Pac-Man AI Agent: Workshop 2

J. D. Córdoba Aguirre, A. F. Carvajal Forero
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia
May 2025

***Abstract*—This document presents a staged development roadmap for building an intelligent reinforcement learning (RL) agent capable of playing Ms.Pacman using deep learning and cybernetic feedback principles. The project begins with foundational RL theory and proceeds through tabular Q-learning, function approximation, and Deep Q-Network (DQN) implementations using PyTorch and OpenAI Gym. Subsequent stages introduce stability-enhancing techniques such as experience replay and target networks, along with structured testing and formal evaluation methodologies. The roadmap culminates in the integration of advanced DQN variants and adaptive dynamic behaviors, including modular policy networks, context-enriched state encoding, and real-time reward modulation. Testing protocols include deterministic simulations and scenario variations to assess robustness and convergence. This structured approach ensures both the theoretical soundness and practical performance of an autonomous, adaptive agent operating in a complex, high-dimensional environment.**

***Index Terms*—Reinforcement learning, Pac-Man, system dynamics, feedback control, simulation**

I. System Dynamics Analysis

A. Mathematical Model of the Ms. Pacman Agent

In this section we formulate, analyse, and verify a mathematical model that captures the behavior of our autonomous Ms. Pacman agent, integrating the systemic elements (map, Pellers, ghosts) and reinforcement-driven decision-making framework described above.

1) Problem Formulation: We view the Ms. Pacman player as a *decision maker* that, at each discrete time step k, observes a fully observable game state $s(k) \in X \subset \mathbb{R}^n$ and chooses a control action $u_M(k) \in U(k)$ to maximize cumulative profit over a finite horizon. Time is indexed $k = 0, 1, \dots, F$, where F is unknown—the game ends when all lives are lost or all Pellers are consumed. A strategy $\sigma = \{c_0, c_1, \dots\}$ is a sequence of mappings

$$u_M(k) = c_k\big[s(k)\big], \qquad c_k\big[s(k)\big] \in U(k).$$

At each decision epoch the *instantaneous profit* combines expected reward V and risk R via

$$L\big[s(k), u_M(k)\big] = w_V\, V\big[s(k), u_M(k)\big] + w_R\, R\big[s(k), u_M(k)\big], \tag{1}$$

where $w_V, w_R > 0$ are user-tunable weights. The objective is to find

$$J_{i,f}\big[x(i), \sigma\big] = \textstyle\sum_{k=i}^{f} L\big[s(k), u_M(k)\big], \qquad \sigma^* = \arg\max_\sigma J_{i,f}\big[x(i), \sigma\big]. \tag{2}$$

This formulation embeds our systemic description of the maze (quadrants, walls, nodes, tunnels) and stimuli (Pellers, Power Pellers, ghost touch) within a rigorous decision-theoretic framework.

2) Agent and Ghost Dynamics: **Continuous-time dynamics.** In continuous time t, Pacman's position is

$$x_M(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \in \mathbb{R}^2,$$

in an inertial frame attached to the lower-left of the maze. Its motion obeys a linear ODE:

$$\dot{x}_M(t) = A(t)\, x_M(t) + B(t)\, u_M(t), \tag{3}$$

where $u_M(t) \in \{-1, 0, 1\}^2$ encodes movement commands.
Discrete-time approximation. Using integration step $\delta t \ll \Delta t$, we obtain

$$x_M(k) = x_M(k-1) + \alpha_M(k-1)\, u_M(k-1)\, \delta t, \tag{4}$$

where $\alpha_M(k)$ is Pacman's speed (bounded by ν and modulated by ghost modes and Peller consumption). Control is then

$$u_M(k) = c_k\big[x_M(k)\big],$$

a state-feedback law derived from our decision-theoretic strategy.

Similarly, each ghost $G \in \{R, B, P, O\}$ has state $x_G(k) \in \mathbb{R}^2$ and obeys

$$x_G(k) = x_G(k-1) + \alpha_G(k-1)\, u_G(k-1)\, \delta t,$$

with mode-dependent speed α_G and admissible action set $U_G(k)$ determined by maze geometry and pursuit/scatter/frightened rules.

B. Profit Function: Rewards and Risks

We decompose profit into *reward* V and *risk* R components.

a) Reward.: Let $d(\cdot)$, $p(\cdot)$, and $f(\cdot)$ denote immediate rewards for consuming a Peller, a Power Peller, and a frightened ghost, respectively. With weights $\omega_d, \omega_p, \omega_f$,

$$V\big[s(k), u_M(k)\big] = \omega_d\, d\big[s(k), u_M(k)\big] + \omega_p\, p\big[s(k), u_M(k)\big] + \omega_f\, f\big[s(k), u_M(k)\big]. \tag{5}$$

For example,

$$d\big[x(k), u_M(k)\big] = \begin{cases} 1, & \text{if a Peller at } x_M(k) \text{ is eaten,} \\ 0, & \text{otherwise.} \end{cases}$$

b) Risk.: The total risk is

$$R\big[x(k), u_M(k)\big] = \textstyle\sum_G R_G\big[x(k), u_M(k)\big], \tag{6}$$

In chase or scatter mode each R_G derives from a repulsive potential

$$U_G(x) = \begin{cases} \left(\frac{1}{\rho_G(x)} - \frac{1}{\rho_0}\right)^2, & \rho_G(x) \le \rho_0, \\ 0, & \rho_G(x) > \rho_0, \end{cases} \qquad R_G = U_G(x)$$

while in frightened mode $R_G = 0$, and upon tile overlap the capture risk is a constant χ.

1) Verification: We verify that the discrete-time dynamics (4), reward aggregation (5), and risk definitions jointly satisfy the instantaneous profit (1), and that maximizing $\sum_k L$ via our decision-theoretic strategy yields sensible, stable trajectories in simulation. Numerical validation against established model-based ghost dynamics shows accurate prediction and high game-score performance, confirming model fidelity.

2) 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.

C. 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.

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. (4).
- **Reward and Risk Evaluators:** Evaluate the reward V and risk R terms using Eq. (5) 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.

2) Simulation Loop: Pseudocode and Implementation: The following code is a small implementation of the simulation.

```
def simulate(environment, agent,
↪   max_steps):
    s = environment.reset()
    total_profit = 0
    for k in range(max_steps):
        u = agent.decide(s)
        s_next, reward, risk =
        ↪   environment.step(u)
        profit = w_V * reward + w_R * risk
        agent.learn(s, u, profit, s_next)
        s = s_next
        total_profit += profit
        if environment.is_terminal(s):
            break
    return total_profit
```

3) Reward and Risk Functions: Reward is defined through the weighted sum:

$$V\left[s(k), u_M(k)\right] = \omega_d \cdot d[s(k), u_M(k)] + \omega_p \cdot p[s(k), u_M(k)] + \omega_f \cdot f[s(k), u_M(k)], \tag{7}$$

and implemented as:

```
def compute_reward(state):
    r = 0
    if state.peller_eaten: r += omega_d
    if state.power_peller_eaten: r +=
    ↪   omega_p
    if state.ghost_eaten: r += omega_f
    return r
```

Risk derives from the ghost proximity model:

```
def compute_risk(state):
    risk = 0
    for ghost in state.ghosts:
        if ghost.mode in ["chase",
        ↪   "scatter"]:
            rho =
            ↪   np.linalg.norm(state.pacman_pos
            ↪   - ghost.pos)
            if rho <= rho0:
                risk += (1/rho - 1/rho0)**2
    return risk
```

4) State Evolution and Control Law: We approximate continuous-time dynamics by their discrete-time counterpart:

$$x_M(k) = x_M(k-1) + \alpha_M(k-1) \cdot u_M(k-1) \cdot \delta t, \tag{8}$$

which is implemented as:

```python
def update_position(x_prev, u, alpha, dt):
    return x_prev + alpha * u * dt
```

The agent's control law is state-feedback:

```python
def decide(state):
    return argmax_a (compute_reward(state)
    ↪  - compute_risk(state))
```

II. Feedback Loop Refinement

A. Feedback Refinement

To improve the agent's responsiveness and learning efficiency, the feedback loop is refined through two key enhancements: more expressive perception modules and adaptive reward granularity. These improvements align the cybernetic control structure with the dynamic nature of the environment:

1) Enhanced Perception.: The perception stage is extended by incorporating additional sensors capable of capturing fine-grained spatial and temporal information. These include indicators of ghost density, distance to the nearest Peller and Power Peller, elapsed time since the last power-up, and simple predictive signals based on ghost trajectories. This augmented sensory layer allows the agent to interpret short-term risks and opportunities more accurately, improving real-time policy responsiveness.

2) Granular Reward Signals.: The reward system is adapted to vary dynamically based on environmental context. Rather than issuing uniform rewards, the system adjusts incentives according to conditions such as ghost proximity, clustering patterns, and urgency of escape. For instance, the reward for consuming a Power Peller may increase when multiple ghosts are nearby, encouraging preemptive and strategic activation. This nuanced feedback reinforces context-aware behaviors and facilitates more efficient learning.

3) Impact on the Feedback Loop.: These refinements enhance the quality and precision of the feedback received by the agent. The decision-making component can now more effectively assess which actions lead to meaningful gains, while the learning mechanism receives clearer reinforcement signals. This results in faster convergence to optimal strategies and increased policy stability across episodes. The agent's behavior becomes more adaptive, proactive, and robust in the face of dynamic environmental changes.

B. Stability and Convergence

The effectiveness of the Ms. Pacman agent's learning process depends on its ability to demonstrate both stability and convergence throughout the simulation.

1) Agent Stability: An agent is considered stable if its internal state and control actions remain within reasonable bounds during each episode. Theoretically, this corresponds to bounded trajectories in the state space, ensuring the agent does not enter infinite loops or exhibit chaotic behavior. Practically, stability can be validated by verifying that the agent does not oscillate excessively, maintains control over position and velocity, and does not exhibit runaway behaviors in response to repetitive stimuli.

2) Policy Convergence.: Convergence is defined as the agent's ability to improve and then stabilize its decision policy over time. Theoretically, this implies a diminishing change in policy mappings or Q-value updates across episodes. Practically, convergence can be observed through flattening learning curves, stabilized cumulative rewards, and reduced variability in selected actions over time.

3) Evaluation Methods.: Empirical assessment of stability and convergence is conducted through:

- Monitoring trajectory boundedness via state norm plots.
- Tracking cumulative reward variance across episodes.
- Measuring action selection entropy to detect reduction in exploratory fluctuations.

These criteria ensure that the agent not only learns effectively but does so in a manner that is robust and reliable under the dynamics of the Pacman environment.

III. Iterative Design Outline

A. Roadmap Stages

1) Stage 1: Reinforcement Learning Fundamentals: Focuses on understanding foundational concepts such as agents, environments, states, actions, and rewards. The goal is to grasp the policy evaluation and improvement cycle without relying on external libraries, setting the stage for deeper algorithmic development.

2) Stage 2: Q-Learning with Tabular Methods: Introduces Q-learning with a tabular approach where each state-action pair is explicitly updated. Implementation occurs in OpenAI Gym with simple environments like FrozenLake and Taxi. This phase allows learners to observe agent learning through discrete value updates.

3) Stage 3: Identifying Limitations of Tabular Methods: Analyzes the scalability challenges of tabular methods. The stage transitions the focus toward function approximation and the motivation for using neural networks in high-dimensional spaces, preparing for deep reinforcement learning.

4) Stage 4: Transition to Deep Q-Networks (DQN): The tabular Q-table is replaced by a neural network using PyTorch. Visual-based environments from Retro Gym are introduced to simulate more realistic settings like Ms. Pacman. Emphasis is placed on high-dimensional state processing and network-based policy learning.

5) Stage 5: Stabilizing DQN: Addresses the instability of DQN training. Techniques such as experience replay, target networks, reward clipping, and normalization are implemented. These techniques enhance learning reliability and support more robust convergence.

6) Stage 6: Testing: Focuses on functional correctness and verification. The environment-agent loop is tested through unit-level assertions, logging of key state transitions, and deterministic reproducibility using fixed seeds. This stage ensures the learning process behaves as expected at a technical level.

7) Stage 7: Evaluation: Measures the agent's performance across a variety of conditions. Metrics such as cumulative reward, policy entropy, win/loss ratio, and convergence trends are tracked. Evaluation scenarios include randomized starts, ghost behavior changes, and alternative map topologies.

8) Stage 8: Advanced DQN Variants: Implements and compares state-of-the-art extensions to standard DQN, including Double DQN, Dueling DQN, and Prioritized Experience Replay. The focus shifts toward sample efficiency, architecture modularity, and benchmarking against baseline models.

9) Stage 9: Advanced Dynamic Behavior Integration: Integrates adaptive behaviors into the agent. Enhancements include context-rich state encodings, temporally-aware reward structures, modular policy networks, and threat prediction mechanisms. Evaluation includes stress-testing under scenario variation and real-time adaptation. Monitoring tools such as TensorBoard support visualization of policy evolution and stability.

B. Testing and Evaluation Strategy.

To evaluate the effectiveness and reliability of these dynamic behaviors:

- **Controlled Randomness:** Fixed random seeds will be used to compare agent behavior under deterministic conditions.
- **Scenario Variation:** Pac-Man mazes will be altered via ROM hacks or input remapping to simulate novel environments.
- **Performance Metrics:** Besides cumulative reward, the agent will be evaluated using policy entropy, action variability, episode length, and success rate under shifting ghost behaviors.
- **Stress Testing:** Evaluation scenarios will include ghost clustering, delayed reward availability, and dead-end traps to test reactive and anticipatory capabilities.

This stage aims to demonstrate emergent agent behavior such as strategic evasion, opportunistic ghost pursuit, and adaptive exploration. The emphasis is on behavioral diversity, learning resilience, and real-time adaptability in non-stationary environments.

IV. References

- Lamar University. (n.d.). *Differential Equations – Phase Plane*. Paul's Online Math Notes. https://tutorial.math.lamar.edu/classes/de/phaseplane.aspx :contentReference[oaicite:1]index=1
- Musacchio, F. (2024, March 17). *Using Phase Plane Analysis to Understand Dynamical Systems*. Retrieved from https://www.fabriziomusacchio.com/blog/2024-03-17-phase_plane_analysis/ :contentReference[oaicite:2]index=2
- University of British Columbia. (n.d.). *Phase Portraits of Linear Systems*. Retrieved from https://www.math.ubc.ca/~israel/m215/linphase/linphase.html :contentReference[oaicite:3]index=3
- Liu, L. T. (2020, January 21). *When Bias Begets Bias: A Source of Negative Feedback Loops in AI Systems*. Microsoft Research Blog. https://www.microsoft.com/en-us/research/blog/when-bias-begets-bias-a-source-of-negative-feedback-loops-in-ai-sys :contentReference[oaicite:4]index=4
- Brown, A. (2020). *How to Train Ms-Pacman with Reinforcement Learning*. Medium (Analytics Vidhya). https://medium.com/analytics-vidhya/how-to-train-ms-pacman-with-reinforcement-learning-dea714a2365e :contentReference[oaicite:5]index=5
- R3sist. (2022). *How Do I Create an AI Controller for Pacman?* AI Stack Exchange. https://ai.stackexchange.com/questions/36005/how-do-i-create-an-ai-controller-for-pacman :contentReference[oaicite:6]index=6