UB Social


# CS50AI with Python

## 9.Nim

Mateus Schwede

# Problemática

- Escreva IA que aprenda jogar Nim via aprendizado por reforço (reinforcement learning);
- Jogo Nim consiste em várias pilhas com objetos;
- Jogadores alternam-se removendo qualquer quantidade de objetos de apenas 1 pilha por vez;
- Jogador que remover último objeto perde partida;
- IA aprende jogando contra si mesma diversas vezes e melhorando a partir da experiência;
- Técnica utilizada é Q-learning, que associa recompensas a pares (estado, ação);
- Ações que levam à derrota recebem recompensa -1;
- Ações que levam à vitória recebem recompensa 1;
- Ações que mantêm jogo em andamento recebem recompensa 0, mas com possibilidade de ganhos futuros;
- Estado é representado pela quantidade de objetos em cada pilha, exemplo: [1, 1, 3, 5];
- Ação é tupla (i, j), onde i é índice da pilha e j é quantidade de objetos a ser retirada, exemplo: (3, 5).

# Problemática

- Aplicar ação (3, 5) ao estado [1, 1, 3, 5] gera novo estado [1, 1, 3, 0];
- Fórmula do Q-learning é: $Q(s, a) \leftarrow Q(s, a) + alpha * (valor\ novo - valor\ antigo)$;
- alpha é taxa de aprendizado, que determina quanto IA considera novas informações;
- Valor novo é soma da recompensa atual e da estimativa das recompensas futuras;
- Valor antigo é valor atual da ação Q(s, a);
- Aplicando tal fórmula continuamente, IA passa a identificar quais ações são mais vantajosas em cada situação.

# Instruções

- Baixe código de https://cdn.cs50.net/ai/2023/x/projects/4/nim.zip e descompacte-o.

# Funcionamento

- nim.py contém 2 classes: Nim e NimAI, além de 2 funções: train e play;
  - Classes Nim, train e play já estão implementadas;
  - Classe NimAI precisa que algumas funções sejam implementadas;
  - Classe Nim representa jogo Nim, com lista de pilhas, jogador atual (0 ou 1) e vencedor;
  - Função available_actions retorna conjunto de ações possíveis em um estado;
  - Exemplo: available_actions([2, 1, 0, 0]) retorna {(0, 1), (0, 2), (1, 1)};
  - Função other_player retorna jogador adversário;
  - Função switch_player alterna jogador atual;
  - Função move aplica ação no estado atual e troca jogador.

# Funcionamento

- Classe NimAI representa IA que aprende a jogar;
- Possui dicionário self.q que armazena Q-values dos pares (estado, ação);
- Como listas não podem ser usadas como chave de dicionário, estados são armazenados como tuplas;
- Exemplo: self.q[(0, 0, 0, 2), (3, 2)] = -1 define Q-value para estado e ação especificados;
- IA também tem parâmetros alpha (taxa de aprendizado) e epsilon (para decisão de ação);

# Funcionamento

- Função update já está implementada e realiza Q-learning chamando:
  - get_q_value: obtém Q-value atual;
  - best_future_reward: calcula melhor recompensa futura possível;
  - update_q_value: atualiza Q-value conforme valores obtidos.
- Essas 3 funções precisam ser implementadas;
- Função choose_action também precisa ser implementada e define qual ação IA deve tomar, de forma gulosa ou com algoritmo epsilon-greedy;
- Função train simula n jogos da IA contra si mesma e retorna IA treinada;
- Função play permite que jogador humano jogue Nim contra IA treinada.

# Especificações

- Complete  funções get_q_value, update_q_value, best_future_reward e choose_action no arquivo nim.py;
- Funções recebem state como lista de inteiros e action como tupla (i, j) com pilha e quantidade de objetos a serem removidos;
- Função get_q_value deve retornar Q-value correspondente ao par (estado, ação);
- Q-values são armazenados em self.q, com chaves no formato ((estado como tupla), (ação));
- Se par (estado, ação) não existir em self.q, função deve retornar 0.

# Especificações

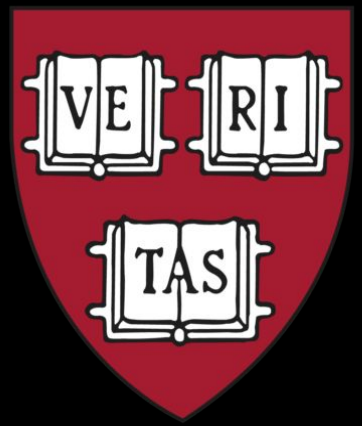
- Função update_q_value recebe state, action, valor atual old_q, recompensa imediata reward e estimativa de recompensas futuras future_rewards, e atualiza Q-value com base na fórmula do Q-learning;
    - Fórmula do Q-learning é: $Q(s, a) \leftarrow$ old_q + alpha * ((reward + future_rewards) - old_q);
    - Valor de alpha é parâmetro de taxa de aprendizado do objeto NimAI;
    - old_q é valor atual do Q-value, e novo valor é soma da recompensa imediata e da estimativa de recompensa futura.
- Função best_future_reward recebe state e retorna melhor recompensa possível entre todas ações disponíveis nesse estado, com base nos valores em self.q;
    - Se ação não estiver em self.q, assuma que seu Q-value é 0;
    - Se não houver ações disponíveis, função deve retornar 0.

# Especificações

- Função choose_action recebe state (e opcionalmente flag epsilon) e retorna ação válida nesse estado;
    - Se epsilon for False, função deve agir de forma gulosa, escolhendo ação com maior Q-value conhecido (ou 0, se não houver);
    - Se epsilon for True, deve aplicar algoritmo epsilon-greedy: com probabilidade self.epsilon, escolher ação aleatória; caso contrário, escolher melhor ação conhecida;
    - Em caso de empate no Q-value entre múltiplas ações, qualquer uma pode ser retornada.
- Não é permitido modificar nenhuma outra parte do arquivo nim.py, exceto funções especificadas.

# Submissão

- Visual Studio Code online: https://cs50.dev
- Testar precisão da lógica do algoritmo: check50 ai50/projects/2024/x/nim
- Testar estilização do código: style50 nim.py
- Para submissão:
  - Em https://submit.cs50.io/invites/d03c31aef1984c29b5e7b268c3a87b7b, entre com GitHub e autorize CS50;
  - Instale pacote Git, Python 3 (e pip), instalando pacotes: pip3 install style50 check50 submit50
  - Submeta o projeto: submit50 ai50/projects/2024/x/nim
- Verificar avaliação: https://cs50.me/cs50ai.




UB Social

Mateus Schwede
HBS ID 202400167108 - DCE ID @00963203