

UNIVERSIDADE DO MINHO

RELATÓRIO DE PROJETO

---

# DSL para Árvores de Comportamento (Behavior Trees)

---

*Autores:*

Miguel Oliveira  
Pedro Mimoso Silva  
Pedro Moura

*Supervisores:*

Pedro Rangel Henriques  
José João Almeida

*Projeto realizado no âmbito da UC de Laboratórios de Engenharia Informática,  
pertencente ao Mestrado em Engenharia Informática*

*do*

Departamento de Informática da Universidade do Minho

11 de Março de 2020

UNIVERSIDADE DO MINHO

## *Resumo*

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

# Conteúdo

<b>Resumo</b>	<b>i</b>
<b>1 Introdução</b>	<b>1</b>
<b>2 Behavior Trees - Básicos</b>	<b>2</b>
2.1 O que são <i>Behavior Trees</i> ? . . . . .	2
2.2 Execução . . . . .	2
2.3 Nós . . . . .	3
2.3.1 <i>Control Flow Nodes</i> - Clássicos . . . . .	3
2.3.2 <i>Control Flow Nodes</i> - Novos . . . . .	4
2.3.3 <i>Execution Nodes</i> . . . . .	4
2.4 Exemplo - <i>PACMAN</i> . . . . .	5
<b>3 Linguagem</b>	<b>6</b>
<b>4 Compilador</b>	<b>7</b>
<b>5 Caso de Estudo</b>	<b>8</b>
<b>6 Conclusão</b>	<b>9</b>
<b>A Frequently Asked Questions</b>	<b>10</b>
A.1 How do I change the colors of links? . . . . .	10

# Lista de Figuras

2.1	Exemplo de uma BT. . . . .	2
2.2	Estrutura de um nodo <i>Sequence</i> . . . . .	3
2.3	Estrutura de um nodo <i>Selector</i> . . . . .	3
2.4	Estrutura de um nodo <i>Parallel</i> com taxa de sucesso $M$ . . . . .	4
2.5	Estrutura de um nodo <i>Action</i> . . . . .	4
2.6	Estrutura de um nodo <i>Condition</i> . . . . .	4
2.7	Exemplo de uma expressão condicional numa BT. . . . .	4

# Lista de Tabelas

# Lista de Abreviações

<b>BT</b>	<b>Behavior Tree</b>
<b>DSL</b>	<b>Domain Specific Language</b>
<b>NPC</b>	<b>Non-Playable Character</b>

## Capítulo 1

# Introdução

O presente relatório descreve o desenvolvimento do projeto da UC de Laboratórios em Engenharia Informática, pertencente ao 1º ano do Mestrado em Engenharia Informática da Universidade do Minho.

Este projeto consiste no desenvolvimento de uma linguagem textual de domínio específico (DSL) para representar árvores de comportamento (às quais nos referiremos daqui em diante por *Behavior Trees*) e um compilador capaz de traduzir esta linguagem para uma linguagem de programação já existente (por exemplo *Python*).

De modo conciso, *Behavior Trees* são formalismos para descrever comportamentos reativos de atores autónomos, como um robô, ou um NPC num jogo. Com esta linguagem, pretendemos criar uma forma simples e intuitiva de especificar estas árvores, e ao mesmo tempo acrescentar novas funcionalidades que, a nosso ver, podem melhorar a qualidade da implementação de comportamentos nestes atores.

- Estrutura do relatório
  - Behavior Trees - Basics
  - Linguagem
  - Compilador
  - Implementação
  - Caso de Estudo
  - Conclusão

## Capítulo 2

# Behavior Trees - Básicos

### 2.1 O que são Behavior Trees?

*Behavior Trees*, ou BT, são estruturas que controlam a execução de um conjunto ações por parte de um agente autônomo, de forma a descreverem o seu comportamento. Este tipo de estrutura começou por ser utilizado especialmente em videojogos (para simular o comportamento de NPCs), mas com o passar do tempo áreas como a Robótica ou a Inteligência Artificial também o começaram a usar, devido à sua capacidade modular e reativa.

Formalmente, uma BT é uma árvore com raiz onde os nodos internos são chamados *control flow nodes* e as folhas são chamadas de *execution nodes*. Para cada nodo conetado usamos a terminologia de *parent* (pai) e *child* (filho). A raiz é o único nodo que não tem pais, todos os outros nodos têm exatamente um pai. Os *control flow nodes* têm pelo menos um filho, já os *execution nodes* não têm nenhum.

A figura 2.1 mostra um exemplo de uma BT, onde o nodo a verde é a raiz, os nodos a cinzento são *control flow nodes* e os restantes são *execution nodes*.

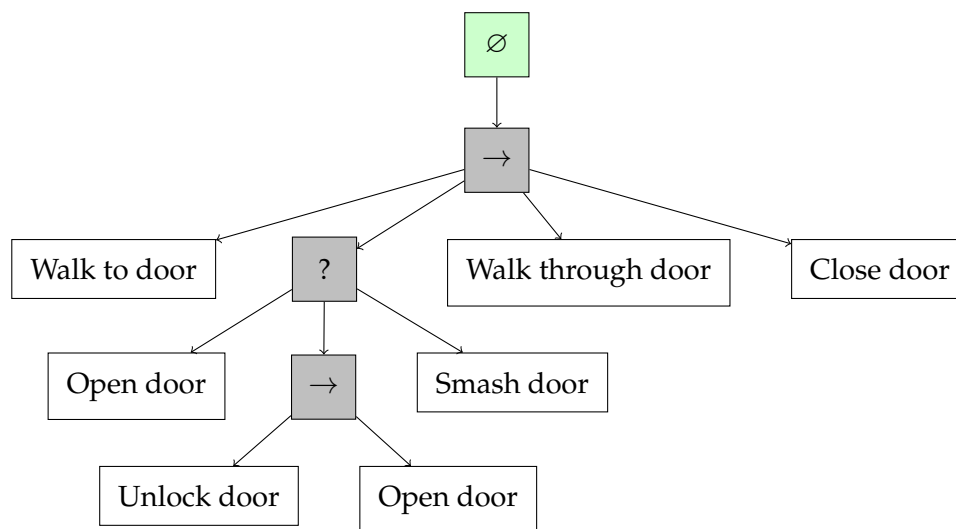


FIGURA 2.1: Exemplo de uma BT.

### 2.2 Execução

Uma BT começa a sua execução pelo nodo raiz que gera sinais chamados *ticks* com uma determinada frequência, que são enviados para os seus filhos. Estes sinais permitem a execução dos nodos.

Qualquer nodo, não importa o tipo, ao ser executado retorna um de três estados:



- *Success* - foi executado com sucesso;
- *Failure* - não conseguiu ser executado;
- *Running* - ainda está a executar.

## 2.3 Nós

### 2.3.1 Control Flow Nodes - Clássicos

*Control flow nodes* são nós estruturais, que não têm qualquer impacto no estado global do sistema. Na formulação clássica, existem 4 categorias deste tipo de nó: *Sequence*, *Selector* (ou *Fallback*), *Parallel* e *Decorator*.

**Sequence** Um nó *Sequence* visita (envia *ticks*) todos os filhos por ordem, começando pelo primeiro, e se este retornar *Success*, chama o segundo, e por aí em diante. Caso um filho falhe, o nó *Sequence* retorna *Failure* imediatamente. Caso todos os filhos sucedam, o nó retorna *Success*. Caso um filho retorne *Running*, o nó retorna também *Running*.

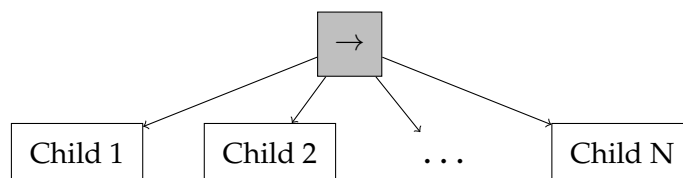


FIGURA 2.2: Estrutura de um nó *Sequence*.

**Selector** Tal como o *Sequence*, o nó *Selector* visita todos os filhos por ordem, mas só avança para o próximo se o filho que está a ser executado retornar *Failure*. Caso um filho suceda, o nó *Sequence* retorna *Success* imediatamente. Caso todos os filhos falhem, retorna *Failure*. Caso um filho retorne *Running*, o nó retorna também *Running*.

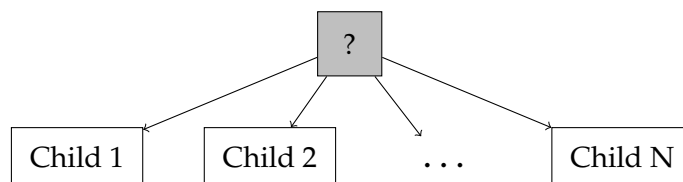
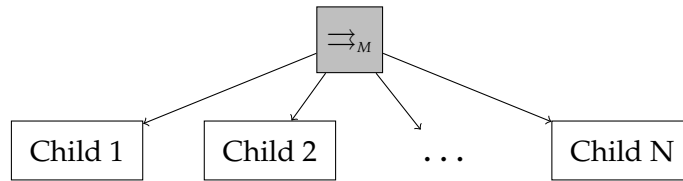


FIGURA 2.3: Estrutura de um nó *Selector*.

**Parallel** Um nó *Parallel*, como o nome indica, visita todos os filhos paralelamente. Para  $M \leq N$ , retorna *Success* caso  $M$  filhos sucedam, e retorna *Failure* caso  $N - M + 1$  filhos retornem *Failure*. Caso contrário, retorna *Running*.

FIGURA 2.4: Estrutura de um nó *Parallel* com taxa de sucesso  $M$ .**Decorator** POR FAZER

### 2.3.2 Control Flow Nodes - Novos

POR FAZER

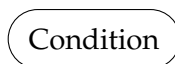
#### 2.3.3 Execution Nodes

*Execution nodes* são os nodos mais simples, porém os mais poderosos, pois contêm os testes ou ações que serão implementados pelo sistema. Existem 2 categorias para este tipo de nodos: *Action* e *Condition*.

**Action** Quando recebe *ticks*, um nó *Action* executa um ou mais comandos. Retorna *Success* se a ação foi corretamente completada ou *Failure* se a ação falhou. Enquanto está a ser executado, retorna *Running*.

FIGURA 2.5: Estrutura de um nó *Action*.

**Condition** Quando recebe *ticks*, um nó *Condition* verifica uma proposição. Retorna *Success* ou *Failure* dependendo se a proposição é válida ou não. De notar que um nó *Condition* nunca retorna um estado *Running*.

FIGURA 2.6: Estrutura de um nó *Condition*.

**NOTA:** Quando juntamos o nó *Condition* com os nodos *Sequence* e *Selector* podemos criar uma expressão condicional (*if-then-else*). Vejamos o seguinte exemplo:

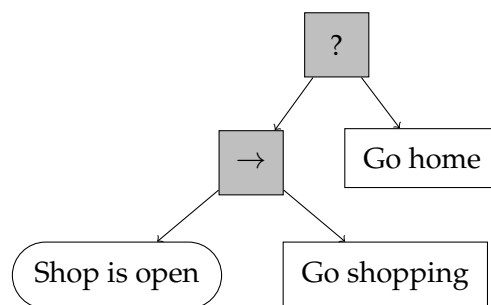


FIGURA 2.7: Exemplo de uma expressão condicional numa BT.

Como podemos ver, a escolha da ação a executar alterna entre ir às compras ou ir embora consoante a loja esteja aberta ou não. Ou seja,

if Shop is open then Go shopping else Go home

---

## 2.4 Exemplo - PACMAN

## Capítulo 3

# Linguagem

## Capítulo 4

# Compilador

## **Capítulo 5**

# **Caso de Estudo**

## Capítulo 6

# Conclusão

## Apêndice A

# Frequently Asked Questions

### A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

```
\hypersetup{urlcolor=red}, or  
\hypersetup{citecolor=green}, or  
\hypersetup{allcolor=blue}.
```

If you want to completely hide the links, you can use:

```
\hypersetup{allcolors=.}, or even better:  
\hypersetup{hidelinks}.
```

If you want to have obvious links in the PDF but not the printed text, use:

```
\hypersetup{colorlinks=false}.
```