Instituto Tecnológico de Aeronáutica

Divisão de Engenharia de Computação

CTC-17: Inteligência Artificial

Projeto I - Buscas

Aluno: Luiz Angel R. Rafael

Profº. responsável:
Paulo André Castro

08 de Setembro de 2016

1 Introdução e objetivos

O projeto I consiste em implementação de algoritmos de busca Greedy e A*, com suas respectivas heurísticas, e de agentes que joguem *Tic Tac Toe* contra um humano, utilizando os algoritmos de minimax (básico) e a poda alpha-beta.

O projeto foi desenvolvido em linguagem Python 2.7 [1] no ambiente Windows 10, havendo controle de versão deste projeto com a ferramenta Git.

2 Descrição

Na pasta do projeto, encontra-se a seguinte composição:

```
├── inputs
│   └── Uruguay.csv
├── source
│   ├── search
│   │   ├── AStar.py
│   │   ├── Greedy.py
│   │   ├── Helper.py
│   │   ├── MainSearch.py
│   │   └── Node.py
│   └── tictactoe
│       ├── Helper.py
│       ├── MainAlphaBeta.py
│       ├── MainMinimax.py
│       ├── NodeTree.py
│       └── Tree.py
├── README.rst
├── requirements.txt
└── setup.py
```

O arquivo principal são aqueles cujos nomes começam com Main. Os arquivos de classes (que especificam uma classe em Python) são todos exceto os principais e os arquivos Helper.py. Estes são arquivos com funções auxiliares utilizadas por arquivos principais e/ou de classe.

Para obter resultado de algum algoritmo, basta executar python <NomeArquivoPrincial>. Por exemplo, para os algoritmos de busca:

```
> python MainSearch.py
```

2.1 Algoritmos Greedy e A*

Para ambos os algoritmos, foi utilizada a classe Node, que guarda informações de cada cidade listada no arquivo de entrada Uruguay.csv: o índice da cidade coordenadas x e y e os vizinhos. Para o Greedy, há um campo da cidade que guarda a informação se ela está marcada ou não (o

quê é necessário para marcar uma cidade será explicado logo a seguir). Para o A*, há um campo que serve para guardar o custo de uma cidade (esse campo também será explicado mais adiante). Com a utilização de classes em vez de várias listas ou maps, o código fica mais claro e organizado. Para o Greedy, decidi não utilizar fila de prioridades, e decidi utilizá-la apenas no A*, atentando para uma maior simplicidade na estrutura de dados do Greedy.

Para o Greedy, foram implementados os seguintes passos:

- Se a cidade atual (no começo é 202) é diferente da cidade destino (601):
  - Para o cidade inicial atual, pegar a lista ordenada de cidades vizinhas pela distância de cada uma delas até a cidade final (601);
  - Para cada cidade da lista acima, verificar se ela ainda não foi percorrida:
    - Se sim, recomeçar o algoritmo a partir desta cidade;
  - Se todas as cidades vizinhas já foram percorridas, deve-se marcar essa cidade (para não ser percorrida novamente), voltar para a cidade percorrida anteriormente à atual, e continuar a partir dela.

Ressaltando que foi feito esse algoritmo para o Greedy porque seria a forma mais básica de implementá-lo com a estrutura de dados utilizada (não foi necessária lista de prioridades, por exemplo, diferentemente do A*).

Para o A*, foram implementados os seguintes passos:

- Adicionar a cidade inicial (202) à fila de prioridade;
- Enquanto a fila de prioridade não estiver vazia:
  - Dar pop na fila de prioridade adicionar essa cidade (C) na lista de cidades que já foram retiradas da fila de prioridade;
  - Verificar se C é a cidade destino (601):
    - Se sim, sair do laço (o algoritmo acabou);
  - Para cada cidade vizinha de C:
    - Se ela já foi percorrida (ou seja, já foi retirada da fila de prioridades), ir para a próxima cidade vizinha de C;
    - Se não, calcular a nova estimativa de custo daquela cidade, que seria o custo da cidade C somado à distância em linha reta da cidade até o destino. Também, deve-se atualizar a melhor estimativa para a cidade caso ela já esteja na fila de prioridade. Adicionar essa cidade à fila de prioridades caso ela não esteja lá.

Foi utilizada a mesma heurística básica do Greedy (distância em linha reta de uma cidade ao destino), porém com o refinamento pela implementação do custo do nó, que serviu para evitar expandir caminhos com custos caros.

2.2 Algoritmos Minimax e Poda alpha-beta

A estrutura de dado utilizada para os dois algoritmos foi a mesma. Há a classe Tree e a classe NodeTree, que são, respectivamente, a árvore de possibilidades e um nó dessa árvore. A árvore de possibilidades guarda o nó atual e, ao ser instanciada, a partir da primeira jogada do humano, cria todos os nós possíveis da árvore, marcando os nós que são folhas e atribuindo-lhes seus respectivos pontos (1 para folha que representa vitória da máquina, 0 para empate e -1 para derrota). Cada nó sabe quem é o seu pai e quem são seus filhos.

A partir de cada jogada do humano, a máquina realiza sempre um movimento que vai resultar, ao final do jogo, em vitória ou empate, já que ambos os algoritmos possuem solução ótima. O que os diferencia é apenas como a árvore de possibilidades vai ser percorrida:

- Para o Minimax, todos os nós, a partir da folha atual, são percorridos todos os nós até as folhas;
- Para a Poda Alpha-Beta, apenas os caminhos necessários para minimizar ou maximizar a pontuação de determinado nó (depende do caso) são percorridos, economizando-se tempo de execução.

3 Resultados

3.1 Algoritmos de Busca

Ao executar o arquivo MainSearch.py, é impresso os menores caminhos com seus respectivos custos para o Greedy e o A*:

```
*****************Greedy Algorithm (basic)*****************
Total cost of the path: 258.403500807
Path followed:
202 199 197 196 195 194 192 191 186 181 182 184 189 187 188 193 198 203 205 207 209 208 210
212 213 215 220 225 229 234 237 236 231 230 227 232 235 239 242 246 245 248 253 256 257 262
265 270 269 272 271 276 277 280 283 281 282 287 289 295 294 296 300 305 301 306 310 312 317
321 325 331 329 326 322 327 330 335 340 338 341 342 343 345 346 348 353 355 352 351 356 357
359 364 365 371 374 375 372 370 368 373 378 380 381 384 387 392 390 386 383 388 393 394 395
391 396 398 401 402 405 409 412 411 413 419 422 424 426 431 436 441 438 439 443 445 446 451
454 453 458 462 465 460 459 463 468 472 475 476 470 466 467 469 473 478 483 485 490 494 499
500 504 508 512 509 511 513 517 519 523 526 527 530 535 539 537 542 536 541 544 548 551 555
558 559 562 557 561 563 567 564 568 573 576 580 584 581 579 575 571 574 578 582 588 591 585
590 586 587 589 592 594 599 601

*****************A Star Algorithm*****************
Total cost of the path: 93.2503538698
Path followed:
202 206 211 217 214 218 223 225 230 232 235 239 241 244 246 248 253 257 262 265 268 272 277
280 283 288 293 297 302 307 308 312 317 321 325 330 335 339 342 344 349 354 359 364 365 371
373 378 382 383 388 393 396 398 400 404 408 411 413 419 422 424 426 431 432 435 438 439 442
447 451 454 458 461 466 470 476 478 483 485 490 494 499 500 504 508 513 515 519 523 526 527
530 535 537 542 546 549 554 556 560 563 561 566 569 574 577 581 584 586 590 595 601
```

Percebe-se uma eficiência (em termos de diminuição do custo do percurso) de 2.77, aproximadamente, pelo caminho percorrido com o A* em comparação pelo percorrido com o Greedy.

3.2 Algoritmos para o Tic Tac Toe

Como relatado anteriormente, o humano nunca consegue vencer; ele apenas empata ou perde, pois o algoritmo possui solução ótima. Abaixo há 2 exemplos: um em que o humano empata e, no outro, ele perde.

```
Human player's turn (type the x, space and the y, followed by enter):
---> 0 0

 X |   |
-----------
   |   |
-----------
   |   |

Computer's turn:

 X |   |
-----------
   | O |
-----------
   |   |

Human player's turn (type the x, space and the y, followed by enter):
---> 1 0

 X |   |
-----------
 X | O |
-----------
   |   |

Computer's turn:

 X |   |
-----------
 X | O |
-----------
 O |   |

Human player's turn (type the x, space and the y, followed by enter):
---> 0 2

 X |   | X
-----------
 X | O |
-----------
 O |   |

Computer's turn:

 X | O | X
-----------
 X | O |
-----------
 O |   |

Human player's turn (type the x, space and the y, followed by enter):
---> 2 1

 X | O | X
-----------
 X | O |
-----------
 O | X |
```

```
Computer's turn:

 X | O | X
-----------
 X | O | O
-----------
 O | X |

Human player's turn (type the x, space and the y, followed by enter):
---> 2 2

 X | O | X
-----------
 X | O | O
-----------
 O | X | X

No one won. :|
```

Para o caso acima (empate), somando apenas os tempos em que a máquina demora para percorrer os caminhos na árvore de possibilidades e achar a sua jogada, com o Minimax foi obtido 0.0540 segundos, enquanto que para a Poda, obteve-se 0.0500 segundos (melhora de 8%).

```
Human player's turn (type the x, space and the y, followed by enter):
---> 1 1

   |   |
-----------
   | X |
-----------
   |   |

Computer's turn:

 O |   |
-----------
   | X |
-----------
   |   |

Human player's turn (type the x, space and the y, followed by enter):
---> 0 1

 O | X |
-----------
   | X |
-----------
   |   |

Computer's turn:

 O | X |
-----------
   | X |
-----------
   | O |

Human player's turn (type the x, space and the y, followed by enter):
---> 0 2

 O | X | X
-----------
   | X |
-----------
   | O |
```

```
Computer's turn:

 O | X | X
-----------
   | X |
-----------
 O | O |

Human player's turn (type the x, space and the y, followed by enter):
---> 2 2

 O | X | X
-----------
   | X |
-----------
 O | O | X

Computer's turn:

 O | X | X
-----------
 O | X |
-----------
 O | O | X

Sorry, You lost! :(
Total of processing time of machine: 0.0469999313354
```

Para o caso acima (derrota do humano), com o Minimax, foi obtido 0.0530 segundos, enquanto que para a Poda, obteve-se 0.0470 segundos (melhora de 12.77%).

4 Conclusões

Como primeiro laboratório, a prática foi bastante interessante e proveitosa, foi realmente útil para o aprendizado dos algoritmos empregados e de algumas heurísticas para os problemas propostos. Na teoria, a maioria desses algoritmos são fáceis de serem utilizados; na prática, é, muitas vezes, complicado atrelar a lógica do problema à estrutura de dados escolhida. Em muitos casos, é melhor analisar e pesquisar, antes de começar a resolver o problema, a estrutura mais adequada para o algoritmo a ser estudado.

Referências

[1] The python wiki. [Online]. Available: https://wiki.python.org

5 Apêndice

O repositório *Git* que possui todo o código do projeto pode ser encontrado em

`https://github.com/Luizangel50/CTC-17_Lab1`