Pac-Man AI Agent: Workshop 3

Juan David Córdoba Aguirre - 20211020097

Andres Felipe Carvajal Forero - 20201020041

Course: Systems Science Foundations

Professor: Carlos Andres Sierra Virguez

Universidad Distrital Francisco José de Caldas

Facultad de Ingeniería

Ingeniería de Sistemas

Bogotá D.C

June 2025

Abstract

This workshop develops a discrete-time decision-theoretic model for a Pac-Man agent operating under cybernetic control. The agent's state includes spatial position and ghost-mode information, while the reward and risk functions are defined using indicator-based sensory inputs and proximity-based penalties via Manhattan distance. A profit function combines these elements, allowing optimal actions to be selected through an argmax policy. The architecture supports feedback-based behavior regulation, embedding positive, negative, and adaptive loops directly into the control logic. A modular simulation design implements this structure, enabling future reinforcement learning integration. This framework formalizes real-time decision-making and adaptability through quantified sensory feedback, advancing the system's alignment with both cybernetic theory and reinforcement learning principles.

Contents

1 Introduction and Context

Ms. Pac-Man is an iconic arcade video game developed by Midway and released in 1982 as a sequel to the original *Pac-Man*. While preserving the core mechanics of its predecessor, *Ms. Pac-Man* introduced a number of enhancements that made the gameplay more dynamic, unpredictable, and challenging, thus increasing its appeal both in arcade settings and in artificial intelligence research.

The objective of the game is deceptively simple: the player controls Ms. Pac-Man through a maze, consuming small pellets (known as Pellers) scattered along the paths. Upon consuming all the Pellers on the map, the player advances to the next level. However, the maze is also inhabited by four autonomous enemy agents—ghosts—each with unique behavioral patterns. Contact with any of these ghosts results in the immediate loss of a life. To aid the player, the game includes larger dots known as Power Pellers which, when consumed, temporarily reverse the predator-prey dynamic: Ms. Pac-Man becomes capable of consuming the ghosts for extra points, while the ghosts enter a vulnerable state marked visually by a change in color to blue.

Each ghost possesses distinct behavior, making the gameplay strategically rich and requiring real-time adaptation. Below is a summary of the individual behaviors:

- **Blinky (Red)** – Known as the chaser, Blinky directly pursues Ms. Pac-Man's current location with minimal deviation. As the game progresses, Blinky increases in speed, becoming especially dangerous in later levels.
- **Pinky (Pink)** – Attempts to ambush Ms. Pac-Man by predicting her future position based on current direction, typically targeting four tiles ahead.

- **Inky (Cyan)** – Exhibits more complex behavior, combining Blinky's and Ms. Pac-Man's positions to generate a vector-based target location, leading to unpredictable and difficult-to-counter movements.

- **Sue (Orange)** – Functions similarly to the ghost Clyde in the original game, alternating between pursuing Ms. Pac-Man when far and retreating to a home corner when close, giving her a misleading and seemingly random pattern.

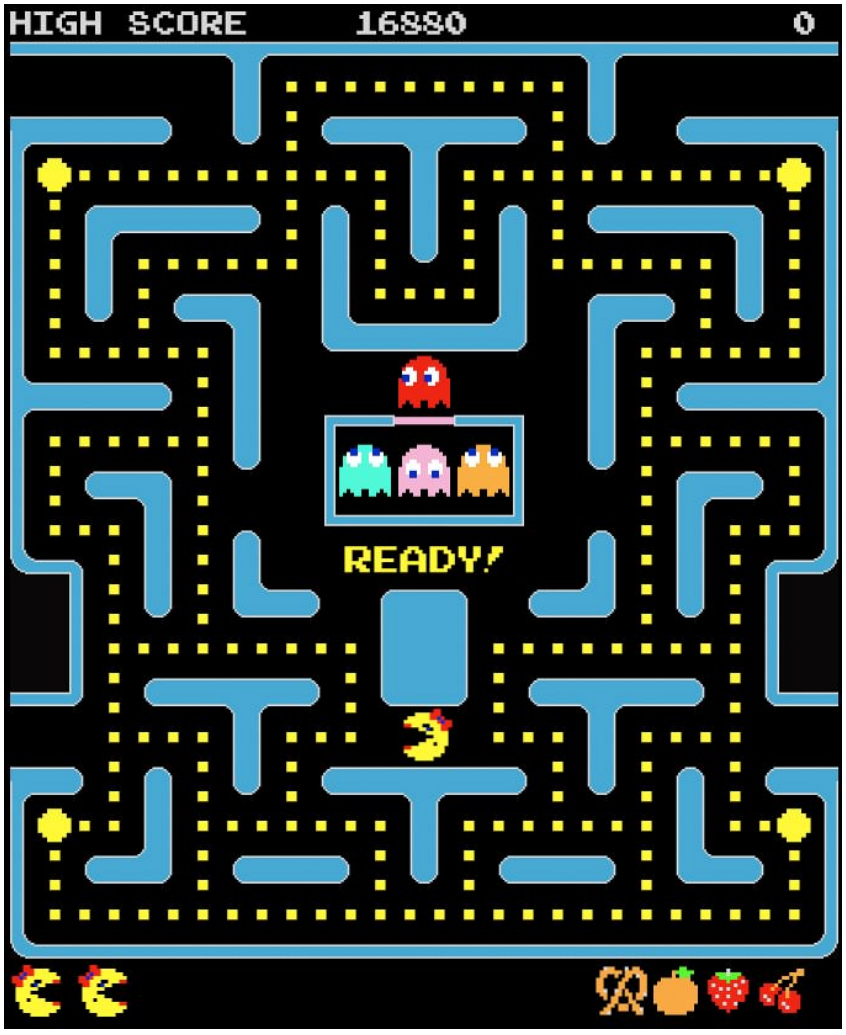

Fig. 1. Gameplay interface of Ms. Pac-Man – original arcade version, adapted to illustration purposes

2 Functional Specifications

In this project, the Pacman environment is recreated as a complex system in which artificial intelligence, using Deep Learning techniques, controls Pacman and makes decisions in real-time. Just as in nature the whole is far more than the sum of its parts, in this system each component interacts synergistically, allowing the agent to learn and adapt autonomously in a dynamic and challenging environment.

2.1 Map

The game map is defined as a vast space composed of multiple quadrants, each bounded by walls that constitute the physical structure of the maze. These quadrants, interconnected by lateral passages, form a network that invites deep systemic analysis. The direction-changing "nodes," strategically distributed across the map, not only allow Pacman to change course but also serve as critical points that ghost AIs use to optimize their navigation and pursuit. This system of nodes and corridors echoes the inherent complexity of spatial organization in natural systems, where every change of direction can be interpreted using Deep Reinforcement Learning algorithms, which seek to learn patterns and regulate the agent's behavior in an ever-changing environment.

2.2 Pellers

Scattered throughout the maze paths are the so-called "Pellers," points that Pacman must consume to accumulate score. These seemingly simple items are an essential part of a system of positive stimuli that encourage exploration and decision-making in the agent. Much like

in biological systems, where small signals build into complex feedback mechanisms, each Peller represents an immediate reward that, once consumed, reinforces the agent's desired behavior. Additionally, the occasional presence of Power Pellers introduces a highly beneficial and disruptive element: upon consumption, Pacman gains the ability to temporarily reverse the game's dynamic, allowing him to consume the ghosts. This transformation can be approached using Deep Learning models that integrate deep neural networks for pattern recognition in the environment and behavior adaptation based on accumulated rewards.

2.3 Pacman (Controlled by AI)

The central character, Pacman, is controlled by an AI that seeks to optimize its path and maximize point collection, all while avoiding the constant threat of ghosts. The challenge in designing an agent that operates in such a complex environment lies in its ability to learn from experience and adapt to unforeseen situations, just as in biological systems. In this context, the implementation of Deep Reinforcement Learning algorithms allows the agent to refine its movement and decision strategies through multiple iterations, learning to balance the exploration and exploitation of resources available in the maze. This task demands a harmonious integration of traditional control techniques with advanced methods of deep learning, giving rise to a truly adaptive system.

2.4 Ghosts

The ghosts, machine-controlled enemies, add an additional layer of complexity, as they operate under three distinct states:

- **Chase:** They actively pursue Pacman when proximity is detected, using algorithms that may incorporate pattern recognition and tracking techniques based on neural networks.

- **Scatter:** They abandon the chase and retreat to a designated area, simulating behaviors found in complex systems resembling cycles of negative feedback.

- **Frightened:** After the activation of a Power Peller, the ghosts change state, becoming vulnerable to being consumed. This shift in dynamic can be modeled using transfer learning techniques, adjusting neural network parameters to accurately interpret new environmental stimuli.

Each of these states not only defines the individual behavior of the ghosts but also influences the global strategy of the agent, which must learn to interpret signals and make quick decisions to maximize rewards and minimize penalties. Implementing these behaviors through Deep Learning models enables the realistic simulation of the complex and adaptive dynamics of the game environment.

3 System Dynamics Analysis

3.1 Simplified Mathematical Model of the Ms. Pacman Agent

In this section, we introduce a simpler discrete-time mathematical formulation of the Pac-Man environment suitable for implementing tabular or function-approximated reinforcement learning methods. This model retains essential dynamical features from Workshop 2, but reduces complexity to enable tractable computation and analysis.

3.1.1 State and Control Representation

At each discrete time step k, the agent observes a simplified game state given by:

$$s(k) = \left(x_M, \{(x_g, m_g)\}_{g=1}^{N} \right) \tag{1}$$

where:

- $x_M \in \mathbb{Z}^2$ is Pac-Man's grid position.
- $x_g \in \mathbb{Z}^2$ is the position of ghost g, each ghost position is evaluated in the state.
- $m_g \in \{0, 1\}$ is the mode of ghost g, with $m_g = 1$ indicating a dangerous ghost and $m_g = 0$ representing a frightened ghost.

3.1.2 Reward Function

Here, each $\{D, P, F\}$ is an indicator function that returns 1 if the specified tile contains the corresponding stimulus and 0 otherwise.

$$V\big[s(k), u_M(k)\big] = \omega_d \cdot D(x_M + u_M(k)) + \omega_p \cdot P(x_M + u_M(k)) + \omega_f \cdot F(x_M + u_M(k)) \quad (2)$$

The agent receives rewards based on the target tile:

$$V(s(k), u_M(k)) = \begin{cases} +10, & \text{if pellet at } x_M + u_M(k) \\ +50, & \text{if power pellet at } x_M + u_M(k) \\ +200, & \text{if frightened ghost at } x_M + u_M(k) \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

3.1.3 Risk Function

Risk is modeled using a discrete repulsive potential field centered at each ghost's location:

$$R(s(k), u_M(k)) = \sum_{j=1}^{N} m_j \cdot \begin{cases} \left(\frac{1}{\rho_j} - \frac{1}{\rho_0}\right)^2, & \text{if } \rho_j \leq \rho_0 \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where ρ_j is the Manhattan distance:

$$\rho_j = \|x_M + u_M(k) - x_j\|_1 \quad (5)$$

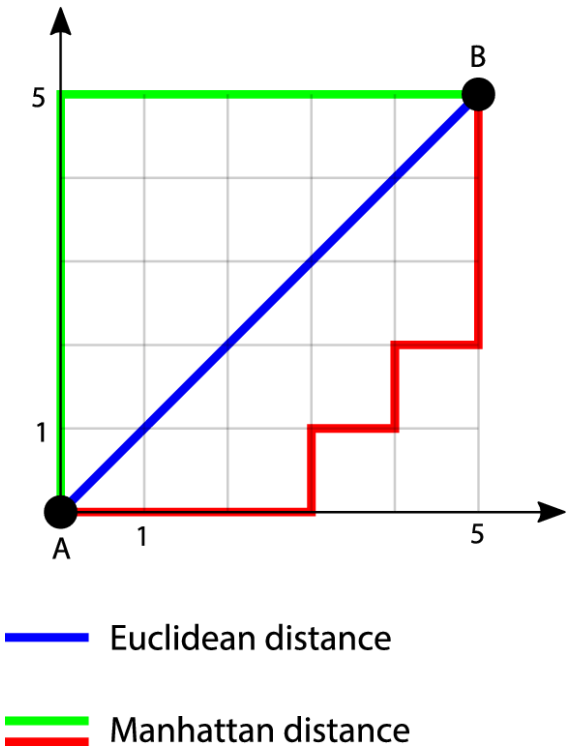

Fig. 2. Manhattan Distance graphic interpretation

3.1.4 Profit and Optimal Decision

The utility of each action is:

$$L(s(k), u_M(k)) = V(s(k), u_M(k)) - \lambda \cdot R(s(k), u_M(k)) \tag{6}$$

with $\lambda > 0$ controlling risk aversion.

The decision rule is:

$$u_M^*(k) = \arg\max_{u \in U} L(s(k), u) \tag{7}$$

Admissible actions:

$$U = \{a, b, c, d\} \quad \text{with} \quad a = \begin{bmatrix} 0 \\ \\ +1 \end{bmatrix},\ b = \begin{bmatrix} 0 \\ \\ -1 \end{bmatrix},\ c = \begin{bmatrix} -1 \\ \\ 0 \end{bmatrix},\ d = \begin{bmatrix} +1 \\ \\ 0 \end{bmatrix} \tag{8}$$

3.1.5 Interpretation

This model offers a simplified yet expressive control formulation that retains essential systemic behaviors from the original dynamical model.

3.1.6 Verification

We verify that the discrete-time dynamics (1), reward aggregation (2), and risk definitions (4) jointly satisfy the instantaneous profit (6), and that maximizing $\sum_k L$ via our strategy yields stable simulation trajectories.

3.1.7 Phase Portraits and State-Space Analysis

Phase portraits or diagrams illustrate how the agent's state evolves under varying inputs, highlighting attractors and potential chaotic regimes. By plotting trajectories in relevant state-space dimensions (e.g., position x_M versus velocity $\dot{x}_M$, or reward accumulation versus time), one can identify stable attractors corresponding to recurrent behaviors (such as sustained pellet-chasing loops) and observe sensitivity to initial conditions when interacting with ghosts. These diagrams reveal basins of attraction, transition boundaries between modes of operation, and regions where the learned policy exhibits robust stability or chaotic responses. Such visual analysis deepens our understanding of the agent's adaptability and the resilience of the reinforcement-driven control strategy.

3.2 Simulation Architecture Based on the Mathematical Model

To implement the mathematical model presented as a working simulation, we construct a discrete-time simulation architecture that reflects the decision-making process, dynamical evolution, and reward-risk structure of the Ms. Pacman agent. This simulation integrates control-theoretic dynamics, reinforcement-driven profit maximization, and environmental state evolution using a modular programming framework.

3.2.1 Simulation Modules and Structure

The simulation comprises the following core modules:

- **State Manager:** Tracks the global state $s(k)$, which includes positions of the agent and ghosts, remaining Pellers, ghost modes, and score.
- **Controller:** Implements the decision strategy $\sigma = \{c_0, c_1, \ldots\}$ as a mapping $u_M(k) = c_k[s(k)]$ based on maximizing cumulative profit.
- **Dynamics Engine:** Applies the discrete-time state update equations for Ms. Pacman and ghost agents based on Eq. (**??**).
- **Reward and Risk Evaluators:** Evaluate the reward V and risk R terms using Eq. (2) and the ghost repulsion model.
- **Simulation Kernel:** Executes the simulation loop over discrete time k, updates states, applies control, and accumulates performance metrics.
- **Visualization Tools:** Generate trajectory plots, heatmaps, and phase diagrams to support interpretability and verification.

3.2.2 Simulation Architecture Based on the Simplified Model

The following simulation architecture implements the simplified decision-theoretic model presented earlier, and serves as a functional abstraction of Pac-Man's behavior using discrete-time control.

State Representation. The game state is composed of Pac-Man's position, a list of dangerous ghosts with their respective modes, and sets for pellets and power pellets. Ghosts are defined as tuples (x_j, m_j) where $m_j \in \{0, 1\}$ indicates frightened or dangerous mode. This is implemented in Python as:

```
pacman = [5, 5]
pellets = [[6, 5]]
power_pellets = [[5, 6]]
frightened_ghosts = [[4, 5]]
ghosts = [([7, 5], 1)]  # (position, mode)
```

Action Space. Pac-Man can move in one of four directions per timestep. Actions are defined as grid movement vectors:

```
ACTIONS = {
    "UP":    [0, 1],
    "DOWN":  [0, -1],
    "LEFT":  [-1, 0],
    "RIGHT": [1, 0]
}
```

Reward Function. A reward is assigned based on what object lies in the target tile. The function prioritizes frightened ghosts, then power pellets, and finally regular pellets:

```
def reward(position):
    if position in frightened_ghosts:
        return 200
    elif position in power_pellets:
        return 50
    elif position in pellets:
        return 10
    else:
        return 0
```

Risk Function. The risk function accumulates a repulsive potential from each dangerous ghost if its Manhattan distance to the target tile is within a threshold ρ_0:

```
def risk(position):
    total_risk = 0
    for ghost_pos, mode in ghosts:
        if mode == 1:
            distance = manhattan(position, ghost_pos)
            if 0 < distance <= rho_0:
                total_risk += (1 / distance - 1 / rho_0) ** 2
    return total_risk
```

Decision Rule. For each admissible action, the simulation computes the expected reward, risk, and total profit $L = V - \lambda R$, selecting the optimal move via argmax:

```
best_action = None
best_score = float('-inf')

for name, move in ACTIONS.items():
    target = add(pacman, move)
    v = reward(target)
    r = risk(target)
    score = v - lambda_risk * r
    if score > best_score:
        best_score = score
        best_action = name

print("Best action:", best_action)
```

4 Cybernetic Regulation

4.1 Feedback Loop Refinement Summary

At the core of the system lies a high complexity mechanism of stimuli and rewards. On one hand, there are positive reinforcers, such as the consumption of Pellers, Power Pellers, and the opportunity to eliminate ghosts during the "Frightened" state. These stimuli encourage behaviors that lead to efficient point collection and are central to reinforcement learning processes. On the other hand, negative stimuli—such as ghost contact or excessive inactivity—penalize the agent and help sustain continuous engagement with the environment.

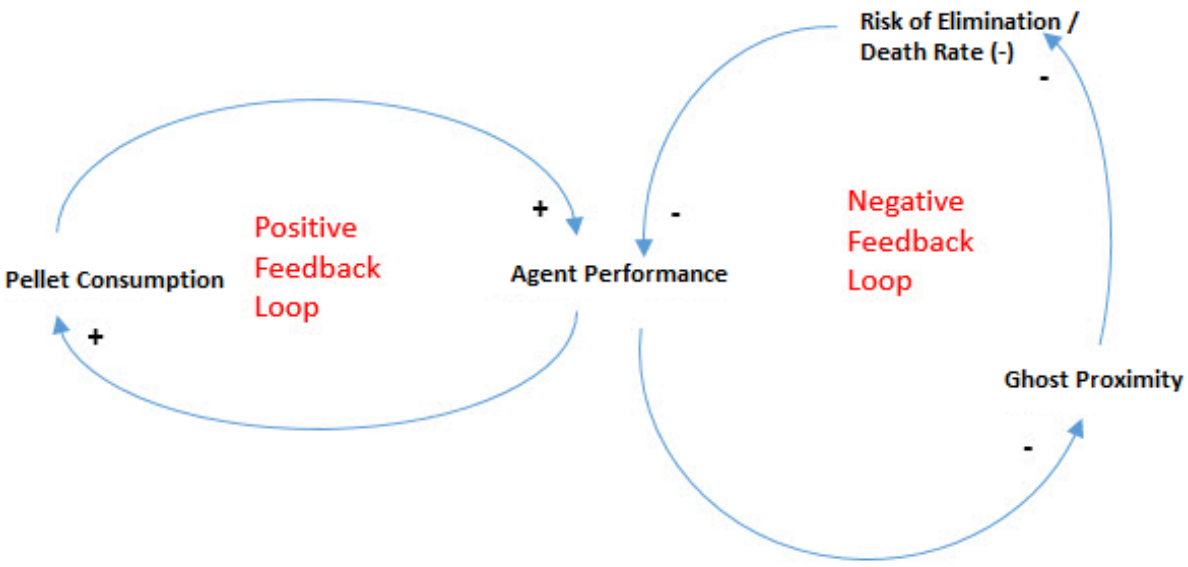

Fig. 3. Pac-Man continuous feedback loop.

In earlier iterations, feedback loops were qualitatively categorized into three types: positive, negative, and adaptive. Theoretically, these loops would align environment interaction with the agent and stabilize its learning path. However, the current formulation refines

these mechanisms by aligning them with the new mathematical model and embedding them directly into the reward-risk model.

- **Refined Positive Feedback:** Positive outcomes are encoded through discrete and weighted reward functions that prioritize high-value stimuli. Their impact is now modeled via instantaneous profit computations that scale with the frequency and intensity of resource consumption, such as frightened ghost captures or clustered pellet zones.

- **Quantified Negative Feedback:** Risk is no longer conceptual but quantified through repulsive potential functions based on proximity to dangerous ghosts. The use of bounded inverse-distance formulations introduces a smooth penalty surface that intensifies as the agent approaches high-risk zones, enforcing real-time spatial awareness.

- **Dynamic Adaptive Feedback:** Awareness control now adapts to ghost behavioral modes (chase, scatter, frightened) through binary flags $m_g \in \{0, 1\}$ that modulate risk contributions. This dynamic modulation directly influences the agent's policy, allowing it to shift between aggressive and evasive strategies based on sensory cues.

4.2 Cybernetic Feedback Control Architecture

The diagram above illustrates a closed-loop control structure that governs Pac-Man's behavior through sensor-based evaluation and decision-making. This architecture embeds cybernetic principles into a discrete-time reinforcement learning model, allowing the agent to adapt its behavior based on environmental stimuli and threats.

At the core of the control mechanism is the profit function defined in Equation (6), which computes the utility of each potential action by balancing expected rewards (Equation (2))

against spatially-aware risk penalties (Equation (4)). The metric used to assess proximity to threats is the Manhattan distance, as described in Equation (5).

Given this formulation, the control policy at time k selects the optimal action through the maximization rule specified in Equation (7), using the finite action space declared in Equation (8).

Once an action is executed, the agent transitions to a new state as defined by the structure in Equation (1). This updated state reflects changes in Pac-Man's position, as well as the positions and behavioral modes of ghosts.

Feature extraction occurs in this new state to retrieve relevant sensory data:

- Local tile content (pellets, power pellets, frightened ghosts)
- Threat proximity ρ_j and mode m_j

These observations are used to recompute the reward and risk components of $L(s(k), u)$ for the next decision step.

Fig. 4. Refined Feedback Loop

Feedback Mechanisms:

- **Positive Feedback Loop:** When the agent consumes pellets or ghosts, the reward function increases. This reinforces high-performing decisions and increases their likelihood of repetition, forming a positive reinforcement cycle through the V term in Equation (2).

- **Negative Feedback Loop:** When ghost proximity increases, risk becomes significant

through Equation (4), thereby lowering the profit function. This inhibits dangerous behaviors, encouraging safer policies based on real-time sensory inputs.

- **Cybernetic Integration:** The entire control cycle—from observation to profit evaluation to action—is recalculated each step. This forms a complete cybernetic feedback loop, where each action feeds into environmental responses, and those responses shape the next action.

This architecture establishes the agent as a decision-making system driven by quantified feedback. It maintains cybernetic stability while optimizing task-specific performance using the defined mathematical structure.

5 Machine Learning Algorithms and Frameworks

The simplified mathematical formulation defines the foundation for implementing reinforcement learning within a cybernetic control system. We now identify appropriate learning algorithms and frameworks to connect the integration of this model into a full simulation environment.

5.1 Algorithmic Foundation

The agent evaluates each action $u \in U$ at time step k by computing a profit signal $L(s(k), u)$, as defined in (6). This profit function aggregates a reward signal $V(s(k), u)$ (2) and a spatial risk signal $R(s(k), u)$ (4), using risk distance metrics described in (5).

To select the optimal control, the agent applies the maximization strategy shown in

(7). This corresponds directly to the greedy policy used in classical off-policy methods like Q-learning, where the Q-value is updated iteratively using Bellman backups.

5.2 Selected Algorithms

Q-Learning (Tabular) is a suitable choice for this simulation due to the discrete nature of the state and action spaces. The agent maintains a table of $Q(s, u)$ values that represent the expected return for taking action u in state s. This approach aligns directly with the structure of the profit-based evaluation in Equation (6). Tabular Q-learning is computationally efficient and interpretable, making it ideal for validating reward-risk tradeoffs during early stages.

Deep Q-Networks (DQN) will be considered for future iterations where the state space grows significantly (e.g., pixel-based input). DQN approximates the Q-function using a neural network, allowing the agent to generalize over large or continuous input spaces. While not implemented in this phase, the architecture and reward shaping remain compatible.

Selected Frameworks

OpenAI Gym will serve as the simulation interface. The Pac-Man environment will be developed as a custom Gym environment that exposes a standard API (‘reset()‘, ‘step(u)‘) and encapsulates the logic for computing $s(k)$, $V(s, u)$, $R(s, u)$, and $L(s, u)$. This framework provides compatibility with reinforcement learning libraries and simplifies integration with learning agents.

PyTorch is chosen for potential future implementation of neural networks (e.g., in DQN). It offers dynamic graph construction, strong debugging support, and integrates seamlessly

with Gym environments. PyTorch will be the primary backend for representing and updating approximated Q-functions once neural architectures are introduced.

OpenAI Retro Gym may be used in more advanced phases to support emulation of NES-style Ms. Pac-Man environments. When combined with Gym, Retro allows training agents directly on real game ROMs, with visual input pipelines. Although this framework introduces higher dimensional input, it remains compatible with the cybernetic feedback model through visual-to-state preprocessing.

6 Testing and Metrics

6.1 Experimental Setup and Evaluation Plan

The experimental setup is designed to validate the behavior of the reinforcement learning agent under the cybernetic control framework defined in previous sections. The experiments will focus on assessing how the agent responds to reward-risk tradeoffs and environmental variability using the discrete-time formulation, particularly the profit maximization model in (6).

6.1.1 Environment Configuration

The experiments will be conducted using a custom Pac-Man environment implemented in OpenAI Gym. The environment simulates a grid-based maze containing:

- Static walls and traversable paths
- Regular Pellers and Power Pellers

- Ghost agents with dynamic behavior and state transitions ($m_g \in \{0, 1\}$)

The initial position of Pac-Man (x_M), the positions and modes of ghosts (x_g, m_g), and pellet distributions will be randomized using fixed random seeds to ensure reproducibility.

6.1.2 Test Scenarios

To systematically assess the model, a variety of controlled simulation scenarios will be defined:

- **Dense reward zone:** multiple pellets clustered near ghost paths
- **High-risk corridors:** narrow passages with frequent ghost presence
- **Sparse map:** low pellet density requiring strategic exploration
- **Mode shift test:** ghost mode changes triggered mid-episode to observe adaptive response

Each scenario will be tested across multiple episodes to capture behavioral consistency.

6.2 Performance Metrics

The agent's effectiveness will be evaluated using a set of quantitative metrics derived from the output of the profit function $L(s(k), u)$ and its components:

- **Cumulative reward:** total $V(s, u)$ accumulated over each episode
- **Cumulative risk:** aggregated $R(s, u)$, indicating exposure to threats
- **Net profit:** the difference $L(s, u)$ over time, indicating reward-risk balance

- **Policy stability:** variation in selected actions per state across multiple episodes
- **Convergence behavior:** whether cumulative metrics stabilize across trials

6.2.1 Evaluation Protocol

Each test configuration will be run over a fixed number of episodes (e.g., 50–100) using consistent seed initialization. Output metrics will be recorded per episode, then aggregated to compute mean and variance.

To complement quantitative evaluation, qualitative behavior (e.g., route choice, ghost avoidance, pellet prioritization) may be visualized in future simulation renderings. These visual traces will help diagnose whether the agent's actions align with the intended cybernetic feedback structure.

7 Multi-Agent Extension

Despite being out of our project focus, a compelling extension involves modeling ghost agents as autonomous decision-makers governed by cybernetic feedback principles. Instead of relying on fixed scripted behaviors, ghosts would become learning-based agents capable of coordinating their actions to pursue Pac-Man in a dynamic and adaptive manner.

7.1 Ghost Agent Modeling

Each ghost G_i for $i = 1, \ldots, N$ is modeled with its own internal control policy π_i, which maps a local observation $s_i(k)$ to an action $u_i(k)$. The state $s_i(k)$ can include:

- Pac-Man's position $x_M(k)$
- Ghost's own position $x_i(k)$
- Positions of nearby ghost teammates $\{x_j(k)\}_{j \neq i}$
- Maze features (walls, tunnels, power pellets)

The ghost's control objective can be framed as minimizing a cost function:

$$J_i[k] = \sum_{t=k}^{F} \gamma^{t-k} \cdot C_i(s_i(t), u_i(t)) \tag{9}$$

where γ is a discount factor and C_i is a cost function encoding pursuit goals and interaction constraints.

7.2 Cooperative Ghost Behavior

To implement cooperation, the cost function C_i may include a coordination term:

$$C_i = \alpha \cdot \rho(x_i, x_M) + \beta \cdot \Phi_i(x_i, \{x_j\}_{j \neq i}) \tag{10}$$

Here:

- $\rho(x_i, x_M)$ is the distance to Pac-Man (shorter is better)
- Φ_i is a ghost distribution potential that encourages flanking or perimeter strategies
- $\alpha, \beta > 0$ are tunable weights

This formulation allows ghosts to spread out to intercept paths or block escape routes while reducing redundancy.

7.3 Competitive Ghost Behavior

In competitive setups, each ghost seeks to be the one who captures Pac-Man. This can be modeled by assigning exclusive reward:

$$R_i = \begin{cases} +1000, & \text{if } x_i(k) = x_M(k) \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

This competitive dynamic leads to ghost behaviors that optimize solo paths, potentially leading to suboptimal team behavior if not coordinated. Balancing this reward with a shared component (e.g., time to first capture) can prevent degenerate outcomes.

7.4 Cybernetic Loop for Ghost Agents

Each ghost agent may adopt a control strategy of the form:

$$u_i^*(k) = \arg\min_{u \in U} [C_i(s_i(k), u)] \tag{12}$$

This mirrors the decision rule used for Pac-Man (Equation (7)), but reverses the optimization objective (cost minimization instead of profit maximization). The ghost agent's sensory loop thus becomes:

State Observation → Cost Evaluation → Action Selection → State Transition

Potential Learning Approaches

Ghost agents could be trained using:

- **Multi-agent reinforcement learning (MARL)**: with decentralized policies and shared replay buffers
- **Centralized Training with Decentralized Execution (CTDE)**: for cooperative flanking strategies
- **Inverse RL or imitation**: to mimic canonical ghost patterns with variation

References

- Corbett, R. (2021). Understanding feedback loops in reinforcement learning. *Towards Data Science*. Retrieved from https://towardsdatascience.com/understanding\protect\penalty\z@-feedback-loops-in-reinforcement-learning-d1bdbf12ddbd

- Foderaro, G., Swingler, A., & Ferrari, S. (2017). A model-based approach to optimizing Ms. Pac-Man game strategies in real time. *IEEE Transactions on Computational Intelligence and AI in Games, 9*(2), 153–163. Retrieved from https://lisc.mae.cornell.edu/LISCpapers/TCIAGGregModelBasedPacmanJune2017.pdf

- Gupta, A. (2020). Feedback loops in AI systems. *Microsoft Research Blog*. Retrieved from https://www.microsoft.com/en-us/research/blog/feedback-loops-in-ai-systems/

- OpenAI. (2023). Gym documentation. *OpenAI Gym*. Retrieved from https://www.gymlibrary.dev/

- OpenAI. (2023). Retro documentation. *GitHub - openai/retro*. Retrieved from https://github.com/openai/retro

- PyTorch Team. (2024). PyTorch documentation. *PyTorch.org*. Retrieved from https://pytorch.org/docs/stable/index.html

- ResearchGate. (2019). Example of Euclidean and Manhattan distances between two points A and B. In *ResearchGate Figure 8*. Retrieved from https://www.researchgate.net/figure/Example-of-Euclidean-and-Manhattan-distances-between-two-points\protect\penalty\z@-A-and-B-The-Euclidean_fig8_333430988