Machine Learning Engineer Nanodegree	Victor Serpa Carvalho Nogueira 01 de Fevereiro, 2018

Navegação Robótica por Flood-Fill num Labirinto Virtual

Definition

Project Overview

Este projeto estudará a exploração e navegação de um robô através de um labirinto, utilizando *Flood-Fill*, com o objetivo de encontrar um caminho para o centro do labirinto o mais rápido possível.

Problem Statement

Serão utilizadas regras semelhantes à de competições de *micromouse*: uma rodada de treinamento, em que o robô explora livremente o labirinto e constrói um mapa interno, seguido de rodadas subsequentes em que o robô tenta chegar ao objetivo o mais rápido possível.

Um simulador provê ferramentas para o teste e verificação da inteligência que será programada. A lógica atua num mapa simplificado do mundo real e tem a performance avaliada através de pontuações dadas pelo simulador. O propósito do projeto é avaliar a efetividade da exploração de um labirinto utilizando *Flood-Fill* e criar uma lógica que encontre o centro de labirintos diversos o mais rápido possível, não apenas encontrar o caminho mais curto.

Metrics

A verificação da performance do robô é feita pelo simulador, através do seguinte cálculo:

$$score = tempo\ de\ solução + \frac{1}{30} * tempo\ de\ exploração$$

Como o objetivo é encontrar o menor tempo de solução possível, multiplica-se o tempo de exploração por um fator de treinamento, para encontrar uma relação apropriada entre treinamento e desempenho do robô.

O tempo de exploração é o número de passos dados na rodada de treinamento, utilizada pelo robô para conhecer o labirinto e testar rotas para a segunda rodada. O tempo de solução é o número de passos que o robô necessita para atingir o centro do labirinto após a exploração inicial, ou *fast run*. O limite máximo de tempo são mil passos para cada rodada.

Outro aspecto que será estudado é a taxa de exploração do labirinto, que indica a proporção de células visitadas pelo robô na rodada de treinamento. A taxa seguirá a seguinte fórmula:

$$exploração(\%) = \frac{número\ de\ células\ visitadas}{número\ total\ de\ células} * 100$$

Analysis

Data Exploration

A lógica atua num quadro simplificado dos labirintos encontrados em competições *micromouse*. São labirintos quadrados, de dimensões variadas, que tem início no canto inferior esquerdo e tem como objetivo as quatro células centrais do labirinto.

A célula de partida, localizada em (0,0), sempre "força" o movimento à frente, com paredes em ambos os lados. O robô interage com o labirinto através de três sensores, que possuem leituras sempre perfeitas, localizados à esquerda, à frente e à direita. As leituras são enviadas do simulador ao robô através de uma tupla (*tuple*), informando a distância do robô em relação às paredes nas direções dos sensores; na Figura 1, as leituras dos sensores, caso o robô estivesse apontando ao norte e na posição (0,0), seriam (0,11,0), indicando a distância das paredes à esquerda, à frente e à direita.

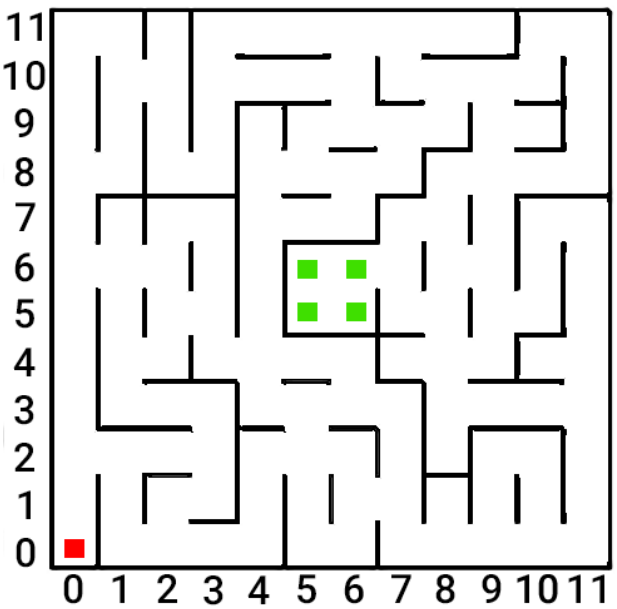

Figura 1- Início e chegada do labirinto 1.

Tal qual a leitura dos sensores, o movimento do robô é perfeito, caracterizado sempre por um valor de rotação e um valor de movimento. A rotação pode ter três valores inteiros distintos, -90 (anti-horário), 0 e 90 (horário). O deslocamento pode ter valores inteiros no intervalo de -3 e 3.

A cada passo da simulação, o robô define qual será a movimentação a ser feita. O movimento é bem-sucedido se o caminho não é bloqueado por nenhuma parede. No próximo passo, a simulação fornece novas leituras dos sensores para o cálculo do novo movimento.

Labirintos

O robô será testado em três labirintos distintos e de diferentes dimensões, com objetivo de testar o desempenho da lógica em desafios distintos.

A Figura 2 retrata o labirinto exemplo 2, que possui dimensão 14 por 14, e a solução ideal destacada. São 43 passos necessários que, seguindo as especificações da simulação, podem ser dados em 23 movimentos.

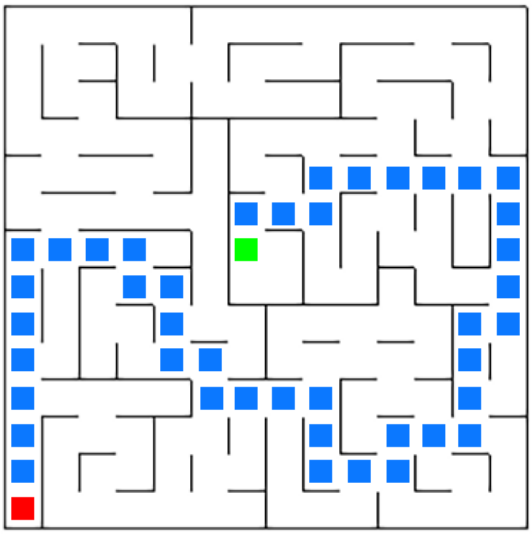

Figura 3 – Solução do labirinto 2.

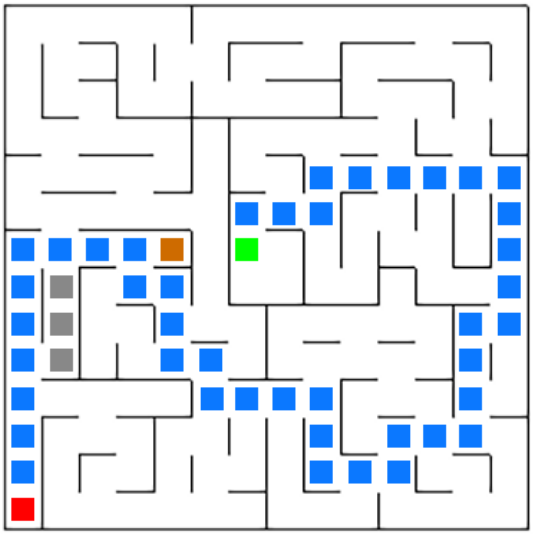

Figura 2 – Becos sem saída e rotações dispensáveis.

O robô deve, como representado na figura 2, evitar caminhos sem saída e dar preferência a caminhos com menos curvas, já que demandam deslocamentos menores mesmo se possuírem o mesmo número de passos. Na exploração, o robô deve evitar *loops* e dar menor preferência a caminhos passados anteriormente, pois gastam tempo desnecessário e não acrescentam novas informações ao mapa do robô.

Em labirintos maiores e complexos, como o labirinto exemplo 3, que possui dimensão 16x16, o tempo de exploração passa a ter uma importância maior, já que o tamanho do labirinto torna custoso visitar todas as células possíveis.

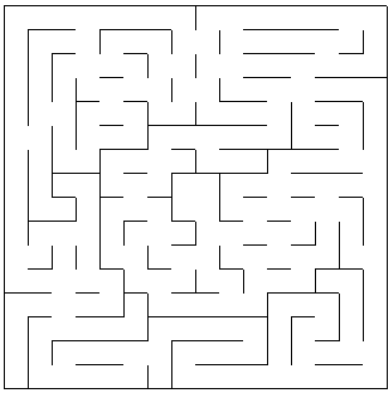

Figura 4- Labirinto 3(16x16).

Algorithms and Techniques

A lógica utilizada neste projeto será fundamentada no algoritmo *Flood-fill*, com modificações para atender as especificações do robô, como por exemplo a impossibilidade de "pular" entre células e movimentação limitada.

Flood-Fill é um algoritmo muito utilizado em competições *micromouse* e possui um bom desempenho em traçar rotas em caminhos desconhecidos. O algoritmo procura operar de maneira análoga a um líquido que flui de um ponto alto para um ponto mais baixo (Law, 2013). É atribuída a cada posição do labirinto um valor de distância que representa o quão distante ela está do destino, que possui um valor 0. Portanto, células com altos valores estão mais distantes do destino – em lugares elevados, segundo a analogia – e células de valores baixos estão mais próximas do destino, ou seja, baixa elevação.

Conforme a movimentação do robô, o mapa é atualizado e distâncias são modificadas de acordo com as possibilidades de movimentação. A Figura 5 representa a distribuição de valores de distância dentro de um labirinto com dimensão 6x6, com valores calculados pela distância Manhattan entre a célula e a sala de destino do labirinto.

4	3	2	2	3	4
3	2	1	1	2	3
2	1	0	0	1	2
2	1	0	0	1	2
3	2	1	1	2	3
4	3	2	2	3	4

Figura 5 - Valores de distância num labirinto 6x6

A cada passo dado, o algoritmo avalia qual é a célula adjacente com menor valor de distância e o robô calculará qual são os movimentos necessários para atingir a célula. O algoritmo atualiza os valores de distância quando há modificação no mapa interno do robô e quando não encontra um movimento apropriado, forçando uma reavaliação do próximo movimento.

Como algoritmo simples de exploração, *Flood-fill* funciona de uma maneira eficiente para encontrar o destino da maneira rápida sem conhecimento de obstáculos, porém não garante que o caminho encontrado é o mais rápido possível, nem visita todas as possibilidades. Para a rodada de exploração, serão testadas duas técnicas além do algoritmo comum.

A primeira delas é a introdução de movimentos aleatórios para forçar o robô a visitar caminhos diferentes e explorar uma maior parcela do labirinto. Espera-se assim ter um mapa mais completo para a segunda corrida e um desempenho melhor do robô. A segunda técnica é forçar o robô a voltar ao início, visitando células diferentes e, assim, possibilitar uma melhor inspeção do labirinto.

Benchmark

O benchmark será a pontuação do algoritmo padrão, além dos números de passos, para estudar o efeito da exploração em comparação ao resultado da rodada final.

Os resultados da segunda rodada serão comparados com os valores das soluções ideais de cada labirinto.

Methodology

Data Preprocessing

Neste projeto não é necessário o pré-processamento dos dados pois as especificações, movimentos e leituras dos sensores são totalmente acurados, tornando todos os dados captados, relevantes e precisos.

Implementation

Toda a implementação, do cálculo do melhor movimento a tarefas necessárias para o funcionamento do algoritmo é feita em robot.py.

As tarefas do robô podem ser divididas em duas: planejamento e realização. O planejamento recebe os valores de sensores, atualiza as matrizes de mapa e distâncias, além de calcular a posição do robô e o próximo passo a ser dado. A realização utiliza as informações calculadas e providas pela fase de planejamento e calcula a rotação e deslocamento necessário para cumprir a tarefa.

Uma das soluções interessantes encontradas no início do processo de construção do algoritmo do robô foi o cálculo da posição. Enquanto a localização de linha e coluna é algo trivial, a posição do robô (norte, sul, leste ou oeste) apresenta características específicas e que simplificam alguma das tarefas necessárias. Na função *updateMap*, que atualiza o *grid* com informações de paredes e visitação, a informação de posição é utilizada para definir onde estão os obstáculos.

```
def updateMap(self, sensors):
    way = [-90, 0, 90]
    #Atualização da informação de visitação
    self.grid[self.y][self.x][4] = 1
    for i in range(len(sensors)):
        direction = (self.angle + way[i]) % 360
        distance = [sensors[i] * d for d in self.convertAngle(direction)]
        self.grid[self.y + distance[1]][self.x +
                                        distance[0]][self.gridConvert(direction)] = 0
```

As funções *convertAngle* e *gridConvert*, vistas no fragmento acima, utilizam a informação de ângulo para cumprir tarefas distintas. Em *convertAngle*, a posição é transformada em uma informação de linha e coluna (x,y).

```
def convertAngle(self, angle):
    if angle == 0:  # Up
        return [0, 1]
    elif angle == 90:  # Right
        return [1, 0]
    elif angle == 270:  # Left
        return [-1, 0]
    elif angle == 180:  # Down
        return [0, -1]
    else:
        return [0, 0]
```

Já em *gridConvert*, a mesma informação é utilizada para definir qual informação deve ser preenchida no *grid*.

```
def gridConvert(self, move):
    if move == 0:  # Up
        return 0
    elif move == 90:  # Right
        return 1
    elif move == 180:  # Down
        return 2
    elif move == 270:  # Left
        return 3
```

Um dos maiores desafios do algoritmo escolhido é a implementação da atualização dos valores de distância. Na função *checkBest*, que escolhe o próximo movimento a ser feito, a verificação simples falha quando o robô se encontra numa célula em que todos os vizinhos possuem valores maiores de distância do que a célula presente. Para atualizar os valores de distância, seguindo o algoritmo *Flood-Fill*, é chamada a função *updateDist*, que estuda as células e movimentos possíveis adjacentes, atribuindo novos valores de distância.

O mapa do labirinto é guardado na lista *grid*, que possui 5 informações para posição do labirinto. As primeiras quatro posições armazenam informações de bloqueio por uma parede nas posições norte, leste, sul e oeste, enquanto a quinta posição armazena informações de visita da célula.

A lista *dist* possui a mesma dimensão da lista *grid* e inicialmente armazena valores de distância seguindo uma fórmula heurística. Após algumas iterações de movimentação, os valores são atualizados, como comentado acima.

A principal diferença entre o algoritmo *Flood-Fill* comum e o algoritmo utilizado pelo robô é a impossibilidade de "saltar" entre posições, ou seja, caso seja necessário voltar por um caminho ou visitar uma célula distante, o robô precisa planejar os

movimentos. Um fragmento de código utilizado para adaptar o algoritmo está na função *checkBest*:

```
if best == []:
        back = (self.angle+180)%360
        nxny = self.convertAngle(back)
        if self.grid[self.y][self.x][self.gridConvert(back)] == 1:
            if self.dist[self.y + nxny[1]][self.x + nxny[0]] < dist_value:
                best = [self.angle,-1]
```

Enquanto a função *checkBest* normalmente só analisaria movimentos possíveis, adicionar o código acima dá a possibilidade para robô verificar movimentos de retorno e continuar a exploração do labirinto.

Apesar de existirem algumas diferenças na fase de exploração, todas elas utilizam o mesmo algoritmo, variando apenas em sua aplicação. O maior desafio neste estágio do projeto foi a implementação da exploração estendida, que deve inverter valores de distância sem perder informações obtidas pelo robô.

Os resultados serão visualizados através de showresult.py que passa a mostrar, além de uma imagem do mapa, o resultado da última simulação obtida em *tester.txt*. O arquivo *tester.txt*, gerado pela simulação, contém todos os passos dados pelo robô e em que rodada cada um deles foi feito. O número de passos será impresso após o fim da simulação.

Refinement

Exploração

Todas as aplicações do algoritmo estão na classe robot.py. Os modos de exploração serão três.

- Aleatório: além de *Flood-Fill*, dá passos aleatórios numa proporção decrescente seguindo a fórmula:

$$\varepsilon = e^{-\varphi * t}$$

Para o qual $\boldsymbol{\varphi}$ é a fator de convergência e $\boldsymbol{t}$ é o passo atual do robô. Quanto maior o fator de convergência, mais rápido épsilon se aproximará do zero, diminuindo a taxa de aleatoriedade.

- Comum: utiliza *Flood-Fill* nas duas rodadas sem modificações;

- Estendido: além de atingir o destino na fase de exploração, força o robô a voltar ao início.

Na rodada de *fast run*, o *Flood-Fill* será utilizado da mesma maneira, independentemente do modo escolhido para a exploração inicial do labirinto.

Simulação

A simulação é iniciada da seguinte maneira:

tester.py <nome do labirinto> <número do modo exploração>

Por exemplo:

tester.py test_maze_01.txt 1

Para a visualização dos caminho tomado após a execução da simulação, utiliza-se:

showresult.py <nome do labirinto>

O script *showresult* sempre mostrará o resultado do último teste feito no labirinto escolhido. Caso não exista, mostrará apenas o labirinto, análogo à *showmaze*.

O número dos modos de exploração disponíveis são:

[1] Aleatório

[2] Comum

[3] Estendido

Results

Model Evaluation and Validation

As soluções ideais, calculadas utilizando A* (Shibuya, 2016) e total conhecimento de cada um dos labirintos, são:

TABELA 1 – NÚMERO DE PASSOS NECESSÁRIOS PARA SOLUÇÃO DOS LABIRINTOS

Labirinto teste	Movimentos necessários
01	17
02	23
03	25

Para a fase de exploração, utilizando *Flood-Fill*, os movimentos necessários para alcançar o centro foram:

TABELA 2 – NÚMERO DE PASSOS NECESSÁRIOS PARA SOLUÇÃO DO LABIRINTO UTILIZANDO FLOOD-FILL

Labirinto teste	Movimentos necessários
01	34
02	60
03	67

Além dos resultados de passos dados na rodada de treinamento de *fast run*,

Exploração Aleatória

A exploração aleatória é configurada de maneira que o robô escolha um movimento aleatório, ao invés de seguir o resultado do algoritmo *Flood-Fill*, numa taxa decrescente de aleatoriedade. A chance de o movimento ser aleatório segue a fórmula descrita na seção de exploração.

Em razão de ser um método que utiliza um fator aleatório e produz resultados diferentes, foram feitas dez corridas utilizando configurações diversas, para cada um dos labirintos.

TABELA 3 – RESULTADOS MÉDIOS PARA MAZE 3

φ	Score	Treino	Fast Run	Exploração(%)
0,01	27896,8	74,9	25,4	41,736
0,05	28690,1	47,7	27,1	29,095
0,1	29243,4	46,3	27,7	27,43

TABELA 4 – RESULTADOS MÉDIOS PARA MAZE 2

φ	Score	Treino	Fast Run	Exploração(%)
0,01	42913,3	114,4	39,1	40,411
0,05	39373,3	83,2	36,6	35,358
0,1	38276,7	74,3	35,8	34,184

TABELA 5 – RESULTADOS MÉDIOS PARA MAZE 3

φ	Score	Treino	Fast Run	Exploração(%)
0,01	57970	95,1	54,8	27,931
0,05	59723,3	72,7	57	22,973
0,1	58620	69,6	57,3	23,205

TABELA 6 – RESULTADOS MÉDIOS PARA MAZE 4

φ	Score	Treino	Fast Run	Exploração(%)
0,01	77889,9	173,7	72,1	41,681
0,05	78083,3	125,5	73,9	38,046
0,1	78626,8	114,8	74,8	37,227

TABELA 7 – RESULTADOS PARA MAZE 1

φ	Score	Treino	Fast Run	Exploração(%)
0,1	33767	53	32	33,33
0,1	29833	55	28	28,47
0,1	27567	47	26	30,56
0,1	29200	66	27	31,94
0,1	27367	41	26	27,08
0,1	29400	42	28	25
0,1	28267	38	27	24,31
0,1	28300	39	27	24,31
0,1	29133	34	28	22,22
0,1	29600	48	28	27,08
0,05	26833	55	25	32,64
0,05	29267	38	28	23,61
0,05	32500	45	31	27,08
0,05	20967	59	19	31,94
0,05	29367	41	28	25,69
0,05	30533	46	29	27,08
0,05	28467	44	27	27,08
0,05	29000	60	27	36,11
0,05	27667	50	26	33,33
0,05	32300	39	31	26,39
0,01	22467	104	19	50,69
0,01	24667	80	22	51,39
0,01	29333	100	26	52,78
0,01	28267	68	26	43,75
0,01	33933	28	33	18,75
0,01	27167	95	24	41,67
0,01	27867	86	25	45,83
0,01	31400	42	30	27,78
0,01	26800	84	24	44,44
0,01	27067	62	25	40,28

TABELA 8 – RESULTADOS PARA MAZE 2

φ	Score	Treino	Fast Run	Exploração(%)
0,1	38333	70	36	31,63
0,1	39300	69	37	31,12
0,1	39467	74	37	47,45
0,1	32033	91	29	38,78
0,1	39267	68	37	30,1
0,1	46000	60	44	26,02
0,1	35867	86	33	37,76
0,1	38100	63	36	28,57
0,1	39333	70	37	31,12
0,1	35067	92	32	39,29
0,05	38400	72	36	32,65
0,05	45767	83	43	37,76
0,05	38933	88	36	37,24
0,05	34133	94	31	41,33
0,05	35933	88	33	38,27
0,05	36167	95	33	38,27
0,05	39500	75	37	33,16
0,05	45400	72	43	26,02
0,05	40067	92	37	37,76
0,05	39433	73	37	31,12
0,01	32800	114	29	43,37
0,01	46600	138	42	44,39
0,01	37200	126	33	43,37
0,01	37967	119	34	38,78
0,01	45300	99	42	39,29
0,01	52000	120	48	43,37
0,01	38533	136	34	48,47
0,01	47133	124	43	39,29
0,01	49367	41	48	19,9
0,01	42233	127	38	43,88

TABELA 9 – RESULTADOS PARA MAZE 3

φ	Score	Treino	Fast Run	Exploração(%)
0,1	56367	71	54	21,88
0,1	57300	69	55	23,83
0,1	56500	75	54	24,22
0,1	56533	76	54	23,83
0,1	60267	68	58	23,44
0,1	59133	64	57	22,66
0,1	59400	72	57	23,83
0,1	62200	66	60	21,09
0,1	58233	67	56	23,83
0,1	60267	68	68	23,44
0,05	68933	58	67	19,59
0,05	57700	81	55	23,05
0,05	56900	87	54	24,61
0,05	58300	69	53	23,83
0,05	62567	77	60	24,22
0,05	56400	72	54	23,44
0,05	60867	56	59	18,75
0,05	65133	64	63	21,88
0,05	56533	76	54	24,61
0,05	53900	87	51	25,75
0,01	60633	109	57	30,08
0,01	65167	65	63	22,66
0,01	67333	70	65	23,83
0,01	56500	75	54	24,22
0,01	52167	95	49	26,56
0,01	59467	44	58	14,45
0,01	62367	101	59	26,56
0,01	61333	130	57	34,38
0,01	60533	106	57	28,52
0,01	34200	156	29	48,05

TABELA 10 – RESULTADOS PARA MAZE 4

φ	Score	Treino	Fast Run	Exploração(%)
0,1	80800	114	77	37,11
0,1	78067	122	74	37,5
0,1	80700	111	77	37,11
0,1	80767	113	77	37,11
0,1	73867	116	70	37,5
0,1	75967	119	72	37,89
0,1	78767	113	75	36,72
0,1	80700	111	77	36,72
0,1	80800	114	77	36,72
0,1	75833	115	72	37,89
0,05	79200	126	75	36,72
0,05	75100	123	71	38,67
0,05	81267	128	77	37,5
0,05	79200	126	75	38,67
0,05	81033	121	77	37,11
0,05	76067	122	72	38,28
0,05	71233	127	67	39,84
0,05	79233	127	75	37,11
0,05	79367	131	75	37,89
0,05	79133	124	75	38,67
0,01	74633	199	68	41,8
0,01	73733	202	67	41,02
0,01	75533	196	69	42,97
0,01	81200	156	76	37,5
0,01	78300	159	73	40,23
0,01	89100	153	84	46,88
0,01	70333	160	65	42,58
0,01	67567	167	62	41,8
0,01	83700	171	78	37,89
0,01	84800	174	79	44,14

Os resultados alcançados, o modo de exploração aleatória, utilizaram, como esperado, um maior número de passos para encontrar o centro na fase de treinamento, o que normalmente indica uma maior exploração do labirinto. A taxa de aleatoriedade também funcionou como esperado, rendendo uma maior exploração quando o fator de convergência é menor. Outro ponto relevante é a aproximação da solução ideal em algumas corridas, diferente do que será visto nos próximos modos de exploração.

Flood-Fill Comum

Diferentemente do algoritmo de exploração aleatória, o resultado do algoritmo *Flood-Fill* se mantém constante. O desempenho do algoritmo está na tabela abaixo:

TABELA 11 – RESULTADOS FLOOD FILL COMUM

Labirinto	Score	Treino	Fast Run	Exploração(%)
1	29133	34	28	20,83
2	46000	60	44	26,02
3	60233	67	58	23,05
4	80667	110	77	36,72

Flood-Fill Estendido

O último método de exploração utiliza em média pouco menos do dobro de passos do método comum de *Flood-Fill* na rodada de exploração. Entretanto, nota-se que o dobro de passos não se traduz no dobro de exploração de células.

TABELA 12 – RESULTADOS FLOOD FILL ESTENDIDO

Labirinto	Score	Treino	Fast Run	Exploração(%)
1	27867	56	26	34,03
2	39200	96	36	42,86
3	77233	127	73	35,94
4	87800	294	78	67,97

Exploração, eficiência e robustez

Os resultados encontrados nos diferentes modos de exploração deixam claro que há correlação entre exploração e eficiência, o que é de se esperar. Entretanto, contrariando a expectativa, observando apenas os números de passos dados na segunda rodada, nota-se que o ganho de exploração representa apenas um pequeno ganho no desempenho, especialmente em labirintos mais complexos, como o três e quatro.

O algoritmo mostrou-se robusto nas condições apresentadas, encontrando o objetivo em todos os casos. O pior desempenho, encontrado no labirinto desenvolvido separadamente e escolhido para atacar fraquezas do algoritmo, ainda está distante do tempo limite fixado pelo simulador.

Justification

O *Flood-Fill* não se aproximou da solução ideal em nenhum dos casos, o que era esperado levando em consideração que o ponto forte do algoritmo é a exploração sem um mapa prévio. Na rodada de treinamento, tanto o modo aleatório quanto o estendido tiveram uma busca mais extensa que o algoritmo comum.

Na tabela seguinte serão comparados os resultados dos três modos nos labirintos disponíveis e benchmarks, utilizando o melhor resultado médio do modo aleatório em cada caso. Para comparar a eficiência dos modos na *fast run*, será utilizada a fórmula:

$$\mathbf{\Delta FR}(\%) = \frac{tempo\ fast\ run - benchmark}{benchmark} * 100$$

TABELA 13 – COMPARAÇÃO DE RESULTADOS MAZE 1

Modo	Score	Treino	Fast Run	Exploração(%)	ΔFR(%)
Comum	29133	34	28	20,83	64,71
Estendido	27867	56	26	34,03	52,94
Aleatório, φ = 0,01	27896,8	74,9	25	41,74	47,06
Benchmark	-	34	17	-	-

TABELA 14 – COMPARAÇÃO DE RESULTADOS MAZE 2

Modo	Score	Treino	Fast Run	Exploração(%)	ΔFR(%)
Comum	46000	60	44	26,02	91,30
Estendido	39200	96	36	42,86	56,52
Aleatório, φ = 0,1	38276,7	74,3	36	34,19	56,52
Benchmark	-	60	23	-	-

TABELA 15 – COMPARAÇÃO DE RESULTADOS MAZE 3

Modo	Score	Treino	Fast Run	Exploração(%)	ΔFR(%)
Comum	60233	67	58	23,05	132,0
Estendido	77233	127	73	35,94	192,0
Aleatório, φ = 0,01	57970	95,1	54,8	27,931	119,2
Benchmark	-	67	25	-	-

Nos resultados apresentados, há a sugestão que uma maior exploração é efetiva em labirintos mais simples, como observado na tabela 13, onde a exploração e ΔFR são inversamente correlacionados. Entretanto, com o aumento da complexidade dos labirintos essa relação não se mantém e uma menor exploração tem resultados melhores, como observado na tabela 15.

O desempenho do algoritmo é consistente através de todos os labirintos. Enquanto maior informação do labirinto altera a eficiência do *Flood-Fill*, o algoritmo

possui velocidade de resolução semelhante nas duas rodadas, comprovando a utilidade da inteligência em cenários de exploração.

Conclusion

Free-Form Visualization

Utilizando como inspiração o labirinto da 13ª Competição de Micromouse e Robôs Inteligentes de Taiwan (Ye, 2017), o labirinto tem como proposta atacar fraquezas comuns de robôs *micromouse*.

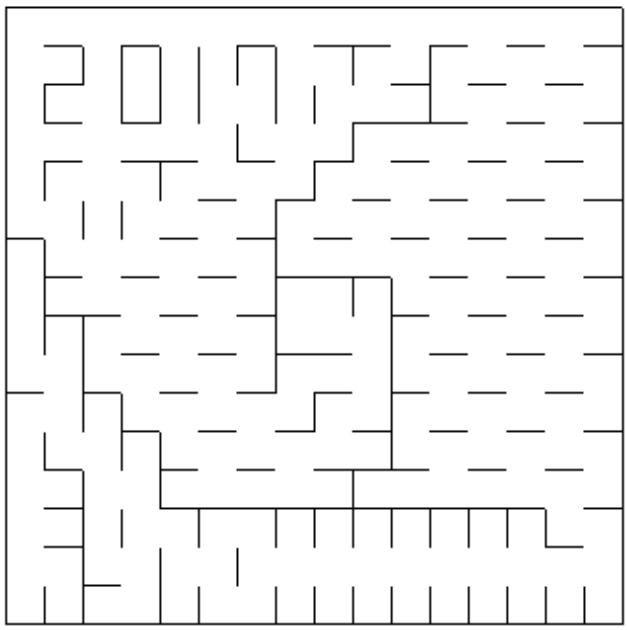

Figura 6- Test Maze 4

No mundo real, o labirinto procura testar a habilidade do robô andar em diagonal, como pode ser visto na solução ideal (Ye, 2017):

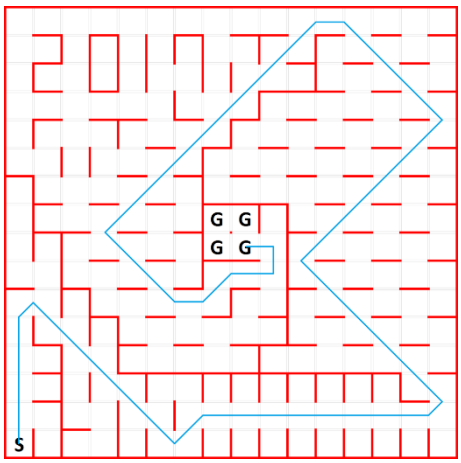

Figura 7- Solução ideal

Como o simulador limita os movimentos possíveis do robô, os passos necessários para atingir o objetivo são elevados, pois é necessário desviar de todas as barreiras. Os resultados atingidos pelo algoritmo são explicitados em seções anteriores e, abaixo, segue a solução atingida pelo algoritmo com exploração comum:

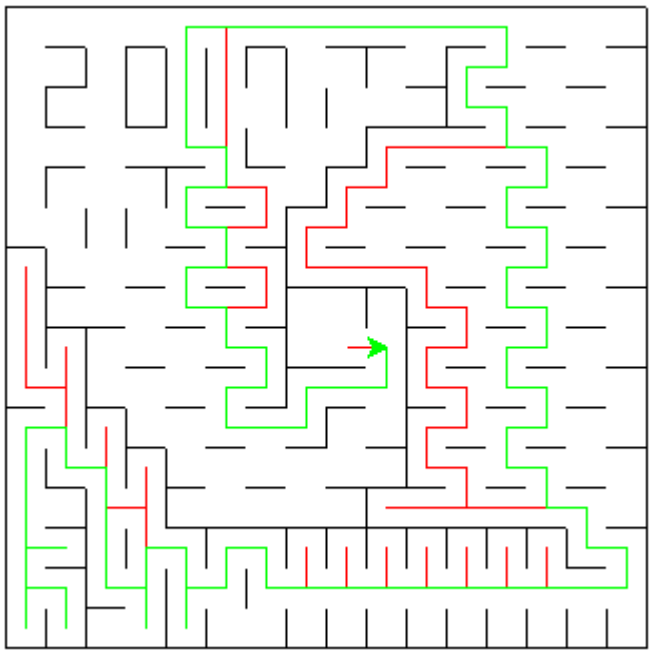

Figura 8- Solução encontrada utilizando exploração comum

Reflection

O primeiro desafio que encontrei no início do projeto foi identificar um método diferente para o projeto. Enquanto a solução por A* é interessante e viável, ela já foi discutida com detalhes no projeto exemplo. A pesquisa de um método alternativo me levou ao estudo de competições de *micromouse*; enquanto robôs campeões raramente detalham qual é o método utilizado, há diversos estudos detalhando métodos novos de *Flood-Fill* (Willardson, 2001) (Cai, Ye, & Yang, 2012) utilizando regras de competições como método de verificação da performance do algoritmo.

O próximo desafio foi a programação de fato. Apesar do algoritmo ser relativamente simples, as limitações da movimentação do robô introduzem alguma dificuldade na implementação. Minha estratégia foi sofisticar a resolução em níveis; primeiramente o robô apenas se movia uma célula por vez e só para frente. Dessa maneira, consegui sanar boa parte dos problemas como becos sem saída e loops ainda na primeira iteração, já que as opções eram reduzidas.

O segundo passo foi introduzir movimentos maiores, melhorando a velocidade do robô e os scores. Não houve grandes mudanças no tempo computacional para

resolução, apesar do robô analisar um maior número de possibilidades para cada célula – de 3 para 9. O passo final foi introduzir o movimento reverso, que introduziu uma melhora na maneira de lidar com becos.

Nesse estágio do projeto, o funcionamento do robô já possibilitava a medição de resultados. Os resultados do *fast run*, no entanto, quando comparado ao benchmark, deixaram a desejar. Outros modos para estender a busca foram introduzidos, mas o ganho foi reduzido, indicando que o problema não é necessariamente pouco conhecimento do labirinto, mas a maneira que os movimentos são decididos.

Sendo assim, o *Flood-Fill* é indicado para resolver problemas de localização de característica exploratória, encontrando o destino quando há pouca informação disponível. Entretanto, quando utilizado para encontrar o caminho eficiente, o algoritmo é muito agressivo e escolhe movimentos que são bons na iteração presente, mas que tomarão mais movimentos nos passos seguintes.

Improvement

A maior limitação do algoritmo é funcionar num mundo discreto e perfeito. A utilização e os cálculos são simples num mundo de centenas de células, mas no mundo real, haveria dificuldades com o posicionamento, movimentação e dimensões do robô, além da informação captada pelos sensores e o armazenamento do mapa. Há técnicas existentes para solução, como SLAM, que podem ser aplicadas para estes casos.

No caso do robô e as paredes terem espessura, seria necessário definir uma célula mínima de comprimento e reconstruir o labirinto sob as novas regras. Além disso, seria necessário levar em consideração o tamanho do robô e o espaço necessários para manobras.

Outro desafio, em relação às competições de *micromouse*, é a solução de testes recorrentes nos mapas. Andar na diagonal e evitar curvas fechadas é essencial para economizar tempo, um ponto que foi só tocado superficialmente neste projeto. O desempenho do algoritmo atual foi testado num labirinto de competição real – o Test Maze 4 – e o número de passos para o objetivo foi alto quando comparado com os testes anteriores.

Utilizando as configurações de mundo deste projeto, as soluções dos labirintos são as soluções ideais encontradas através de A* quando o labirinto foi totalmente explorado e não há outro benchmark possível. Um ponto mais interessante de estudo, que desde o início foi um dos objetivos do projeto, é a otimização da exploração necessária para a construção de tal resposta ideal. Neste campo, a performance do *Flood-Fill* é satisfatória, mas não carrega informações que podem ser úteis em labirintos diversos, como configurações já vistas anteriormente e movimentos preferidos.

Referências

Cai, Z., Ye, L., & Yang, A. (2012). FloodFill Maze Solving with Expected Toll of Penetrating Unknown Walls for Micromouse. *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems .* Chengdun: IEEE.

Law, G. (Junho de 2013). Quantitative Comparison of Flood Fill and Modified Flood. *International Journal of Computer Theory and Engineering*, pp. 503-508.

Shibuya, N. (20 de March de 2016). *Plot and Navigate a Virtual Maze.* Fonte: Udacity: http://www.udacity.com

Willardson, D. M. (2001). *Learning from Data*. Fonte: Analysis of Micromouse Maze Solving Algorithms: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.307.3305&rep=rep1&type=pdf

Ye, G. (25 de 09 de 2017). *2017 Taiwan micromouse and intelligent robot contest*. Fonte: Micromouse USA: http://micromouseusa.com/?p=2179