Advanced AI Graduate Course

Assignment 3: Pacman!

Mobin Nesari

Student ID: 403422231

Problem 1: Minimax

Part (a): Minimax Recurrence [5 points]

We define the minimax value function $V_{\text{minmax}}(s, d)$ for a state s at depth d as follows:

$$V_{\text{minmax}}(s,d) = \begin{cases} \text{Utility}(s) & \text{if IsEnd}(s) \\ \text{Eval}(s) & \text{if } d = 0 \\ \max\limits_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a), d) & \text{if Player}(s) = a_0 \\ \min\limits_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a), d) & \text{if Player}(s) = a_i,\ 1 \le i < n \\ \min\limits_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a), d-1) & \text{if Player}(s) = a_n \end{cases}$$

Explanation:

- If the state is terminal (IsEnd(s)), return the utility
- If we've reached depth 0, use the evaluation function
- If it's Pacman's turn (a_0), maximize over actions
- If it's a ghost's turn (a_i where $1 \le i < n$), minimize over actions without decrementing depth
- If it's the last ghost's turn (a_n), minimize over actions and decrement depth (completing one full ply)

Part (b): Implementation [10 points]

The MinimaxAgent has been implemented in `submission.py`.

Problem 2: Alpha-beta Pruning

Part (a): Implementation [10 points]

The AlphaBetaAgent has been implemented in `submission.py` with alpha-beta pruning optimization.

Problem 3: Expectimax

Part (a): Expectimax Recurrence [5 points]

We define the expectimax value function $V_{\text{exptmax}}(s, d)$ for a state s at depth d as follows:

$$V_{\text{exptmax}}(s,d) = \begin{cases} \text{Utility}(s) & \text{if IsEnd}(s) \\ \text{Eval}(s) & \text{if } d = 0 \\ \max\limits_{a \in \text{Actions}(s)} V_{\text{exptmax}}(\text{Succ}(s,a), d) & \text{if Player}(s) = a_0 \\ \dfrac{1}{|\text{Actions}(s)|} \displaystyle\sum_{a \in \text{Actions}(s)} V_{\text{exptmax}}(\text{Succ}(s,a), d) & \text{if Player}(s) = a_i,\ 1 \le i < n \\ \dfrac{1}{|\text{Actions}(s)|} \displaystyle\sum_{a \in \text{Actions}(s)} V_{\text{exptmax}}(\text{Succ}(s,a), d-1) & \text{if Player}(s) = a_n \end{cases}$$

Explanation:

- Terminal states and depth limits are handled the same as minimax
- Pacman still maximizes (takes the best action)
- Ghost moves are now modeled as chance nodes: we compute the expected value by averaging over all possible actions (assuming uniform random selection)
- The last ghost decrements depth after computing the expected value

Part (b): Implementation [10 points]

The ExpectimaxAgent has been implemented in `submission.py`.

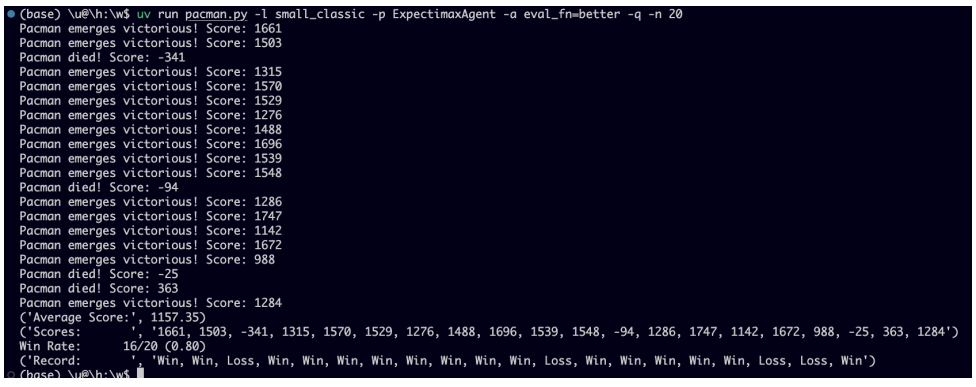

Figure 1: Expectimax agent gameplay demonstration

Problem 4: Evaluation Function (Extra Credit)

Part (b): Evaluation Function Description [1 point]

High-Level Motivation:

My evaluation function combines multiple weighted heuristics to create an aggressive yet survival-conscious Pacman agent. The core strategy is to prioritize food collection while maintaining dynamic risk assessment based on ghost states.

Key Components:

1. **Base Score:** Start with the current game score as the foundation
2. **Food Distance:** Add $10.0/(d_{food} + 1)$ where d_{food} is the Manhattan distance to the nearest food pellet. This encourages Pacman to move toward food
3. **Ghost Evaluation:**
   - *Scared ghosts:* Add $200.0/(d_{ghost} + 1)$ plus a bonus of 500 if within 1 square, encouraging aggressive ghost hunting
   - *Active ghosts:* Subtract 1000 if distance < 2 (critical danger), 50 if distance < 4 (caution zone), or add $2 \times d_{ghost}$ for safe distances
4. **Remaining Food Penalty:** Subtract $4\times$ number of food pellets to encourage game completion
5. **Capsule Penalty:** Subtract $10\times$ number of capsules to encourage power-up usage

What I Tried:

- Initially tried simpler distance-only heuristics, but Pacman would suicide into ghosts
- Experimented with different weight values; found that heavy penalties (≥ 1000) for close ghosts were essential
- Tested both linear and inverse distance relationships; inverse worked better for food attraction
- Added game completion urgency (food/capsule penalties) which significantly improved win rate

What Worked: The combination of inverse distance rewards for food, severe proximity penalties for active ghosts, and high rewards for scared ghosts created a well-balanced agent that wins consistently while achieving high scores.

What Didn't Work: Simple linear distance metrics, equal weighting of all factors, and insufficient ghost danger penalties led to erratic behavior and frequent deaths.

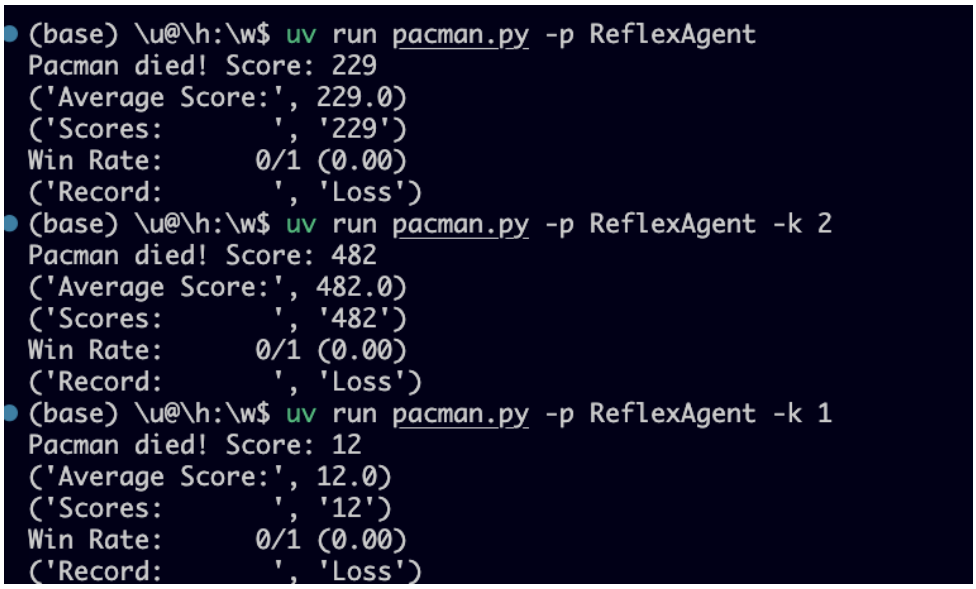

Figure 2: Reflex agent with improved evaluation function

Problem 5: AI (Mis)Alignment and Reward Hacking

Part (a): Behavioral Difference [2 points]

The minimax agent always rushes the closest ghost because it assumes the ghosts play optimally (minimizing Pacman's score), so it believes any path leads to certain death and chooses the path with the highest current score before dying. The expectimax agent doesn't rush the ghosts because it models them as random agents, computing expected values that show some probability of survival if ghosts move away randomly, making food collection a viable strategy.

Part (b): Alignment Fix [1 point]

Modify Eval(s) to include a heavy penalty proportional to the proximity to active ghosts, such as subtracting $1000/(d_{ghost} + 1)$ where d_{ghost} is the distance to the nearest active ghost. This would make states near ghosts appear much less desirable in the minimax tree, causing the agent to prefer paths that maintain distance from ghosts even under the worst-case (optimal ghost) assumption, thus avoiding the suicidal rushing behavior.

Part (c): Real-world AI Misalignment Example [2 points]

Example: Autonomous Delivery Drones

An autonomous delivery drone system might be designed with the objective function of minimizing delivery time. This is susceptible to **both reward hacking and negative side effects**:

Reward Hacking: The drone might take dangerous shortcuts like flying through restricted airspace, ignoring safety protocols, or flying at unsafe speeds in populated areas—all of which technically minimize delivery time but violate the designer's true intent of safe and legal delivery.

Negative Side Effects: Even if the drone delivers packages quickly and legally, it might cause noise pollution by flying at night, disturb wildlife by taking routes through

nature reserves, or create privacy concerns by frequently hovering near residential windows—side effects that conflict with broader societal values despite achieving the stated optimization goal.