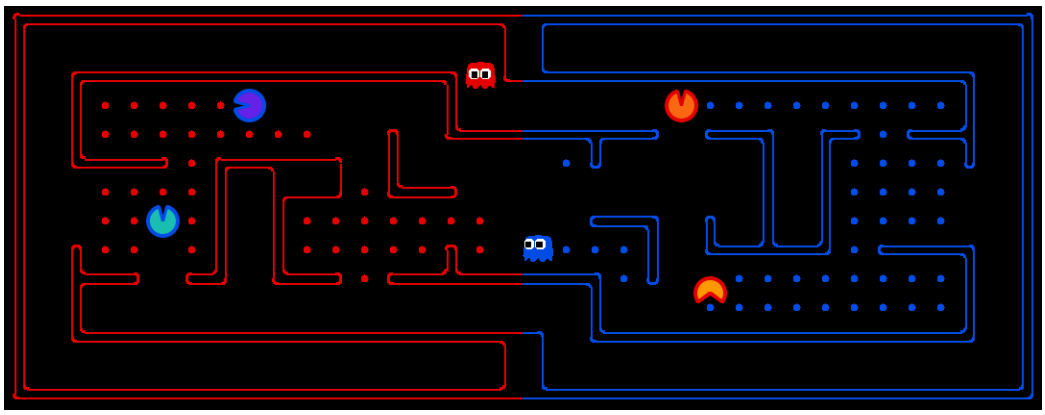
PUCRS
Pontifícia Universidade Católica do Rio Grande do Sul

Inteligência Artificial

Prof. Lucas R. C. Pessutto

Trabalho 1 (Parte 2): Busca Competitiva



Pacman Competitivo

1 Introdução

O objetivo deste trabalho é implementar uma variante multiplayer do jogo Pacman, onde agentes controlam tanto o Pacman quanto os fantasmas em estratégias coordenadas baseadas em equipe. Sua equipe tentará comer a comida do lado oposto do mapa, enquanto defende a comida do seu lado.

2 Requisitos Administrativos

O trabalho deve ser realizado em **duplas**. NÃO SERÃO ACEITOS TRABALHOS INDIVIDUAIS.
A data da entrega será **10/junho (segunda-feira)** pelo moodle. Um membro da dupla deverá enviar o arquivo onde consta o código desenvolvido pela dupla, conforme descrito na seção 6.

Além da entrega, ocorrerão outros dois eventos relacionados a este trabalho: (i) Apresentação e (ii) Competição. Maiores informações consulte a seção 6. A nota deste trabalho contará como parte da média de trabalhos da disciplina.

3 Código Base

O código base do projeto que utilizaremos para nossas tarefas foi desenvolvido por uma equipe da Universidade de Berkeley, na Califórnia.
Este projeto consiste em diversos arquivos Python, alguns dos quais você precisará compreender para completar suas tarefas, e alguns que você pode ignorar. O código está disponível no moodle da disciplina.

Arquivos que você irá editar:

`myTeam.py`	Neste lugar você definirá seus agentes para jogar Pacman (este deverá ser o único arquivo submetido pelo time)

Arquivos que você deve estudar (mas não pode editar!)

`capture.py`	Arquivo que roda o jogo. Ele também conta com a descrição do Estado do jogo e implementa as regras do Pacman
`captureAgents.py`	Implementação de uma classe e métodos úteis para agentes de captura
`baseline.py`	Código de Exemplo, que implementa dois agentes reflexivos simples, que podem ser úteis para você iniciar sua implementação

Arquivos de suporte que você pode ignorar

`game.py`	Implementa a lógica do mundo do Pacman. Possui diversas classes de interesse, como `AgentState`, `Agent`, `Direction` e `Grid`.
`util.py`	Contém a implementação de estruturas de dados úteis para algoritmos de busca
`distanceCalculator.py`	Calcula o caminho mais curto entre as posições do labirinto
`graphicsDisplay.py`	Gráficos do jogo Pacman
`graphicsUtils.py`	Funções de suporte ao jogo
`textDisplay.py`	Gráficos ASCII para o Pacman
`keyboardAgents.py`	Interfaces de controle do jogo pelo teclado
`layout.py`	Código para ler arquivos de layout e guardar seu conteúdo

4 Regras do Pacman Multiplayer

Layout: O mapa do Pacman agora está dividido em duas metades: azul (direita) e vermelha (esquerda). Os agentes vermelhos (todos com índices pares) devem defender a comida vermelha enquanto tentam comer a comida azul. Quando no lado vermelho, um agente vermelho é um fantasma. Ao cruzar para o território inimigo, o agente torna-se um Pacman.

Pontuação: Quando um Pacman captura uma comida, ela é armazenada dentro desse Pacman e removida do tabuleiro. Quando um Pacman retorna para o seu lado do tabuleiro, ele "deposita" a comida que ele está carregando, ganhando um ponto por cada comida entregue. As pontuações da equipe vermelha são positivas, enquanto as da equipe azul são negativas.

Se um Pacman for comido por um fantasma antes de chegar ao seu próprio lado do tabuleiro, ele explodirá e a comida que ele carrega será depositada de volta no tabuleiro.

Capturando um Pacman: Quando um Pacman é capturado por um fantasma adversário, ele retorna à sua posição inicial (como um fantasma). Nenhum ponto é concedido por capturar um oponente.

Comidas Grandes: Se um Pacman capturar os pontos grandes de comida, os agentes da equipe adversária ficarão "assustados" pelos próximos 40 movimentos, ou até serem comidos e reaparecerem, o que ocorrer primeiro. Os agentes "assustados" são suscetíveis enquanto estiverem na forma de fantasmas (ou seja, enquanto estiverem no lado da sua própria equipe) de serem comidos por Pacman. Especificamente, se Pacman colidir com um fantasma "assustado", Pacman não é afetado e o fantasma reaparece em sua posição inicial (não mais no estado "assustado").

Observações: Os agentes só podem observar a configuração de um oponente (posição e direção) se eles ou seu companheiro de equipe estiverem dentro de 5 quadrados (distância de Manhattan). Além disso, um agente sempre recebe uma leitura de distância ruidosa para cada agente no tabuleiro, que pode ser usado para localizar aproximadamente oponentes não observados.

Vitória: Um jogo termina quando uma equipe consegue capturar até $n - 2$ dos n pontos dos oponentes. Os jogos também são limitados a 1200 movimentos de agentes (300 movimentos para cada um dos quatro agentes). Se este limite de movimento for atingido, a equipe que devolveu mais comida vence. Se a pontuação for zero (ou seja, empatada), isso é registrado como um jogo empatado.

Tempo de Computação: Cada agente tem 1 segundo para retornar cada ação. Cada movimento que não retorne dentro de um segundo incorrerá em um aviso. Após três avisos, ou qualquer movimento único que leve mais de 3 segundos, o jogo será anulado. Haverá uma concessão inicial de inicialização de 15 segundos (use a função `registerInitialState`).

5 Projetando Agentes

Nesse projeto, os agentes terão uma tarefa mais complexa, que é equilibrar entre ataque e defesa para poder funcionar efetivamente tanto como um fantasma quanto como um Pacman, em uma configuração de equipe. Podem ser projetados Pacmans exclusivos para ataque/defesa, que foquem em apenas uma das tarefas, ou agentes que tentem equilibrar os dois componentes.

Equipe Baseline: Para iniciar o design do seu agente, fornecemos uma equipe de dois agentes baselines, definidos em `baselineTeam.py`. Eles são bastante ruins. O `OffensiveReflexAgent` simplesmente se move em direção à comida mais próxima no lado oposto. O `DefensiveReflexAgent` vaga pelo seu próprio campo e tenta perseguir invasores que ele eventualmente vê.
Formato do Arquivo: Você deve incluir seus agentes em um arquivo no mesmo formato que `myTeam.py`. Seus agentes devem estar completamente contidos neste único arquivo.
Interface: O arquivo `capture.py` contém a classe `GameState`, que contém diversos métodos úteis como `getRedFood`, que retorna uma matriz de booleanos que mostra as posições da comida no lado vermelho (observe que a grade é do tamanho do tabuleiro, mas contém True apenas para células no lado vermelho com comida). Além disso, observe que você pode listar os índices de uma equipe com `getRedTeamIndices`, ou testar o time de um agente com o método `isOnRedTeam`.
Finalmente, você pode acessar a lista de observações de distância ruidosa por meio de `getAgentDistances`. Essas distâncias estão dentro de 6 unidades da distância verdadeira, e o ruído é escolhido uniformemente ao acaso no intervalo [-6, 6] (por exemplo, se a distância verdadeira for 6, então cada um de $\{0, 1, ..., 12\}$ é escolhido com probabilidade 1/13). Você pode obter a probabilidade de uma leitura ruidosa usando `getDistanceProb`.
Cálculo de Distância: Para facilitar o desenvolvimento do agente, é fornecido código em `distanceCalculator.py` para fornecer distâncias de labirinto pelo caminho mais curto.
Métodos do CaptureAgent: Para começar a projetar seu próprio agente, é recomendado que você crie ele como uma subclasse da classe `CaptureAgent`. Isso fornece acesso a vários métodos que a classe pai já possui implementados. Alguns métodos úteis são:

- `def getFood(self, gameState)`: Returna as comidas que seu jogador pode comer. O retorno será na forma de uma matriz onde `m[x][y] = True` se existir uma comida que pode ser capturada pelo seu time na posição (x, y).
- `def getFoodYouAreDefending(self, gameState)`: Returna uma matriz indicando as posições onde há comidas que seu time deve defender (ou seja, as comidas que seu adversário deve capturar). O retorno será na forma de uma matriz onde `m[x][y] = True` se existir uma comida que seu time deve defender na posição (x, y).
- `def getOpponents(self, gameState)`: Returna os índices dos agentes que são seu oponente. O retorno é na forma de uma lista de valores inteiros, por exemplo, para o jogador vermelho o retorno será $[1, 3]$.
- `def getTeam(self, gameState)`: Returna os índices dos agentes do seu time, na forma de uma lista de inteiros. Para o time azul essa função retornará a lista $[1, 3]$.
- `def getScore(self, gameState)`: Retorna um número inteiro que representa o escore atual da partida, como sendo a diferença entre seu placar e o placar do oponente. Se este número for negativo, então você está perdendo o jogo.
- `def getMazeDistance(self, pos1, pos2)`: Retorna a distância entre duas posições no labirinto. A distância é calculada usando o objeto `Distancer` fornecido. Por padrão, esse método usa a distância de tabuleiro. Ele também pode fornecer a distância de Manhattan.
- `def getPreviousObservation(self)`: Retorna o objeto `GameState` que corresponde ao último estado visto pelo agente (como era o estado do mundo da última vez que o agente se moveu. Lembre-se que essa observação pode não incluir as posições exatas dos agentes oponentes).
- `def getCurrentObservation(self)`: Retorna o objeto `GameState` que corresponde à observação atual do agente (o estado observado atual do jogo. Lembre-se que essa observação pode não incluir as posições exatas dos agentes oponentes).
- `def debugDraw(self, cells, color, clear=False)`: Desenha uma caixa colorida nas celulas especificadas. Se `clear` for `True`, as caixas desenhadas anteriormente serão excluídas. Este método é útil para realizar debugging das posições que seu código está considerando. Atributo `color`: lista de valores RGB entre 0 e 1 (por exemplo, [1,0,0] para vermelho). Atributo `cells`: lista de posições do labirinto para serem desenhadas (por exemplo, [(20,5), (3,22)])

Restrições: Você é livre para projetar qualquer agente que desejar. No entanto, será necessário respeitar as APIs fornecidas para conseguir participar da competição. Agentes que realizarem cálculos durante o turno do oponente serão desqualificados. Em particular, qualquer forma de multi-threading é proibida, pois encontramos dificuldades em garantir que nenhum cálculo seja realizado durante o turno do oponente.

6 Entrega (70% da nota) e Avaliação

Um elemento da dupla deve entregar o arquivo `myTeam.py` com o código desenvolvido para o trabalho. O arquivo deve ser nomeado como `time-<nome-time>.py`. O agente em funcionamento corresponderá a 70% da nota desse trabalho.

6.1 Apresentação (30% da nota)

A apresentação será realizada nas aulas dos dias **11 e 13 de junho**. Neste dia, cada grupo deverá apresentar em 5 minutos a estratégia usada pelos agentes e os resultados de um experimento onde o seu agente joga 20 partidas contra os agentes baseline. Esse experimento pode ser rodado mais rapidamente e com um único comando da seguinte maneira:

```
python .\capture.py -r <seu time> -b baselineTeam -n 20 -Q
```

6.2 Competição

A competição de Pacman será um evento extra que não contará pontos para este trabalho (mas haverá a distribuição de pontos extras aos vencedores!). Este evento será organizado no decorrer do semestre a depender da participação da turma no trabalho. Ele ocorrerá em horário extra-classe e sem cobrança de presença.

As três equipes que vencerem o torneio receberão pontos extras na nota de trabalhos do semestre de acordo com sua colocação no evento. O primeiro lugar receberá 2 pontos, o segundo lugar 1.5 e o terceiro lugar receberá 1 ponto.