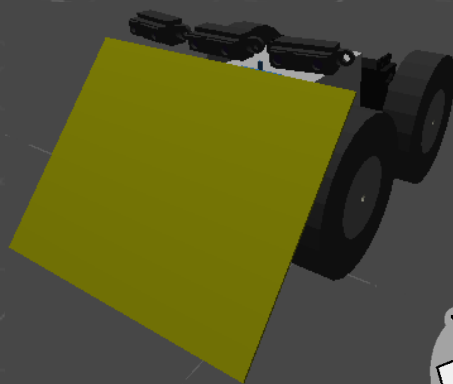


# ROBÓTICA COM O UNITY:

## Um framework simples e efetivo

Compatível  
com o  
Unity 5!!



Claudio Mota Oliveira  
Marco Túlio Chella



# ROBÓTICA COM O UNITY:

Um framework simples e efetivo

Robótica com o Unity: Um framework simples e efetivo.

Volume 1

Ficha catalográfica:

**Robótica com o Unity: Um framework simples e efetivo.**  
OLIVEIRA, C. M.; CHELLA, M. T.. São Cristovão-SE, 2015.

Claudio Mota Oliveira  
Marco Túlio Chella

# ROBÓTICA COM O UNITY:

Um framework simples e efetivo

São Cristovão-SE  
2015



Dedico esta obra a minha  
mãe Jacy, e a toda minha família.  
Por todo amor, paciência que  
tiveram, e toda felicidade que me  
proporcionaram.





“O conhecimento é uma ferramenta, e como todas as ferramentas, o seu impacto está nas mãos de quem a usa.”

Dan Brown



# Sumário

Introdução.....	11
1º Capítulo – Unity? O que é isso?.....	12
1.1 – Criando o primeiro projeto.....	15
Noções de packages.....	16
Noções de configuração de projeto.....	17
Cenários.....	18
Objetos.....	20
Luz.....	21
Câmera.....	22
Componentes.....	23
1.2 – Scripts com Unity.....	28
1.3 – Criando prefabs.....	34
Hierarquia de objetos.....	36
Criando/Importando packages.....	37
1.4 – Exercícios de fixação.....	40
2º Capítulo – O FrameWork.....	41
2.1 – Configurando o framework.....	42
Física.....	42
Tempo.....	43
2.2 – Prefabs do framework.....	43
Servomotor Limitado.....	44
Servo de rotação.....	46
Cola.....	48
Sensor Infra Vermelho.....	49
Sonar.....	50
Sensor RGB.....	51
Acelerômetro.....	54
Botão.....	55
Bussola.....	56
Roda.....	57
2.3 – A comunicação cliente-servidor.....	58
O protocolo TCP.....	59
Exemplo de Uso.....	59
2.4 – Ferramentas e Dicas.....	65
O Sketchup.....	65

Dicas importantes.....	67
3º Capítulo – Simulações simples.....	68
3.1 – Construindo um ventilador.....	68
3.2 – Construindo um carro de controle remoto.....	72
3.3 – Construindo robô programável remotamente.....	78
4º Capítulo – Simulação de robô sumô.....	85
4.1 – Preparando o cenário e as regras.....	86
4.2 – Construindo os robôs competidores.....	89
4.3 – Conectando os sumôs a rede.....	93
4.4 – Programando os sumôs remotamente.....	100
4.5 – Executando a competição.....	103
Referências.....	105

# Introdução

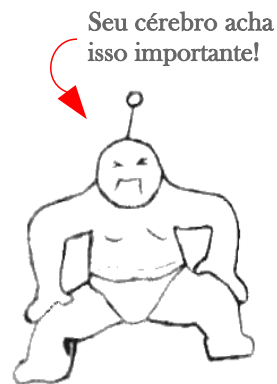
Este livro aborda o uso do Unity como ferramenta para desenvolvimento de simuladores em robótica, e para isso introduz um framework que busca preparar um ambiente básico para estes desenvolvedores de simuladores em robótica.

A abordagem do livro se dá baseada na fórmula usada pela Head First, que é uma organização de produção de livros instrucionais, que possui como visão o desenvolvimento de livros que sejam amigáveis ao cérebro do leitor. Para isso eles se utilizam de teorias de psicologia cognitiva, e teorias de como aprender a aprender.



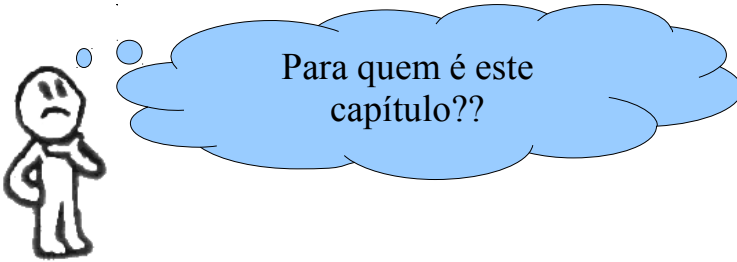
O uso de imagens que tenham que se relacionem ao assunto, juntamente com sarcasmo, redundância do conteúdo, e estímulo de diferentes regiões do cérebro são algumas das técnicas pregadas pela organização.

Basicamente se você está andando na rua, e de repente aparecem dois sumôs brigando... Seu cérebro vai achar essa informação mais importante de ser guardada, que decorar 329 páginas de um livro chato que seu chefe te mandou ler.

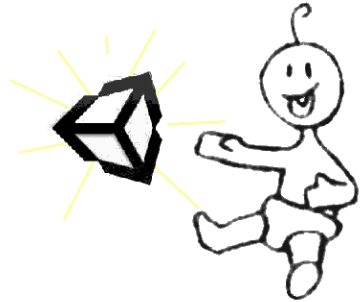


# 1º Capítulo: Unity? O que é isso?

Neste capítulo veremos os passos básicos para criar e configurar projetos no Unity, criação de cenários, importação de assets e prefabs e desenvolveremos um pequeno projeto.



Se você é um usuário iniciante do Unity este capítulo irá te guiar para que saiba o essencial para executar os projetos dos próximos capítulos.



Se você já é um usuário experiente do Unity, este capítulo não é obrigatório.

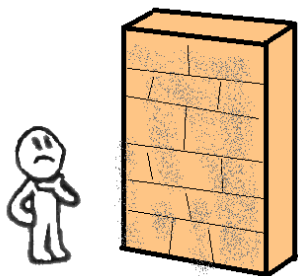


Bem vindo ao  
mundo Unity!!



O Unity é uma ferramenta bastante poderosa para criação de jogos por possuir um conjunto de ferramentas que facilitam o desenvolvimento como: possuir uma vasta API (Application Programming Interface), motores gráfico e de física, bem documentado e por ser de uso simples.

Numa analogia rápida: Se construir um jogo fosse como derrubar uma parede, o Unity seria o martelo de Thor!!



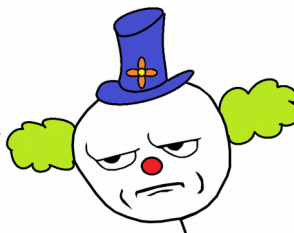
Você desenvolvendo jogos sem Unity



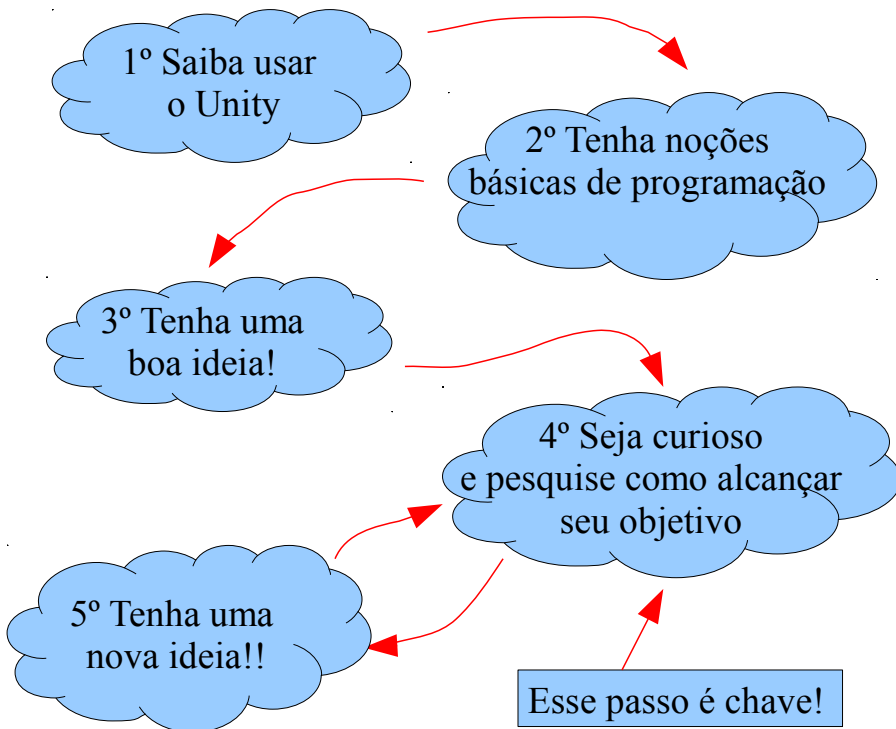
Você desenvolvendo jogos com o Unity

Certo... O Unity é ótimo para desenvolver jogos, contudo sua finalidade não é apenas essa. O Unity possui o motor de física PhysX da Nvidia que por sua vez possibilita, a nós desenvolvedores Unity, criar jogos com a física bastante próxima da realidade, ou seja... Simuladores! Ou em outra forma de falar... Serious Games!

Isso mesmo.. **Serious Game**



E para que seja possível o desenvolvimento de simuladores utilizando o Unity é necessário que você possua alguns poucos requisitos básicos:



A partir da próxima seção, iniciaremos um manual de sobrevivência com o Unity, para que você tenha noções básicas do uso desta ferramenta.

Para fazer o download do Unity acesse o link:

[unity3d.com/pt/get-unity/download](https://unity3d.com/pt/get-unity/download)

Feito o download, instale o Unity e vamos pôr a mão na massa!!!

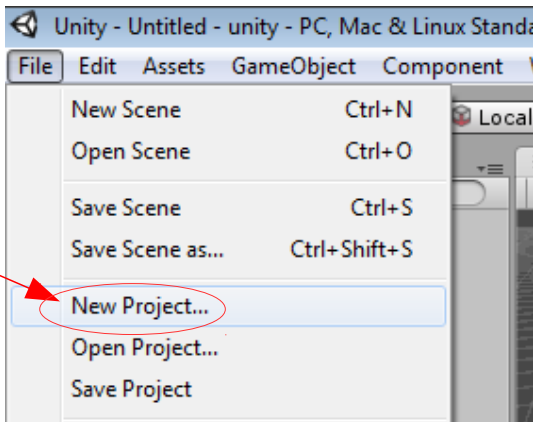


## 1.1 – Criando o primeiro projeto

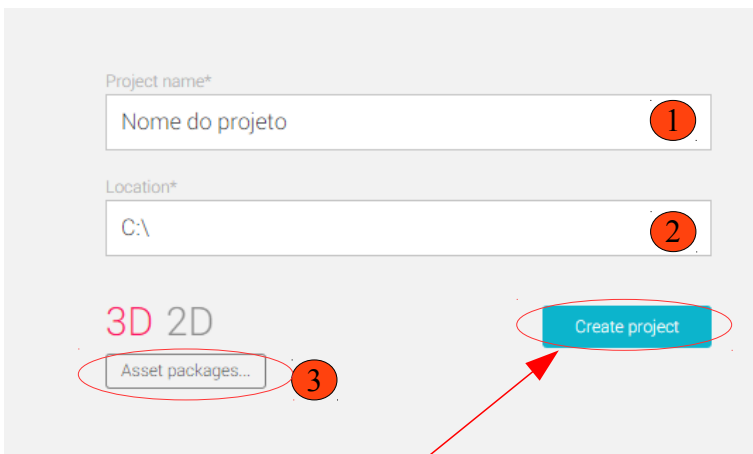
A primeira etapa para o desenvolvimento do seu simulador é criar um projeto, e para isso siga os seguintes passos:

1 - Execute o Unity

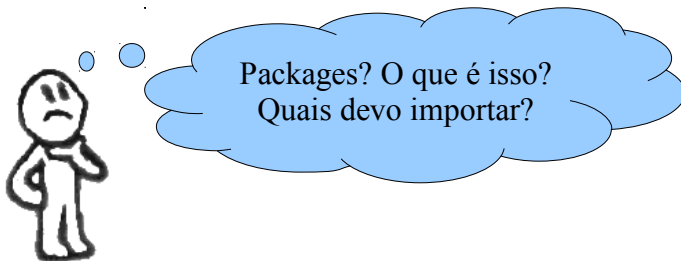
2 – Selecione a opção da aba: “File → New Project”



3– Uma nova janela irá surgir, nela você irá inserir o nome do projeto (1), a pasta dele (2), e selecionará os packages (3) que virão inclusos nele



4 – Aperte o botão “Create”



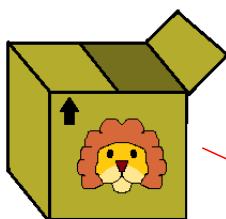
Os packages são um conjunto de elementos do Unity que foram compactados para serem usados em qualquer outro projeto. Estes elementos podem ser qualquer coisa... Texturas, Objetos, scripts, prefabs entre outros. Numa analogia simples:

1 - Quando você cria um projeto no Unity é como se você cria-se um mini-mundo vazio.



2 – Suponha que você quer que no seu mini-mundo tenha animais, mas você não quer ter que modelá-los manualmente.

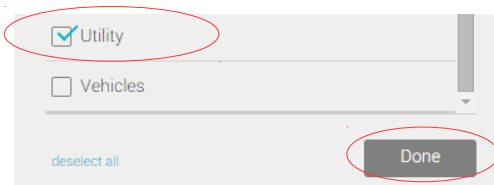
3 – Então a solução para você é procurar num repositório de packages um pacote de animais para seu simulador. daí é só importar este package para seu projeto



Importar  
animais.upkg

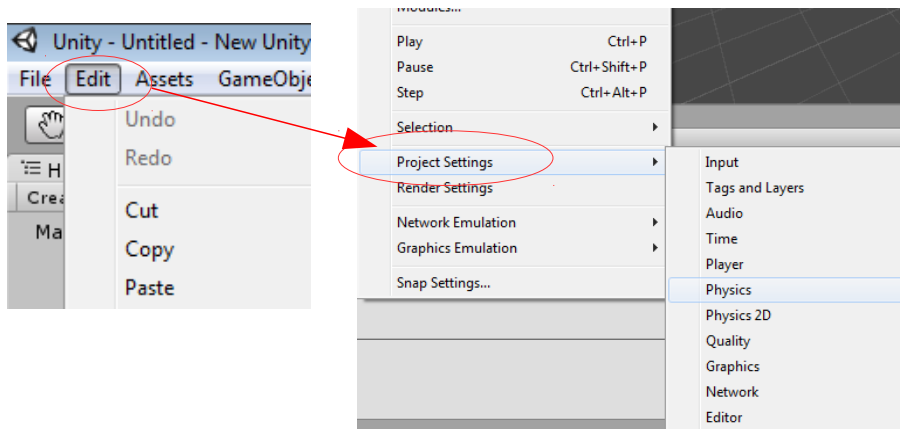


Em relação aos packages padrões do Unity recomendamos que você importe para seus projetos pelo menos o “Utility” package por possuir elementos de utilidade para o projeto.



Uma vez com o projeto criado pode ser necessário alterar as configurações deste dependendo da necessidade do seu simulador.

As configurações de projeto podem ser acessadas nas abas: “Edit → Project Settings”.



No caso do framework abordado neste livro será necessário alterar a configuração “Physics” e “Time” do projeto. Mas isso será abordado mais profundamente no segundo capítulo.

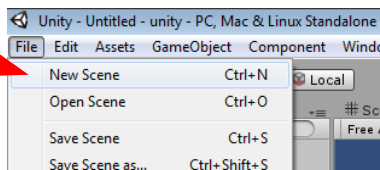
Para mais informações sobre cada tipo de configuração de projetos acesse o link:

<http://docs.unity3d.com/Manual/comp-ManagerGroup.html>

Estando o projeto criado e configurado, está na hora de começar o trabalho.. Vamos criar nosso primeiro cenário!!

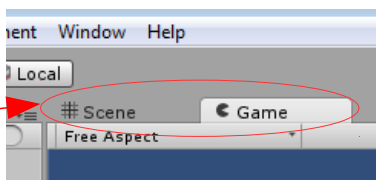
O Unity é baseado em cenários onde você monta um ambiente que deverá ser simulado, posicionando objetos no cenário por meio da tela de edição.

Para criar um novo cenário basta selecionar a opção da aba: “File → New Scene”.



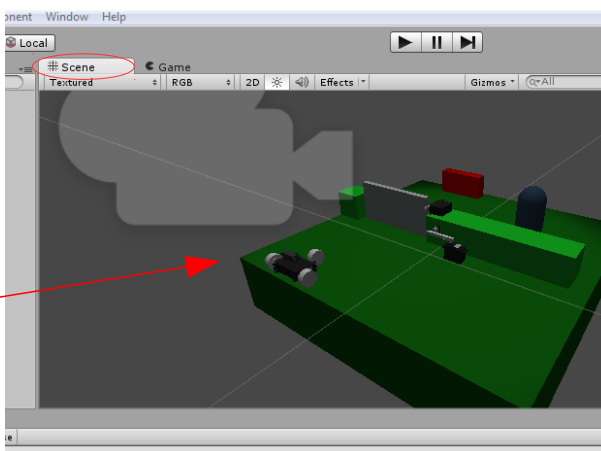
Existem duas telas de visualização do cenário são elas: a “Scene” e “Game”.

Basta clicar para alternar o modo de visualização



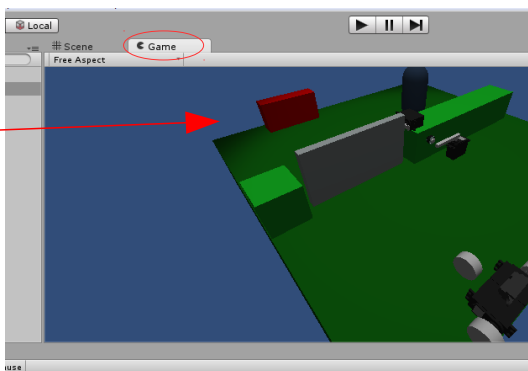
Toda atividade relacionada à construção da cena de simulação é feita na tela “Scene”, onde você irá posicionar objetos, câmeras e iluminação.

Cenário visualizado no modo “Scene”

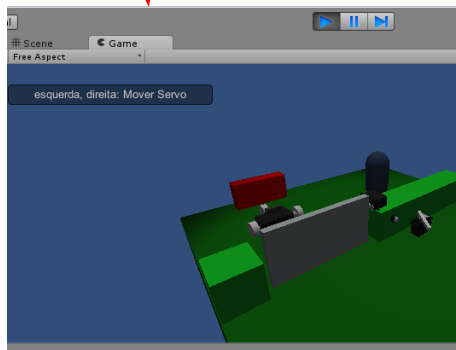
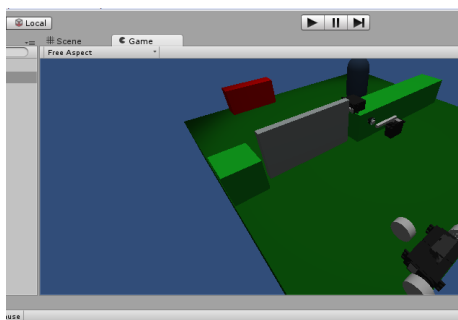


Já a tela “Game” mostra o que o usuário final do seu simulador visualizará quando executá-lo. Note que não é possível modificar o cenário utilizando esta tela.

Cenário visualizado no modo “Game”



Uma vez com os objetos posicionados no cenário, para simular o funcionamento do seu destre, basta clicar no botão de execução.



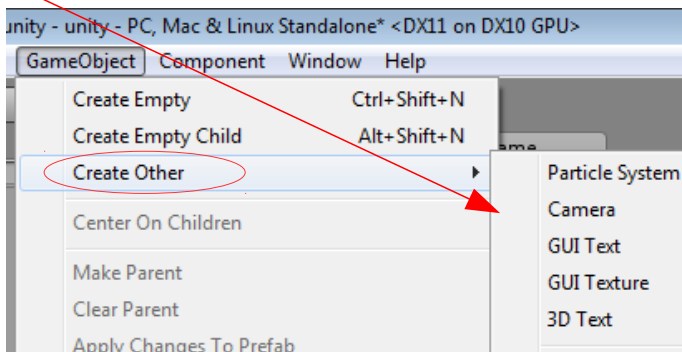
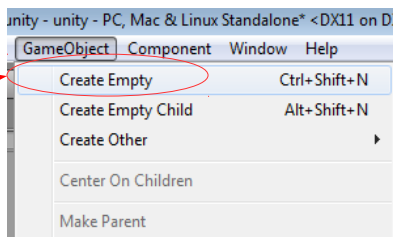
A execução da simulação pode ser visualizada em ambos os modos (Game/Scene), apesar do mais usual ser o “Game”. Nesse caso o modo “Scene” geralmente é mais útil em situações de debug, ou caso queira visualizar o funcionamento do cenário de diferentes ângulos.



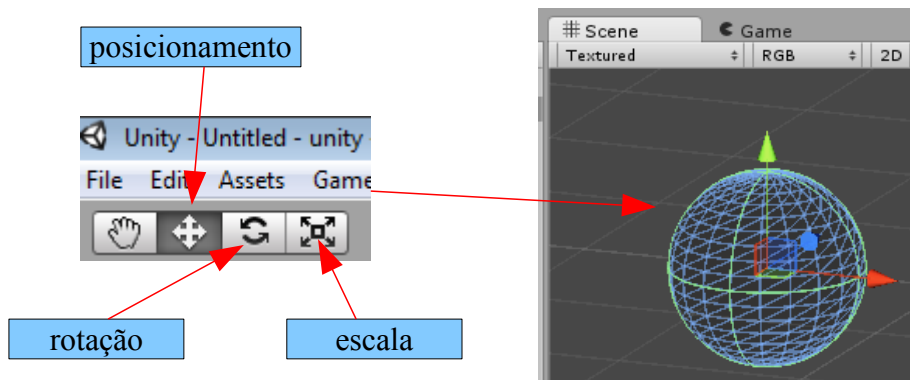
Todo cenário é formado por um conjunto de objetos, alguns destes objetos, como câmera e fontes de luz, são essenciais para qualquer cenário. Agora nós veremos como criá-los.

Para criar um novo Objeto, basta ir na aba: “GameObject”, Lá você terá a opção de criar um objeto vazio ou escolher dentre os objetos padrões existentes na aba: “GameObject → Create Other”.

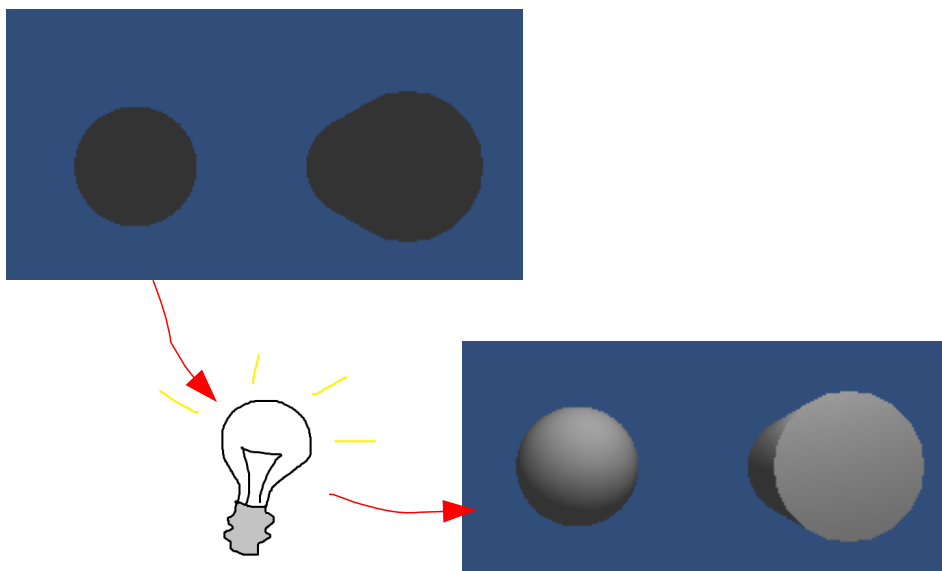
Cria objeto vazio  
Ou  
Seleciona já existentes



Após criar o objeto, temos que posicioná-lo no cenário. Para isso, basta selecionar modo posicionamento, e depois clicar e arrastar as setas do objeto na tela "Scene".



Para a boa visualização dos objetos do cenário é importante criar um objeto emissor de luz, pois com o efeito de reflexão da luz sobre um objeto, fica bem mais fácil perceber o formato dele.

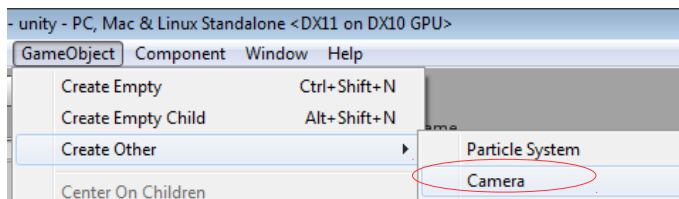


Existem vários tipos de iluminação no Unity. Convido você a experimentar cada um deles. Os objetos de emissão de luz estão localizados na aba: "GameObject → Create Other".

Para que o usuário final do seu simulador, possa ver a sua cena é necessário que essa possua câmera. A câmera define o ponto de onde seu cenário será visto.

Ao criar um cenário, este já vem por padrão com uma câmera chamada de “Main Camera”, você pode usa-la ou criar outras câmeras.

Para criar uma câmera basta selecionar a opção da aba: “GameObject → Create Other → Camera”.

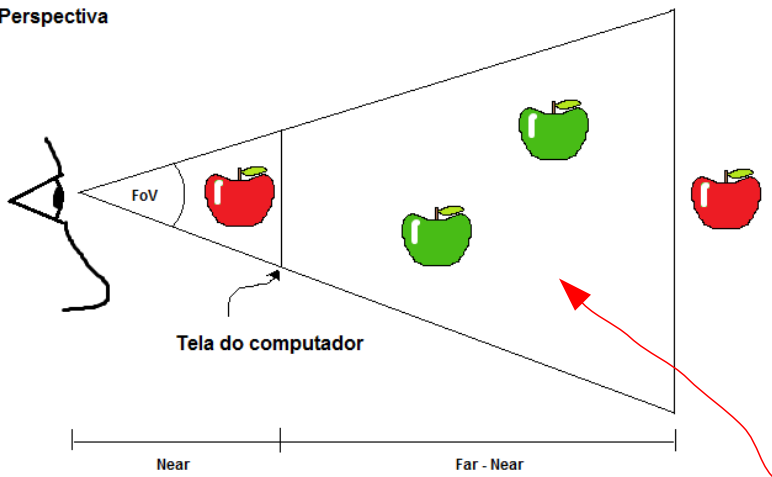


Existem dois tipos de câmera: perspectiva e ortográfica, abordaremos apenas a perspectiva por ser mais usada para simulação (Pela proximidade com a visão humana).

As principais configurações de uma câmera perspectiva são:

- FoV (Field of View): O ângulo de visualização.
- Near: Distância mínima captada pela câmera.
- Far: Distância máxima captada pela câmera.

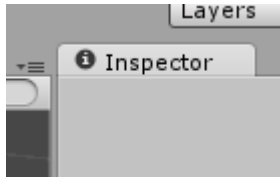
#### Perspectiva



Somente as maçãs Verdes serão captadas pela câmera



Ao selecionar um objeto, note que aparecerá um conjunto de configurações (geralmente à direita da tela) numa aba chamada “Inspector”. Estas configurações são os componentes do objeto.



Os componentes de um objeto são elementos que definem o comportamento do mesmo. Existem vários tipos de componentes, como por exemplo o “RigidBody” que define propriedades físicas do objeto, ou o “Collider” que define a forma do objeto, para que este possa colidir com outros.

Uma câmera é apenas um objeto com o componente “Camera”, este componente possui todos os atributos necessários para configurar uma câmera.

Principal componente da câmera

Define a cor de fundo da tela

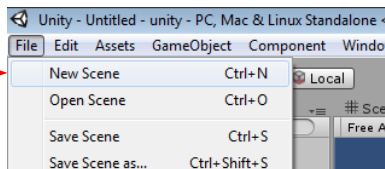
A screenshot of the Unity Camera component Inspector panel. The 'Camera' component is selected, and its settings are displayed. Red arrows point from the text boxes on the left to specific settings: one points to the 'Camera' component header, and another points to the 'Background' color field. The 'Projection' is set to 'Perspective', 'Field of View' is 60, 'Clipping Planes' are set to 'Near' 0.3 and 'Far' 1000, 'Viewport Rect' is (0, 0, 1, 1), 'Depth' is -1, 'Rendering Path' is 'Use Player Settings', 'Target Texture' is 'None (Render Textu)', 'Occlusion Culling' is checked, and 'HDR' is unchecked. The 'Background' color field is highlighted with a blue border.



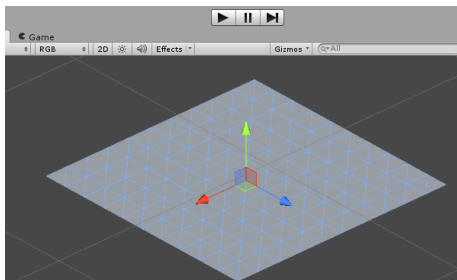
Lembre-se de posicionar a câmera corretamente!!  
Para que seu usuário tenha uma melhor visão da cena.

Já que entramos no assunto de componentes... Vamos criar passo a passo um cenário com uma câmera em queda livre usando os conhecimentos mostrados nessa seção!

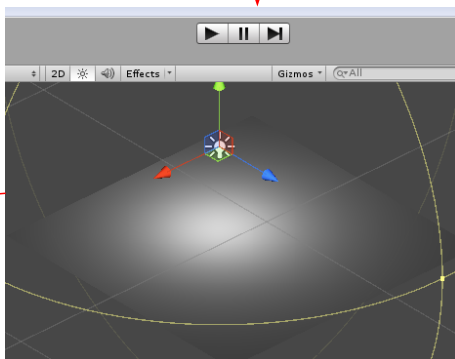
1- Crie uma nova cena



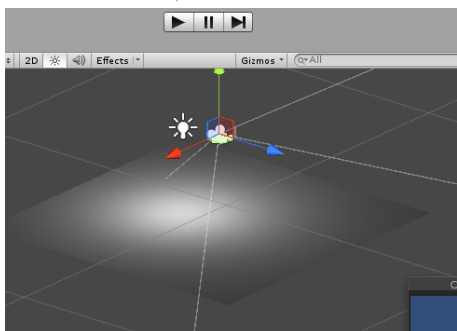
2 – Crie um plano para servir como solo  
(na aba: “GameObject → Create Others → Plane”)



3- Crie uma iluminação  
e posicione sobre o plano



4 – Posicione a  
“Main Camera”  
acima o plano



5 – Execute a cena





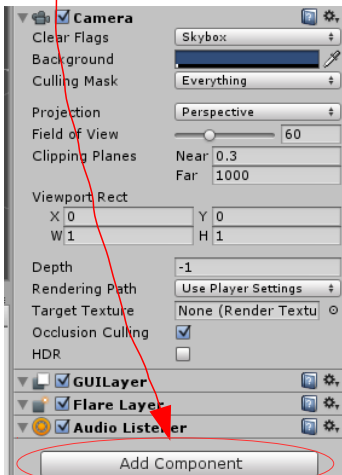
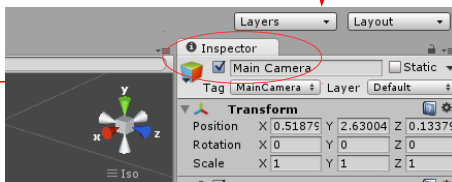
Executei a cena...  
Mas nada aconteceu!??

**É essa a idéia!**

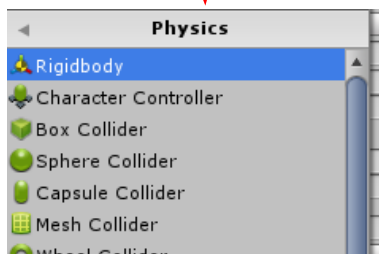
Ainda não adicionamos comportamento ao objeto! Vamos fazer isso agora!!

Selecione a “Main Camera” e adicione um componente Rigidbody a ela.  
Para isso vá na aba “Inspector” e siga os seguintes passos:

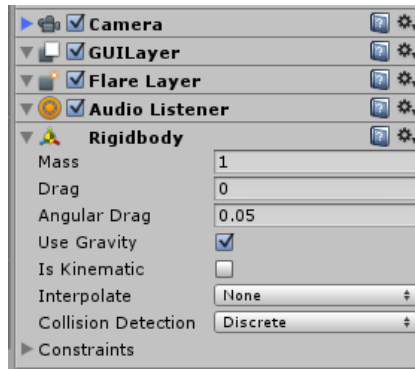
Clique no botão  
“Add Component”



Selecione a aba:  
“Physics → Rigidbody”



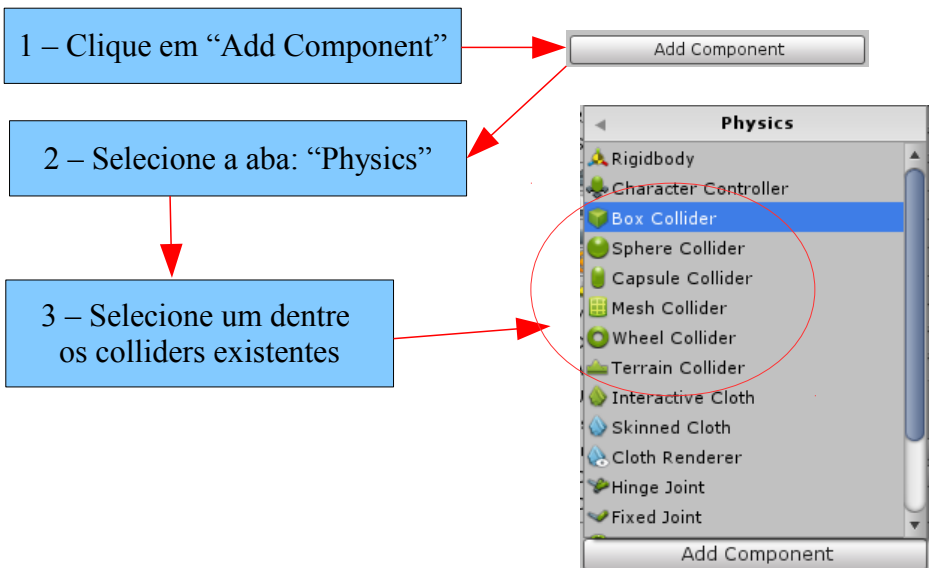
Note que se você executar o cenário com essas alterações, a câmera cairá. Isso ocorre porque o componente “Rigidbody” é o responsável por aplicar Física ao objeto. Se você observar os atributos do componente são todos associados à Física.



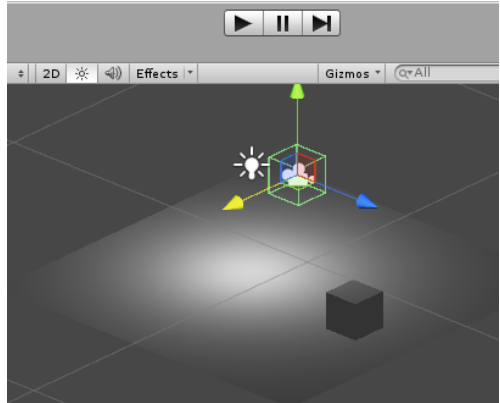
Certo, nossa câmera possui um comportamento... Mas ela ao cair atravessa o chão!!!

Isso ocorre porque, apesar da câmera ser aplicada à física, ela não possui uma forma lógica, ou seja, um colisor.

Para resolver esse problema adicione um componente “Collider” ao objeto “Main Camera”. Para isso siga os seguintes passos:



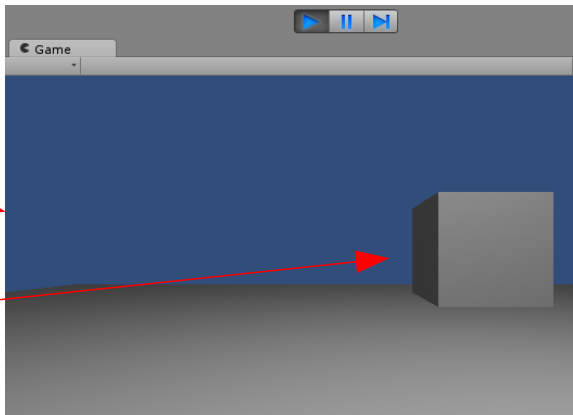
Feito esses passos o cenário deve ficar mais ou menos assim



E ao executar...



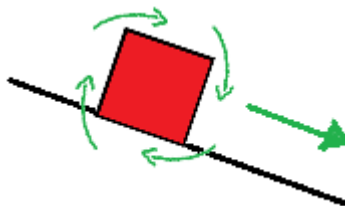
\*Adicionei também este objeto para ter um ponto de referência



Note que o “Collider” define a forma lógica do objeto, ou seja, como a física do Unity interpreta ele.

A forma visual do objeto é dada pelo componente “Mesh Filter” e “Mesh Renderer”, ou seja, estes definem o que o usuário verá na tela.

Num exemplo simples... Um objeto com “Mesh Filter” de cubo e “Collider” de esfera é bem estranho porque ele vai ser visto como um cubo que rola.



## 1.2 – Scripts com Unity

Na seção anterior descobrimos como adicionar comportamentos predefinidos aos nossos objetos. Mas e se quisermos adicionar comportamentos que não estão disponíveis nos componentes padrões do Unity? Nesse caso necessitaremos criar um novo script para nossos objetos.

O script é um código escrito em uma linguagem de programação, que no nosso caso vai criar um comportamento para determinados objetos aos quais associarmos o script.

**Está na hora de Programar!**

O Unity possui um editor de scripts embutido, o MonoDevelopment, de forma que na interface do Unity basta clicar duas vezes sobre um script qualquer que irá abrir uma janela de edição de scripts. As linguagens de programação que podem ser utilizadas atualmente para o Unity são: Boo, JavaScript e C#. Utilizaremos C# como padrão para este livro.

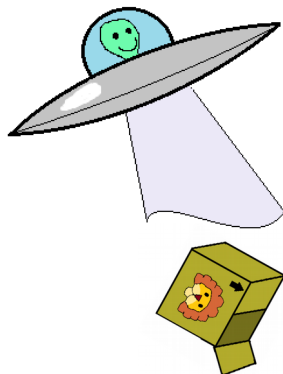
**Então vamos fazer um Projeto!!**

O projeto é o seguinte, os engenheiros da Área 52 estão precisando de um simulador de abdução e te contratam para o trabalho de desenvolvimento.

Como o Unity não tem um script de comportamento de “Nave Abductora” nós teremos que criá-lo!

As especificações da cena são:

- Deve possuir um objeto voador que se movimenta pelo cenário.
- Quando este passar sobre um objeto com área menor que 3x3 deve puxá-lo para cima.
- Quando o objeto puxado chegar a uma certa altitude deve sumir.



Para a execução do projeto siga os seguintes passos:

1- Crie um novo cenário com iluminação

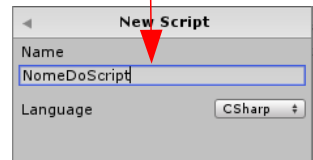
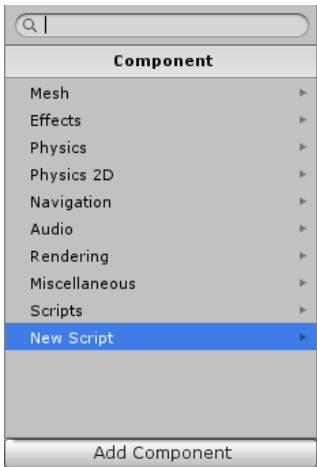
2- Crie um plano 5x5 e posicione na origem do cenário

3- Crie um objeto para ser a nave espacial

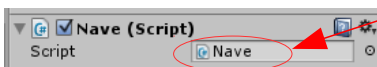
4- Adicione à nave o componente “New Script”



5- Insira o nome do script “Nave”



6- Clique 2 vezes no Componente Nave que foi Criado, para abrir o editor



7- Crie a programação da especificação



Antes de ver as respostas...  
Pense, pesquise, Crie!!  
Não vicie seu cérebro a  
respostas prontas.



Um dos possíveis comportamentos da nave pode ser descrito da seguinte maneira:

```
using UnityEngine;
using System.Collections;

public class Nave : MonoBehaviour {

    //variaveis que definem a abducao
    /*As variaveis publicas podem ser alteradas na
    * interface do Unity!!!
    */
    public float velocidade = 0.05f;
    public float potenciaAbducao = 0.1f;
    public float diferencaAbducaoCompleta = 1.2f;
    public float distanciaAbducao = 0.5f;

    private GameObject objetoSendoAbduzido = null;

    // Esse metodo eh chamado sempre que o objeto eh instanciado
    void Start () {

    }

    // Esse metodo eh chamado a todo ciclo de execucao
    void Update () {
        bool moveu;
        moveu = checarMovimentacao();
        if (moveu) {
            objetoSendoAbduzido = null;
        } else {
            if(objetoSendoAbduzido == null){
                procurarVitima ();
            }else{
                continuarAbducao();
            }
        }
    }

}
```



// Esse metodo eh chamado a todo ciclo de execucao

```
void Update () {  
    bool moveu;  
    moveu = checarMovimentacao();  
    if (moveu) {  
        objetoSendoAbduzido = null;  
    } else {  
        if(objetoSendoAbduzido == null){  
            procurarVitima ();  
        }else{  
            continuarAbducao();  
        }  
    }  
}
```

// movimenta a nave dado a entrada do teclado

```
bool checarMovimentacao(){  
    float x, y;  
    x = Input.GetAxis("Horizontal")*velocidade;  
    y = Input.GetAxis("Vertical")*velocidade;  
    Vector3 posicao = this.transform.position;  
    posicao.x += x;  
    posicao.z += y;  
    this.transform.position = posicao;  
  
    return (x != 0 || y != 0);  
}
```



//verifica se a posicao e o tamanho do objeto possibilitam a abducao

```
bool checarLocalizacaoAbducao(GameObject candidatoVitima){  
    if (candidatoVitima == this.gameObject) {  
        return false;  
    }  
  
    Vector3 tamanhoVitima = candidatoVitima.transform.lossyScale;  
    Vector2 posicaoVitima;  
    Vector2 posicaoNave;  
  
    posicaoVitima.x = candidatoVitima.transform.position.x;  
    posicaoVitima.y = candidatoVitima.transform.position.z;  
  
    posicaoNave.x = this.transform.position.x;  
    posicaoNave.y = this.transform.position.z;  
  
    float diferencaY = this.transform.position.y - candidatoVitima.transform.position.y;  
  
    if((posicaoNave - posicaoVitima).magnitude < distanciaAbducao &&  
        diferencaY > 0){  
        if(tamanhoVitima.x * tamanhoVitima.z < 3*3){  
            return true;  
        }  
    }  
    return false;  
}
```

```

// abduz o objeto
void continuarAbducao(){
    if (objetoSendoAbduzido.rigidbody != null) {
        objetoSendoAbduzido.rigidbody.velocity = Vector3.zero;
    }

    Vector3 posicaoNave = this.transform.position;
    Vector3 posicaoVitima = objetoSendoAbduzido.transform.position;

    float y = posicaoVitima.y;
    posicaoVitima += (posicaoNave - posicaoVitima)*potenciaAbducao;
    posicaoVitima.y = y + potenciaAbducao;

    objetoSendoAbduzido.transform.position = posicaoVitima;

    print (posicaoNave.y - y);
    print (diferencaAbducaoCompleta);

    if (posicaoNave.y - y < diferencaAbducaoCompleta) {
        GameObject.Destroy(objetoSendoAbduzido); //Remove obj do cenario
        objetoSendoAbduzido = null; //Abducao completa!
    }
}

//Varre o cenario checando qual vitima esta no alcance da abducao
void procurarVitima(){

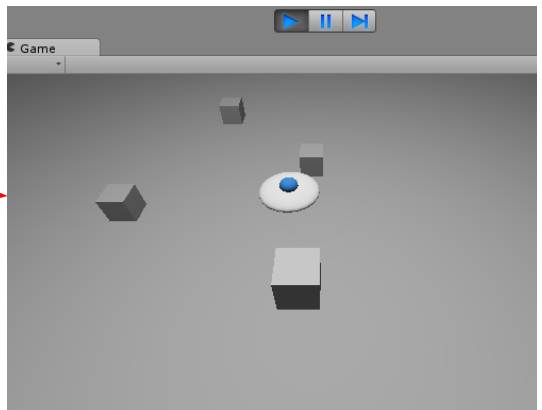
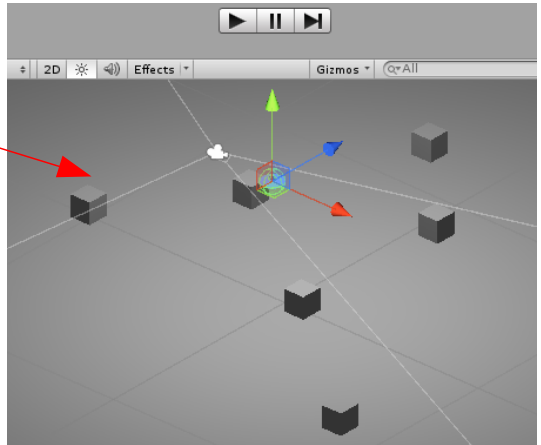
    GameObject [] todosObjetos =
    (GameObject [])   GameObject.FindObjectsOfType(typeof(GameObject));

    int i;
    for (i = 0; i < todosObjetos.Length; i++) {
        bool abduzivel;
        abduzivel = checarLocalizacaoAbducao(todosObjetos[i]);
        if(abduzivel){
            objetoSendoAbduzido = todosObjetos[i];
        }
    }
}
}

```

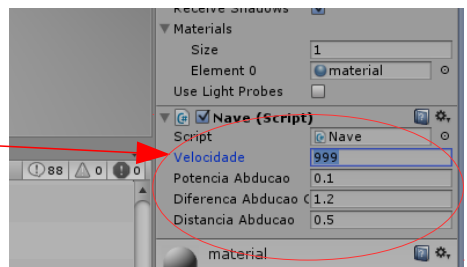
8- Posicione cubos pelo cenário

9- Execute a cena e tente Capturar alguns cubos!!



Se você criou algumas variáveis públicas. Note que elas podem ser modificadas na própria janela do Unity, a qualquer momento mesmo se o cenário estiver sendo executado!!

Variáveis públicas da Nave



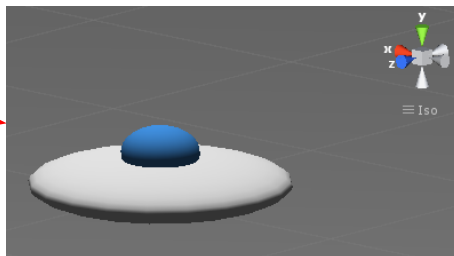
### 1.3 – Criando prefabs

Ao criar um cenário criamos vários objetos com diferenciados comportamentos, vários desses objetos bem complexos. Em grande parte dos casos gostaríamos de reutilizar estes objetos que criamos em outros cenários, sem ter que reconstruí-los.

Por isso no Unity existe um conceito muito útil para este problema, chamado de prefab. Um prefab é um objeto que foi criado anteriormente, e foi salvo para ser usado em cenários futuros.

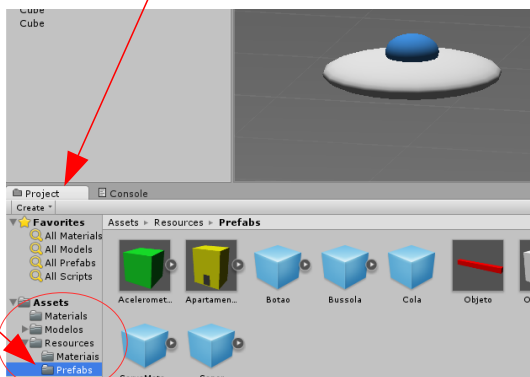
Suponha que eu  
tive muito trabalho  
para fazer isso

E quero salvá-lo  
ara usá-lo em  
cenários futuros



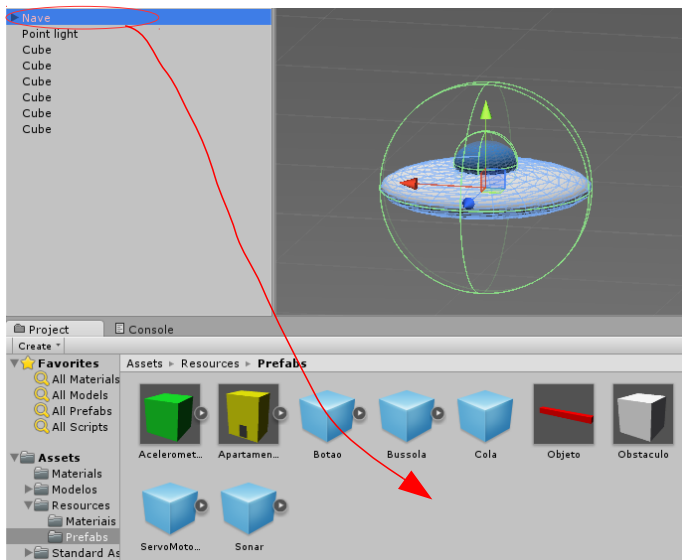
Para isso basta:  
1- Selecionar uma pasta no projeto na aba “Project”

Já existe uma padrão  
(/Resources/Prefabs)



2- Clicar e arrastar o objeto para a aba

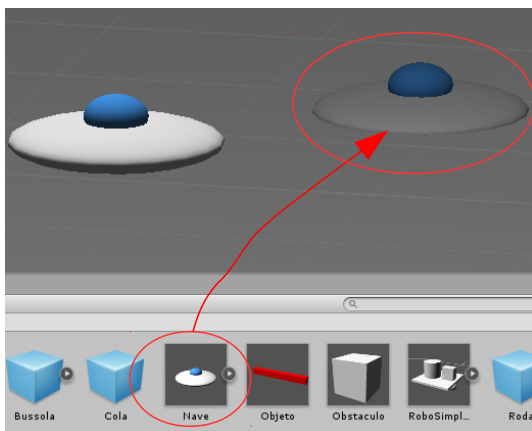
Feito isso o prefab está criado.



Depois de criado o prefab...  
Como faço para reutilizá-lo?



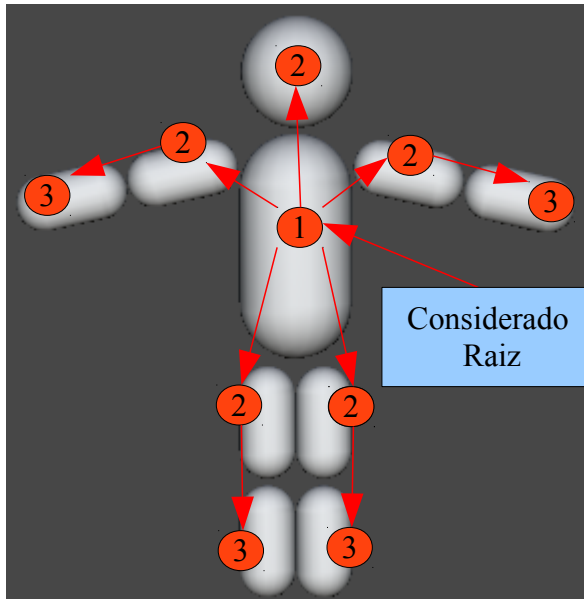
Basta clicar e arrastar!!!



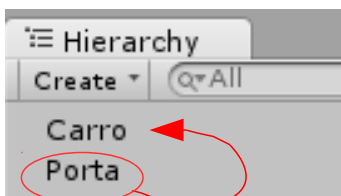
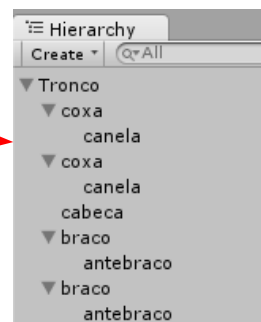
Uma característica importante para um prefab é a ideia de hierarquia. A hierarquia no Unity significa que os objetos podem ser organizados de forma que qualquer movimento que ocorra a um objeto na hierarquia superior, será transmitido aos seus objetos subordinados na hierarquia. Isso possibilita a criação de objetos complexos, com maior facilidade.

Numa analogia com o corpo humano:

- 1 - Tudo que o tronco faz  
O resto do corpo faz.
- 2- Os braços e pernas,  
controlam as suas  
respectivas extremidades.
- 3- As extremidades são  
Apenas subordinadas.

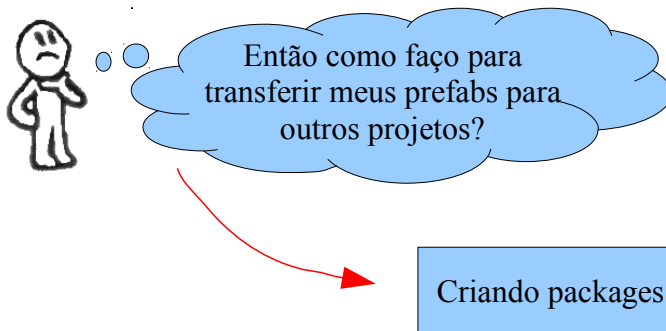


Na interface do Unity  
A hierarquia seria na forma:



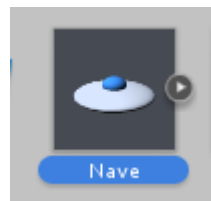
Para construir uma hierarquia basta clicar e arrastar o nome do objeto para seu respectivo superior.

Já vimos que o prefab facilita bastante o desenvolvimento de um projeto, por proporcionar o reaproveitamento do trabalho feito de cenários anteriores para cenários atuais. Porém existe um pequeno problema, um prefab de um projeto não pode ser utilizado em outro.

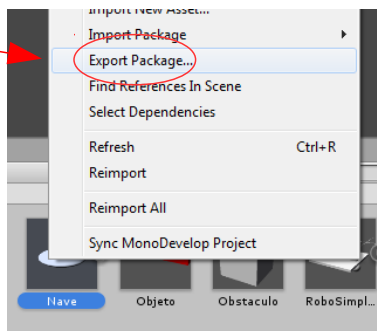


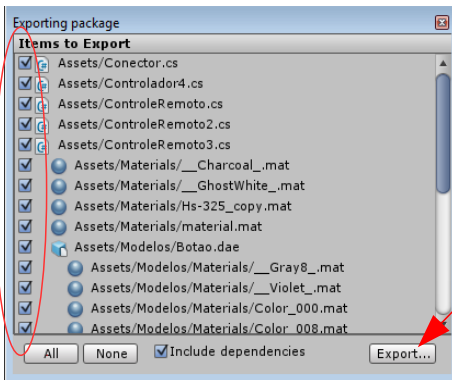
Para criar um package siga os seguintes passos:

1 - Selecione os prefabs que deseja exportar



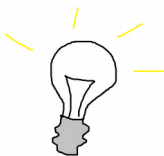
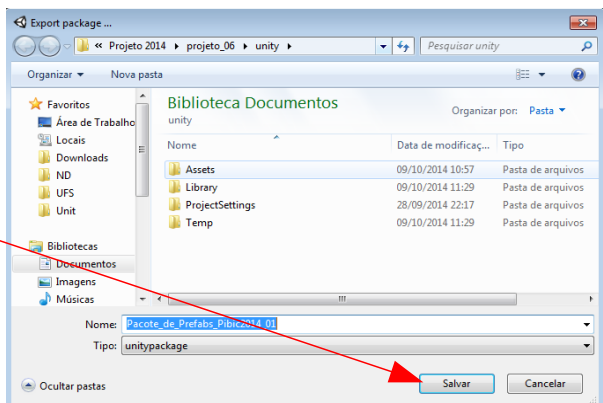
2 – Clique com o botão direito do mouse e selecione a opção



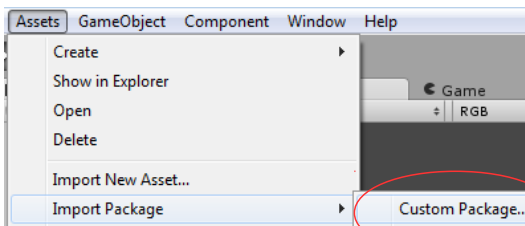


3- Selecione todo elemento que for vinculado ao seus Prefabs (Scripts, texturas, materiais...)

4- Salve no diretório que achar mais conveniente

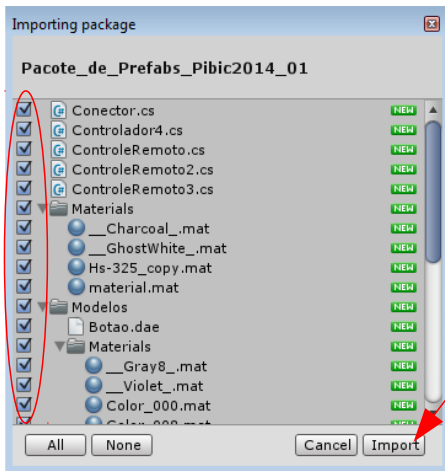
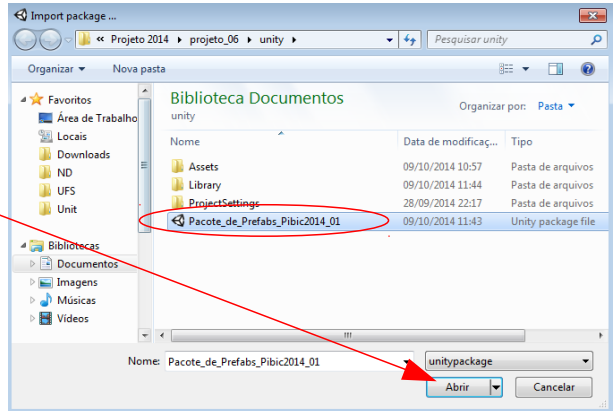


Para importar um package não padrão do Unity num outro projeto basta selecionar a opção da aba: “Assets → Import Package → Custom Package” e seguir os seguintes passos:





1- Vá ao diretório onde o package está salvo e abra-o.



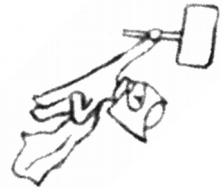
2 - Selecione os elementos que deseja que sejam importados ,com atenção, pois alguns destes são interdependentes.

## 1.4 – Exercícios de Fixação

1 – Faça um Cubo que role sobre um plano Inclinado como se fosse uma esfera



2 – Faça um tipo de câmera que possa ser rotacionada pelo mouse de forma que, o centro de rotação dela seja o objeto que ela está vinculada



3 – Faça um jogo de puzzle, onde seu personagem, deve pular sobre uma sequência de cubos para atravessar o cenário sem cair. Se cair, deve reiniciar o puzzle

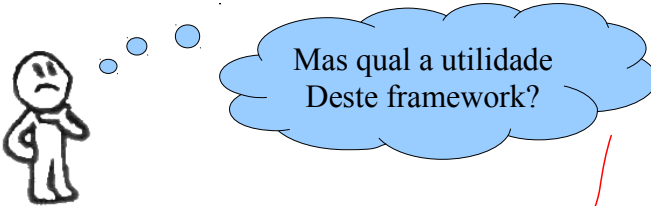
4 – Faça um prefab helicóptero que possa ser controlado pelo teclado, com os comandos de acelerar, desacelerar, e direcionar



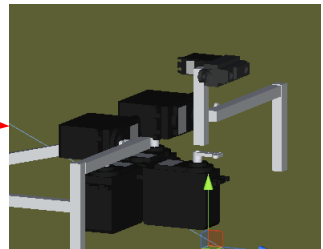
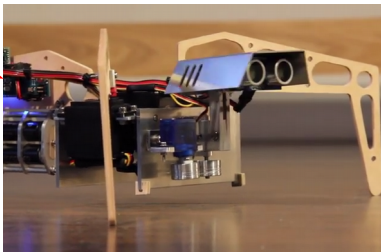
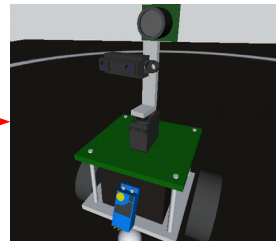
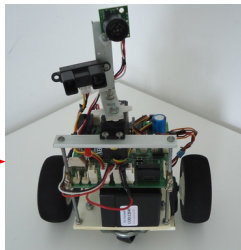
5 – Crie um script que dado um comando do Teclado, gere um objeto em uma posição aleatória do cenário

## 2º Capítulo: O Framework

Neste capítulo será introduzido o uso framework proposto por este livro. Aqui será abordado aspectos desde configuração a desenvolvimento de pequenos projetos para ajudar a fixar o conhecimento.



Foi desenvolvido com o objetivo  
de facilitar o desenvolvimento de  
Simuladores aplicados à robótica!

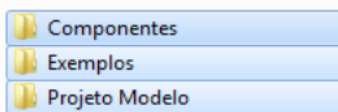


## 2.1 – Configurando o Framework

O primeiro passo para que possamos utilizar o framework é instalá-lo. Para isso basta acessar o seguinte link, e efetuar o download da pasta compactada lá presente:

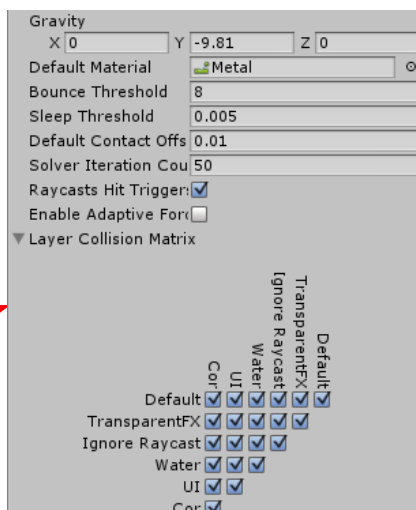
<https://www.dropbox.com/sh/2gx2egr3plnwkpm/AABtuv4rxX6F1LN9t19cO0VTa?dl=0>

Dentro da pasta compactada do framework existem 3 pastas: A primeira possui um conjunto de packages que contém prefabs de componentes robóticos, outra pasta possui um conjunto de exemplos de simuladores simples produzidos com a ajuda do framework, e a última pasta é um projeto modelo, já configurado, para que você possa pular as etapas de configuração de projeto e importação de packages.



Caso você queira criar um novo projeto sem utilizar o projeto modelo, é necessário que você configure-o para que seja compatível com o framework.

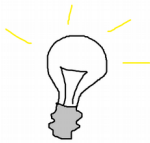
Para isso será necessário modificar a configuração de Física para a seguinte:



Também será necessário modificar a configuração de tempo para a seguinte:

Fixed Timestep	0.002
Maximum Allowed Time Step	0.3333333
Time Scale	1

Feito isso o projeto está pronto para utilizar o framework!!



Para utilizar os prefabs do framework basta importar os packages da pasta de componentes. cada package representa um prefab e estes serão descritos na próxima seção.

## 2.2 – Prefabs do Framework

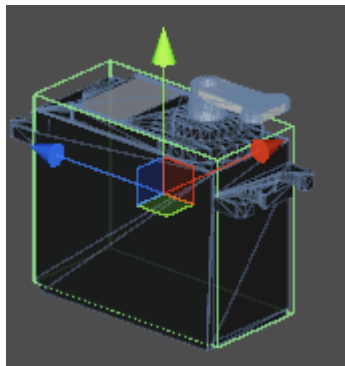
Os prefabs contidos no framework são um conjunto de sensores e atuadores utilizados frequentemente em projetos de robótica móvel que foram modelados para que possuíssem a função arquitetural de sua versão real. Os prefabs são os seguintes:

Servo motor limitado  
Servo motor de rotação  
Cola  
Sensor Infravermelho  
Sonar  
Sensor RGB  
Acelerômetro  
Botão  
Bússola  
Roda

Descreveremos nessa seção cada um destes prefabs.

## Servo motor Limitado

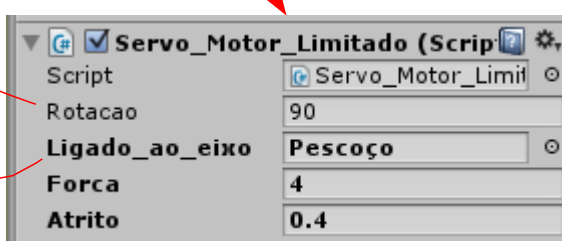
O Servo-motor limitado é um prefab que possui um eixo ao qual conecta-se objetos. A função do servo-motor limitado é rotacionar os objetos conectados ao seu eixo, direcionando-o para o ângulo especificado, onde este ângulo pode variar entre 0° até 180°.



As configurações possíveis do servo motor são as seguintes:

Direção a rotacionar

Objeto a ser rotacionado

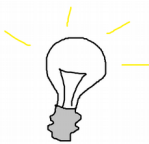
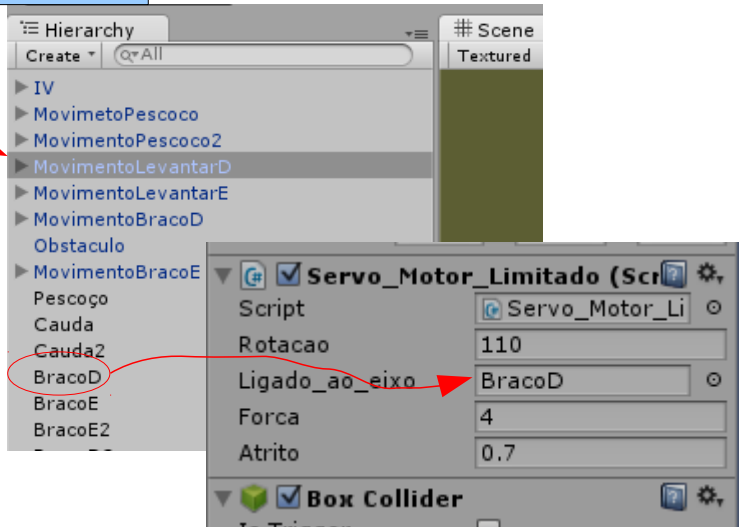


Descrição detalhada:

- **Rotacao:** é um atributo público que determina a direção a qual o objeto ligado ao eixo deverá ser rotacionado.
- **Ligado\_ao\_eixo:** é um atributo público que define o objeto a ser rotacionado pelo servo motor.
- **Força:** é um atributo público que indica o torque que o motor aplicará para rotacionar o objeto ligado ao eixo.
- **Atrito:** é um atributo público que indica a perda de torque do servo padrão.

Para definir o objeto que o servo motor irá rotacionar basta:

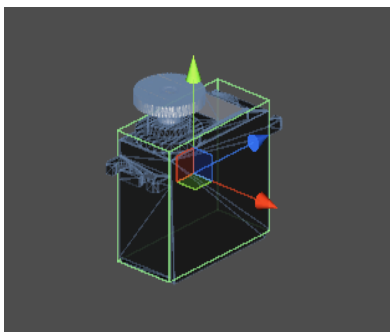
Selecionar o objeto com o mouse, e arrastar para o atributo ligado\_ao\_eixo do servo motor



Como exercício de fixação monte um cenário simples com um servo motor limitado, e conecte a ele um paralelogramo de massa 20, feito isso rotacione o paralelogramo modificando os atributos de força e atrito. Verifique os resultados.

## Servo motor de rotação

O Servo motor de rotação é um prefab que possui um eixo ao qual conecta-se objetos os quais se deseja obter uma rotação contínua. A função deste prefab é rotacionar os objetos conectados ao seu eixo em um determinado sentido (1: horário ou 0: anti-horário) a uma determinada velocidade configurável.



Um exemplo de uso deste prefab é para rotação de rodas em robôs móveis.

As configurações possíveis do servo motor de rotação são:

Velocidade angular

Objeto a ser rotacionado



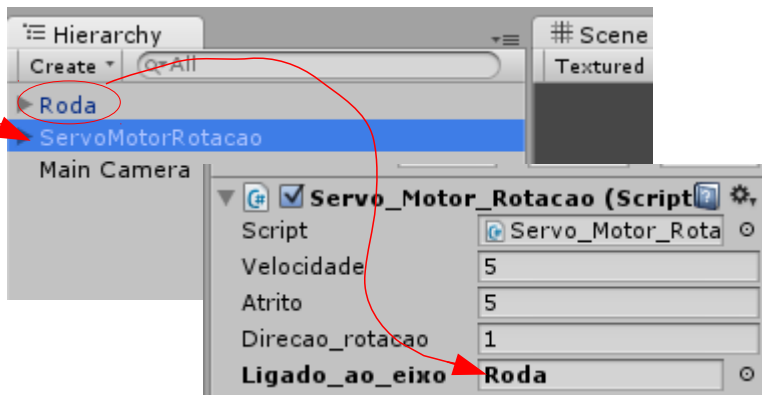
Descrição detalhada:

- **Velocidade:** é um atributo público que determina a velocidade a qual o objeto ligado ao eixo deverá ser rotacionado.
- **Força:** é um atributo público que indica o torque que o servo motor possui.
- **Direcao\_rotacao:** é um atributo público que indica o sentido de rotação do eixo do motor.
- **Ligado\_ao\_eixo:** é um atributo público que define o objeto a ser rotacionado pelo servo motor.



Para definir o objeto que será rotacionado basta:

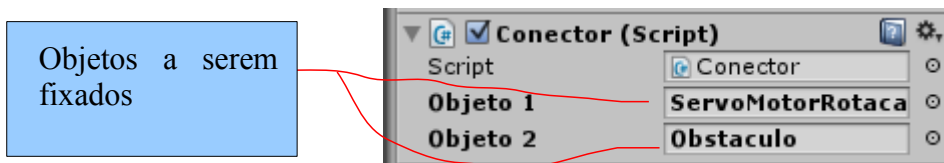
Selecionar o objeto com o mouse, e arrastar para o atributo ligado\_ao\_eixo do servo motor



## Cola

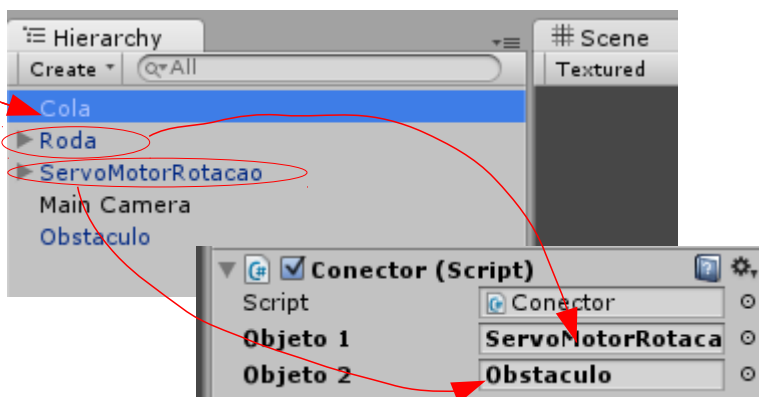
O prefab cola é um prefab auxiliar que tem a função de fixador. Seu objetivo é fazer com que os dois objetos, especificados em seus atributos, fiquem fixados um em relação ao outro.

Este prefab possui como atributos, apenas os objetos a serem fixados.



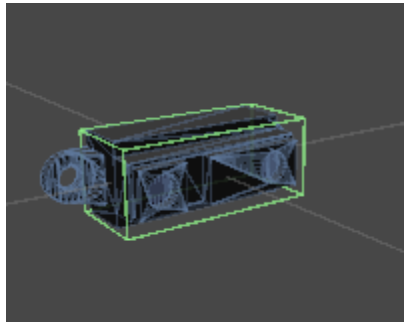
Para definir os objetos a serem fixados basta:

Selecionar o objeto com o mouse, e arrastar para o atributo ligado\_ao\_eixo do servo motor



## Sensor Infravermelho

O prefab Sensor Infravermelho é um prefab cujo objetivo é medir a distância de um objeto a sua frente, se este objeto estiver dentro de seu alcance.

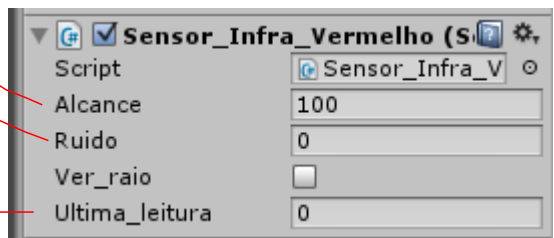


As configurações possíveis do Sensor Infravermelho são:

Alcance de leitura (cm)

Ruído de leitura

Último valor sensoreado

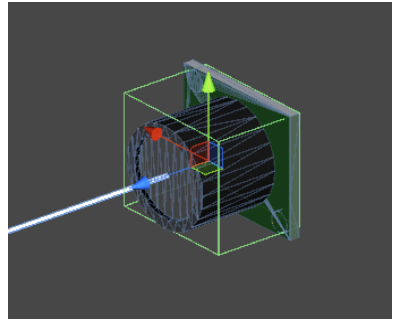


Descrição detalhada:

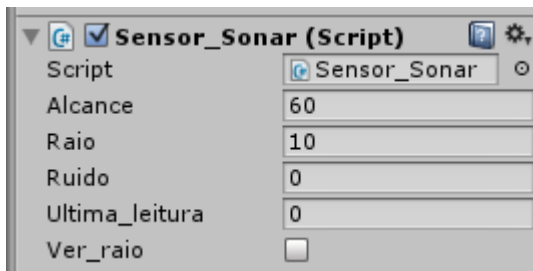
- Alcance: é um atributo público que indica o valor máximo de distância que o sensor consegue detectar.
- Ruído: é um atributo público que indica a taxa de desvio de leitura que o sensor sofre .
- Ver\_raio: é um atributo público que mostra/esconde um raio que serve de debug para indicar o que o sensor está detectando.
- GetDistancia(): é um método público que faz a leitura do sensor e retorna a distância por ele sensoreada.
- Ultima\_leitura: é um atributo público que representa o ultimo valor que foi sensoreado pelo sensor.

## Sonar

O sonar é um prefab que tem função de informar a distância de objetos que estão a sua frente. Possui a singularidade de capturar a distancia de objetos num certo volume a sua frente o que faz com que possua um erro associado relativamente alto.



As configurações possíveis do prefab são:

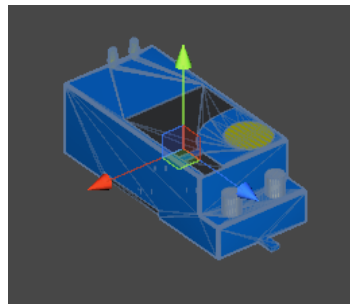


Descrição detalhada:

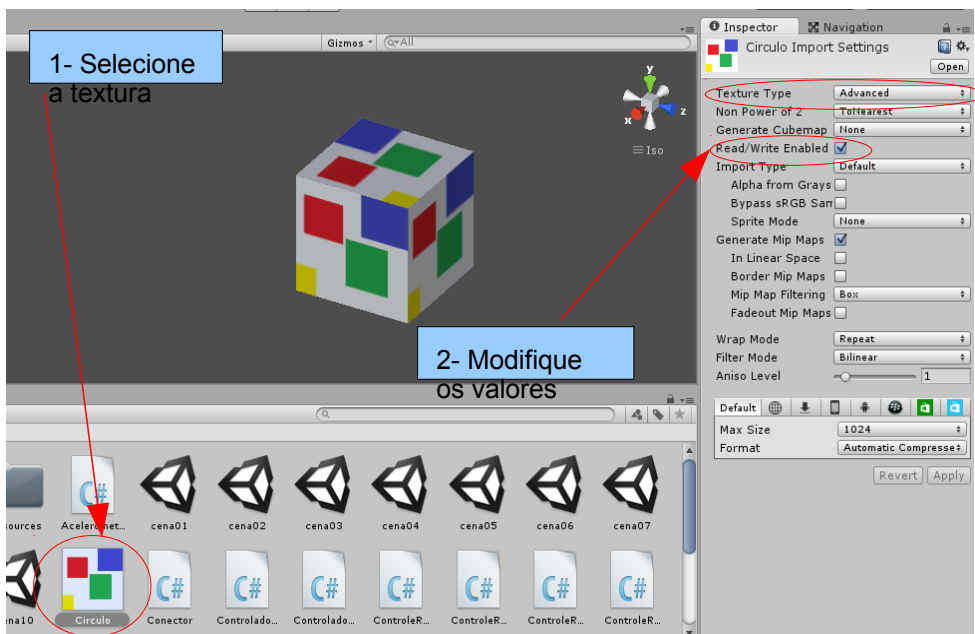
- Alcance: é um atributo público que indica o valor máximo de distância que o sensor consegue detectar.
- Raio: Indica o raio maior do cone de leitura (volume de leitura).
- Ruido: é um atributo público que indica a taxa de desvio de leitura que o sensor sofre .
- Ver\_raio: é um atributo público que mostra/esconde um raio que serve de debug para indicar o que o sensor está detectando.
- GetDistancia(): é um método público que faz a leitura do sensor e retorna a distância por ele sensoreada.
- Ultima\_leitura: é um atributo público que representa o ultimo valor que foi sensoreado pelo sensor.

## Sensor RGB

O sensor RGB é um prefab que tem função de informar a cor em escala vermelho, verde, e azul, de objetos que estão a sua frente (considerando como a frente a coordenada local -Z).

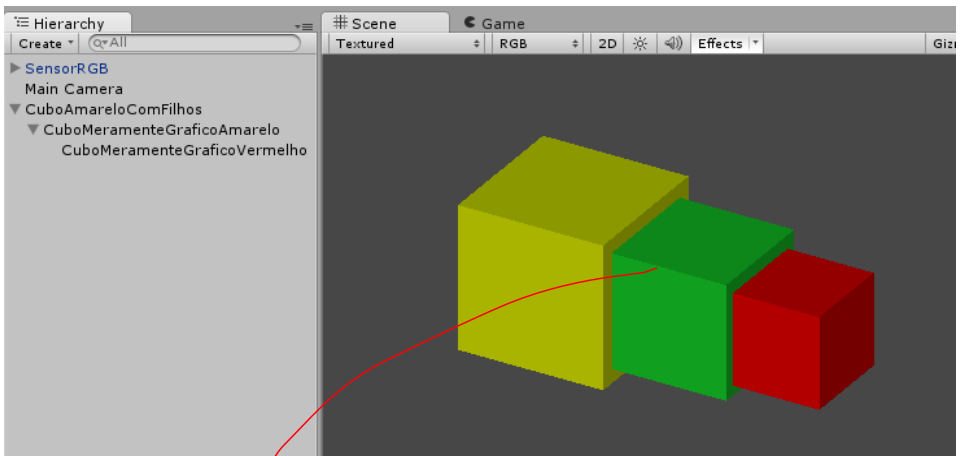


Este sensor possui dois casos para detecção das cores do objeto (Considere por objeto qualquer “GameObject” que possuir um componente “Collider”). O primeiro é o caso em que o objeto é baseado em textura, neste caso é necessário que ao adicionar a textura ao projeto, modifique-se o valor do atributo “Texture Type” para “Advanced” e marcar o atributo “Read/Write Enabled” como segue:



O segundo caso é quando objeto não possui textura, dessa forma o algoritmo do sensor infravermelho irá fazer uma busca na hierarquia do objeto até encontrar um componente do tipo “MeshRenderer”, este componente é responsável por guardar a cor de determinada parte de um objeto.

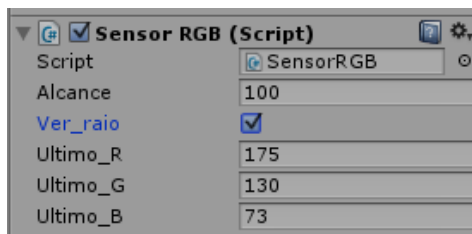
É importante notar que neste caso o objeto possuirá apenas uma cor irá representá-lo isto implica que se o objeto possuir varias cores, apenas uma delas será lida. A cor que será lida será a que possuir maior grau na hierarquia do objeto.



Somente o cubo de maior Hierarquia terá a cor sensoreada



As configurações possíveis do Sensor RGB são:

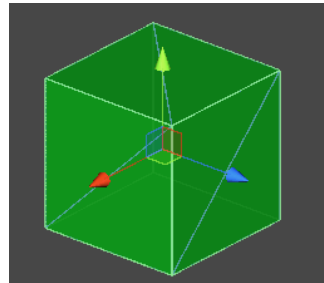


### Descrição detalhada:

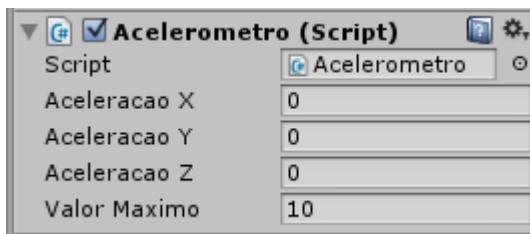
- Alcance: é um atributo público que indica o valor máximo de distância que o sensor consegue detectar cor.
- Ultimo\_R: é um atributo público a magnitude do cromo vermelho na ultima cor sensoreada.
- Ultimo\_G: é um atributo público a magnitude do cromo verde na ultima cor sensoreada.
- Ultimo\_B: é um atributo público a magnitude do cromo azul na ultima cor sensoreada.
- GetCor(): é um método público que faz a leitura do sensor e retorna um inteiro no formato 0x00RRGGBB.

## Acelerômetro

O acelerômetro é um prefab que tem a função de mensurar as acelerações que estão sendo impostas sobre ele, dividindo em três direções linearmente independentes.



As configurações possíveis do Acelerômetro são:



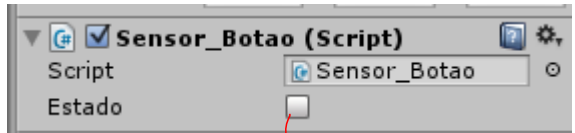
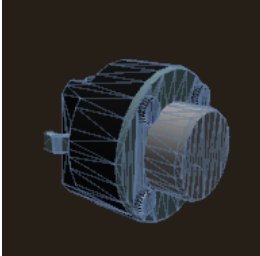
Descrição detalhada:

- Aceleracao X: é um atributo público que indica a aceleração que o acelerômetro está sofrendo em relação a sua coordenada local X.
- Aceleracao Y: é um atributo público que indica a aceleração que o acelerômetro está sofrendo em relação a sua coordenada local Y.
- Aceleracao Z: é um atributo público que indica a aceleração que o acelerômetro está sofrendo em relação a sua coordenada local Z.
- Valor máximo: limite de leitura do sensor em relação a cada eixo local.



## Botão

O prefab botão é um verificador de pressão ou colisão. Possui apenas um atributo, estado, que indica se está sendo pressionado ou não.



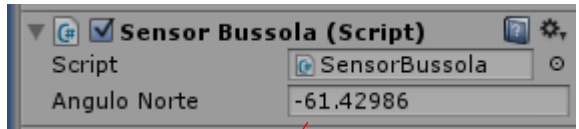
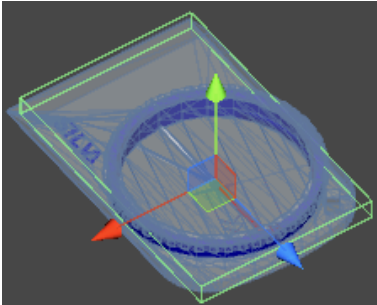
Indica se está sendo pressionado

Descrição detalhada:

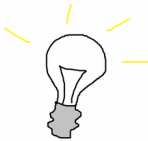
- Estado: é um atributo público que indica se o botão está sendo pressionado ou não.

## Bússola

O prefab bússola tem função de indicar a rotação atual de determinado objeto em relação ao eixo global Y. Dessa forma sua única variável é o ângulo que indica o norte.



Indica quantos graus o objeto está deslocado em relação ao norte



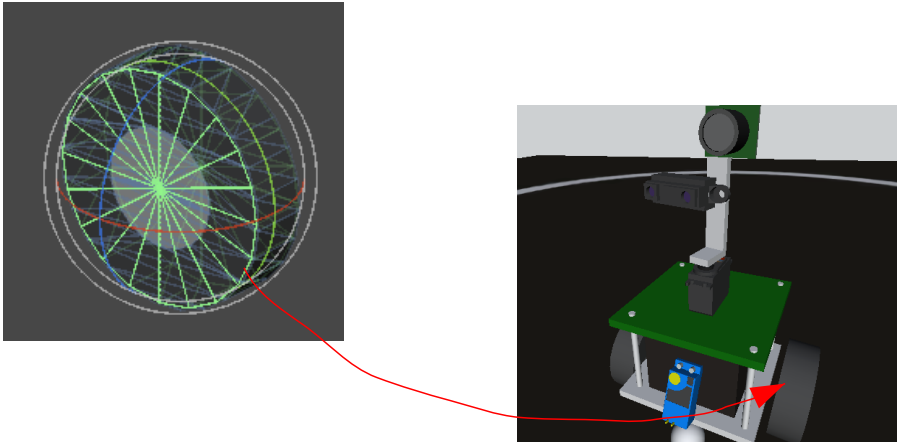
O framework considera como norte o eixo Z global

Descrição detalhada:

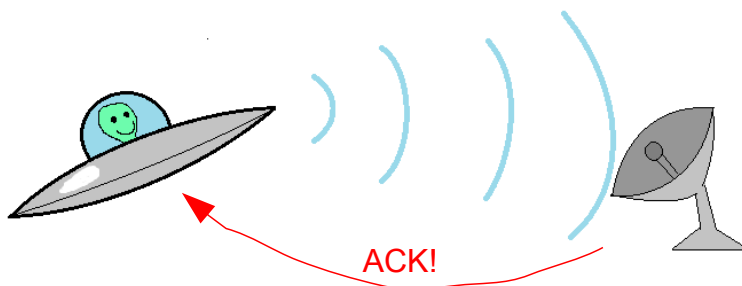
- Angulo Norte: é um atributo público que indica o ângulo que o eixo global Z faz em relação à bússola.

## Roda

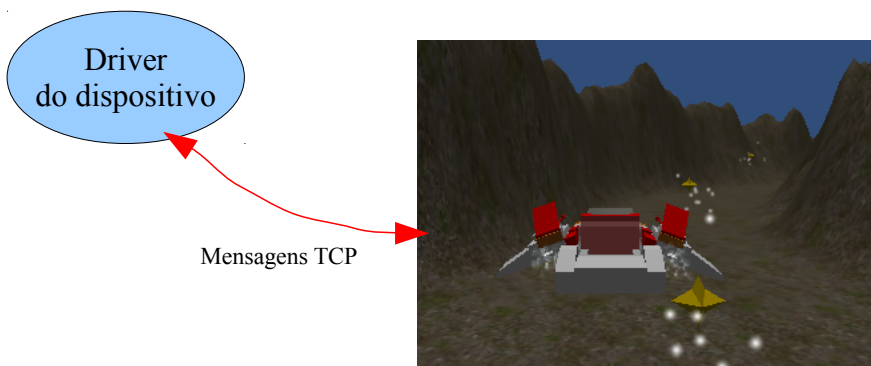
O prefab roda é um prefab auxiliar que possui a função, na maior parte dos casos, de ser rotacionado por um servo motor de rotação para possibilitar o deslocamento de um robô móvel.



## 2.3 – A comunicação cliente-servidor



Uma das propostas do framework é o uso dos prefabs disponibilizados para se comunicar com Hardwares reais. Essa comunicação pode ser possível através da comunicação entre sockets numa rede, onde o driver do hardware é um software a parte que se comunica com o simulador via mensagens em rede.



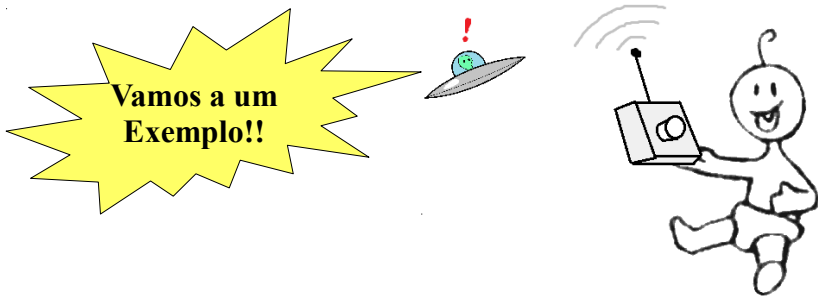
Dessa forma, o simulador desenvolvido, terá uma interface independente do tipo de controle do usuário, pois com a interface de rede, qualquer hardware poderá operar o sistema do simulador, se este possuir o driver de comunicação apropriado.

Essa abordagem também permite uma maior liberdade, no que se refere a compatibilidade, tanto da linguagem de programação quanto sistema operacional, entre o driver do dispositivo e o simulador.

## O protocolo TCP

TCP(Transmission Control Protocol) é o nome do protocolo utilizado para comunicação em rede, quando há a necessidade que a mensagem seja entregue com sua total integridade e ordem.

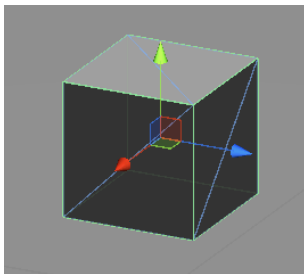
Grande parte dos sistemas operacionais e linguagens de programação possuem suporte a este protocolo, o que o torna praticamente livre de plataforma.



Vamos fazer um veículo de controle remoto. Para isso teremos que usar os seguintes prefabs:

Cola  
Roda  
Servo motor de rotação

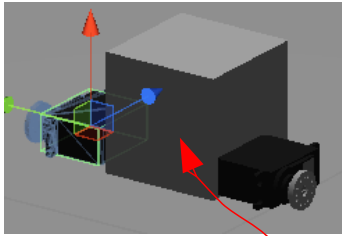
Sendo assim, basta seguir os seguintes passos:



Crie um objeto que sirva como chassi do veículo



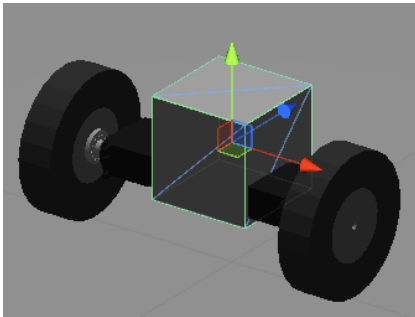
Lembre de configurar o rigidbody do chassi



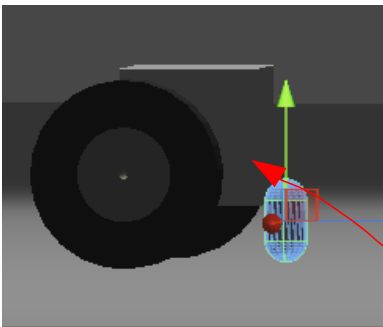
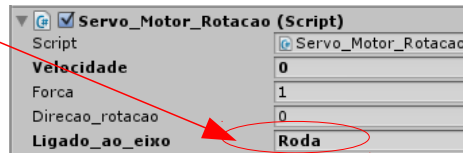
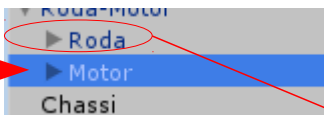
Crie e posicione dois  
Servo motores de  
rotação

Utilizando o prefab Cola,  
fixe os servo motores ao  
chassi

Crie uma roda para cada  
Servo motor, e acople  
ao seu respectivo



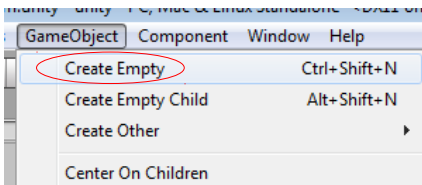
Para acoplar a roda ao servo  
motor, basta seleccionar o servo  
e arrastar o objeto roda para  
o atributo “ligado ao eixo”  
do servo



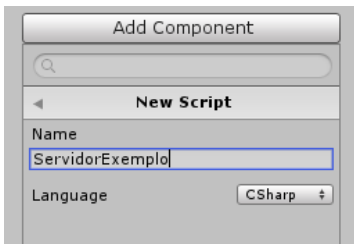
Crie um objeto que  
Sirva como ponto de apoio

Utilizando a Cola,  
Fixe o ponto de apoio  
ao chassi

Para controlar o veículo pela rede, precisamos criar uma interface entre o simulador e a rede. Para isso criaremos um script servidor, seguindo os seguintes passos:



Crie um objeto vazio  
No cenário, que será nosso  
servidor



Adicione ao objeto  
servidor, um componente  
“New Script”

Abra o script que acabou de criar, e insira o seguinte código:

```
using UnityEngine;
using System.Collections;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Globalization;

public class ServidorExemplo : MonoBehaviour {

    private Socket cliente;
    private TcpListener server = new TcpListener(IPAddress.Any, 12598);

    // Use this for initialization
    void Start () {
        server.Start(); // inicia o servidor
    }
}
```

```

void Update () {
    int tamanho = 0; //tamanho da mensagem recebida
    if (server.Pending()){
        cliente = server.AcceptSocket(); //aceita conexao
        cliente.Blocking = false;

        byte[] mensagem = new byte[1024];
        string strMessage = "";
        while(!strMessage.Contains(";")){
            try{tamanho = cliente.Receive (mensagem);
                strMessage = strMessage +
                System.Text.Encoding.UTF8.GetString(mensagem);
            }catch(System.Exception e){}
        }

        strMessage = strMessage.Split(';')[0];

        byte[] envioBuffer = new byte[4];
        envioBuffer[0] = (byte)'a';
        envioBuffer[1] = (byte)'c'; // mensagem a ser enviada ao cliente
        envioBuffer[2] = (byte)'k';
        envioBuffer[3] = 10;
        cliente.Send(envioBuffer);

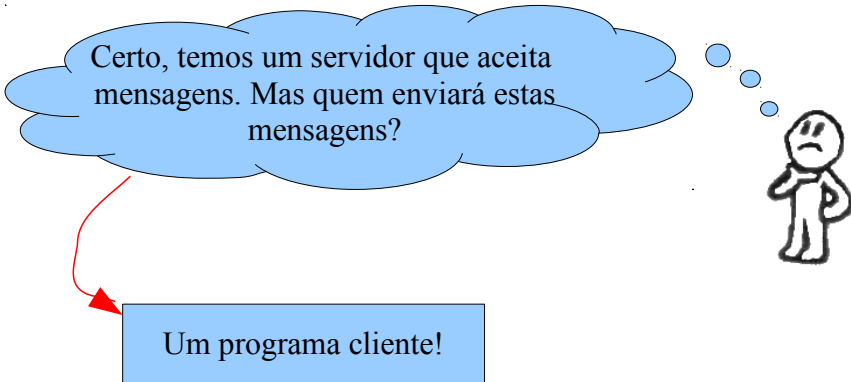
        Servo_Motor_Rotacao [] servos =
            GameObject.FindObjectsOfType<Servo_Motor_Rotacao>();
        for(int i = 0; i < servos.Length; i++)
        {
            if(strMessage.Contains("paraFrente")){
                servos[i].velocidade = 200;
                servos[i].direcao_rotacao = i;
            }else if(strMessage.Contains("paraTras")){
                servos[i].velocidade = 200;
                servos[i].direcao_rotacao = 1-i;
            }else if(strMessage.Contains("parar")){
                servos[i].velocidade = 0;
            }
        }
    }
}

```

Este script inicia um servidor local, que aguarda conexões na porta 12598. Cada conexão deverá enviar apenas uma mensagem terminada com o caractere ';'. Cada mensagem será traduzida a um comando para nosso veículo no cenário. Os possíveis comandos são:

“paraFrente”  
 “paraTras”  
 “parar”





Cliente é o nome dado ao programa que se conecta a um servidor. Dessa forma, o que nosso veículo precisa de um programa cliente que o controle. Para isso usaremos o seguinte código escrito em Python:

```
import socket
import time
import sys

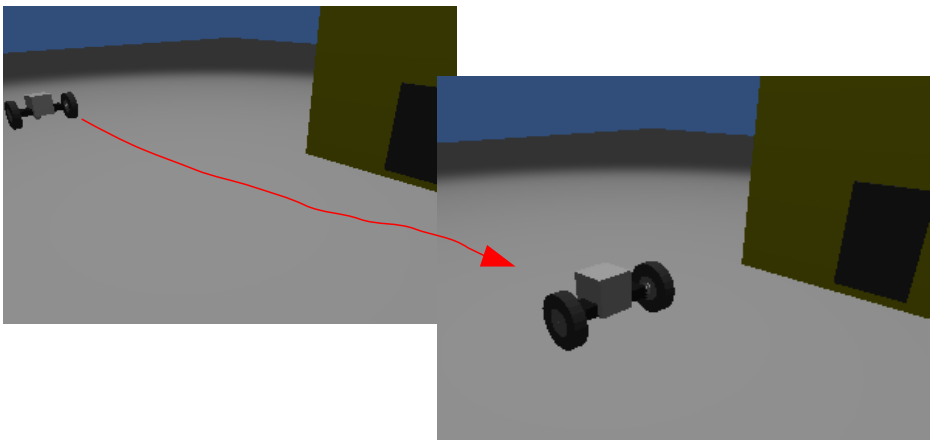
def enviar_para_unity(msg):
    TCP_IP = '127.0.0.1'
    TCP_PORT = 12598 #porta do servidor Unity
    BUFFER_SIZE = 1024
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((TCP_IP, TCP_PORT))
    s.send(msg)
    data = s.recv(BUFFER_SIZE)
    s.close()
    print "mensagem recebida:", data

while(1):
    enviar_para_unity("paraFrente;\n")
    time.sleep(2)
    enviar_para_unity("parar;\n")
    time.sleep(2)
    enviar_para_unity("paraTras;\n")
    time.sleep(2)
    enviar_para_unity("parar;\n")
    time.sleep(2)
```

Este código simples, apenas comanda ao veículo que vá para frente e para trás alternadamente.

Note que este é um exemplo simples por fins didáticos. Este cliente poderia ser um driver de joystick, ou um software de telecontrole, ou até mesmo um driver de sensor de ondas cerebrais! Enfim as possibilidades de uso são diversas.

Para verificar o funcionamento do passo a passo que realizamos basta:



## 2.4 – Ferramentas e Dicas

O uso de ferramentas para diminuir a carga de trabalho no desenvolvimento de determinada atividade é muito útil, pois assim, o tempo que seria perdido em tarefas repetitivas ou muito difíceis, pode ser canalizado para seus principais objetivos.

### O Sketchup

O Sketchup é uma ferramenta de uso livre, para modelagem de objetos 3D. Esta ferramenta é recomendada não só pela simplicidade de uso, mas também por possuir um vasto repositório com diversos modelos 3D que podem ser baixados grátis.



O download do Sketchup pode ser feito através do link:

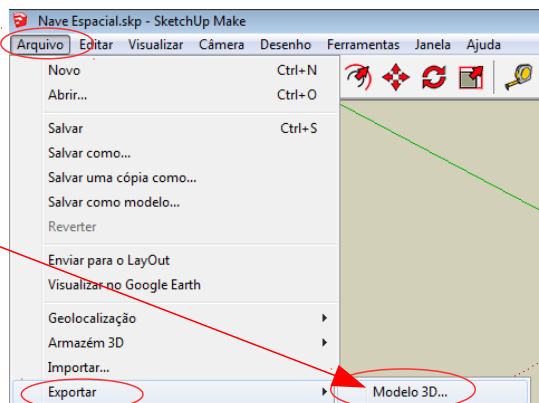
<http://www.sketchup.com/pt-BR/download>

E o próximo link leva ao repositório de objetos 3D para download:

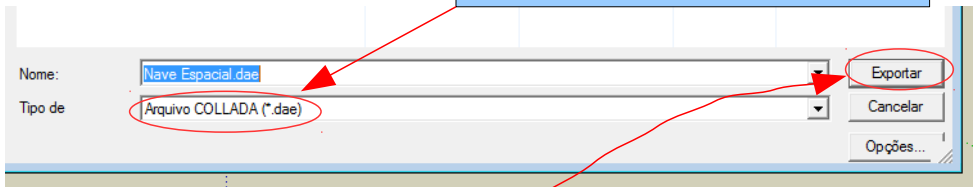
<https://3dwarehouse.sketchup.com/by/SketchUp>

Para que um modelo 3D feito no Sketchup possa ser utilizado no Unity é necessário exportá-lo para um formato compatível. E para fazer isso basta seguir os seguintes passos:

Selecione a opção da aba:  
“File → New Project”



Selecione um formato compatível  
(neste caso: “.dae”)

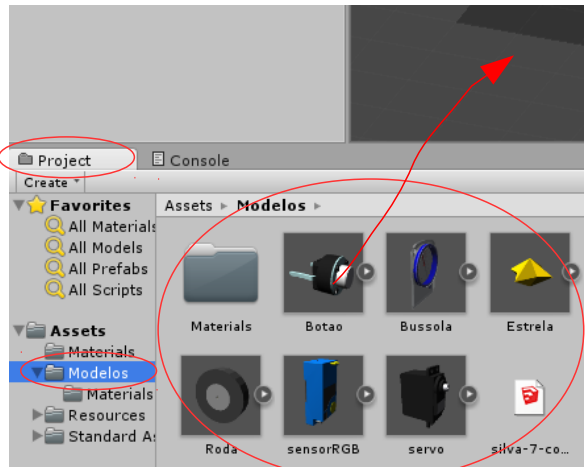


Exporta para uma das pastas  
do projeto do Unity

Feito isto, abra o  
projeto no Unity3D

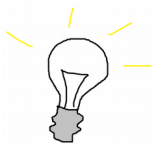
Na aba "Project",  
abra a pasta a qual  
salvou o modelo

Note que o seu  
modelo já está pronto  
para uso. Basta arrastá-lo  
para o cenário



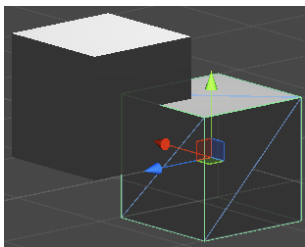
## Dicas Importantes

→ Lembre-se sempre do Rigidbody!



Todo objeto que se comporta segundo as leis da Física deve possuir o componente Rigidbody, caso não possua, o framework não funcionará corretamente.

→ Cuidado ao posicionar objetos muito próximos!



É importante sempre deixar uma pequena folga entre as partes de seu robô, pois caso algumas destas colidam entre si, seu robô poderá não funcionar como esperado.

→ Sistema de Unidades

As unidades básicas consideradas pelo Framework são as seguintes:

Comprimento → Metro

Tempo → Segundo

Massa → Quilogramas



→ Uma imagem vale mais que mil palavras!

Acesse nosso canal no youtube:

<https://www.youtube.com/channel/UCluacPjBusXUrIF58c8n8-g>

Lá você encontrará dicas e tutoriais de como utilizar o framework

# 3º Capítulo: Simulações Simples

Neste capítulo será abordado o desenvolvimento de projetos simples, que façam uso dos componentes presentes no framework proposto pelo livro.

Para prosseguir na leitura deste capítulo, é recomendado domínio dos assuntos dos capítulos anteriores.

## 3.1 – Construindo um ventilador

Neste projeto construiremos uma espécie de ventilador. Este deverá ter uma hélice que girará continuamente, e seu corpo deverá alternar sua direção entre 0° e 180°.

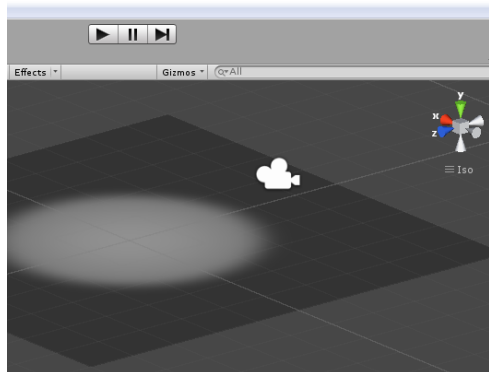
Para a execução deste projeto siga a seguinte sequência de passos:

Crie ou abra um projeto  
Configurado de acordo  
com o framework

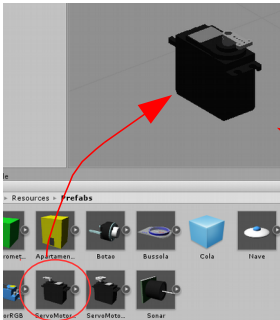
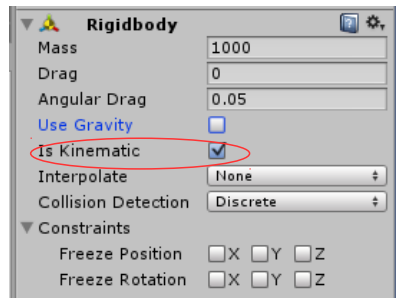
Se necessário,  
importe os prefabs:  
Cola,  
Servo motor limitado, e  
Servo motor de rotação

Crie um plano para servir  
De solo para o ventilador

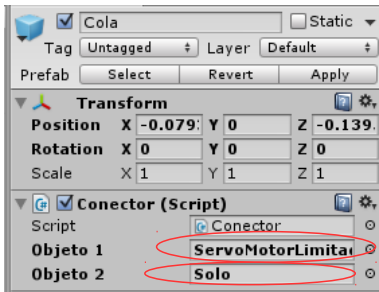
Posicione a câmera e  
Iluminação de forma  
A facilitar a visualização  
da cena



Adicione o componente  
Rigidbody ao plano,  
marcando a opção “is Kinematic”  
e definindo a massa igual a 1000kg

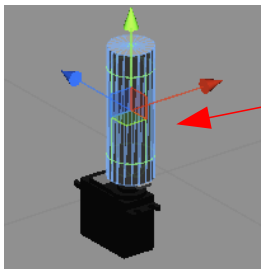


Crie uma instância do  
prefab Servo motor  
limitado



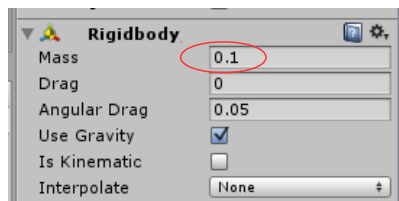
Crie uma instância do  
prefab Cola

Use o prefab cola para  
Fixar o servo motor ao  
Solo como segue:

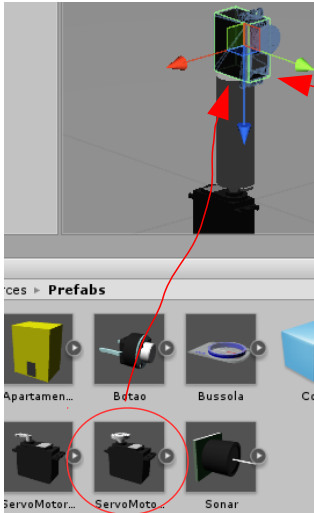
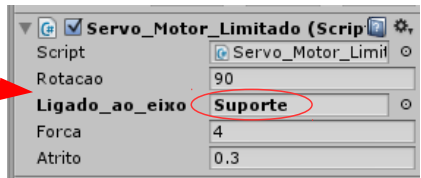


Crie um novo objeto  
que servirá como suporte  
do rotor e da hélice

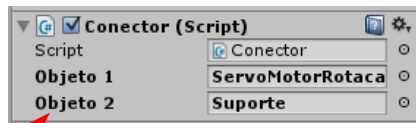
Adicione o componente  
Rigidbody ao suporte do rotor  
Definindo a massa igual a 0.1kg



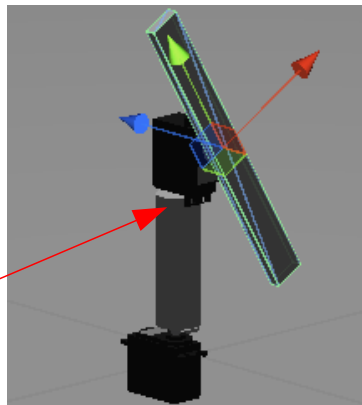
Conecte este suporte ao eixo do servo motor Limitado como segue:



Crie uma instância do Servo motor de rotação

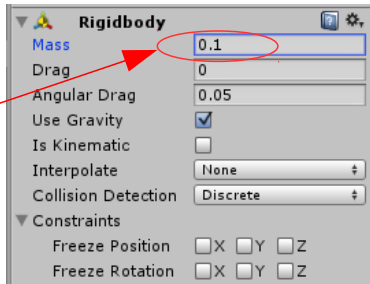


Utilizando uma nova Instancia do prefab Cola, fixe o servo motor de rotação ao suporte do rotor, como segue:



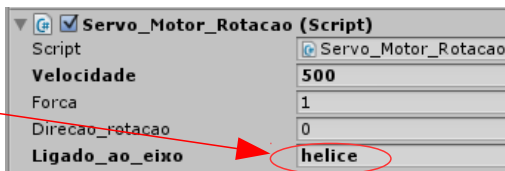
Crie um objeto que servirá como hélice do ventilador

Crie um componente Rigidbody para a hélice E defina a massa igual a 0.1kg





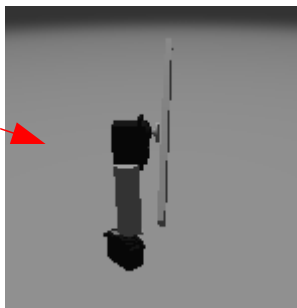
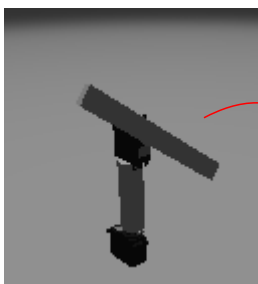
Conecte a hélice do ventilador  
Ao eixo do servo motor de  
rotação, como segue:



Agora basta executar para ver  
os resultados



Selecionando o servo motor limitado, note que ao  
modificar o valor do atributo rotação o ventilador  
terá sua direção alterada.



**Desafio:** Pense e implemente uma forma do ventilador ficar  
alternando sua direção automaticamente!

### 3.2 – Construindo um carro de controle remoto

Este projeto consiste no desenvolvimento de um carro controlado por um aplicativo externo ao Unity. Este carro deve possuir um eixo direcionador que define sua direção de movimento, e deverá ter propulsão nas 4 rodas.

Como o carro deverá ser controlado remotamente, os comandos que ele poderá receber são:

“acelerar”

“frear”

“direcionar”(ângulo)

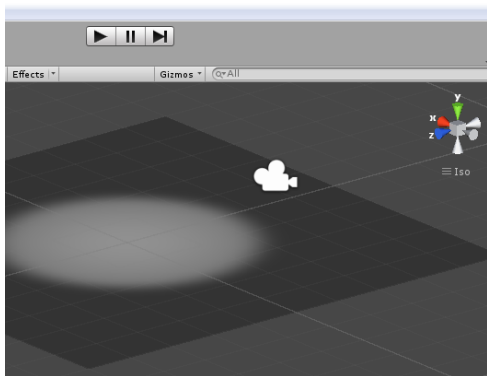
Para a execução deste projeto basta seguir os seguintes passos:

Crie ou abra um projeto configurado de acordo com o framework

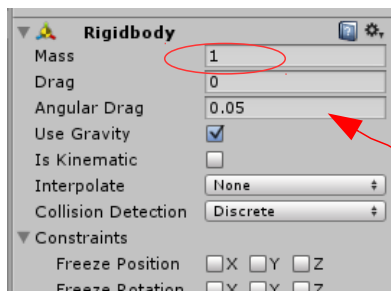
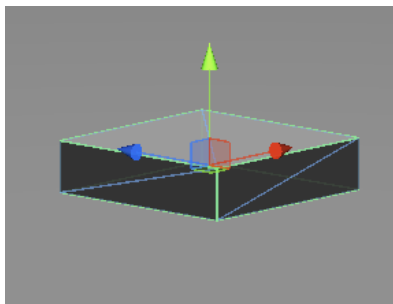
Se necessário,  
importe os prefabs:  
Cola,  
Servo motor limitado,  
Servo motor de rotação,  
e Roda

Crie um plano para servir  
de superfície para o carro  
se deslocar

Posicione a câmera e  
iluminação de forma  
a facilitar a visualização  
da cena

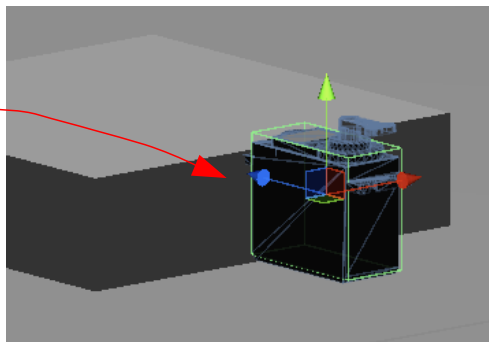


Crie um objeto para servir de chassi para o carro

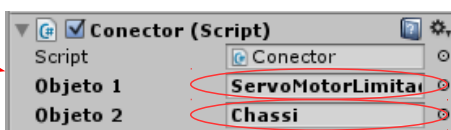


Crie um componente Rigidbody para o chassi do carro, e defina sua massa igual a 1kg

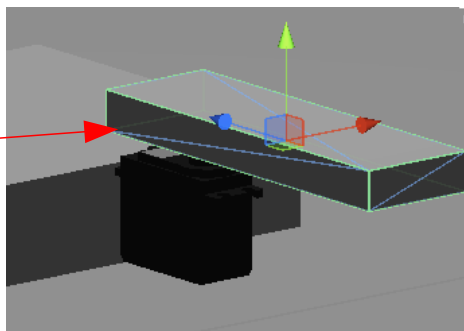
Crie uma instância do prefab Servo motor limitado,



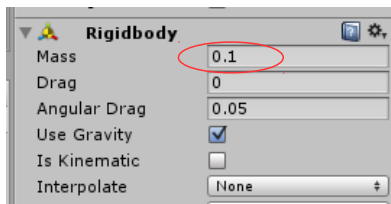
Utilizando uma nova instância do prefab Cola, fixe o servo motor ao chassi do carro, Como segue:



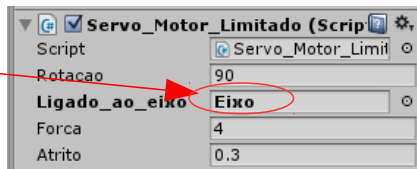
Crie um objeto que servirá de eixo para direcionamento do carro



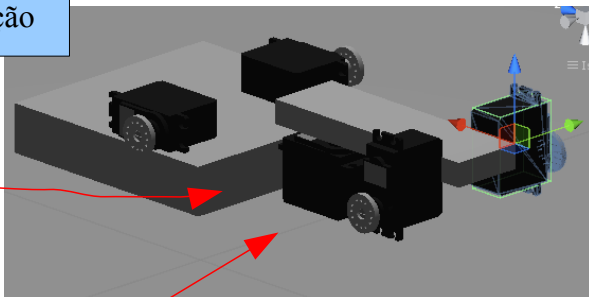
Crie um componente Rigidbody para o eixo do carro e defina a massa igual a 0.1kg



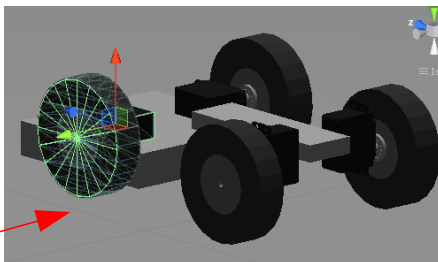
Conecte o eixo de direcionamento do carro ao eixo do servo motor, como segue:



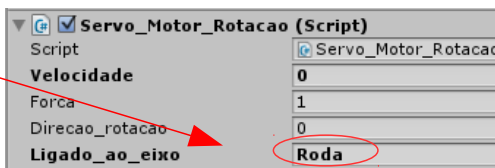
Crie quatro instâncias de Servo motores de rotação



Utilizando uma instancia do prefab Cola para cada servo motor de rotação, fixe cada servo motor de rotação ao chassi ou ao eixo do carro

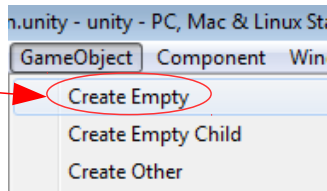


Para cada servo motor de rotação, crie uma instância do prefab Roda e conecte, cada roda a seu respectivo Servo motor de rotação, como segue:

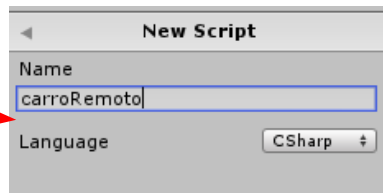


Uma vez com a estrutura montada temos que criar um servidor para que o carro possa ser controlado por um software externo ao Unity. Para isso basta seguir os seguintes passos:

Crie um novo objeto que servirá como servidor



Para este objeto, crie um novo componente do tipo "New Script" com o nome "carroRemoto.cs"



Abra o script criado e insira o seguinte código:

```
using UnityEngine;
using System.Collections;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Globalization;

public class carroRemoto : MonoBehaviour {

    private Socket cliente;
    private TcpListener server = new TcpListener(IPAddress.Any, 12598);

    // Use this for initialization
    void Start () {
        server.Start(); // inicia o servidor
    }
}
```

```

void Update () {
    int tamanho = 0; //tamanho da mensagem recebida
    if (server.Pending()){
        cliente = server.AcceptSocket(); //aceita conexao
        cliente.Blocking = false;

        byte[] mensagem = new byte[1024];
        string strMessage = "";
        while(!strMessage.Contains(";")){
            try{tamanho = cliente.Receive (mensagem);
                strMessage = strMessage +
                    System.Text.Encoding.UTF8.GetString(mensagem);
            }catch(System.Exception e){}
        }

        string comando = strMessage.Split(';')[0];

        byte[] envioBuffer = new byte[4];
        envioBuffer[0] = (byte)'a';
        envioBuffer[1] = (byte)'c'; // mensagem a ser enviada ao cliente
        envioBuffer[2] = (byte)'k';
        envioBuffer[3] = 10;
        cliente.Send(envioBuffer);

        Servo_Motor_Rotacao [] servos =
            GameObject.FindObjectsOfType<Servo_Motor_Rotacao>();

        if(strMessage.Contains("direcao")){
            int direcao = int.Parse(strMessage.Split(';')[1]);
            Servo_Motor_Limitado servo =
                GameObject.FindObjectOfType<Servo_Motor_Limitado>();
            servo.rotacao = direcao;
        }
        else for(int i = 0; i < servos.Length; i++)
        {
            if(strMessage.Contains("acelerar")){
                servos[i].velocidade = 100;
            }else if(strMessage.Contains("frear")){
                servos[i].velocidade = 0;
            }
        }
    }
}
}

```

Este script inicia um servidor local, que aguarda conexões na porta 12598. Cada conexão deverá enviar apenas uma mensagem terminada com o caractere ';'. Cada mensagem será traduzida a um comando para o carro no cenário. Os possíveis comandos são:

“acelerar”  
 “frear”  
 “direcionar”(ângulo)

Uma vez com o lado servidor pronto, use o seguinte código em Python para o programa cliente:

```
import socket
import time
import sys

def enviar_para_unity (msg):
    TCP_IP = '127.0.0.1'
    TCP_PORT = 12598 #porta do servidor Unity
    BUFFER_SIZE = 1024
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((TCP_IP, TCP_PORT))
    s.send(msg)
    data = s.recv(BUFFER_SIZE)
    s.close()
    print "mensagem:", data

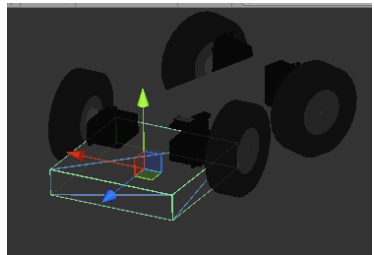
while(1):
    enviar_para_unity("acelerar;\n")
    time.sleep(2)
    enviar_para_unity("direcionar(70);\n")
    time.sleep(2)
    enviar_para_unity("direcionar(90);\n")
    time.sleep(2)
    enviar_para_unity("direcionar(110);\n")
    time.sleep(10)
    enviar_para_unity("direcionar(90);\n")
    time.sleep(2)
    enviar_para_unity("frear;\n")
    time.sleep(5)
```

Este código simples em Python, apenas controla o veículo para ficar andando em círculos.

Feito isso, execute o cenário



E então execute o cliente



O carro simulado no Unity está sendo controlado pelo software escrito em Python. Estude o código para poder entender a fundo como o sistema funciona... É simples!!

### 3.3 – Construindo robô programável remotamente

O conceito do robô programável remotamente é bastante parecido com o conceito do carro abordado na seção 3.2. A principal ideia relacionada a este projeto, é que o robô possuirá um conjunto de comandos básicos, e estes comandos serão recebidos por meio de mensagens mandadas pelo cliente (controlador).

Dessa forma, o comportamento do robô será relacionado ao que for programado pelo cliente, ou seja, programação remota.

Os comandos básicos que o robô possuirá serão:

“frente”  
“tras”  
“direita”  
“esquerda”  
“distancia”

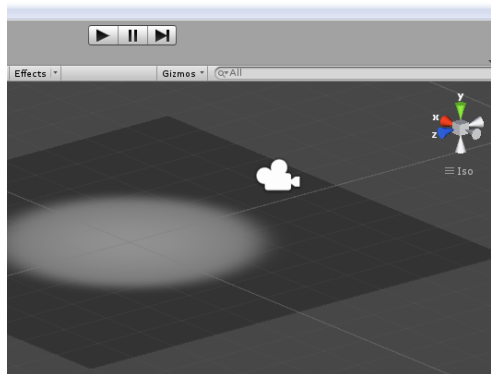
Onde, o comando “distancia” deve retornar a distância do objeto a frente do robô.

Para a execução deste projeto faça os passos que seguem:

Crie ou abra um projeto configurado de acordo com o framework

Se necessário,  
importe os prefabs:  
Cola,  
Servo motor de rotação,  
Sensor IV,  
e Roda

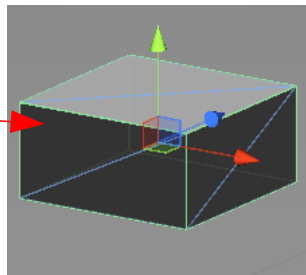
Crie um plano para servir  
de superfície para o robô  
se deslocar



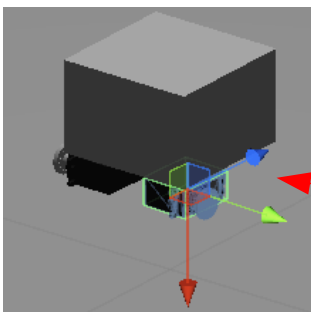
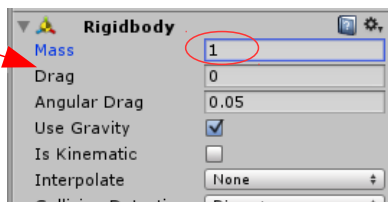
Posicione a câmera e  
Iluminação de forma  
A facilitar a visualização  
da cena



Crie um objeto para servir de chassi para o robô

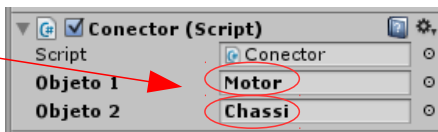


Crie um componente Rigidbody para o chassi do robô, e defina sua massa igual a 1kg

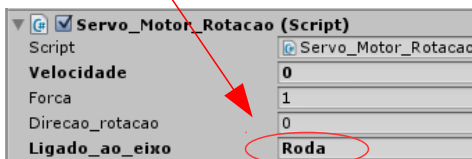
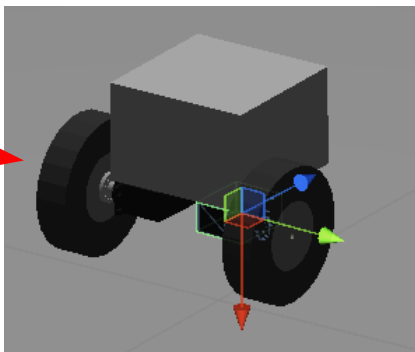


Crie duas instâncias do prefab Servo motor de rotação

Utilizando uma instancia do prefab Cola para cada servo motor de rotação, fixe cada servo motor de rotação ao chassi do robô, como segue:



Para cada servo motor de rotação, crie uma instância do prefab Roda, e acople cada roda ao seu respectivo motor, como segue:

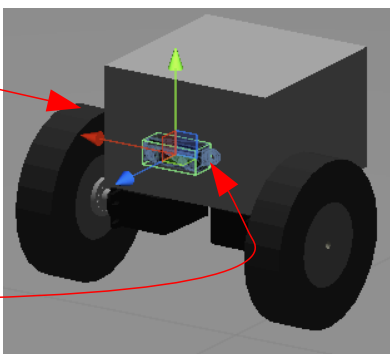
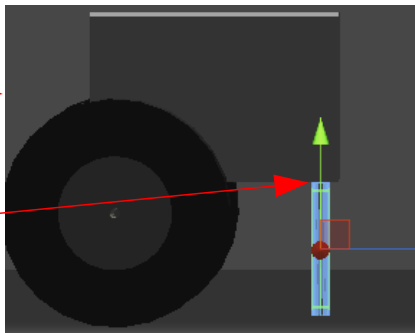


Crie um objeto para servir como ponto de apoio para o chassi do robô, com massa de 0,1kg

Crie uma nova instância do prefab Cola, para fixar o ponto de apoio ao chassi

Crie uma nova instância do prefab Sensor IV Posicionando-o a frente do robô

Utilizando uma nova instância do prefab Cola fixe o Sensor IV ao chassi do robô

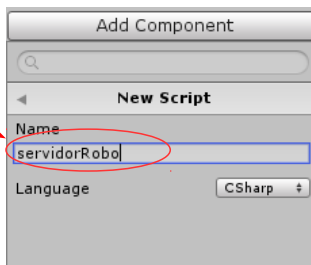
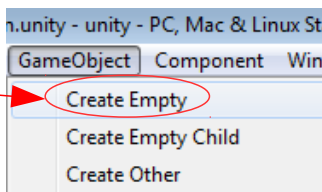


Com a estrutura terminada, temos que preparar o nosso robô para receber as instruções do programa que será executado num software remoto. Para isso criaremos um servidor que interpretará as mensagens recebidas. Sendo assim, siga os seguintes passos:

Crie um novo objeto que será o servidor

Crie um novo componente do tipo “New Script”, com o nome de sevidorRobo.cs

Abra o Script e insira o código que segue:



```

using UnityEngine;
using System.Collections;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Globalization;

public class servidorRobo : MonoBehaviour {

    private Socket cliente;
    private TcpListener server = new TcpListener(IPAddress.Any, 12598);

    // Use this for initialization
    void Start () {
        server.Start(); // inicia o servidor
    }

    void Update () {
        int tamanho = 0; //tamanho da mensagem recebida
        if (server.Pending()){
            cliente = server.AcceptSocket(); //aceita conexao
            cliente.Blocking = false;

            byte[] mensagem = new byte[1024];
            string strMessage = "";
            while(!strMessage.Contains(";")){
                try{tamanho = cliente.Receive (mensagem);
                    strMessage = strMessage +
                        System.Text.Encoding.UTF8.GetString(mensagem);
                }catch(System.Exception e){}
            }

            string comando = strMessage.Split(';')[0];
            byte[] envioBuffer = new byte[4];
            envioBuffer[0] = (byte)'a';
            envioBuffer[1] = (byte)'c'; // mensagem a ser enviada ao cliente
            envioBuffer[2] = (byte)'k';
            envioBuffer[3] = 10;

            Servo_Motor_Rotacao [] servos =
            GameObject.FindObjectsOfType<Servo_Motor_Rotacao>();
            if(strMessage.Contains("distancia")){
                Sensor_Infra_Vermelho sensorIV =
                GameObject.FindObjectOfType<Sensor_Infra_Vermelho>();
                float distancia = sensorIV.GetDistancia();
                envioBuffer[0] = (byte)distancia;
                envioBuffer[1] = 10;
            }
        }
    }
}

```

```

else for(int i = 0; i < servos.Length; i++)
{
    if(strMessage.Contains("frente")){
        servos[i].velocidade = 150;
        servos[i].direcao_rotacao = 1-i;
    }else if(strMessage.Contains("tras")){
        servos[i].velocidade = 150;
        servos[i].direcao_rotacao = i;
    }else if(strMessage.Contains("direita")){
        servos[i].velocidade = 150;
        servos[i].direcao_rotacao = 0;
    }else if(strMessage.Contains("esquerda")){
        servos[i].velocidade = 150;
        servos[i].direcao_rotacao = 1;
    }else if(strMessage.Contains("parar")){
        servos[i].velocidade = 0;
    }
}
cliente.Send(envioBuffer);
}
}
}

```

Este script inicia um servidor local, que aguarda conexões na porta 12598. Cada conexão deverá enviar apenas uma mensagem terminada com o caractere ';'. Cada mensagem será traduzida a um comando atômico para o robô no cenário. Os possíveis comandos são:

“frente”  
 “tras”  
 “direita”  
 “esquerda”  
 “distancia” → inteiro

Com o lado servidor pronto, basta implementar um cliente que envie e receba as mensagens relativas aos comandos do robô. Como o seguinte exemplo:

```

import socket
import time
import sys

def enviar_para_unity (msg):
    TCP_IP = '127.0.0.1'
    TCP_PORT = 12598 #porta do servidor Unity
    BUFFER_SIZE = 1024
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((TCP_IP, TCP_PORT))
    s.send(msg)
    data = s.recv(BUFFER_SIZE)
    s.close()
    return data

def frente():
    enviar_para_unity("frente;\n")

def direita():
    enviar_para_unity("direita;\n")

def esquerda():
    enviar_para_unity("esquerda;\n")

def tras():
    enviar_para_unity("tras;\n")

def parar():
    enviar_para_unity("parar;\n")

def distancia():
    return ord(enviar_para_unity("distancia;\n")[0])

while(1):
    if(distancia() > 30):
        frente()
    else:
        parar()

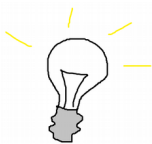
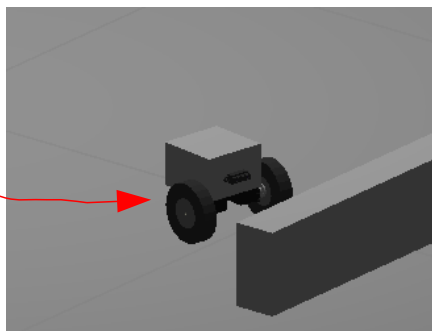
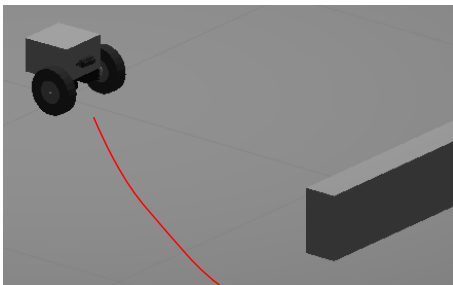
```

Este código em Python implementa uma espécie de API(Application User Interface), que define funções que encapsulam, o envio de mensagens pela rede. O programa principal nele presente, é a programação para o robô seguir em frente até encontrar um obstáculo, e então parar.

Com o cliente e o servidor prontos, execute o cenário no Unity



E então execute o cliente.  
para verificar o  
funcionamento do robô



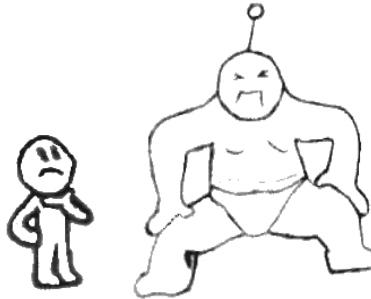
Neste exemplo, você pode programar o robô em python e verificar seu comportamento nos mais variados ambientes. Que tal fazer experimentos?

Você também pode programar o robô, em outra linguagem, com outro cenário, e outro código!!  
Sinta-se livre!!



## 4º Capítulo: Simulação de Robô Sumô

Neste capítulo capítulo desenvolveremos um simulador de robô sumô completo. O robô sumô é uma competição bastante conhecida na área da robótica móvel, que consiste em uma arena onde dois robôs disputam um espaço da mesma. O robô vencedor é aquele que permanecer mais tempo ativo dentro deste espaço.



O projeto será dividido em cinco etapas, descritas a seguir:

**Preparação do cenário:** Nesta etapa criaremos um cenário que simulará uma competição de robô sumô. Nela serão feitas a arena e as regras relativas à execução da competição.

**Construção dos sumôs:** Nesta etapa prepararemos dois robôs, que serão adversários na competição.

**Estabelecimento da comunicação em rede:** Nesta etapa construiremos a infraestrutura necessária para a programação remota dos robôs.

**Programação dos robôs:** Nesta etapa elaboraremos comportamentos para os robôs utilizando a infraestrutura de rede da etapa anterior.

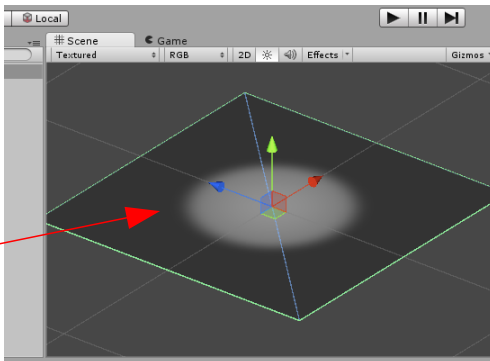
**Execução da competição:** Nesta etapa finalmente executaremos o projeto e verificaremos o vencedor!

## 4.1 – Preparando o cenário e as regras

Vamos agora construir nossa arena! Para isso precisamos seguir os seguintes passos:

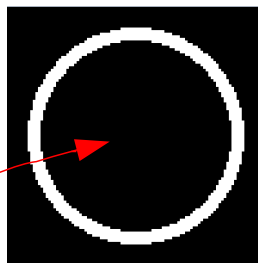
Num projeto configurado de acordo com o framework, crie um novo cenário

Neste novo cenário crie uma iluminação e um plano



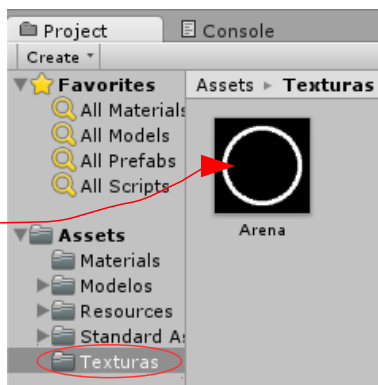
O nosso cenário precisa de uma arena que limite o espaço que os robôs poderão ficar. Para isso criaremos uma textura para o plano.

Sendo assim, usando um editor de desenhos, crie uma imagem preta com dimensão de 100x100 pixels

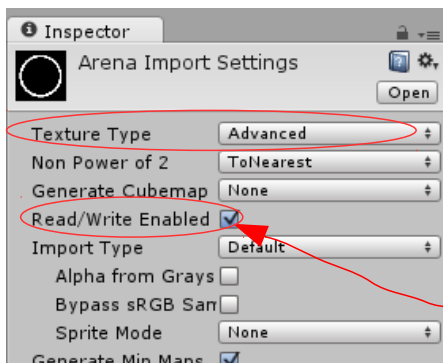


No centro desta imagem insira um círculo branco com diâmetro de 80 pixels e espessura razoável

Salve a imagem numa pasta do projeto seu projeto





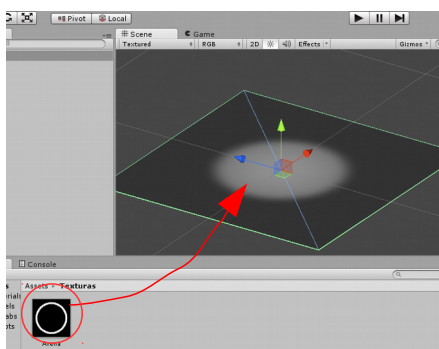


Ao selecionar a textura, note que existe uma lista de configurações na aba “Inspector”

Modifique o atributo “Texture type” para “Advanced”, e marque a caixa Read/Write Enabled. Isso é necessário para que o sensor RGB possa identificar as cores da textura

Uma vez com a textura salva e configurada selecione o plano. E arraste a textura sobre este

Posicione o plano na origem do sistema de coordenadas, e defina sua dimensão para 2x2



Uma vez com a arena pronta vamos definir as regras da competição. São elas:

Se algum competidor ficar mais de 10 segundos na mesma posição, este será desclassificado por inatividade.

Se algum competidor estiver a uma distância maior que 0.81 unidades da origem em relação ao plano XOZ, este será desclassificado.

Para atender as especificações criaremos um script chamado “Competidor”. Este script interpretará as regras da competição, e todo robô sumô deverá possuí-lo como componente.

Sendo assim... Crie um script C# em uma pasta do projeto com nome: “Competidor.cs”, e insira o seguinte código:

```
using UnityEngine;
using System.Collections;

public class Competidor : MonoBehaviour {
    static int numeroCompetidores = 0;
    int identificador;
    Vector3 posicaoAnterior;
    float tempoParado;
    bool desclassificado = false;

    // Use this for initialization
    void Start () {
        identificador = numeroCompetidores;
        numeroCompetidores++;
    }

    void desclassificar(){
        if(numeroCompetidores > 1){
            desclassificado = true;
            numeroCompetidores--;
            print("Desclassificado!!");
        }
    }

    // Update is called once per frame
    void Update () {
        if (posicaoAnterior == transform.position) {
            tempoParado += Time.deltaTime;
        }else{
            tempoParado = 0.0f;
        }
        if (tempoParado >= 10) {
            desclassificar();
        }
        posicaoAnterior = transform.position;
        Vector2 posicaoPlanar = new Vector2 (posicaoAnterior.x,
posicaoAnterior.z);
        if (posicaoPlanar.magnitude > 0.81) {
            desclassificar ();
        }
    }
}
```

Este código basicamente, implementa as regras citadas anteriormente em relação a desclassificação dos competidores.

Como próximo passo teremos que construir os robôs que serão os competidores.

## 4.2 – Construindo os robôs competidores

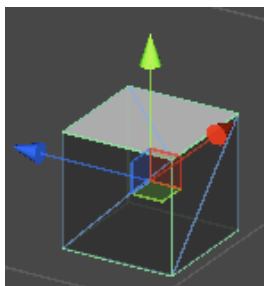
Nesta seção iremos construir os robôs sumô utilizando os componentes propostos pelo framework. Estes serão dois robôs que possuirão formas e estratégias diferentes.

O primeiro robô possuirá a estratégia de predador, procurando e empurrando seu oponente para fora da arena.

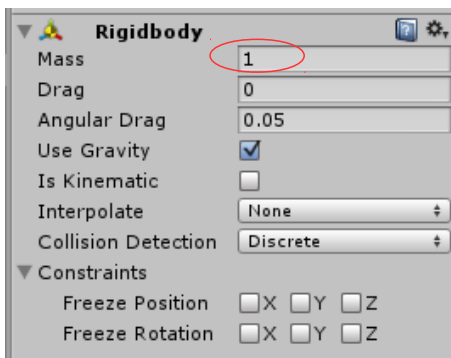
O segundo robô possuirá a estratégia da presa, buscando vencer seu oponente baseado na chance deste se jogar para fora da arena.

Para construir o robô predador siga os seguinte passos:

Crie um cubo que servirá como chassi do robô



Adicione o componente Rigidbody ao chassi, atribuindo massa de 1kg



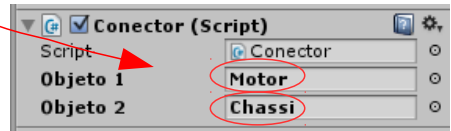
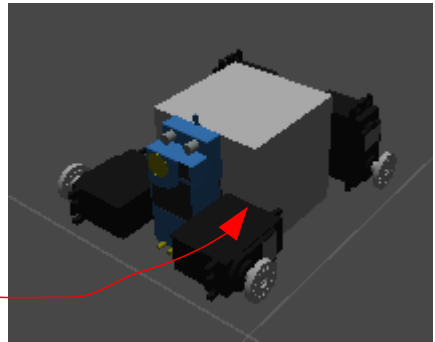
Adicione agora o script Competidor ao chassi do robô

Crie uma instância do prefab Sensor RGB e posicione-o à frente do chassi direcionado para baixo

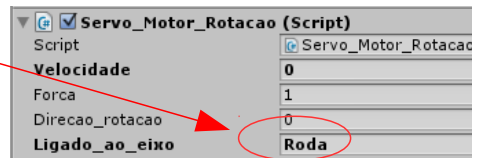


Crie e posicione quatro Servo motores de rotação

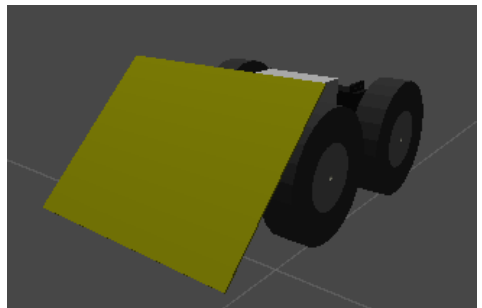
Utilizando instâncias do prefab Cola, fixe os servos motores e o Sensor RGB ao chassi do robô



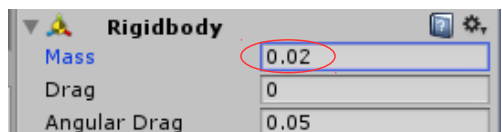
Crie quatro instâncias do prefab Roda, conecte cada uma ao eixo de um motor



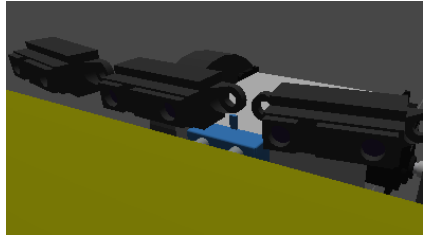
Para aumentar a eficiência do nosso robô vamos adicionar uma rampa a sua frente. Para isso adicione um cubo e faça uma das dimensões ser relativamente baixa.



Adicione também um componente Rigidbody à rampa com massa 0.02

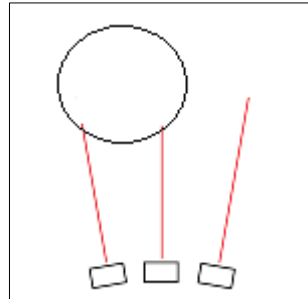


Agora nosso predador precisa localizar sua presa e para isso vamos utilizar sensores de distância infravermelho



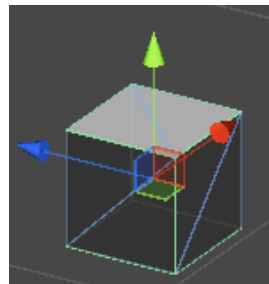
Utilizando instâncias do prefab Cola fixe três instâncias do prefab Sensor IV no chassi do robô

É importante que haja um ângulo entre os sensores para que possamos identificar a posição de objetos pela diferença das distâncias de cada sensor.

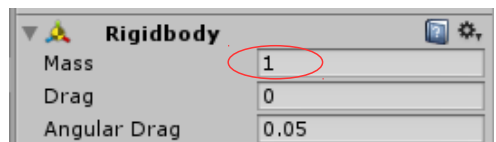


Agora já construímos a estrutura do nosso primeiro sumô, vamos ao nosso segundo competidor, o robô presa. Para isso siga os seguintes passos:

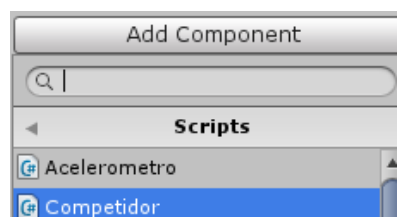
Crie um cubo que servirá como chassi do robô



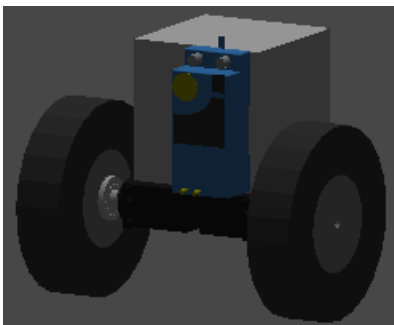
Adicione o componente Rigidbody ao chassi, atribuindo massa de 1kg



Adicione agora o script Competidor ao chassi do robô

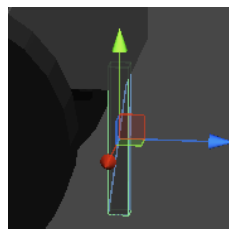


Acople a frente do chassi, na ponta a esquerda uma instância do prefab Sensor RGB

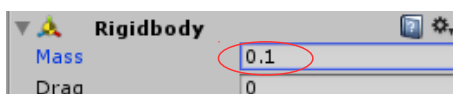


Fixe dois Servo motores de Rotação ao chassi, utilizando o prefab Cola. E acople uma Roda em cada motor.

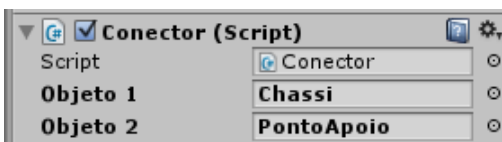
Crie um objeto que sirva como ponto de apoio para o chassi



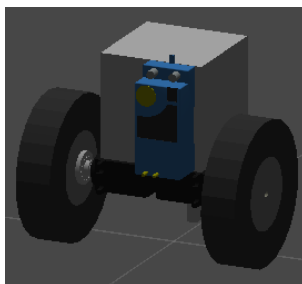
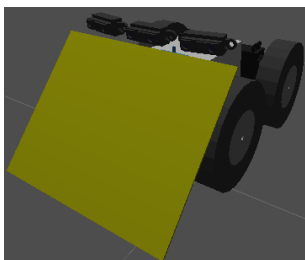
Adicione ao ponto de apoio o componente Rigidbody com massa igual a 0.1



Fixe o ponto de apoio ao chassi utilizando o prefab Cola



### 4.3 – Conectando os sumôs a rede



Agora que terminamos de construir nossos competidores, temos que prepará-los para receber os comandos vindos da rede. Para isso iremos desenvolver dois servidores que operarão em cada um dos robôs no simulador.

A função dos servidores é disponibilizar os serviços, que cada robô pode executar, para que estes possam ser utilizados por mensagens de outros processos. Sendo assim os serviços que os servidores dos robôs deverão oferecer são:

#### Predador

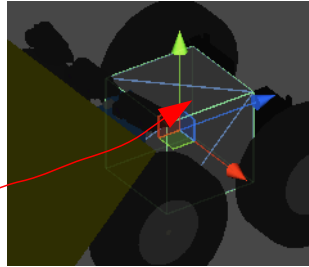
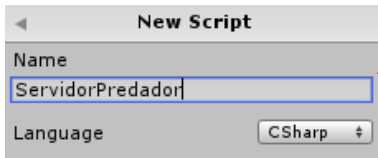
Ir para frente  
Parar  
Ir para traz  
Direita  
Esquerda  
Ler cor  
Ler distancia 1  
Ler distancia 2  
Ler distancia 3

#### Presas

Ir para frente  
Parar  
Ir para traz  
Direita  
Esquerda  
Ler cor

Primeiramente iremos desenvolver o servidor do predador para isso siga os seguintes passos:

Adicione um script C# ao chassi do Robô predador com nome: “ServidorPredador.cs”



Insira o seguinte código no novo script:

```
using UnityEngine;
using System.Collections;
using System.IO;
using System.Net;
using System.Net.Sockets;

public class ServidorPredador : MonoBehaviour {

    public Servo_Motor_Rotacao motorEsquerdoFrontal;
    public Servo_Motor_Rotacao motorDireitoFrontal;
    public Servo_Motor_Rotacao motorEsquerdoTraseiro; //partes do robo
    public Servo_Motor_Rotacao motorDireitoTraseiro; //devem ser definidas no Unity
    public SensorRGB sensorRGB;
    public Sensor_Infra_Vermelho sensorIVEsquerdo;
    public Sensor_Infra_Vermelho sensorIVCentral;
    public Sensor_Infra_Vermelho sensorIVDireito;

    private Socket cliente;
    private TcpListener server = new TcpListener(IPAddress.Any, 12598);

    // Use this for initialization
    void Start () {
        server.Start(); // inicia o servidor
    }

    //Comandos de movimento-----
    void frente(){
        motorEsquerdoFrontal.direcao_rotacao = 0;
        motorDireitoFrontal.direcao_rotacao = 1;
        motorEsquerdoTraseiro.direcao_rotacao = 0;
        motorDireitoTraseiro.direcao_rotacao = 1;
        andar();
    }
}
```



```

void andar(){
    motorEsquerdoFrontal.velocidade = 300;
    motorDireitoFrontal.velocidade = 300;
    motorEsquerdoTraseiro.velocidade = 300;
    motorDireitoTraseiro.velocidade = 300;
}

void parar(){
    motorEsquerdoFrontal.velocidade = 0;
    motorDireitoFrontal.velocidade = 0;
    motorEsquerdoTraseiro.velocidade = 0;
    motorDireitoTraseiro.velocidade = 0;
}
//-----

void Update () {
    int tamanho = 0; //tamanho da mensagem recebida
    if (server.Pending()){
        cliente = server.AcceptSocket(); //aceita conexao
        cliente.Blocking = false;

        byte[] mensagem = new byte[1024];
        string strMessage = "";
        while(!strMessage.Contains(";")){ //recebe a mensagem
            try{tamanho = cliente.Receive (mensagem);
                strMessage = strMessage +
                    System.Text.Encoding.UTF8.GetString(mensagem);
            }catch(System.Exception e){}
        }

        string comando = strMessage.Split(';')[0];
        byte[] envioBuffer = new byte[4];
        envioBuffer[0] = (byte)'a';
        envioBuffer[1] = (byte)'c'; // mensagem a ser enviada ao cliente
        envioBuffer[2] = (byte)'k';
        envioBuffer[3] = 10;

        //decodifica a mensagem
        if(strMessage.Contains("cor")){
            int cor = sensorRGB.getCor();
            envioBuffer[0] = (byte)((cor >> 16) & 255); // R
            envioBuffer[1] = (byte)((cor >> 8) & 255); // G
            envioBuffer[2] = (byte)(cor & 255); // B
        }else if(strMessage.Contains("distanciaE")){
            float distancia = sensorLVEsquerdo.GetDistancia();
            envioBuffer[0] = (byte)distancia;
            envioBuffer[1] = 10;
        }else if(strMessage.Contains("distanciaC")){
            float distancia = sensorVCentral.GetDistancia();
            envioBuffer[0] = (byte)distancia;
            envioBuffer[1] = 10;
        }else if(strMessage.Contains("distanciaD")){
            float distancia = sensorVDireito.GetDistancia();
            envioBuffer[0] = (byte)distancia;
            envioBuffer[1] = 10;
        }else if(strMessage.Contains("frente")){
            frente();
        }else
    }
}

```

```

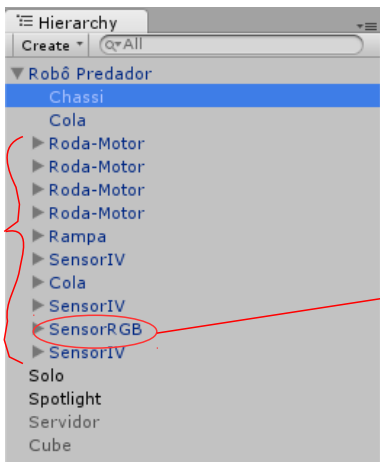
        if(strMessage.Contains("tras")){
            tras();
        }else if(strMessage.Contains("direita")){
            direita();
        }else if(strMessage.Contains("esquerda")){
            esquerda();
        }else if(strMessage.Contains("parar")){
            parar();
        }
        cliente.Send(envioBuffer); //responde ao cliente
    }
}

```

Este script inicia um servidor local, que aguarda conexões na porta 12598. Cada conexão deverá enviar apenas uma mensagem terminada com o caractere ';'. Cada mensagem será traduzida a um comando para nosso robô sumô predador. Os possíveis comandos que poderão ser recebidos do cliente são:

- “cor” → retorna a cor lida pelo sensor RGB;
- “distanciaE” → retorna a distância lida pelo sensor infravermelho esquerdo;
- “distanciaC” → retorna a distância lida pelo sensor IV central;
- “distanciaD” → retorna a distância lida pelo sensor IV direito;
- “frente” → mover para frente;
- “tras” → mover para trás;
- “direita” → direciona para a direita;
- “esquerda” → direciona para a esquerda;
- “parar” → para os motores;

Após inserir o código, defina os atributos públicos do script. Para defini-los basta arrastar cada parte do robô para o seu respectivo atributo.

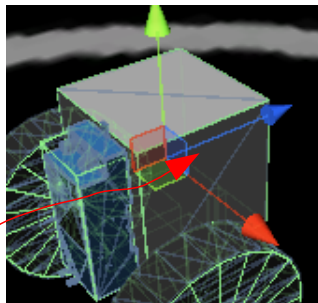
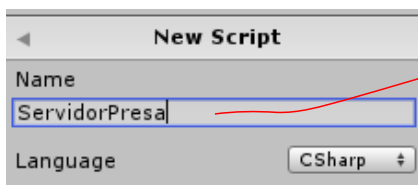


Agora nosso predador está a uma “conexão” de distância para a batalha!!!



Agora que nosso predador está pronto para conexão, vamos preparar nossa presa, para que possa tentar se defender. Para isso vamos fazer basicamente a mesma coisa que foi feita para o predador, as únicas mudanças serão os serviços que o servidor da presa disponibilizará, uma vez que ela possui menos sensores que o predador. Sendo assim siga os seguintes passos:

Adicione um script C# ao chassi do Robô presa com nome: “ServidorPresa.cs”



E então insira o seguinte código no novo script:

```
using UnityEngine;
using System.Collections;
using System.IO;
using System.Net;
using System.Net.Sockets;

public class ServidorPresa : MonoBehaviour {

    public Servo_Motor_Rotacao motorEsquerdo;
    public Servo_Motor_Rotacao motorDireito;
    public SensorRGB sensorRGB;

    private Socket cliente;
    private TcpListener server = new TcpListener(IPAddress.Any, 12599);

    // Use this for initialization
    void Start () {
        server.Start(); // inicia o servidor
    }

    //Comandos de movimento-----
    void frente(){
        motorEsquerdo.direcao_rotacao = 0;
        motorDireito.direcao_rotacao = 1;
        andar();
    }

    void tras(){
        motorEsquerdo.direcao_rotacao = 1;
        motorDireito.direcao_rotacao = 0;
        andar();
    }

    void esquerda(){
        motorEsquerdo.direcao_rotacao = 1;
        motorDireito.direcao_rotacao = 1;
        andar();
    }

    void direita(){
        motorEsquerdo.direcao_rotacao = 0;
        motorDireito.direcao_rotacao = 0;
        andar();
    }

    void andar(){
        motorEsquerdo.velocidade = 200;
        motorDireito.velocidade = 200;
    }

    void parar(){
        motorEsquerdo.velocidade = 0;
        motorDireito.velocidade = 0;
    }
}
```

```

void Update () {
    int tamanho = 0; //tamanho da mensagem recebida
    if (server.Pending()){
        cliente = server.AcceptSocket(); //aceita conexao
        cliente.Blocking = false;

        byte[] mensagem = new byte[1024];
        string strMessage = "";
        while(!strMessage.Contains(";")){ //recebe a mensagem
            try{tamanho = cliente.Receive (mensagem);
                strMessage = strMessage +
                    System.Text.Encoding.UTF8.GetString(mensagem);
            }catch(System.Exception e){}
        }

        string comando = strMessage.Split(';')[0];
        byte[] envioBuffer = new byte[4];
        envioBuffer[0] = (byte)'a';
        envioBuffer[1] = (byte)'c'; // mensagem a ser enviada ao cliente
        envioBuffer[2] = (byte)'k';
        envioBuffer[3] = 10;

        //decodifica a mensagem
        if(strMessage.Contains("cor")){
            int cor = sensorRGB.getCor();
            envioBuffer[0] = (byte)((cor >> 16) & 255); // R
            envioBuffer[1] = (byte)((cor >> 8) & 255); // G
            envioBuffer[2] = (byte)(cor & 255); // B
        }else if(strMessage.Contains("frente")){
            frente();
        }else if(strMessage.Contains("tras")){
            tras();
        }else if(strMessage.Contains("direita")){
            direita();
        }else if(strMessage.Contains("esquerda")){
            esquerda();
        }else if(strMessage.Contains("parar")){
            parar();
        }
        cliente.Send(envioBuffer); //responde ao cliente
    }
}
}

```

Este script inicia um servidor local, que aguarda conexões na porta 12599. Cada conexão deverá enviar apenas uma mensagem terminada com o caractere ';'. Cada mensagem será traduzida a um comando para nosso robô sumô presa. Os possíveis comandos que poderão ser recebidos do cliente são:

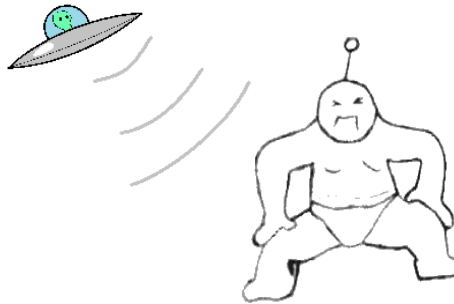
“cor” → retorna a cor lida pelo sensor RGB;  
 “frente” → mover para frente;  
 “tras” → mover para trás;

“direita” → direciona para a direita;  
“esquerda” → direciona para a esquerda;  
“parar” → para os motores;

Após inserir o código, defina os atributos públicos do script. Para defini-los basta arrastar cada parte do robô para o seu respectivo atributo.



#### 4.4 – Programando os sumôs remotamente



Agora nossos robôs estão prontos para serem controlados a distância! Resta apenas elaborar algoritmos em alguma linguagem de programação, e assim iniciar a competição para verificar o funcionamento dos nossos sumôs!

Esta seção apresentará um exemplo de programação para cada robô. Primeiramente programaremos o robô predador.

O robô predador funcionará baseado na diferença das distâncias lidas pelos seus sensores. Caso o sensor da esquerda possua valor menor que os outros, o robô deverá virar a esquerda. No caso do sensor do centro possuir o menor valor, o robô deverá seguir direto. E caso contrario, o robô deverá virar a direita. Este último caso também se aplica ao caso de não haver objetos a frente do robô, isso fará com que ele gire continuamente até encontrar seu oponente.

Sendo assim um código em Python para este comportamento é o seguinte:

```

import socket
import time

#API basica do para comunicar com o robo predador
def enviar_para_unity (msg):
    TCP_IP = '127.0.0.1'
    TCP_PORT = 12598 #porta do servidor Unity
    BUFFER_SIZE = 1024
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((TCP_IP, TCP_PORT))
    s.send(msg)
    data = s.recv(BUFFER_SIZE)
    s.close()
    return data

def frente():
    enviar_para_unity("frente;\n")

def direita():
    enviar_para_unity("direita;\n")

def esquerda():
    enviar_para_unity("esquerda;\n")

def tras():
    enviar_para_unity("tras;\n")

def parar():
    enviar_para_unity("parar;\n")

def distanciaE():
    return ord(enviar_para_unity("distanciaE;\n"))[0]

def distanciaC():
    return ord(enviar_para_unity("distanciaC;\n"))[0]

def distanciaD():
    return ord(enviar_para_unity("distanciaD;\n"))[0]

def cor():
    string = enviar_para_unity("cor;\n")
    numero = ord(string[0])*256*256 + ord(string[1])*256 + ord(string[2]) #0xRRGGBB
    return numero

while(1):
    dEsquerda = distanciaE()
    dDireita = distanciaD()
    dCentral = distanciaC()
    if(dEsquerda < dDireita and dEsquerda < dCentral):
        esquerda()
    else:
        if(dCentral < dDireita):
            frente()
        else:
            direita()

```

Uma vez com o comportamento do nosso robô predador feito, vamos agora elaborar o comportamento do nosso robô presa.

O robô presa possuirá o comportamento de fazer voltas próximo à linha limite da arena. Para isso, quando o sensor de cor detectar a cor branca o robô fará curva para a direita, caso contrário seguirá em frente.

Sendo assim um código em Python que define este comportamento é o seguinte:

```
import socket
import time

#API basica do para comunicar com o robo presa
def enviar_para_unity (msg):
    TCP_IP = '127.0.0.1'
    TCP_PORT = 12599 #porta do servidor Unity
    BUFFER_SIZE = 1024
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((TCP_IP, TCP_PORT))
    s.send(msg)
    data = s.recv(BUFFER_SIZE)
    s.close()
    return data

def frente():
    enviar_para_unity("frente;\n")

def direita():
    enviar_para_unity("direita;\n")

def esquerda():
    enviar_para_unity("esquerda;\n")

def tras():
    enviar_para_unity("tras;\n")

def parar():
    enviar_para_unity("parar;\n")

def cor():
    string = enviar_para_unity("cor;\n")
    numero = ord(string[0])*256*256 + ord(string[1])*256 + ord(string[2]) #0xRRGGBB
    return numero

while(1):
    tomVermelho = cor()/(256*256)
    if(tomVermelho < 100):
        frente()
    else:
        direita()
```

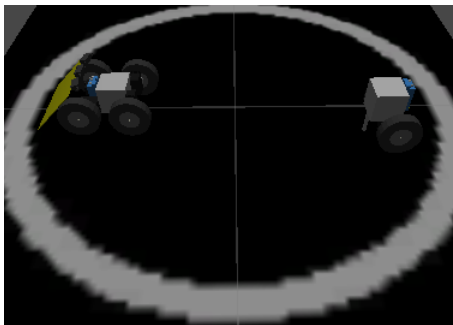


## 4.5 – Executando a competição

Nosso sumô de robôs está finalmente completo!! Agora só precisamos executar nossa simulação, o que é simples comparado a todo resto. Para executar o sumô de robôs basta seguir os seguintes passos:



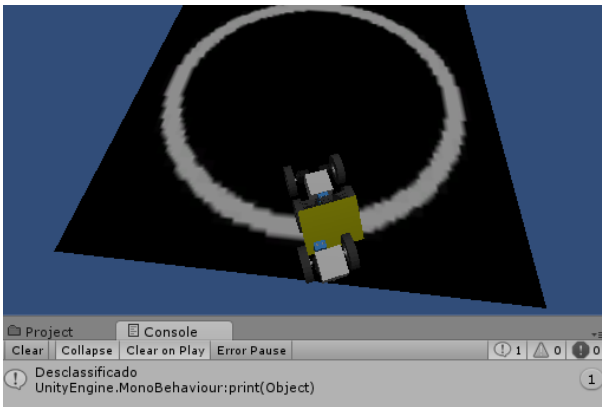
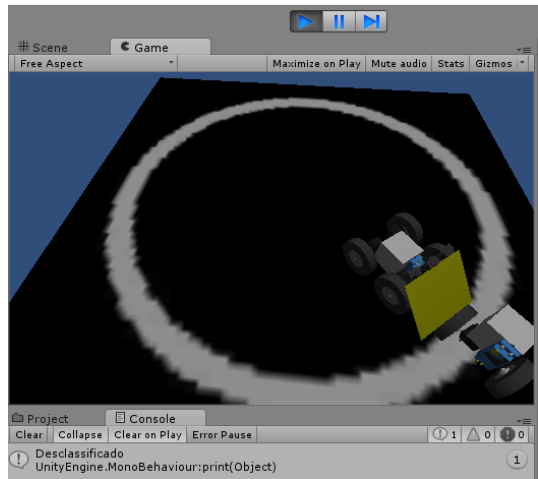
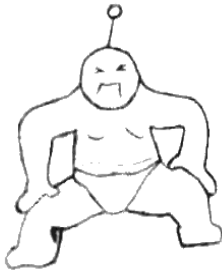
Posicione os robôs no cenário dentro do espaço da arena



Execute o cenário

E então execute os dois programas clientes





É parece que hoje é o dia do caçador...



**Desafio:** Agora que você já possui o conhecimento, construa um oponente a altura do campeão!!

# Referências

**Head First Labs.** Disponível em:

<http://www.headfirstlabs.com/readme.php>, acesso em 10/07/2015.

**Unity References.** Disponível em: <http://forum.unity3d.com/>, acesso em 10/07/2015.

**Unity Forums.** Disponível em:

<http://docs.unity3d.com/ScriptReference/>, acesso em 10/07/2015.

TANENBAUM A.S.: **Computer Networks** 4th edition.

**Regras SUMÔ.** Disponível em:

[https://www.robocore.net/upload/attachments/robocore\\_\\_regras\\_sumo\\_241.pdf](https://www.robocore.net/upload/attachments/robocore__regras_sumo_241.pdf)

, acesso em 18/07/2015.



