Botão.....	197

LISTA DE ABREVIATURAS

LMMS (Linux *Multimedia Studio* – Estudo de multimídia Linux)

UDK - Unreal *Development Kit* – Kit de desenvolvimento da Unreal

XNA - XNA's Not Acronymed – XNA não é Acronymed

SUMÁRIO

INTRODUÇÃO AO DESENVOLVIMENTO DE JOGOS.....	12
1 CRIAÇÃO DO PROJETO UNITY	13
1.1 DESENVOLVENDO AS REGRAS DE ANIMAÇÃO.....	16
1.2 CRIAÇÃO DE SCRIPTS UNITY	21
1.3 CRIANDO AS ANIMAÇÕES DE PULO DO PLAYER	26
1.4 ADICIONANDO EVENTOS A ANIMAÇÕES CRIADAS.....	31
2 TRABALHANDO COM CENÁRIOS	36
2.1 ADICIONANDO ELEMENTOS DECORATIVOS AO CENARIO	39
2.2 CORRIGINDO PROBLEMAS NO COLISOR.....	42
2.3 CRIANDO SISTEMA DE INTERAÇÃO COM OBJETOS.....	44
3 TRABALHANDO COM ARMAS NO PLAYER	51
3.1 CRIANDO O SCRIPT PARA AS ARMAS	53
3.2 SISTEMA DE COLISÃO DA ARMA.....	55
3.3 SCRIPT DANO DA ARMA	56
4 KNOCKBACK.....	59
4.1 INIMIGO TOMANDO DANO E DIMINUINDO VIDA.....	64
4.2 TROCANDO A TRAGETÓRIA DO INIMIGO	68
4.3 SISTEMA DE RETORNO	69
4.4 SISTEMA DE IMUNIDADE POR TEMPO.....	70
5 CAMADA HUD	74
5.1 ANIMAÇÕES INIMIGO	79
5.2 ANIMAÇÃO DE HIT	81
5.3 ANIMAÇÃO DE MORTE DO INIMIGO	84
6 SISTEMA DE LOOT.....	87
6.1 ANIMATION COIN.....	88
6.2 GERENCIANDO LOOTS DO BAÚ	92
6.3 IMPLEMENTANDO CONTADOR DE COINS.....	94
6.4 SISTEMA DE MUDANÇA DE PERSONAGEM	98
6.5 TRABALHANDO COM CANVAS.....	100
6.6 HP BAR	102
6.7 GOLD BAR	106
7 SISTEMA DE ALERTA DE INTERAÇÃO	113

7.1 CORREÇÃO DE LOOT	116
7.2 TRANSIÇÃO DE PERSONAGEM	117
7.3 SCRIPT PORTAS.....	118
7.4 SISTEMA DE TRANSIÇÃO	120
7.5 – ANIMAÇÃO PORTA	124
7.6 LUZ NAS TOCHAS.....	127
7.7 ANIMAÇÃO DO FOGO.....	131
7.8 ADICIONANDO A LUMINOSIDADE AS ARMAS	132
8 CORRIGINDO ALGUMAS FUNCIONALIDADES ATUAIS.....	135
8.1 SISTEMA DE CÂMERA CINEMACHINE	135
8.2 IMPLEMENTANDO NOVAS ARMAS	141
8.3 TRABALHANDO COM MÚLTIPLAS CENAS	146
8.4 IMPLEMENTANDO MELHORIAS NA TROCA DE SKIN	153
8.5 IMPLEMENTANDO ARMAS A DISTÂNCIA	155
8.6 ANIMAÇÃO DAS ARMAS ARCO E STAFF	168
8.7 CRIANDO OS PROJETEIS DAS ARMAS	172
8.8 DIREÇÃO DA FLECHA	175
8.9 CLASSE DE CADA PERSONAGEM	177
9 MENU DE SELEÇÃO DE PERSONAGENS	180
9.1 MUDANDO DE ARMA IN GAME	184
9.2 CRIANDO MENUS E SUBMENUS.....	185

INTRODUÇÃO AO DESENVOLVIMENTO DE JOGOS

Quando se começa a trilhar o caminho da programação, aprende-se inicialmente a lógica de programação, que nada mais é do que formas de abstrair conceitos aplicados no mundo real e implementar a computação e programação, com a dominância sobre o assunto, é passado para uma linguagem de programação, aprende-se elementos básicos como criação e declaração de variáveis, envio de textos e informações ao usuário e recebimento dos valores digitados pelo menos, utilização de estruturas condicionais e de repetição e vai implementando novos conceitos na programação mudando o grau de conhecimento para intermediário e até avançado com criação de classe, métodos, programação orientada a objetos e até conexão com banco de dados, entre outros elementos mais avançados.

Sempre utilizando a mesma ferramenta, não necessitando (na maioria dos casos, se não considerar o banco de dados como elemento externo) o uso de ferramentas externas ou mesmo de múltiplas ferramentas para conseguir a criação de um aplicativo final. Porém quando se pensa em programação para o desenvolvimento de jogos, tal paradigma cai pelo chão, uma vez que não é possível desenvolver, utilizando apenas uma ferramenta, um jogo do início ao final. É necessárias diversas ferramentas para conseguir realizar um jogo simples e quanto mais complexo se torna o jogo, maior a quantidade de ferramentas foram necessárias para conseguir obter o resultado final esperado.

Pensando nisso, foi criado esse livro, a fim de ilustrar as finalidades das principais ferramentas que poderão ser utilizadas no desenvolvimento de um jogo enquanto desenvolve um projeto de um jogo de RPG em 2D. Após o aprendizado do projeto, é possível ao leitor desenvolver seu próprio projeto e, caso seja necessário implementar outras funcionalidades não demonstradas neste livro, ele possuirá uma base sólida para continuar seus estudos autonomamente.

1 CRIAÇÃO DO PROJETO UNITY

Crie um projeto Unit 2D.

Na aba Project crie uma pasta chamada Assets e em seguida uma outra chamada Sprites, selecione todos os arquivos que o Anexo 1 possui e, em seguida, arraste para dentro da pasta criada no Unit.

Clique na imagem do personagem, clique nas propriedades Sprite Mode e mude para Multiple, Apply em baixo, em seguida Sprite Editor.

Selecione a aba Slice, troque o Type dele para Grid by Cell Size, digite 240 x e 240 y, Slice e Finalmente Apply, como podem ver estão todos cortados, Add um boneco a tela

Clique em câmera, Projektion tem que estar Ortographic e Size mudem para 1.5 ou próximo disso, clique no personagem e mude o Filter Mode para Point e mostre como ele fica mais nítido assim.

Vamos agora voltar no personagem e de novo Sprite Editor, agora vamos renomear nossos Sprites para:

Figura 1- Nome das Sprites Iniciais

Renomeie o seu personagem para player, adicione o Componente Micelanius/ Animator

Crie dentro da pasta Assents uma pasta chamada Animations e dentro dela crie uma nova pasta chamada Controller

Dentro da pasta Controller, botão direito, Create / Animator Controller e chame ele de Player

Arraste o Player para dentro do personagem em Animator / Controller:

Figura 2 - Animator Player

Vá lá em cima no menu Window escolha Animation, em seguida pressione o botão Create.

Vai perguntar onde quer salvar a animação, escolha a pasta Animations e dê o nome de Idle (parado), Salvar, Monte as animações do seu personagem até todos estiverem prontas:

Figura 3 - Lista de Animações

Para criar uma nova animação é só clicar no Idle e em seguida create new clip.

Walk (andando) / Jump (pulando jump1,2 e 3) / Fall (queda é só o Sprite Jump4)/ Attack (Atacando) / Die (morrendo) / Hit (tomando dano) / Crouch (Agachando)/

Estamos fazendo isso pois vamos automatizar esses scripts para todos os outros personagem e armas que eles vão usar como pode ser visto na figura 04 abaixo.

Figura 4 - Alguns personagens e Armas usados no jogo

1.1 DESENVOLVENDO AS REGRAS DE ANIMAÇÃO

Agora clique novamente em Window só que agora em Animator e vamos criar as regras para as animações:

Figura 5 - Regras de Animação

Inicialmente temos de criar os parâmetros necessários para executar cada uma das nossas animações:

Bool grounded //falar que está pisando no chão sim ou não

Int idAnimation

Trigger attack

Trigger hit

Criando nossas Regras:

- Idle , botão Direito Make Transition, clique no Attack, Regras:

Reset

Desmarca Has Exit Time

E Adiciona a condição Attack

- Attack -> Idle:

Reset

Marcado Has Exit Time

Grounded True

idAnimation = 0

- Idle -> Walk :

Reset

Desmarca

Grounded True

idAnimation = 1

- Walk -> Idle:

Reset

Desmarca

Grounded True

idAnimation = 0

- Walk -> Attack:

Reset

Desmarca

Attack

- Attack -> Walk:

Reset

Deixa Marcado

Grounded True

idAnimation = 1

- Idle -> Crouch

Reset

Desmarca

Grounded True

idAnimation = 2

- Crouch -> Idle

Reset

Desmarca

Grounded True

idAnimation = 0

- Walk -> Crouch

Reset

Desmarca

Grounded True

idAnimation = 2

- Crouch -> Walk

Reset

Desmarca

Grounded True

idAnimation = 1

- Attack -> Crouch (não tem o contrário pois não poderá atacar enquanto estiver abaixado)

Reset

Deixar Marcado

Grounded True

idAnimation = 2

- Crouch -> Hit

Reset

Desmarca

Hit

- Hit -> Crouch

Reset

Desmarca

Grounded True

idAnimation = 2

- Idle -> Hit

Reset

Desmarca

Grounded True

Hit

- Hit -> Idle

Reset

Deixar marcado

idAnimation = 0

- Walk -> Hit

Reset

Desmarca

Hit

- Hit -> Walk

Reset

Deixar Marcado

Grounded True

idAnimation = 1

- Attack -> Hit

Reset

Desmarca

Hit

- Idle -> Die

Reset

Desmarca

idAnimation = 3

- Walk -> Die

Reset

Desmarca

idAnimation = 3

- Attack -> Die

Reset

Desmarca

idAnimation = 3

- Die -> Exit (só adicionar, não precisa mexer em mais nada)

Agora é só testar nossa animação e ver se as regras estão funcionando

1.2 CRIAÇÃO DE SCRIPTS UNITY

Agora crie uma pasta chamada Scripts dentro da pasta chamada Assets

Crie um script C# e dê o nome de playerScript

Arraste este novo componente para dentro do seu player

Abra ele e digite o seguinte código:

Figura 6 - Inicio do Script Player

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class aulaScript : MonoBehaviour {
6
7     private Animator playerAnimator;
8
9     public bool Grounded; //INDICA SE O PERSONAGEM ESTÁ PISANDO EM ALGUMA SUPERFÍCIE
10    public int idAnimation; // INDICA O ID DA ANIMAÇÃO
11
12    // Use this for initialization
13    void Start () {
14
15        playerAnimator = GetComponent<Animator>();
16
17    }
18
19    // Update is called once per frame
20    void Update () {
21
22
23        playerAnimator.SetBool("grounded", Grounded);
24        playerAnimator.SetInteger("idAnimation", idAnimation);
25    }
26
27 }
28

```

Teste a aplicação

Em seguida adicione ao seu personagem um componente em Physical2D / Rigidbody 2D

Em Constraints marque o Freeze Rotation Z

Se der play seu personagem agora vai cair

Adicione o TileSet_03 e jogue na cena (o chão) , renomeie para chao e mude as propriedades:

Draw Mode: Tiled

E arraste seus lados para fazer um chão largo

Adicione ao chão o componente Physical2D /BoxCollider 2D

Edite a área de colisão através do Edit Collider e aumente o quadrado verde até estar do mesmo tamanho da imagem

No personagem adicione outro componente Physical2D / BoxCollider 2D

Edite a área de colisão do tamanho do seu personagem

Agora se der play o personagem fica no chão

Deixe o grounded marcado como true no script do personagem, edite seu script na parte void Update e adicione:

```
float h = Input.GetAxisRaw("Horizontal"); //pega as teclas de frente e traz
<- ou -> e A ou D
```

```
float v = Input.GetAxisRaw("Vertical"); //pega as teclas de baixo como
seta baixo e S
```

```
if(v <0) //caso pressione para baixo
```

```

{
    idAnimation = 2;
}
else if(h != 0) //caso seja pressionado uma seta direcional <- ou ->
{
    idAnimation = 1;
}
else //caso contrario o personagem fica parado
{
    idAnimation = 0;
}

```

Da play e veja se está funcionando. Se ele abaixar, perceba que ele não anda pois criamos essa regra anteriormente;

Para fazer ele atacar vamos adicionar no fim desse código:

if(Input.GetButtonDown("Fire1") && v >= 0) //caso pressione a tecla CTRL ou esquerdo do mouse e não esteja agachado será executado a trigger de attack

```

{
    playerAnimator.SetTrigger("attack");
}

```

Com o objeto player selecionado vamos criar um novo objeto vazio que será o filho desse objeto player com o nome grounCheck ele será responsável

em ver se o personagem está ou não com os pés em alguma plataforma sólida. Adicione um ícone amarelo a ele para que possa ser visível na programação, arraste ele para baixo do pé do personagem, deixando um pouco afastado do pé dele para não colidir com o próprio pé do objeto pai.

No script adicione junto das outras variáveis publicas :

```
public Transform groundCheck; //responsavel por detectar se o
personagem está pisando em uma superficie
```

arraste o objeto filho para dentro das propriedades do script do objeto pai com o nome groundCheck;

adicione mais essas duas variáveis publicas:

```
public float speed; //velocidade de movimento do
personagem
```

```
public float jumpForce; //força aplicada ao gerar o pulo do
personagem
```

coloque como valor speed 2 e jumpForce 200, aproveite e desmarque o grounded como true já que agora quem vai marcar ou desmarcar ele é o próprio objeto filho criado.

Entre o Void Start e o void Update, adicione:

```
void FixedUpdate()
{
    Grounded =
    Physics2D.OverlapCircle(groundCheck.position, 0.02f);
}
```


Vamos fazer o personagem pular então declare mais uma variável:

```
playerRb = GetComponent<Rigidbody2D>();
```

dentro do void Start adicione:

```
playerAnimator = GetComponent<Animator>();
```

Embaixo da condição do attack no void update adicione:

```
if(Input.GetButtonDown("Jump") && Grounded == true) //caso seja
pressionado a tecla de pulo (espaço) o personagem receberá força para sair do
chão e a função de verificar chão tem que ficar falsa
```

```
{
    playerRb.AddForce(new Vector2(0, jumpForce));
}
```

Para fazer o personagem se mover no mapa adicionamos abaixo do código que acabamos de criar as linhas:

```
playerRb.velocity = new Vector2(h*speed, playerRb.velocity.y); //faz o
personagem se mover no mapa
```

Com isso ele vai parecer um Michael Jacson andando pra trás, para arrumar isso vamos fazer ele virar dependendo do lado que ele estiver andando, para isso adicione esta nova variável publica:

```
public bool lookLeft; //indica se o personagem esta
virado para a esquerda
```

crie essa função no fim do seu código, após o fim do void Update:

```
void flip()
```

```

{
    lookLeft = !lookLeft; //inverte o valor da variavel booleana
entre true e false

    float x = transform.localScale.x;

    x *= -1; //inverte o sinal do scale x

    transform.localScale = new Vector3(x,
transform.localScale.y, transform.localScale.z);
}

```

E no começo do seu código dentro do void Update adicione como primeira condição agora:

if(h > 0 && lookLeft == true) //se está andando pra frente e o personagem está virado para tras, vire ele

```

{
    flip();
}

```

else if (h < 0 && lookLeft == false) //se está andando para trás e o personagem está virado para frente, vire ele

```

{
    flip();
}

```

Teste seu personagem

1.3 CRIANDO AS ANIMAÇÕES DE PULO DO PLAYER

Vamos agora fazer a animação do pulo, então abram o Animator (window/ Animator), criem um novo parâmetro float chamada speedY, em seguida vamos criar um novo estado para nossa animação, pressione com o

botão direito do mouse, na área onde ficam nossas animações e selecione Create State -> From new Blend Tree e renomeie ele para JumpFall.

Dê um duplo clique no JumpFall, e renomeie o Blend Tree para novamente JumpFall, em Motion adicione 2 Motion Field e arraste o script dentro da pasta Animations do Fall para o primeiro e o Jump para o segundo.

Adicione os valores de -0.02 e 0.02 conforme imagem abaixo:

Figura 7 - Speedy em JumpFall

No Script do player adicionaremos agora a seguinte linha de comando como a ultima linha de programação dentro do void Update:

```
playerAnimator.SetFloat("speedY", playerRb.velocity.y);
```

Agora precisamos continuar a criar as condições para as transições de pulando para as outras ações que o personagem pode realizar, então vamos criar no Animator:

- JumpFall -> Idle:

Reseta

Desmarca

Grounded True

idAnimation = 0

- JumpFall -> Walk:

Reseta

Desmarca

Grounded True

idAnimation = 1

- JumpFall -> Atack:

Reseta

Desmarca

Atack

- JumpFall -> Hit:

Reseta

Desmarca

Hit

- JumpFall -> Crouch:

Reseta

Desmarca

Grounded True

idAnimation = 2

- Idle ->JumpFall:

Reseta

Desmarca

Grounded false

- Walk ->JumpFall:

Reseta

Desmarca

Grounded false

- Attack ->JumpFall:

Reseta

Deixa Marcado

Grounded false

- Crouch ->JumpFall:

Reseta

Desmarca

Grounded false

Vamos agora arrumar algumas coisas que estão fora do local correto no PLAYERScript nosso, primeiramente o que é relacionado a elementos da física do jogo tem que estar dentro do método void fixedUpdate, então vamos recortar a linha abaixo e colar dentro do método:

```
playerRb.velocity = new Vector2(h*speed, playerRb.velocity.y); //faz o
poersonagem se mover no mapa
```

outra coisa que está no local errado é a declaração das variáveis V e H , então vamos colocar elas junto com as public e privates do começo do código, neste caso vamos transformar as duas em privates:

```
private float h, v;
```

E na onde elas estavam vamos apenas apagar os float do começo da linha

Figura 8 - Código Void Update

```
// Update is called once per frame
void Update () {
    h = Input.GetAxisRaw("Horizontal"); //pega as teclas de frente e traz <- ou -> e A ou D
    v = Input.GetAxisRaw("Vertical"); //pega as teclas de baixo como seta baixo e S
```

Atualmente nosso personagem, quando está abaixado ainda pode se mover para direita e esquerda então para corrigir isso vamos adicionar na condicional de quando apertamos para baixo o valor de 0 (zero) para a variável h, desde que o personagem esteja com o pé no chão:

Figura 9 - Condicional Abaixado

```
if(v < 0) //caso pressione para baixo
{
    idAnimation = 2;
    if(Grounded == true){
        h=0;
    }
}
```

Outra coisa que nosso personagem ainda está fazendo errado é que ele consegue atacar e se mover ao mesmo tempo então agora queremos que quando nosso personagem ataque, que ele fique parado no mesmo local.

Primeiramente vamos criar uma variável pública booleana para avisar sempre que nosso personagem atacar então declaramos lá em cima:

```
public bool attacking;           //indica se o personagem está
atacando
```

agora vamos criar uma função embaixo da void flip que vai fazer esta variável receber verdadeiro ou falso :

```
void attack(int atk)
```

```

{
    if (atk == 0){
        attacking = false;
    }
    else{
        attacking = true;
    }
}

```

E também vamos criar uma condição que faz o personagem ficar parado como fizemos com o abaixado:

if(attacking == true && Grounded == true) //se o personagem estiver atacando no chão não é para ele se mover

```

{
    h = 0;
}

```

1.4 ADICIONANDO EVENTOS A ANIMAÇÕES CRIADAS

Próxima coisa que precisamos fazer é ir na aba Animation (Window / Animation) com o personagem selecionado, vamos até a animação de attack e vamos repetir o 3º frame mais uma vez só copiando e colando ele na posição 0:3, em seguida vamos no frame zero e vamos adicionar um evento

Figura 10 - Eventos em uma animação

Selecione a função `attack` e dê o valor de 1 para ela e depois na 0:3 crie mais um evento e deixe zero mesmo

Outra coisa para garantir erros na animação do nosso personagem é que agora só será possível atacar caso ele não esteja atacando, então vamos mudar a condição e deixar dessa forma

```
if(Input.GetButtonDown("Fire1") && v >= 0 && attacking == false) //caso
pressionar a tecla CTRL ou esquerdo do mouse e não esteja agachado será
executado o trigger de attack
```

```
{
    playerAnimator.SetTrigger("attack");
}
```

Uma coisa que nosso personagem está fazendo de errado é que podemos mudar a direção dele enquanto atacamos, para arrumar isso só precisamos de adicionar na condição de `flip` que o personagem não esteja atacando:

```
if(h > 0 && lookLeft == true && attacking == false) //se está andando pra
frente e o personagem está virado para trás, vire ele
```

```
{
    flip();
}
```



```

    }

    else if (h < 0 && lookLeft == false && attacking == false) //se
está andando para trás e o personagem está virado para frente, vire ele

    {

        flip();

    }

```

E também ele consegue atacar e pular então também vamos arrumar isso:

```

if(Input.GetButtonDown("Fire1")&& v >= 0 && attacking == false &&
Grounded == true) //caso pressione a tecla CTRL ou esquerdo do mouse e não
esteja abaixado será executado a trigger de attack

```

```

{

    playerAnimator.SetTrigger("attack");

}

```

Uma outra coisa que precisamos arrumar é que quando nosso personagem se abaixe a área de colisão mude pois mesmo abaixado a colisão continua como se estivesse em pé.

Figura 11 - Personagem abaixado

Então vamos criar um novo Box Collider 2D dentro de componentes Physical 2D e vamos editar a área para preencher nosso personagem quando o mesmo estiver abaixado.

Para que haja essa mudança de áreas de colisão vamos abrir nosso script do player e vamos declarar duas variáveis:

```
public Collider2D standing, crouching; //colisor quando em pé e abaixado
```

Agora vai aparecer no nosso script no unity os dois campos dessas duas variáveis, então vamos arrastar cada boxCollider para o seu respectivo local.

Figura 12 - Adicionando multiplos BoxColliders no Player

Falta apenas trabalharmos com a condição dentro do nosso script, como sabemos já possuímos uma condição caso o usuário queira se abaixar, então vamos aproveitar ela e adicionar para habilitar um boxcollider quando estiver em pé e outro quando estiver abaixado:

Figura 13 - Mudando BoxColider via Script

```
if(v < 0) //caso pressione para baixo
{
    idAnimation = 2;
    if(Grounded == true){
        h=0;
        crouching.enabled = true;
        standing.enabled = false;
    }
}
else if(h != 0) //caso seja pressionado uma seta direcional <- ou ->
{
    idAnimation = 1;
}
else //caso contrario o personagem fica parado
{
    idAnimation = 0;
    crouching.enabled = false;
    standing.enabled = true;
}
```

2 TRABALHANDO COM CENÁRIOS

Crie uma pasta dentro dos Assets chamada TileMaps e dentro dela uma pasta chamada Tiles

Crie uma pasta dentro de Sprites chamada Pixel-Art Pack e coloque dentro dela a imagem da aula de hoje. Defina ela como Multiple em Sprite Mode e como Point em Filter Mode.

Corte a Imagem (Grid by Cell Slice) por x 32 y 32 e Slice.

Abra o Tile Palette em Window -> Tile Palette. Create New Palette chame de pixel-art pack 1 (mesmo nome da imagem) e salve na pasta TileMaps dentro da pasta Assets.

Arraste a imagem para dentro do Tile Palette e Salte essa palheta dentro do Tiles em TileMaps, está pronto nossa paleta de imagens para desenhar o cenário.

Agora vamos criar a área para poder desenhar o nosso cenário, pressione o botão direito do mouse na hierarquia do projeto, escolha 2D Object -> TileMap, percebam que ele criou uma Grid para gente.

Percebam o seguinte, dependendo do tamanho da imagem que estivermos trabalhando na nossa paleta de cenário, o tamanho grid tem que ser alterado. Então caso estejamos trabalhando com mais de uma paleta de cenário, sendo que cada uma tem um tamanho diferente de imagem, teremos de possuir mais de um grid, um de cada tamanho necessário.

Nossa paleta de cenário tem imagens de x 32 e y 32 e o grid tem o tamanho de cell size em x e y igual a 1 o que é a mesma coisa que falamos que ele é um grid de 100 por 100, como nossa imagem tem 32 de altura e largura, temos de mudar o tamanho do nosso grid para 0.32 em x e y para que nossas imagens deem certo.

Vamos desenhar um chão para nosso personagem (apagar o chão antigo caso ainda não tenha feito isso) e vamos aproveitar e desenhar um fundo para o chão, reparem que o novo chão não possui colisor fazendo nosso

personagem cair infinitamente, para isso vamos selecionar na grid que acabamos de criar o Tilemap e em seguida adicionar AddComponent -> TileMap -> TileMap Colider 2D.

Agora nosso terreno tem colisor então não vai deixar nosso personagem cair, mas também possuímos agora um monte de quadradinho de colisor, cada quadrado de colisão ocupa um espaço na memória, então quanto mais quadrados de colisão, mais memória precisamos para nosso jogo, uma dica que temos de fazer é trabalhar com 2 grids, uma só para elementos de fundo que não precisam de colisão e outra grid com colisão em seus elementos.

Vamos criar um novo tileMap para nossa aplicação, renomeie a atual criada para Cenário com colisor e clique com o botão direito em grid -> 2D Object -> Tilemap e chame esse de Cenário Sem Colisor, crie mais um Tilemap chamado Fundo Sem Colisor, apague os cenários que não precisam de colisor, e redesenhe-os dentro do Tilemap certo agora.

Figura 14 - Colisores

Cuidado com a camada que cada objeto dentro do cenário está, para que nenhuma imagem fique na frente de seu personagem, caso não queira que isso ocorra, caso necessário mude o Sorting Layer e o Order in Layer do Tilemap ou do próprio personagem para conseguir arrumar esse problema.

Vamos já arrumar isso, vamos criar uma nova Sorting Layer chamada cenário e mudar os 2 grids para estarem nela e vamos criar mais um chamado fundo e arrastar ele para antes do default .

Figura 15 -Criando Sorting Layers

Desenhe o chão na Grid Cenario com Colisor, desenhe a terra no Cenario sem Colisor e o fundo da Caverna na grid Fundo sem Colisor;

Figura 16 - Desenhando Cenario Basico

Nosso personagem está muito grande para o cenário que estamos usando então vamos mudar o tamanho dele, clique no personagem e altere a Scale dele para 0.67.

2.1 ADICIONANDO ELEMENTOS DECORATIVOS AO CENARIO

Vamos agora criar um novo objeto em branco chamado Decoracao, criar uma pasta dentro de Sprites chamada TileObjects e importar os objetos do nosso cenário. Não esqueçam de selecionar todos e colocar que o tipo de Filtro dele é Point (no filter). Todo objeto avulso que formos inserir no nosso cenário, fará parte do nosso objeto vazio Decoracao, não esqueça de pensar em que camada ele tem que estar para não prejudicar a jogabilidade.

Figura 17 - Adicionando Itens ao Cenário

Crie um objeto chamado BackGround e dentro dele crie mais dois Objetos chamados bg1 e bg2, adicione uma Sprite render em cada um deles e arraste para a bg1 o céu azul e para a bg2 as árvores. Crie uma nova Sorting Layer chamada BackGround e a coloque no topo da lista.

Figura 18 - Criando lista de Layers

Duplique seus backgrounds para cobrir todo o espaço do cenário (Ctrl D para duplicar). Uma dica boa para conseguir posicionar uma imagem do lado da outra corretamente é pressionar a tecla V e arrastar a imagem segurando pelo quadrado dela do vetor.

Figura 19 - Multiplos BackGrounds

Se a imagem não der certo o encaixe delas marque o flip de x e veja se melhorou o encaixe;

Figura 20 - Juntando os BackGrounds

Crie diversas camadas e adicione diversos objetos ao seu cenário, uma boa dica é que podemos aumentar e diminuir alfa da imagem para deixa-la menos visível; Clique em Color, Letra A e reduza um pouco para ver que a imagem fica menos visível;

Figura 21 - Alterando o ALFA

Ultima coisa deixem o fundo do cenário da cor do céu, então selecionem Main Camera, vão em Background, cliquem no conta gota e no fundo azul do céu e pronto, agora a imagem do céu se mesclará com o fundo da câmera.

Tentem criar um cenário como esse:

Figura 22 - Desafio Cenario

Pensem em que objeto vai passar na frente ou atrás do personagem e quais vão poder sofrer colisão e quais não.

2.2 CORRIGINDO PROBLEMAS NO COLISOR

Vamos agora arrumar o nosso personagem pois ele está travando em alguns momentos do mapa.

Para isso selecionem o personagem e adicionem o novo componente Circle Colider 2D, ajustem o tamanho desse colisor para ficar no começo do pé e fim da largura do personagem, aproveite também para editar os outros colisores para que agora ele fique dentro do colisor do círculo conforme imagem abaixo, em seguida testem o jogo e veja se está travando ainda.

Figura 23 - Circle Colider

Outra coisa que precisamos de arrumar é que quando ele se colide com uma parede ele trava, isso ocorre por 2 fatores, 1º por padrão os objetos com colisores possuem um pouco de atrito o que faz o personagem “grudar” na parede. Para corrigir isso precisamos criar uma pasta em Assets chamada PhysichMaterial e dentro dele criamos um objeto Physics Material 2D chamado SemAtrito e deixamos o valor de atrito (friction) dele como zero e para que eele comece a valer arrastamos esse objeto pra dentro dos nossos box colider (os 3).

Figura 24 - Material Sem Atrito

A 2ª coisa que precisamos de fazer é que quando importamos nossa paleta de objetos para desenhar o cenário deixamos a opção de Mesh Type como Tyght o que faz com que na hora que ele recorte o objeto, ele tente acompanhar o corte do desenho e precisamos que ele o recorte como um quadrado inteiro. Para corrigir isso precisamos mudar a propriedade para Full Rect e aplicar, em seguida observamos se os objetos colisores ainda continuam com as bordas onduladas, caso isso persista, removemos a camada de colisão deles e adicionamos ela novamente.

Podemos adicionar novos objetos ao nosso cenário com animação, um exemplo disso é a tocha, vamos adicionar uma tocha ao nosso cenário e em seguida vamos selecionar os 3 fogos e arrastar para dentro do cenário perto do objeto tocha que acabamos de inserir, ele vai perguntar onde queremos salvar a animação, escolheremos a pasta Animations e criaremos uma nova pasta chamada Misc onde iremos colocar qualquer animação de cenário, e chamaremos de Fire1.anim, em seguida arraste o Fire_01_anim para dentro do objeto Touch, para que quando duplicarmos o objeto ele já venha com a animação do fogo junto. Não esqueça de arrumar as camadas para que o fogo fique dentro da tocha.

Com a animação do fogo selecionada podemos abrir a janela Animation e configurar para que o fogo fique mais devagar (6 ou 8 já deve ficar bom). Pronto agora quando duplicar o objeto (Ctrl + D) tocha estará tudo certo, com as camadas e com a animação.

2.3 CRIANDO SISTEMA DE INTERAÇÃO COM OBJETOS

Vamos aproveitar e fazer nosso personagem se interagir com itens da fase, vamos por um baú no cenário, adicionar um Box Collider 2D(edite a área de colisão, se necessário) a ele e vamos agora começar a programar nosso personagem para poder interagir com alguns itens específicos, então abram o script do nosso player.

Na parte de declaração de variáveis criem um vetor:

```
private Vector3 dir = Vector3.right; //vetor responsável por movimentar a linha de interação do player
```

E vamos adicionar a seguinte linha no final da programação, após o void attack:

```
void interagir()
{
    RaycastHit2D hit = Physics2D.Raycast(transform.position, dir, 1); //cria uma linha invisível de comprimento 1 para interação a objetos na fase
    Debug.DrawRay(transform.position, dir * 1, Color.red); //cria uma linha visual do mesmo tamanho da invisível apenas para teste
}
```

No fixedupdate() adicione a chamada da função que acabamos de criar:

```
Interagir();
```

E para que quando nosso personagem se mover a linha também se movimente basta adicionar na função flip a linha:

```
dir.x = x;
```

Agora nossa linha se move junto com o personagem, mas para que funcione para abrir um baú ou puxe uma alavanca, essa linha está muito acima e não pega a interação com o objeto, então vamos criar um objeto filho dentro do nosso player chamado hand e vamos posicioná-la na mão do nosso personagem.

No Script crie uma variável publica chamada hand:

```
public Transform hand;
```

E arraste para dentro da variável o objeto hand;

Figura 25 - Objeto Hand

E edite a função interagir para que a linha se inicie pela mão e não mais pelo personagem:

```
void interagir()
{
    RaycastHit2D hit = Physics2D.Raycast(hand.position, dir, 1); //cria
    uma linha invisível de comprimento 1 para interação a objetos na fase
    Debug.DrawRay(hand.position, dir * 1, Color.red); //cria uma linha
    visual do mesmo tamanho da invisível apenas para teste
}
```

A linha está muito grande para a interação, assim nosso personagem vai poder abrir baús estando muito longe deles, vamos diminuir o tamanho da linha para 0.1f

Agora vamos criar uma layerMask para falar que os objetos que vão poder ter alguma interação vão ficar em uma layer diferente na fase, diminuindo o processamento necessário para que isso ocorra. Nas declarações de variáveis vamos digitar:

```
public LayerMask interacao; //layerMask para os objetos que vão
interagir no cenário
```

Em seguida vamos mudar nossa função de interagir e depois do tamanho 0.1f, vamos adicionar o nome da mascara interação;

```
RaycastHit2D hit = Physics2D.Raycast(hand.position, dir, 0.1f,
interacao);//cria uma linha invisivel de comprimento 0.1 para interação a objetos na fase
```

Outra coisa que precisamos fazer é criar a nossa layerMask para interação, então cliquem em add layer:

Figura 26 - LayerMask

E na 8ª posição chamem ela de interacao

Figura 27 - Camada Interação

E mude a layer do baú para interação;

No nosso script crie uma variável publica para sabermos que objeto estamos interagindo naquele momento:

```
public GameObject objetoIntecao; //variavel para armazenar qual objeto está sendo interagido;
```

E na nossa função interação adicionamos a linha:

```
void interagir()
{
    RaycastHit2D hit = Physics2D.Raycast(hand.position, dir, 0.1f,
interacao);//cria uma linha invisivel de comprimento 0.1 para interação a objetos na fase
```

Debug.DrawRay(hand.position, dir * 0.1f, Color.red);*//cria uma linha visual do mesmo tamanho da invisível apenas para teste*

```

if (hit == true)
{
    objetoIntecao = hit.collider.gameObject;
}
else
{
    objetoIntecao = null;
}
}

```

Mude no player o valor da variável interação para interagir a camada interação que acabamos de criar e teste o jogo.

Figura 28 - Amarrando o Script Interação

Pronto, nosso usuário está em contato com objetos que possam interagir, vamos fazer agora que quando pressionarmos a tecla de attack perto de objetos de interação que ele não execute o script de attack e sim interaja com o objeto, para isso vamos mudar a condição de ataque para:

```

if(Input.GetButtonDown("Fire1")&& v >= 0 && attacking == false &&
Grounded == true && objetoIntecao == null) //caso pressione a tecla CTRL ou
esquerdo do mouse e não esteja agaixado será executado a trigger de attack

```

Em seguida vamos criar um script chamado chest na pasta Scripts e arraste este script para dentro do baú.

Abram o script do chest, removam o Update pois não vamos usar e digitem:

```
public void interacao()//função para interagir com objetos
{

}
```

No scrip do nosso personagem vamos adicionar essa condição embaixo da condição de ataque:

```
if (Input.GetButtonDown("Fire1") && v >= 0 && attacking == false &&
Grounded == true && objetoIntecao != null) //caso o personagem esteja em
contato com um objeto possível de ineração, chamar a função interagir
{
    objetoIntecao.SendMessage("interacao",
SendMessageOptions.DontRequireReceiver); //chama o método de interação e
se não houver não retorna erro
}
```

Agora vamos fazer com que o baú mude de imagem quando sofrer interação, vamos no script dele adicionar 3 variáveis e no void start:

Figura 29 - Script Bau

E no objeto chest mover as imagens para as 2 posições:

Figura 30 - Objeto Bau

No 0 a imagem do baú fechado e no 1 o baú aberto;

No script deixe:

```

public void interacao()//função para interagir com objetos
{
    open = !open; //muda de verdadeiro para falso e vice-versa

    switch (open)
    {
        case true:
            spriteRenderer.sprite = imagemObjeto[1];

```

```
        break;
    case false:
        spriteRenderer.sprite = imagemObjeto[0];
        break;
    }
}
```

3 TRABALHANDO COM ARMAS NO PLAYER

Adicionando uma arma ao nosso personagem, primeiro criamos uma pasta chamada Armas dentro da pasta Sprites e importamos as armas para essa pasta. Lembra de Conferir o Sprite Mode para Single e o Wrap Mode para Point (No filter).

Em seguida, criamos um objeto filho dentro de Player chamado Armas e dentro de Armas 3 objetos filhos chamados A, B e C que serão as posições das armas com base na animação que criamos de Attack.

Figura 31 - Armas A, B e C

Vamos adicionar um Sprite Render aos objetos A, B e C e para cada uma, adicionar o Sprite da arma equivalente em cada objeto A – Axe 1, B – Axe 2 e C – Axe 3. Troquem o Ordem in Layer dos 3 objetos em 1 já que nosso personagem está em 0 (zero) e não queremos que ele cubra a arma durante a animação em nenhum momento. Só para não atrapalhar muito vamos tirar o ícone do GroundCheck, assim podemos ver melhor nossa animação.

Agora, vamos abrir a aba Animation, com o player selecionado, escolhemos a animação de Attack, arrastem com o mouse para ver os machados e o nosso personagem atacando, reparem que os machados estão quase na posição correta já do nosso player, precisando mudar muito pouco.

Figura 32 - Personagem com armas

Para não atrapalhar a correção, vamos deixar visível apenas a arma que vamos trabalhar a posição e ocultar as outras, corrigir a posição da arma na mão do personagem, ocultar ela e exibir a próxima, até fazer isso com as 3 posições.

Figura 33 - Arma posição C

Terminou deixe todas as armas invisíveis.

3.1 CRIANDO O SCRIPT PARA AS ARMAS

Abra o Script do player e na parte de declaração de variáveis adicionem a linha de comando:

```
//parte de animação das armas
public GameObject[] armas; //carregando as armas para o player
```

Coloquem o valor 3 e arraste para cada uma das posições uma arma conforme a figura abaixo:

Figura 34 - Amarrando o Script com o Objeto Armas

No Script do Player, na área de Void Start() digitem :

```
foreach (GameObject o in armas)
{
    o.SetActive(false);
}
```

Esse comando garante, por segurança, que nenhuma arma vai ser visível quando se iniciar um cenário. No final do nosso código, criem um novo Void chamado controleArma e digitem:

```
void controleArma(int id) //função para controle da animação das
armas
{
    foreach (GameObject o in armas)
    {
        o.SetActive(false);
    }

    armas[id].SetActive(true);
}
```

No nosso void attack adicionem a linha de comando em amarelo para garantir que ao final do nosso ataque, a arma sumirá:

```
void attack(int atk)
{
    if (atk == 0){
        attacking = false;
        armas[2].SetActive(false);
    }
    if(atk == 1){
        attacking = true;
    }
}
```

Agora para cada frame no Animation Attack, vamos adicionar um novo Evento, em Function escolham o controleArma e no valor deixem 0 para a primeira posição, 1 para a segunda e 2 para a terceira.

Figura 35 - Evento Arma

Rodem o jogo e vejam se a animação está correta.

3.2 SISTEMA DE COLISÃO DA ARMA

Agora precisamos criar um sistema de colisão para a nossa arma, primeiro vamos adicionar um novo personagem a nossa cena para ser nosso boneco de colisão de teste da arma, mudamos um pouco a cor dele e invertemos sua posição no X (caso necessário) em seguida adicionamos um Box Collider a ele e editamos a área de colisão.

Em seguida vamos criar uma TAG chamada arma e vamos definir que os 3 objetos armas nossos fazem parte da TAG Arma:

Figura 36 - TAG ARMA

Agora vamos adicionar um Circle Collider na ponta de cada machado para se nosso colisor, altere as propriedades de Radius, Offset em X e Y para deixar a área de colisão mais parecida possível com essa, NÃO ESQUEÇA DE MARCAR ELA COMO UM TRIGGER:

Figura 37 - Circle Colider no machado

3.3 SCRIPT DANO DA ARMA

Em seguida vamos criar um nosso Script chamado controlDanoInimigo e adicionamos esse script ao nosso inimigo, vamos aproveitar para já inserir no nosso inimigo um Rigidbody 2D e mudar algumas propriedade dela que são: Collision Detection para Continuous, Sleeping Mode para Never Sleep e Freeze Rotation em Z já que nosso personagem inimigo não pode inclinar e cair quando sofrer alguma colisão. Só lembrando que temos de deixar nessas propriedades pq nosso inimigo não está se movendo no mapa, então para ele não parar de sofrer processamento e não receber o dano quando o machado passar por ele temos de falar que a colisão é continua e o personagem não pode entrar no modo de “sleep” para economizar processamento, quando formos fazer nossos inimigos de verdade eles andarão pela sala e, com isso, não entrarão em “sleep”.

Figura 38 - Controle Dano Inimigo

E no Script digitem antes do ultimo } do programa:

```
void OnTriggerEnter2D(Collider2D col)
{
    switch (col.gameObject.tag)
    {
        case "arma":
            print("Tomei Dano");
            break;
    }
}
```

Percebam que no console vai ficar aparecendo a mensagem toda vez que tomar dano:

Figura 39 - Mensagem de dano recebido

Caso mudemos aquelas opções no personagem inimigo, alguns problemas podem começar a aparecer e não marcar corretamente os danos tomados por nossos inimigos.

Figura 40 - Script mensagem de Dano

```
public class controleDanoInimigo : MonoBehaviour {  
  
    // Use this for initialization  
    void Start () {  
  
    }  
  
    // Update is called once per frame  
    void Update () {  
  
    }  
  
    void OnTriggerEnter2D(Collider2D col)  
    {  
        switch (col.gameObject.tag)  
        {  
            case "arma":  
                print("Tomei Dano");  
                break;  
        }  
    }  
}
```

4 KNOCKBACK

Vamos agora fazer com que quando o nosso inimigo tome dano, que ele de um pequeno salto pra trás, para isso crie um objeto vazio dentro do inimigo chamado K e posicione ele embaixo do pé do seu personagem:

Figura 41 - knockback

Agora vamos criar uma nova Layer chamada Inimigo e atribuir ao nosso inimigo essa layer, por ultimo vamos criar um objeto vazio fora de todos os objetos e chama-lo de knockback, adicionar um circle colider a ele e reduzir o tamanho (radius) para 0.15 e marque ele como um Trigger e como um Used By Effector e adicione um novo componente chamado Point Effector. No collider Mask do Point Effector, deixe apenas marcado a Layer inimigo:

Figura 42 - Layer Inimigo

Troque o Force Source para Collider e a Force Magnitude para 75:

Figura 43 - Força da explosão

Crie uma pasta chamada Prefab dentro de Assets e arraste o objeto KnockBack para lá e apague ele da nossa Hierarquia, deixando apenas no Prefab.

Abra o Script do ControleDanoInimigo e no começo do código em declaração de variáveis:

```
//KNOCKBACK
public GameObject knockForcePrefab; //força de repulsão
public Transform knockPosition; //posição da explosão
```

E antes do break coloque:

```
GameObject knockTemp = Instantiate(knockForcePrefab,
knockPosition.position, knockPosition.localRotation); //cria um objeto
temporario para gerar a explosão
```

```
Destroy(knockTemp, 0.03f); //remove o objeto temporario da cena
```

Agora no objeto inimigo temos que arrastar o KnockBack para dentro de KnockForcePrefab e o K para dentro do KnockPosition:

Figura 44 - Prefab Knockback

Teste nosso inimigo. Reparem que mesmo que ataquemos as costas dele ele pula para o mesmo lado então agora vamos arrumar isso.

Vamos declarar essa variável nova:

```
private playerScript playerScript; //cria uma var para armazenar todos os
dados do player
```

Em seguida no void Start vamos deixar:

```
void Start () {
    playerScript = FindObjectOfType(typeof(playerScript)) as
playerScript; //recebe todos os dados do player para trabalhar com as
informações
}
```

E no void Update:

```
//verifica se o player está a direita ou esquerda do inimigo
```

```
float xPlayer = playerScript.transform.position.x;
```

```

if (xPlayer < transform.position.x)
{

}

else if (xPlayer > transform.position.x)
{

}

```

Agora temos de declarar uma variável pública embaixo das do KnockBack chamada:

```

public float knockx; //valor padrão da posição X

private float kx; //variavel responsavel por inverter o valor de x

```

E atribuir o valor de 0.3 a ela no Script:

Figura 45 - knock X

Deixem o void Update assim agora:

//verifica se o player está a direita ou esquerda do inimigo

```

float xPlayer = playerScript.transform.position.x;

if (xPlayer < transform.position.x)
{
    kx = knockx;
}

else if (xPlayer > transform.position.x)

```

```
{
    kx = knockx * -1;
}
```

```
knockPosition.localPosition = new Vector3(kx,
knockPosition.localPosition.y, 0);
```

Agora testem o jogo e reparem que se muda o local do personagem também muda a posição da explosão, só falta agora acertarmos quando o inimigo está virado. Deixem seu Void Update assim:

```
void Update () {

    //verifica se o player está a direita ou esquerda do inimigo

    float xPlayer = playerScript.transform.position.x;

    if (xPlayer < transform.position.x && transform.localScale.x <
0)//Personagem Antes Inimigo e Inimigo Olhando para Esquerda
    {
        kx = knockx;

    }
    else if (xPlayer < transform.position.x && transform.localScale.x >
0)//Personagem Antes Inimigo e Inimigo Olhando para Direita
    {
        kx = knockx * -1;
    }
    else if (xPlayer > transform.position.x && transform.localScale.x >
0)//Personagem Depois Inimigo e Inimigo Olhando para Direita
    {
        kx = knockx;
    }
}
```

```

else if (xPlayer > transform.position.x && transform.localScale.x <
0)//Personagem Depois Inimigo e Inimigo Olhando para Esquerda
{
    kx = knockx * -1;
}

    knockPosition.localPosition = new Vector3(kx,
knockPosition.localPosition.y, 0);
}

```

E testem o nosso jogo.

4.1 INIMIGO TOMANDO DANO E DIMINUINDO VIDA

Vamos agora fazer nosso inimigo tomar dano e ter uma vida, para isso vamos criar um script chamado armaInfo para poder informar os dados da arma do nosso personagem e seu dano.

Arraste esse script para todas as 3 posições da arma.

Abram o Script e digitem:

```
public int dano; //valor do dano que causa ao colidir com um inimigo
```

Apenas isso é suficiente por hora, digitem 5 como valor do dano;

No script ControleDanoInimigo dentro do case de “arma” deixem ele assim:

```

case "arma":

    int danoTomado =
col.gameObject.GetComponent<armaInfo>().dano;

    print("Tomei" +danoTomado +" Dano");

```


Em declaração de variáveis vamos criar uma variável para armazenar a vida do nosso inimigo e digitar o valor 20.

```
public int vidaInimigo; // quantidade de vida que o inimigo terá
```

Agora embaixo da mensagem dentro do CASE ARMA digitamos:

```
vidaInimigo -= danoTomado; // reduz a vida do personagem inimigo
```

E testem para ver se a vida está abaixando como deveria. Por último sumimos com o personagem inimigo quando a vida chegar a zero ou menos:

```
if (vidaInimigo <= 0)
{
    Destroy(this.gameObject);
}
```

Teste o jogo. Agora adicionem mais 2 inimigos na fase duplicando o nosso inimigo e testem o jogo. Reparem que eles são individuais, eles se destroem individualmente e se deslocam individualmente.

Na próxima vamos arrumar que o personagem fique imune um pequeno período de tempo sem poder receber dano para não morrer tão rapidamente.

Vamos arrumar para que nosso inimigo possa andar sem problema pelo terreno, então adicionem um circle collider a ele e façam o mesmo que fizeram ao personagem.

Figura 46 - Inimigo

Em seguida vamos criar dentro da pasta Scripts uma pasta chamada IA e dentro dela um Script chamado goblin, abram o script, na área de declaração de variáveis declarem:

```
private Rigidbody2D rBody;
```

```
public float distanciaMudarRota;
```

```
private Vector3 dir = Vector3.right; //vetor responsável por  
movimentar a linha de interação do player
```

```
public LayerMask layerObstaculos;
```

```
public float velocidadeBase;
```

```
public float velocidade;
```

```
public bool lookLeft;
```

Em Void Start inicie o Rigidbody do inimigo:

```
rBody = GetComponent<Rigidbody2D>();
```

Embaixo do void Update crie ou copie do script do player a função Flip():

```
void flip()
```

```

{
    lookLeft = !lookLeft; //inverte o valor da variavel booleana entre true
e false
    float x = transform.localScale.x;
    x *= -1; //inverte o sinal do scale x
    transform.localScale = new Vector3(x, transform.localScale.y,
transform.localScale.z);

    velocidadeBase *= -1;

    dir.x = x;

}

```

E no void Update digite:

```

void Update () {
    RaycastHit2D hit = Physics2D.Raycast(transform.position, dir,
distanciaMudarRota, layerObstaculos); //cria uma linha invisivel de comprimento
0.1 para interaçao a objetos na fase
    Debug.DrawRay(transform.position, dir * distanciaMudarRota,
Color.red); //cria uma linha visual do mesmo tamanho da invisivel apenas para
teste

    if (hit == true)
    {
        flip();
    }
    rBody.velocity = new Vector2(velocidade, rBody.velocity.y);

    velocidade = velocidadeBase;
}

```

4.2 TROCANDO A TRAJETÓRIA DO INIMIGO

Agora só precisamos criar o objeto responsável pela colisão para que nosso inimigo possa virar quando colidir com ele, crie um objeto vazio chamado `ObstaculoInimigo` e adicione a ele um `BoxCollider`. Deixe o `Size` dele como 0.2. Crie uma `Layer` chamada `obstaculosInimigo` e defina ela como a `layer` do nossos obstáculos.

No inimigo deixe as variáveis com esses valores:

Figura 47 - Obstaculo

Duplicate o obstáculo e deixe cada um em um canto, teste o jogo e veja se está tudo certo.

Para fazer com que a câmera comece a seguir o nosso player precisamos mudar a TAG dele para `Player` e em seguida crie um Script chamado `camera` na pasta de `Scripts` e adicione ele ao objeto câmera, digite:

```
private Transform alvo;
float velocidade = 100000;

void Start () {

    alvo = GameObject.FindGameObjectWithTag("Player").transform;
```

```

    }

    // Update is called once per frame
    void Update () {

        if (alvo)
        {
            transform.position = Vector3.Lerp(transform.position,
alvo.position, velocidade) + new Vector3(0, 0, -10);
        }

    }

```

Teste o jogo.

4.3 SISTEMA DE RETORNO

Outra coisa que vamos aproveitar e arrumar é quando o personagem sai do cenário para ele voltar ao ponto inicial, para isso criamos um objeto vazio chamado retorno, adicionamos um box collider que fique em baixo da fase toda, extensão toda dela, marcamos ele como um trigger.

Figura 48 - Retorno

Em seguida criamos um script chamado retorno e adicionamos ao objeto retorno, sempre que o objeto player colidir com o objeto retorno, a posição do

player será alterada para o começo da fase novamente, para isso digitem no script:

```
public GameObject player;

public float x;
public float y;
public float z;

void OnTriggerEnter2D(Collider2D alvo)
{
    if (alvo.tag == "Player")
    {
        player.transform.position = new Vector3(x, y, z);
    }
}
```

Depois deixem as propriedades do Script retorno assim:

Figura 49 - tempo para o retorno

Testem o jogo.

4.4 SISTEMA DE IMUNIDADE POR TEMPO

Agora para que nosso inimigo e, futuramente até nosso personagem não possa tomar mais de um hit ao mesmo tempo, vamos fazer com que quando ele

tomar dano, ele não possa tomar outro por um curto período e depois volte ao normal, para isso vamos no script controleDanoInimigo e declarem essa variável:

```
private bool getHit; //indica se tomou um hit
```

No case Arma, remova aquele print TOMEI DANO já que ele foi feito apenas para testar a colisão e adicione um if:

```
case "arma":
```

```
    if (getHit == false)
```

```
    {
```

```
        getHit = true;
```

```
        int danoTomado =
```

```
col.gameObject.GetComponent<armaInfo>().dano;
```

```
        vidaInimigo -= danoTomado; // reduz a vida do
```

```
personagem inimigo
```

```
        if (vidaInimigo <= 0)
```

```
        {
```

```
            Destroy(this.gameObject);
```

```
        }
```

```
        GameObject knockTemp = Instantiate(knockForcePrefab,
knockPosition.position, knockPosition.localRotation); //cria um objeto
```

```
temporario para gerar a explosão
```

```
        Destroy(knockTemp, 0.03f); //remove o objeto temporario
```

```
da cena
```

```
    }
```

```
break;
```

Agora declaro essas novas variáveis e inicie elas no Start:

```

public Color[] characterColor;//controle de cor do nosso personagem
para piscar quando tomar dano

private SpriteRenderer sRender; //instanciando o sprite renderer do
inimigo para fazê-lo piscar ao tomar dano

```

```

// Use this for initialization
void Start () {
    sRender = GetComponent<SpriteRenderer>(); //inicia o srender
    sRender.color = characterColor[0]; //atribui a cor inicial ao inimigo

```

Em seguida mude as propriedades do nosso inimigo para deixar assim:

Figura 50 - piscando após o HIT

A primeira cor branca com a camada Alfa em 255 e a segunda tbm branca mas com a camada Alfa em 100;

Para fazer o inimigo pisca vamos criar um corrotina para ser executada em paralelo a nossa aplicação, após o void OnTriggerEnter2D digite:

```

IEnumerator invulneravel()
{
    sRender.color = characterColor[1];
    yield return new WaitForSeconds(0.2f);
    sRender.color = characterColor[0];
    yield return new WaitForSeconds(0.2f);

```



```
sRender.color = characterColor[1];  
yield return new WaitForSeconds(0.2f);  
sRender.color = characterColor[0];  
yield return new WaitForSeconds(0.2f);  
sRender.color = characterColor[1];  
yield return new WaitForSeconds(0.2f);  
sRender.color = characterColor[0];  
  
getHit = false;  
}
```

E para chamar a corrotina na última linha do IF da arma digite:

```
StartCoroutine("invulneravel"); //chama a corrotina para piscar o inimigo  
ao sofrer dano
```

Teste o jogo;

5 CAMADA HUD

Vamos implementar uma barra de vida em cima de nossos inimigos e personagem, para isso importe a barra nos materiais disponíveis e mude as propriedades dela de Filter Mode para (poin no filter) e Pivot para LEFT, em seguida crie um objeto vazio filho para o inimigo e chame ele de hpBar, arraste a imagem da barra para ele, posicione em cima da cabeça do inimigo; Crie uma nova Sorting Layer chamada HUD e atribua ela a barra, lembre-se que a HUD sempre será nossa última layer para que nenhum objeto fique na frente dela;

Figura 51 - camada HUD

Agora duplique a barra e deixe a primeira com a cor de fundo vermelha e a segunda amarela e mude a Order in Layer da segunda para 1, outra coisa, renomeie a primeira barra para barraBG e a segunda para barraVida e deixe o Scale das duas barras em 1.

Figura 52 - barra de vida

No controleDanoInimigo declare as variáveis embaixo da variável VidaInimigo:

```
public int vidaAtual; //vida atual do inimigo após tomar dano
public GameObject barrasVida; //objeto contendo todas as barras
public Transform hpBar; //objeto indicando a barra de vida
```

E agora arraste os objetos para nossa barrasVida e hpBar:

Figura 53 - Controle dano Inimigo

Em seguida no void Start digite:

```
barrasVida.SetActive(false);
```

E embaixo do getHit no Arma digite:

```
barrasVida.SetActive(true);
```

Agora vamos começar a alterar o tamanho da barra com base no dano tomado, volte na declaração das variáveis e declare:

```
private float perVida; //controla o percentual de vida
```

No void Start():

```
vidaAtual = vidaInimigo;
hpBar.localScale = new Vector3(1, 1, 1);
```

E troque o cálculo de dano, lá no case Armas a variável de vidaInimigo para vidaAtual:

```
vidaAtual -= danoTomado; // reduz a vida do personagem inimigo
```

```
if (vidaAtual <= 0)
{
    Destroy(this.gameObject);
}
```

Agora entre o vidaAtual e o if vamos colocar a atualização da barra de vida:

```
percVida = (float)vidaAtual / (float)vidaNimigo;
if (percVida < 0) {percVida = 0;}
```

```
hpBar.localScale = new Vector3(percVida, 1, 1); //atualiza a barra de vida
```

Testem o jogo. Reparem que quando o inimigo olha para a esquerda, a barra de vida também vida e fica estranha a forma como mostra o dano tomado, para arrumar isso precisamos virar a barra sempre que o personagem virar. Para isso, adicione esse código ao final do void Update:

```
barrasVida.transform.localScale = new Vector3(transform.localScale.x,
barrasVida.transform.localScale.y, barrasVida.transform.localScale.z); //muda o sentido da barra de vida se o inimigo virar
```

E agora a barra de vida acompanha a rotação do inimigo. Testem o jogo.

Para melhorar o sistema de vida do nosso jogo, vamos adicionar um texto a nossa cena que informe quanto de dano que o inimigo está tomando, para isso criem um objeto vazio chamado danoTXT e adicione a ele o componente Text Mesh, em text digite um número qualquer, deixe a fonte como arial e o Font Size como 100 para remover aquele embaçado da fonte, em seguida vamos mudar a Scale dela para 0.03 em X e Y; Adicione um Rigidbody e marque o Freeze Rotation em Z. Salve o objeto na pasta dos Prefabs e depois exclua da cena;

Figura 54 - Freeze Rotation Z

Abra o script de controleDanoInimigo e declare a variável:

```
public GameObject danoTXTPrefab; //objeto com o texto do dano tomado
```

Em seguida arraste o danoTXT em Prefab para dentro da variável:

Figura 55 - Dano TXT Prefab

E digitem:

```

GameObject danoTemp = Instantiate(danoTXTPrefab,
transform.position, transform.rotation); //cria um objeto temporario para exibir o
dano tomado

danoTemp.GetComponent<TextMesh>().text =
danoTomado.ToString(); //atribui o valor do dano tomado ao objeto criado
int forcaX = 50;
if (transform.position.x < playerScript.transform.position.x)
//se o personagem estiver a direita do inimigo
{
    forcaX *= -1;
}
danoTemp.GetComponent<Rigidbody2D>().AddForce(new
Vector2(forcaX, 230)); //adiciona força para o objeto ir para cima
Destroy(danoTemp, 1.0f); //remove o objeto da cena

```

Teste o jogo, Reparem que dependendo de onde é gerado o texto do dano, ele tende a ficar atrás de alguns objetos, para garantir que ele fique na frente de tudo para que possamos ver com mais clareza o valor do dano que o inimigo está tomando, para isso adicionem esse comando antes do `int forcaX`:

```

danoTemp.GetComponent<MeshRenderer>().sortingLayerName =
"HUD"; //deixa o texto na frente de todos os objetos dentro da cena

```

Agora sim nosso texto está na frente de tudo, testem o jogo.

5.1 ANIMAÇÕES INIMIGO

Vamos trabalhar com as animações dos nossos inimigos, para isso duplique as animações do Player (controller em animations) e chame de inimigo. Em seguida abra o animation e exclua a animação de pular e abaixar.

Figura 56 - animação inimigo

Adicione uma nova transição de HIT -> DIE :

Reset

Desmarque Has Exit Time

IdAnimation = 3

Adicione a nova animação inimigo ao Controller do Animator do nosso inimigo.

No Script ControleDanoInimigo declare :

```
private Animator animator; //da acesso a animação do inimigo
```

No Start digite:

```
animator = GetComponent<Animator>();//inicia o animator
```

Na ultima linha do Update:

```
animator.SetBool("grounded", true);//define manualmente que o inimigo  
está pisando no chão
```

No case Arma, após o cálculo de dano tomado:

```
animator.SetTrigger("hit");//chama a animação de hit
```

Abra o Animation e mude a animação de Die para que o ultimo frame dure 3 segundos:

Figura 57 - Time Line

Em seguida quando a vida dele abaixa ou chega a zero, deixe assim:

```
if (vidaAtual <= 0)
{
    animator.SetInteger("idAnimation", 3);//chama a  
animação de DIE
    Destroy(this.gameObject, 2);
}
```

Teste o jogo

5.2 ANIMAÇÃO DE HIT

Agora vamos fazer uma animação para que quando acertar alguém, crie uma animação de explosão. Crie uma pasta chamada Effects em Sprites e importe as três imagens da aula de hoje. Não esqueça de mudar para Multiple e Point. Recorte as imagens por 192 x 192. Crie um objeto vazio e chame ele de hit_normal, adicione um Animator e um Sprite Render, no Sprite Render coloque a primeira imagem do Attack_1 apenas para vermos a imagem na cena. Em seguida crie um novo animator chamado hit_normal na pasta Controller:

Figura 58 - hit normal

Arraste o hit_normal no animator do objeto novo:

Figura 59 - objeto Hit Normal

Abra a aba Animation, crie uma nova chamada hit_normal dentro de Animations -> fx arraste os 3 frames para a animação e mude o tempo para 8 de duração:

Figura 60 - Animação Hit

No Animator, crie um State vazio e marque uma transição entre o hit_normal e o new State:

Hit_normal -> New State

Reset

Deixe marcado Has Exit

Agora no objeto hit_normal, em Sprite Render, volte para NONE.

Teste o Jogo. Percebam que vai ocorrer 1 vez a explosão e vai parar. A animação está muito grande para o nosso personagem então vamos mudar o Scale de x e y do hit_normal para 0.2.

Figura 61 - prefab hit Normal

Em prefab crie uma pasta chamada fx e arraste o hit_normal para lá

No controleDanoInimido declare:

```
public GameObject anima_dano;//objeto com a animação de quando  
sofre dano
```

Arraste para cá a animação:

Figura 62 - Script Dano Inimigo

Quando ocorre o hit digite:

```
GameObject animaTemp = Instantiate(anima_dano, transform.position,  
transform.localRotation);//cria um objeto temporario para fazer a animação do  
dano
```

```
Destroy(animaTemp, 1.0f);//remove o objeto
```

Teste o jogo. Faça a outra animação caso queira alterar essa.

5.3 ANIMAÇÃO DE MORTE DO INIMIGO

Agora vamos fazer a animação de quando o inimigo morre, para isso só fazemos o que acabamos de fazer, crie um objeto vazio chamado death, adicione o Sprite render e um animator, na pasta Controller, crie um novo animator controller chamado morte, arraste a morte para dentro do controller do objeto death, na aba Animation crie uma nova animação chamada death dentro da pasta fx, arraste os sprites que achar mais interessante para a sua animação de morte e mude o tempo do Sprite para 8, ou o que achar melhor.

Figura 63 - Animação de Morte

Não precisa mudar a escala da animação já que ela vai acontecer enquanto o personagem está deitado, uma coisa que precisa mudar é o Order in Layer dela para pelo menos 5 ou maior para garantir que a animação de morte não fique na frente de nenhum objeto da mesma camada.

Vá na aba animator, cria um new State vazio, coloque a transição de Death para New State:

Reset

Deixe marcado e só

Salve a Death como um prefab em FX e apague da cena do jogo.

Crie um objeto chamado groundCheck no nosso inimigo, coloque um ícone e arraste o objeto para debaixo do pé do nosso inimigo, ele será o responsável por informar de onde a animação irá sair, e depois pode remover o ícone já que não vamos mais usar por um bom tempo.

Abra o script ControleDanoInimigo, declare as variáveis :

`//LOOT e Death`

`public Transform grounCheck; //objeto responsável por informar onde é chão`

`public GameObject fxMorte; //objeto com a animação de morte`

Em seguida arraste para cada um o objeto dele:

Figura 64 - Ground Check

Agora para programar a animação precisamos de fazer uma corrotina chamada loot :

`IEnumerator loot()`

```
{
    yield return new WaitForSeconds(1);
```

```
GameObject fxMorteTemp = Instantiate(fxMorte,
grounCheck.position, transform.localRotation);//chama o objeto com a
animação da morte
```

```
yield return new WaitForSeconds(0.5f);
sRender.enabled = false;
yield return new WaitForSeconds(0.7f);
Destroy(fxMorteTemp);
Destroy(this.gameObject);
}
```

Deixe o if (vida <= 0) desse jeito:

```
if (vidaAtual <= 0)
{
    animator.SetInteger("idAnimation", 3);//chama a
animação de DIE
    StartCoroutine("loot"); //chama a corrotina para matar o
inimigo e gerar o loot
}
```

Teste o jogo para ver se está certo a animação.

6 SISTEMA DE LOOT

Agora vamos adicionar o loot ao matar um inimigo ou abrir um baú, por exemplo, para isso crie um objeto vazio chamado Coin dentro dele crie um objeto vazio chamado spriteCoin, em spriteCoin, adicionem os componentes Sprite Renderer dentro dele adicionem um Sprite de uma moeda, adicionem um Circle Colider 2D e editem a área de colisão da moeda, criem uma nova Sorting Layer chamada Loot e coloquem ela antes do HUD, criem uma Layer chamada Coletável e apliquem essa Layer ao spriteCoin, adicionem um componente de Animator, agora no objeto Coin, adicionem o componente RigidBody 2D, marquem o Freeze Rotation Z, adicionem um Circle Collider 2D um pouco maior que o anterior da moeda e marquem como isTrigger.

Figura 65 - icone coin

Figura 66 - Config Coin

6.1 ANIMATION COIN

Criem uma Tag também chamada Coletável e marquem seu Coin como ela. Na pasta Animations -> Controller, crie um novo Animator Controller chamado Coin e arraste ele pra dentro do Controller do Animator do objeto spriteCoin:

Figura 67 - Animation Coin

Abra a aba Animation, crie uma nova Animation salve ela com o nome de Coin dentro da pasta Animattions -> Misc. Arraste para dentro da sua animação os sprites do coin, mude o tempo para 8. Dentro da pasta dos Prefabs crie uma nova pasta chamada Coletáveis e arraste o objeto Coin para dentro dessa pasta. Apague da cena;

Declare a variável loots dentro do Script ControleDanolnimigo:

```
public GameObject loots; //objeto com os possíveis loots do inimigo
```

Arraste o Coin para a nova variável, lembrando que por enquanto só vai cair o coin do inimigo mas, depois de um tempo, vamos criar um vetor que vai armazenar os diversos loots que podem cair de cada inimigo;

Declare a variável:

```
public int totalLoot; //informa quantos itens no máximo um inimigo pode derrubar
```

Atribua o valor 5 a essa variável, na função Loot adicione:

```
IEnumerator loot()
```



```

{
    yield return new WaitForSeconds(1);
    GameObject fxMorteTemp = Instantiate(fxMorte,
grounCheck.position, transform.localRotation);//chama o objeto com a
animação da morte
    yield return new WaitForSeconds(0.5f);
    sRender.enabled = false;

    //controle de loot
    int qtdloot = Random.Range(1, totalLoot);//gera uma quantidade
randomica de itens
    for (int i = 0; i < qtdloot; i++)
    {
        GameObject lootTemp = Instantiate(loots, transform.position,
transform.localRotation);//cria o objeto loot na cena
        lootTemp.GetComponent<Rigidbody2D>().AddForce(new
Vector2(Random.Range(-40, 40), 100));//adiciona uma força aleatória para o
objeto loot
        yield return new WaitForSeconds(0.1f);//espera o frame
terminar para criar um novo loot
    }

    yield return new WaitForSeconds(0.7f);
    Destroy(fxMorteTemp);
    Destroy(this.gameObject);
}

```

Vá em Edit -> Projet Settings -> Physics 2D e deixe apenas que a Layer Coletável colide com a Layer Default apenas :

Figura 68 - Matrix Camadas de Colisão

Agora no script do Player crie uma função no fim do código:

```
void OnTriggerEnter2D(Collider2D col)
{
    switch (col.gameObject.tag)
    {
        case "Coletavel":
            Destroy(col.gameObject);
            break;
        default:
            break;
    }
}
```

Teste o jogo. Agora vamos fazer com que o Bau também saia Loot dele, para isso, abram o Script do Chest, declare como fizemos antes:

```
//controle de loot
public GameObject loots; //objeto com os possíveis loots do inimigo
public int totalLoot; //informa quantos itens no máximo um inimigo
pode derrubar
```

Arraste o objeto Coin para lá

Figura 69 - Coin ao Loot do Baú

Copiem o controle de loot do Script ControleDanoInimigo, dentro da função loot e cole dentro de uma corrotina que vamos criar chamada gerarLoot:

```
IEnumerator gerarLoot()
{
    //controle de loot
    int qtdloot = Random.Range(1, totalLoot); //gera uma quantidade
    //randomica de itens
    for (int i = 0; i < qtdloot; i++)
    {
        GameObject lootTemp = Instantiate(loots, transform.position,
        transform.localRotation); //cria o objeto loot na cena
        lootTemp.GetComponent<Rigidbody2D>().AddForce(new
        Vector2(Random.Range(-40, 40), 100)); //adiciona uma força aleatória para o
        //objeto loot
        yield return new WaitForSeconds(0.1f); //espera o frame
        //terminar para criar um novo loot
    }
}
```

Chamem a corrotina quando o baú é aberto:

```
switch (open)
{
    case true:
        spriteRenderer.sprite = imagemObjeto[1];
        StartCoroutine("gerarLoot");
        break;
    case false:
        spriteRenderer.sprite = imagemObjeto[0];
        break;
}
```

Testem o jogo.

6.2 GERENCIANDO LOOTS DO BAÚ

Percebam que o loot gerado pelo baú é meio que infinito, já que sempre que abrimos o baú, será gerado mais e mais loot, para arrumar isso, vamos criar uma variável booleana de controle:

```
private bool gerouLoot; //variável de controle que faz com que gere loot apenas uma vez
```

E vamos criar um condicional if que faz gerar loot apenas se o valor da variável de controle for false:

```
case true:
    spriteRenderer.sprite = imagemObjeto[1];
    if (gerouLoot == false)
    {
        gerouLoot = true;
        StartCoroutine("gerarLoot");
    }
    break;
```

Teste o jogo. Já está melhor, só falta arrumarmos para poder passar por dentro do baú, já que não conseguimos pegar as moedas que são geradas no meio dele. Para isso vamos entrar de novo em Edit -> Projects Settings -> Physical 2D e falar que a Layer que o baú está, não colide com a Layer do personagem nesse caso, a Layer Interação, não colide com a Layer Default:

Figura 70 - Matrix de Colisão Loot

Se quiser também podemos colocar o baú em uma camada atrás do nosso player para que quando o mesmo passe pelo baú, passe pela frente e não por trás.

Figura 71 - Sorting Layer Fundo

6.3 IMPLEMENTANDO CONTADOR DE COINS

Agora sim, testem o jogo e percebam que o player passa na frente do baú e é gerado apenas o loot na primeira vez que é aberto. Para finalizar o sistema de Loot, vamos criar mais de uma opção de moeda neste momento mas o mesmo pode implementar diversos outros itens na hora de gerar o loot. Crie na pasta Scripts um script chamado `_GameController`, crie uma variável para armazenar o gold nosso:

```
public int gold; //armazena quanto gold o player possui
```

Crie um objeto vazio chamado `_GameController` e arraste o script `_GameController` para ele, agora no script do player chame a nova variável:

```
private _GameController _GameController;
```

No void Start iniciamos a `_GameController`:

```
_GameController = FindObjectOfType(typeof(_GameController)) as  
_GameController; //iniciamos a variável ao objeto externo
```

E na onde estamos tendo a interação com a coleta de gold adicionamos mais um ao nosso contador:

```
case "Coletavel":  
    _GameController.gold += 1;  
    Destroy(col.gameObject);  
    break;
```

Teste o jogo, vejam se está contando corretamente, agora vamos criar dentro da pasta Scripts uma pasta chamada Coletaveis e dentro dela um script chamado coin, arraste este script para dentro do nosso objeto prefab Coin.

No script deixe assim:

```
public class coin : MonoBehaviour {  
  
    private _GameController _GameController;
```

```

public int valor; //valor da moeda

// Use this for initialization
void Start () {
    _GameController = FindObjectOfType(typeof(_GameController))
as _GameController; //iniciamos a variável ao objeto externo
}

// Update is called once per frame
void Update () {

}

public void coletar()
{
    _GameController.gold += valor;
    Destroy(this.gameObject);
}
}

```

Na onde chama a função coletar deixe assim:

```

switch (col.gameObject.tag)
{
    case "Coletavel":
        col.gameObject.SendMessage("coletar",
SendMessageOptions.DontRequireReceiver); //chama a função coletar do
objeto em interação, caso não haja a função não retorna erro
        break;
    default:
        break;
}

```

E teste o jogo. Agora vamos criar uma nova moeda, arraste uma Moeda do Prefab para a cena e a renomeie para Coin2, mude a cor no Sprite Renderer

dela para outra, altere o valor da moeda de 1 para 5, Salve a nos Prefabs de Coletáveis.

Vamos primeiro mudar o loot do baú e depois mudamos o do inimigo, então no Script do Chest na onde declaramos a variável loots, troque para um array só colocando [] :

```
public GameObject[] loots; //objeto com os possíveis loots do inimigo
```

E na onde ele chamava a função de loot, comente as 2 linhas:

```
//GameObject lootTemp = Instantiate(loots, transform.position,
transform.localRotation); //cria o objeto loot na cena
//lootTemp.GetComponent<Rigidbody2D>().AddForce(new
Vector2(Random.Range(-40, 40), 100)); //adiciona uma força aleatória para o
objeto loot
```

Agora salve e troque o valor da variável para 2, arraste o Coin para a primeira e o Coin2 para o segundo:

Figura 72 - Valor do Coin

Em seguida deixe a função de gerar loot do chest dessa forma:

```
IEnumerator gerarLoot()
{
    //controle de loot
    int qtdloot = Random.Range(1, totalLoot); //gera uma quantidade
    randomica de itens
    for (int i = 0; i < qtdloot; i++)
```



```

{
    int id = 0;
    int rand = Random.Range(0, 100);
    if (rand > 50)//se o numero gerado foi maior que 50, então será
gerado a outra moeda
    {
        id = 1;
    }
    GameObject lootTemp = Instantiate(loots[id], transform.position,
transform.localRotation);//cria o objeto loot na cena
    lootTemp.GetComponent<Rigidbody2D>().AddForce(new
Vector2(Random.Range(-40, 40), 100));//adiciona uma força aleatória para o
objeto loot
    yield return new WaitForSeconds(0.1f);//espera o frame
terminar para criar um novo loot
}
}

```

Va no Script do Coin e mude para:

```

public void coletar()
{
    _GameController.gold += valor;
    Destroy(this.gameObject);
}

```

Teste o jogo percebam que agora está sendo gerado tanto a primeira moeda, quando a segunda moeda de forma aleatória, vale lembrar que dessa forma podemos gerar diversos itens diferentes no mesmo baú, fazendo com que o nosso loot seja mais diversificado como poções, armas, reagentes e por ai vai. Faça o mesmo com o loot do inimigo.

6.4 SISTEMA DE MUDANÇA DE PERSONAGEM

Vamos criar o sistema de mudança de personagens, para que isso funcione precisamos ter a certeza de duas coisas, primeiro que todas as imagens dos nossos personagens estão dentro da mesma pasta e segundo que todos os sprites de cada personagem possuam o mesmo nome Attack_01, Walk_03 e por aí vai, vale lembrar que é sensitive case, então se escreveu com letra maiúscula tem que estar todas escritas do mesmo jeito, se estiver tudo minúsculo também.

Vamos agora criar uma pasta chamada Resources dentro de Assets, em seguida vamos criar um Script chamado reSkin dentro da pasta Scripts.

Declare isso no começo do código:

```
using System.Linq; //da acesso a mudar o dicionario das animações
```

Em seguida declare as variáveis:

```
private SpriteRenderer sRender;
```

```
public Sprite[] sprites; //vetor que irá armazenar todos os sprites do
personagem escolhido
```

```
public string spriteSheetName; //nome da imagem que contem os
sprites do personagem escolhido
```

```
public string loadedSpriteSheetName; //nome do spriteSheet em uso
caso o personagem tenha mudado
```

```
private Dictionary<string, Sprite> spriteSheet; //dicionario que
armazena as informações dos sprites
```

Após isso crie uma função:

```
private void loadSpriteSheet()
{
    sprites = Resources.LoadAll<Sprite>(spriteSheetName); //carrega
os sprites do nome que passamos pela variável spriteSheetName
```

```

        spriteSheet = sprites.ToDictionary(x => x.name, x =>
x);//alimentando o Dictionary com os sprites carregados
        loadedSpriteSheetName = spriteSheetName;//informa qual o
nome da imagem q esta atualmente em uso
    }

```

Mude o Update para LateUpdate (o late para caso haja mudança de Sprite, que ele carregue uma por vez ao contrário do Update que carregaria tudo de uma vez, podendo ocasionar erro e recebimento de múltiplos valores em apenas um espaço no vetor ao contrário de em vários) e digite:

```

void LateUpdate () {
    if (loadedSpriteSheetName != spriteSheetName)//caso haja
mudança de personagem
    {
        loadSpriteSheet();
    }

    sRender.sprite = spriteSheet[sRender.sprite.name];//amarra as
animações com o nome atual do Dictionary
}

```

E no Start inicie o sRender e chame a função:

```

sRender = GetComponent<SpriteRenderer>();//inicia o rRender
loadSpriteSheet();

```

Arraste o Barbarian e o Goblin para dentro da pasta Resources, arraste o script reSkin para dentro do player e do inimigo e no Sprite Sheet Name do player e do inimigo dê o nome certo para cada um:

Figura 73 - Re Skin

Teste o jogo, percebam que agora temos como mudar as animações apenas mudando o nome digitado, mude o reSkin do player para goblin e veja como é jogar com o goblin.

Façam a importação dos outros personagens, seus cortes e testem no player. Não esqueçam Multiple e Point no Filter, a imagem é 240 por 240.

6.5 TRABALHANDO COM CANVAS

Agora vamos adicionar alguns elementos visuais a tela do nosso personagem, primeiro vamos importar o material da aula tudo na pasta Sprites (hud1 e 2), mude as propriedades já de costume, no HUD 1, entre em Sprite Editor, Slice, Automatic, Pivot Center e Slice, Apply. No HUD 2 corte com X 30 Y 45 , Slice e Apply.

Agora em Hierarchy, botão direito, UI, Canvas, ele vai criar o Canvas e um EventSystem, pode deletar o EventSystem.

Figura 74 - canvas

Vá na Aba Game e do lado do Display 1, mude, caso não esteja, para 16:9.

Figura 75 - Scene

Em Canvas, mude a propriedade do Render Mode para Screen Space – Camera e no campo Render Camera, arraste a nossa câmera para dentro deste campo, em Sorting Layer escolha a camada que já tínhamos criado antes chamada HUD e em order in Layer digite 50 para ter certeza que não vai ter nada na frente dele.

Figura 76 - Order in Layer Canvas

Agora em Canvas Scaler, mude o UI Scale Mode para Scale With Screen Size e coloque a resolução de x 1920 e y 1080, em Match digite 0.5 para que ele possa se adaptar as variações de resolução tanto na ALTURA quanto na LARGURA.

Figura 77 - Script Canvas

Renomeie o Canvas para HUD e deixe-o abaixo da câmera:

Figura 78 - HUD Canvas

Lembrando que quanto mais elevado o objeto na posição Hierárquica, mais rápido ele é carregado na construção da cena do jogo.

6.6 HP BAR

Botão direito em cima do HUD -> UI -> Image, mude o Pivot para X 0 e Y 1, na Anchor Presents, mude para TOP LEFT

Figura 79 - Position

Mude o Pos X e Pos Y para 45 e -45, arraste o coração do HUD 1 para dentro do Source Image, marque para preservar o Aspecto e mude o Width e Height para 75.

Figura 80 - Rect Transform

Figura 81 - Coração Logo

Renomeie para HP Bar, crie mais uma imagem como filho de HP Bar, mude o Width e Height para 50 e arraste no HUD 2 (a primeira imagem) para dentro do Source Image, masque para preservar o aspecto, mova a imagem para ficar ao lado do coração e renomeie para Bar1:

Figura 82 - HP BAR

Crie uma outra imagem como filho de Bar1, chame ela de hp1, arraste no HUD2 a imagem da vida cheia, preservar o aspecto, altura e largura em 50:

Figura 83 - bar1

Figura 84 - Logo barra vida

Duplicate o Bar1, chame de Bar2, renomeie o hp1 do Bar2 para hp2, posicione ao lado do Bar1 e refaça isso até ter 5:

Figura 85 - Barra de vida cheia

Figura 86 - Bar5

6.7 GOLD BAR

Agora para o Coin, crie uma imagem filho do HUD, mude o Pivot dela para 1 em X e Y e TOP RIGHT, arraste a Moeda para a imagem e marque para preservar o Aspecto, Pos X e Pos Y -45 e altura e largura como o coração em 75, renomeie para Gold Bar:

Figura 87 - Gold Bar

Adicione uma imagem filho ao Gold Bar chamada bar e adicionem a barra amarela, Mude para Middle Right, marque o preserve aspect, mova ela para o lado e aumente até o tamanho que achar estar bem:

Figura 88 - Logo Barra Gold

Adicionem como filho do bar um TextMeshPro Text, se pedir para fazer download, faça e espere terminar a importação, mude o tamanho da área para caber dentro da barra de Gold, mude o alinhamento para direita e Centro, e coloque o número 0 como texto para orientação durante essa etapa de formatação. Renomeie para GoldTxt.

Figura 89 - Contador Barra Gold

Abra o Script do _GameController, declare:

```
using TMPro;
```

Em declarações de variáveis declare:

```
public TextMeshProUGUI goldTxt; //inicia o objeto que vai receber o valor  
do ouro atual
```

Salve, arraste o GoldTxt para lá

Figura 90 - Script Gold Bar

No void Update:

```
goldTxt.text = gold.ToString();//passando o valor do gold para a tela
```

Teste o jogo.

Agora vamos programar a vida do nosso player. Para isso abra o Script do player:

```
public int vidaAtual, vidaMax; //declara as variáveis que vão receber o valor da vida atual
```

No void Start()

```
vidaAtual = vidaMax;
```

Em Seguida informe que vida máxima é 10, crie um script chamado hud e atribua ele ao objeto HUD nosso:

```
using UnityEngine.UI; //dá acesso a UI do jogo
```

Declare as variáveis:

```
private playerScript playerScript; //dá acesso ao script do player
public Image[] hpBar; //objeto com a barra de vida
public Sprite half, full; //objeto que armazena a imagem de meia ou cheia vida
```

No void Start:

```
playerScript = FindObjectOfType(typeof(playerScript)) as playerScript;
//inicia o script do player
```

Em seguida salve seu script, em HPBar digite o valor 5 arraste para cada posição uma hpbar do player, no half e full arraste os objetos correspondentes:

Figura 91 - Elements HP BAR

Agora criamos uma função para calcular a % de vida chamada controleBarraVida:

```
void controleBarraVida()
{
    float percVida = (float)playerScript.vidaAtual /
(float)playerScript.vidaMax;//calcula da % da vida do player

    foreach (Image img in hpBar)
    {
        img.enabled = true;
        img.sprite = full;
    }
    if (percVida == 1)//100%
    {

    }
    else if (percVida >= 0.9f)//90 a 99%
    {
        hpBar[4].sprite = half;
    }
    else if (percVida >= 0.8f)//80 a 89%
```

```
{
    hpBar[4].enabled = false;
}
else if (percVida >= 0.7f)//70 a 79%
{
    hpBar[4].enabled = false;
    hpBar[3].sprite = half;
}
else if (percVida >= 0.6f)//60 a 69%
{
    hpBar[4].enabled = false;
    hpBar[3].enabled = false;
}
else if (percVida >= 0.5f)//50 a 59%
{
    hpBar[4].enabled = false;
    hpBar[3].enabled = false;
    hpBar[2].sprite = half;
}
else if (percVida >= 0.4f)//40 a 49%
{
    hpBar[4].enabled = false;
    hpBar[3].enabled = false;
    hpBar[2].enabled = false;
}
else if (percVida >= 0.3f)//30 a 39%
{
    hpBar[4].enabled = false;
    hpBar[3].enabled = false;
    hpBar[2].enabled = false;
    hpBar[1].sprite = half;
}
else if (percVida >= 0.2f)//20 a 29%
{

```

```

        hpBar[4].enabled = false;
        hpBar[3].enabled = false;
        hpBar[2].enabled = false;
        hpBar[1].enabled = false;
    }
    else if (percVida >= 0.1f)//10 a 19%
    {
        hpBar[4].enabled = false;
        hpBar[3].enabled = false;
        hpBar[2].enabled = false;
        hpBar[1].enabled = false;
        hpBar[0].sprite = half;
    }
    else if (percVida <= 0.0f)//0 a 9%
    {
        hpBar[4].enabled = false;
        hpBar[3].enabled = false;
        hpBar[2].enabled = false;
        hpBar[1].enabled = false;
        hpBar[0].enabled = false;
    }
}

```

Chamamos a função no Update

```
controleBarraVida();//chama a função
```

Testem o jogo.

Vamos arrumar agora a formatação do número do gold em nossa tela, lembrando que por padrão americano ele está substituindo o ponto pela virgula. Uma das formas de conseguir arrumar esse problema é alterando o caractere da virgula pelo ponto. Para isso abram o `_GameController` na pasta `Scripts`. Criem uma variável para receber o valor do gold nosso e em seguida troquem a “,” pelo “.” Da seguinte forma:

```
void Update () {  
    string s = gold.ToString();//passando o valor do gold para a  
    variavel  
  
    goldTxt.text = s.Replace(",", ".");//alterando as virgulas para pontos  
    no display do gold  
}  
  
Testem o jogo.
```


7 SISTEMA DE ALERTA DE INTERAÇÃO

Agora vamos criar um alerta que quando o personagem estiver em contato com algum objeto que possa sofrer interação, que esse alerta apareça, indicando assim, possível interação com o objeto.

Importem a imagem do balão para a pasta Sprites, Sprite Mode (Multiple) e Filter Mode (Point no Filter), apliquem, vá no Sprite editor e corte para o tamanho 32 alturas por 32 largura

Figura 92 - Imagens Alerta

Crie um objeto vazio chamado alertBaloon, adicione o componente Sprite Renderer, arraste no Sprite a primeira imagem do balão, arraste o objeto para dentro do player e posicione o balão em cima da cabeça do nosso personagem:

Figura 93 - Balão Player

Vamos agora criar a animação do balão na cabeça do player, para isso vamos em controller (dentro de animation), botão direito e Create / Animation Controller o chame de ballon.

No objeto alertBallon, adicione o componente Animator, em controller arraste o animation controller que acabamos de criar (ballon), agora abra o Animation (Window / Animation / Animation), crie um novo animation dentro da pasta Assets / Animations e o chame de alert.

Agora vamos seleccionar os sprites do balão e arrastar para a animação, mude o tempo para 8:

Figura 94 - Timeline AlertBallon

Abra o player Script declarem uma variável :

```
public GameObject balaoAlerta; //objeto com o balão de alerta de interação
```

Em seguida arraste para dentro do Balão Alerta o objeto do alertBallon:

Figura 95 - Objeto Balão Alerta

Volte para o script e mude o void interagir para que quando em contato com um objeto interativo ele dispara o alerta, caso contrários ele deixa invisível o balão:

Figura 96 - Script AlertBallon

```

void Interagir()
{
    RaycastHit2D hit = Physics2D.Raycast(hand.position, dir, 0.1f, interacao); //cria uma linha invisível de comprimento 0.1 para interação a objetos na fase
    Debug.DrawRay(hand.position, dir * 0.1f, Color.red); //cria uma linha visual do mesmo tamanho da invisível apenas para teste

    if (hit == true)
    {
        objetoIntecao = hit.collider.gameObject;
        balaoAlerta.SetActive(true); //deixa visível o balão de interação
    }
    else
    {
        objetoIntecao = null;
        balaoAlerta.SetActive(false); //deixa invisível o balão de interação
    }
}

```

Teste o jogo.

7.1 CORREÇÃO DE LOOT

Uma coisa que precisamos arrumar é o nosso baú, pois ele gera loot apenas uma vez, sempre que o personagem passar por ele vai ficar o alerta, mesmo que já tenhamos aberto o baú e se um inimigo estiver perto, tem a chance de abrímos o baú e não atacarmos o inimigo, então para garantir que tudo ocorra bem, vamos fazer com que o baú possa ser aberto uma única vez e que após aberto o alerta saia da cabeça do nosso personagem.

Abram o script do chest e deixem assim:

```

public void interacao()//função para interagir com objetos
{
    if (gerouLoot == false)
    {
        gerouLoot = true;
        StartCoroutine("gerarLoot"); //chama a função de gerarLoot
        spriteRenderer.sprite = imagemObjeto[1]; //muda a imagem
        para a de um baú aberto
        GetComponent<Collider2D>().enabled = false; //desativa a
        camada de interação
    }
}

```

Reparem que desativamos a camada de colisão na ultima linha justamente para que o nosso player não possa mais interagir com esse objeto, vale lembrar que estamos fazendo isso apenas pelo baú só gerar loot uma vez, caso fosse um botão ou alavanca no jogo, provavelmente teríamos de deixa-lo interagir diversas vezes.

Testem o jogo.

7.2 TRANSIÇÃO DE PERSONAGEM

Agora vamos criar um sistema de transição do nosso personagem, coloquem uma porta no cenário e outra na caverna ou outra área fechada para que o personagem tenha acesso a ela, mude a Sorting Layer dentro do Sprite Renderer das duas portas para fundo e a order in layer para 1, garantindo que a porta fique atrás do player mas na frente do cenário de fundo, crie uma tag chamada door e atribua as duas portas essa tag. Renomeie uma porta para door A1 e a outra para door A2 apenas para ficar mais fácil a organização das portas.

Figura 97 - Sprite Renderer

Caso haja necessidade mude a scale das portas para que o tamanho delas fique compatível com o do personagem.

Figura 98 - Portas na Fase

Adicione um box collider e coloque na layer interação. Nesse momento se nosso player chegar perto da porta, ela exibirá o alerta, se necessário mude a área de colisão para ajustar onde pode ocorrer a interação com a porta.

7.3 SCRIPT PORTAS

Crie um novo script na pasta dos scripts chamado door, arraste ele para as duas portas e digite:

```
public class door : MonoBehaviour
{
    public Transform tplayer; //localização do player para retorno na
cena
    public Transform destino; //localização da outra porta

    public void interacao()
    {
        tplayer.position = destino.position;
    }
}
```

E no script do player procure a condição de interação e adicione o if marcado abaixo:

Figura 99 - Script Porta

```
if (Input.GetButtonDown("Fire1") && v >= 0 && attacking == false && Grounded == true && objetoIntecao != null) //caso o
{
    if (objetoIntecao.tag == "door")
    {
        objetoIntecao.GetComponent<door>().tplayer = this.transform;
    }
    objetoIntecao.SendMessage("interacao", SendMessageOptions.DontRequireReceiver); //chama o método de interação e se n
}
```

Arraste para dentro do destino da door A1 o objeto door A2 e vice-versa. Crie um objeto vazio na cena chamado Portas na posição x, y e z em 0 (zero) e arraste para dentro dele as duas portas.

Figura 100 - Objetos Portas

Agora sempre que o player interagir com a porta ele vai mudar sua localização para a próxima porta de destino e poder voltar.

Lembrando que é uma forma um pouco mais difícil que a utilizada para fazer o retorno do player caso caia da plataforma, mas foi apresentada aqui apenas para mostrar a variedade de formas de se realizar a mesma coisa.

Em seguida vamos fazer um sistema de que quando o player usar uma porta, a tela escureça e clareie novamente para esconder esse teletransporte do personagem pelo jogo.

7.4 SISTEMA DE TRANSIÇÃO

Agora vamos criar a transição para nossa aplicação, quando entrar em uma porta ela executará o efeito de transição para não ficar aparecendo um teletransporte do Goku, para isso crie um novo Canvas (botão direito em Hierarchy / UI / Canvas) chame de HUD_Transicao, arraste lá para cima abaixo apenas da Câmera:

Figura 101 - Hud Transição

Nas propriedades dele mude o canvas para ficar como a imagem abaixo:

Figura 102 - Canvas

E no Canvas Scale (Script)

Figura 103 - Canvas Script

Dentro do HUD_Transicao, crie uma imagem (botão direito / UI / Image) e chame ela de fume:

Figura 104 - Objeto Fume

Em Rect Transform e em image (Script) deixe assim:

Figura 105 - Rect Transform

Desmarque dentro do HUD_Transicao o Graphic Raycaster (Script)

Figura 106 - Script Raycaster

Na pasta de Script crie um novo chamado fade, arraste ele para o HUD_Transicao, no começo dele digite:

```
using UnityEngine.UI;
```

Em seguida, digite isso:

```
public class fade : MonoBehaviour
{
    public GameObject painelFume;
    public Image fume;
    public Color[] corTransicao;
    public float step;
}
```

Salve e mude as propriedades do script para isso:

Figura 107 - Fade

Volta para o script e digite após as variáveis criadas:

```
public void fadeIn()
{
    painelFume.SetActive(true);
    StartCoroutine("fadeI");
}

public void fadeOut()
{
    StartCoroutine("fadeO");
}

IEnumerator fadeIn()
{
    for (float i = 0; i <= 1; i += step)
    {
        fume.color = Color.Lerp(corTransicao[0], corTransicao[1], i);
        yield return new WaitForEndOfFrame();
    }
}

IEnumerator fadeO()
{
    for (float i = 0; i <= 1; i += step)
    {
        fume.color = Color.Lerp(corTransicao[1], corTransicao[0], i);
        yield return new WaitForEndOfFrame();
    }

    painelFume.SetActive(false);
}
```

Em seguida abra o _GameController e declare a variável:

```
private fade fade; //objeto de transição
```

No void Start :

```
fade = FindObjectOfType(typeof(fade)) as fade; //inicia o objeto

fade.fadeOut(); //chama a transição de saída para clarear a tela e
desativar a tela preta
```

Testem o jogo, se tudo deu certo é para clarear a tela e começar o jogo.

7.5 – ANIMAÇÃO PORTA

Em seguida vamos amarrar essa animação as portas, abra o script da porta e vamos fazer quase a mesma coisa que antes, declare a variável private e no void Start():

```
private fade fade;

void Start()
{
    fade = FindObjectOfType(typeof(fade)) as fade; //inicia o objeto
}
```

Agora vamos criar uma corrotina e mudar a função para chamar a corrotina:

```
public void interacao()
{
    StartCoroutine("acionarPorta");
}

IEnumerator acionarPorta()
{
    fade.fadeIn();
    yield return new WaitForSeconds(1); //espera
    até estar quase completamente claro para depois mudar a localização do
    player

    tplayer.gameObject.SetActive(false); //desativa o player para não
    ver ele antes de terminar a animação
```

```

tplayer.position = destino.position;
yield return new WaitForSeconds(0.5f);
tplayer.gameObject.SetActive(true); //reativa o player
fade.fadeOut();
}

```

Teste o jogo

Agora vamos mudar a iluminação dos nossos ambientes fechados como a caverna, para isso primeiro precisamos apagar o chão e as paredes da nossa caverna.

Figura 108 - Iluminação cenário

Em seguida vamos criar 2 tilemaps novos, então botão direito em cima de Grid, 2D Objects, Tilemap chame ela de paredeLuz e redesenhe nessa camada as paredes da caverna (não esqueça de trocar o Sorting Layer dela para fundo):

Figura 109 - Fundo Caverna

Agora vamos criar outro Tilemap e chama-lo de plataformaLuz, adicione o componente TileMap Collider 2D, e redesenhe as paredes da caverna:

Figura 110 - chão caverna

Em seguida crie uma pasta nos assets chamada Material 2D, dentro dela crie dois materiais, (botão direito Create / Material) e chame ele de Luz2D e 2DPadrao;

No 2DPadrao vá em Shade e Sprite / Default

Figura 111 - 2D Padrão

E no Luz2D vá em Shade e Sprite / Diffuse

Figura 112 - Luz 2D

Agora selecione os dois Grids que criamos e em Material arraste o Luz2D para ele:

Figura 113 - Material Luz2D

Em seguida faça o mesmo com o portal dentro da caverna e apague todas as tochas, deixe apenas uma para trabalharmos.

Figura 114 - Caverna e Tochas

7.6 LUZ NAS TOCHAS

Agora vamos adicionar a luz das tochas e a intensidade delas, para isso selecione o fogo da tocha, botão direito, light , point light.

Figura 115 - Luz na Tocha

Depois disso, vá mudando a posição em Z do ponto de luz até ficar do seu agrado a iluminação. Agora vá em Window -> Rendering -> Lighting Settings

Figura 116 - Configurações de luminosidade

Na opção ambiente Color reparem que é um cinza, o que significa que mesmo sem luz a sala não será completamente escura, então vamos mudar essa opção de cor para o preto.

Figura 117 - Ambiente Color

Caso ainda assim a luz esteja iluminando uma área muito grande, basta diminuirmos a sua área.

Figura 118 - tochas com luz

Terminando os ajustes, só precisamos duplicar a tocha e preencher a sala com ela.

Figura 119 - cenário com tochas

E para terminar precisamos fazer com que quando nosso personagem vá para a caverna que ele também seja afetado pela luz. Abra o script da porta e declare as variáveis:

```
public bool escuro; //indica se a porta vai levar para um local escuro
public Material luz2d, padrao2d; //carrega os materiais sensíveis a luz e padrão
```

Em seguida clique em cada uma das portas do seu jogo e arraste os materiais de luz e padrão para eles.

Figura 120 - Adicionando o Script as tochas

E apenas marque escuro em portas que vão te levar a locais escuros no mapa.

Agora, embaixo do código que desativamos o player adicionamos uma condição para ver se é um ambiente escuro ou não, se sim informamos que nosso player tem que ser sensível a luz também, se não, deixamos ele com o padrão de luz.

```
IEnumerator acionarPorta()
{
    fade.fadeIn();
    yield return new WaitForSeconds(1); //espera
    //até estar quase completamente claro para depois mudar a localização do
    //player
    tplayer.gameObject.SetActive(false); //desativa o player para não
    //ver ele antes de terminar a animação

    switch (escuro)
    {
        case true:
            tplayer.gameObject.GetComponent<SpriteRenderer>().material = luz2d; //deixa
            //o personagem sensível a luz
            break;
```

```

        case false:

tplayer.gameObject.GetComponent<SpriteRenderer>().material = padrao2d;
//deixa o personagem com luz padrão
        break;
    }

    tplayer.position = destino.position;
    yield return new WaitForSeconds(0.5f);
    tplayer.gameObject.SetActive(true); //reativa o player
    fade.fadeOut();
}

```

Teste o jogo.

7.7 ANIMAÇÃO DO FOGO

Vamos agora melhorar a animação do fogo e tornar nossa arma também sensível a luz, para isso criem dentro da pasta Animatios > Controller um new Animation Controller chamado luzTocha em seguida selecione o ponto de luz da tocha e adicione um componente Animator a ele.

Arraste o luzTocha para dentro do Controller do Animator do Point Light.

Abra a aba Animation (com o ponto de luz selecionado) clique no botão Create e salve a animação na pasta Assets> Animations com o nome luzTocha.

Clique no botão de gravar no Animation, repare que a linha do tempo ficou vermelha, em seguida clique no 0:02 mude a intensidade da luz para 0.8 na propriedade Intensity no objeto Point Light e depois, clique em 0:04 e mude novamente a intensidade para 1.

Figura 121 - Time Line animação Tocha

Figura 122 - Point Light

Agora para dar certo nas outras tochas, basta selecionarmos os outros Points Light da cena, adicionarmos o componente Animator e arrastarmos para dentro dele o Controller luzTocha.

Testem o jogo e caso haja a necessidade de mudanças, faça-as.

7.8 ADICIONANDO A LUMINOSIDADE AS ARMAS

Agora arrumando a arma, abram o playerScript no começo das declarações de variáveis digitem:

```
private SpriteRenderer sRender;
```

No void start vamos iniciar:

```
sRender = GetComponent<SpriteRenderer>();
```

Em seguida, no fim do código digitem:

```
public void changeMaterial(Material novoMaterial)
{
```

```

sRender.material = novoMaterial; //muda o material do player
(sensível a luz ou não)

foreach (GameObject o in armas) //muda o material de todas as
armas (sensível a luz ou não)
{
    o.GetComponent<SpriteRenderer>().material = novoMaterial;
}
}

```

Agora vamos mudar como recebemos o nosso material, então abram o script do door e o deixem assim:

Primeiro apague tudo que tenha o Tplayer

Declare:

```
private playerScript playerScript;
```

Inicie:

```
playerScript = FindObjectOfType(typeof(playerScript)) as
playerScript; //inicia o objeto
```

deixe o acionarPorta() assim agora:

```
IEnumerator acionarPorta()
{
    fade.fadeIn();
    yield return new WaitWhile(() => fade.fume.color.a < 0.9f); //espera
até estar quase completamente claro para depois mudar a localização do
player

    playerScript.gameObject.SetActive(false);

    switch (escuro)
    {
        case true:
            playerScript.changeMaterial(luz2d);
            break;
    }
}

```

```

    case false:
        playerScript.changeMaterial(padrao2d);
        break;
    }

    playerScript.transform.position = destino.position;
    yield return new WaitForSeconds(0.5f);
    playerScript.gameObject.SetActive(true); //reativa o player
    fade.fadeOut();
}

```

E por último no playerScript apague esse código:

```

    if (objetoIntecao.tag == "door")
    {
        objetoIntecao.GetComponent<door>().tplayer =
this.transform;
    }

```

Agora teste o jogo, reparem que nem a arma está brilhando no escuro mais e as tochas estão trepidando na fase toda.

8 CORRIGINDO ALGUMAS FUNCIONALIDADES ATUAIS

Vamos aproveitar e rever algumas funcionalidades já existentes durante o decorrer do projeto e melhorar elas antes de adicionar novos mecanismos e elementos ao projeto atual.

8.1 SISTEMA DE CÂMERA CINEMACHINE

Vamos aproveitar e melhorar a forma como a nossa câmera acompanha o cenário e o personagem, primeiro baixe o Cinemachine (window > asset store), depois do download não esqueça de importar para seu unity project (basta ir clicando em import).

Figura 123 - logo Cinemachine

Se tudo der certo aparecerá do lado do window a cinemachine

Figura 124 - aba cinemachine

Cliquem em Cinemachine e escolham a opção Create 2D Camera, chame ela de CamController e arraste ela pra baixo do main câmera:

Figura 125 - objeto cinemachine

Em seguida arraste o objeto player para dentro do Follow nas propriedades da CamController e altere o valor do orthographic size para 3:

Figura 126 - propriedades do Cinemachine

Rode o jogo, repare que agora seu personagem se move e depois que ele se move que a câmera acompanha ele, isso é por causa do ajuste que está na propriedade X Damping e Y Damping dentro do Body, então vamos mudar os valores de 1 para 0.1:

Figura 127 - Z Damping

Teste novamente.

Repare que agora seu personagem tem que mover um certo espaço para depois a câmera seguir ele, isso acontece por causa da Dead Zone que é

esse quadrado abaixo, onde o personagem é o ponto amarelo e a dead zone é o quadrado em volta dele, só quando o personagem chega a uma extremidade da dead zone que ele a obriga a mover.

Figura 128 - Área da Câmera

Podemos alterar esse espaço as propriedades da câmera:

Figura 129 - Dead Zone

Basta mudar esse tamanho com base nas suas necessidades.

Outra coisa legal que a câmera pode fazer é programar uma área para limitar o que a câmera pode exibir na cena durante a execução do jogo ou não pode exibir, um exemplo disso são as nossas beiradas no jogo, quando estamos na caverna ele exibe o chão da caverna, mas também mostra uma área não desenhada em nosso jogo com o fundo todo azul:

Figura 130 - Cenário com dead Zone

Então podemos programar para só exibir até as beiradas do nosso cenário, ocultando coisas que não queremos que sejam mostradas ou por ser o fim da fase ou por ser uma parte secreta que o jogador no momento não tem acesso a ver.

Para isso vamos criar um objeto vazio e chama-lo de LimitCamera e coloca-lo abaixo da CamController e adicionar a esse LimitCamera o componente Polygon Collider:

Figura 131 - Polygon Collider 2D

Vamos em Edit Collider segure Ctrl do teclado e clique com o mouse em um ponto para apagar e manter apenas 4, agora basta ajustarmos cada um para uma extremidade do nosso cenário:

Figura 132 - Cenário com Cinemachine

Muito provavelmente sua marcação ficará torta com os valores todos diferentes, então para arrumar isso vamos copiando um e colando no outro até deixar todos retos, não esqueça de deixar como Is Trigger para que não possa interagir com nosso personagem:

Figura 133 - Is Trigger no Polygon Collider

Agora só precisamos ir até a CamController, lá embaixo vai achar o Extensions, escolha a CineMachine Confiner e basta arrastarmos para dentro de Bounding Shape 2D o objeto LimitCamera que ele já ajustará nossa câmera para os limites estipulados e não exibirá mais nada depois desse ponto:

Figura 134 - Script Cinemachine

Posso estar bem na beirada da cena que ele não me mostra nada depois daquele ponto, bem do jeito que queríamos:

Figura 135 - Cenário com Cinemachine

8.2 IMPLEMENTANDO NOVAS ARMAS

Hoje vamos adicionar ao material do jogo mais armas a serem utilizadas pelo personagem, primeiramente importe as quatro imagens para dentro da pasta Armas em seu projeto, mude o filter mode e o Sprite Mode.

Em seguida recortem no tamanho 240 por 240 todas as imagens.

Agora precisamos criar um vetor para armazenar as imagens desejadas das armas, para isso abra o script do _GameController e crie 3 vetores para as 3 posições que a arma pode aparecer:

```
public Sprite[] spriteArmas1; //vetor com as imagens de todas as armas na posição 1
```

```
public Sprite[] spriteArmas2; //vetor com as imagens de todas as armas na posição 2
```

```
public Sprite[] spriteArmas3; //vetor com as imagens de todas as armas na posição 3
```

Em seguida salve e no script coloque 3 posições iniciais, com o passar do tempo vamos cadastrando as outras armas:

Figura 136 - Vetor com o tamanho das armas

Agora, vamos arrastar as imagens das espadas para cada uma das posições:

Figura 137 - Vetor com as imagens das armas

Abra o Script do playerScript localize a parte onde criamos as variáveis das nossas armas e declare:

```
public int idArma; //armazena o id da arma
public int idArmaAtual; //armazena o id da arma atual
```

Agora, vamos criar uma função, então no final do seu código digite:

```
public void trocarArma(int id)
{
    idArma = id;
    armas[0].GetComponent<SpriteRenderer>().sprite =
_GameController.spriteArmas1[idArma];
    armas[1].GetComponent<SpriteRenderer>().sprite =
_GameController.spriteArmas2[idArma];
    armas[2].GetComponent<SpriteRenderer>().sprite =
_GameController.spriteArmas3[idArma];
}
```

```

        idArmaAtual = idArma;
    }

```

No void Start:

```
trocarArma(idArma);
```

Finalmente teste o jogo e mude a arma. Só precisamos agora de melhorar fazendo com que a troca realize durante o jogo. Então vamos criar uma segunda função em cima do trocarArma:

```

private void LateUpdate()
{
    if (idArma != idArmaAtual)
    {
        trocarArma(idArma);
    }
}

```

Teste o jogo e veja se muda a arma mesmo com o jogo emulado. Façam agora para as demais armas.

Agora vamos adicionar mais alguns dados sobre nossas armas como o dano mínimo e dano máximo delas além de mais algumas informações que nos serão úteis no futuro.

Primeiro abram o scrip do _GameController e cadastrem as variáveis:

```

public int[] danoMinArma; //dano minimo de cada arma cadastrada
public int[] danoMaxArma; //dano máximo de cada arma cadastrada

```

Em seguida, adicione o valor de dano das armas no `_GameController` (objeto):

Figura 138 - Vetor com o dano da arma

Continuando, no script `armaInfo` deixe-o assim:

```
public int danoMin; //valor do dano mínimo que causa ao colidir com um
inimigo
public int danoMax; //valor do dano máximo que causa ao colidir com
um inimigo
```

No `playerScript`, adicione na função `trocarArma`:

```
public void trocarArma(int id)
{
    idArma = id;
    armas[0].GetComponent<SpriteRenderer>().sprite =
_GameController.spriteArmas1[idArma];

    armaInfo tempInfoArma = armas[0].GetComponent<armaInfo>();
    tempInfoArma = armas[0].GetComponent<armaInfo>();
    tempInfoArma.danoMin = _GameController.danoMinArma[idArma];
    tempInfoArma.danoMax =
_GameController.danoMaxArma[idArma];
```



```

        armas[1].GetComponent<SpriteRenderer>().sprite =
        _GameController.spriteArmas2[idArma];

        tempInfoArma = armas[1].GetComponent<armaInfo>();
        tempInfoArma.danoMin = _GameController.danoMinArma[idArma];
        tempInfoArma.danoMax =
        _GameController.danoMaxArma[idArma];

        armas[2].GetComponent<SpriteRenderer>().sprite =
        _GameController.spriteArmas3[idArma];

        tempInfoArma = armas[1].GetComponent<armaInfo>();
        tempInfoArma.danoMin = _GameController.danoMinArma[idArma];
        tempInfoArma.danoMax =
        _GameController.danoMaxArma[idArma];

        idArmaAtual = idArma;
    }

```

E no script controleDanoInimigo, deixe assim:

```

if (getHit == false)
{
    getHit = true;
    barrasVida.SetActive(true);

    armaInfo infoArma =
    col.gameObject.GetComponent<armaInfo>();

    int danoTomado = Random.Range(infoArma.danoMin,
    infoArma.danoMax);

```

Agora teste o jogo;

Quando mudar o idArma no player, as armas nas posições A, B e C tem que ter mudado os seus valores, bem como o dano sofrido pelo inimigo.

Aproveitando que já estamos trabalhando nisso, vamos implementar no script _GameController 2 novas variáveis que vão nos ajudar muito no futuro:

```
public string[] nomeArma; //vetor com o nome de todas as armas
public int[] custoArma; //vetor com o valor de compra de todas as armas
```

faça o mesmo para todas as armas cadastradas, adicionando um nome e um valor a elas:

Figura 139 - vetor com descrição das armas

8.3 TRABALHANDO COM MÚLTIPLAS CENAS

Vamos agora trabalhar com diversas Cenas no nosso jogo, para isso temos duas opções: ou criamos uma cena nova do zero, onde recriaríamos todos os elementos da nova cena, inimigos, objetos, animações, personagem, tudo de novo ou podemos duplicar uma cena já existente e modificar o que acharmos necessário. Vou duplicar a atual e mudar algumas coisas e chama-la de cena2.

Em seguida vamos adicionar na cena atual, um portal para fazer a transição entre as cenas:

Figura 140 - Portal

Adicione um Box Collider ao seu objeto, marque o isTrigger, mude a layer dele para Interacao, crie um script chamado mudarCena na pasta de scripts e adicione ele ao seu objeto, em seguida digite nos usings:

```
using UnityEngine.SceneManagement;
```

Em seguida declare a variável:

```
public string cenaDestino; //recebe o nome da cena que será chamada
```

E por ultimo chame a função interação:

```
public void interacao()
{
    SceneManager.LoadScene(cenaDestino);
}
```

Teste o jogo digitando Cena2 no portal.

Figura 141 - Cena 2 Script

Repare que deu erro pois a nossa cena não está adicionada as cenas do jogo, para isso basta fazermos o que o erro está pedindo.

Figura 142 - Erro

Estão vá em file -> Build Settings em seguida

Figura 143 - Build Settings

Se clicar em add Open Scene ele irá pegar e adicionar a cena aberta sua, para adicionar as demais basta arrastar para dentro dessa tela e ordenar em primeiro lugar a sua cena principal, se tiver uma cena de menu, é aqui que falamos que ele será a primeira cena a carregar. Vou renomear minha cena principal para Cena1 para manter uma ordem já.

Agora sim, testem o jogo e vejam se está dando certo.

Reparem que quando o player vai para a próxima Cena ele não armazena as configurações que já implementamos, como skin do player, arma, gold, vida, e por ai vai. Isso vamos resolver daqui a pouco, primeiro vamos criar

um efeito de transição como fizemos com a porta para quando mudar de cenário, apagar a tela e clarear depois quando estivermos na outra cena;

Então abra o script do mudarCena, em declaração de variáveis:

```
private fade fade;
```

No void start:

```
fade = FindObjectOfType(typeof(fade)) as fade;
```

Crie uma corrotina:

```
IEnumerator mudarcena()
{
    fade.fadeIn();
    yield return new WaitWhile(() => fade.fume.color.a < 0.9f);
    SceneManager.LoadScene(cenaDestino);
}
```

No public void interacao:

```
public void interacao()
{
    StartCoroutine("mudarcena");
}
```

Teste o jogo. Já está funcionando, mas a animação está muito rápida, podemos mudar isso, então vamos abrir o script do fade e no fade o adicione:

```
Enumerator fadeO()
{
    yield return new WaitForSeconds(0.5f);
    for (float i = 0; i <= 1; i += step)
```

E no script da door na corrotina acionarPorta, remova a linha:

```
yield return new WaitForSeconds(0.5f);
```

Teste o jogo. Agora o jogo está fazendo uma animação interessante.

Vamos agora começar a salvar as informações do usuário de uma cena para a outra como a vida, o gold, itens, armas equipadas e por ai vai. Primeiro abra o script do _GameController e digite no void Start:

```
DontDestroyOnLoad(this.gameObject); //faz com que não destrua o
objeto _GameController durante o load de Cenas
```

Agora vá na cena2 e apague o _GameController dele, caso contrário quando o personagem for para a segunda cena, criará 2 _GameController e o jogo não saberá a qual responder.

Em seguida vamos mudar a animação de transição pois dessa maneira, quando mudar da cena 1 para a 2, não será iniciado o fade out e a tela sempre ficará preta. Abra o script do fade e crie o void start:

```
private void Start()
{
    StartCoroutine("fadeO");
}
```

Agora temos que remover do _GameController o fade out então apague essas linhas do código:

```
private fade fade; //objeto de transição

fade = FindObjectOfType(typeof(fade)) as fade; //inicia o objeto
fade.fadeOut(); //chama a transição de saída para clarear a tela e
desativar a tela preta
```

Teste o jogo.

Repare que ainda não está armazenando o gold em jogo, para isso arraste o HUD para dentro do _GameController da cena 1 e apague o HUD da cena 2. Teste o jogo.

Agora vamos salvar as outras informações, abram o script _GameController e criem as variáveis para armazenar as Skins do personagem:

```
public string[] nomePersonagem;  
public string[] spriteSheetName;
```

Em seguida cadastre as skins, em nome do personagem escolha qualquer nome e na SpriteSheet Name o nome do Resource dele:

Figura 144 - Nome das Skins

Nome Personagem	
Size	10
Element 0	Connan
Element 1	Apollo
Element 2	Kaos
Element 3	
Element 4	
Element 5	
Element 6	
Element 7	
Element 8	
Element 9	

Sprite Sheet Name	
Size	10
Element 0	Barbarian1
Element 1	Barbarian2
Element 2	Dwarf 1
Element 3	Dwarf 2
Element 4	Elf Female
Element 5	Elf Male
Element 6	Knight
Element 7	Peasant
Element 8	Soldier
Element 9	Wizard 1

Crie no Script do _GameController as variáveis:

```
public int idPersonagem;  
public int vidaMaxima;  
public int idArma;
```

Agora, no Script do Player, no void Start, digite, para inicializar os valores digitados no _GameController:

```
vidaMax = _GameController.vidaMaxima; //inicia o valor da vida
máxima
idArma = _GameController.idArma; //inicia qual arma o player está
usando
```

E no script do ReSkin, inicie o _GameController

```
private _GameController _GameController;
```

No void Start dp Reskin:

```
_GameController = FindObjectOfType(typeof(_GameController)) as
_GameController; //iniciamos a variável ao objeto externo
```

```
spriteSheetName =
_GameController.spriteSheetName[_GameController.idPersonagem]; //carrega a
skin do player
```

Digitem o ID do personagem, vida e ID da Arma e testem o jogo.

Figura 145 - ID ARMA

Element 9	Wizard 1
Id Personagem	0
Vida Maxima	100
Id Arma	1

Só temos um problema, o inimigo também está mudando a Skin junto pois ambos usam o mesmo script, para corrigir isso precisamos criar uma variável booleana que diferencia inimigos de jogadores, então criem no Script do ReSkin a variável:


```
public bool isPlayer; //indica se é um jogador ou inimigo
```

No void Start, onde carregava a Skin criamos uma condição:

```
if (isPlayer)
{
    spriteSheetName =
    _GameController.spriteSheetName[_GameController.idPersonagem]; //carrega a
    skin do player
}
```

E no jogo, marcamos que é player na cena um e 2 o nosso personagem:

Figura 146 - Re Skin

Agora testem o jogo e vejam se está tudo certo.

8.4 IMPLEMENTANDO MELHORIAS NA TROCA DE SKIN

O problema desse jeito é que quando possuímos muitas opções, fica difícil ficar digitando corretamente nome, por nome como fizemos para cadastrar as 10 skins de personagens que possuímos, então vamos mudar nosso projeto fazendo-o mais fácil nesses momentos de muitas opções.

Vá no Script do _GameController e troque o tipo da variável spriteSheetName de string para texture:

```
public Texture[] spriteSheetName;
```

Em seguida, vamos no Script do ReSkin e adicione a propriedade .name ao final da linha com erro:

```
spriteSheetName =
_GameController.spriteSheetName[_GameController.idPesonagem].name;//car
rega a skin do player
```

Com isso, Basta segurar e arrastar para os locais onde antes tivemos de digitar o nome:

Figura 147 - Skin por Objeto

Agora vamos fazer com que possamos mudar de Skin durante o funcionamento do jogo, para isso criem no _GameController a variável:

```
public int idPersonagemAtual;
```

E no ReSkin dentro do LateUpdate:

```
if (isPlayer)
{
    if (_GameController.idPesonagem !=
_GameController.idPersonagemAtual)
    {
```

```

        spriteSheetName =
_GameController.spriteSheetName[_GameController.idPesonagem].name;//car
rega a skin do player
        _GameController.idPersonagemAtual =
_GameController.idPesonagem;//muda o valor para igualar as variáveis
    }
}

    if (loadedSpriteSheetName != spriteSheetName)//caso haja
mudança de personagem
    {
        loadSpriteSheet();
    }

```

Agora teste o jogo e veja se está mudando a Skin, durante a execução do mesmo.

8.5 IMPLEMENTANDO ARMAS A DISTÂNCIA

Nesse momento vamos implementar o ataque dos personagens a distância como os arqueiros e magos, para começarmos adicione as imagens das armas ao Unity junto com as demais na pasta Armas em Sprites:

Figura 148 - armas a distância

Defina elas como multiple, e Point No Filter, recorte as imagens 240 por 240.

Agora podemos começar a trabalhar com as novas armas, primeiro duplique o grupo da arma dentro do personagem e renomeie como Arcos:

Figura 149 - Arcos

Selecione as 3 Letras e Remova o Circle Collider e o Script Armas Info.

Em seguida adicione ao projeto as imagens das flechas e das magias do mago dentro da pasta Armas:

Figura 150 - Pasta Armas

Não esqueça de mudar as propriedades das imagens para Point No Filter.

Agora mude o Sprite do player para o de um arqueiro, e o sprite da arma para a primeira posição do arco:

Figura 151 - Elfa com Arco

Crie um objeto vazio filho de A em Arco chamado Flecha:

Figura 152 - Objeto Flecha

Adicione um Sprite Render muted o Order in Layer para 2 e arraste para ele o sprite da flecha:

Figura 153 - Flecha Grande no Arco

Note que a flecha é muito grande com relação ao player, então vamos mudar nas propriedades da imagem o Pixels per Unit de 100 para 200, dê um apply e agora nossa flecha está em um tamanho menor:

Figura 154 - Flecha pequena com Arco

Pronto, agora basta posicionar a flecha em cada posição do personagem A e B:

Figura 155 - Alinhando com a mão

Não esqueça de ir mudando o sprite do arco e do player para cada posição.

C não precisa pois quando chegar na posição C, criaremos um objeto temporário para ser nossa flecha, então crie um objeto vazio filho da flecha na

posição B chamado spawnFlecha, em seguida arraste o objeto para baixo de C conforme imagem:

Figura 156 - Spawn Flecha

Duplique uma Flecha em B e arraste ela lá para baixo, ela será o nosso Prefab, chame ela de FlechaPrefab, adicione um Rigidbody, troque o Gravity Scale dela para 0 (zero), Freeze Rotation Z, Adicione um Box Collider e edite a área de colisão (pode deixar a área apenas a ponta da flecha), coloque na TAG arma, marque o IS TRIGGER, e adicione o Script do Arma Info a flecha (adicione um valor 1 e 2).

Crie na Pasta dos Prefabs uma pasta chamada Projeteis e coloque a flecha lá dentro.

Figura 157 - Flecha Prefab

E pode apagar ela da Cena, agora vamos fazer o mesmo, mas para o Mago, uma coisa que muda é que o Mago tem 4 posições de Ataque e não 3

como é de costume nosso. Mude o Sprite do personagem para o Mago, duplique o Arco e chame de Staff, na posição A, coloque a primeira magia, e assim por diante até a letra D, apague as flechas e mude o nome de spawnFlecha para SpawnMagia, posicione o SpawnMagia bem no meio da ponta do cajado na posição D do ataque.

Figura 158 Mago

Agora precisamos fazer as animações dos ataques do Mago, então adicione a cena uma magia, não esqueça de mudar para 200 o pixel per unit, chame ela de magiaA, adicione um Rigidbody, troque o Gravity para 0, Freeze Rotation em Z, Circle Collider e edite a área de colisão, Marca como trigger e Tag ARMA, e add Arma Info, dano mínimo 3 Máximo 10. Abra a aba Animation, clique em Create, crie uma pasta chamada Animations dentro da pasta contendo as magias (Wizard Attack VFX) .

Figura 159 - Pasta Animations

Chame de fx1 e pressione salvar, arraste para a time line os sprites da magia e mude a velocidade para 8, ou a que julgar mais adequada. Se ainda assim achar que sua magia está muito grande mude o pixel per unit ou o próprio

scale dela. Tudo pronto, basta salvar a magiaA na pasta do prefab de projéteis e apagar a magiaA da cena.

Faça o mesmo para todas as outras Magias e Flechas.

Em seguida precisamos criar os efeitos de transição das animações então abram o script do Player e criem 2 novos arrays de GameObject como tem o Armas, mas agora para Arcos e Staffs:

```
public GameObject[] armas, arcos, flechaArco, staffs; //carregando as  
armas para o player
```

Agora criamos os valores necessários aos nossos arrays e arrastamos os objetos referentes a cada situação para ele:

Figura 160 - Array com as armas

Com isso feito, vamos trabalhar com o ponto inicial da Magia e da Flecha, para isso criem as seguintes variáveis ainda no Script do player:

```
public GameObject flechaPrefab, magiaPrefab;
public Transform spawnFlecha, spawnMagia;
```

E alimentas as variáveis com os objetos referentes a cada um deles:

Figura 161 - Spawn Armas Prefab

Agora, no Script do Player, no void Start, precisamos desativar os arcos e os Staffs, como fizemos com as armas então:

```
foreach (GameObject o in arcos)
{
    o.SetActive(false);
}
foreach (GameObject o in staffs)
{
    o.SetActive(false);
}
```

Em public void changeMaterial adicionamos a mudança para arco, flechas e cajado:

```
foreach (GameObject o in arcos) //muda o material de todos os
arcos (sensível a luz ou não)
{
    o.GetComponent<SpriteRenderer>().material = novoMaterial;
}
foreach (GameObject o in flechaArco) //muda o material de todas
as flechas dos arcos (sensível a luz ou não)
```

```

    {
        o.GetComponent<SpriteRenderer>().material = novoMaterial;
    }
    foreach (GameObject o in staffs) //muda o material de todos os
cajados (sensível a luz ou não)
    {
        o.GetComponent<SpriteRenderer>().material = novoMaterial;
    }

```

No Script do _GameController crie a seguinte variável:

```

public int[] idClasseArma; //0: Machado, espada e Martelo - 1: Arcos - 2:
Cajados

public Sprite[] spriteArmas4; //vetor com as imagens de todas as armas
na posição 4

```

Figura 162 - Código Vetores Armas

```

public string[] nomeArma; //vetor com o nome de todas as armas
public int[] custoArma; //vetor com o valor de compra de todas as armas
public int[] idClasseArma; //0: Machado, espada e Martelo - 1: Arcos - 2: Cajados
public Sprite[] spriteArmas1; //vetor com as imagens de todas as armas na posição 1
public Sprite[] spriteArmas2; //vetor com as imagens de todas as armas na posição 2
public Sprite[] spriteArmas3; //vetor com as imagens de todas as armas na posição 3
public Sprite[] spriteArmas4; //vetor com as imagens de todas as armas na posição 4
public int[] danoMinArma; //dano mínimo de cada arma cadastrada
public int[] danoMaxArma; //dano máximo de cada arma cadastrada
public string[] nomePersonagem;
public Texture[] spriteSheetName;

```

Com isso podemos cadastrar nossas novas armas, então no objeto _GameController do jogo, podemos agora cadastrar corretamente cada arma nova

Figura 163 - GameController

Já possuímos 3 armas cadastradas, então as 3 atuais possuem o ID zero por serem uma espada, mas vamos mudar o valor de todos os itens para 5 a fim de poder cadastrar pelo menos um arco e um staff novo e alimentem corretamente as informações conforme abaixo:

Figura 164 - Vetores com valores

▼ Nome Arma	
Size	5
Element 0	Espada de Madeira
Element 1	Espada de Ferro
Element 2	Espada Reforçada de Ferro
Element 3	Arco de Madeira
Element 4	Cajado Iniciante
▼ Custo Arma	
Size	5
Element 0	5
Element 1	10
Element 2	20
Element 3	10
Element 4	15
▼ Id Classe Arma	
Size	5
Element 0	0
Element 1	0
Element 2	0
Element 3	1
Element 4	2

Repare que a classe dos 3 primeiros itens é 0 por serem uma espada

Figura 165 - Vetores com Valores

Outra coisa que é bom perceber é que no Sprite Arma 4 só onde tem o cajado é que possuirá arma, o resto só completar normalmente.

Figura 166 - Vetores com danos da arma

Agora no Script do Player Script, possuímos o void trocaArma, vamos mudar algumas coisas, primeiro, após a linha do idArma= id vamos adicionar um Switch para as espadas, machados e martelos:

Figura 167 - Player Script

```
public void trocarArma(int id)
{
    idArma = id;

    switch (_GameController.idClasseArma[idArma])
    {
        case 0:
            armas[0].GetComponent<SpriteRenderer>().sprite = _GameController.spriteArmas1[idArma];

            armaInfo tempInfoArma = armas[0].GetComponent<armaInfo>();
            tempInfoArma = armas[0].GetComponent<armaInfo>();
            tempInfoArma.danoMin = _GameController.danoMinArma[idArma];
            tempInfoArma.danoMax = _GameController.danoMaxArma[idArma];

            armas[1].GetComponent<SpriteRenderer>().sprite = _GameController.spriteArmas2[idArma];

            tempInfoArma = armas[1].GetComponent<armaInfo>();
            tempInfoArma.danoMin = _GameController.danoMinArma[idArma];
            tempInfoArma.danoMax = _GameController.danoMaxArma[idArma];

            armas[2].GetComponent<SpriteRenderer>().sprite = _GameController.spriteArmas3[idArma];

            tempInfoArma = armas[1].GetComponent<armaInfo>();
            tempInfoArma.danoMin = _GameController.danoMinArma[idArma];
            tempInfoArma.danoMax = _GameController.danoMaxArma[idArma];
            break;

        idArmaAtual = idArma;
    }
}
```

Fecha o Switch antes do idArma = armaAtual;

Agora vamos fazer para Arcos e Staffs, então embaixo do break crie o Case 1 e 2:

Figura 168 - Case arma

```
case 1:
    arcos[0].GetComponent<SpriteRenderer>().sprite = _GameController.spriteArmas1[idArma];
    arcos[1].GetComponent<SpriteRenderer>().sprite = _GameController.spriteArmas2[idArma];
    arcos[2].GetComponent<SpriteRenderer>().sprite = _GameController.spriteArmas3[idArma];
    break;
case 2:
    staffs[0].GetComponent<SpriteRenderer>().sprite = _GameController.spriteArmas1[idArma];
    staffs[1].GetComponent<SpriteRenderer>().sprite = _GameController.spriteArmas2[idArma];
    staffs[2].GetComponent<SpriteRenderer>().sprite = _GameController.spriteArmas3[idArma];
    staffs[3].GetComponent<SpriteRenderer>().sprite = _GameController.spriteArmas4[idArma];
    break;
```

E no scrit _gameController crie a variável idArmaAtual também:

```
public int idArma, idArmaAtual;
```

Voltando para o Script do Player agora o trocarArma fica assim:

```
public void trocarArma(int id)
{
    _GameController.idArma = id;
```

E no final deixe assim:

```
_GameController.idArmaAtual = _GameController.idArma;
```

Agora no Late Update deixe-o assim:

```
private void LateUpdate()
{
    if (_GameController.idArma != _GameController.idArmaAtual)
    {
        trocarArma(_GameController.idArma);
    }
}
```

No player Script copie e cole o controleArma e deixe assim, basta trocar arma por arcos e staffs:

```
void controleArco(int id) //função para controle da animação dos arcos
{
    foreach (GameObject o in arcos)
    {
        o.SetActive(false);
    }

    arcos[id].SetActive(true);
}
```

```

void controleStaff(int id) //função para controle da animação dos
cajados
{
    foreach (GameObject o in staffs)
    {
        o.SetActive(false);

        staffs[id].SetActive(true);
    }
}

```

8.6 ANIMAÇÃO DAS ARMAS ARCO E STAFF

Em seguida, na pasta Animations, possui a animação Attack, duplique ela e renomeie para AtaqueFlecha e AttackStaff:

Figura 169 - Animação Cajado

Em Animator, selecione o Attack, em seguida crie uma nova variável do tipo Float e chame ela de idClasseArma:

Figura 170 - Regras de animação

Clique com o botão direito do mouse e escolha a opção Create a New Blend Tree in State, dê um duplo clique e aparecerá essa tela:

Figura 171 - Speed Y

Com o Blend Tree selecionado mude o parâmetro na aba Inspector para idClasseArma e crie 3 Motion File:

Figura 172 - Regras para as 3 armas

Desmarque o Automate Threshold e deixe com valores 0, 1 e 2, em seguida mova para cada campo a animação a qual cada um pertence:

Figura 173 - Blend Tree

Agora no Player Script onde está sendo passado os valores para as animações, temos de passar o novo valor que criamos:

```
playerAnimator.SetBool("grounded", Grounded);
playerAnimator.SetInteger("idAnimation", idAnimation);
playerAnimator.SetFloat("speedY", playerRb.velocity.y);
playerAnimator.SetFloat("idClasseArma",
_GameController.idClasseArma[_GameController.idArmaAtual]);
```

Em seguida selecione o player, abra o Animation, mude para AtackFlecha, clique no evento ControleArma e mude a Function para ControleArco, faça isso nas 3 posições, o mesmo para a AtackStaff mas no lugar coloque ControleStaff só muda uma coisa, o evento que estava no 3, avance para o 4 e crie um novo evento na posição 3:

Figura 174 - Timeline Attack Staff

Testem o jogo, reparem que a animação está funcionando mas ainda não sai a arma da cena ao final do evento.

No player Script copie e cole a função do ataque e mude para arco e Staff conforme abaixo:

```
void atackFlecha(int atk)
{
    if (atk == 0)
    {
        attacking = false;
        arcos[2].SetActive(false);
    }
    if (atk == 1)
    {
        attacking = true;
    }
}

void atackStaff(int atk)
{
    if (atk == 0)
    {
        attacking = false;
        staffs[3].SetActive(false);
    }
    if (atk == 1)
    {
        attacking = true;
    }
}
```

Com o player selecionado volte, na aba Animation, selecione o AttackFlecha e mude onde está Attack para attackFlecha e faça o mesmo para o Staff, em seguida teste novamente as animações.

Só está faltando criar as flechas e as magias quando terminar a animação e fazê-las serem arremessadas durante um período de tempo, pela fase, e depois sumirem.

Figura 175 - Atacando com Arco

Figura 176 - Atacando com Cajado

8.7 CRIANDO OS PROJETEIS DAS ARMAS

Agora precisamos criar os projeteis (magias ou flechas) para isso abra o playerScript, localize o void AttackFlecha que criamos e adicione o código abaixo

(que é quando terminamos de realizar a animação do ataque) pois é nesse momento que vamos instanciar nosso projétil:

```

void attackFlecha(int atk)
{
    if (atk == 0)
    {
        attacking = false;
        arcos[2].SetActive(false);
    }
    if (atk == 1)
    {
        attacking = true;
    }
    if (atk == 2)
    {
        GameObject temPrefab = Instantiate(flechaPrefab,
spawnFlecha.position, spawnFlecha.localRotation); //cria uma flecha a partir do
spawnFlecha
        temPrefab.GetComponent<Rigidbody2D>().velocity = new
Vector2(5, 0); //adiciona uma força a flecha
    }
}

```

Selecione o Player na Hierarchy, abra a aba Animation, selecione o AttackFlecha e na time line, na posição 0:2 crie um novo evento, escolha o AttackFlecha e coloque o valor 2 a ele:

Figura 177 - Função Attack Flecha

Teste o jogo. A animação está correta, o player atira a flecha mas a flecha vai ao infinito e se o player estiver virado para a esquerda, a flecha ainda sairá para a direita.

Antes de arrumar esses problemas vamos já aproveitar e fazer o mesmo para o cajado, em seguida testem o jogo. Copie e cole o código anterior mudando apenas os valores das flechas para as magias:

```
if (atk == 2)
{
    GameObject temPrefab = Instantiate(magiaPrefab,
spawnMagia.position, spawnMagia.localRotation); //cria uma magia a partir do
spawnMagia
    temPrefab.GetComponent<Rigidbody2D>().velocity = new
Vector2(5, 0); //adiciona uma força a magia
}
```

Em seguida mude a animation e adicione um novo evento na posição 0:2 também com o attackStaff com o valor 2:

Figura 178 - Função Attack Staff

Agora só testem o jogo para ver se está tudo certo.

Para destruir o objeto depois de um tempo basta adicionar a linha do Destroy conforme abaixo para os dois projeteis:

```
Destroy(temPrefab, 2); //destroi a magia depois de 2 segundos
```

Teste o jogo, lembrando que podemos colocar a duração diferente para cada projétil e a velocidade também, podemos ter magias ou flechas que vão mais rápido ou que duram mais ou menos do que outras.

8.8 DIREÇÃO DA FLECHA

Última coisa para arrumar é a direção, se lembrarem já fazemos isso quando o nosso player muda de posição temos o flip, que muda a direção de x positivo (quando para a direita) para x negativo (quando para a esquerda) então vamos aproveitar dessa programação feita e multiplicarmos a velocidade por X (sendo ele 1 para a direita e -1 para a esquerda):

```
temPrefab.GetComponent<Rigidbody2D>().velocity = new Vector2(5 *  
dir.x, 0); //adiciona uma força a magia
```

Faça isso para os dois. Testem o jogo.

Agora temos só mais um problema, quando é a flecha, ela sai na direção correta porém ao contrário, então temos que depois de instanciar a flecha, precisamos multiplicar ela por $\text{dir} \cdot X$ do jeito que fizemos anteriormente, assim, quando o player estiver virado para a esquerda, a flecha sairá também com a ponta para a esquerda:

```
temPrefab.transform.localScale = new
Vector3(temPrefab.transform.localScale.x * dir.x,
temPrefab.transform.localScale.y, temPrefab.transform.localScale.z);//rotaciona
a ponta da flecha quando o player estiver virado para a
direitatemPrefab.transform.localScale
```

Teste o jogo.

Tudo perfeito, tudo funcionando, mas ainda tem um porém, o nosso projétil não pode acertar múltiplos alvos (a não ser que vocês assim queiram) então precisamos destruir o projétil se ele não acertar nada pelo tempo que colocamos na regra do Destroy ou, caso haja colisão com algum inimigo, ser destruído imediatamente. Então vamos criar agora um novo script na pasta de Scripts chamado destroyCollision, adicionar ele a todos os nossos prefabs de Projeteis, em seguida editar ele para ficar assim:

```
public LayerMask destruirLayer;//camada que irá fazer o objeto ser
destruido instantaneamente
```

```
void Update()
{
    RaycastHit2D hit = Physics2D.Raycast(transform.position,
Vector2.right, 0.1f, destruirLayer);

    if (hit == true) //se houver colisão com a camada INIMIGO, o
objeto será destruido na mesma hora
    {
```



```

        Destroy(this.gameObject);
    }
}

```

Agora mudem no prefab dos objetos o valor da variável para a camada Inimigo:

Figura 179 - Script Destroy Collision

Teste o jogo.

Última coisa que está faltando é só permitirmos uma quantidade x de vezes de soltar magias ou flechas, caso não tenha no estoque mais flechas ou mana para as magias. Não pode sair mais projeteis.

8.9 CLASSE DE CADA PERSONAGEM

Uma coisa que precisamos arrumar com urgência é que nossos personagens só possuem uma animação de ataque e dependendo do player ou ataca com arma física(espada, machado ou martelo), arco ou cajado. Então quando mudarmos de personagem, a arma tem que ser compatível com a animação do mesmo. Para isso abra o script do `_GameController` e adicione mais 2 variáveis:

```

public int[] idClasse;
public int[] idArmaInicial;

```

Em seguida, no unity, cadastre 10 id classe sendo 0 para os guerreiros 1 para os arqueiros e 2 para o único mago que temos e em Id arma inicial, faça o cadastro dos guerreiros com a espada de madeira, arqueiros com o arco básico e o mago com o cajado inicial conforme a imagem a baixo:

Figura 180 - Id class e arma inicial

▼ Id Classe	
Size	10
Element 0	0
Element 1	0
Element 2	0
Element 3	0
Element 4	1
Element 5	1
Element 6	0
Element 7	0
Element 8	0
Element 9	2

▼ Id Arma Inicial	
Size	10
Element 0	0
Element 1	0
Element 2	0
Element 3	0
Element 4	3
Element 5	3
Element 6	0
Element 7	0
Element 8	0
Element 9	4

Agora no `_GameController` criamos uma função que valida a troca de skin caso mude para um personagem que usa uma arma diferente, a arma também muda junto. Embaixo do `Void Update` crie:

```
public void validaArma()
{
    if (idClasseArma[idArma] != idClasse[idPersonagem])
    {
        idArma = idArmaInicial[idPersonagem];
    }
}
```

Com isso abra o script do `ReSkin` e no void `LateUpdate`, dentro do `if(isPlayer)` chame a função que acabamos de criar:

```
if (isPlayer)
{
    if (_GameController.idPersonagem !=
_GameController.idPersonagemAtual)
```

```

    {
        spriteSheetName =
        _GameController.spriteSheetName[_GameController.idPersonagem].name;//ca
        rrega a skin do player
        _GameController.idPersonagemAtual =
        _GameController.idPersonagem;//muda o valor para igualar as variáveis
    }
    GameController.validaArma();
}

```

Teste o jogo e veja se está mudando a arma conforme muda a skin do personagem.

No player Script onde era IdArma mude para ID no public void trocaArma, apenas nas linhas de código iguais a essa:

```

switch (_GameController.idClasseArma[id])
{
    case 0:
        armas[0].GetComponent<SpriteRenderer>().sprite =
        _GameController.spriteArmas1[id];

```

Agora teste o jogo. Tente mudar o skin de um personagem de arma física e em seguida mude a arma do mesmo tipo que ele pode e depois uma diferente; Cadastre todas as armas e teste.

9 MENU DE SELEÇÃO DE PERSONAGENS

Vamos agora criar um menu para poder selecionar o player inicial, para isso, duplique a cena 2 e chame de Título, em seguida vamos apagar os elementos que não são interessantes para nós no momento deixando apenas:

- vá em MainCamera e remova o Cinemachine Brain
- HUD_Transição (Desativado)
- Grid
- BackGround

Redesenhe o Cenário deixando apenas o Básico:

Figura 181 - Menu de seleção

Adicione um Canvas (botão direito, UI, Canvas), nas propriedades em Canvas Render Mode, deixe Screen Space – Camera, Render Mode arraste a câmera para lá. Em Canvas Scaler: UI Scaler Mode deixe Scale With Screen Size x 1920 y 1080, Match 0.5

Figura 182 - Canvas Scaler

Botão direito em cima do canvas, UI, Button, No Image Script, Source Image selecione o Sprite do primeiro personagem de espada, marque o preserve Aspect e redimensione a sua escolha a imagem do player até achar adequada, apague o Text do Button e duplique ele mudando o sprite para um arqueiro e um Mago.

Figura 183 - Barbaro

Figura 184 - 3 Personagens

Em EventSystem tem o First Selected, arraste o Button 1 para lá, agora selecione os 3 botões, vá em Button Script, deixe marcado a opção Interactable, Em Normal Color deixe a camanda A (transparência) em 100.

Figura 185 - Button

Agora vamos criar um script, então na pasta de Scripts crie um chamado titulo, crie um objeto vazio chamado _GameController e arraste o Script para lá:

Digitem :

```
public void selecionarPersonagem(int idPersonagem)
{
}
}
```

Em seguida selecione os 3 botões, clique no + em On Click():

Figura 186 - On Click

Arraste o _GameController para dentro e na outra opção escolha Titulo
-> selecionarPersonagem, em seguida clique no primeiro botão e deixe com o

valor 1, segundo 5 e terceiro 9 que são os ids deles na nossa lista de personagens que cadastramos.

Figura 187 - Condicional on Click

Agora volta para o Script Titulo e digite dentro da nossa função:

```
using UnityEngine.SceneManagement;

public void selecionarPersonagem(int idPersonagem)
{
    PlayerPrefs.SetInt("idPersonagem", idPersonagem);
    SceneManager.LoadScene("Cena1");
}
```

Em seguida abra o Build Settings em File -> Build Settings arraste o titulo para dentro e deixe-o em primeiro lugar:

Figura 188 - Build Settings Titulo

Feito isso basta irmos no script do _GameController no void Start e pedir para ele receber o valor da variável que acabamos de criar para carregar o player:

```
idPersonagem = PlayerPrefs.GetInt("idPersonagem");//recebe a opção  
escolhida na cena titulo
```

Agora teste o jogo e veja se está vindo corretamente o player selecionado.

9.1 MUDANDO DE ARMA IN GAME

Apenas para ilustrar algumas possibilidades vou mostrar que tem como mudar a arma do personagem durante o jogo sem precisar ficar mudando as configurações manualmente, então abram o script do player e digitem no final do void Update(), antes do playerAnimator:

```
if(Input.GetKeyDown(KeyCode.Alpha1))//ao apertar o botão 1
{
    trocarArma(0);//espada de madeira
}
if (Input.GetKeyDown(KeyCode.Alpha2))//ao apertar o botão 2
{
    trocarArma(1);//espada de ferro
}
if (Input.GetKeyDown(KeyCode.Alpha3))//ao apertar o botão 3
{
    trocarArma(2);//espada de reforçada de ferro
}
```

Testem o jogo, reparem que quando pressionamos as teclas, ele muda a arma, podemos fazer isso para trocar de personagem (Skin) durante o jogo, uma parte que é melhor usar o mago, trocamos para o mago e em outras partes mudamos para um guerreiro ou um arqueiro, tudo dependendo da fase e da estratégia.

9.2 CRIANDO MENUS E SUBMENUS

Agora vamos começar a desenvolver nosso inventário, a primeira coisa que precisamos fazer no nosso jogo é um método que sempre q acessarmos o inventário o jogo deve ser pausado (parar ações de inimigos, movimentos e tudo mais) para isso acessem a cena 1, _GameController e botão direito em cima de HUD -> escolham o UI e Panel, arraste para dentro do Panel o HP BAR e GOLD BAR e renomeie seu Panel para InfoGamePlay:

Figura 189 - InfoGamePlay

Nas propriedades do InfoGamePlay removam a opção RayCast Target (pois não queremos que ele seja clicavel) e em color na camada A, deixe tudo em zero – totalmente transparente.

Figura 190 - Image Script

Clique novamente em cima de HUD e crie um novo Panel chamado PauseGame, nas propriedades dele deixe muda a cor para escuro e a camada A em 170 e desmarque o Raycast Target, a ideia é quando pausar o jogo, escurecerá a tela, tem uma linha branca do lado desse painel, para remover ele, vamos deixar em stretch x stretch e com -10 em todos os lados no Rect Transform do painel:

Figura 191 - Rect Tranform

Botão direito no PauseGame e em UI crie um objeto IMAGE, arraste para Source Image a imagem que se encontra em Sprite -> Hud -> Hud_0 renomeie para PainelBG

Figura 192 - Hud_0

Deixe marcado o preserve aspect e pressione o botão Set Native Size, nas propriedades Rect Transform deixe tudo no centro (Pos X e Y em zero):

Figura 193 - Posição do Menu

Agora, botão direito em cima do PainelBG e em UI adicione um TextMechPro Text e digite PAUSE, deixe tudo centralizado e arraste o texto para

o cabeçalho da sua imagem conforme abaixo, formate o texto como achar mais interessante:

Figura 194 - Painel de Pause

Desative o PauseGame, abra o script do _GameController e declare no final a seguinte variável:

```
public GameObject painelPause;
```

Em seguida salve e arraste para ele o PauseGame:

Figura 195 - Objeto PauseGame

No Script _GameController crie uma função no final do código chamada pauseGame que será responsável por parar o jogo quando pressionarmos a tecla ESC:

```
public void pauseGame()
{
}
}
```

E no void Update vamos chamar essa função:

```
if (Input.GetButtonDown("Cancel"))//quando pressionar a tecla ESC
abre/fecha o menu
{
    pauseGame();
}
```

Agora na função do pauseGame vamos abrir e fechar o painel sempre que pressionar o ESC:

```
public void pauseGame()
{
    bool pauseState = painelPause.activeSelf;
    pauseState = !pauseState;
    painelPause.SetActive(pauseState);

    switch (pauseState)
    {
        case true:
            Time.timeScale = 0;
            break;

        case false:
            Time.timeScale = 1;
            break;
    }
}
```

Em seguida vamos desativar o painelPause para que nunca tenha a chance de começar com o pause ativo durante o início do jogo, então no _GameController no Void Start() digite:

```
painelPause.SetActive(false); //inicia o jogo com o painelPause
desativado
```

Teste o jogo, reparem que ainda nosso personagem consegue fazer algumas ações, então vamos corrigir isso, no script do _GameController embaixo do ultimo USING vamos digitar:

```
public enum GameState
{
    PAUSE,
    GAMEPLAY
}
```

E na primeira linha onde declaramos as nossas variáveis vamos digitar:

```
public GameState currentState; //estado atual do JOGO
```

Salve o código, abra o unity e escolha a opção de GAMESTATE:

Figura 196 - Script Game Controller

Agora abra o Script do playerScript e no início do FixedUpdate e no Update vamos adicionar o seguinte código:

```
if (_GameController.currentState != GameState.GAMEPLAY)
{
    return;
}
```

Em seguida abram o Script do _GameScripController e criem uma nova função no final de todo nosso código:

```
public void changeState(GameState newState)
{
    currentState = newState;
}
```

Com isso podemos modificar o código da função pauseGame para ficar dessa forma:

```
case true:
    Time.timeScale = 0;
    changeState(GameState.PAUSE);
    break;
```

```
case false:
```

```
    Time.timeScale = 1;
```

```
    changeState(GameState.GAMEPLAY);
```

```
    break;
```

Teste o jogo!

Vamos implementar os botões do menu, então botão direito em PainelBG, adicione um botão a ele, apague o texto que vem dentro e adicione um TextMesh, adicione a imagem do HUD_18, duplique mais 2 vezes:

Figura 197 - Hud_18

Figura 198 - textMesh

No primeiro Botão chame de Status, no segundo ITENS e no Terceiro OPTIONS:

Figura 199 - Menu Completo

Agora vamos duplicar o objeto PauseGame e chamá-lo de Itens a ideia é quando pressionar um botão ITENS, ele desabilite um menu e ative outro.

Para trabalhar no menu de itens, primeiro desabilite o menu PauseGame em seguida edite o cabeçalho para ITENS, no painelBG mude a propriedade da imagem de Simple para Sliced, ele vai falar que a imagem não possui bordas, por isso vamos criar as bordas da imagem, para isso clique na imagem HUD dentro da pasta Sprites, clique em Sprite editor, selecione a imagem do menu e digite 10 nos valores de border, em seguida arraste as linhas das bordas para que elas peguem a maior parte dos cantos desenhados, conforme imagem abaixo:

Figura 200 - Sub Menu

Dê um apply, e redimensione nosso menu para completar a maior parte do espaço da tela, apague os 3 botões e deixe o texto Itens dentro do cabeçalho:

Figura 201 - Itens

Agora vamos programar para que quando apertarmos o botão Itens no Menu, abra o Menu de Itens, para isso, vamos abrir o Script do _GameController e declarar mais um painel:


```
public GameObject painellItens;
```

Depois arraste para ele o objeto Itens:

Figura 202 - Objeto Itens

Agora adicione ITENS ao GameState

Comentado [PR1]: Parei aqui

```
public enum GameState
{
    PAUSE,
    GAMEPLAY,
    ITEN
}
```

Em seguida mude a condição dentro do Update(), já que teremos que apertar duas vezes o esc agora, para sair de itens e depois do menu:

```
if (Input.GetButtonDown("Cancel") && currentState !=
GameState.ITEN)//quando pressionar a tecla ESC abre/fecha o menu
{
    pauseGame();
}
```

Depois disso vamos criar uma função no final do código chamado btnItensDown():

```
public void btnItensDown()
{
    painelPause.SetActive(false);
    painellItens.SetActive(true);
    changeState(GameState.ITEN);
}
```

Agora no Unity, no objeto PauseGame, temos os 3 botões, vamos renome alôs conforme abaixo:

Figura 203 - PauseGame

Em seguida selecione o botão ItensBtn e nas propriedades do botão, em On Click (), pressione o + , em seguida arraste o _GameController para abaixo do Runtime Only e no lado escolha _GameController -> BtnItensDown():

Figura 204 - Event On Click

Vamos fazer o mesmo que no painelPause, vamos no Script do _gameController e no void Start colocar que o painelltens também começará sempre desativado:

```
painelltens.SetActive(false); //inicia o jogo com o painelPause desativado
```

Testem o jogo , reparem que quando entrarmos no menu de itens, não consigo sair dele apertando ESC, mas consigo fechar o Menu com o ESC, então agora vamos criar um botão no menu ITENS para poder fechá-lo, para isso vamos desativar o painel de pause e reativar o painel de Itens, Botão direito em cima do paineIBG de ITENS, UI -> Button, como imagem vamos usar o HUD_4,

apague o texto do botão, chame-o de FecharBtn, deixe o preserve aspect ativado e pressione o botão Set Native Size:

Figura 205 - Image Script

Em seguida vamos mudar o Rect Transform para Top Right, pivot em 0 no x e y, e pos x -40, y -20 e z em 0:

Figura 206 - Rect Transform

O botão vai ficar no canto assim:

Figura 207 - Botão X

Agora vamos programar para quando pressionar esse botão, ele fechar o Menu de Itens, em `_gameController`, vamos criar uma função chamada `fecharPainel()` no final do código:

```
public void fecharPainel()
{
    painelItens.SetActive(false);
    //outros paineis quando estiverem prontos

    painelPause.SetActive(true);
    changeState(GameState.PAUSE);
}
```

Em seguida, vamos no objeto `FecharBtn`, pressionamos o + no Button em `On Click()`, arrastamos para dentro o objeto `_GameController` e em `No Function` escolhemos a `_GameController -> FecharPainel()`:

Figura 208 - Event Click

Teste o jogo.

Se quisermos fazer o mesmo para o Menu, basta duplicar o botão de fechar, arrastar ele para dentro do `PainelBG` no `Menu PauseGame`, recolocar a pos X para -40 e pos y -20 e mudar o evento do botão para `_GameController -> PauseGame()`:

Figura 209 - Event Click

Teste o jogo.

Apenas para nos localizar melhor, podemos adicionar uma cor de fundo diferente aos botões do Menu PauseGame quando passarmos o cursor do mouse em cima, para saber em que botão vamos clicar, para isso, basta selecionar os 3 botões, ir em Highlighted Color, escolher uma cor e pronto, quando o mouse passar em cima, ficará da cor escolhida. Testem o jogo.

Figura 210 - Cor de Fundo Botão