Hackathon Report

Name	SRN
Nandita R Nadig	PES2UG23CS365
Moulya K A	PES2UG23CS351
Najmus Seher	PES2UG23CS359
Sharath Gowda GR	PES2UG24CS823

Key Observations:

1. What were the most challenging parts?
    - Converting the output of the forward-backward algorithm into a usable form for the agent was challenging. While the algorithm gives the probability of the true letter at each position, the agent needed the probability that a letter appears in at least one blank, which required careful handling of independent probabilities.
    - The state representation couldn't rely on the full game screen, as this would create an unmanageably large state space. Instead, a 53-dimensional hybrid vector was used, combining probabilistic estimates, guessed letters, and remaining lives, allowing the neural network to process the information effectively.
    - Designing a reward function that guided learning was tricky, as it had to balance immediate feedback with the goal of winning. Rewards for correct guesses, penalties for mistakes, and large terminal rewards for winning or losing were carefully tuned to ensure the agent focused on achieving victory rather than just avoiding errors.
    - The neural network needed to process the hybrid state vector and make accurate predictions about correct guesses. This required ensuring it could generalize effectively from the features provided, allowing the agent to learn efficiently and perform well in the game.

2. What insights did you gain?
    - The hybrid system combines HMM's probabilistic insight with RL's goal-driven learning, creating a model that's both informed and strategic. Neither method alone could achieve this balance of language intuition and gameplay awareness.
    - Action masking was crucial to efficiency, preventing the agent from repeatedly guessing the same letter. By setting invalid actions' Q-values to negative infinity, the agent instantly learned the game's basic rules.

Strategies:

1. Discuss your HMM design choices.
    - The model was built so that each hidden state represented one of the 26 English letters, allowing it to infer likely letter sequences in a word based on learned language patterns.
    - Transition probabilities were learned from a large text corpus, capturing how likely one letter is to follow another and reflecting realistic English structure.
    - Emission probabilities described the relationship between observed and hidden letters, with blank positions treated as providing no information and visible letters given a high likelihood of matching the true hidden letter, while including a small uncertainty for robustness.
    - Inference used a forward-backward procedure to calculate the probability of each possible letter at every position, providing informed inputs that guided the reinforcement learning agent's decision-making.

2. Detail your RL state and reward design and why you chose them.
    - The agent's state was represented as a 53-dimensional vector capturing all key aspects of the game. It combined probabilistic estimates of which letters were most likely to appear, memory of letters already guessed, and the number of remaining lives scaled between zero and one to reflect urgency.
    - The probabilistic component provided informed intuition for likely letters, guiding data-driven guesses. The memory component prevented repeated guesses and supported efficient decision-making, while the normalized life count allowed strategy to adjust, becoming more cautious or bold depending on remaining chances.

- The reward system mirrored the game's final scoring while offering continuous feedback. Incorrect or repeated guesses were penalized to discourage wasteful actions, while winning or losing delivered large positive or negative rewards, reinforcing overall objectives.
- A small reward for each correct guess encouraged steady progress. This immediate positive feedback helped prevent overly cautious behavior and motivated taking thoughtful risks to achieve victory.

Exploration:

How did you manage the exploration vs. exploitation trade-off?

- A standard epsilon-greedy approach with exponential decay was used for exploration, which is effective for DQNs. At each step, the agent either picked a random unguessed letter to explore or chose the letter it thought was best based on what it had learned.
- The balance between exploring and exploiting was controlled by gradually reducing the exploration rate over time. At the start, the agent explored completely to see a variety of game situations, and as training went on, it relied more on its learned strategy.
- The decay was slow on purpose, so the agent didn't get stuck in a bad early strategy. This way, it had enough time to learn and gradually became more confident in making the best moves.

Future Improvements:

If you had another week, what would you do to improve your agent?

- Switching to a Dueling Deep Q-Network could improve learning by separately evaluating how good a state is and how much better a specific action is, making training more stable and efficient.
- Using a system that prioritizes learning from surprising or unexpected outcomes would allow the agent to focus on its mistakes and improve faster.
- Training separate models for words of different lengths could make predictions more accurate, as letter patterns often vary depending on word length, giving the agent better guidance for its guesses.