

```
    check = false;
```

```
}
```

```
if (automatosGerados[i].nome == "backtrack" && !check)
```

```
{
```

ALGORITMO DE PROFOUNDIDADE PARA VIAGEM

Inteligência Artificial

Equipe: Eduardo Tavares, Guilherme Andrade,

Nadianne Galvão, Rafael Rezende,

Valdir Santana & Verenilson da Silva

```
string log = "";
```

```
foreach (Automaton aut in automatosGerados)
```



```
    check = false;
```

```
}
```

```
if (automatosGerados[i].nome != "backtrack")
```

```
{
```

```
    indexFix = i - 1;
```

```
    check = true;
```

```
}
```

```
if (check && indexFix > 0)
```

```
{
```

```
    automatosGerados[i] = novoAutomaton(automatosGerados[indexFix]); // caminho[indexFix];
```

```
    ix--;
```



Implementação realizada em C# usando a engine Unity para visualização e interação com o usuário.



OBJETIVO:

Encontrar um caminho de viagem entre duas cidades em um conjunto de autômatos.

```
foreach (Automaton a in automatosGerados)
```

AUTÔMATOS:

Representa as conexões possíveis entre as localidades e permite determinar a sequência de transições necessárias para alcançar a localidade de destino a partir da localidade de origem.

```
    check = false;
```

```
}
```

```
if (automatosGerados[i].nome != "backtrack")
```

```
{
```

```
    ind
```

```
    che
```

```
}
```

```
if (che
```

```
{
```

```
    aut
```

```
    ind
```

```
}
```

```
/aqui sem
```

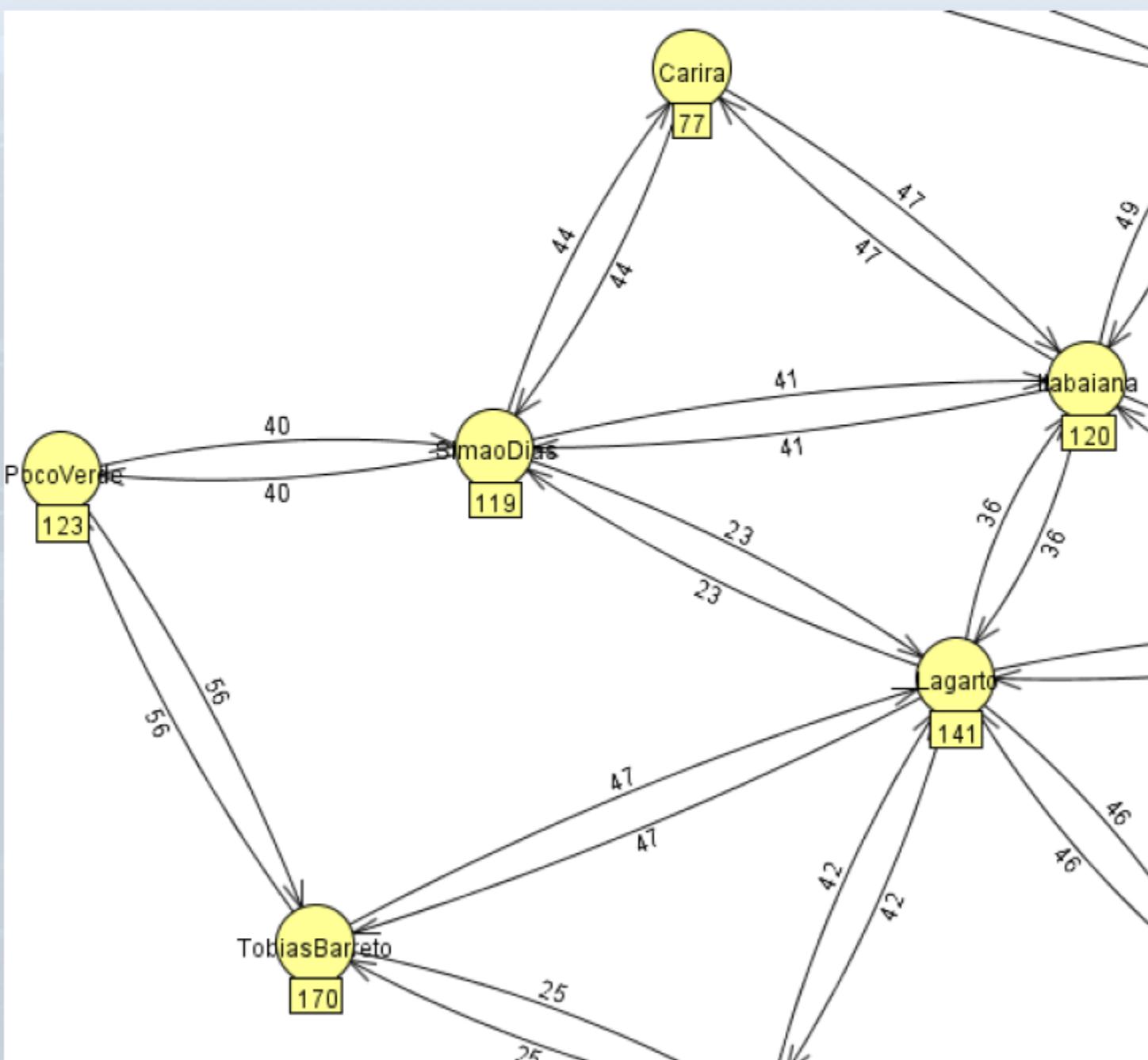
```
if (debugPr
```

```
{
```

```
    string log = "
```

```
foreach (Automaton aut in automatosGerados)
```

AUTÔMATOS:



```
<state id="0" name="SimaoDias">#13;  
| <x>369.755</x>#13;  
| <y>396.0</y>#13;  
| <label>119</label>#13;  
</state>#13;
```

```
<transition>#13;  
| <from>0</from>#13;  
| <to>5</to>#13;  
| <read>44</read>#13;  
</transition>#13;
```

ALGORITMO DE PROFOUNDIDADE PARA VIAGEM

```
    check = false;
}
if (automatosGerados[0] == null)
{
    indexFix = i - 2;
    check = true;
}

if (check && indexFix >= 0)
{
    automatosGerados[indexFix] = null;
    indexFix--;
}
}

// aqui serve só pra printar
if (debugPrint)
{
    string log = "";
```

```
namespace AutomatonNameSpace
{
    using System.Collections.Generic;
    public class Automaton
    {
        public int id;
        public float x, y, label;
        public string nome;
        public List<int[]> transition;
        public bool visitado;
        public Automaton(int id, string nome, float x, float label) //cria um novo automato
        {
            this.id = id;
            this.nome = nome;
            this.x = x;
            this.y = y;
            this.label = label;
            transition = new();
            visitado = false;
        }
        public Automaton(Automaton automaton) //recebe uma classe dela mesma para copiar
        {
            id = automaton.id;
            nome = automaton.nome;
            x = automaton.x;
            y = automaton.y;
            label = automaton.label;
            transition = automaton.transition;
            visitado = automaton.visitado;
        }
    }
}
```

ALGORITMO

de busca em profundidade

Explora todos os vértices de um grafo seguindo uma ramificação específica até alcançar o objetivo ou até não haver mais vértices a explorar.



ALGORITMO DE
PROFOUNDIDADE
PARA VIAGEM

```
    check = false;
```

```
}
```

```
if (automatosGerados[i].nome != "backtrack")
```

```
{
```

```
    indexFix = i - 2;
```

```
    check = true;
```

É aplicado quando a busca em profundidade atinge um ponto em que não é possível avançar para nenhum estado não visitado a partir do estado atual.

```
}
```

```
// aqui serve só pra printar no terminal da unity pra ver se ta tudo certo
```

```
if (debugPrint)
```

```
{
```

```
    string log = "";
```



LER ARQUIVO

```
for (int i = 0, indexI = 0, i < automatos.Count, i++)
{
    Automaton auto = automatos[i];
    if (auto.id == indexI)
    {
        check = false;
    }
    else
    {
        if (check)
        {
            i++;
            continue;
        }
        else
        {
            if (auto.name != null)
            {
                string log = "Lendo arquivo XML...\n";
                foreach (Automaton aut in automatosGerados)
                {
                    log += "Automato " + aut.name + ":\n";
                    foreach (Transition tran in aut.transitions)
                    {
                        log += "  Transição para " + tran.to + " com leitura " + tran.read + "\n";
                    }
                }
                Console.WriteLine(log);
            }
            else
            {
                string log = "Lendo arquivo XML...\n";
                foreach (Automaton aut in automatos)
                {
                    log += "Automato " + aut.name + ":\n";
                    foreach (Transition tran in aut.transitions)
                    {
                        log += "  Transição para " + tran.to + " com leitura " + tran.read + "\n";
                    }
                }
                Console.WriteLine(log);
            }
        }
    }
}

public class Automaton
{
    public int id { get; set; }
    public string name { get; set; }
    public List<Transition> transitions { get; set; }
}

public class Transition
{
    public string from { get; set; }
    public string to { get; set; }
    public string read { get; set; }
}
```

ALGORITMO DE PROFOUNDIDADE PARA VIAGEM

```
check = false;
}
if (automatosGerados != null)
{
    indexFix = i;
    check = true;
}
if (check && indexFix > 0)
{
    automatosGerados.Add(automato);
    indexFix--;
}
// aqui serve só pra debug
if (debugPrint)
{
    string log = "";
    for (int i = 0; i < automatosGerados.Count; i++)
    {
        log += automatosGerados[i].ToString();
    }
    Console.WriteLine(log);
}
```

```
public class Profundidade
{
    public string origem, destino;
    public bool chegou = false;
    List<Automaton> automatos = new();
    Automaton automatoStart;
    public List<Automaton> automatosGerados;
    public bool itsOkay, debugPrint;
    public Profundidade(string origem, string destino, List<Automaton> automatos)
    {
        this.origem = origem;
        this.destino = destino;
        this.automatos = automatos;
        itsOkay = false;
        automatosGerados = new List<Automaton> ();
    }
    public void RodarAutomato()
    {
        if (check)
        {
            debugPrint = false;
            foreach(Automaton automaton in automatos)
            {
                if(automaton.nome == origem)
                {
                    automatoStart = automaton;
                    automatoStart.visitado = true;
                    automatosGerados.Add(new Automaton(automaton));
                    break;
                }
            }
        }
    }
}
```

```
    if (automatosGerados[i].nome != "backtrack")
    {
        check = ProfundidadeLookForWay(new Automaton(automatoStart));
        bool check = false;

        for(int i = 0, indexFix=0; i < automatosGerados.Count; i++)
        {
            if (automatosGerados[i].nome != "backtrack")
            {
                check = false;
            }
            if (automatosGerados[i].nome == "backtrack" && !check)
            {
                indexFix = i - 2;
                check = true;
            }
            if (check && indexFix >= 0)
            {
                automatosGerados[i] = new Automaton(automatosGerados[indexFix]);
            }
        }
        //aqui serve para debug
        if (debugPrint)
        {
            string log = " ";
            foreach (Automaton aut in automatosGerados)
```

```
    automatosGerados[il].nome := "Baía de São João"
    check = false
}
if (automatosGerados[il].nome == "Baía de São João")
{
    indexFix = i;
    check = true;
}

if (check && indexFix != -1)
{
    automatosGerados[indexFix].info[indexFix];
    indexFix--;
}
}

// aqui serve só pra
if (debugPrint)
{
    string log = "";
    foreach (Automaton automato in automatosGerados)
        log += automato.info[automato.id] + " ";
    Console.WriteLine(log);
}
```

```
public void ProfundidadeLookForWay(Automaton automatoSwitch)
{
    if(automatoSwitch.nome == destino)
    {
        chegou = true;
        return;
    }
    foreach(int[] tran in automatoSwitch.transition)
    {
        if(chegou)
        {
            break;
        }
        foreach(Automaton automato in automatos)
        {
            if (chegou)
            {
                break;
            }
            if (tran[0] == automato.id && automato.visitado == false)
            {
                automato.visitado = true;
                automatosGerados.Add(new Automaton(automato));
                ProfundidadeLookForWay(new Automaton(automato));
            }
        }
    }
}
```