+Scripts

O Processo de desenvolvimento

Atividade: Gravity Gloves (Half-Life Alyx)

A proposta desta atividade é desenvolver, passo a passo, um script que tem o efeito de atrair objetos distantes para a mão do usuário, algo muito útil (em qualquer realidade) e por enquanto possível somente na Realidade Virtual.

Esse recurso reduz deslocamentos, exigindo portanto menos espaço físico, além do esforço de pular ou abaixar, pois muitos usuários não desejam ou não podem executar tais esforços repetitivos.

Etapas lógicas

1 - Ativar modo “GravityRay” ao pressionar um botão/tecla (semelhante ao teleporte)

2 - Emitir um raio e filtrar alvos de acordo com requisitos (a definir)

3 - Ao pressionar gatilho/clique, atrair alvo para mão do usuário

4 - Ao desativar o modo “GravityRay” usuário pode manipular objeto normalmente

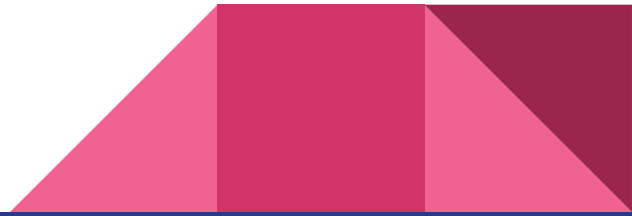
5 - Soltar o gatilho/clique em modo “GravityRay” reverte efeito, repelindo o alvo

Requisitos do Alvo

Para poder ser atraído pelo raio, o objeto deve atender aos requisitos:

1 - Ter um componente Throwable (e Rigidbody, por consequencia);

2- Estar a uma distância mínima do player (fora do alcance das mãos)



Criando os Métodos

CastRay -> gera o raio e define um alvo (target)

StopCast -> cancela o raio

Attract -> atrai o alvo para a mão do usuário

StopAttract -> encerra o efeito de atração

Repel -> repele o alvo

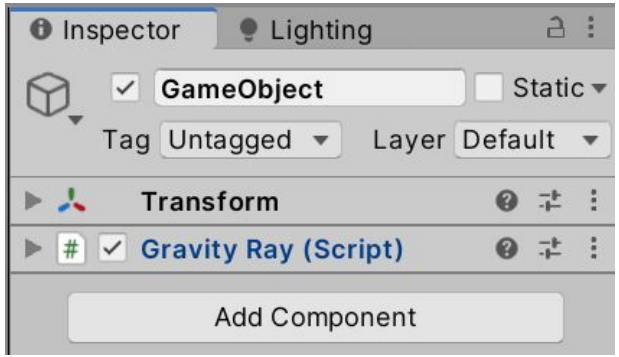
CheckInputs -> captura os comandos

```
public class GravityRay : MonoBehaviour
{
    public Rigidbody target = null;
    0 references
    private void CastRay() { }
    0 references
    private void StopCast() { }
    0 references
    private void Attract() { }
    0 references
    private void StopAttract() { }
    0 references
    private void Repel() { }
    0 references
    private void CheckInputs() { }
    0 references
    private void Start() { }
    0 references
    private void Update() { }
}
```

Onde aplicar o Script?

Geralmente os scripts que herdam de Monobehaviour são componentes de um objeto em cena. Portanto é preciso arrastá-lo para dentro de um objeto.

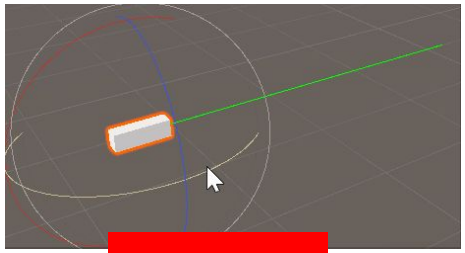
Para nossas atuais necessidades, podemos simplesmente aplicá-lo em um objeto Empty sem problema algum, pois nosso script irá atrás de quase tudo que precisamos.



Raycast (entendendo)

A Unity possui um método para lançar raios, “Physics.Raycast()”, com o qual podemos detectar colisão com objetos. Para ele funcionar, devemos prover um Ponto de Origem, uma Direção e uma distância máxima.

O Raycast é invisível, então para visualizar, podemos recorrer a um método semelhante, “Debug.DrawRay()”. A diferença é que o DrawRay exige ponto de Início e Fim.



Exemplo

```
void Update()
{
    Physics.Raycast(transform.position, transform.forward, 2f);
    Debug.DrawRay(transform.position, transform.forward * 2f, Color.green);
}
```

Origem

Direção

Distancia

Inicio

Fim

Ativando e Desativando

Vamos criar uma variável chamada “isCasting” (1). Se verdadeira, o raio deverá estar ativo.

Quando o usuário pressionar a tecla “espaço” (2) a variável é alterada.

(futuramente basta implementar o input dos controles 6DoF neste mesmo método - ver exemplo ao fim)

Por fim, vamos checar o input no Update, que é executado a cada frame, e dependendo da variável “isCasting”, emitimos ou interrompemos o raio (3)

```
public Rigidbody target = null;
private bool isCasting = false;
```

1

```
private void CheckInputs()
{
    if (Input.GetKeyDown("space"))
        isCasting = true;

    if (Input.GetKeyUp("space"))
        isCasting = false;
}
```

2

```
private void Update()
{
    CheckInputs();

    if (isCasting)
        CastRay();
    else
        StopCast();
}
```

3

Etapas lógicas

Concluido!

1 - Ativar modo “GravityRay” ao pressionar um botão/tecla (semelhante ao teleporte)

2 - Emitir um raio e filtrar alvos de acordo com requisitos (a definir)

3 - Ao pressionar gatilho/clique, atrair alvo para mão do usuário

4 - Ao desativar o modo “GravityRay” usuário pode manipular objeto normalmente

5 - Soltar o gatilho/clique em modo “GravityRay” reverte efeito, repelindo o alvo

Preparando o Raio: classes auxiliares

Guarda informações de colisão do raio

```
private void CastRay()
{
    RaycastHit hitInfo;
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
```

Guarda a origem e direção de um raio

Origem: camera
Direção: cursor mouse

(Futuramente basta alterar este Ray para ter o controle 6DoF como origem e direção)

Disparando e Filtrando

1

SE o raio acertar algo...

```
...
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(ray, out hitInfo, 100)
        && hitInfo.collider.attachedRigidbody != null)
{
    target = hitInfo.collider.attachedRigidbody;
}
else
    target = null;
```

2

...**E** o algo tiver um Rigidbody

ENTÃO o algo é um “target”

3

4

SENÃO “target” é nulo

Teste 1

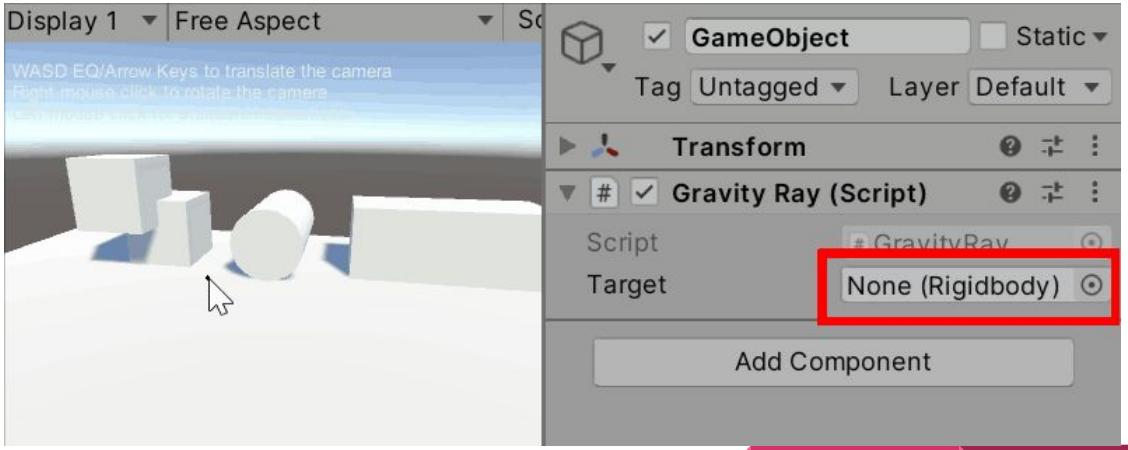
Se estiver tudo certo até aqui, este será o resultado esperado:

Cena de teste:

Cubo: throwable + velocity estimator

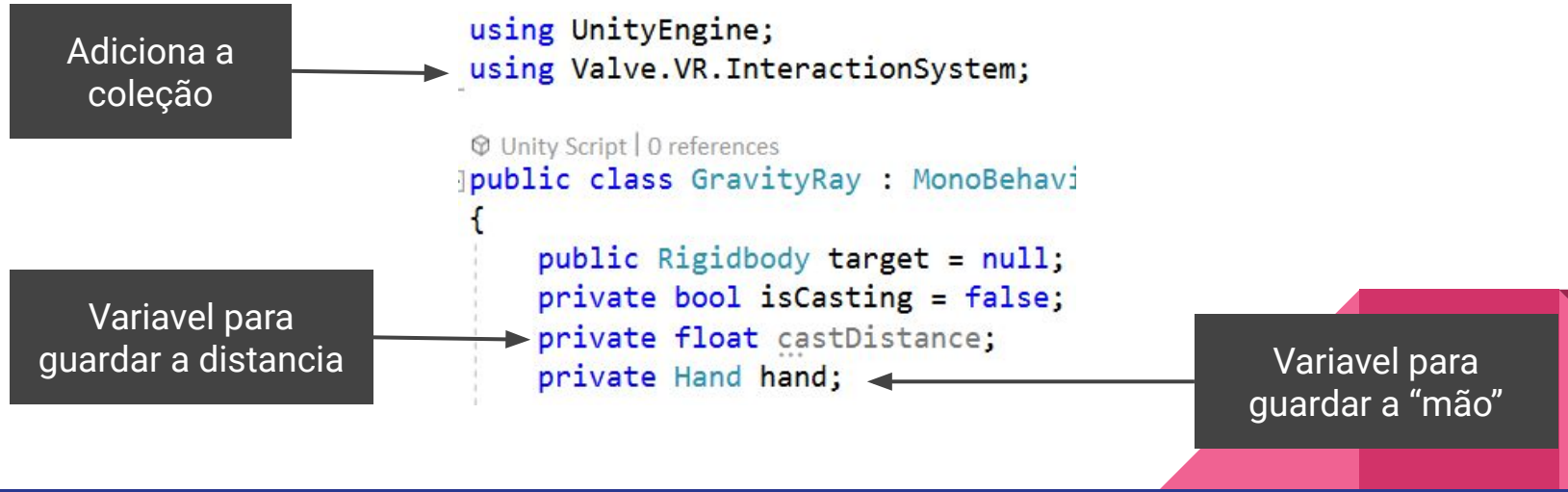
Cilindro: throwable + velocity estimator

Retangulo: apenas rigidbody



Adicionando as classes

Os **Requisitos do Alvo** específica objetos fora do alcance das mãos. Para isso podemos importar as classes do SteamVR e economizar algum trabalho. Aproveitamos e criamos duas variáveis que usaremos a seguir.



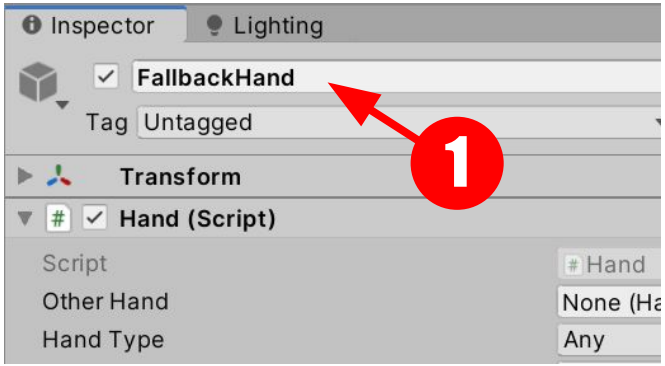
Alcance das Mãos

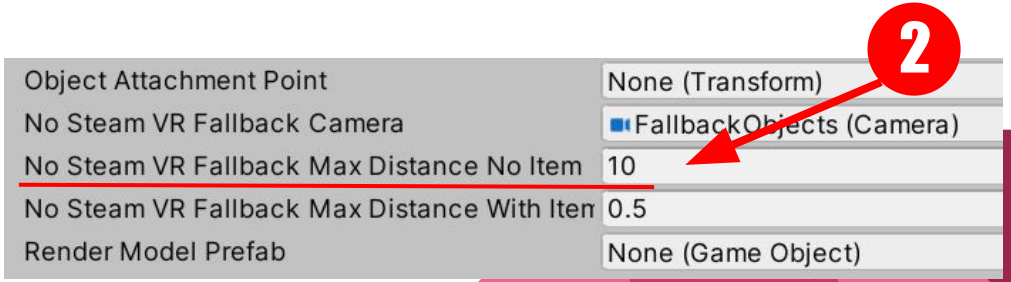
Vamos pegar a variável que define o alcance das mãos no modo 2D debug.

Por padrão é 10m, mas é possível ajustar bastando acessar o objeto FallbackHand (1) no Player Prefab e modificar o parâmetro (2)

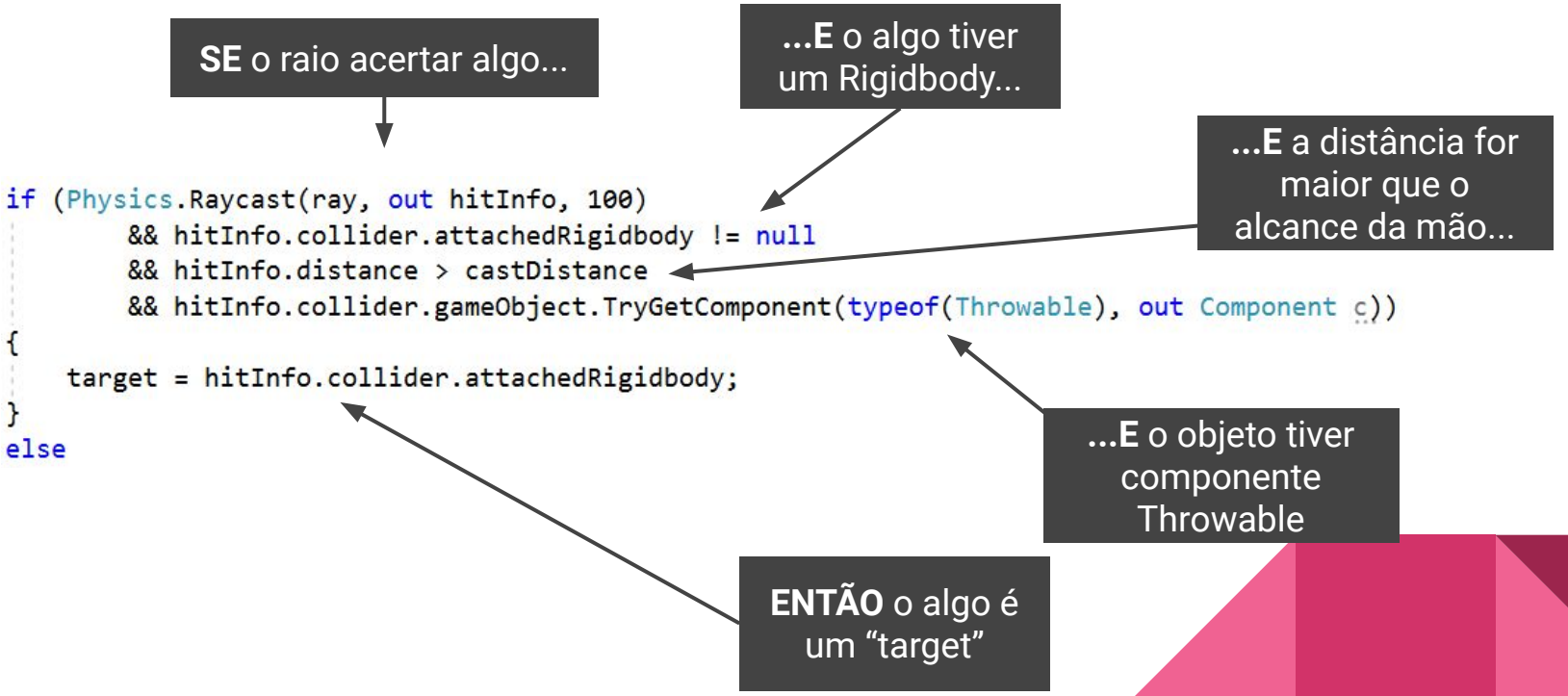
Alcance das mãos em modo 2D

```
private void Start()
{
    hand = Player.instance.GetHand(0);
    castDistance = hand.noSteamVRFallbackMaxDistanceNoItem;
}
```





Adicionando Mais Filtros



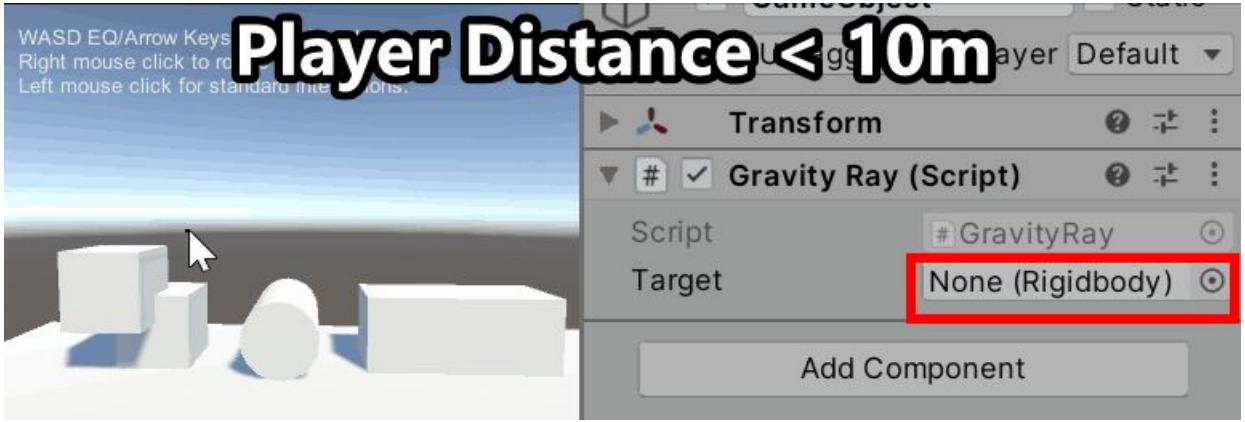

```
if (Physics.Raycast(ray, out hitInfo, 100)
        && hitInfo.collider.attachedRigidbody != null
        && hitInfo.distance > castDistance
        && hitInfo.collider.gameObject.TryGetComponent(typeof(Throwable), out Component c))
{
    target = hitInfo.collider.attachedRigidbody;
}
else
```

Teste 2

Aqui pode ser necessário ajustar a cena de teste, aumentando os objetos

Ou

Reduzir o parâmetro de alcance das mãos (em FallbackHand) para 2m, por exemplo.



Etapas lógicas

✖ Ativar modo “GravityRay” ao pressionar um botão/tecla (semelhante ao teleporte)

✖ Emitir um raio e filtrar alvos de acordo com requisitos (a definir)

3 - Ao pressionar gatilho/clique, atrair alvo para mão do usuário

4 - Ao desativar o modo “GravityRay” usuário pode manipular objeto normalmente

5 - Soltar o gatilho/clique em modo “GravityRay” reverte efeito, repelindo o alvo

Atraindo o objeto

Novamente vamos declarar uma variável booleana “isAttracting” e uma RigidBody “targetLocked” (1)

Capturar o Input do mouse no método “CheckInputs” para alterar a variável (2)

E no loop “Update”, vamos verificar a condição da variável e disparar nossos métodos “Attract” ou “StopAttract” (3).

```
public Rigidbody target = null;
public Rigidbody targetLocked = null;
private bool isAttracting = false;
private bool isCasting = false;
```

1

```
if (Input.GetKeyUp("space"))
    isCasting = false;

if (Input.GetMouseButtonDown(0) && target != null)
    isAttracting = true;

if (Input.GetMouseButtonUp(0))
    isAttracting = false;
```

2

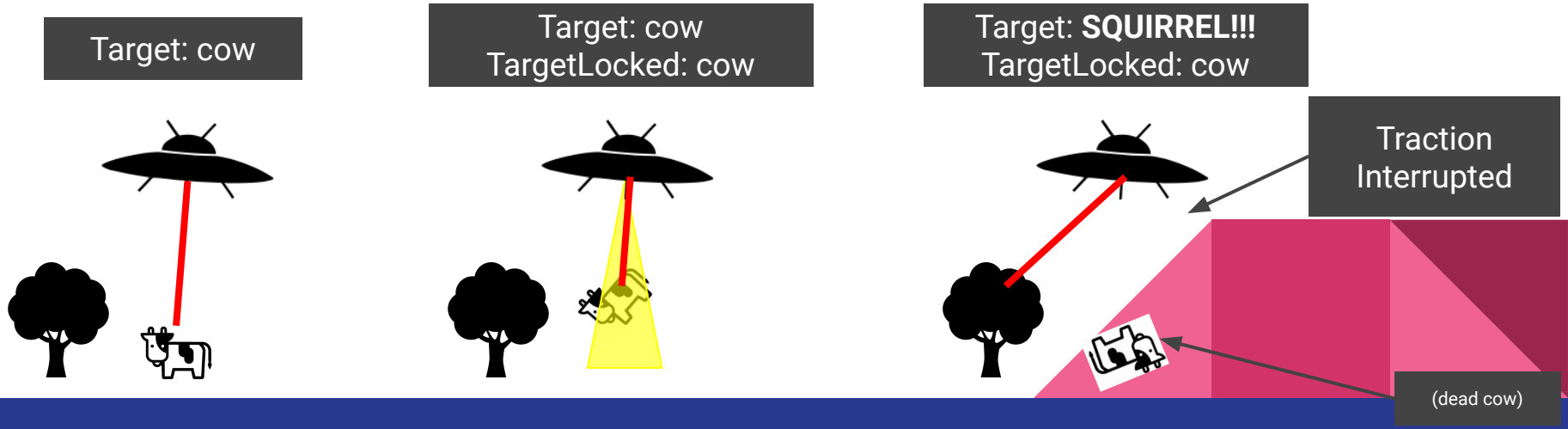
```
if (isCasting)
    CastRay();
else
    StopCast();

if (isAttracting)
    Attract();
else
    StopAttract();
```

3

Target x TargetLocked

Target guarda o atual alvo válido identificado pelo raio.
Target Locked guarda o alvo selecionado pelo usuário para ser atraído

Se a variável Target mudar ou anular, devemos interromper o efeito trator sobre o objeto



Checando as condições

Target nulo: interromper

Target definido: iniciar

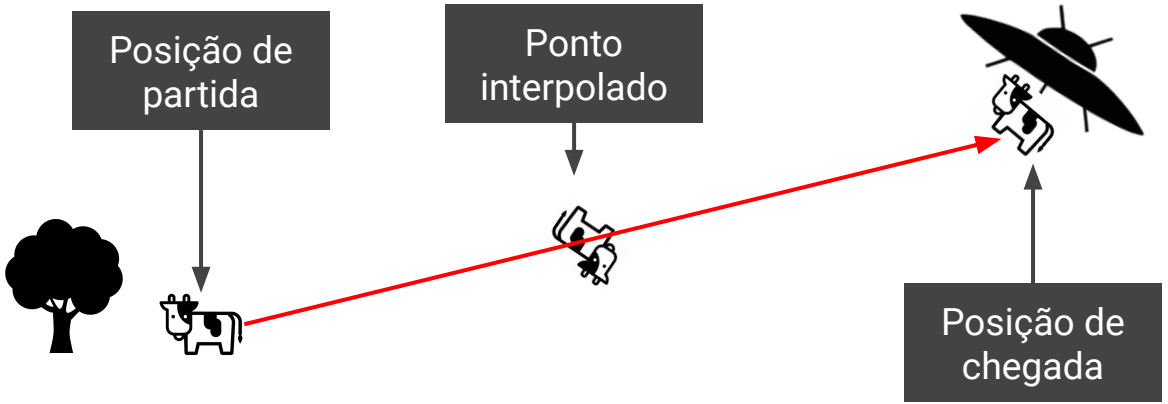
Target alterado: interromper e recomeçar

```
private void Attract()
{
    if (target == null && targetLocked != null)
    {
        isAttracting = false;
    }

    if (targetLocked == null && target != null)
    {
        targetLocked = target;
        targetLocked.useGravity = false;
    }

    if (targetLocked != target && target != null)
    {
        isAttracting = false;
        return;
    }
```

Desligando a gravidade do alvo

Movimento Interpolado

A interpolação retorna um dado valor dentro de uma extensão conhecida. No caso do movimento, a extensão é o espaço entre o início e o fim do movimento, sendo que o terceiro valor é o ponto interpolado.



Movimento Interpolado

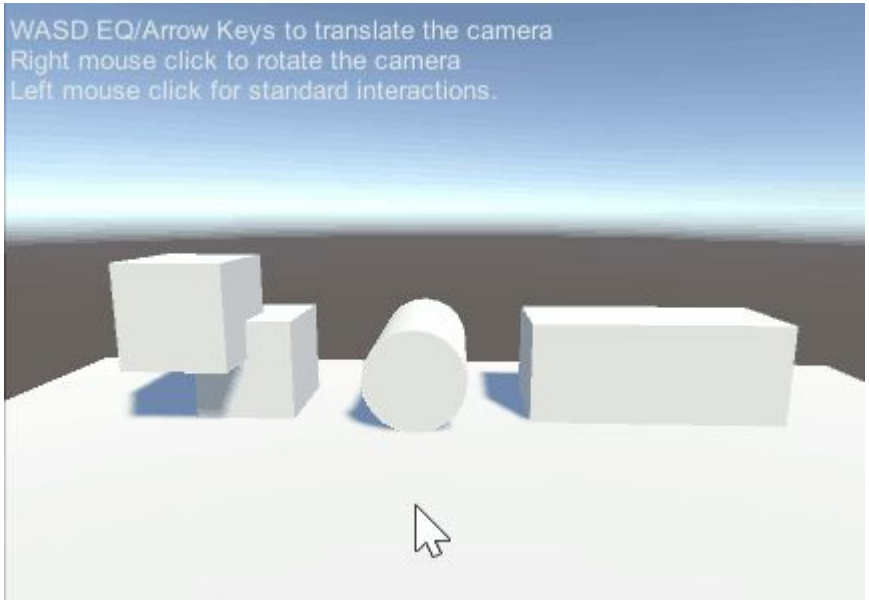
Desativando o Raio e o efeito

Ao desativar o raio, o único cuidado que devemos ter é limpar as variáveis, para deixar pronto para o próximo uso.

```
private void StopCast()
{
    target = null;
    StopAttract();
}
```

```
private void StopAttract()
{   //encerra o objeto a ser movido
    if(targetLocked != null)
    {
        targetLocked.useGravity = true;
        targetLocked = null;
    }
}
```

Teste 3


WASD EQ/Arrow Keys to translate the camera
Right mouse click to rotate the camera
Left mouse click for standard interactions.

Agarrando Alvo ao fim do curso

No CastRay()

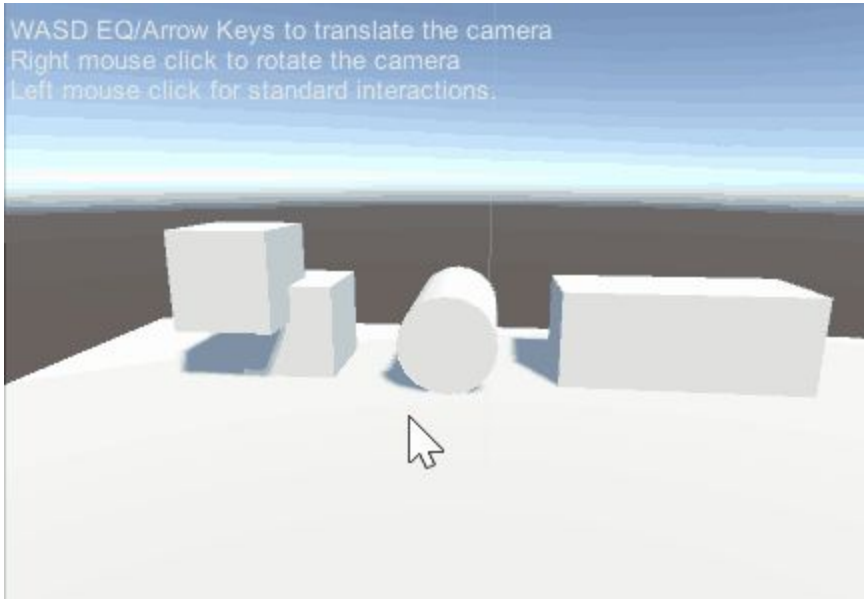
```
if (Physics.Raycast(ray, out hitInfo, 100)
        && hitInfo.collider.attachedRigidbody != null
        && hitInfo.distance > castDistance - 0.2f
        && hitInfo.collider.gameObject.TryGetComponent(typeof(Throwab
{
    target = hitInfo.collider.attachedRigidbody;
    if (hitInfo.distance <= castDistance && target == targetLocked)
    {
        hand.AttachObject(target.gameObject, GrabTypes.Trigger);
    }
}
```

Mantém raio ativo 20cm antes do alcance da mão

SE a distância até objeto for menor ou igual o alcance da mão **E** targetLocked for válido

ENTÃO agarra objeto

Teste 4


WASD EQ/Arrow Keys to translate the camera
Right mouse click to rotate the camera
Left mouse click for standard interactions.

Etapas lógicas

- ✖ Ativar modo “GravityRay” ao pressionar um botão/tecla (semelhante ao teleporte)
- ✖ Emitir um raio e filtrar alvos de acordo com requisitos (a definir)
- ✖ Ao pressionar gatilho/clique, atrair alvo para mão do usuário
- ✖ Ao desativar o modo “GravityRay” usuário pode manipular objeto normalmente

5 - Soltar o gatilho/clique em modo “GravityRay” reverte efeito, repelindo o alvo

Recuperando o target perdido

Existe um pequeno problema com o método “AttachObject” que utilizamos. Ao agarrar um objeto muito grande, o raio pode ser cortado e perder o “target”

Por isso, antes de usar o AttachObject, temos que copiar o valor de target em uma outra variável que chamaremos de “targetGrabbed” (1)

Sem esquecer de declarar a variável no início da classe (2) e limpar ao fim (3)

CastRay()

```
target = hitInfo.collider.attachedRigidbody;
if (hitInfo.distance <= castDistance && target == targetLocked)
{
    targetGrabbed = target;
    hand.AttachObject(target.gameObject, GrabTypes.Trigger);
}
```

1

```
public Rigidbody target = null;
public Rigidbody targetLocked = null;
public Rigidbody targetGrabbed = null;
private bool isAttracting = false;
```

2

```
private void StopCast()
{
    target = null;
    targetGrabbed = null;
    StopAttract();
}
```

3

Repelindo o Alvo

SE soltar o click do mouse **E** o raio estiver ativo **E** houver um objeto na mão

CheckInput

```
if (Input.GetMouseButtonUp(0) && isCasting && targetGrabbed != null)
    Repel();
```

Direção: mouse pointer

Reativa física (desativada pela mão)

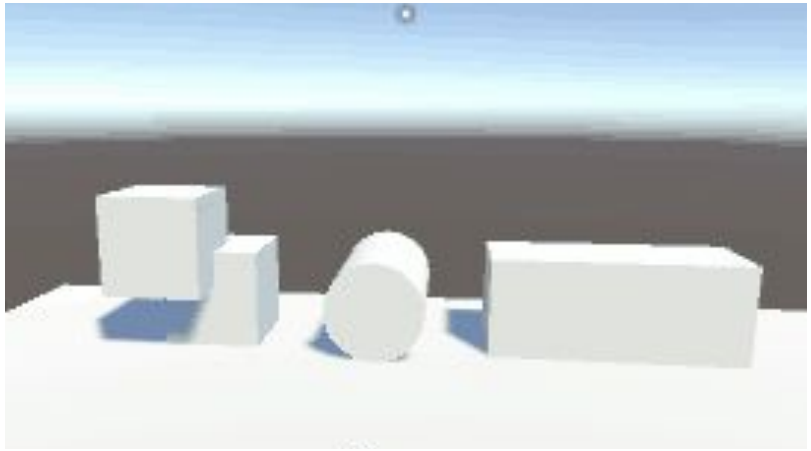
Aplica uma força

```
private void Repel()
{
    Vector3 repelDirection = Camera.main.ScreenPointToRay(Input.mousePosition).direction;
    targetGrabbed.isKinematic = false;
    targetGrabbed.AddForce(repelDirection * 50f, ForceMode.Impulse);
    targetGrabbed = null;
}
```

Multiplica o vetor da força por 50 (ajuste a gosto)

Especifica tipo de força

Teste 5 - Final!



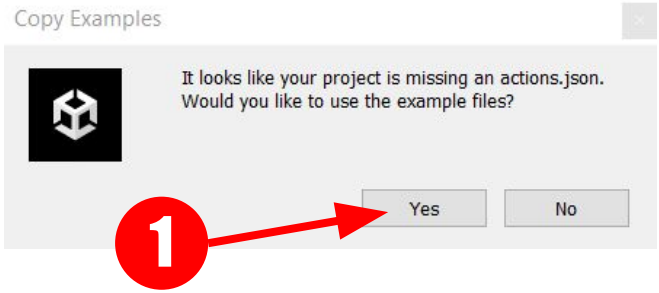
Etapas lógicas

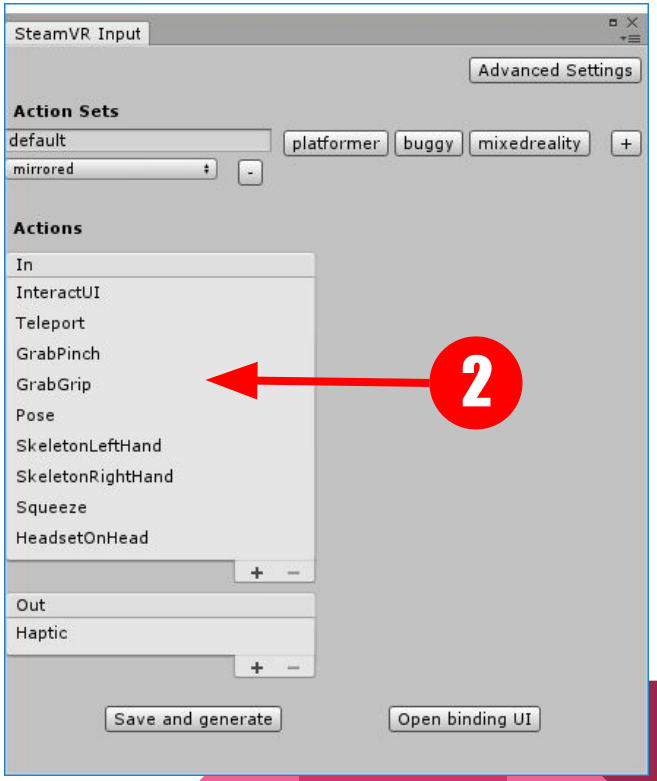
- [x] Ativar modo “GravityRay” ao pressionar um botão/tecla (semelhante ao teleporte)
- [x] Emitir um raio e filtrar alvos de acordo com requisitos (a definir)
- [x] Ao pressionar gatilho/clique, atrair alvo para mão do usuário
- [x] Ao desativar o modo “GravityRay” usuário pode manipular objeto normalmente
- [x] Soltar o gatilho/clique em modo “GravityRay” reverte efeito, repelindo o alvo

SteamVR Input -> Actions

Na Toolbar do unity selecione Window>SteamVR Input

Clique em yes para gerar a configuração padrão de inputs (2).





Exemplo de chamada de input para controles RV

Mais informações em:
valvesoftware.github.io/steamvr_unity_plugin/tutorials/SteamVR-Input.html

Recapitulando...

Criamos um script para interação do tipo "Gravity Glove". Aprendemos a emitir um raio e identificar o objeto atingido, capturar e identificar Inputs de teclado e mouse, acessar parâmetros da Hand do SteamVR e aplicar forças em direções específicas.

Recomenda-se que scripts de lógica global estejam em um objeto empty criado especificamente para este propósito. O script foi dividido em diversos métodos, cada um com ação bem determinada. Recomenda-se na função Update realizar apenas chamadas de métodos e evitar escrever muito código ali - lembre-se que o código de todas as funções Update (de cada Script), roda a cada quadro. Também usamos booleanas para determinar quando o Update deve rodar um método. Nos métodos, sempre crie condições bem definidas para controlar o fluxo e evitar execução desnecessária de código

Recapitulando...

A Unity possui muitos tipos de objetos a disposição, como Rigidbody, Ray, Vector3, MeshFilter, MeshRenderer e RaycastHit. As bibliotecas também oferecem objetos, como o Hand do SteamVR. Objetos são muito úteis, pois agrupam as características, parâmetros e métodos aplicáveis a cada entidade. A classe ou objeto Mesh oferece os métodos mais comuns para manipular a geometria pura (vértices, faces), por exemplo.

É muito importante sempre consultar a documentação da API para verificar tudo que já existe a disposição, para agilizar o desenvolvimento e otimizar o código. Funções existentes certamente terão desempenho superior a qualquer tentativa de réplica, pois são desenvolvidas e testadas por especialistas. Consulte também as API das bibliotecas externas, como a SteamVR. Os links para essas API podem ser facilmente encontrados em buscas online.

Conclusão

O objetivo da Realidade Virtual **não é recriar a realidade fielmente** (que vantagem haveria então?), mas explorar as inúmeras possibilidades existentes precisamente nas diferenças além dos **limites físicos da realidade**. A “Gravity Glove” que desenvolvemos é apenas um exemplo de como uma interação básica (pegar e soltar objetos) pode ser transformada em uma interação “mágica” muito relevante para aplicações de realidade virtual, reduzindo esforço físico e a quantidade de espaço necessária.