Programação II

Professor Jesse Nery

Quem sou?

- Professor do Instituto Federal Baiano:
  - Programação, Engenharia de Software, Games...
- Engenheiro de Computação (UNIVASF)
- Especialista em Redes de Computadores (ESAB)
- Mestre em Gestão e Tecnologias Aplicadas à Educação (UNEB)
- Doutorando em Educação e Contemporaneidade (UNEB)
- Participo do Centro de pesquisa e desenvolvimento de Jogos Comunidades Virtuais (UNEB)

O que veremos no curso

- Revisão de Algoritmos
- Elementos de programação de jogos
- Motores de Jogos e Revisão do Unity3d
- GameObjects e Propriedades
- Cenas, Sprites e Física
- Animação a partir de Spriteshet
- Entrada de dados (Teclado e Mouse)
- Movimentação de GameObjects

O que veremos no curso

- Colisões e Destruições de GameObjects
- Mudança de Cenas
- Inserir Efeitos Sonoros e Trilha
- Botões de interface e interação
- Armazenamento
- Menu Inicial e GameOver
- Protótipo de Plataforma 2D
- Exportando para web, desktop e mobile

Onde tirar dúvidas

- Com o professor
- Nos canais de youtube
- Nos manuais de motores de jogos
- Em fóruns de discursões
- Testando e tentando

Onde tirar dúvidas

unity | DOCUMENTATION Manual Scripting API Search manual... unity3d.com

Language: English

Unity Manual

- Unity Manual
- Working In Unity
- Graphics
- Physics
- Scripting
- Multiplayer and Networking
- Audio
- Animation
- UI
- Navigation and Pathfinding
- Unity Services
- Virtual Reality
- How to contribute to Unity
- Platform Specific
- Legacy Topics

Unity Manual

Unity Manual

Unity is made to empower you to create the best interactive entertainment or multimedia experience that you can. This manual is designed to help you learn how to use Unity, from basic to advanced techniques. It can be read from start to finish or used as a reference.

If you find that any question you have is not answered in this manual please ask on Unity Answers or Unity Forums. You will be able to find your answer there.

For documentation on the newest features introduced in 5.3, see What's New in 5.3

To start learning about some of the major new features introduced in Unity 5, you can check out the manual pages for Physically Based Shaders, Real-time Global Illumination, The Audio Mixer, HDR Reflection Probes, PhysX 3.3 and the new Animation System Features.

For information about upgrading your Unity projects from older versions, Upgrade Guide.

Revisão de Algoritmos

- O que é algoritmos?
  - Passo a passo para realizar uma tarefa
    - Receita de Bolo
    - Fluxograma de trabalho
    - Código de programas de computadores
  - Possuem uma estrutura bem definida
    - Inicio
    - Calculo ou Inputs/Outputs
    - Fim

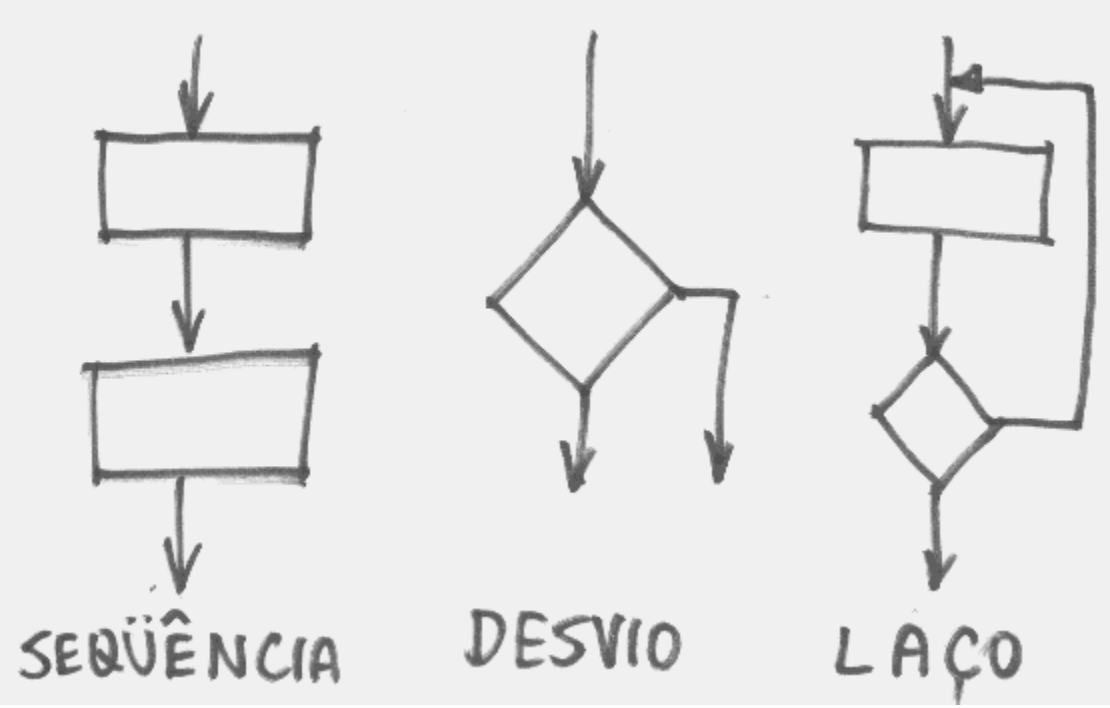
Exemplo de um algoritmo textual

- Algoritmo para atendimento de abertura de conta em banco
  1. Chama o próximo da fila
  2. Recebe os documentos
     a. Se documentos incompletos, pedir para buscar os documento e volta ao passo 1
  3. Preenche os formulário
  4. Arquiva os documentos e volta ao passo 1.

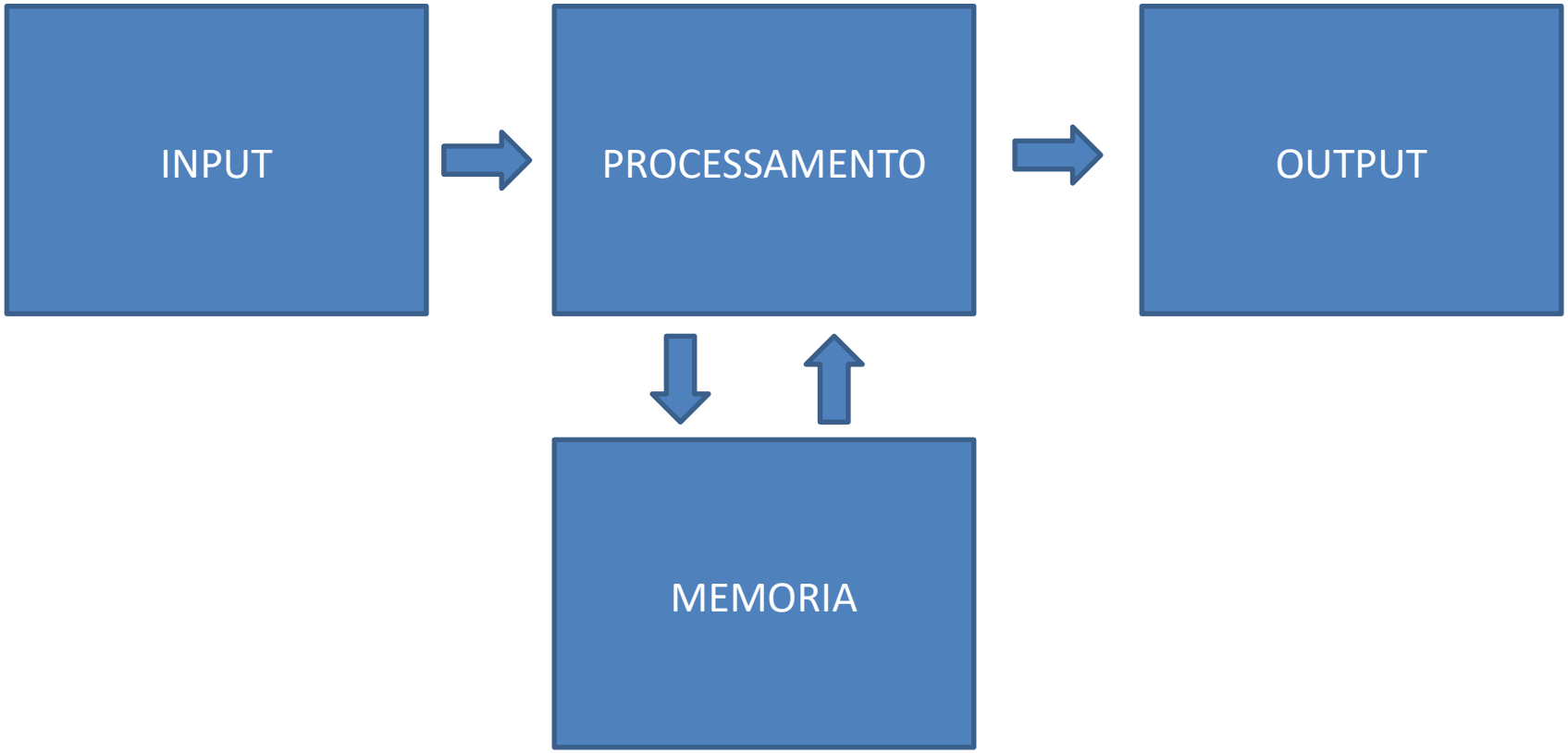
Exemplo de algoritmos

```
algoritmo "Equação do segundo grau"
// Seção de Declarações
var
x1,x2,a,b,c,delta: real
inicio
// Seção de Comandos
Escreval ("Escreva os coeficientes da equação.")
escreval("Coeficiente de x².")
Leia (a)
escreval("Coeficiente de x.")
Leia (B)
escreval("Termo independente.")
Leia (c)
delta<-(b*b-(4*a*c))
se delta<0 entao
  escreval ("As raízes não são reais")
senao
  x1<-(-b+delta^(1/2))/2*a
  x2<-(-b-delta^(1/2))/2*a
  escreval("As raízes são ",x1," e ",x2)
fimse
fimalgoritmo
```

Tipos de blocos em programação

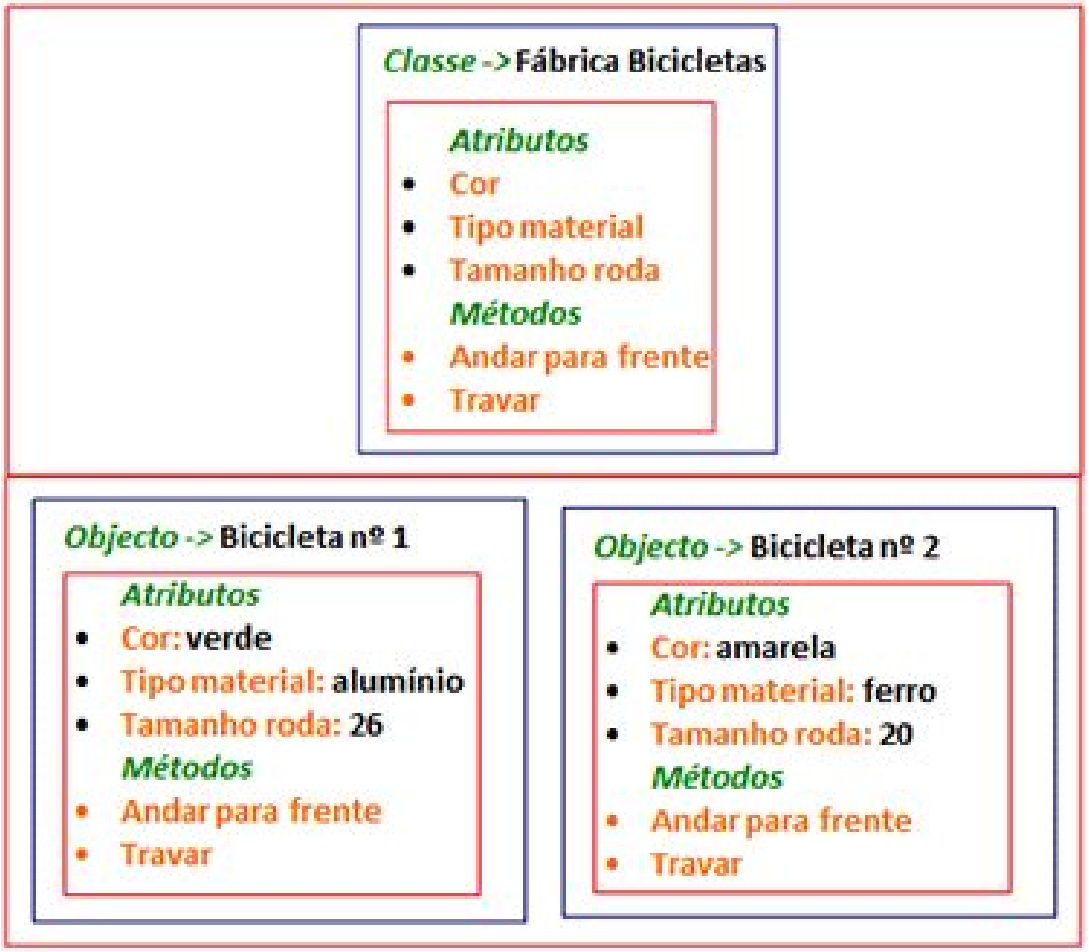


Algoritmos

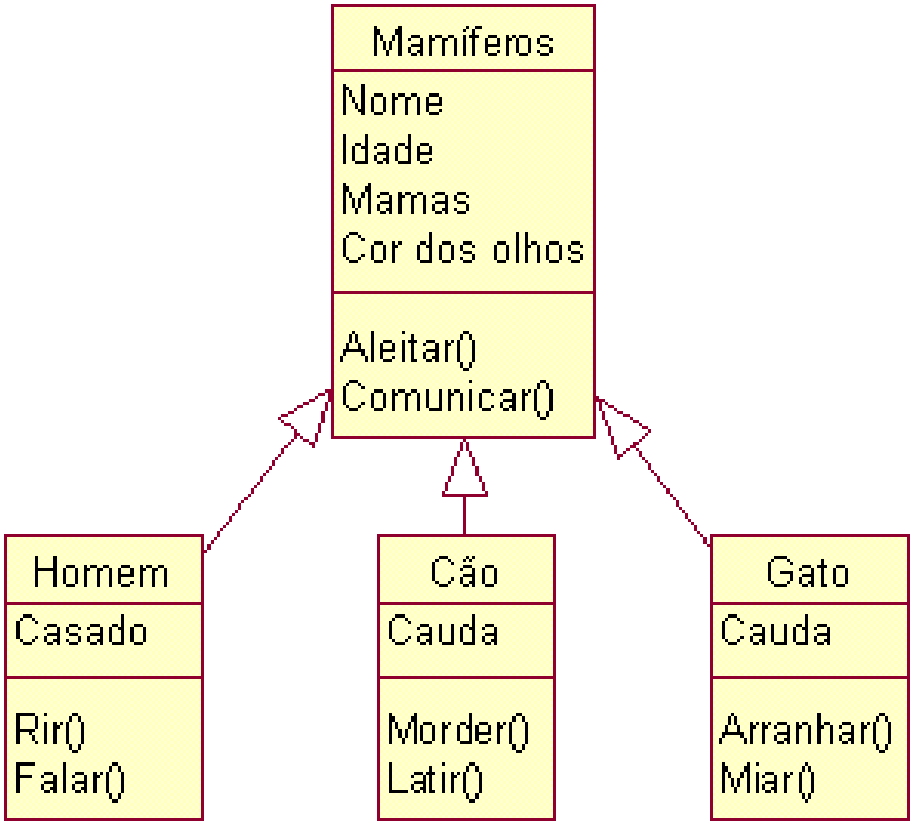


INPUTS/OUTPUTS?

Programação Orientada a Objetos



Programação Orientada a Objetos



Elementos da Programação de Jogo

- Todo jogo possui um loop infinito que sempre:
  - Checa se houve algum input do usuário
  - Tratar os inputs
  - Atualização da cena
    - GameObjects
    - Áudio
    - IA
  - Desenho de cena na tela

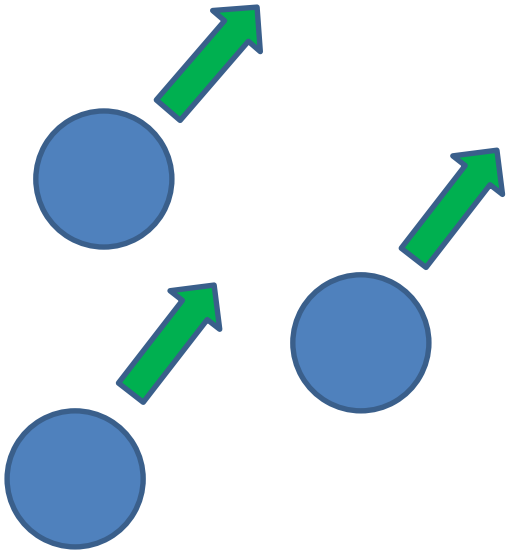
Elementos da Programação de Jogo

- Por conta desse loop infinito, o jogo fica sempre esperando um evento acontecer
  - Poder ser um input do usuário
  - Pode ser uma colisão de objetos na cena
  - Pode ter coletado todos os itens
  - Pode ter acabado o tempo para a quest
  - Pode ter acabo os recursos: balas, life, mana...

Elementos da Programação de Jogo

- Através dos códigos é possível dar vida aos elementos do jogo;
- Pode existir um código que opera em vários elementos, como é o caso do POO, quando se cria uma instância do objeto em uma cena uma região de memória é reservado para aquele objeto e os métodos só servirão para aquele objeto;

Exemplo de Inimigos



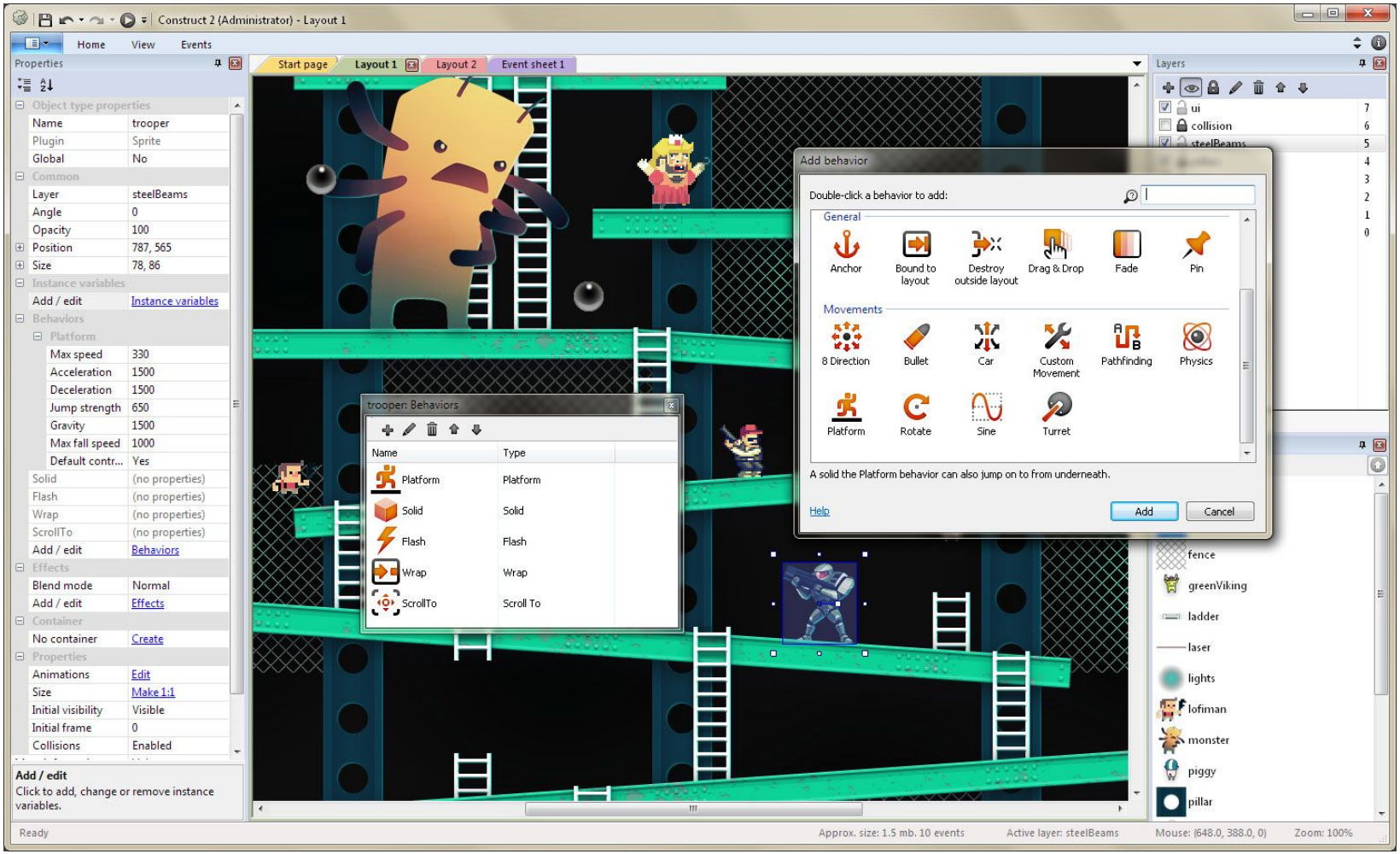
Inimigo:
-ângulo -vida
+resetarAngulo() +girar() +atacar()

Ordas:
-Inimigo[]
+resetarTodosAngulos() +TodosAtacar()

Motores de Jogos

- O que os motores fazem?
  - Física
  - Som
  - Compressões
  - Editor
  - Camera
- Quais os tipos de motores de jogos?
- Os motores de jogos fazem tudo?

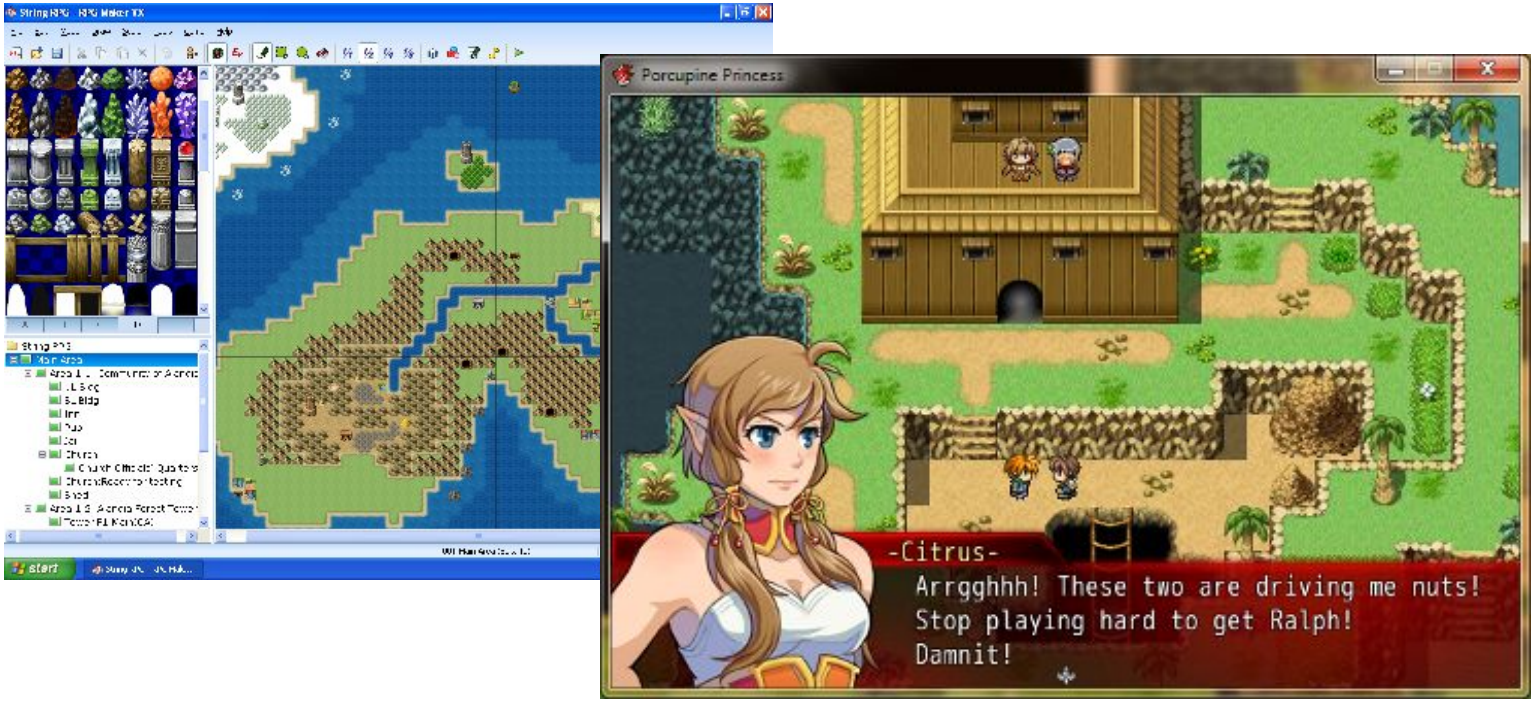
Construct 2


Construct 2 (Administrator) - Layout 1
Home
View
Events
Properties
Start page
Layout 1
Layout 2
Event sheet 1
Layers
Object type properties
Name
trooper
Plugin
Sprite
Global
No
Common
Layer
steelBeams
Angle
0
Opacity
100
Position
787, 565
Size
78, 86
Instance variables
Add / edit
Instance variables
Behaviors
Platform
Max speed
330
Acceleration
1500
Deceleration
1500
Jump strength
650
Gravity
1500
Max fall speed
1000
Default contr...
Yes
Solid
(no properties)
Flash
(no properties)
Wrap
(no properties)
ScrollTo
(no properties)
Add / edit
Behaviors
Effects
Blend mode
Normal
Add / edit
Effects
Container
No container
Create
Properties
Animations
Edit
Size
Make 1:1
Initial visibility
Visible
Initial frame
0
Collisions
Enabled
Add / edit
Click to add, change or remove instance variables.
trooper: Behaviors
Name
Type
Platform
Platform
Solid
Solid
Flash
Flash
Wrap
Wrap
ScrollTo
Scroll To
Add behavior
Double-click a behavior to add:
General
Anchor
Bound to layout
Destroy outside layout
Drag & Drop
Fade
Pin
Movements
8 Direction
Bullet
Car
Custom Movement
Pathfinding
Physics
Platform
Rotate
Sine
Turret
A solid the Platform behavior can also jump on to from underneath.
Help
Add
Cancel
ui
collision
steelBeams
fence
greenViking
ladder
laser
lights
lofiman
monster
piggy
pillar
Ready
Approx. size: 1.5 mb. 10 events
Active layer: steelBeams
Mouse: (648.0, 388.0, 0)
Zoom: 100%

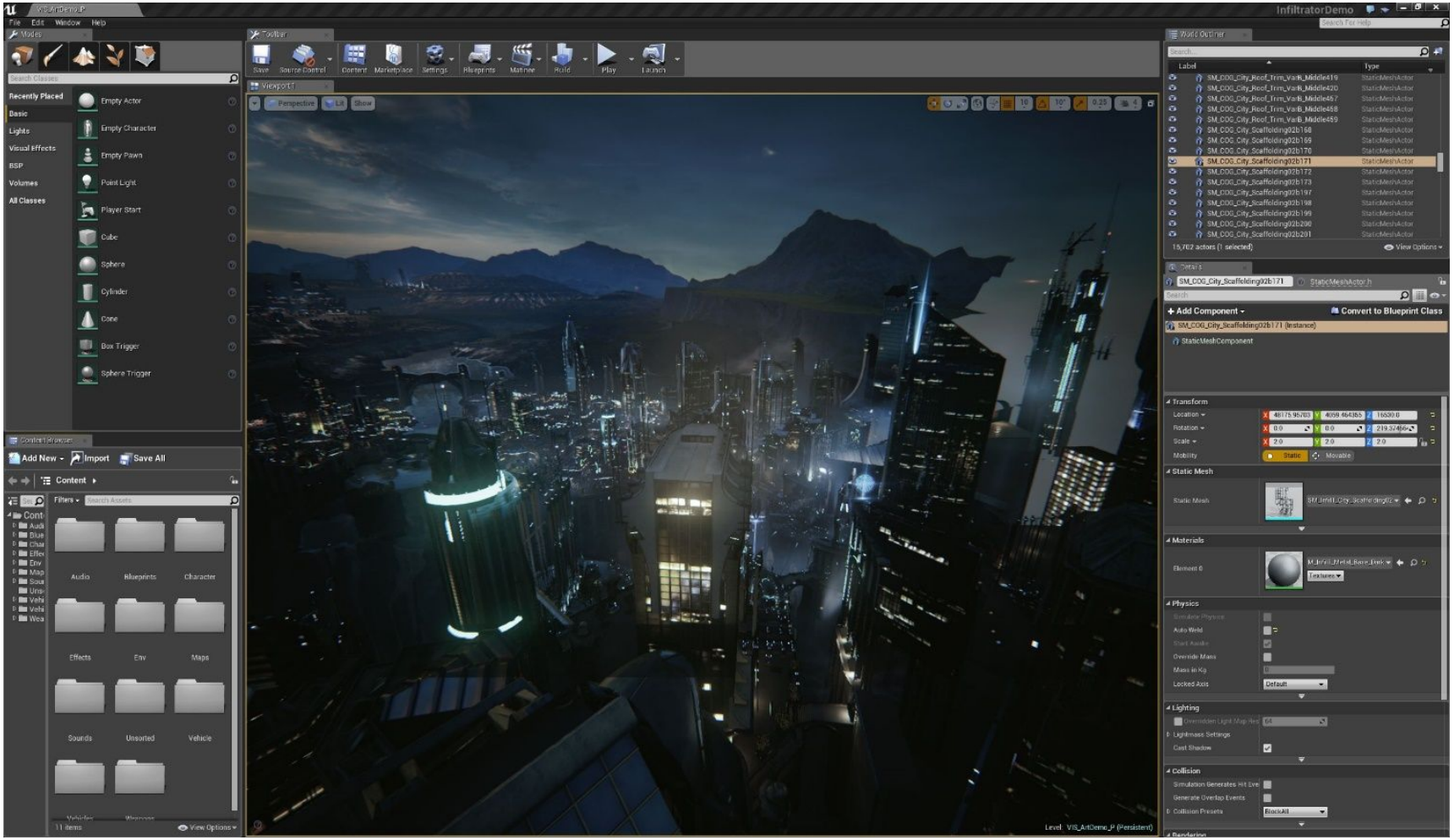
Game Maker


BugHunt.project.gmx* - GameMaker:Studio
File Edit Resources Scripts Run Window Help
Target: HTML5
Configuration: Default
Manage
Sprites
Froad
sFrog
sFrogIdle2
sFrogIdle3
sFrogDown
sFrogUp
sFrogRight
sFrogSwallow
sFrogSwallowGood
sFrogSwallowBad
sFrogMouth
sTongue
sTongueTip
sTongueTest
sFrogDie
sFrogFart
sFartCloud
sFroadFallMenu
sFroadCelebrate
sFroadTitleMask
sFroadDownBits
Bugs
Menus
HUD Elements
Tutorial
sPondLocked
sTutSign
Sounds
Backgrounds
bForestDay
bForestNight
bPondDay
bPondNight
Paths
Scripts
Fonts
Time Lines
Objects
Rooms
rTitle
rMenu
rGame
rTutorial
Included Files
Extensions
Game Information
Global Game Settings
Sprite Properties: sFrogSwallow
Sprite Properties: sFrogSwallowGood
Name: sFrogSwallowGood
Load Sprite
Edit Sprite
Width: 128 Height: 204
Number of subimages: 15
Show: 0
Origin
X 67 Y 135
Center
OK
Collision Checking
Precise collision check
Separate collision mas
Modify Mask
Texture Settings
Tile: Horizontal
Tile: Vertical
Used for 3D
(Must be a power of 2
Texture Group:
Default
Sprite Editor: sFrogSwallowGood
File Edit Transform Images Animation
Show Preview
image 9
image 12
Frames: 15 Size: 128 x 204 Memory: 1.0 MB
Image Editor: sFrogSwallowGood
File Edit View Transform Image
Size
Hardness
128
Colors
Left: Right:
Opacity
255
Color Mode
Blend
Replace
Spray with the mouse, <Shift> for hor/vert
(42,81)
Zoom: 400%
Size: 128 x 204
Memory: 104 KB
CompileForm - Compiled at: 11:43:46
Compile
Source Control
Repository UUID: 80c69118-2996-634b-8bfe-a1a527812bd6
Revision: 15051
Node Kind: directory
Schedule: normal
Last Changed Author: jack
Last Changed Rev: 14365
Last Changed Date: 2012-05-02 09:59:40 +0100 (Wed, 02 May 2012)
exitcode = 0
Refreshing SVN Status...Finished
YOYO GAMES

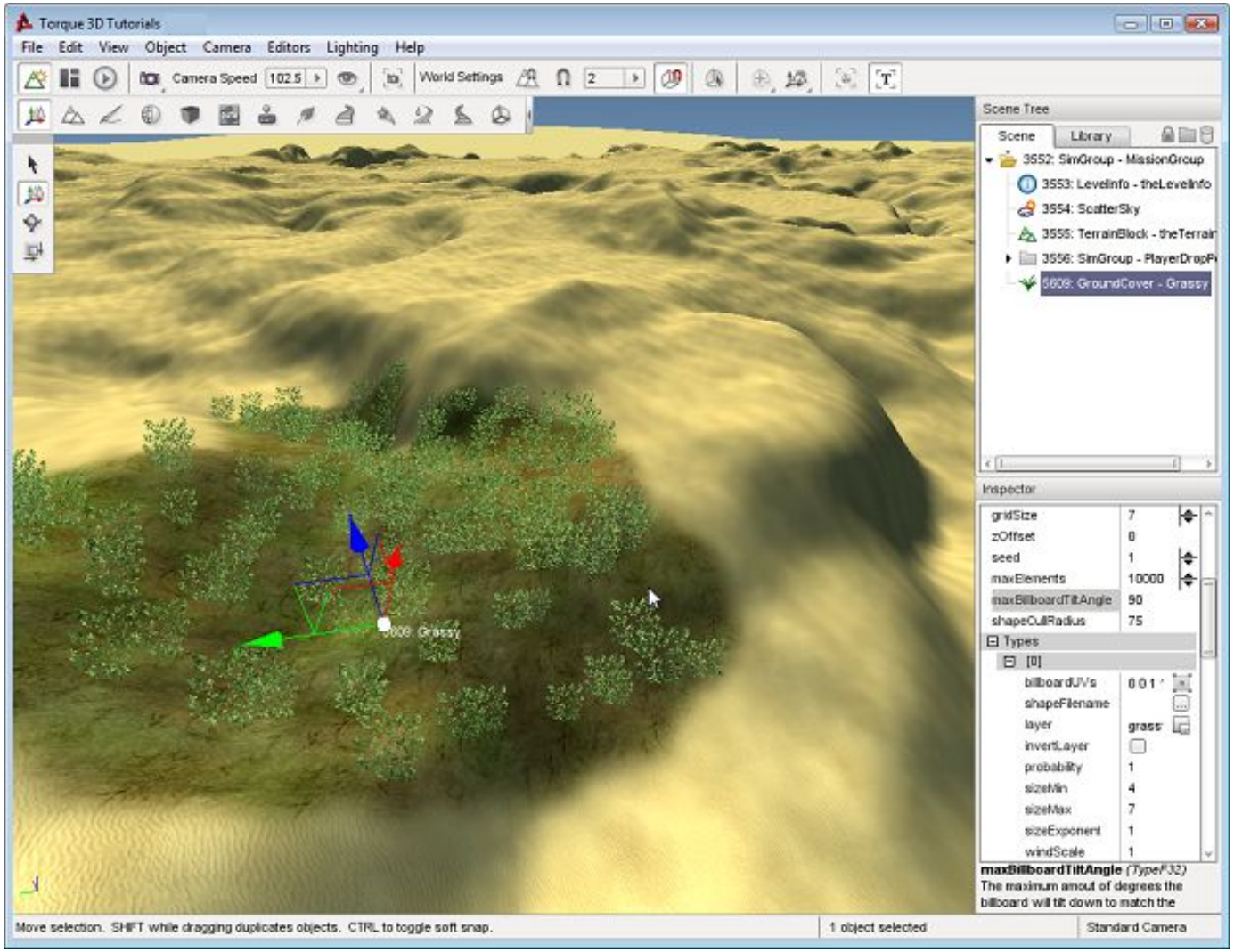
RPG MAKER


Porcupine Princess
-Citrus-
Arrgghhh! These two are driving me nuts!
Stop playing hard to get Ralph!
Damnit!

Unreal Engine



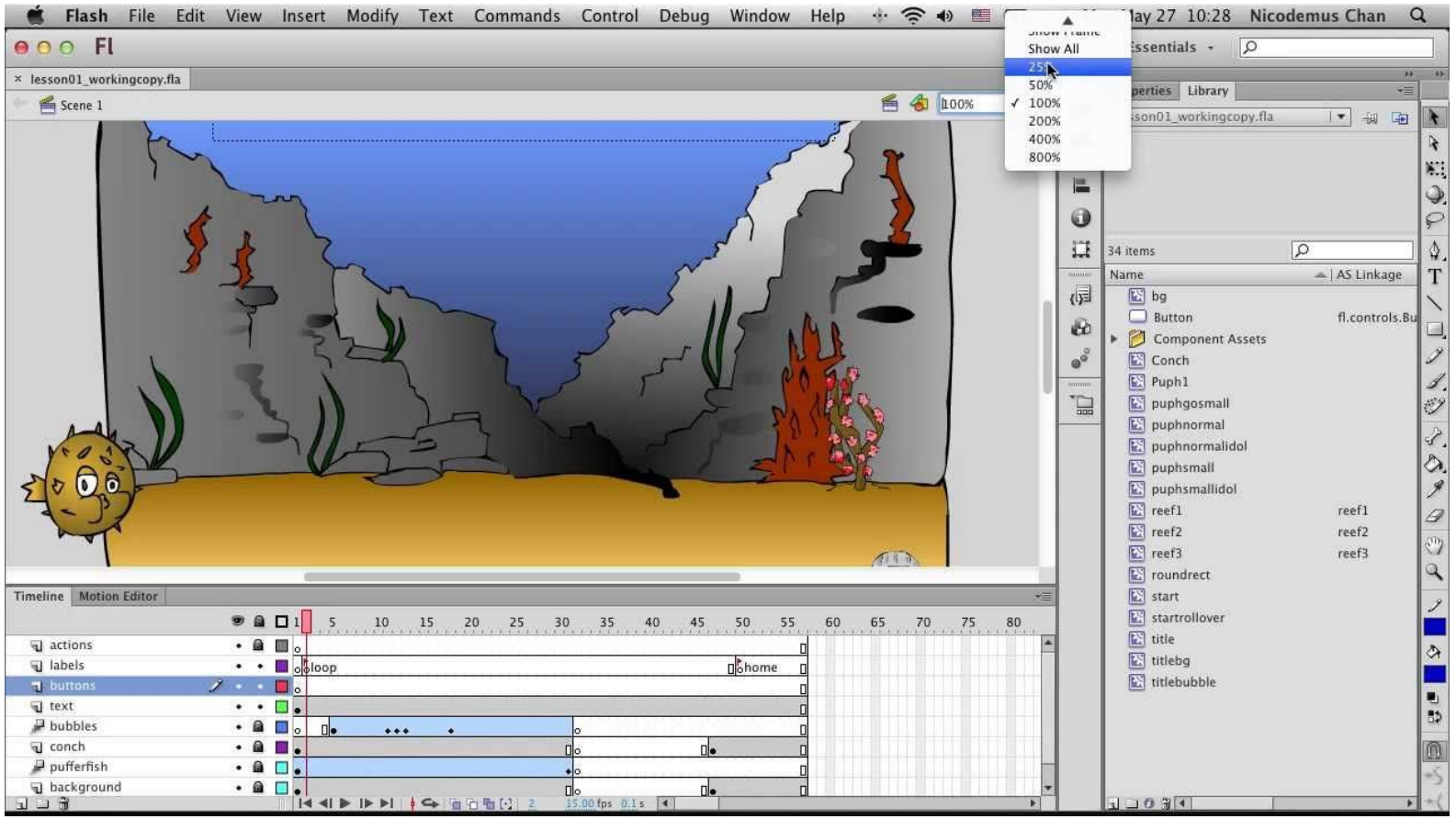
Torque3D


Torque 3D Tutorials
File Edit View Object Camera Editors Lighting Help
Camera Speed 102.5
World Settings
Scene Tree
Scene Library
3552: SimGroup - MissionGroup
3553: LevelInfo - theLevelInfo
3554: ScatterSky
3555: TerrainBlock - theTerrain
3556: SimGroup - PlayerDropP
6609: GroundCover - Grassy
Inspector
gridSize 7
zOffset 0
seed 1
maxElements 10000
maxBillboardTiltAngle 90
shapeCullRadius 75
Types
[0]
billboardUVs 0 0 1
shapeFilename
layer grass
invertLayer
probability 1
sizeMin 4
sizeMax 7
sizeExponent 1
windScale 1
maxBillboardTiltAngle (TypeF32)
The maximum amout of degrees the billboard will tilt down to match the
Move selection. SHIFT while dragging duplicates objects. CTRL to toggle soft snap.
1 object selected
Standard Camera

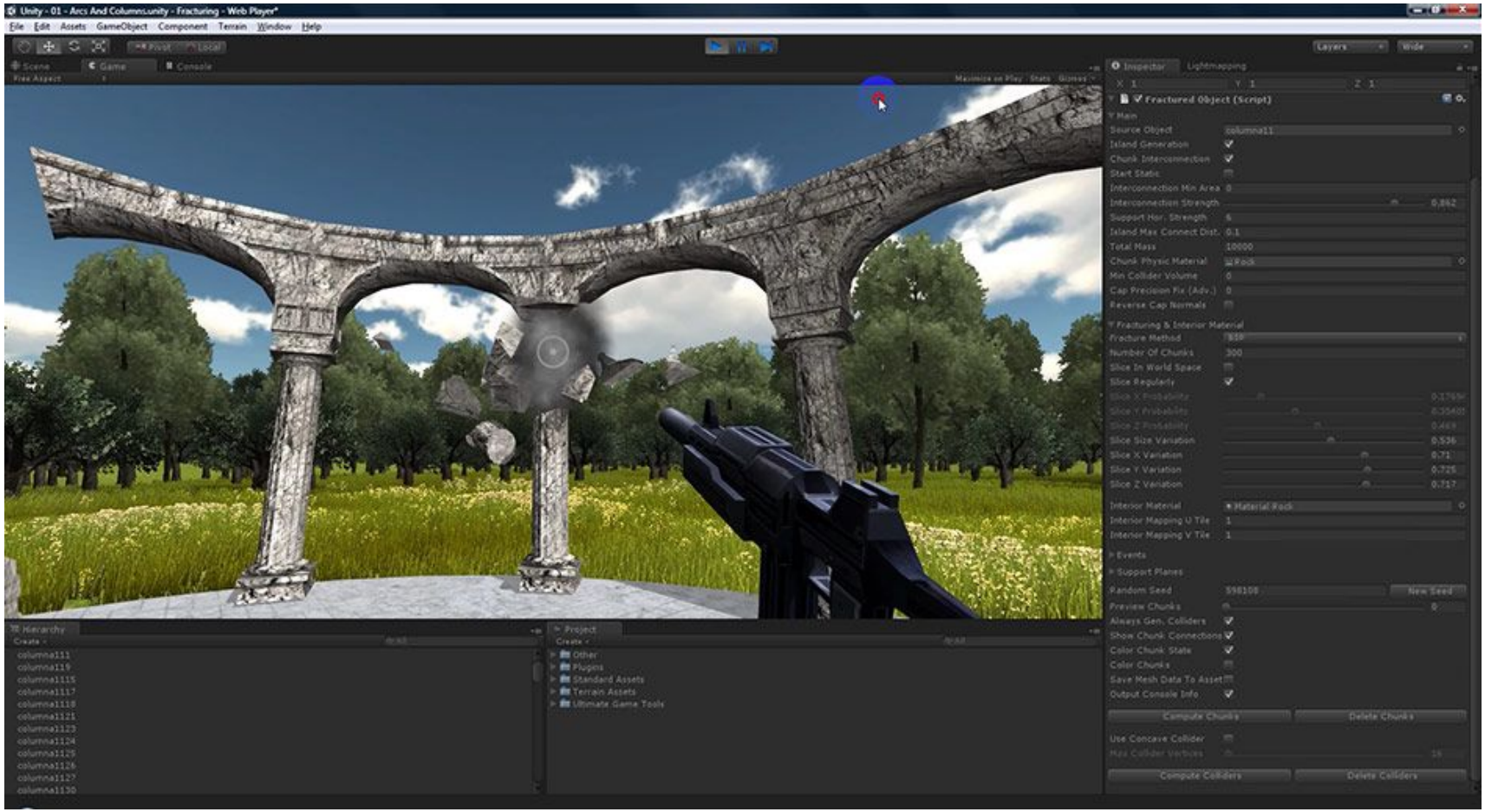
Blender


File Add Timeline Game Render Help
View Select Curve
lower brows
raise brows
mouth close
mouth smaller
smile left
smile right
nose up
View Select Key
View Select Object
Panels

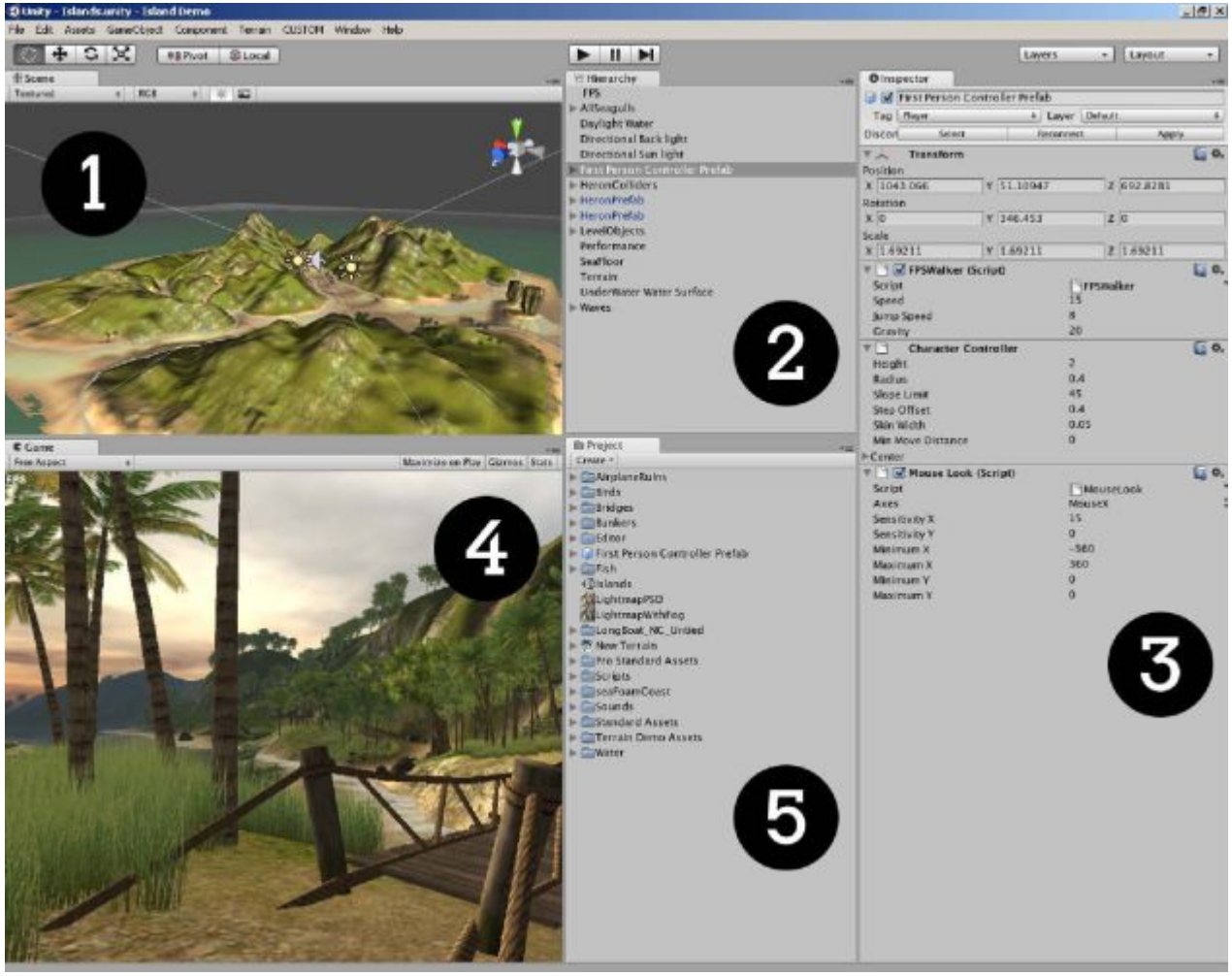
Action Script e não motores



Unity3D


Unity - 01 - Arcs And Columns.unity - Fracturing - Web Player*
File Edit Assets GameObject Component Terrain Window Help
Fractured Object (Script)
Source Object
Island Generation
Chunk Interconnection
Start Static
Interconnection Min Area
Interconnection Strength
Support Hor. Strength
Island Max Connect Dist.
Total Mass
Chunk Physic Material
Min Collider Volume
Cap Precision Fix (Adv.)
Reverse Cap Normals
Fracturing & Interior Material
Fracture Method
Number Of Chunks
Slice In World Space
Slice Regularly
Slice Size Variation
Slice X Variation
Slice Y Variation
Slice Z Variation
Interior Material
Interior Mapping U Tile
Interior Mapping V Tile
Events
Support Planes
Random Seed
Preview Chunks
Always Gen. Colliders
Show Chunk Connections
Color Chunk State
Color Chunks
Save Mesh Data To Asset
Output Console Info
Compute Chunks
Delete Chunks
Use Concave Collider
Compute Colliders
Delete Colliders
Other
Plugins
Standard Assets
Terrain Assets
Ultimate Game Tools

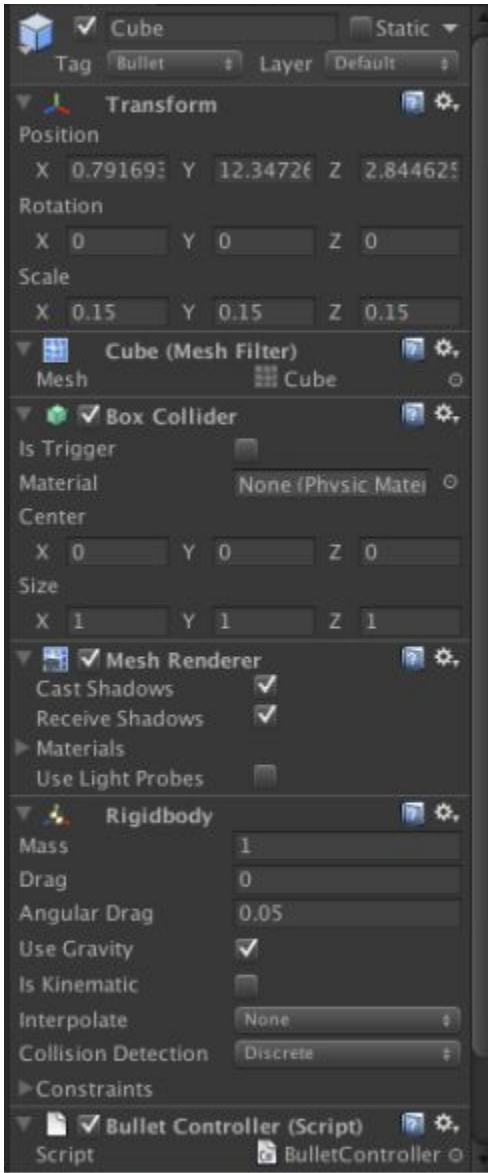
Elementos do Unity



1. **Cena:** Onde o jogo é construído
2. **Hierarquia:** Lista de GameObjects na cena
3. **Inspector:** Configurações do objeto selecionado
4. **Game:** Prévia da tela do jogo
5. **Project** Lista de assets do projeto

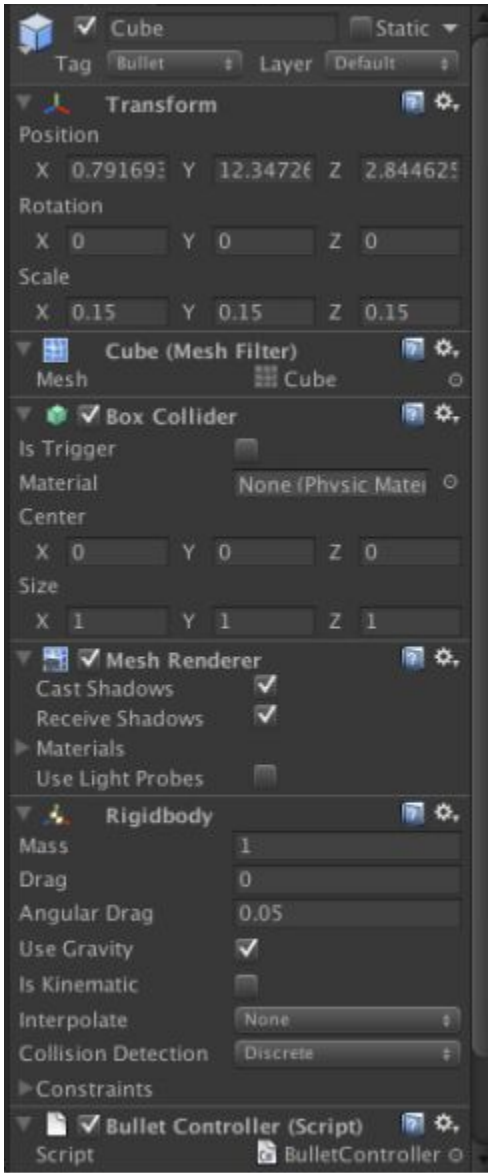
GameObjects e Propriedades

- Ao colocarmos qualquer coisa na cena, esse objeto surgirá na lista da hierarquia e recebe o nome de GameObject;
- Todo GameObject possui pelo menos a propriedade Transform que aparece no inspector quando esse está selecionado, inclusive os objetos vazios;



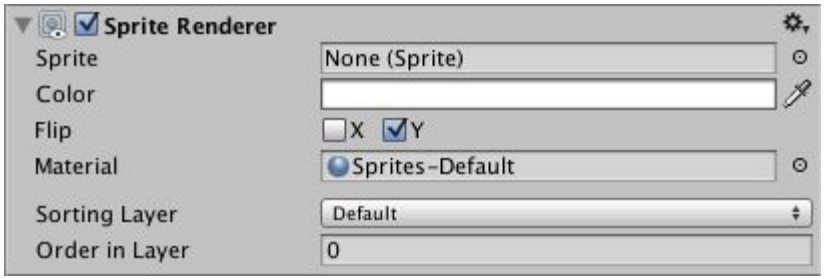
GameObjects e Propriedades

- A propriedade transform possui: Posição, Rotação e Escala. Todas em 3 dimensões, inclusive para os trabalhos feitos em 2D.
- Outras propriedades podem ir sendo adicionadas ao objetos: Corpo Rígido, Animação, Sprite e Scripts



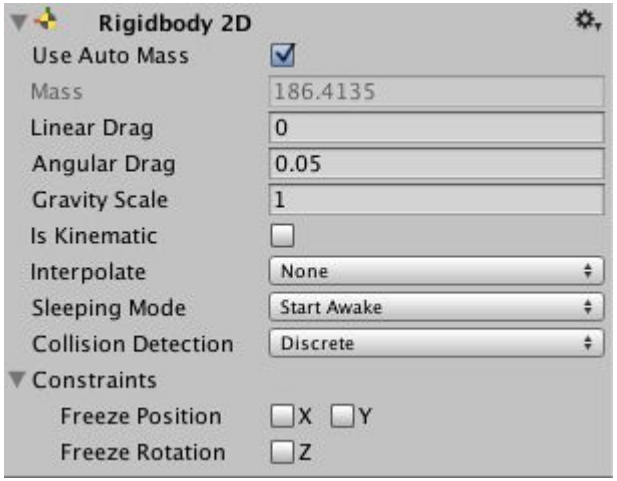
Sprite

- Um GameObject pode possuir uma Imagem/Sprite;
- Essa Sprite possui propriedades
  - Imagem
  - Cor
  - Camada
  - Ordenação



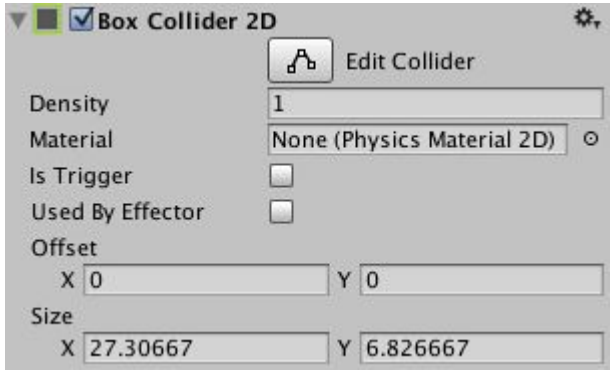
Corpo Rígido

- Um objeto pode possuir um elementos de física, como:
  - Massa
  - Força de gravidade
  - Travar as movimentações em um eixo ou um ângulo
  - Pode ser em 2D ou 3D



Colisores

- Um objeto pode possuir caixas de colisões, círculos ou polígonos (2D) ou cubos, esferas e outros em 3D;
- Serve para detectar quando um elemento colidiu com outro e tomar determinadas ações;



Animações 2D

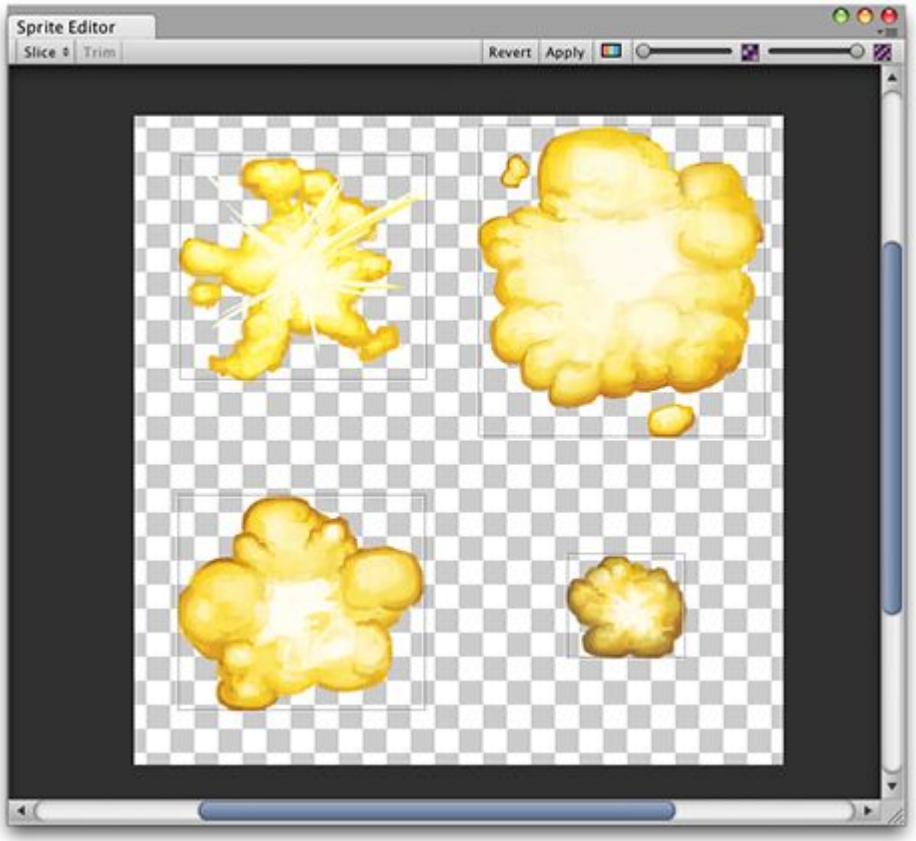
- As animações podem ser realizadas a partir de uma sequencia de frames de um objeto fazendo com que essa se repita
- O unity já possui um editor de Sprite que pode fazer os recortes automáticos ou ate manuais

Animações 2D

- Selecionado uma imagem do projeto, pode-se escolher a imagem como uma textura do tipo sprite e no modo múltiplos.
- Depois abrir o editor de sprites para separar os sprites

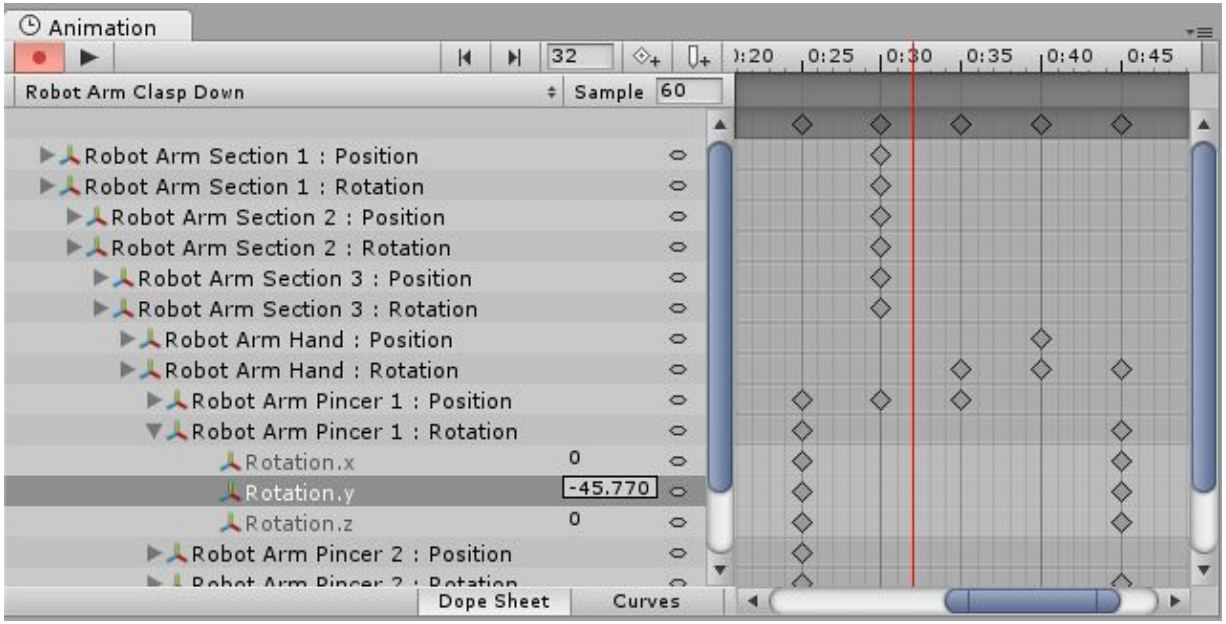
Sprite Editor

- Em slice pode ecolher:
  - Automatico
  - Manual
  - Grid



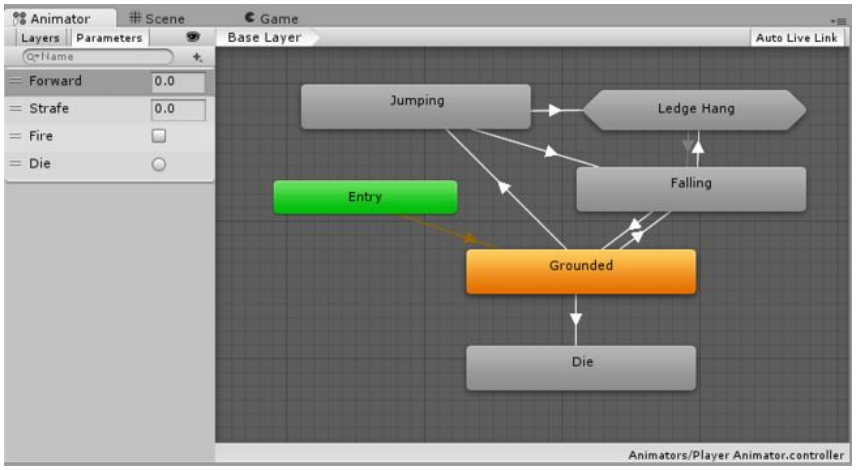
Criando Animação

- Após possuir os sprites, é só seleciona-los e arrastar para a aba Animation, podendo ajustar os FPS, tempo de cada frame, tudo em uma timeline;



Criando Animação

- Cada animação criada, gera um estado para aquele GameObject
- Esses estados podem ser visualizados na aba Animator
- Deve-se criar as transições e as condições para mudanças de estados



Criando Animações

- As variáveis criadas no Animator para mudanças de estados podem ser acessados via código de acordo com o contexto:
  - Se o jogador faz com que o personagem ande, muda-se o valor da variável de transição de velocidade e ativa a animação de andar
  - Se o para, a variável fica sem algum valor e a animação de parado será ativado

Códigos do C# no Unity

- Possuem as Bibliotecas
- E uma classe que sempre inicia variáveis (Start) e outra que sempre irá repetir o que estiver dentro dela (Update)

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class MainScript : MonoBehaviour {
5
6  // Use this for initialization
7  void Start () {
8  Debug.Log ("Hello World");
9  }
10 // Update is called once per frame
11 void Update () {
12 }
13 }
```

Códigos do C# no Unity

- o Debug.Log(“Hello World!”), imprime no console do unity valores ou palavras para auxiliar a encontrar erros.
- Se existir 2 GameObjects com esse script, aparecerá 2 “Hello World!” no console do unity, pois cada GameObject irá executar um script, isso nos dá a ideia de paralelismo

Códigos do C# no Unity

- Se existir no Update possuir Debug.Log(“Hello World!”), o jogo irá indefinidamente imprimindo essa frase no console do unity
- Pode ser utilizado para fazer um inimigo andar e quando chegar em um certo limite girar, até encontrar o jogador

Tipos de Variáveis em C#

- Tipos numéricos
  - Int ...-2 -1 0 1 2 ...
  - Float -1.0...0.99999 ... 0 ... 0.999999 .... 1 .....
- Tipos Literais
  - Char ‘a’, ‘z’, ‘0’, ‘1’, ‘\n’...
  - String “nome”, “Olá”, “999111”
- Tipos Boleanos
  - True e False
- Coletaneas
  - Array
  - Vetores

Operações com as variáveis

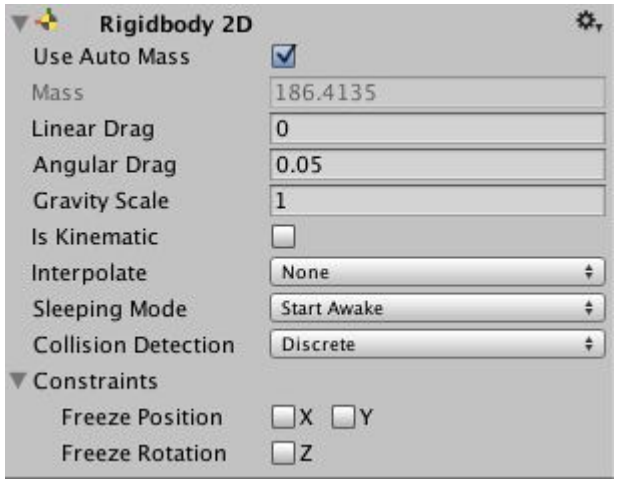
- Geralmente utilizado nos comandos
- Atribuição: a=b;
- Com os numéricos
  - Parênteses (maior prioridade): a= (b+c)
  - Multiplicar e Dividir = a: 5*b/2;
  - Somar e Subtrair (menor prioridade): a=a+1;
- Literais
  - Concatenar (+): nome = primeiroNome + “Filho”

Comparações

- Geralmente utilizado em if else ou em laços
  - Igualdade: a==b
  - Maior: a>b
  - Maior igual a>=b
  - Menor: a<b
  - Menor igual: a<=b
  - Diferente: a!=b

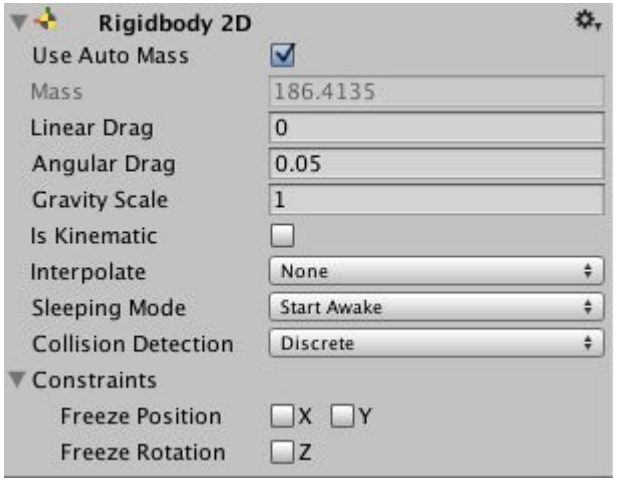
Variáveis Publicas e Privadas

- Quando se cria um script/classe as variáveis podem assumir dois tipos: publicas e privadas
- Se omitir essa declaração, entende-se que seja privada;
- Serve para que outras classes possam acessar os atributos, inclusive o próprio unity poder manipula-lo no editor



Variáveis Publicas e Privadas

- No caso do Rigidbody 2D que já vem pronto no unity, a gravidade poderia ter sido um parâmetro que não mudasse para nenhum objeto e esse atributo deveria ser privado, porém a massa pode ser mudado de acordo com o objeto da cena e alterado no próprio unity, não necessitando o acesso ao código;



Variáveis Publicas e Privadas

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class MainScript : MonoBehaviour {
5
6  public float massa;
7  private float gravidade;
8
9  // Use this for initialization
10 void Start () {
11 gravidade = 10.0f;
12 Debug.Log (“Gravidade” + gravidade);
13 }
   // Update is called once per frame
   void Update () {
   }
   }
```

Declaração de variáveis

- Regras
  - Não devem ser nomes de palavras reservadas da linguagem (ex: for, if, public, void, int...)
  - Devem começar com letras e não números ou caracteres especiais (exemplo correto: nome, sexo, idade, media1)
  - Não devem possuir espaço em branco (exemplo correto: primeiro_nome, sexo_masculino)
  - Da forma que declarar devem ser utilizados (minúsculas e maiúsculas)
- Convenções
  - Devem começar com letras minúsculas e se for composto o inicio da segunda e/ou outras palavras maiúsculas (ex: primeiraMediaDoAno, ultimoNome...)

Declaração de Variáveis

- Existe variáveis do tipo estático que podem auxiliar uma contagem global, como o número de inimigos já criados ou de pontos ou life e na medida que perde ou ganha esses pontos outras classes podem acessá-las e atualizar-las ou se várias instâncias dos objetos forem criados e queira saber quantos já foram criados, uma variável estática pode acumular esse valor, lembra as variáveis globais.
- Public Static int quantidade = 0;

Funções Condicionais

- Condicional simples
  - If(condição){
    Comando
  }

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class MainScript : MonoBehaviour {
5    public float massa;
6    private float gravidade;
7
8    void Start () {
9      massa = 0.0f;
10     gravidade = 10.0f;
11     Debug.Log ("Gravidade" + gravidade);
12   }
13   void Update () {
15     if(massa<10.0f){
16       massa = massa + 1.0f;
17     }
18   }
19 }
```

Funções Condicionais

- Condicional composta
  - if(condição){
    Comando1;
    }
    Else {
    Comando2;
    }

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class MainScript : MonoBehaviour {
5    public float massa;
6    private float gravidade;
7    private bool crescimento;
8
9    void Start () {
10     massa = 0.0f;
11     gravidade = 10.0f;
12     crecimento = true;
13     Debug.Log (“Gravidade” + gravidade);
14   }
15   void Update () {
16     if(crescimento){
17       massa = massa + 1.0f;
18     }
19     else {
20       massa = massa - 1.0f;
21     }
22     if(massa==10.0f || massa == 0.0f){
23       crescimento = - crescimento;
24     }
25   }
26 }
```

Função de repetição

- for(inicio; condição; acréscimo){

  Comandos;

  }

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class MainScript : MonoBehaviour {
5    public float massa;
6    private float gravidade;
7    private int i;
8
9    void Start () {
10     massa = 0.0f;
11     gravidade = 10.0f;
12     Debug.Log (“Gravidade” + gravidade);
13     for(i=0;i<10;i=i+1){
15       massa = massa + 1;
16     }
17   }
18   void Update () {
19   }
   }
```

Vector2 e Vector3

- São variáveis que auxiliam armazenar posições, ângulos, escalas, tanto em 2D ou em 3D;
- Podem ser feitos cálculos com essas variáveis;
- As propriedades de posição podem ser acessadas a partir do transform. Ex:
  - transform.position retorna um vector3
  - transform.position.x retorna um float
- Possuem constantes que auxiliam alguns cálculos, como vetor unitários ou direcionais. Ex: vector3.up ou vector3.identity

Funções de Movimentação

- transform.Translate(Vector3.forward * Time.deltaTime);
- Time.deltaTime faz com que normalize o tempo idependente de plataforma para nao acontecer de ser 10 avanços por quadro e sim 10 avanços por segundos
- Lembrar que quando usamos o “transform” com letras minuculas estamos falando dos valores do gameObject que está utilizando, quando utilizamos o “Time” com letra maiúsculas estamos utilizando as funções estáticas da Classe.

Funções de Movimentação

- transform.Translate(Vector3.forward * Time.deltaTime);
- Essa função pode está dentro do Update, porém o objeto irá andar para sempre
- Se estiver dentro de um if que depende de uma entrada do usuário vai andar quando o usuário apertar o botão;

Funções de Movimentação

- transform.Rotate(Vector3.right * Time.deltaTime);
- Essa função faz rotações em uma direção dada;
- Para acessar os valores de posição, ângulo e escala pode ser direto, porém para altera-los diretamente devem ser feitos com um vetor novo. Ex:
  - transform.position = new Vector3(0, 0, 0);
  - transform.position = new Vector3(transform.position.x, 0, 0);

Funções de Input

- Do teclado pode ser capturado das seguintes formas, dentro do Update:
- Enquanto estiver apertado

```
if (Input.GetKey("up"))
transform.Translate(Vector3.forward * Time.deltaTime);
```

- Quando apertar

```
if (Input.GetKeyDown(KeyCode.W))
```

- Quando Soltar

```
if (Input.GetKeyUp(KeyCode.W))
```

Funções de Input

- De um mouse, para saber se algum botão foi pressionado, ou localização do ponteiro:

```
if (Input.GetMouseButtonDown(0))
        Debug.Log(“Botão esquerdo.");

    if (Input.GetMouseButtonDown(1))
        Debug.Log(“Botão direito.");

    if (Input.GetMouseButtonDown(2))
        Debug.Log(“Botão do meio.");
```

Funções de Input

- Do mouse ou touch de dispositivos, a partir de um método, que um objeto com um Colider ou um guiTexture possuir

```
public class ExampleClass : MonoBehaviour {
    void OnMouseDown() {
        Debug.Log("Apertou");
    }
}
```

Colisão

- Os objetos que possuem colisores podem detectar a colisão com outros objetos através da função:

```
 void OnCollisionEnter(Collision collision) {
      Debug.Log(“Colidiu”);
}

void OnCollisionEnter2D(Collision2D coll) {
      Debug.Log(“Para objetos em 2D”);
}
```

Colisões

- Mas precisa saber com o que colidiu, é possível saber pois aquilo que acabou de colidir fica armazenado na variável passado na função

```
void OnCollisionEnter2D(Collision2D coll) {
    if (coll.gameObject.tag == “Inimigo")
        Debug.Log(“Colidiu no inimigo”);
}
```

- A “tag” pode ser adicionado a vários objetos, mas se queira um objeto especifico pode comparar com a propriedade name.

Destruir

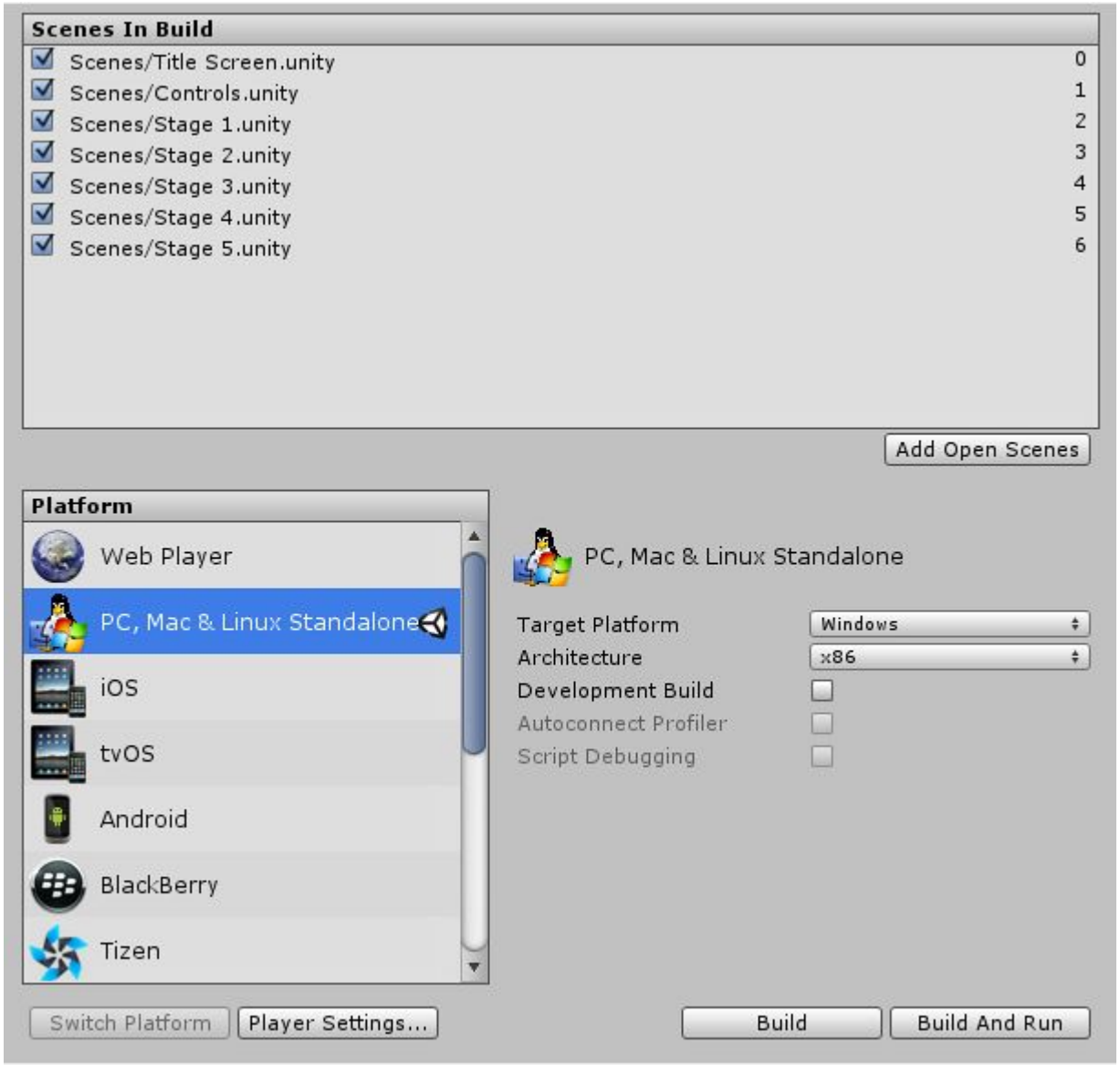
- As vezes destruir um gameObject quando uma moeda é coletado ou um life ou uma bala acerta um inimigo, usa-se a função Destroy
- Geralmente é usado com um colisor e vai depender se quer destruir quem carrega o script Destroy(gameObject) ou a variável que acabou de colidir. Ex:

```
void OnCollisionEnter2D(Collision2D coll) {
        if (coll.gameObject.tag == “Inimigo")
         Destroy(coll.gameObject);
    }
```

Cena

- Cada cena criada deve dar ideia de uma fase, e não ter muitos elementos
- Pode ter cenas para menus, fases, game over e outros...
- As cenas devem ser adicionados no projeto no build para que possa ser gerado o jogo
- As cenas podem ser acessados com os números da lista ou pelos nomes das cenas com o método Application.LoadLevel(“nomeDaCena”)

Lista de Cenas no Build



Cenas

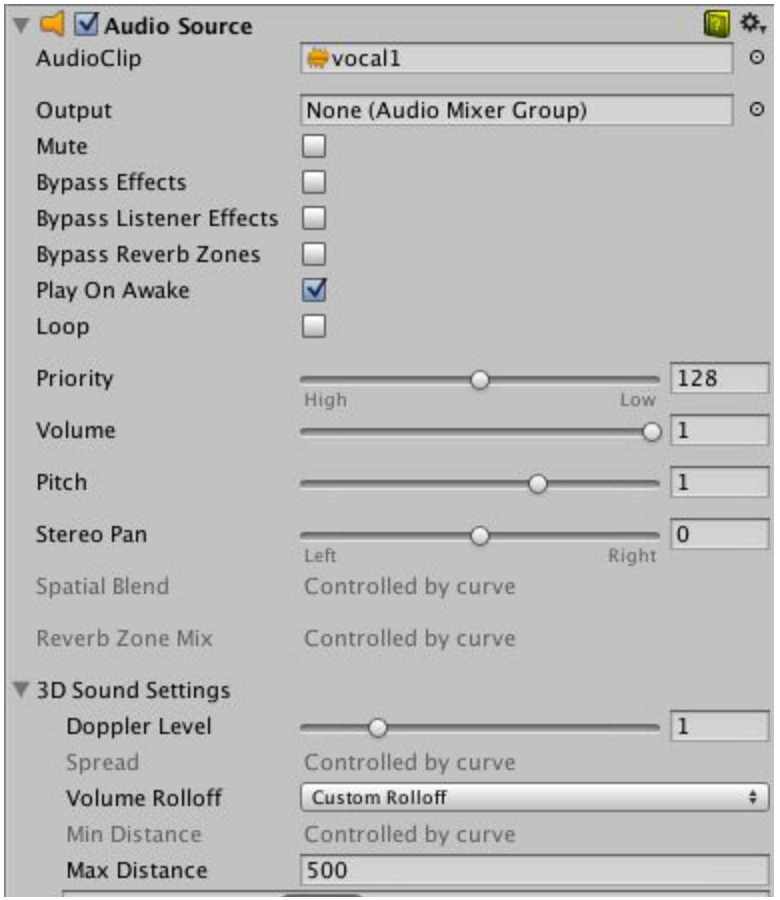
- As cenas podem ser chamadas em um menu com botões usando a função OnMouseDown e depois o LoadLevel;
- Pode ainda ser chamado quando colide com uma determinada porta, usando as funções de colisões
- Ou ainda quando acaba um determinado recurso ou conquista o objetivo e na função Update ter um if que sempre esteja verificando as condições

Cenas

- Cenas de menu podem ser criadas com sprites de BG e botões feitos a partir de colisores, ou ainda utilizar as funções de interfaces quer as versões mais novas do unity já possuem
- Na lista de cenas no build é possível já escolher para qual tipo de plataforma irá ser feito. Levando sempre em consideração que os controles devem ser observados pois nos dispositivos moveis não possuem teclado e devem ser feitos as adaptações, como colocar botões na tela.

Sons

- Os áudios podem ser adicionados aos objetos e possuem bastante propriedades, basta um drag e drop dos assets para o objeto
- Ter cuidado pois no projeto possui um lugar que escuta os áudios e se o emissor de áudio estiver distante não ouvirá bem o som



Sons

- As trilhas sonoras geralmente são colocados na própria câmera do projeto pois é nela que possui o listener (escutador) e são colocados para tocar logo quando criados.
- Já o de efeitos sonoros devem ser tocados quando algo acontece, geralmente uma colisão. Ex:

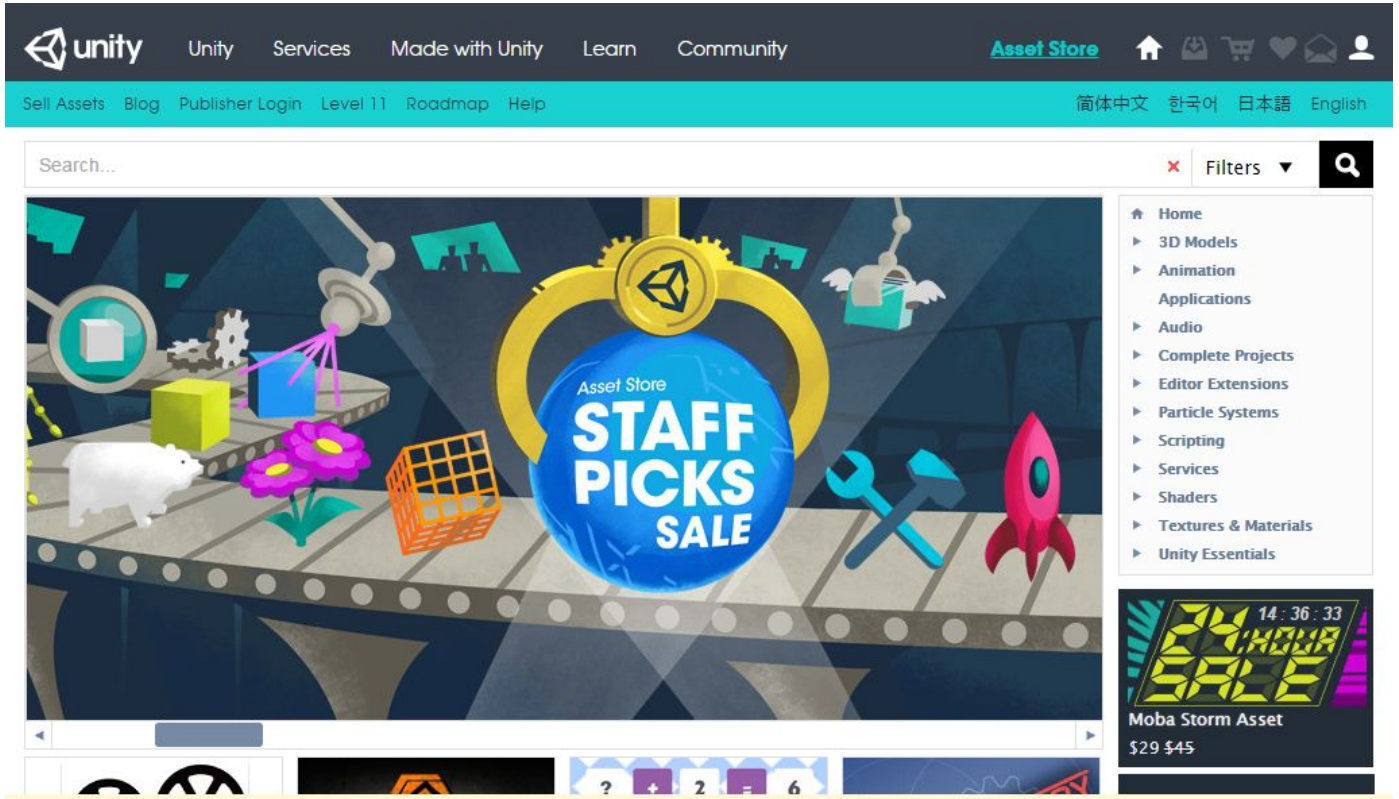
```
public class ExampleClass : MonoBehaviour {
    AudioSource audio;

    void Start() {
    audio = GetComponent<AudioSource>();
    }

   void OnCollisionEnter(Collision collision) {
      if (collision.name==“bomba”)
         audio.Play();

   }
}
```

Armazenamento

- O Unity possui funções que armazenam informações (int, float e strings) para uma posterior recuperação de dados
- Serve para criar ranking, score, save e outros
- Para salvar = PlayerPrefs.SetString("Player Name", “Jesse");
- Para recuperar = PlayerPrefs.GetString("Player Name");

Loja da Unity

- Para comercializar assets, scripts e projetos completos, acessa pela web ou pelo unity:



Protótipo de Jogo 3D

- Veremos na prática todo o material visto, de modo sistemático, a criação de um cenário 3D e um jogador em primeira pessoa com as ferramentas do unity e a inserção de scripts prontos

Projeto 2D

- Reunir todo o matéria de arte, roteiro, áudio e game design para a elaboração do jogo em 2D
- Preferencialmente plataforma para que todos acompanhem o desenvolvimento juntamente com a aula.
- Equipes de 2 ou 3 pessoas ou como já vem desenvolvendo nas outras disciplinas;

Referencias

- Documentação Unity. http://docs.unity3d.com/
- Making animations with Unity 2D. Disponível em :http://pixelnest.io/tutorials/2d-game-unity/animations-1/