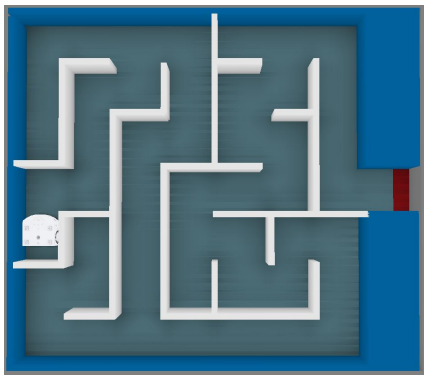

Thymio MazeRunner

André Correia, fc49450 & Xiaoyi Cheng, fc49446
Faculdade de Ciências da Universidade de Lisboa

Departamento de Informática - Mestrado em Engenharia Informática - Robôs Móveis

Professor Responsável: Luís Correia

Contactos:
André Correia: fc49450@alunos.fc.ul.pt;
Xiaoyi Cheng, fc49446@alunos.fc.ul.pt

Resumo

A ideia do nosso projeto consiste na utilização de diferentes funcionalidades do robô Thymio para chegar a um determinado lugar dentro de um Labirinto adaptando-se ao mundo sobre o qual é colocado, memorizando o caminho certo.

O robô terá uma ação de exploração à força bruta para achar o lugar; o robô não sabe onde se encontra o seu objetivo, logo vai ter que tentar encontrar através dos caminhos que encontre. Uma vez encontrado o local o Thymio terá o caminho correto guardado.

Palavras-chave: Thymio, Robôs Móveis, Labirinto

Método

Abordagem:

Começámos por definir os principais problemas que poderíamos encontrar e o resultado foi:

⇒ Um mundo (playground) seria necessário para o desenvolvimento do trabalho, devido à situação de pandemia foi tudo desenvolvido em simulador.[1]

⇒ Alguma maneira de guardar a informação pretendida, o Thymio não está concebido para guardar muita informação, logo tivemos de fazer alguma pesquisa sobre o assunto.

De seguida verificámos que funcionalidades seriam necessárias o nosso robô ter, pelo que utilizámos fortemente os sensores de forma a detetar proximidade.

Depois de várias tentativas (como demonstram as imagens abaixo) chegámos a uma conclusão.

O nosso Labirinto 3D teria de ser um modelo relativamente pequeno, suficientemente complexo, e os sensores teriam de conseguir detetar as paredes (obstáculos) do mesmo.

Teria de possuir uma zona de partida (onde por defeito se encontra o Thymio) e uma zona de chegada (objetivo do Thymio).

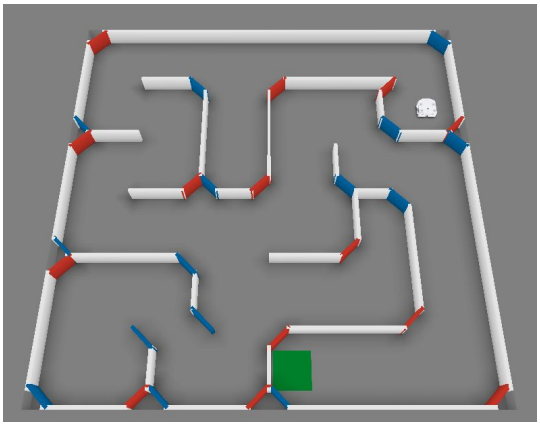

Figura 1 - Primeira versão do Labirinto. Muito complexo e os cantos com ângulo interferiam com o normal funcionamento dos sensores.

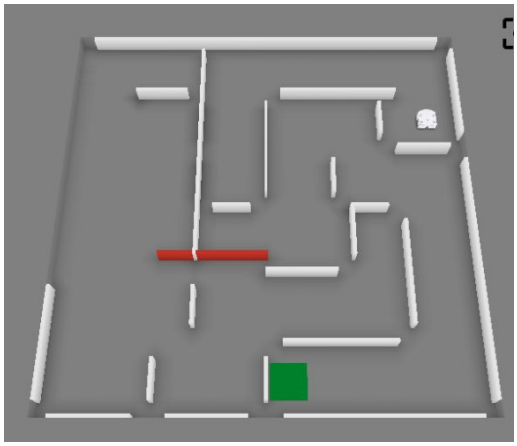

Figura 2 - Deconstruímos a 1ª versão do Labirinto e começámos a ajustar o mesmo conforme as necessidades notadas no Aseba Studio.

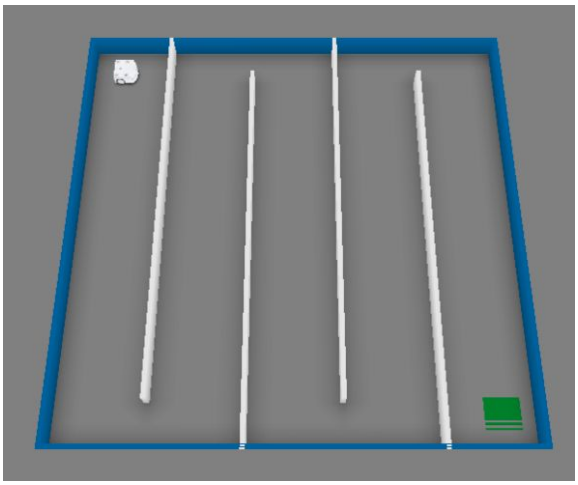

Figura 3 - 3ª versão do Labirinto. Versão de teste. Bastante simples de percorrer, mas servia principalmente para verificar que valores os sensores deveriam ter na deteção de colisão. Ainda se encontra bastante grande, necessário ajustes à escala..

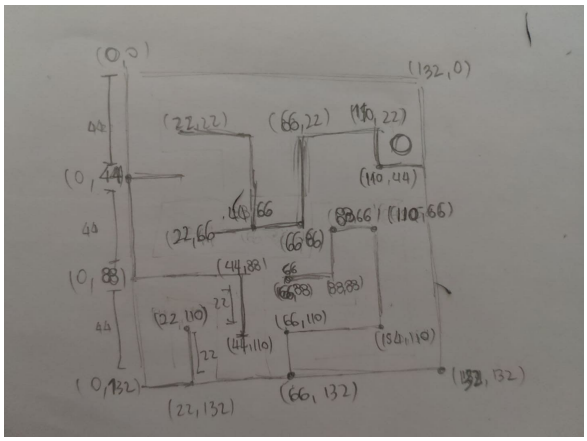

Figura 4 - Foi desenhado em papel um protótipo de como deveria ser o Labirinto tendo em conta os erros dos modelos anteriores.

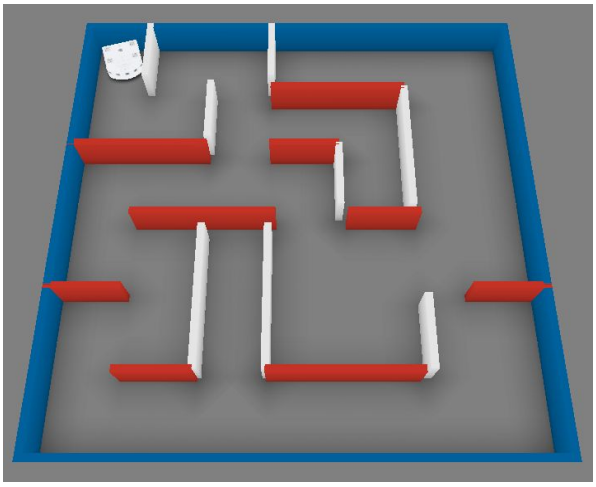

Figura 5 - 4ª versão do Labirinto. Praticamente terminado, apenas faltava ajustar uns pontos.

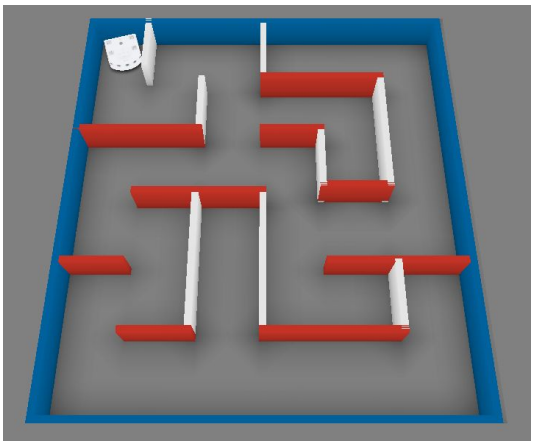

Figura 6 - 5ª Versão do Labirinto. Foram feitos pequenos ajustes. Serviu principalmente para testes devido à sua simplicidade.

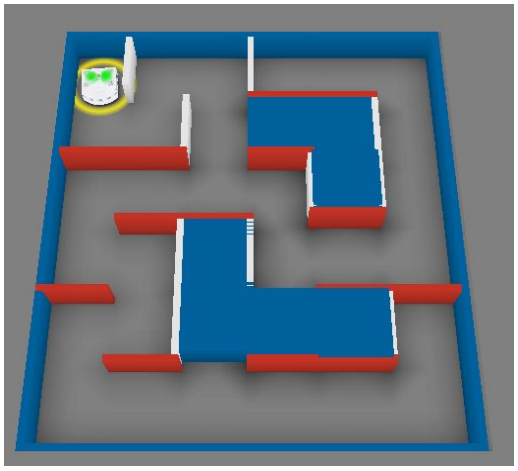

Figura 7 - 6ª Versão do Labirinto.Adicionadas "ilhas" preenchidas em vez de becos uma vez que o algoritmo de procura do caminho foi alterado. Também serviu para testes.

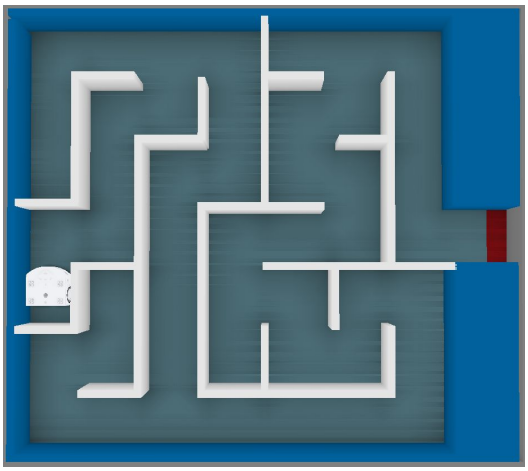

Figura 8 - Adicionado complexidade ao Labirinto. Remoção de ilhas, pois com esta arquitetura é possível manter o algoritmo desenhado.

Adicionado um fundo de modo ao Thymio detectar a zona de chegada (vermelho).

Implementação:

O programa desenvolvido consiste num ficheiro *aesl* e num ficheiro *playground*, nomeadamente, o ficheiro *thymioMazeRunner.aesl* e o ficheiro ******.playground* que segue em anexo, utilizado para definir o comportamento pretendido.

O nosso projeto inicia-se com o Thymio parado dentro do Labirinto.

São declaradas constantes de decisão (Estados):

- YES - verifica-se a condição
- NO - não se verifica a condição
- SNIFF_LEFT - verifica a proximidade à esquerda
- SNIFF_RIGTH - verifica a proximidade à direita
- DECIDE - escolhe esquerda ou direita conforme os valores dos sensores
- TURN_LEFT - o robô vira à esquerda até encontrar caminho em frente
- TURN_RIGTH - o robô vira à direita até encontrar caminho em frente
- GO_BACK - o robô vira à esquerda até encontrar caminho em frente
- FRONT - o robô segue em frente
- SUCCESS - o robô chegou no meta e fica parado

Tentámos implementar uma máquina de estados, isto é, conforme cada situação detectada, existem já condições específicas que indicam o estado seguinte que deve ser realizado para essa situação (estado atual).

O robô tentará sempre seguir em frente e começa com uma velocidade de 300 em cada roda:

```
18 event.args[0] = 0
19 #movimento inicial
20 motor.left.target = 200
21 motor.right.target = 200
22
23 call leds.top(0,0,32)
24
```

Figura 9 - Estado inicial do robô

Utilizámos os sensores de proximidade horizontais:

- prox.horizontal[0] : front left;
- prox.horizontal[1] : front middle-left;
- prox.horizontal[2] : front middle;
- prox.horizontal[3] : front middle-right;
- prox.horizontal[4] : front right;
- prox.horizontal[5] : back left;
- prox.horizontal[6] : back right;

Utilizámos também os LED como um sistema de cores que indica visualmente em qual dos estados se encontra o nosso Thymio.

1. Azul:
    a. Decisão tomada
        i. Estados - TURN_LEFT, TURN_RIGTH, GO_BACK, FRONT
2. Amarelo:
    a. Decisão pendente
        i. Estados - SNIFF_LEFT, SNIFF_RIGTH
3. Verde
    a. Fim do percurso
        i. Estado - SUCESS

Sempre que se verificava valores superiores a 3800 (testados empiricamente), significava que o nosso robô encontrou um obstáculo (parede). Aí ele começa por verificar se existe um caminho possível à direita (SNIFF_RIGHT) ou se se encontra num canto.

```
37 onevent prox
38
39   when prox.horizontal[1] >= 3800
40   and prox.horizontal[2] >= 3800
41   and prox.horizontal[3] >= 3800 do
42
43     if timer == NO
44     and event.args[1] != SUCCESS then
45       event.args[0] = SNIFF_RIGHT
46
47     elseif event.args[1] == SUCCESS then
48       event.args[0] = DECIDE
49     end
50   end
51 timer.period[1] = 10
52
```

Figura 10 - Primeiro estado caso se encontre proximidade - SNIFF_RIGHT

Após ser declarado SNIFF_RIGHT, o nosso robô vira à direita através da mudança de velocidade nas rodas durante 2,3s:

```
124   elseif event.args[0] == SNIFF_LEFT then
125     call leds.top(32,32,0)
126     motor.left.target = 0
127     motor.right.target = 200
128     timer.period[0] = 2300
129     timer = YES
130     button.center = 1
```

Figura 11 - SNIFF_RIGHT - A roda esquerda fica parada enquanto a direita roda

Aqui o robô parado irá decidir o que deve ser feito com base nas leituras dos sensores:

```
204 onevent timer0
205
206   motor.left.target = 0
207   motor.right.target = 0
208   if prox.horizontal[1] >= 0
209   and prox.horizontal[2] >= 0
210   and prox.horizontal[3] >= 0 then
211     if  prox.horizontal[0] == 0
212     or prox.horizontal[1] == 0 then
213       LEFT = YES
214     elseif prox.horizontal[4] == 0
215     or prox.horizontal[3] == 0 then
216       RIGHT = YES
217     end
218   end
219   if timer == NO then
220     event.args[0] = SNIFF_LEFT
221   else
222     event.args[0] = DECIDE
223   end
224
225   timer.period[0] = 0
226
```

Figura 12 - Encontra um obstáculo e decide o próximo estado.

Se for decidido que é para virar à esquerda / direita:

```
58   if event.args[0] == TURN_LEFT then
59     call leds.top(0,0,32)
60     motor.left.target = 0
61     motor.right.target = 100
62     calc = abs(prox.horizontal[5] -
   prox.horizontal[6])
63     calc2 = abs(prox.horizontal[3] -
   prox.horizontal[4])
64
65     if prox.horizontal[5] != 0
66     and prox.horizontal[6] != 0
67     and calc < 30
68     and prox.horizontal[2] < 2000 then
69       event.args[0] = FRONT
70     elseif (prox.horizontal[5] == 0
71     or prox.horizontal[6] == 0)
72     and calc2 < 1000 and calc2 > 400
73     and prox.horizontal[2] < 2000 then
74         event.args[0] = FRONT
75
76     end
```

```
78   elseif event.args[0] == TURN_RIGHT then
79       call leds.top(0,0,32)
80     motor.left.target = 100
81     motor.right.target = 0
82     calc = abs(prox.horizontal[5] -
   prox.horizontal[6])
83     calc2 = abs(prox.horizontal[0] -
   prox.horizontal[1])
84
85     if prox.horizontal[5] != 0
86     and prox.horizontal[6] != 0
87     and calc < 30
88     and prox.horizontal[2] < 2000 then
89       event.args[0] = FRONT
90     elseif (prox.horizontal[5] == 0
91     or prox.horizontal[6] == 0)
92     and calc2 < 1000 and calc2 > 400
93     and prox.horizontal[2] < 2000 then
94         event.args[0] = FRONT
95
96     end
```

Figura 13 - A diferença entre os dois valores testados comparada com uma constante determina o próximo estado.

Aqui mostramos o que o robô realiza caso seja para seguir e frente ou andar para trás:

```
100   elseif event.args[0] == GO_BACK then
101     call leds.top(0,0,32)
102     motor.left.target = 100
103     motor.right.target = 0
104     calc = abs(prox.horizontal[5] -
    prox.horizontal[6])
105
106     if calc < 30
107     and prox.horizontal[2] == 0 then
108       event.args[0] = FRONT
109     end
110
111   elseif event.args[0] == FRONT then
112     call leds.top(0,0,32)
113     motor.left.target = 200
114     motor.right.target = 200
115     timer = NO
116     RIGHT = NO
117     LEFT = NO
```

Figura 14 - Voltar atrás é basicamente o mesmo que virar à esquerda.

Aqui verifica-se o que acontece caso o robô tenha de escolher num entroncamento para que lado vira:

```
163
164         event.args[0] = TURN_LEFT
165      elseif  RIGHT == YES then
166         if help == 0
167         and event.args[1] != SUCCESS then
168            queue[count] = TURN_RIGHT count++
169            queue[count] = 1 count ++
170         elseif help > 0 then
171            help--
172         end
173
174         event.args[0] = TURN_RIGHT
175      else
176         aux = count-1
177         while queue[aux] == 1 do
178            help ++
179            queue[aux] = 0
180            aux = aux - 2
181         end
182         queue[aux-1] = TURN_RIGHT
183         event.args[0] = GO_BACK
184      end
185   elseif count < 199 then
186      while queue[count+1] == 0 do
187         count = count+2
188      end
189      count = count+2
190      event.args[0] = queue[count-2]
191   end
```

```
31   elseif event.args[0] == DECIDE then
32      if event.args[1] != SUCCESS then
33
34
35         call leds.top(0,0,32)
36         motor.left.target = 0
37         motor.right.target = 0
38         if prox.horizontal[1] >= 0
39         and prox.horizontal[2] >= 0
40         and prox.horizontal[3] >= 0 then
41            if  prox.horizontal[0] == 0
42            or prox.horizontal[1] == 0 then
43               LEFT = YES
44            elseif prox.horizontal[4] == 0
45            or prox.horizontal[3] == 0 then
46               RIGHT = YES
47            end
48         end
49
50         if LEFT == YES
51         and RIGHT == YES then
52            queue[count] = TURN_LEFT count++
53            queue[count] = 2 count ++
54            event.args[0] = TURN_LEFT
55         elseif LEFT == YES then
56            if help == 0
57            and event.args[1] != SUCCESS then
58               queue[count] = TURN_LEFT count++
59               queue[count] = 1 count ++
60            elseif help > 0 then
61               help --
62            end
63
```

Figura 15 - Conforme o valor do sensor 0 e 4 decide se vai para a esquerda ou direita.

Mais especficamente aqui é que se determina o estado que o robô deve realizar a seguir:

```
150         if LEFT == YES
151         and RIGHT == YES then
152            queue[count] = TURN_LEFT count++
153            queue[count] = 2 count ++
154            event.args[0] = TURN_LEFT
155         elseif LEFT == YES then
156            if help == 0
157            and event.args[1] != SUCCESS then
158               queue[count] = TURN_LEFT count++
159               queue[count] = 1 count ++
160            elseif help > 0 then
161               help --
162            end
163
164            event.args[0] = TURN_LEFT
165         elseif  RIGHT == YES then
166            if help == 0
167            and event.args[1] != SUCCESS then
168               queue[count] = TURN_RIGHT count++
169               queue[count] = 1 count ++
170            elseif help > 0 then
171               help--
172            end
173
174            event.args[0] = TURN_RIGHT
175         else
176            aux = count-1
177            while queue[aux] == 1 do
178               help ++
179               queue[aux] = 0
180               aux = aux - 2
181            end
182            queue[aux-1] = TURN_RIGHT
183            event.args[0] = GO_BACK
184         end
185      else
186         while queue[count+1] == 0 do
187            count = count+2
188         end
189         count = count+2
190         event.args[0] = queue[count-2]
191      end
192
```

Figura 16 - Sempre que possível, quando encontra um obstáculo vira à esquerda.

O robô cumpre o seu objetivo quando chega à área sinalizada no Labirinto e pára de se mover quando o sensor de proximidade do chão (*prox.ground.delta*) detecta mudança de cor:

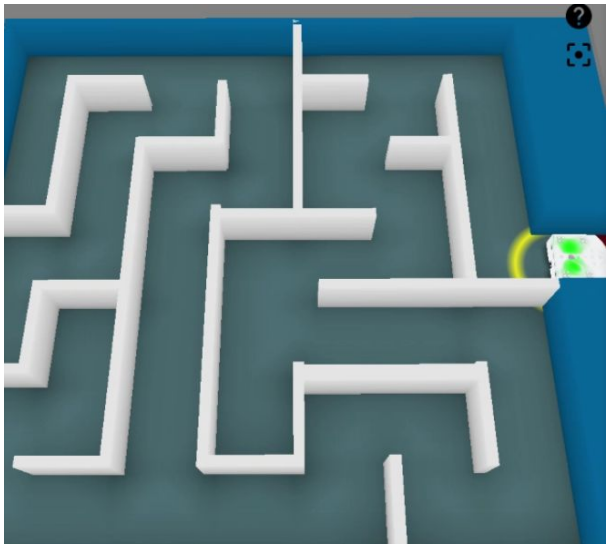

Figura 17 - Os LEDs acendem-se com a cor verde, indicando SUCESS

Sempre que o nosso Thymio se movimenta é registada a informação numa variável chamada *queue* que é um array que guarda em cada posição um número corresponde a um dos estados descritos acima:

Figura 18 - Exemplo de um caminho percorrido e os valores da *queue* e o seu significado.

Caso seja necessário voltar atrás pois chegou a um caminho sem saída (beco), então a informação registada desde a bifurcação é alterada, visto que essa informação não é válida para a descoberta do caminho válido. Isto é feito através de números na *queue* em que o 1 significa que

é a única opção, o 2 significa que existe 2 opções, a direita e a esquerda e o 0 são tentativas erradas.
Uma vez que o Thymio chega a um caminho sem saída ele inverte o valor da último opção com o valor 2, e coloca a 0 os decisões tomadas até chegar ao estado GO_BACK.

No final temos os movimentos guardados apenas localmente com as informações necessárias do melhor caminho, devido a correr em simulador, mas caso houvesse a oportunidade de se testar fisicamente poder-se-ia utilizar um cartão SD para guardar a informação e depois não seria necessário utilizar o nosso método de "tentativa e erro" pois o Thymio consegue ler instruções de um cartão SD[2].

Resultados:
Conseguimos obter o resultado esperado, apesar de poderem existir alguns erros e do ambiente em simulador. Embora a ideia original tenha sofrido algumas alterações, o Thymio segue o comportamento desejado.

Discussão:
No geral considerámos que o projeto correu bem. Apesar de apenas termos o simulador para testar o nosso projeto e desenhar o Labirinto deste modo, e de ainda existir pouca documentação sobre o que o Thymio permite e não permite fazer, conseguimos implementar de acordo com o *left hand algorithm* (tentar virar sempre à esquerda quando existe mais caminhos)[4].
Esta estratégia não permite a totalidade da exploração do mapa. Podem haver zonas inacessíveis como acontece no canto inferior esquerdo do nosso Labirinto (ver fig. 8).

Um dos fatores necessários é a velocidade do nosso robô que apesar de ser considerada "lenta", foi a que considerámos melhor de forma a evitar o erro. Quanto mais rápido se movesse mais difícil era coordenar os movimentos mais "apertados", devido a sensibilidade dos sensores.

Em termos de funcionamento do grupo, nada a apontar.
A carga de trabalhos notou-se na concretização final do projeto, visto que todos os projetos se acumularam para as mesmas datas de entrega. Considerámos o projeto acessível contudo o factor mencionado anteriormente não facilitou o desenvolvimento do mesmo, logo deveria haver mais comunicação entre unidades curriculares de modo a evitar este problema.
Tentámos implementar funcionalidades que algumas extensões forneciam, mas não eram compatíveis com o simulador com que estivemos a funcionar[3].

Para além disso, notámos que o simulador do ASEBA Studio depois de correr uma vez guarda sempre os valores nas variáveis da última vez que correu, por isso temos de ter cuidado para com esses valores.

Conclusão:
Apesar dos objetivos propostos terem sido atingidos, este projeto foi mais difícil que o esperado, e mesmo assim sentimos que poderíamos ter realizado algumas melhorias.
Não tivemos tempo de implementar a funcionalidade "inverter caminho".

Tivemos dificuldade principalmente com as medidas e valores corretos dos sensores devido a ser um processo de tentativa e erro demorado com o simulador, sentimos a falta de conhecimento dos instrumentos que temos, mas também devido a limitações que o simulador apresenta.

A criação do modelo ideal para um simulador não é fácil.
Existem várias componentes que não esperámos ter em conta, pois existem naturalmente no mundo real, que se revelaram mais difíceis de recriar no mundo digital.
Um dos principais problemas no desenvolvimento do Labirinto foi por vezes não termos em conta a espessura das paredes como parâmetro e se o corredor entre paredes se encontrava muito estreito ou muito largo dificultando a implementação do código (os sensores chegaram a não detectar nada e o robô circulava sempre à esquerda à procura).

No processo da descoberta do caminho, o factor decisão caso chegasse a um entroncamento também foi um fator do qual não pensámos ter problemas. Tivemos várias abordagens relativamente a este processo, mas todas as ideias que apresentávamos pareciam ter sempre vantagens e desvantagens, pelo que acabámos por optar pela estratégia de só mudar de direção quando for estritamente fundamental daí o uso de becos a meio do caminho não ter sido preciso e uma reestruturação do Labirinto ter sido necessária várias vezes (ver fig. 1-8).

Saber exatamente quanto é que o Thymio teria de rodar e como o fazer, foi um desafio.
Saber ajustar tanto o tempo de rotação como a velocidade das rodas em conjunto de modo a que em todas as situações a condição se provasse satisfeita foi algo que não esperámos ser tão difícil.

Poderíamos estender o nosso projeto fazendo com que o robô, ao verificar que o caminho é incorreto e tem de voltar atrás, sinalizasse com caneta um traço preto de modo a que esse caminho nunca mais fosse explorado por outro Thymio que procurasse ao mesmo tempo, fazendo uso tanto da caneta como do sensor de baixo.

O nosso projeto poderia ser aplicado na vida real como uma solução ao *Shortest path problem* se os veículos das estradas fossem automáticos e monitorizados como um Thymio num Labirinto, por exemplo, ou para a exploração de grutas instáveis que pudessem pôr em causa a integridade física dos seres humanos.

Referências:

1. Ideia original para o Labirinto: *thymio-maze-6x6.playground*, Disponível em: *https://github.com/aseba-community/aseba/blob/master/aseba/targets/playground/examples/thymio-maze-6x6.playground*
2. *Read and write data from the SD card*, Disponível em: http://wiki.thymio.org/en:thymioapi#toc12
3. *ScratchX extension*, Disponível em: https://mobsya.github.io/thymio-scratchx/index-en.html
4. *thymio robot solve changing maze with left hand algorithm*, Disponivel em: https://www.youtube.com/watch?v=gQTkqk50oro

Pesquisa:

1. *Robot Thymio Labyrinth*e, Disponível em: https://www.youtube.com/watch?v=459JZ3Di1IM
2. *Thymio II - Robot labyrinthe*, Disponível em: https://www.youtube.com/watch?v=yrO3WaGJIIA
3. *Thymio II - The Maze Runner*, Disponivel em: http://wiki.thymio.org/creations-fr:thymio-ii-the-maze-runner