**CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS**
**ENGENHARIA DE COMPUTAÇÃO**
**LABORATÓRIO DE INTELIGÊNCIA ARTIFICIAL**
**Prof. Flávio Cruzeiro**

**2016-1**

# TP2: Pac-Man Multi-Agente

**por Pedro Felipe Froes e Saulo Antunes**

## Passo 1

Foi construída uma função que considera a distância Manhattan da comida e dos fantasmas em relação ao Pac-Man, utilizando o inverso de ambos para compor o `score` da função.

`python pacman.py -p ReflexAgent -l testClassic`

```
Pacman emerges victorious! Score: 564
Average Score: 564.0
Scores:        564
Win Rate:      1/1 (1.00)
Record:        Win
```

`python pacman.py --frameTime 0 -p ReflexAgent -k 1`

```
Pacman emerges victorious! Score: 1220
Average Score: 1220.0
Scores:        1220
Win Rate:      1/1 (1.00)
Record:        Win
```

`python pacman.py --frameTime 0 -p ReflexAgent -k 2`

```
Pacman died! Score: 252
Average Score: 252.0
Scores:        252
Win Rate:      0/1 (0.00)
Record:        Loss
```

**Como é o desempenho do seu agente?**

O Pac-Man consegue vencer no labiritinto `testClassic`, porém suas chances diminuem ao passo que o tamanho do labirinto e o número de fantasmas aumentam. Apesar de poder obter uma pontuação razoável nesses casos (como mostrada na vitória com `Score: 1220`), elas são demoradas, dado que o agente não atua quando os fantasmas se encontram distantes. Isso aumenta o tempo de jogo significamente, já que o Pac-Man só se move quando os fantasmas se aproximam.

## Passo 2

Para construir o *minimax*, foi primeiramente construída uma função `isPacmanTurn` que define se o turno é do Pac-Man ou dos Fantasmas de acordo com o módulo da quantidade de agentes: é zero somente para o Pac-Man, caso contrário, o turno pertence a um fantasma.

Após isso, há a implementação da função `minimax`. Caso a profundidade limite tenha sido atingida, ou se o `gameState` atual corresponde à vitória ou à derrota, é retornado o valor da `evaluationFunction` para o estado atual.

Listas de ações permitidas e de estados sucessores de acordo com cada ação são armazenada em `actions` e `sucessors`. O `minimax` começa dando então avaliando o `score` para cada estado sucessor na profundidade seguinte, com o Pac-Man retornando o `max` dentre os `scores`, e os fantasmas retornando `min`.

A função `minimax` codificada é iniciada com um `maxScore` de -∞, que vai sendo substituído a medida que a exploração na árvore ocorre.

`python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=1`

```
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516
Win Rate:      1/1 (1.00)
Record:        Win
```

`python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=2`

```
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516
Win Rate:      1/1 (1.00)
Record:        Win
```

`python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=3`

```
Pacman emerges victorious! Score: 513
Average Score: 513.0
Scores:        513
Win Rate:      1/1 (1.00)
Record:        Win
```

`python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4`

```
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516
Win Rate:      1/1 (1.00)
Record:        Win
```

Foi possível confirmar que os valores *minimax* do estado inicial são, de fato, 9, 8, 7 e -492 para as profundidades 1, 2, 3, e 4, respectivamente. Além disso, apesar do agente perder algumas vezes, a vitória é possível de ser obtida, mesmo na profundidade 4.

#### Por que o Pac-Man corre para a morte?

O objetivo do Pac-Man é obter o maior `score`, não o maior tempo de vida. Portanto, se ele percebe que não há possibilidade de vitória, como ele supõe cada fantasma como um oponente ótimo, ele prefere morrer mais rapidamente para ser menos penalizado em seu `score`.

## Passo 3

A poda alfa-beta foi construída de maneira similar ao *minimax*, adicionando os parâmetros correspondentes ao alfa e ao beta na chamada da função, e modificando o modo como o Pac-Man e os fantasmas avaliam o estado sucessor.

Os valores de alfa e beta são inicializados como -∞ e +∞, respectivamente. Enquanto o Pac-Man escolhe o valor `max` entre os sucessores, parando a avaliação quando `b <= a`, os fantasmas realizam o mesmo procedimento, porém considerando o valor `min`.

Houve um aumento significativo na velocidade de execução do algoritmo devido à poda. Executando com a semente fixa o `MinimaxAgent` e o `AlphaBetaAgent` na profundidade 4, é possível verificar que o tempo gasto para o *minimax* é quase o dobro de tempo gasto para a poda.

`time python pacman.py -p -a -f -q MinimaxAgent -a depth=4 -l smallClassic`

```
real    0m0.121s
user    0m0.045s
sys     0m0.046s
```

`time python pacman.py -p -a -f -q AlphaBetaAgent -a depth=4 -l smallClassic`

```
real    0m0.071s
user    0m0.043s
sys     0m0.023s
```

Além disso, foi possível verificar que os valores do estado inicial da poda alfa-beta continuam os mesmos do *minimax*, como era esperado.

`python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=1`

```
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516
Win Rate:      1/1 (1.00)
Record:        Win
```

`python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=2`

```
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516
Win Rate:      1/1 (1.00)
Record:        Win
```

`python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=3`

```
Pacman emerges victorious! Score: 513
Average Score: 513.0
Scores:        513
Win Rate:      1/1 (1.00)
Record:        Win
```

`python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4`

```
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516
Win Rate:      1/1 (1.00)
Record:        Win
```

## Passo 4

Ao invés de encarar os fantasmas como oponentes ótimos como o *minimax* e a poda alfa-beta, o *expectminimax* não os vê como tal, e escolhe um caminho baseando-se que os fantasmas tomam suas ações aleatoriamente de maneira uniforme. Como a probabilidade dos fantasmas irem para o próximo estado é uniforme, a decisão do fantasma é baseada na média entre todos os `scores`, enquanto a decisão do Pac-Man permanece como o valor `max` entre eles.

Isso explica porque o Pac-Man não vai de encontro à morte quando não vê possibilidade de vitória. Como o oponente não é encarado como ótimo, o Pac-Man tem mais flexibilidade para tentar sobreviver ao invés de acreditar que sua morte é iminente, e que seu `score` deve ser preservado.

`python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10`

```
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Average Score: -501.0
Scores:        -501, -501, -501, -501, -501, -501, -501, -501, -501, -501
Win Rate:      0/10 (0.00)
Record:        Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss
```

`python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10`

```
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Average Score: 221.8
Scores:        532, 532, 532, 532, -502, -502, 532, 532, -502, 532
Win Rate:      7/10 (0.70)
Record:        Win, Win, Win, Win, Loss, Loss, Win, Win, Loss, Win
```

## Passo 5

Para criar uma nova função de avaliação que considera somente estados e não mais ações, foram considerados diversos parâmetros para aumentar as chances do Pac-Man vencer o jogo, a maioria envolvendo as comidas e os fantasmas no tabuleiro.

```
# FOOD
foods = currentGameState.getFood().asList()
if len(foods):
  score += 1/len(foods)

# FOOD DISTANCE
foods = [manhattanDistance(currentGameState.getPacmanPosition(), food) for food in fo
ods]
score += 1000/max(foods)

# CAPSULES
capsules = currentGameState.getCapsules()
if len(capsules):
  score += 100/len(capsules)

# GHOSTS
ghosts = currentGameState.getNumAgents() - 1
score += 1/ghosts

# DEAD END
actions = currentGameState.getLegalActions()
score -= 100/len(actions)
```

Para melhorar a função de avaliação, considera-se o inverso do número de comidas e de fantasmas no tabuleiro, como no `ReflexAgent`. Entretanto, são considerados ainda a distância do Pac-Man para com as comidas restantes, a presença das cápsulas (as comidas que tornam os fantasmas vulneráveis) no tabuleiro e se o Pac-Man chegou em um estado sem ações possíveis (*dead-end*).

A distância do Pac-Man e das comidas teve o maior peso na decisão, seguida das cápsulas e dos *dead-ends* (que foram penalizações ao invés de incrementações no `score` final), e os números de comidas e fantasmas mantiveram seu peso inicial. No fim, todos esses valores são somados ao `scoreEvaluationFunction` do estado em questão.

A função consegue atingir um `score: 1260` em alguns casos, uma grande melhoria em relação ao `ReflexAgent` inicial.

`python pacman.py -l smallClassic -p ExpectimaxAgent -a evalFn=better -q -n 10`

```
Pacman emerges victorious! Score: 1260
Average Score: 1260.0
Scores:        1260
Win Rate:      1/1 (1.00)
Record:        Win
```