Universidade Federal de Pernambuco
Graduação em Ciência da Computação
Centro de Informática

Metáforas de interação: desenvolvimento de metáforas para interfaces digitais baseadas nos modelos de cognição corpórea

Trabalho de Graduação

Marcelo Ferrão Feodrippe

Recife, 2016

Metáforas de interação: desenvolvimento de metáforas para interfaces digitais baseadas nos modelos de cognição corpórea

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado de Ciências da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do Título de Bacharel em Ciências da Computação.

Orientador: Pedro Martins Alessio

Marcelo Ferrão Feodrippe

Metáforas de interação: desenvolvimento de metáforas para interfaces digitais baseadas nos modelos de cognição corpórea

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado de Ciências da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do Título de Bacharel em Ciências da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 21, de julho de 2016.

Banca examinadora:

Prof. Dr. Pedro Martins Alessio
Orientador
UFPE

Prof. Dr. Ruy J. Guerra B. de Queiroz
Avaliador
UFPE

Dedico este trabalho aos meus pais, primeiros companheiros que tive nesta vida. Aos amigos que me apoiaram, principalmente meus irmãos.

Agradeço a Deus por tudo.

Agradecimentos

Primeiramente aos meus pais e irmãos. A minha família que me apoiou. Ao Prof. Dr. Pedro Martins Alessio por ajudar a desenvolver o tema com muito mais precisão. Ao professor: Geber Ramalho por abrir a porta, indicando o professor Pedro Alessio. Abrindo assim uma grande oportunidade. A todos os professores que me ajudaram nesta carreira, não só de universidade mas de minha vida educacional toda. A todos os colegas e amigos de curso que me apoiaram.

“Eu prometo viver até morrer” - **Homer J. Simpson** (Sim, os Simpsons)

Resumo

Metáforas estão presentes em nosso dia a dia, em vários aspectos de nossa rotina de linguagem e comunicação. Exemplos disto são termos como a "quebrar as regras" para falar de desrespeitar convenções ainda que regras não são algo físico que se possa quebrar. Em outro exemplo usamos gestos como o de levantar o dedo polegar para expressar um sentimento de positividade, abanar a mão para indicar uma despedida e muito outros gestos e para expressar sentimentos e conceitos abstratos. Assim Metáfora consiste na definição de algo abstrato a partir de outra coisa concreta ou um conceito mais familiar, que pode ou não ser de mesma natureza. Este trabalho tem um intuito de analisar e desenvolver metáforas de interface baseados no modelo cognitivo dito corpóreo que se baseia na experiência sensorial e corpórea como metáforas para traduzir conceitos abstratos.

Este trabalho tem o intuito de mostrar por meio de aplicações os conceitos no campo das metáforas de cognição corpórea desenvolvidos pelo Dr. Prof. Pedro Martins Alessio. Escolhemos três exemplos de metáforas oriundas de sua tese de doutorado [1] para a construções de interações em interfaces gráficas. As metáforas de interface serão implementadas utilizando a ferramenta Unity e suas bibliotecas físicas e de comportamento. Os resultados do desenvolvimento serão apresentados a usuários que responderão a um questionário de usabilidade. Serão descritos os resultados e trabalhos futuros.

Abstract

Metaphors are present in our daily lives in various aspects of our language and communication. Examples like "brake the law" to talk of disrespeting rules, although a law is not something physical that you can break. In another example the use of gestures such as lifting the thumb to express a sense of positivity, wave his hand to indicate a farewell and other gestures used metaphorically to express feelings and abstract tought. A metaphor is the traduction of an abstraction in terms of a concrete thing or a more familiar concept, which may or may not be of the same nature. This work has the purpose to analyze and develop interface metaphors based on the embodied cognitive model that reveal how sensory and corporeal experience are used by our minds as metaphors to translate abstract concepts. This work aims to show through applications concepts in the field of body cognition metaphors developed by Prof. Dr. Pedro Martins Alessio. We have chosen to implement three examples of metaphors derived from his doctoral thesis [1] as interactions techniques in Graphic User Interfaces. The Interface metaphors will be implemented using the 3d engine Unity3D and its physical and behavior libraries. The development results will be presented to users and a usability questionnaire will be proposed. The results and future work will be described.

Sumário

Introdução

A relação com o usuário vem se revelando um dos aspectos mais importantes do avanço da tecnologia. E desde a revolução iniciada pela inovação do Xerox Star e dos computadores Apple, as metáforas de interação tem um papel central nesta busca por interfaces que possam revelar usos e ações para o usuário. Portanto neste projeto estamos interessados na interação do usuário com a tecnologia, precisamente referente às metáforas de interface. Surge a necessidade de estudar mais a fundo como deve ser realizada a relação de usuário com a máquina. Neste contexto que surge a metáfora de interface, que tende a melhorar a comunicação com estas tecnologias mas que é preciso fazê-la evoluir dos paradigmas criados nos anos 80 (desktop metaphor) e para isso é necessário adentrar brevemente nos estudos de nosso sistema cognitivo de onde nos baseamos na tese do Prof. Pedro Aléssio.[1]

As reflexões a respeito da natureza da cognição vem sendo enriquecidas com o estudo da relação de nossa mente e sua condição corpórea, ou seja, um cérebro com uma relação intrínseca com o corpo que o contem. Em vez de enfatizar operações formais em símbolos abstratos, ou tratar o cérebro apenas como um computador que trata *inputs* sensoriais, esta abordagem focaliza no fato que nossa forma de pensar o mundo real são baseadas na forma como interagimos com o mundo externo. Isto coloca em foco, assim, o fato de que a cognição é uma atividade corpórea ou situada sugerindo que os seres pensantes devem, portanto, ser considerado em primeiro lugar como seres de ação.[4] Desta forma foi apresentado por Lakoff & Johnson [2] o paradigma que descreve nossa forma de pensar como metáforas de nossa ação e interação com o mundo real e externo à mente.

Um dos estudos apresentados por Lakoff & Johnson classifica estas metáforas de ações de onde nos inspiramos e selecionamos 3 metáforas distintas: fluidos, fricção e forma. Apresentaremos estes três tipos de metáforas da cognição dita corpórea em forma de interfaces interativas, ou seja, foram desenvolvidos 3 aplicações com auxilio da ferramenta Unity, cada uma de acordo com a classificação de Lakoff & Johnson como apresentados na tese de doutorado do Prof. Dr. Pedro Alessio.

Metáforas Corpóreas

Metáfora

Definida de diversas formas, poderemos usá-la para fins práticos deste trabalho como um meio de explicar um conceito “A” por meio de um conceito “B”. É basicamente empregar um conceito que pode ser totalmente diferente, em meio semântico, porém que explica de forma intuitiva um conceito abstrato. Uma analogia de fato para com algo. Utilizar um meio concreto para exprimir uma noção abstrata.

“Evocar uma imagem para clarear uma situação” M. Maffesoli

A frase acima de Maffesoli explica exatamente um uso de metáfora em uma frase quando se refere a clarear uma situação por exemplo, utilizando a palavra clarear (jogar luz sobre) para definir um meio de “entender” a situação, neste contexto. Outro exemplo de termos usados para facilitar a compreensão de processos abstratos seria de nos referirmos à internet como uma teia.

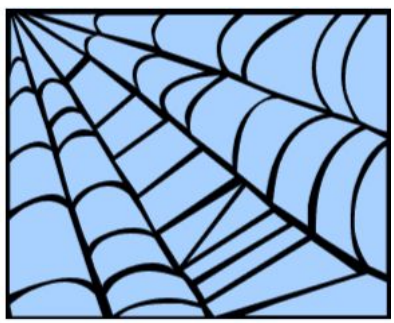

Figura 1 - Internet como uma teia

No campo da interação homem máquina as metáforas vem sendo utilizadas desde os anos 80 com o advento das interfaces gráficas para substituir as linhas de comando. A relação entre a compreensão e ação, e o desenvolvimento de modelos cognitivos voltados para a informática vem sendo construídas há mais de trinta anos e a metáfora do desktop foi

popularizado pela Xerox, Apple e Microsoft, que têm espalhado o uso de computadores, facilitando a criação e organização de nossas atividades eletrônicas até hoje. Estas ferramentas são fáceis de entender, graças aos conjuntos de gráficos dispostos em uma lógica visual inspirada em atividades de escritório. Esta solução cognitiva foi criada na década de 80 e inspirada pelo trabalho em escritórios de grandes empresas da época [3]. Além das analogias a objetos como lixeiras e pastas, a metáfora do desktop foi enriquecida com uma série de ferramentas abstratas, como menus suspensos, janelas, arquivos e técnicas de interação, como a manipulação direta, a barra de rolagem ou Drag & Drop. Este conjunto de soluções é parte do que é chamado o paradigma sigla WIMP das palavras do Windows, ícones, menus e ponteiros. A base deste paradigma é ainda a tradução de tarefas complexas e abstratas do universo digitais, tais como a criação de um novo documento ou coleção de arquivos em uma série de metáforas - ou seja analogias visuais ao mundo físico - como ícone de uma cesta ou um disquete. Atualmente vemos que tais metáforas estão longe de ser a representação ideal para as tarefas computacionais contemporâneas. A internet das coisas, as formas de visualização da informação entre outras formas de lidar com informação não podem apenas ser representadas como um ambiente de trabalho dos anos 80. É necessário pensar novas maneiras de traduzir conceitos e sistemas complexos com metáforas.

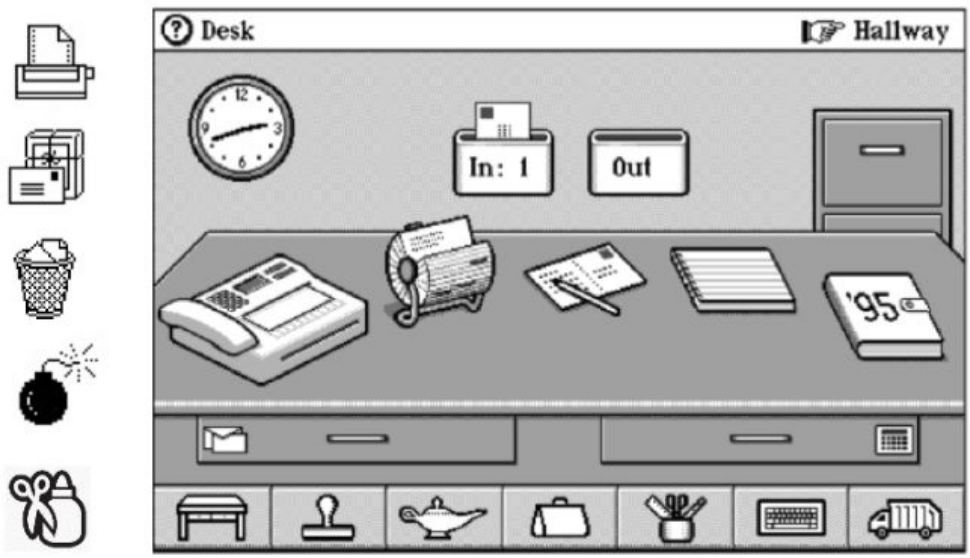

Figura 2 - Desktop de um computador

A **Figura 2** representa um típico exemplo de metáfora que muito de nós lidamos no dia a dia. Simplesmente imagens são utilizadas para nos ajudar na interação com a máquina, como por exemplo podemos ver a lixeira no qual arquivos deletados estarão ou o símbolo da impressora que representa que ao selecioná-la a ativará.

Metáforas Corpóreas

Agora sim estamos chegando próximo do conceito que queremos apresentar. As metáforas corpóreas são portanto uma pista para explorar novas formas de se construir traduções para universos complexos e abstratos. Segundo Lakoff & Johson [2] a metáfora não é apenas um refinamento linguístico, mas um mecanismo cognitivo. Tem uma frase bastante interessante de Paul Riccoeur: *"Se uma metáfora consiste em falar de uma coisa em termos de outra, não consistiria também em perceber, pensar ou sentir, sobre uma coisa em termos de outra coisa?"*

Nosso sistema cognitivo é voltado para o meio que vivemos, é o que compreendemos. A teoria da cognição corpórea sugere que nossos componentes corporais e nosso sistema perceptivo ou motor são partes de nosso sistema de compreensão. É bastante importante os sentidos como meios de de compreensão e não apenas como *inputs* para que o sistema cognitivo trate a informação recebida. Nosso sistema de compreensão não é isolado estando assim enraizado na experiência física e humana social.

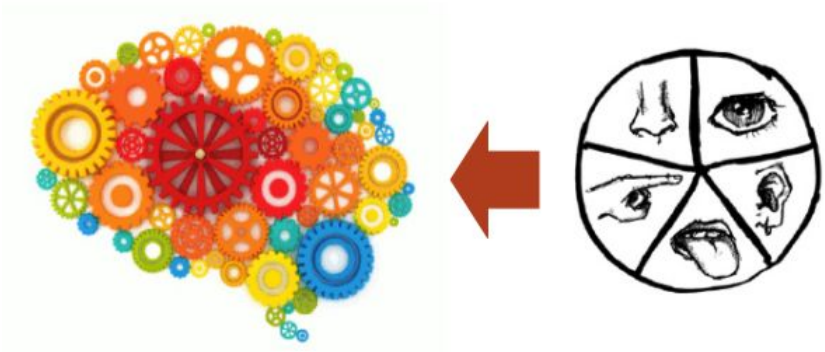

Figura 3 - Sistema de compreensão baseada na corporeidade faz do sentido parte do sistema cognitivo e não apenas porta de entrada da informação.

Desta maneira surgem as metáforas da corporeidade que permeiam nosso discurso e que na verdade são parte de nossa forma de pensar. Quando comentamos que “a bolsa caiu”, estamos nos referindo a um sistema financeiro abstrato mas que ao associar um momento ruim desse sistema ao ato de “cair” estamos nos referindo à uma experiencia concreta de nossa corporeidade, quando o fato de uma pessoa cair de um lugar alto é ruim, ser aplicado ao conceito de a bolsa cair, ou estar abaixo do esperado. E assim é construído um vocabulário coerente à partir dessa metáfora onde termos como despencar ou quebrar passa a fazer sentido. Vários fatores podem estar englobados neste ao fazer uma analogia do mesmo tipo. Outros exemplos estão demonstrados abaixo no qual a descrição de cada figura representa um conceito ou da figura desenvolver um conceito:

Figura 4 - “O clima está pesado”

Figura 5- “Justiça, sentido de lado ou de peso”

Figura 6 - “Agora ficou tudo mais claro”

Nosso modo de compreender os conceitos abstratos são influenciados ou até mesmo determinados pelos nossos sentidos e com nossa experiencia com objetos como podemos ver na **Figura 7** abaixo:

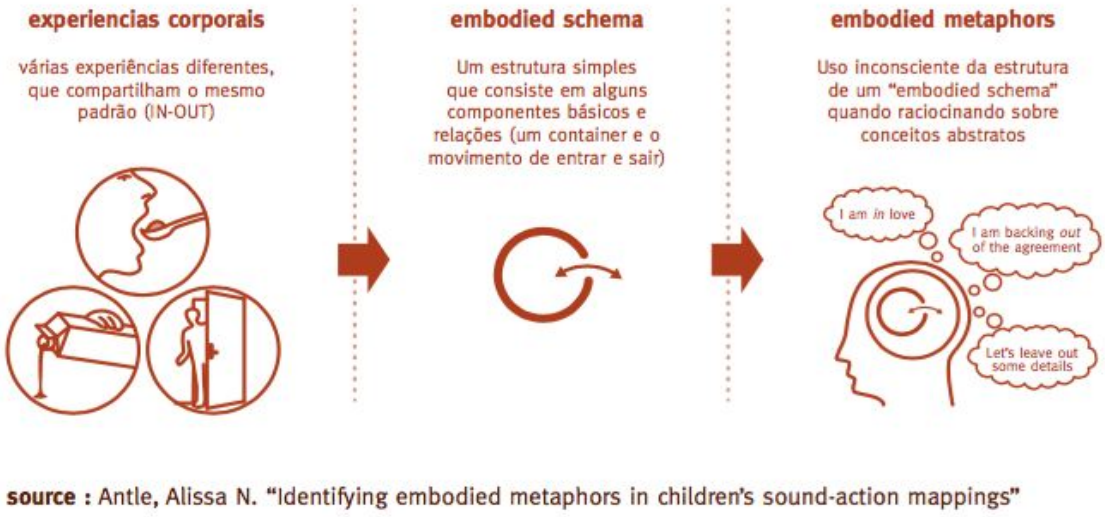

Figura 7

Classificação

As metáforas corpóreas podem ser divididas em 3 tipos, estruturais, orientacionais e ontológicas. Estamos aqui interessados nos 2 últimos tipos pois abordam os conceitos mais cognitivos completos.

Metáforas Orientacionais

Quando utilizamos nosso meio de orientação para fazer analogia a algo. “Para cima é bom” e “Para baixo é ruim”. Podemos indicar a positividade como algo com e acima e a negatividade sendo relacionadas a coisas abaixo. Tomando assim a orientação como meio de explicar algo, exemplo disto é um o “a bolsa caiu” como já mencionado. “O futuro em frente”; “O passado atrás”.

Figura 8 - Classificação por Lakoff & Johsson

A **Figura 8** mostra a classificação feita por Lakoff & Johsson[2] a respeito da metáfora orientacional. Quando fala-se em corpo, tem-se a ideia de o que é frente, atrás, longe, próximo. Indicando assim uma relação com o corpo e o ambiente, e a maneira a qual podemos fazer analogia a algo (“o mês está próximo”, "vamos deixar isso para trás"). A física é utilizada tipicamente como meio de analogia (“o clima está pesado”; "Uma pessoa leve"). E claro, não podemos deixar de fora o sócio cultural, pois estamos sempre interagindo com pessoas. A maneira que podemos indicar que estamos tristes, alegres ou qualquer sentimento pode ser expressado em termos concretos ligados a experiencias socio culturais (Casamentos, Funerais, Carnaval).

Metáforas Ontológicas

As metáforas Ontológicas se referem à nossa experiência com objetos e entidades externas. Nossa rotina de interação com fluidos, objetos, quantidades etc. Esta classificação que fazemos ao dizer que “está cheio” ou “está vazio”. Quando utilizamos a metáfora de forma intensificadora para nos referir a coisas de nosso cotidiano.

Figura 9 - Classificação das metáforas ontológicas por Lakoff & Joshson

A **Figura 9** demostra a classificação feita pelos mesmo com respeito as metáforas ontológicas. Então substancias, entidades e metonímias podem ser utilizados na representação. Utilizando fluidos, forma, consistência para representar por exemplo a filtragem de um som no qual vozes ficarias na superfície e instrumentos ao fundo. Um objeto frágil pode indicar o deadline de um projeto. Objetos em um meio podem indicar os e-mails que estão ali visivelmente.

Então é possível ter uma combinação de todos os conceitos orientacionais e ontológicos e aplicar em nosso mundo, seja ele por meio de tecnologias ou não. Tendo isto em mente, surgiu a ideia de criar aplicações voltadas nestes conceitos, no qual a cognição seria fundamental para o entendimento do aplicativo.

Pedro Aléssio em sua tese de doutorado, baseando-se nas classificações feitas por Lakoff & Joshson[2], organiza estas ideias e aborda uma subclassificação que podem ser desenvolvidas, como podemos ver na **Figura 10** abaixo.

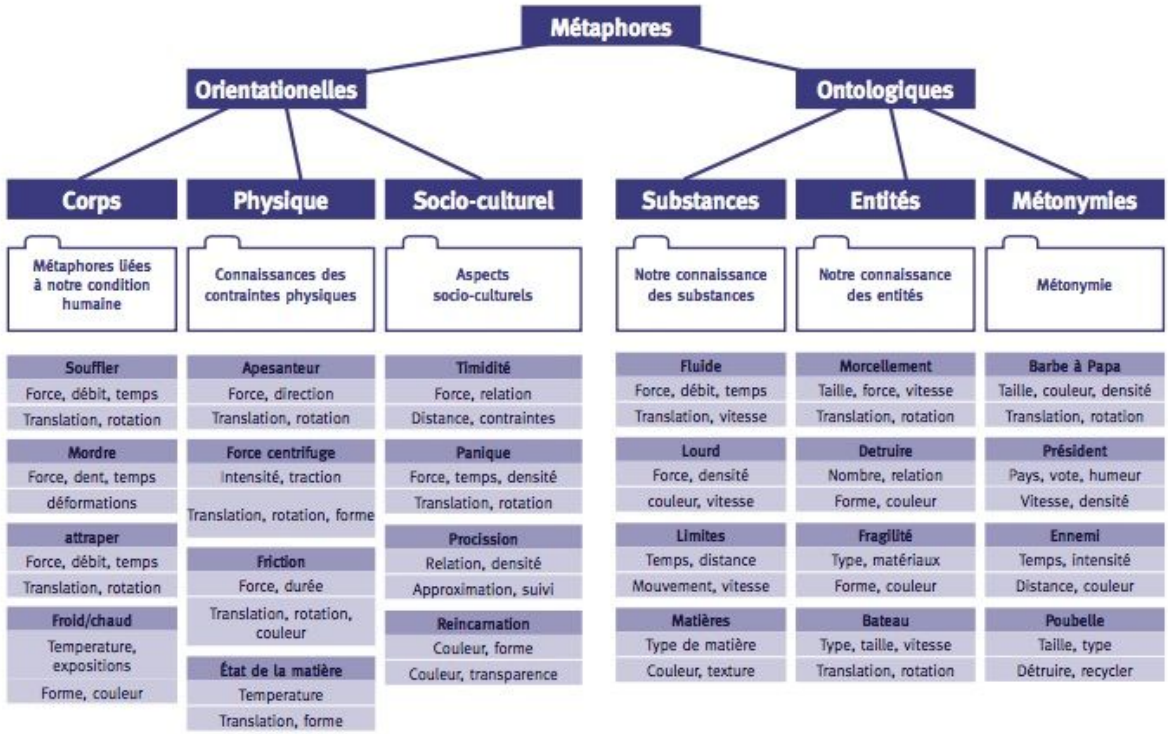

Figura 10 - Conjunto de metáforas selecionadas durante o projeto do Prof. Pedro Aléssio [1] dentro da classificação de Lakoff & Johnson

Dentre as metáforas corpóreas explicitadas selecionamos 3 metáforas para implementar sob forma de técnica de interação em UNITY: Fricção, Fluido e Forma. Iremos mostrar seus funcionamentos em termos de uma interface gráfica em 3D capaz de interagir de forma fácil com o usuário.

Fluido: Uma metáfora do tipo Ontológica, ou seja de nossa experiência com coisas e objetos. Esta metáfora pode ser de objetos que se comportam dentro de fluidos e as características desse fluido (viscosidade, densidade etc..)

Fricção: Uma metáfora do tipo Orientacional onde o que nos interessa é o comportamento dos objetos entre si. Essa metáfora pode ser observada em interfaces já existentes como nas rolagens de listas do IPhone por exemplo quando o usuário faz um gesto para cima e a lista avança e para sozinha como se por força de um atrito do papel da lista com o telefone.

Forma: De carácter Ontológico. A forma dos objetos nos diz bastante sobre ele. Se está quebrado, vazio, cheio de ar. Queríamos explorar que tipo ação ou informação o usuário pode extrair apenas observando a forma que o objeto está representado.

Unity

Aqui apresentamos um pouco sobre a ferramente utilizada para o desenvolvimento das aplicações. Utilizamos a versão do Unity 5.3.4f1 juntamente com o sistema operacional Max OS X El Capitan (Versão 10.11.4) rodando em um Mac mini (Late 2014) com processador 1.4 GHz Intel Core i5; memória 4GB 1600 MHz DDR3; placa gráfica Intel HD Graphics 5000 1536MB.

O Unity é um chamado Game Engine. Uma ferramenta de produção de jogos em 3d que evoluiu em um contexto de estimular novos usuários com uma interface amigável e fácil de implementar. É uma Ferramenta multiplataforma extremamente poderosa, desenvolvida pela Unity Technologies[5] que permite o desenvolvimento de diversas aplicações incluindo jogos, animações, em geral produtos finais que utilizam o gráfico como meio de interação ou entretenimento para com usuário final.

Bastante interessante o fato de ser multiplataforma pois pode-se criar aplicações para diversos tipos de plataformas, Windows, Mac, Xbox e muitos outros, variando de dispositivos móveis e mais complexos. Neste link (http://unity3d.com/pt/unity/multiplatform), caso estejam interessados em aprofundar mais a respeito está disponível informações, sobre quais plataformas ele fornece suporte.

A plataforma Unity possui diversas ferramentas para se trabalhar com a simulação de física, do mundo real, além de diversas APIs para o auxílio na construção da aplicação.

Funcionamento do Unity

Ao iniciar um novo projeto é possível escolher entre ambientes configurados para 3D ou 2D assim como quais pacotes o projeto poderá iniciar. Nomeamos um projeto, escolhemos seu destino e pronto, podemos começar a trabalhar.

O ambiente do Unity é descrito como o exemplo da **Figura 16** abaixo. No qual possuímos o ambiente do projeto como um todo. Na parte da Hierarchy ficam todos os os *Games Objects*(todo tipo de objeto que faz parte do ambiente, seja visual ou não) da *Scene* (o a tela mostrada onde o ambiente é desenvolvido em si). Podemos ver que nesta *Hierachy* temos a câmera principal que é a qual mostra, dependendo de sua posição, como os objetos são vistos, além de outros objetos na tela como o recipiente por exemplo contendo a água.

Mais acima podemos ver o simbolo de *Play Stop* e *Pause*, no qual o programa é testado em tempo de desenvolvimento quando requerido pelo programador, facilitando assim seus testes com relação projeto corrente. Em seguida temos as seções das pastas onde ficam os arquivos salvos pelos programas possibilitando o desenvolvedor a adicionar novas pastas, arquivos ao projeto em questão. Pode-se trabalhar com qualquer tipo de arquivo e adiciona-los a seção de *Project*, como vemos abaixo. Dentre estes arquivos existem arquivos que são padrões do próprio Unity, como os de extensões, .mat, . prefab, etc. Que variam de acordo com a necessidade do que está sendo desenvolvido. Para se aprofundar mais acessar o site da Unity para saber outros tipos de extensões. Aqui basicamente utilizamos .mat e .prefab que são Materials e Prefabs, respectivamente.[5]

Materials são utilizados para definir a textura de um objeto, basicamente sua cor e suas propriedades físicas, de reflexão, tudo que é referente a luz. São definições de como uma superfície deve ser redenrizada, incluindo referencias para texturas utilizadas, informação sobre o tilling, cores de tintas e outros mais. As opções disponíveis para um material depende de qual *Shader*(pequenos scripts que contem calculos matemáticos e algorítimos para o calculo da cor de cada pixel redenrizado, baseado na entrada de luz e da configuração do Material) o material está usando.[7]

Prefabs são convenientes para construir Games *Objects* na *Scene* nos quais pode-se criara vários objetos do mesmo tipo que possuem propriedades semelhantes no geral, assim mudando certas propriedades de um *Prefab*, todos os objetos da tela que foram adicionados com o *Prefab*, sofrem alterações em suas propriedades, sem a necessidade de alterar um por um.[8]

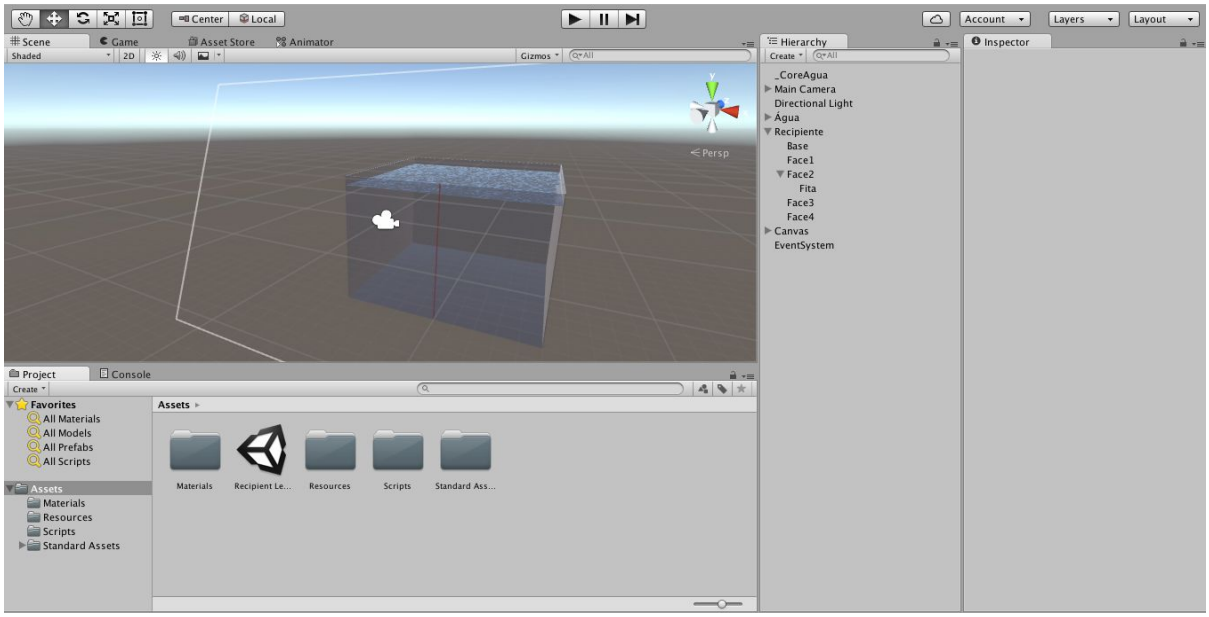

Figura 16 - Ambiente do Unity 3D

É possível trabalhar com scripts nas linguagens em **C#** ou **JavaScript**, desde que a API é a mesma independente da linguagens utilizada.[3] Para estre trabalho utilizamos scripts em C#. Script é a parte onde o programador realmente trabalha de maneira a produzir as interações entre a cena e o usuário, onde toda ação necessária para o funcionamento da aplicação é realizada. Resumindo é onde a codificação é feita utilizando um dos seguintes tipos de scripts como mencionado anteriormente.[6]

Metáforas Desenvolvidas

Apresentaremos aqui as metáforas implementadas sob forma de interfaces interativas em Unity Fluidos, Fricção, Forma. Os aplicativos foram desenvolvidos de maneira que seja possível desenvolver a aplicação passo a passo. É necessário conhecer a ferramenta **Unity 3D** para entender o que foi elaborado, não focaremos em explicar linha por linha do código, porém o meio necessário para sua reprodução.

Fluidos (densidade)

Pensemos na ideia de flutuação e comportamento de objetos em fluídos. Imaginemos um aquário cúbico preenchido com água, onde objetos que representam e-mails estejam neste recipiente. No qual a ideia de que o e-mail mais acima, é o mais recente chegado. Ou seja relacionar o conceito de densidade de materiais com o parâmetro de tempo ou data. Fixados as dimensões do cubo, a medida que vão chegando recados novos, a tendência é que e-mail mais antigos vão ficando mais densos, logo vão afundando, representando o quais estão mais ao fundo, como tendência de tempo prolongado existente.
Baseada nesta ideia desenvolvemos uma protótipo que simula um cubo com fluído dentro, no qual objetos mais recentes tendem a ficar mais a superfície (menos densos) e objetos que estão a mais tempo, mais próximos ao fundo(mais densos), exemplo disto na **Figura 11**, o qual é uma imagem do protótipo desenvolvido.

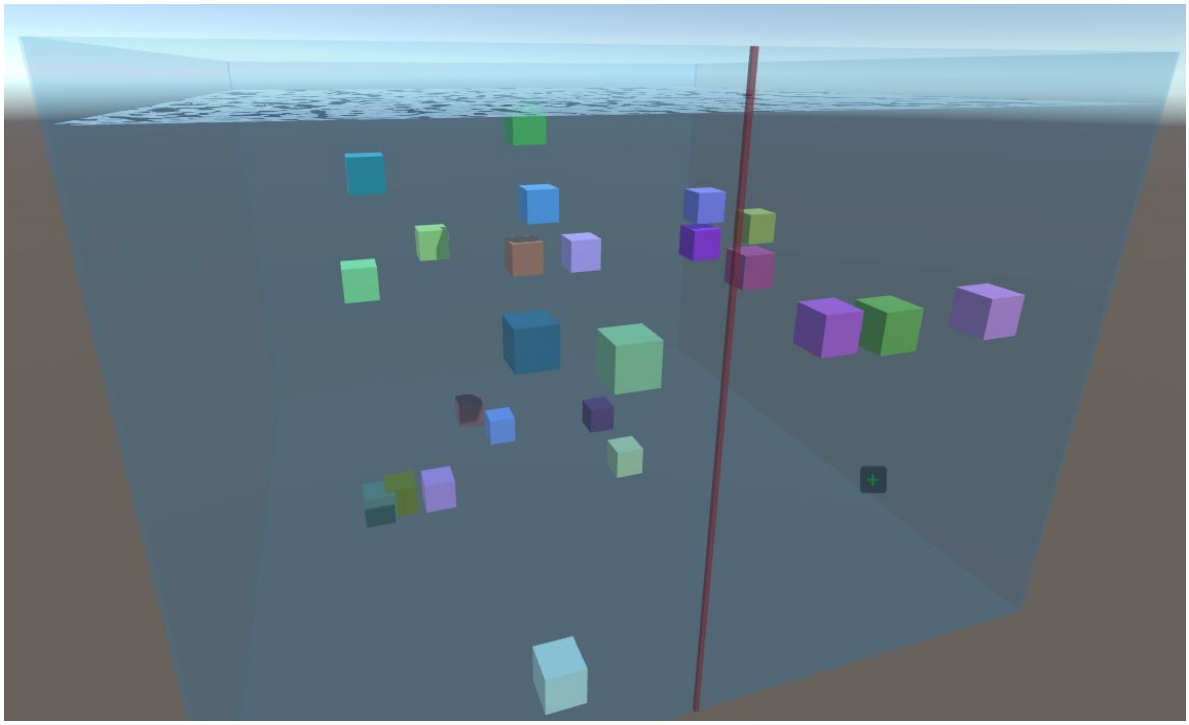

Figura 11 - Protótipo baseado no modelo de cognição corpóreo voltado aos Fluidos

A além da metáfora utilizada na flutuação de fluidos, podemos também utilizar a noção do que está próximo a você é algo importante. Então é flexível ao usuário mover o cubo, para direita, esquerda, frente, trás.
Aqui foi utilizada a própria API da unity com respeito a física 3d, no qual o objeto fica mais denso a medida que outros objetos chegam de forma proporcional em espaço e tempo. A seguir mostraremos como a aplicação foi desenvolvida.

Implemetação em Unity 3D

Na raiz, dentro de Assets, foram criadas as seguintes pastas: Materials, Resources e Scripts. Existe uma outra pasta chamada Standard Asset a qual foi iniciada ao importar packages referentes a Environment, no qual selecionamos Water, que é esta camada acima que representa a água como vemos na **Figura 11**. Na Hierarchy tem basicamente o Event System, a camera principal, o recipiente, formados por cubos, e a água. Também temos um Objeto _CoreAgua que serve para utilizar os scripts CoreWaterCollision.cs, ClickMoveObject.cs, ButtonAction.cs.

Na pasta Materials existem dois arquivos chamados ColorRedVariated.mat e RecipientColor.mat, que representam a cor vermelha da linha na frente do recipiente e a cor do recipiente em si, a transparente, então cabe a quem vai utilizar ajustar para como quiser sua cor.

Em Resources temos Arquivo.prefab, que são esses cubos flutuantes que vimos na imagem, que é basicamente um cubo com o script Floating.cs adicionado como componente.

Por fim nossa pasta Script onde usaremos para guardar os nossos scripts que serão mostrados abaixo:

Scripts

Script - ButtonAction.cs

```
using UnityEngine;
using System.Collections;

public class ButtonAction: MonoBehaviour {
public void NovoArquivo () {
Instantiate (Resources.Load ("Arquivo"));
}
}
```

Este script simplesmente pega um prefab, neste caso chamado de Arquivo, na pasta de Resources e instancia. Fazendo assim com que uma cópia sua seja criada.

Script - ClickMoveObject

```
using UnityEngine;
using System.Collections;

public class ClickMoveObject : MonoBehaviour {

private RaycastHit colisor;
private bool pressionando = false;
private Transform axisR;

void Start()
{
axisR = Camera.main.transform.GetChild(0);
}

void Update () {
if (Input.GetMouseButton (0)) {
```

```
30. var ray = Camera.main.ScreenPointToRay (Input.mousePosition);
31. if (pressionando) {
32. if (colisor.transform.gameObject.layer != 8)
33. return;
34.
35. colisor.transform.Translate(Input.GetAxis ("Mouse X"), 0, Input.GetAxis("Mouse Y") ,
    axisR);
36. }else {
37. var layerMask = (1 << 8);
38. if (Physics.Raycast (ray, out colisor, 10000, layerMask)) {
39. if (colisor.transform.gameObject.layer != 8)
40. return;
41. pressionando = true;
42. }
43. }
44. }
45.
46. if (Input.GetMouseButton (1)) {
47. Camera.main.transform.RotateAround (Vector3.zero, Vector3.up, Input.GetAxis
    ("Mouse X"));
48. }
49. }
50.
51. void LateUpdate()
52. {
53. if (pressionando) {
54. if (Input.GetMouseButtonUp(0)){
55. pressionando = false;
56. }
57.
58. if (colisor.transform.position.z > 5f - colisor.collider.bounds.size.z/2) {
59. colisor.transform.position = new Vector3 (colisor.transform.position.x,
60. colisor.transform.position.y, (5f - Time.deltaTime)  - colisor.collider.bounds.size.z/2);
61. } else if (colisor.transform.position.z < -5f + colisor.collider.bounds.size.z/2) {
62. colisor.transform.position = new Vector3 (colisor.transform.position.x,
63. colisor.transform.position.y, -(5f - Time.deltaTime) + colisor.collider.bounds.size.z/2);
64. }
65.
66. if (colisor.transform.position.x > 5f - colisor.collider.bounds.size.x/2) {
67. colisor.transform.position = new Vector3 ((5f - Time.deltaTime) -
    colisor.collider.bounds.size.x/2,
68. colisor.transform.position.y, colisor.transform.position.z);
69. } else if (colisor.transform.position.x < -5f + colisor.collider.bounds.size.x/2) {
70. colisor.transform.position = new Vector3 (-(5f - Time.deltaTime) +
    colisor.collider.bounds.size.x/2,
71. colisor.transform.position.y, colisor.transform.position.z);
72. }
73. }
74. }
```

```
75. }
```

Este script serve para mover o objeto selecionado em relação a posição da câmera e delimitar seus limites.

Script - CoreWaterCollision.cs

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using System.Linq;
5.  using System.Linq.Expressions;
6.  public class CoreWaterCollision : MonoBehaviour {
7.
8.  public static float alturaAgua = 6.5f;
9.  public static float alturaMin = 0.5f;
10.
11. private static IList<Floating> listaArquivos;
12.
13. void Start () {
14. listaArquivos = new List<Floating> ();
15.
16. }
17.
18. public static void AddArquivo(Floating arquivo)
19. {
20. if (listaArquivos.Contains (arquivo))
21. return;
22. listaArquivos.Add (arquivo);
23.
24. listaArquivos = listaArquivos.OrderBy (a => a.data).ToList();
25. AtualizaPesos();
26. }
27.
28. private static float Conversao(float max, float min, long u, long p, long x)
29. {
30. var resultado = ((x-u)*(min-max)/(p-u)) + max;
31. return resultado;
32. }
33.
34. private static void AtualizaPesos()
35. {
36. var primeiro = listaArquivos.ElementAt (0);
37. var ultimo = listaArquivos.LastOrDefault ();
38. if (primeiro == ultimo) {
39. primeiro.SetAlturaOscilacao (alturaAgua);
```

```
40. return;
41. }
42.
43. foreach(var elem in listaArquivos)
44. {
45. elem.SetAlturaOscilacao(Conversao (alturaAgua, alturaMin, ultimo.data,
    primeiro.data, elem.data));
46. }
47. }
48.
49. }
```

Este script é bastante importante para os cálculos das densidades dos objetos. Basicamente ele guarda uma lista de objetos organizada pela data atribuída, atualizado as novas posições dos objetos a medida que novos vão chegando.

A posição dos objetos é feita de maneira simples no qual fixamos a altura da água e o que muda são as alturas dos objetos, simulando a densidade alterada. Quem aqui não lembra daquela velha aula de termodinâmica na conversão de escalas de temperaturas? (Kelvin para grau Celsius por exemplo). A mesma ideia é aplicada aqui assim nos cálculos na função de Conversão [private static float Conversao(float max, float min, long u, long p, long x)], simplesmente tomamos como parâmetros os limites da água, que é fixo, as datas dos extremos, e então a partir da data em questão a ser atualizada, temos a altura como resultado, representando assim a densidade em questão.

Script - Floating.cs

```
1. using UnityEngine;
2. using System.Collections;
3.
4. public class Floating : MonoBehaviour {
5.
6. public long data = 0;
7. public float multiplicador = 0.5f;
8. public float fatorOscilaçao= 1.0f;
9.
10. private float alturaOscilaçao;
11. private Rigidbody corpo;
12. private GameObject aguaCena;
13. private Vector3 forçaEquilibrio;
14.
15. private float distancia = 0.0f;
16.
17. void Awake()
18. {
19. Color colR = new Color(Random.Range(0.0f, 1.0f), Random.Range(0.0f, 1.0f),
    Random.Range(0.0f, 1.0f), 1.0f);
```

```
20. gameObject.GetComponent<Renderer> ().material.color = colR;
21. var posxR = Random.Range (-4.5f, 4.5f);
22. var posxZCorrecao = Random.Range (-0.1f, 0.1f);
23. gameObject.transform.position = new Vector3 (posxR, 10, posxZCorrecao);
24. data = System.DateTime.Now.Ticks;
25. }
26. void Start () {
27. corpo = GetComponent<Rigidbody> ();
28. }
29.
30. void Update () {
31. if (aguaCena != null) {
32. if (transform.position.y <= alturaOscilaçao) {
33. distancia = transform.position.y - alturaOscilaçao;
34. forçaEquilibrio.y = (Physics.gravity.y * distancia - corpo.velocity.y) * multiplicador;
35. corpo.AddForce (forçaEquilibrio);
36. }
37. }
38. }
39.
40. public void SetAlturaOscilacao (float fator)
41. {
42. this.alturaOscilaçao = fator;
43. }
44.
45. public float GetAlturaOscilaçao ()
46. {
47. return this.alturaOscilaçao;
48. }
49. void OnTriggerEnter(Collider coll)
50. {
51. if (coll.tag == "Water" && aguaCena == null) {
52. aguaCena = coll.gameObject;
53. CoreWaterCollision.AddArquivo (this);
54. }
55. }
56. }
```

Este é o script utilizado por cada objeto(cubo), que para este programa, ao iniciar (ser instanciado), tem uma cor aleartóriae data definida para atual. Quando objeto ao cair passa da altura máxima do fluido, ele começa a utilizar a força contrária ao peso a medida de controlar a flutuação em determinado nível. Basicamente é responsável por manter o cubo flutuando a certa altura devido a sua densidade.

Fricção

Nesta parte utilizaremos os conceitos de fricção. Imagine uma caixa retangular e dentro desta caixa existem botões enumerados, e ao selecionar um deles e arrastar para certa altura, aqueles os quais possuem o número com mais afinidade ai que está selecionado, ficará mais próximo a ele, e do outro lado os que tem menos afinidade, mais distantes. Afinidade aqui se diz por diferença entre um número e outro, em módulo. Se eu seleciono 15, e tiver 10 e 20, então 10 e 20 serão representados na mesma altura. Após parar de selecionar o número, os botões caem com o tempo com velocidade diferentes, no qual os que estão mais elevados demoram mais para cair, possuindo velocidade menos, ou mais fricção, neste caso, uma força contrária a gravidade a medida de diminuir sua velocidade.
A **Figura 12** mostra nosso protótipo, podemos notar claramente o porque os números mais abaixo estão estão longe, dado que o número do topo é o selecionado.

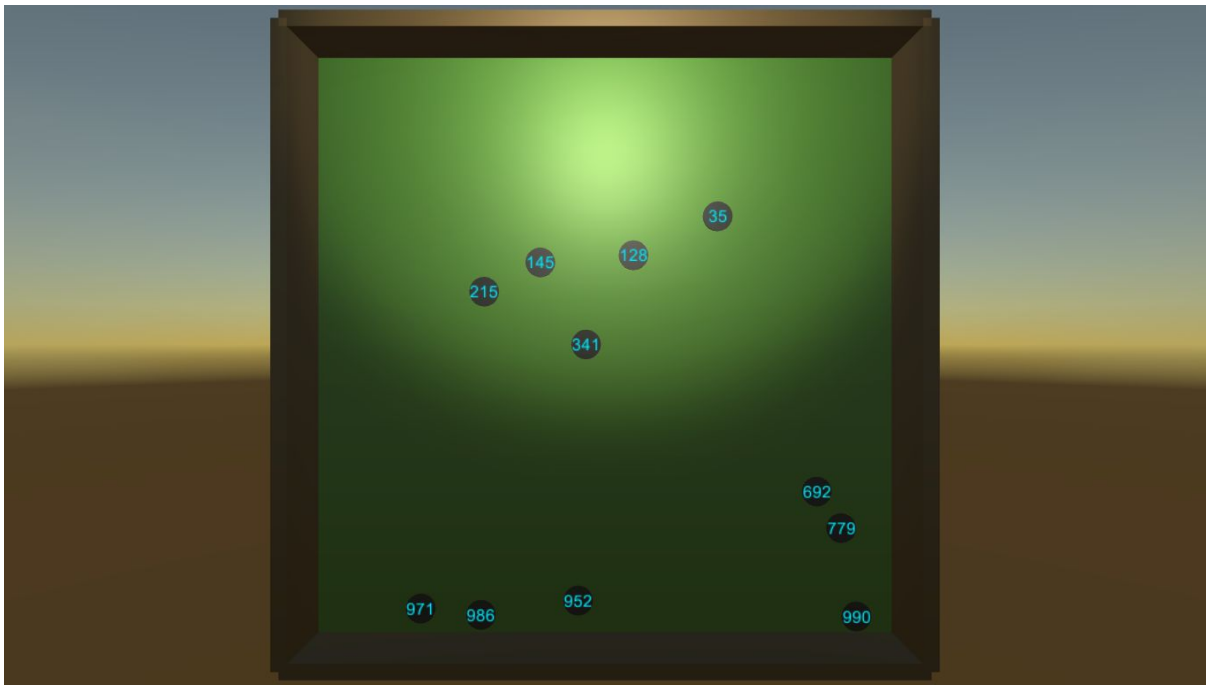

Figura 12 - Protótipo baseado no modelo de cognição corpóreo voltado à fricção

Do mesmo jeito que a respeito dos fluidos, utilizamos a própria API do Unity para simular a fricção. A seguir veremos mais a respeito de como foi criado.

Implemetação em Unity 3D

Possui a estrutura de pastas semelhante a aplicação anterior aqui mostrada, dentro de Assets: Materials, Scripts. Não foi utizado resources aqui, pois a aplicação, como protótipo, possui um numero definido de botões, o que mudam são suas enumerações. Então fica ao programador aberto a criar um prefab e ir instanciando, aqui para fim de apresentação simplesmente deixamos objetos na Hierarchy. Este por sua vez possui a câmera principal, EventSystem e a plataforma, formadas por um plano, e 4 cubos, a qual vemos na **Figura 12.** Possuí também um Objeto _Events que tem como componentes os scripts ClickToMove.cs e CoreOrganizer.cs. Cada botão na tela possui em seus componentes o script ObjetoCore.cs. Aqui utilizamos 11 objetos deste tipo.

Em Materials só guardamos três cores, Color1.mat, Color2.mat, Color3.mat que são do plano, dos cubos e do botão, que fica a rigor do programador qual será.

Em Scripts, os scripts utilizados para esta aplicação que são os seguintes:

Scripts

Script - ClickToMove.cs

```
1. using UnityEngine;
2. using System.Collections;
3.
4. public class ClickToMove : MonoBehaviour {
5.
6. public float velocidadeArrastar = 60f;
7. public float gravidade = -10f;
8. public static bool zeroVeloc = false;
9.
10. public static float limitXAxis = 4.89f;
11. public static float limitYAxis = 4.89f;
12.
13. private RaycastHit colisor;
14. private bool pressionando = false;
15.
```

```
16. void Update () {
17.
18. if (Input.GetMouseButton (0)) {
19.
20. var ray = Camera.main.ScreenPointToRay (Input.mousePosition);
21. if (pressionando) {
22.
23. if (colisor.transform.gameObject.layer != 8)
24. return;
25. colisor.transform.Translate (Input.GetAxis ("Mouse X"), 0, Input.GetAxis ("Mouse Y")
    * velocidadeArrastar * Time.deltaTime);
26.
27. } else {
28. var layerMask = (1 << 8);
29. if (Physics.Raycast (ray, out colisor, 10000, layerMask)) {
30. if (colisor.transform.gameObject.layer != 8)
31. return;
32. pressionando = true;
33. Physics.gravity = Vector3.zero;
34. zeroVeloc = true;
35. CoreOrganizer.Calcula (colisor.transform.GetComponent<ObjetoCore> ());
36. }
37. }
38. }
39. }
40.
41. void LateUpdate()
42. {
43. if (pressionando) {
44.
45. if (colisor.transform.gameObject.layer != 8)
46. return;
47.
48. if (colisor.transform.position.y > limitYAxis - colisor.collider.bounds.size.y/2) {
```

```
49. colisor.transform.position = new Vector3 (colisor.transform.position.x,
50. (limitYAxis-Time.deltaTime) - colisor.collider.bounds.size.y / 2
    ,colisor.transform.position.z);
51.
52. } else if (colisor.transform.position.y < -limitYAxis + colisor.collider.bounds.size.y/2) {
53. colisor.transform.position = new Vector3 (colisor.transform.position.x,
54. -(limitYAxis-Time.deltaTime) + colisor.collider.bounds.size.y / 2,
    colisor.transform.position.z );
55. }
56.
57. if (colisor.transform.position.x > limitXAxis - colisor.collider.bounds.size.x / 2) {
58. colisor.transform.position = new Vector3 ( (limitXAxis-Time.deltaTime) -
    colisor.collider.bounds.size.x / 2,
59. colisor.transform.position.y, colisor.transform.position.z);
60.
61. }
62. else if (colisor.transform.position.x < -limitXAxis + colisor.collider.bounds.size.x / 2) {
63. colisor.transform.position = new Vector3 ( -(limitXAxis-Time.deltaTime)
64. + colisor.collider.bounds.size.x / 2, colisor.transform.position.y,
    colisor.transform.position.z);
65. }
66.
67. CoreOrganizer.Calcula (colisor.transform.GetComponent<ObjetoCore> ());
68.
69. if (Input.GetMouseButtonUp (0)) {
70. pressionando = false;
71. zeroVeloc = false;
72. Physics.gravity = new Vector3 (0, gravidade, 0);
73. }
74. }
75. }
76. }
```

Este script é responsável por encontrar o objeto selecionado na tela e então chamar uma outra função em outro script para nivelar os objetos em relação ao selecionado. Além de definir os limites para onde pode-se arrastar. Ao selecionar um objeto a gravidade ambiente é colocada em 0 afim de deixar os objetos sem velocidade a medida que são ajustados. Após parar de selecionar a gravidade é posta para trabalhar resultando no decaimento com relação a altura dos objetos, com velocidade de decaimento menor para aqueles que estão mais acima, ou seja, os que possuem mais fricção.

Script - ObjetoCore.cs

```
1. using UnityEngine;
2. using System.Collections;
3. using UnityEngine.UI;
4.
5. public class ObjetoCore : MonoBehaviour {
6.
7. public int numero = 0;
8.
9. public void setAltura(float y)
10. {
11. GetComponent<Rigidbody> ().velocity = Vector3.zero;
12. GetComponent<Rigidbody> ().drag = (y + 3.5f)/5.0f;
13. gameObject.transform.position = new Vector3 (gameObject.transform.position.x, y,
    gameObject.transform.position.z);
14.
15. }
16. void Start () {
17.
18. CoreOrganizer.AddObjeto (this);
19. var text = GetComponentInChildren<Canvas> ().GetComponentInChildren<Text>();
20. numero = Random.Range (1, 999);
21. text.text = numero.ToString ();
22.
```

23. gameObject.transform.position = new Vector3 (Random.Range(-(ClickToMove.limitXAxis*0.95f)
    1. , ClickToMove.limitXAxis*0.95f), Random.Range(0f, ClickToMove.limitYAxis*0.95f), -0.1f);
24.
25. }
26.
27. void Update () {
28. if (ClickToMove.zeroVeloc) {
    1. GetComponent<Rigidbody> ().velocity = Vector3.zero;
29. }
30.
31. }
32. }
33. Aqui é onde cada circunferência é enumerada ao iniciar o programa. Para cada botão existe um script desse, assim servindo como seu core. Como ele é inicializado e qual o seu valor. Também utilizado para controlar a velocidade quando um objeto está selecionado.
34.
35. Script - CoreOrganizer.cs
36.
37. using UnityEngine;
38. using System.Collections;
39. using System.Collections.Generic;
40. using System.Linq;
41. using System.Linq.Expressions;
42.
43. public class CoreOrganizer : MonoBehaviour {
44.
45. private static IList<ObjetoCore> objetos;
46.
47. public void Awake()
48. {
49. objetos = new List<ObjetoCore > ();

```
50. }
51.
52. public static void AddObjeto(ObjetoCore objeto)
53. {
54. objetos.Add (objeto);
55. }
56.
57. public static void Calcula(ObjetoCore objeto)
58. {
59. if (objetos.Count () < 2)
60. return;
61.
62. objetos = objetos.OrderBy (a => Mathf.Abs(objeto.numero - a.numero)).ToList();
63. var total = objetos.Count ();
64. var posMax = objeto.transform.position.y;
65. var ultimo = objetos.LastOrDefault ();
66. var dif = Mathf.Abs (objeto.numero - ultimo.numero);
67.
68. for (int i = 0; i < total; i++) {
69. ObjetoCore obj = objetos.ElementAt(i);
70. float posY = Conversao (posMax, -(ClickToMove.limitYAxis*0.95f), 0, dif, Mathf.Abs
    (obj.numero - objeto.numero));
71. obj.setAltura (posY);
72. }
73. }
74.
75. private static float Conversao(float max, float min, long u, long p, long x)
76. {
77. var resultado = ((x-u)*(min-max)/(p-u)) + max;
78. return resultado;
79. }
80. }
```

Este aqui é bem semelhante ao CoreWaterCollision.cs referente ao fluidos, onde o mesmo tipo de conversão é utilizada para nivelar os números, desta vez ordenados pelo módulo da diferença dos de cada número com o selecionado.

Forma

A forma de um objeto influência a gente no fator de gostar dele ou não. E é isso que nos baseamos aqui. Um objeto com menos deformidades é mais aceitável do que um com mais defeitos. Vamos imaginar que eu quero construir um controle de versão de projetos. No qual posso representar cada projeto por um conjunto de esferas, e essas esferas representam as versões de tal (versão 1, versão 2, versão 3…). Com isso a versão 3 possui menos deformidades do que a versão 1, por estar mais trabalhada. As **Figuras 13**, **14** e **15** mostram melhor isto, cada figura foi retirada de um frame em andamento, de nosso protótipo desenvolvido, as esferas possuem animações em suas deformações.

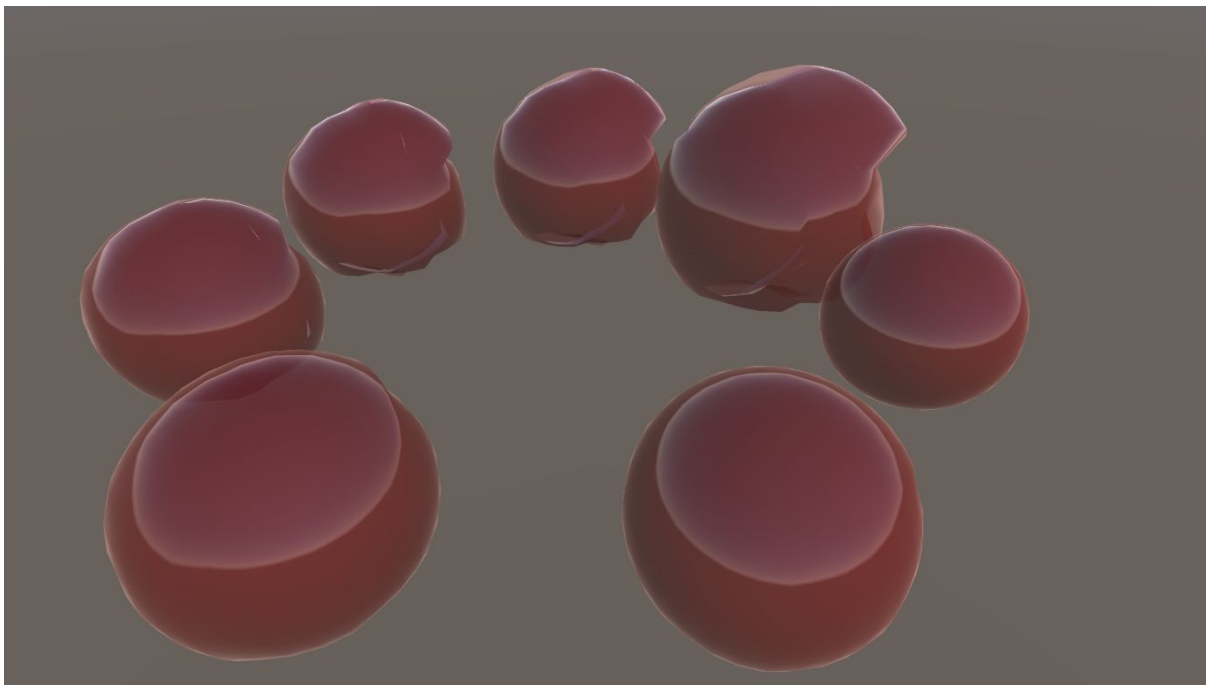

Figura 13

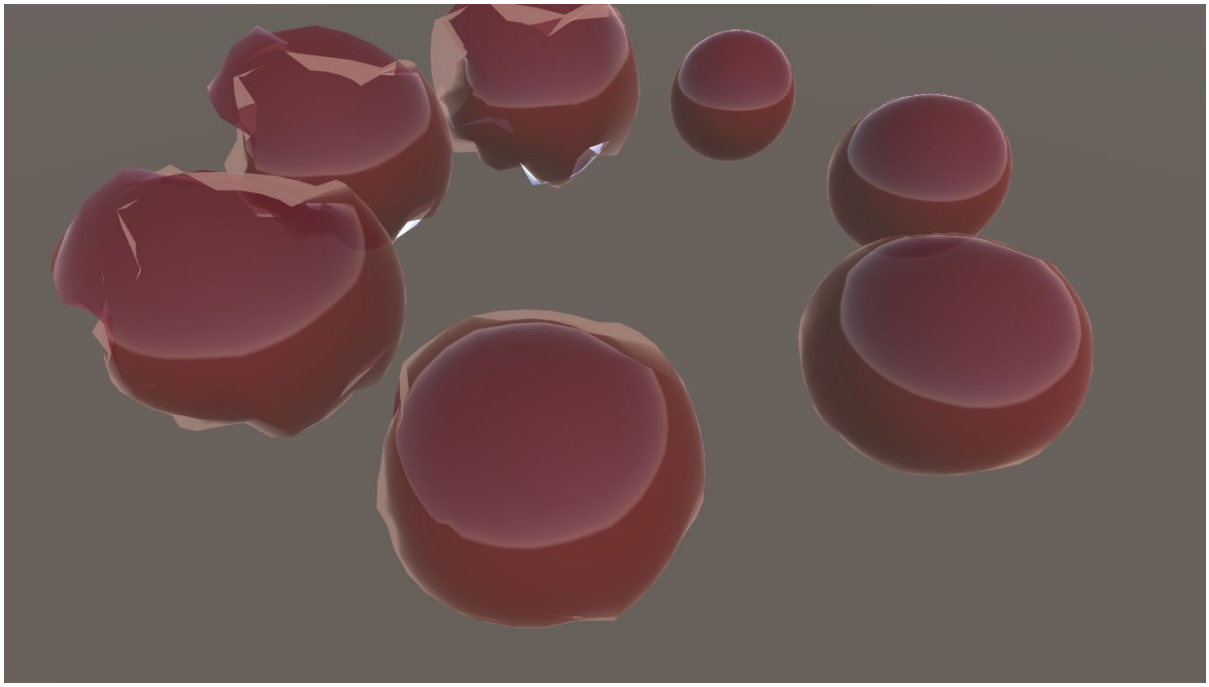

Figura 14

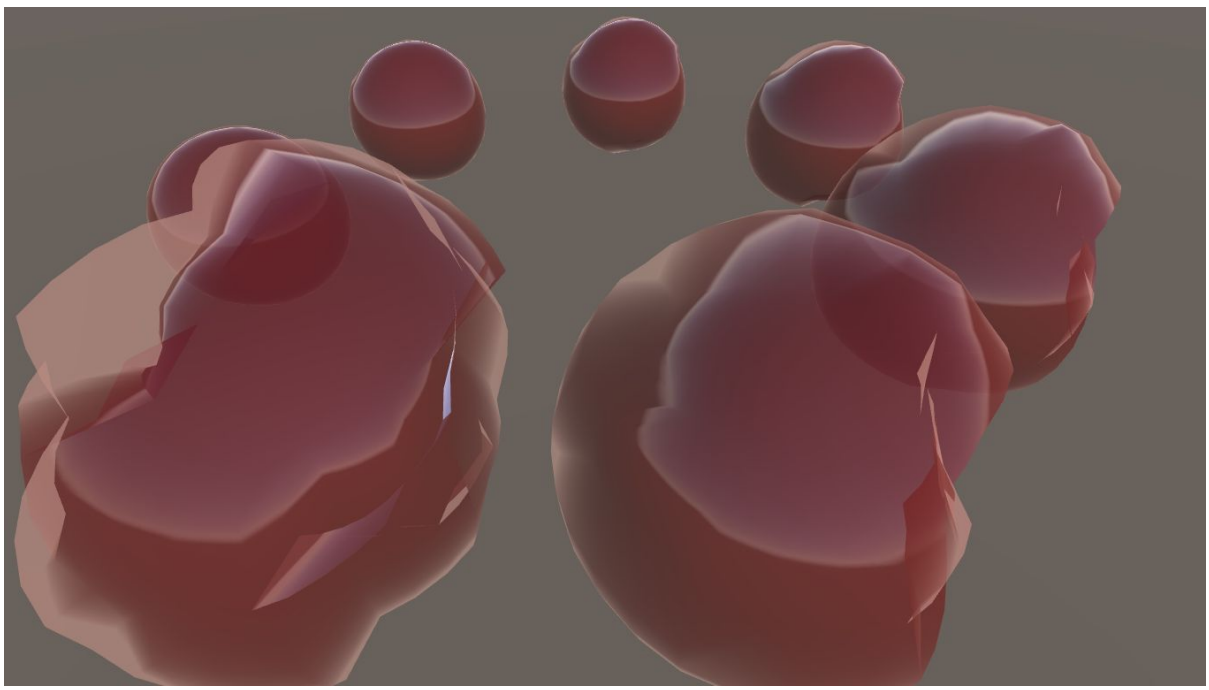

Figura 15

Então esses objetos com mais deformidades são versões antigas e os com aparência de mais “liso” são versões mais recentes representadas. A temos o desenvolvimento do mesmo.

Implemetação em Unity 3D

Dentro de assets temos: Materials, Prefabs e Scripts. Na Hierarchy temos a camera principal, um conjunto contendo as versões, representadas pelas esferas como mostra a **Figura 15**. Temos também um Objeto chamado _Events que possui como componente o script ClickToRotate.cs. O objeto que chamamos de Conjunto possui o script SetVersions.cs.

Em Materials temos armazenada a cor da esfera, que chamamos de Test Texture.mat.

Em prefabs temos o arquivo VersaoPfab.prefab, que é representada por uma esfera contendo o script ModifierVertix.cs. Elas são utilizadas para representar as versões, que na Hierarchy são filhos do Game Object Conjunto.

Na pasta Scripts, temos os seguintes: ClickToRotate.cs, ModifierVertix.cs, RotateMouse.cs, SetVersions.cs. Abaixo estão explicitados os scripts.

Scripts

Script - RotateMouse.cs

```
1. using UnityEngine;
2. using System.Collections;
3.
4. public class RotateMouse : MonoBehaviour {
5.
6. public float minX = -360.0f;
7. public float maxX = 360.0f;
8.
9. public float minY = -45.0f;
10. public float maxY = 45.0f;
11.
12. public float sensX = 100.0f;
13. public float sensY = 100.0f;
14.
```

```
15. float rotationY = 0.0f;
16. float rotationX = 0.0f;
17.
18. void Update () {
19. if (Input.GetMouseButton (1)) {
20. Screen.lockCursor = true;
21. rotationX += Input.GetAxis ("Mouse X") * sensX * Time.deltaTime;
22. rotationY += Input.GetAxis ("Mouse Y") * sensY * Time.deltaTime;
23. rotationY = Mathf.Clamp (rotationY, minY, maxY);
24. transform.localEulerAngles = new Vector3 (-rotationY, rotationX, 0);
25. } else if (Input.GetMouseButtonUp (1)) {
26. Screen.lockCursor = false;
27. }
28. }
29. }
```

30.

31. Simplesmente este script rotaciona a câmera com relação a posição do mouse, travando o cursor na tela e limitada rotação pelos fatores minY e maxY com respeito a Y. E deixando a flexibilidade com respeito a x.

32.

33. Script - ClickToRotate

34.

```
35. using UnityEngine;
36. using System.Collections;
37.
38. public class ClickToRotate : MonoBehaviour {
39.
40. public float rotateSpeed = 10f;
41. public float rotateCamera = 5f;
42.
43. private RaycastHit colisor;
44. private bool pressionado = false;
45.
46. void Update () {
```

```
47.
48. if (Input.GetMouseButton (0)) {
49.
50. var ray = Camera.main.ScreenPointToRay (Input.mousePosition);
51.
52. if (pressionado) {
53. var Conjunto = colisor.transform.parent;
54. Conjunto.Rotate (new Vector3 (0, -Input.GetAxis ("Mouse X") * rotateSpeed, 0));
55. } else {
56. if (Physics.Raycast (ray, out colisor, 100)) {
57.
58. if (colisor.transform.gameObject.layer != 8)
59. return;
60.
61. var Conjunto = colisor.transform.parent;
62. Conjunto.Rotate (new Vector3 (0, -Input.GetAxis ("Mouse X") * rotateSpeed, 0));
63.
64. pressionado = true;
65.
66. }
67. }
68. } else if (Input.GetMouseButtonUp (0)){
69. pressionado = false;
70.
71. }
72. }
73. }
```

Controla a rotação das esferas na tela para uma melhor visualização das mesmas. Basicamente rotaciona o conjunto das esferas em relação ao seu centro.

Script - SetVersions.cs

```
1. using UnityEngine;
```

```
2.  using System.Collections;
3.  using System.Linq.Expressions;
4.  using System.Collections.Generic;
5.
6.
7.  public class SetVersions : MonoBehaviour {
8.
9.  public float raioBase = 5f;
10. public float raioVersao = 2f;
11.
12. private IList<Transform> Versoes;
13.
14. void Start () {
15.
16. Versoes = new List<Transform> ();
17. foreach (var v in transform) {
18. if (((Transform)v).gameObject.layer == 8)
19. Versoes.Add ((Transform)v);
20. }
21.
22. var total = Versoes.Count;
23. var incremento = 2 * Mathf.PI / total;
24.
25. var xc = transform.position.x;
26. var yc = transform.position.z;
27.
28. var i= 0f;
29.
30. var raio = raioBase + raioVersao;
31.
32. foreach (var v in Versoes) {
33. var x = raio * Mathf.Cos (i) + xc;
34. var y = raio * Mathf.Sin (i) + yc;
35. v.position = new Vector3 (x,v.position.y,y);
```

```
36. v.localScale = new Vector3 (raioVersao, raioVersao, raioVersao);
37. i += incremento;
38. }
39. }
40. }
```

Faz a checagem de quantas versões existem para criar um conjunto de esferas que seguem um alinhamento circular. O conjunto é determinado por quantas esferas terão, e divididas no espaço a medida q formem uma circunferência de 360º igualmente espaçadas.

Script - ModifierVertix.cs

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using System.Linq.Expressions;
5.  using System.Linq;
6.
7.  public class ModifierVertix : MonoBehaviour {
8.
9.  [Range(0f, 3f)]
10. public float amplitude = 0.5f;
11.
12. private Vector3[] vertsBkup;
13. private Vector3[] verts;
14. private Dictionary<Vector3, List<int> > dictionary;
15. private float angulo = 0;
16.
17. void Start()
18.
19. {
20. Mesh mesh = GetComponent<MeshFilter>().mesh;
21. vertsBkup = mesh.vertices;
22. verts = (Vector3[])vertsBkup.Clone ();
```

```
23. dictionary = new Dictionary<Vector3, List<int> > ();
24. for (var x = 0; x < verts.Length; x++) {
25. if (!dictionary.ContainsKey (verts [x])) {
26. dictionary.Add (verts [x], new List<int> ());
27. }
28. dictionary [verts [x]].Add (x);
29. }
30. }
31.
32. void Update() {
33.
34. Mesh mesh = GetComponent<MeshFilter>().mesh;
35.
36. float inc = 2*Mathf.PI/dictionary.Count;
37. float aux = 0;
38. foreach (var pair in dictionary) {
39. foreach (var i in pair.Value) {
40. var ang = (angulo + aux)% (2*Mathf.PI);
41. verts [i] =  (1+amplitude*Mathf.Abs(Mathf.Sin(ang)))*vertsBkup[i];
42. }
43. aux += inc;
44. }
45.
46. angulo = (angulo  + Time.deltaTime) % (2*Mathf.PI);
47.
48. mesh.vertices = verts;
49. mesh.RecalculateBounds();
50. }
51. }
```

Toda modificação dos vértices da esfera é realizada aqui. No qual dependendo da amplitude ajustada o nível de deformação aumenta, demonstrado por uma deformação senoidal com respeito aos vértices. A estrutura inicial de cara esfera é guardada para fins de cálculos e então aplicadas as deformações senoidais em cada vértice.

Conclusões e Trabalhos Futuros

Bastante interessante este desenvolvimento do trabalho por mostrar um campo pouco anteriormente conhecido pelo autor deste trabalho. A principio pensava-se que tomaria mais do tempo para a construção das aplicações, mas a tecnologia cresceu a tal ponto que tudo foi facilitado(como sempre). Por meio do Unity e seus diversos pacotes foi possível passar a ideia com rapidez e facilidade. Claro que no inicio da primeira aplicação seu desenvolver foi um pouco mais lento que as outro, porém depois de estar acostumado com o ambiente de trabalho foi possível agilizar o processo da criação destes protótipos.

As aplicações em si neste trabalho poderiam ter uma forma mais automatizada para testes e até para já facilitar um meio de interação mais rápida do programador com a máquina (exemplo, ler arquivos de excel, e gerar os dados a partir deles, ou ler um e-mail válido e testar). Ficou bem aberto para quem vai utilizar estas aplicações como forma de API como fazer esta ligação, o que achamos bastante interessante pois não fixa o programador a seguir determinada forma, tornando assim mais flexível a maneira de codificar seus protótipos.

Para quem tiver interesse em desenvolver metáforas de interface, voltadas para esta área, o Prof. Dr. Pedro Alessio tem um doutorado na área cheio de ideias para trabalhar bem muito. Para quem tiver interessado checar a bibliografia[1] pois seu trabalho é realmente bom e mostrado a possibilidade de conseguir extrair as ideias projetadas a maneira de serem importadas para aplicações reais.

Bibliografia

[1] Martins Alessio, Pedro. "De la Métaphore à la tâche: une bibliothèque de concepts métaphoriques pour le prototypage de techniques d'interactions." PhD diss., Paris, CNAM, 2013.

[2] Lakoff, George & Mark Johnson (1980) Metaphors We Live By. Chicago: University of Chicago Press.

[3] Malone, T.W., 1983. How do people organize their desks?: Implications for the design of office information systems. ACM Transactions on Information Systems, 1(1), pp.99–112.

[4] Embodied Cognition: A field guide, Michael L. Anderson, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA.

[5] Unity, Game engine, tools and multiplataform, http://unity3d.com/pt/unity, acessado a ultima vez em 10 de julho de 2016.

[6] Unity Scripting API, https://docs.unity3d.com/ScriptReference/, acessado a ultima vez em 10 de julho de 2016

[7] Unity - Manual: Materials, Shaders & Textures, ,https://docs.unity3d.com/Manual/Shaders.html, acessado a ultima vez em 11 de julho de 2016.

[8] Unity - Manual: Prefabs, https://docs.unity3d.com/Manual/Prefabs.html, acessado a última vez em 11 de julho de 2016