**Instituto Tecnológico de Aeronáutica – ITA**

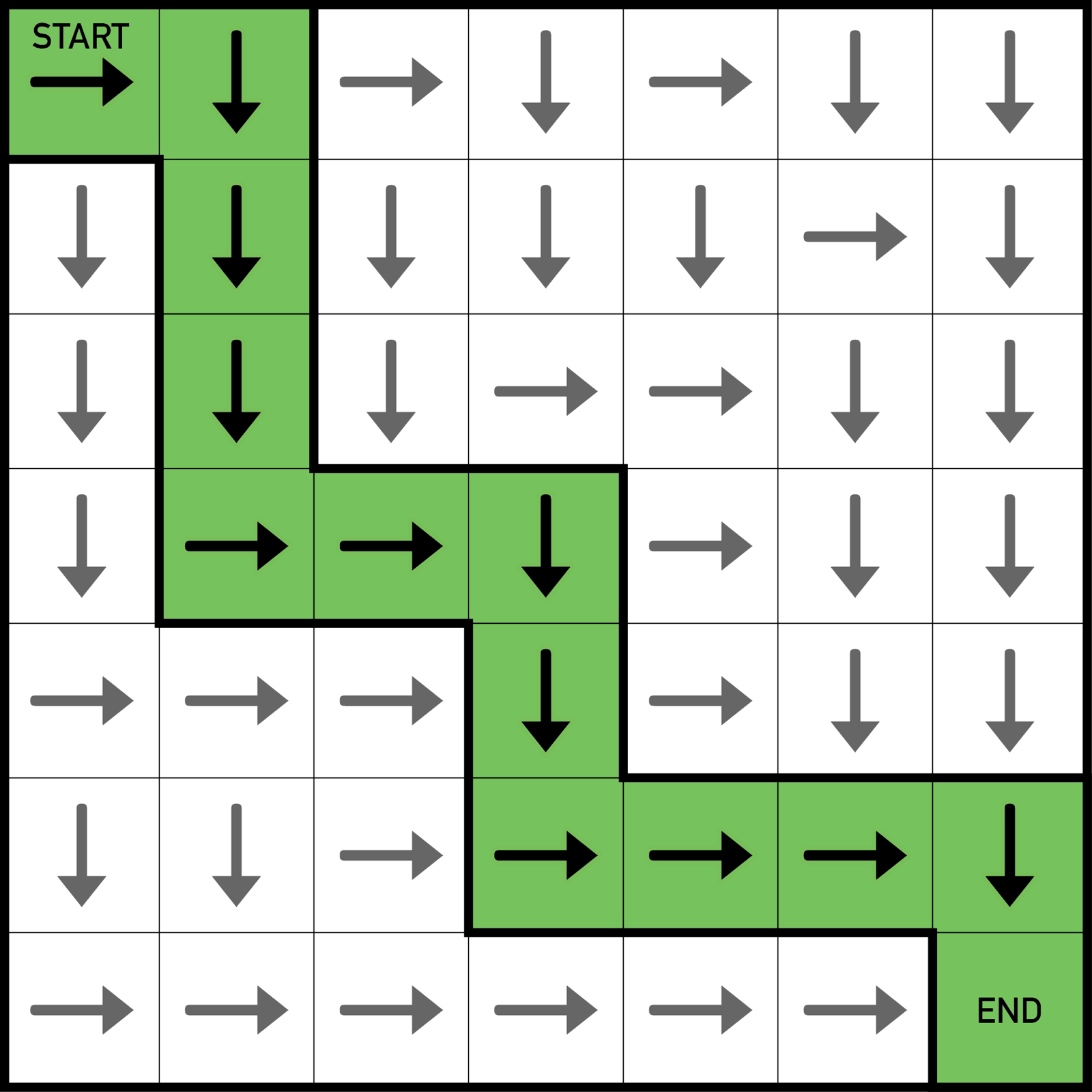
**Inteligência Artificial para Robótica Móvel – CT-213**

**Professor: Marcos Ricardo Omena de Albuquerque Maximo**

**Laboratório 10 (Aula 11) – Programação Dinâmica**

1. **Introdução**

Nesse laboratório, seu objetivo é implementar algoritmos de programação dinâmica no contexto de solução de um Processo Decisório de Markov (*Markov Decision Process* - MDP). Os algoritmos implementados serão avaliação de política (*policy evaluation*), iteração de política (*policy iteration*) e iteração de valor (*value iteration*). No caso, o objetivo é avaliar políticas e determinar políticas ótimas para um grid world, conforme ilustra a Figura 1. Perceba que esses algoritmos resolvem o problema de Aprendizado por Reforço (*Reinforcement Learning* - RL) no caso em que o modelo do MDP é conhecido.



**Figura 1:** exemplo de política ótima em *grid world*.

1. **Descrição do Problema**

O problema consiste em um *grid world* com 5 ações possíveis:

* **S**TOP: continua parado na mesma posição.
* **U**P: move-se uma célula para cima no tabuleiro.
* **R**IGHT: move-se uma célula para direita no tabuleiro.
* **D**OWN: move-se uma célula para baixo no tabuleiro.
* **L**EFT: move-se uma célula para esquerda no tabuleiro.

Considera-se que a ação STOP sempre é executada com perfeição, i.e. com probabilidade 1, o agente permanece na mesma posição após executar essa ação. Já as demais ações tem uma probabilidade de serem executadas corretamente. Se a ação não for executada corretamente, o resultado de uma das demais ações acontece com igual probabilidade, i.e. com . No *grid world*, há obstáculos, que ocupam algumas células do *grid*. Se um determinado movimento for levar o agente para um obstáculo, então o agente permanece na sua posição. Ademais, os limites do *grid* são também barreiras. Além dessas questões, o MDP tem fator de desconto e a recompensa é -1 para cada instante que o agente passa em uma célula que não é a objetivo. Há uma única célula objetivo no *grid*, onde o agente recebe recompensa 0.

Conforme discutido em sala, o algoritmo de avaliação de política é baseado na equação de Bellman de expectativa:

A ideia do algoritmo é realizar a solução iterativa do sistema de equações lineares associado.

Já o algoritmo de iteração de política alterna entre avaliação de política e aprimoramento de política. O aprimoramento é realizado através de tomar uma política gulosa (*greedy*) a partir da função valor da atual política. Pode-se mostrar que esse algoritmo converge para a política ótima. Uma política gulosa (*greedy*) determinística é obtida da seguinte forma:

Nesse caso, quando há mais de uma ação ótima para um certo estado, considera-se que qualquer ação ótima pode ser executada, com igual probabilidade. Com isso, tem-se uma política gulosa estocástica. Um ponto interessante de iteração de política é que não é necessário avaliar a política atual até convergência do algoritmo de avaliação de política para que a iteração de política como um todo convirja para a política ótima.

O algoritmo de iteração de valor abandona a ideia de alternar entre avaliação de política e aprimoramento de política e se baseia em iterar diretamente sobre a função valor de acordo com a equação de otimalidade de Bellman:

Com isso, pode-se mostrar que a iteração de valor converge para a função valor ótima. Após a convergência deste algoritmo, a política ótima pode ser obtida de forma gulosa:

Como comentado anteriormente, caso haja mais de uma ação ótima para um estado, pode-se adotar uma política estocástica em que cada ação ótima é escolhida com igual probabilidade.

1. **Código Base**

O código base traz as seguintes implementações prontas:

* GridWorld: classe que implementa um *grid world*, considerando o que foi descrito na Seção 2. Principais métodos que são interessantes para as tarefas deste laboratório:
  + get\_valid\_sucessors(position): retorna uma lista de pares ordenados contendo os sucessores válidos da posição passada como argumentos. Para filtrar quem são sucessores válidos, elimina-se posições que fiquem fora do *grid* ou que contenham um obstáculo.
  + transition\_probability(s, a, s’): retorna o valor de .
  + reward(s, a): retorna o valor de .
* dynamic\_programming.py: as funções random\_policy() e greedy\_policy() já estão implementadas e criam uma política aleatória e gulosa (com respeito a uma função valor), respectivamente. Você deve implementar as seguintes funções:
  + policy\_evaluation(): realiza avaliação de política.
  + policy\_iteration(): realiza iteração de política.
  + value\_iteration(): realiza iteração de valor.

Para testar suas implementações, use o script test\_dynamic\_programming.py. Um exemplo de saída deste código é apresentado na Tabela 1. Perceba que \* indica células com obstáculos. Além disso, no caso em que a política escolhe várias ações com mesma probabilidade, todas as ações são apresentadas (SURDL significa STOP, UP, RIGHT, DOWN, e LEFT).

| Value function:  [ -384.09, -382.73, -381.19, \* , -339.93, -339.93]  [ -380.45, -377.91, -374.65, \* , -334.92, -334.93]  [ -374.34, -368.82, -359.85, -344.88, -324.92, -324.93]  [ -368.76, -358.18, -346.03, \* , -289.95, -309.94]  [ \* , -344.12, -315.05, -250.02, -229.99, \* ]  [ -359.12, -354.12, \* , -200.01, -145.00, 0.00]  Policy:  [ SURDL , SURDL , SURDL , \* , SURDL , SURDL ]  [ SURDL , SURDL , SURDL , \* , SURDL , SURDL ]  [ SURDL , SURDL , SURDL , SURDL , SURDL , SURDL ]  [ SURDL , SURDL , SURDL , \* , SURDL , SURDL ]  [ \* , SURDL , SURDL , SURDL , SURDL , \* ]  [ SURDL , SURDL , \* , SURDL , SURDL , S ] |
| --- |

**Tabela 1:** exemplo de função valor determinada por avaliação de política e política usada na avaliação.

1. **Tarefas**
   1. **Implementação de Avaliação de Política**

Implemente avaliação de política na função policy\_evaluation() de dynamic\_programming.py. Use o script test\_dynamic\_programming.py para testar sua avaliação de política. A política usada escolhe ação aleatória para todos os estados, exceto no estado objetivo, onde sempre escolhe ação STOP. Preste atenção nos argumentos que são passados para a função para uma correta implementação. O critério de parada consiste de número máximo de iterações e de uma condição . Implemente a avaliação de política no estilo síncrono.

* 1. **Implementação de Iteração de Valor**

Para isso, implemente value\_iteration() de dynamic\_programming.py. Preste atenção nos argumentos que são passados para a função para uma correta implementação. O agente deve então encontrar a função ótima e, a partir dela, a política ótima. Implemente a iteração de valor no estilo síncrono.

* 1. **Implementação de Iteração de Política**

Implemente iteração de política. Você deve alterar entre avaliação de política e aprimoramento de política (greedy\_policy). Para isso, implemente policy\_iteration() de dynamic\_programming.py. Preste atenção nos argumentos que são passados para a função para uma correta implementação. O argumento evaluations\_per\_policy controla quantas iterações de avaliação de política são feitas para cada atualização de política (realizada de forma gulosa). O agente deve então encontrar a política ótima. Implemente a iteração de política no estilo síncrono.

* 1. **Comparação entre *Grid Worlds* diferentes**

Por fim, compare os resultados de avaliação de política, iteração de valor e iteração de política para os seguintes casos:

* CORRECT\_ACTION\_PROB () = 1.0 e GAMMA = 1.0.
* CORRECT\_ACTION\_PROB () = 0.8 e GAMMA = 0.98.

Inclui no seu relatório os resultados obtidos (pode apenas copiar a saída do programa de forma organizada). Comente se os resultados obtidos. Diga se os resultados condizem com o que você esperava.

1. **Entrega**

A entrega consiste do código e de um relatório, submetida através do Google Classroom. Modificações nos arquivos do código base são permitidas, desde que o nome e a interface dos scripts “main” não sejam alterados. A princípio, não há limitação de número de páginas para o relatório, mas pede-se que seja sucinto. O relatório deve conter:

* Breve descrição em alto nível da sua implementação.
* Figuras que comprovem o funcionamento do seu código.

Por limitações do Google Classroom (e por motivo de facilitar a automatização da correção), entregue seu laboratório com todos os arquivos num único arquivo **.zip** (**não** utilize outras tecnologias de compactação de arquivos) com o seguinte padrão de nome: “**<login\_email\_google\_education>\_labX.zip**”. Por exemplo, no meu caso, meu login Google Education é **marcos.maximo**, logo eu entregaria o lab 10 como “**marcos.maximo\_lab10.zip**”. **Não** crie subpastas para os arquivos da sua entrega, **deixe todos os arquivos na “raiz” do .zip**. Os relatórios devem ser entregues em formato **.pdf**.

1. **Dicas**

* Para facilitar o entendimento de como utilizar as estruturas fornecidas no código base, tente primeiramente entender a implementação da função greedy\_policy().