AI Agents Learn to Play Exploding Kittens

Summer Royal
Stanford University
sroyal@stanford.edu

Shriya Reddy
Stanford University
reddysh@stanford.edu

***Abstract*—In this project, we developed three AI agents to play the card game *Exploding Kittens*: a Maximum Likelihood Estimation Agent, a Q-Learning agent, and a Bayesian Network agent. *Exploding Kittens* is a card game where users take turns drawing cards from a deck. A player loses the game if they draw an Exploding Kitten card without defusing it. Various action cards can help players avoid the Exploding Kitten cards. We first created a simulation of the game with probabilistic deck composition, deterministic cards, game rule constraints, and a hashed state representation with 100,000 possible states. Unlike previous projects, our game simulation exactly replicated the original *Exploding Kittens* game environment and rules without any simplifying assumptions. The MLE agent estimates transition and reward models for value-iteration planning, the Q-Learning agent learns action-value functions through temporal-difference updates, and the Bayesian agent updates beliefs about the deck composition and exploding kitten probabilities using a Dirichlet-Beta inference model. We built an interactive website where users can play against the three AI agents and an agent that follows a random policy.**

We evaluated all three agents across 500 games against diverse baseline opponents in 4-player configurations. Q-Learning achieved the highest win rate at 7.4% with 22.4% strategic card usage, outperforming MLE (2.6%, 11.4% strategic) and Bayesian (0.8%, 1.2% strategic). We define a strategic card play to be any turn in which the player plays a card besides a defuse card. In multiplayer tournaments with all agents competing simultaneously, Q-Learning exceeded the theoretical random baseline of 25%, achieving 27% win rate. Our results reveal that the full game's heavy reliance on draw luck (in our simulations, the average probability of drawing an Exploding Kitten across all turns was approximately 9limits the strategic advantage reinforcement learning agents can extract.

***Index Terms*—Exploding Kittens, MLE, Q-Learning, Bayesian Network, reinforcement learning.**

I. Introduction

Exploding Kittens is a card game that is similar to Russian Roulette. Each player starts with five cards, including one defuse card. Players take turns drawing cards from the deck, and a player loses the game if they draw an exploding kitten card, unless they play their defuse card to stop the bomb. Several action cards can allow users to avoid drawing an exploding kitten: attack (skip your turn and make the next player take 2 cards), favor (force any other player to give you one card of their choice), nope (stop any action except an exploding kitten), shuffle (randomly shuffle the draw pile), skip (skip your turn), see the future (see three cards into the future), and cat cards (match 2 cat cards to steal a random card from any player).

A. Prior Work

In 2018, a CS 238 project group worked on a reinforcement learning project for Exploding Kittens [4]. They modeled the game as a Markov Decision Process (MDP) and developed an agent using Monte Carlo Tree Search (MCTS) and neural network value function approximation. They found that there was a very high variance in the performance of their agent based on the duration of training, but they were able to find an optimal training duration such that their agent tied in most simulations and won in the remaining simulations.[1]

In 2024, a CS238 project group expanded on the 2018 project by modifying the MDP with improvements to the state representation and incorporating various simplifications for tractability [2]. They simplified the game by only having two players and limited the deck to nine card types: Skip, Attack, Exploding Kitten, Defuse, and five cat cards with no effect (Tacocat, Hairy Potato Cat, Rainbow Cat, Beard Cat, and Cattermelon). The special Favor and Nope cards from the original game were excluded since they would require the game state to update mid-turn. They also removed serial actions so players could only draw one card before ending their turn. In the original 2018 project, players could take multiple actions in series, which would require the game state to update mid-turn. The 2024 group expanded each player's state to track additional observable information, such as the counts of previously-played cards by both players, the number of cards remaining in the deck, and the number of cards in the opponent's hand.

B. Our Solution

For our project, we created three AI agents that optimally play the game of Exploding Kittens: a Maximum Likelihood Estimation Agent, a Q-Learning Agent, and a Bayesian Network Agent. We also created a Random Agent to act as our baseline for comparison. The Random Agent follows a basic policy, but it does not incorporate any AI algorithms. We created a simulation where all agents (MLE Agent, Q-Learning Agent, Bayesian Agent, and Random Agent) can play a game of Exploding Kittens against each other. We recorded the results of each game to determine which agent performed the best. We also created an interactive website where a user can play against all four agents.

[1]In the actual Exploding Kittens game, it is not possible to have a tie, but their game state simplifications made it possible to tie.

II. Approach

A. State and Action Representation

We started by creating a simulation of the Exploding Kittens game. Our state representation included the following features: Hand Size (Number of cards in the player's hand, Defuse Count (number of Defuse cards available), Deck Size (remaining cards in the draw deck), Alive Count (number of players still in the game), Alive Mask (binary vector indicating which players are alive), Top-3 Cards (the next three cards in the deck, which is only known if the agent just played a See Into the Future card). We encoded states using a deterministic hashing function that converts the feature dictionary into a unique integer state ID. We wanted to have a large enough state space to avoid collisions, but we wanted to remain computationally tractable, so we set the total number of states to 100,000.

TABLE I
Action Space

Action #	Action	Description
0	Draw	Draw a card from the deck
1	PlaySkip	Play a skip card (end turn without drawing a card)
2	PlayAttack	Play an Attack card (Force the next player to draw 2 cards)
3	PlayShuffle	Play a Shuffle card (shuffle the deck)
4	PlayNope	Play a Nope card (deny previous player's request)
5	PlaySeeFuture	Play a See Future card (see next 3 cards in the deck)
6	PlayCat	Play a cat card
7	PlayDefuse	Play a Defuse card (omit exploding kitten bomb)

B. Transition Dynamics

Our simulated game environment includes all the core elements of the Exploding Kittens game. We have each card type featured (Table 1), we dynamically generate the deck with a probabilistic card distribution, players are eliminated if they draw an Exploding Kitten card without having a Defuse Card in their hand, and the last player automatically wins.

Our transition function $T(s, a, s')$ models the stochastic nature of card draws. The possible card drawing actions are probabilistic based on the deck's probabilistic composition. Each player's options for legal cards to play is based on deterministic state changes. Whether or not an Exploding Kitten takes out a player is based on the probabilistic draw the player makes as well as the deterministic state of the player's current hand (whether the player has a Defuse card).

Our reward function has a negative reward of -10 for losing the game by drawing an Exploding Kitten without any available defuse cards. Our reward function incentivizes winning overall by having a very large positive reward of +100 for winning the game. We also wanted to incentivize strategic behavior and disincentivize risky behavior by having a small positive reward of +10 for successfully playing a strategic card and having a very small negative reward of -0.1 for taking a risky draw. Previous CS 238 Exploding Kittens projects had a simplified game simulation where it was possible to have a draw, but we did not make any simplifying assumptions; it is not possible to have a tie in the Exploding Kittens game, so our reward function did not specify a reward value for this case.

C. AI Agents

Our MLE agent follows a model-based approach by estimating the Markov Decision Process parameters from the transition probabilities that it observes. The algorithm collects (s, a, r, s') tuples through random exploration and maintains a count of $N(s, a, s')$ for transitions and a count of $R_{sum}(s, a)$ for rewards. We estimate the transition probabilities as $P(s'|s, a) = \frac{N(s,a,s')}{\sum_{s''} N(s,a,s'')}$ and the reward function as $R(s, a) = \frac{R_{sum}(s,a)}{\sum_{s''} N(s,a,s'')}$. The algorithm solves the estimated MDP using value iteration and extracts the optimal policy as $\pi^(s) = \arg\max_a Q^(s, a)$. The value iteration update function is: $V_{k+1} = max_a[R(s, a) + \gamma \sum_{s'} P(s'|s, a)V_k(s')]$.

Our Q-learning agent uses the standard off-policy Q-learning update: $Q(s, a) \leftarrow Q(s, a)+\alpha(r+\gamma max_{a'} Q(s', a')-Q(s, a))$. We set the learning rate (α) to be 0.1, the discount factor (γ) to 0.99, and set the exploration rate (ϵ) to 0.1. We chose to use a model-free method to avoid explicitly estimating a model of the transition function. Our approach handles the stochastic transitions of the game naturally.

Our Bayesian agent combines model-free learning with Bayesian inference. The deck composition is modeled as a Dirichlet distribution over card types. The belief about the probability of drawing an Exploding Kitten card is $P(\text{bomb}|\text{deck composition})$. We use a Beta-Bernoulli model for the probability of an Exploding Kitten (p):

$$\mathbb{E}(\text{p}) = \frac{\alpha + k}{\alpha + \beta + \text{ n}}$$

where our Beta prior parameters are $\alpha = 1$, $\beta = 1$, n = number of card draws agent has observed, and k= the total number of drawn cards that were Exploding Kitten cards. The agent modifies Q-values based on current beliefs:

$$Q(s, a) = Q(s, a)+\beta\times\text{uncertainty}(s, a)-\lambda\times\text{risk}(s, a \text{ beliefs})$$

where $\text{Uncertainty}(s, a)$ encourages exploration in uncertain states and $\text{risk}(s, a|\text{beliefs})$ penalizes risky actions when bomb probability is high.

D. Training Process

During training, our algorithm monitors the percentage of games won by the agent, the average reward per episode, how often agents explore vs exploit, and the belief accuracy over the probability of drawing an Exploding Kitten card. Each model's strategy was saved as a policy file for simulation-based performance evaluation and for use on an interactive website.

TABLE II
TRAINING PARAMETERS

Parameter	MLE	Q-Learning	Bayesian
Episodes	20,000	50,000	20,000
Exploration	random	ϵ-greedy	uncertainty-based
Learning Rate	N/A	0.1	0.1
Discount Factor	0.99	0.99	0.99
Batch Size	Full dataset	Online	Online & Full dataset

E. Why We Chose This Approach

The game of Exploding Kittens is a stochastic, partially observable, multi-player environment with a highly skewed reward function (some strategic actions with small immediate effects and a rare, severe negative event). The game is very complex, the state space is extremely large (100,000 possible states in our case), and there's a high degree of luck involved. Prior projects simplified the game design to reduce the complexity, but we intentionally wanted to replicate the exact game dynamics. Since previous projects focused on using a Monte Carlo Tree Search algorithm, we wanted to try something different. We wanted to find the best possible agent, so we decided to create more than one AI agent. The game of Exploding Kittens is meant to be played with five players, and we knew we wanted to allow the user to play and wanted to have a Random Agent (as a baseline), which left us with three AI agents.

For our MLE agent, we selected a model-based approach because the transition dynamics of Exploding Kittens are highly stochastic and depend on a lot of features such as the deck composition (with changes with each turn), the interplay of action cards, and the current game status of each player. Since we decided to implement an unsimplified game environment (with serial actions, 5 players, all possible card types, etc.), it would have been intractable to manually specify accurate transitions probabilities. MLE gave us a way to estimate these transitions directly from observations, which allowed us to use value iteration and optimal planning on the MDP during training. Our MLE agent allows us to see whether model estimation is feasible in an environment with such high variance.

Our Q-Learning algorithm allows us to bypass model estimation to just direction learn action values from the highly dynamic game environment. Q-Learning is well suited for learning effective behavior without requiring an explicit model of the deck's changing composition. The game environment has hidden information (since each individual player doesn't know the other player's hands, the cards left in the deck, and the order of the remaining cards in the deck). We wanted to include a Q-Learning agent because it can handle noise outcomes well, directly optimizes action values with repeated trials, doesn't require simplifications for the game environment, and likely scales better than MLE as we increase the number of states.

A key part of the strategy of playing Exploding Kittens is to make an educated guess about the probability of drawing an Exploding Kitten card, using that guess to inform decisions about your next move, and then updating your belief about the probability of drawing an Exploding Kitten after each move. Model-free reinforcement learning alone does not explicitly keep track of beliefs over unseen deck states. Thus, we chose a Bayesian approach to incorporate a Dirichlet prior over card counts to capture beliefs about the deck's composition, a Beta-Bernoulli model to estimate explosion probability, and to use these beliefs to influence the agent's next action by adding risk and uncertainty terms to the Q-values. When we played the game with our friends, all of us were calculating the probability of drawing an Exploding Kitten to inform our strategy; we wanted to see if creating an agent that mimics a human's probabilistic reasoning would perform better than a purely value-based method.

In designing the state representation, we wanted to include all features that can affect the user's strategy, which is why we chose to include hand size, defuse count, deck size, list of players who are alive, and whether or not the next three cards in the deck are known. All these variables determine which of the possible actions are legal as well as the probability of drawing an Exploding Kitten card. Since players in the real game do not know their opponent's hands nor the full deck composition, we did not include this in the state representation. We chose to hash states into 100,000 discrete IDs as a middle ground; we figured this state space was large enough to reduce collisions and allow for a meaningful training process while still being small enough to allow for tabular methods to run within their computational limits. The action space includes all core actions fro the real game since we wanted to ensure that our agents played in a fully accurate environment without any simplifications.

The game involves both stochastic card draws and deterministic card effects. Thus, we modeled transitions probabilistically for card draws and shuffles and modeled transitions deterministically where the game rules constrain us to fixed outcomes (the Skip card ends the turn, Defuse prevents elimination, See the Future reveals the next three cards, etc.). Again, we wanted to maintain the exact game environment as closely as possible while keeping our transitions interpretable and easy to learn from simulation data. We did not simplify the multi-player interactions or eliminate serial actions, unlike previous projects, because those choices would remove important strategic dependencies of the actual game.

Our reward function was originally: +1.0 for player winning, -1.0 for player exploding, +0.1 for playing a strategic card, and -0.05 for taking a risky draw. To provide a greater incentive for winning, we changed the reward function to: -10 for losing the game, +100 for winning the game, +10 for playing a strategic card, and -0.1 for taking

a risky draw. When we trained the agents using this reward function, we found that agents almost never drew cards from the deck since there wasn't a significant reward for drawing cards. They would use up all of their cards until they ran out of cards and had to draw from the deck. Thus, we changed the reward function to reduce the relative punishment for drawing cards: +100.0 for winning, -10.0 for losing, +10 for playing a card, -0.1 for drawing a card, and +1 for survival after each turn. We found that this updated rewards function allowed the agents to meaningfully interact with the game and produced better results.

When we decided the number of training episodes for each agent, we reasoned that the MLE and Bayesian agents required less episodes because they rely on aggregated counts as opposed to Q-learning that required longer training to ensure that it could adequately explore the state space with tens of thousands of repeated state visits. We used an ϵ-greedy strategy for the Q-Learning approach to promote exploration of the highly variable environment. The Bayesian agent used an uncertainty-based exploration method to represent risk. We used a high discount factor of $\gamma = 0.99$ because most of the reward comes from the win/loss outcome, which doesn't happen until the end of the game, after a long series of actions have been chosen.

F. Evaluation

To evaluate our trained agents, we conducted head-to-head tournaments where each agent competed across 500 games with diverse opponent baselines. Our evaluation framework tracked win rates, action distributions during actual gameplay, and game length statistics. We tested each policy against four opponent types: a defensive baseline (preferring Skip over See Future over Draw), aggressive baseline (40% Attack when available, else Skip over Draw), random baseline (uniform selection over valid actions), and a varied stochastic baseline (30% Skip, 15% Attack, 10% See Future, 45% Draw). We conducted pairwise head-to-head matchups between our three learned agents (Q-Learning vs MLE, Q-Learning vs Bayesian, MLE vs Bayesian) to compare their relative performance when competing against each other rather than baselines.

III. Analysis and Results

A. Individual Agent Performance

Table III presents the performance metrics for each agent. Q-Learning had the highest win rate at 7.4%, outperforming MLE (2.6%) and Bayesian (0.8%). Q-Learning also had the most diverse action selection, using strategic cards in 22.4% of plays compared to 11.4% for MLE and 1.2% for Bayesian.

TABLE III
Agent Performance vs. Varied Baseline (500 games)

Agent	Win Rate	Avg Game Length	Strategic Cards[2]
Q-Learning	7.4%	21.6	22.4%
MLE	2.6%	21.3	11.4%
Bayesian	0.8%	20.9	1.2%

B. Action Distribution Analysis

Table IV presents the actual gameplay action distributions for each agent. Q-Learning exhibited the most strategic behavior, utilizing Skip in 10.2% of actions and distributing plays across all seven non-DRAW card types. This diversity suggests Q-Learning successfully learned to balance immediate drawing with defensive and offensive strategies. MLE showed more conservative play with 88.6% DRAW actions, while Bayesian adopted a highly risk-averse strategy with 98.8% DRAW selections.

TABLE IV
Action Distribution in Gameplay (500 games)

Action	Q-Learning	MLE	Bayesian
DRAW	77.6%	88.6%	98.8%
SKIP	10.2%	2.2%	0.5%
ATTACK	2.4%	1.5%	0.0%
SHUFFLE	5.0%	4.3%	0.0%
NOPE	2.2%	0.2%	0.7%
SEE FUTURE	0.5%	2.1%	0.0%
CAT	2.0%	1.0%	0.0%
Total Actions	1,767	1,531	1,449

C. Head-to-Head Competition

Table V presents results from pairwise matchups between agents. Q-Learning demonstrated consistent superiority in direct competition, achieving higher win rates against both MLE (20.4% vs. 19.8%) and Bayesian (22.8% vs. 16.0%). The MLE vs. Bayesian matchup showed near-equivalent performance (17.6% vs. 17.8%), indicating similar learned strategies between these two approaches. We see that it is difficult to outperform the baseline since Exploding Kittens is a luck-based game with little room for strategy.

TABLE V
Head-to-Head Results (500 games per matchup, 4 players)

Matchup	Agent A	Agent B	Baseline
Q-Learning vs. MLE	20.4%	19.8%	59.8%
Q-Learning vs. Bayesian	22.8%	16.0%	61.2%
MLE vs. Bayesian	17.6%	17.8%	64.6%

D. Multiplayer Tournament

Table VI shows results when all three agents competed simultaneously with one varied baseline opponent. Q-Learning achieved the highest win rate among learned agents at 27.0%, exceeding the theoretical random baseline of 25%. Bayesian achieved 23.6% and MLE 16.8%. Q-Learning also demonstrated the best average reward at -3.81, compared to -4.50 for Bayesian and -5.04 for MLE, indicating longer survival and more successful gameplay overall.

TABLE VI
MULTIPLAYER TOURNAMENT (500 GAMES, 4 PLAYERS)

Agent	Wins	Win Rate	Avg Reward
Q-Learning	135	27.0%	-3.81
Bayesian	118	23.6%	-4.50
MLE	84	16.8%	-5.04
Varied Baseline	163	32.6%	-

E. Performance Against Different Baselines

Table VII compares each agent's performance against three baseline opponent types. MLE achieved the highest win rate against varied opponents (19.6%), while Q-Learning performed best against defensive opponents (14.2%). All agents showed lower win rates against random opponents, with Q-Learning maintaining the highest at 6.2%.

TABLE VII
WIN RATES VS. DIFFERENT BASELINE TYPES (500 GAMES EACH)

Agent	vs. Defensive	vs. Varied	vs. Random
Q-Learning	14.2%	18.2%	6.2%
MLE	16.0%	19.6%	5.2%
Bayesian	12.8%	18.2%	5.0%

IV. DISCUSSION AND CONCLUSION

Our evaluation reveals a strong correlation between strategic action diversity and agent performance, with Q-Learning achieving 7.4% win rate and 22.4% strategic card usage compared to MLE's 2.6% win rate with 11.4% strategic actions and Bayesian's 0.8% win rate with only 1.2% strategic actions. This nearly three-fold increase in win rate from MLE to Q-Learning, and nine-fold increase from Bayesian to Q-Learning, demonstrates that learning when and how to use non-DRAW actions is critical for competitive play in Exploding Kittens.

Q-Learning's superior performance (7.4% vs 2.6% and 0.8%) can be attributed to several other factors as well. As a model-free method, Q-Learning directly learns action values without requiring accurate transition probability estimation. In a 4-player game with high stochasticity (9% bomb probability per draw), the true transition dynamics are difficult to estimate accurately. MLE's dependence on collecting sufficient (s,a,s')(s,a,s') (s,a,s') tuples means it may have undersampled critical state transitions, leading to a poorly estimated model. The Bayesian agent's extreme risk aversion (98.8% DRAW) suggests its belief-tracking mechanism over-penalized all strategic actions, likely because the Beta-Bernoulli prior combined with the risk penalty parameter created an overly conservative policy.

Our reward function and discount factor configuration significantly influenced the learned behaviors. We assigned +100 for wins, -10 for loses, +10 for strategic card play, -0.1 for risky draws, and a high discount factor of γ=0.99. This created an incentive misalignment between intermediate and terminal rewards. The discounted value of winning at average game length (21 steps) is 81 points, while playing strategic cards provides an immediate +10 reward. Thus, agents could accumulate similar Q-values through strategic card play (8 cards$\times$10 = 80 points) without necessarily learning to win. Random opponents don't present opportunities for strategic card plays in predictable patterns, which may explain why our models performed weakly against random opponents.

We encoded states using a deterministic hashing function that converts feature dictionaries (player index, hand size, defuse count, deck size, alive count, alive mask, top-3 visible cards) into unique integer state IDs. Our state representation was player-centric and did not explicitly track opponent hands or their previous actions, limiting the agents' ability to develop opponent-modeling strategies. The 2018 Exploding Kittens project used Monte Carlo Tree Search with neural network value function approximation, providing generalization across similar states. Our decision to implement full game rules (serial actions, all card types, 4 players) increased realism but introduced challenges that appear to require deep reinforcement learning rather than tabular methods.

These findings collectively suggest that successful reinforcement learning for multi-player Exploding Kittens requires either substantial game simplification (as in prior work) or more sophisticated learning approaches. Future work should consider implementing deep reinforcement learning (DQN, PPO) with neural network function approximation to handle larger state spaces without hash collisions or incorporate explicit opponent modeling similar to poker AI approaches. Rather than viewing our agents' performance as failures, we can interpret them as discovering the inherent limits of reinforcement learning in luck-dominated games. The strategic actions available in Exploding Kittens (Skip, Shuffle, See Future, and Attack) do not fundamentally alter the probability of someone drawing a bomb; they merely redistribute when and who draws it.

V. OUR WEBSITE

We built a website where a user can visualize and play Exploding Kittens against our AI agents in real-time. The frontend is built with React 18 and TypeScript. We used Convex, a severless real-time database platform, to synchronize the states and integrate the game logic. Since we used Convex, we didn't need to have manual WebSocket management. The website has automatic real-time updates to all connected clients, and server-side validation of game moves. The main interface shows player panels with the hand sizes, defuse count, and current game status of all other players. The player can choose to draw a card from the deck or select one of the cards in their current hand. The user is able to see what cards each of the AI agents choose to play when its their turn. Each player's statistics (wins, explosions, cards in hand, and defuse

card count) are also displayed. A screenshot of our website is shown below in Figure 1.

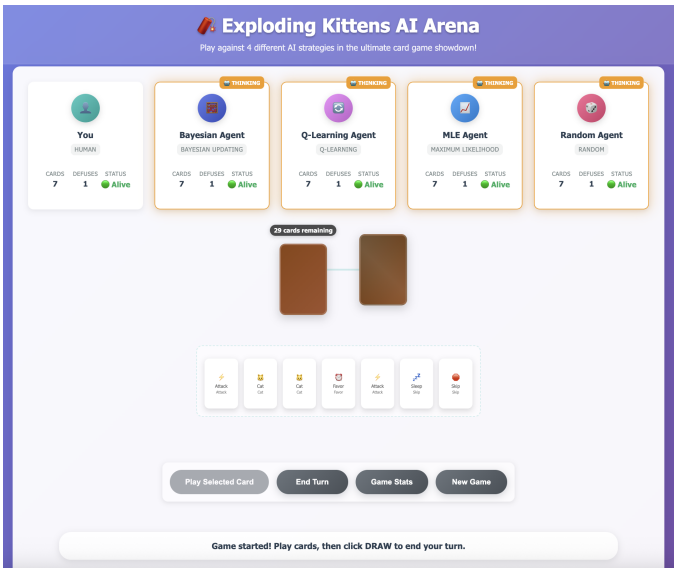

Fig. 1. Screenshot of Our Interactive Exploding Kittens Game Website

Our website is not publicly available yet, but it can be run locally on your computer by:

1) Cloning our Github Repo: git clone https://github.com/summer-royal/cs238-explodingKittens.git
2) Installing the dependencies: npm install
3) Running our web app: npm run dev

VI. Team Member Contributions

Both Summer and Shriya took the class for 4 units and contributed equally to this project. Both Summer and Shriya discussed and designed the project ideation and architecture together.

Summer was responsible for: modifying our existing Project 2 Q-Learning and MLE algorithm code to work with the project, creating the code for the Bayesian agent, creating the initial scripts to train the AI agents, creating the code for the random agent, integrating the agent policies into the website, and creating the website UI. Summer wrote the Abstract section, Background subsection, Prior Work subsection, Our Solution subsection, State and Action subsection, Transition Dynamics subsection, AI Agents subsection, and the Why We Chose This Approach subsection.

Shriya was responsible for: adding functionality to the action cards, debugging the simulation environment, reviewing and modifying the training scripts, and evaluating the performance of the AI agents. Shriya wrote the Simulation Environment subsection, Evaluation subsection, Analysis and Results section, and the Discussion and Conclusion section.

VII. AI Use

- We did all of the project ideation ourselves without using AI. We came up with the project idea and design decisions without consulting AI.
- We designed the architecture for the AI agents (MLE, Q-Learning, and Bayesian Network) by ourselves without using AI.
- We wrote the code for all three AI agents (MLE, Q-Learning, and Bayesian Network) by ourselves without using AI.
- We wrote this report ourselves without using AI.
- We used ChatGPT, DeepSeek, Poe.com, and You.com to create the interactive Exploding Kittens website. We used DeepSeek to create the code to load the agent policies into our website and the code to encode the agent states in our website. The majority of the code for the UI of our website was generated using Poe.com and ChatGPT, but we made significant modifications to fix all the bugs. We used You.com in the process of debugging.

References

[1] Exploding Kittens Inc. How to play exploding kittens (original edition). https://www.explodingkittens.com/pages/rules-kittens , 2015. Accessed: 2025-10-24.

[2] C. Jirachotkulthorn, N. Jatusripitak, Meow-kov Decisions: Reward Functions for an Exploding Kittens MDP, https://drive.google.com/file/d/1-uzz0O50A3S9WikxXWqPWwFDdl_buu2t/view?usp=sharing (accessed Nov. 29, 2025).

[3] Mykel J. Kochenderfer, Tim A. Wheeler, and Kyle H. Wray. Algorithms for Decision

[4] R. Patil, R. Silva, A. Parulekar, Reinforcement Learning for Exploding Kittens, https://web.stanford.edu/class/aa228/reports/2018/final135.pdf (accessed Nov. 29, 2025).