# CECS 451 Artificial Intelligence

Spring 2023 – Final Project

**Deliverables**

1. **Source code** on GitHub repo
2. **Presentation slides** (upload on Canvas)
3. **Project report** in Microsoft word file (upload on Canvas)

**Submission instruction and deadline:** Please zip “presentation slide” and “project report” and upload them on Canvas by 05/01/2023.

**Note**: Only one submission is required from each group and Group leader should upload the files on Canvas.

**Group 1**

|  |
| --- |
| Van Nguyen |
| Kieran King |
| Brandon Jue |
| Edward Kuoch |
| Jessica Ann Consuelo Alejandro |

**Problem Statement:** Game design using Reinforcement Learning

**Concepts:** q-learning, approximate q-learning

**Task Description:** As we have learned, to create an RL model we need to create an environment. OpenAI provides a toolkit for training Reinforcement Learning agents called Gym. The gym comes into play here and helps us to create abstract environments to train our agents on it.

**Your tasks/Instructions:**

* After installing Gym, create an episode in the Gym environment. (Preferably a custom Gym environment)
* Create a Deep Q-Network with the help of Neural Networks
  + Perform hyperparameter tuning, to add the number of nodes in your hidden layers
  + Try using various/appropriate activation functions for the fully connected layers
* Train and test your RL agent
* Interpret and record your observations and output

**References/Resources:**

<https://www.gymlibrary.dev/>

<https://github.com/openai/gym>

<https://github.com/rltraffic/gym-graph-traffic>

**Group 2**

|  |
| --- |
| Jorge Marquez |
| Jose Fraire |
| Connor Koch |
| Oscar Saavedra |

**Problem Statement:** Design a role-playing game of your choice using Expectimax

**Concepts:** expectimax algorithm

**Task Description:** Unlike deterministic games, the stochastic ones have a [probabilistic](https://www.baeldung.com/java-probability) element. This AI algorithm is like the minimax algorithm, except it has been adapted for stochastic games.

**Hint:** To account for the probabilistic nature of the game, the values of the chance nodes are the weighted average of each possible dice roll (if you consider creating a game which involves dice)

**Your tasks/Instructions:**

* Implement the game of your choice in Python
* Follow OOPS Programming style
* Your code should be structured and modularized
* Your code should be able to handle error inputs - Exception Handling

**References/Resources:**

* <https://www.geeksforgeeks.org/python-oops-concepts/>
* Refer to the Expectimax Implementation worked on in one of our lab sessions.

**Group 3**

|  |
| --- |
| Jocelyn Gonzalez |
| Jose Jimenez |
| Ruby Nguyen |
| Bich-Tram Pham |
| Dimpal Shah |

**Problem Statement:** Design a Tic-Tac-Toe by making use of Alpha-Beta Pruning

**Concepts:** Alpha-Beta Pruning, Minimax Algorithm

**Task Description:** Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm. Your task is to create a Tic-Tac-Toe (3x3 matrix) which uses Alpha-Beta Pruning optimization technique in the Minimax algorithm.

**Your tasks/Instructions:**

* Implement the Tic-Tac-Toe game in Python
* Follow OOPS Programming style
* Your code should be structured and modularized
* Your code should be able to handle error inputs: Exception Handling

**References/Resources:**

* <https://www.geeksforgeeks.org/python-oops-concepts/>
* Refer to the MiniMax Implementation and Alpha Beta Pruning algorithm worked on in one of our lab sessions.

**Group 4**

|  |
| --- |
| Lionel Quintanilla |
| Justin Le |
| Sierra Harris |
| Amanuel Reda |
| Salvador Villanueva |

**Problem Statement:** Implement Connect-4 game using Minimax Algorithm

**Concepts:** Minimax Algorithm

**Task Description:** Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, if your opponent also plays optimally.

**Connect-4 Game instructions:**

* It is also called “Four-in-a-Row” and “Plot Four.”
* Two players play this game on an upright board with six rows and seven empty holes. (6x7 matrix)
* Each player has an equal number of pieces (21) initially to drop one at a time from the top of the board. Then, they will take turns to play and whoever makes a straight line either vertically, horizontally, or diagonally wins.

**Your tasks/Instructions:**

* Implement Connect-4 game in Python
* Follow OOPS Programming style
* Your code should be structured and modularized
* Your code should be able to handle error inputs- Exception Handling

**References/Resources:**

* <https://www.geeksforgeeks.org/python-oops-concepts/>
* Refer to the MiniMax Implementation worked on in one of our lab sessions.

**Group 5**

|  |
| --- |
| Jordan Hilado |
| Nicole Cerda |
| Stephen Lyons |
| Aldo Dagio-Ortega |
| Matthew Kriesel |

**Problem Statement:** Implement a game using Search algorithms using BFS and DFS

**Concepts:** Depth-First Search, Breadth-First Search

**Task Description:** Create a maze, where the player reaches the reward at the end of the maze using the search algorithms. Create two mazes - one simple maze, the other a slightly more complex mazeand run your search algorithms.

* Use the two search algorithms and interpret the results, which algorithm finds the optimised path to the reward- in terms of time?
* What are your inferences and observations? Create a report.

**Your tasks/Instructions:**

* Implement the maze game in Python
* Follow OOPS Programming style
* Your code should be structured and modularized
* Exception Handling should be incorporated in you project

**Hint:** The implementation is somewhat like the PACMAN implementation done in one of the lab sessions

**References/Resources:**

1. <https://github.com/Tynab/Searching-Algorithms>
2. <https://www.geeksforgeeks.org/python-oops-concepts/>
3. <https://github.com/rahvis/CECS451-ISA/tree/main/assignment_01>

**Group 6**

|  |
| --- |
| Ghabrille Ampo |
| Eric Nguyen |
| Celia Ramperez Martin |
| Nishaan Amin |
| Yazmin Yanez |

**Problem Statement:** Implement a game using Search algorithms using Uniform-Cost Search and A\* Search

**Concepts:** Uniform-Cost Search, A\* Search

**Task Description:** Create a maze, where the player reaches the reward at the end of the maze using the search algorithms. Create two mazes - one simple maze, the other a slightly more complex mazeand run your search algorithms.

* Use the two search algorithms and interpret the results, which algorithm finds the optimised path to the reward- in terms of time?
* What are your inferences and observations? Create a report.

**Your tasks/Instructions:**

* Implement the maze game in Python
* Follow OOPS Programming style
* Your code should be structured and modularized
* Exception Handling should be incorporated in you project

**Hint:** The implementation is somewhat like the PACMAN implementation done in one of the lab sessions

**Group 7**

|  |
| --- |
| Corrine Rivera |
| Eric Wang |
| Francisco Rivera |
| Laura Bocquet |
| Dhruv Savla |

**Problem Statement:** Implement Tic-Tac-Toe Game using the Monte Carlo Tree Search

**Concepts:** Monte-Carlo Tree Search

**Task Description:** There are many ways to search for the best move in a game tree. The simplest one is to search every path in the tree, choosing the move that is guaranteed to produce the best outcome even against an optimally playing opponent. This is what minimax does. However, for many complex games, this is not possible because the game tree is very large. MCTS deals with large trees effectively by *sampling* many paths down the game tree. This means it repeatedly goes down many, but not all, of the paths.

MCTS has four phases:

1. Selection

2. Expansion

3. Simulation

4. Backpropagation

**Your tasks/Instructions:**

* Implement the Tic-Tac-Toe game in Python
* Follow OOPS Programming style
* Your code should be structured and modularized
* Each of the four phases mentioned above should be implemented in separate methods
* Create a report detailing exactly what is happening in the tree at each phase in the iteration - proper diagrams and explanation is required.

**References/Resources:**

1. <https://www.geeksforgeeks.org/python-oops-concepts/>
2. Refer to the notes and the zoom recording of the lab session
3. Refer to the code implementation of MCTS done in the lab sessions

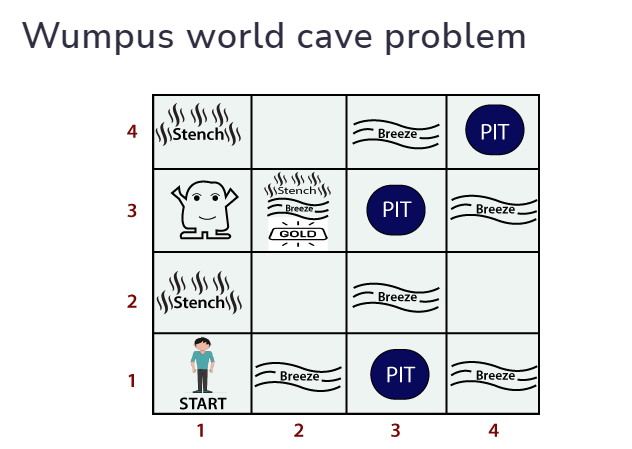
**Group 8**

|  |
| --- |
| Steven Yacoub |
| Devin Elmore |
| Daryl Nguyen |
| Thomson Nguyen |
| Kenneth Valero |

**Problem Statement:** Implement a Markov Decision Process for Wumpus World Problem and use Policy Iteration for convergence

**Concepts:** Policy Iteration

**Task Description:**



**States:** 16 States

**Actions:** Left, Right, Up, Down

**Rewards:**

* The monster in [3,1] location in the grid where Wumpus, the monster is present has a penalty of -1000
* The gold bar present in [3,2] has a reward of +1000

**Assumptions:**

* Assume the pits (in this case [1,3], [3,3], [1,4]) are walls and these grids cannot be entered into
* Ignore the Cells which are labelled as ‘stench’ and ‘breeze’, Treat them as empty cells that the player can enter information.
* Define a random Policy

**HINT:** For each empty cell in the grid (leaving out the PIT CELLS, WUMPUS CELL, GOLD CELL), assign an action for example: For [1,1] your policy states the action to be taken in this cell is ‘UP’

Later in the subsequent iterations during Policy Improvement, the actions to be taken in each state would change as you proceed to convergence

**Your tasks/Instructions:** This is a modified (custom) Wumpus World Problem. If you want to read more about Wumpus World, I have attached the resource under the References section. However, we will be leaving out some of the assumptions of the Actual Wumpus World.

The implementation of the MDP is like the one taught during the lab sessions with just slight modifications. You can reuse the code in the lab if it makes your job easier. But keep in mind to implement the following assumptions as you would be graded accordingly

* How many iterations of Policy Evaluation and Policy Improvement have you done to get convergence?
* While submitting the report of your project, use the Bellman update rule (derived from the Bellman equation) to compute the utility for the states for two iterations- Submit your calculations in a PDF (Refer to the notes of Policy Iterations given in the GitHub Link)

**References/Resources:**

1. Refer to the notes and recorded lab sessions for a clearer understanding of the concept - <https://github.com/rahvis/CECS451-ISA/tree/main/Lab_03_06_2023>
2. Wumpus World: <https://www.educative.io/answers/what-is-the-wumpus-world-in-artificial-intelligence>
3. Refer to the implementation of Policy Iteration code and pseudocode done during the lab session.

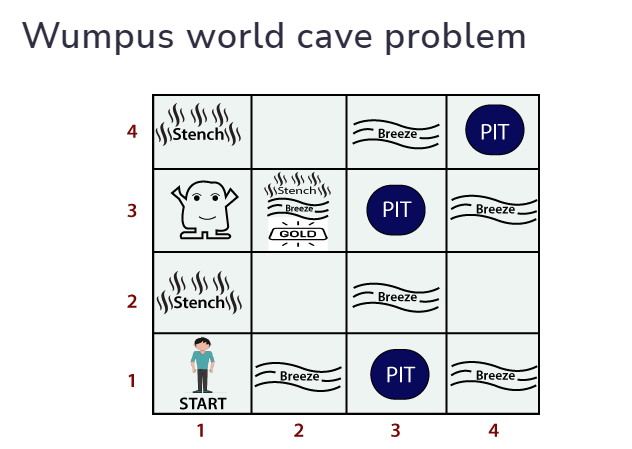
**Group 9**

|  |
| --- |
| Steven Yacoub |
| Devin Elmore |
| Daryl Nguyen |
| Thomson Nguyen |
| Kenneth Valero |

**Problem Statement:** Implement a Markov Decision Process for Wumpus World Problem and use Value Iteration for convergence

**Concepts:** Value Iteration

**Task Description:**



**States:** 16 States

**Actions:** Left, Right, Up, Down

**Rewards:**

* The monster in [3,1] location in the grid where Wumpus, the monster is present has a penalty of -1000
* The gold bar present in [3,2] has a reward of +1000

**Assumptions:**

* Assume the pits (in this case [1,3], [3,3], [1,4]) are walls and these grids cannot be entered into
* Ignore the Cells which are labelled as ‘stench’ and ‘breeze’, Treat them as empty cells that the player can enter information.
* Define a random Policy

**HINT:** For each empty cell in the grid (leaving out the PIT CELLS, WUMPUS CELL, GOLD CELL), assign an action for example: For [1,1] your policy states the action to be taken in this cell is ‘UP’

**Your tasks/Instructions:** This is a modified (custom) Wumpus World Problem. If you want to read more about Wumpus World, I have attached the resource under the References section. However, we will be leaving out some of the assumptions of the Actual Wumpus World.

The implementation of the MDP is like the one taught during the lab sessions with just slight modifications. You can reuse the code in the lab if it makes your job easier. But keep in mind to implement the following assumptions as you would be graded accordingly

* How many iterations were needed to get convergence?
* While submitting the report of your project, use the Bellman update rule (derived from the Bellman equation) to compute the utility for the states for two iterations - Submit your calculations in a PDF (Refer to the notes of Value Iterations given in the Github Link)

**References/Resources:**

1. Refer to the notes and recorded lab sessions for a clearer understanding of the concept - <https://github.com/rahvis/CECS451-ISA/tree/main/Lab_03_06_2023>
2. Wumpus World: <https://www.educative.io/answers/what-is-the-wumpus-world-in-artificial-intelligence>
3. Refer to the implementation of Value Iteration code and pseudocode done during the lab session

**Group 10**

|  |
| --- |
| Darius Koroni |
| Jett Sonoda |
| Bryan Tran |
| Tien Nguyen |
| Nhi Pham |

**Problem Statement**: Implement the Travelling Salesman Problem using Simulated Annealing and Hill Climbing Algorithm

**Concepts**: Simulated Annealing, Hill Climbing Algorithm

**Task Description**: Given a set of cities and the distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. This is the Travelling Salesperson Problem

**Your tasks/Instructions**

1. Showcase the implementation of solving TSP using Hill Climbing Algorithm
2. Showcase the implementation of solving TSP using Simulated Annealing Algorithm
3. Report the inferences made between the two implementations.
4. Report how these implementations different from using Dynamic Programming
5. Your code should be structured and modularized and follow OOPS Programming style

**References/Resources**

1. Refer to the notes and recorded lab sessions for a clearer understanding of the concept