# Project Presentation

## Introduce your project: what kind of problem are you working on

This project addresses an obstacle avoidance problem in a grid-based environment using reinforcement learning. The objective is to train an agent to navigate through a grid, starting from a predefined position, while avoiding obstacles and reaching a designated goal position. This problem is commonly encountered in robotics, pathfinding applications, and autonomous systems.

## Describe your problem as a MDP. Mention what the states, actions, rewards, etc., are

The problem is modeled as a Markov Decision Process (MDP), which includes:  
- \*\*States\*\*: Each state represents the agent's position in the grid. For example, (x, y) indicates the agent's coordinates.  
- \*\*Actions\*\*: The possible moves the agent can take, such as moving up, down, left, or right.  
- \*\*Rewards\*\*: Rewards are assigned to guide the agent's learning process. A reward of +100 is given for reaching the goal, -10 for colliding with an obstacle, and -1 for each step taken to encourage efficient navigation.  
- \*\*Transitions\*\*: These describe how the agent's actions influence the next state, factoring in the grid's boundaries and obstacle placement.

## Describe your project approach

The approach involves designing a custom environment using OpenAI's Gym framework to simulate the grid world. The agent interacts with this environment and learns a policy that maps states to optimal actions using Q-learning. Key steps in the approach include defining the environment's dynamics, designing the reward structure, and implementing the learning algorithm.

## What algorithms did you use? Describe how they work briefly

The Q-learning algorithm is employed, which is a model-free reinforcement learning technique. It uses a Q-table to store the estimated value of taking a specific action in a given state. The Q-value updates follow the Bellman equation:  
Q(s, a) = Q(s, a) + α [R + γ \* max\_a' Q(s', a') - Q(s, a)]  
Here:  
- \*\*α (learning rate)\*\*: Determines the extent to which new information overrides old information.  
- \*\*γ (discount factor)\*\*: Balances immediate and future rewards.  
The algorithm iteratively updates the Q-table based on the agent's interactions with the environment, eventually converging to an optimal policy.

## Describe your implementation including things like data pre-processing, replay memories, etc.

The implementation process includes:  
1. \*\*Environment Design\*\*: Creating a grid-based environment with predefined start, goal, and obstacle positions.  
2. \*\*Q-Table Initialization\*\*: Initializing the Q-table to zero for all state-action pairs.  
3. \*\*Action Selection\*\*: Using an epsilon-greedy policy to balance exploration and exploitation during training.  
4. \*\*Reward Mechanism\*\*: Defining rewards for reaching the goal, colliding with obstacles, and taking steps.  
5. \*\*Visualization\*\*: Plotting the agent's progress over episodes using Matplotlib to monitor learning trends.

## If you're working on algorithm comparison, how did you keep the variables to a minimum

## In this project, the focus was on comparing four different learning agents: DQN, DDQN, Dueling DQN, and PER. To minimize the number of variables, I kept factors such as grid size, number of obstacles, and the goal's position constant across all training episodes. Initially, I tested all four agents, and the average rewards were as follows:

## DQN: Average Reward = 23.17

## DDQN: Average Reward = -17.35

## Dueling DQN: Average Reward = 23.41

## PER: Average Reward = -24.95

## Based on these results, only the DQN and Dueling DQN agents showed positive average rewards. To streamline the experiment and reduce unnecessary complexity, I focused the comparison solely on these two agents, discarding the others. This allowed for a more controlled and meaningful comparison, ensuring that only the agents with positive performance were considered in the final analysis.

## What are the metrics you're using to assess the success of your agent

Metrics used to evaluate the agent's performance include:  
- \*\*Cumulative Rewards\*\*: Monitoring the total rewards earned by the agent over episodes.  
- \*\*Goal Completion Rate\*\*: Tracking the percentage of episodes where the agent successfully reaches the goal.  
- \*\*Collision Frequency\*\*: Measuring the number of times the agent collides with obstacles during training.

## Difficulties faced and how you fixed them

The main challenges faced during the project include:  
- \*\*Hyperparameter Tuning\*\*: Finding optimal values for the learning rate (α), discount factor (γ), and epsilon.  
- \*\*Exploration-Exploitation Tradeoff\*\*: Ensuring the agent explores sufficiently while exploiting learned knowledge.  
- \*\*Reward Design\*\*: Balancing the reward structure to prevent undesired agent behaviors, such as looping in one area.  
These challenges were addressed through systematic testing, parameter adjustments, and frequent visualization of the agent's behavior.

## Future work/remaining work

Future enhancements to the project include:  
- \*\*Scalability\*\*: Extending the environment to larger grids with more complex obstacle layouts.  
- \*\*Dynamic Obstacles\*\*: Introducing moving obstacles to increase the problem's complexity.  
- \*\*Algorithm Comparison\*\*: Comparing Q-learning with advanced methods like Deep Q-Networks (DQN) to assess performance improvements.