Workshop 1
Systems Design

Juan David Córdoba Aguirre - 20211020097

Andrés Felipe Carvajal Forero - 20202020041

Universidad Distrital Francisco José de Caldas

Ingeniería de Sistemas

Fundamentos de Ciencias de Sistemas

Carlos Andrés Sierra Virguez

2025

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.

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.

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.

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.

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.

Stimuli and Rewards

At the core of the system lies a sophisticated 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 positive stimuli are designed to encourage behaviors that lead to more efficient point collection, being a clear example of reinforcement learning algorithms in action. On the other hand, there are negative stimuli, such as being touched by a ghost or prolonged inactivity without consuming a Peller, which penalize the agent and reinforce the need to maintain constant interaction with the environment. This feedback system, when implemented with deep neural networks, allows the agent to dynamically adjust its action policy, holistically integrating the information received and learning to act in highly complex situations.

Use Cases

The use cases described below illustrate in detail and from a systemic perspective the key interactions within the simulation, showing how each system component integrates to generate intelligent and adaptive behavior in the *Ms. Pacman* environment.

Use Case 1: Maze Navigation and Peller Consumption

In this scenario, the main objective is for AI-controlled Pacman to efficiently navigate the maze and consume all Pellers. The agent must analyze the map—composed of interconnected quadrants—identify optimal paths, and manage point collection sequentially. This task resembles an optimization problem within complex systems, where each decision influences the agent's future state. Here, Deep Reinforcement Learning algorithms play a crucial role, as the neural network learns to identify patterns in Peller distribution and adjust its trajectory to maximize accumulated reward. The AI continuously evaluates the environment, integrating sensory

information and adapting its route to avoid high-risk areas, which translates into a process of continuous learning and strategy refinement based on acquired experience.

Use Case 2: Power-Up Activation and Ghost Consumption

This use case focuses on the dynamic transformation of the environment following the consumption of a Power Peller. Upon activation of the power-up, Pacman enters a phase where he can eliminate ghosts, inverting the usual dynamics of the game. This paradigm shift can be likened to how complex systems respond to external disturbances by adjusting internal parameters to adapt to new conditions. The AI must detect the precise moment the Power Peller is activated and immediately reorient its strategy, shifting from a defensive to an offensive posture to exploit the ghosts' temporary vulnerability. The use of Deep Learning techniques facilitates the detection of such changes in the environment, allowing the neural network to instantly adjust its action policy. In this way, the agent maximizes additional rewards by eliminating ghosts, optimizing resource use and dynamically adapting to the new configuration of the environment in an effective and timely manner.

Use Case 3: Ghost Avoidance and Reaction to Negative Stimuli

This use case highlights the system's capacity to minimize risk and respond to negative stimuli. Pacman must navigate the maze with constant attention to the location and behavior of the ghosts, who represent an immediate threat. Here, the agent uses direction-change nodes to adjust its trajectory and evade contact with enemies—a task that requires detailed, real-time analysis of the environment. Integrating Deep Reinforcement Learning allows the system to interpret sensory data and learn to predict ghost movements, establishing evasion strategies that improve with each iteration. Additionally, the agent is penalized for remaining too long without consuming a Peller, adding a sense of urgency to its behavior. This mechanism of negative feedback drives the system to maintain a constant flow of action, avoiding complacency and ensuring rapid response in critical situations. The agent's adaptive capacity, reinforced by Deep Learning models, results in

continuous improvement in decision-making, consolidating a robust and efficient system in the face of environmental complexity.

Component Diagram

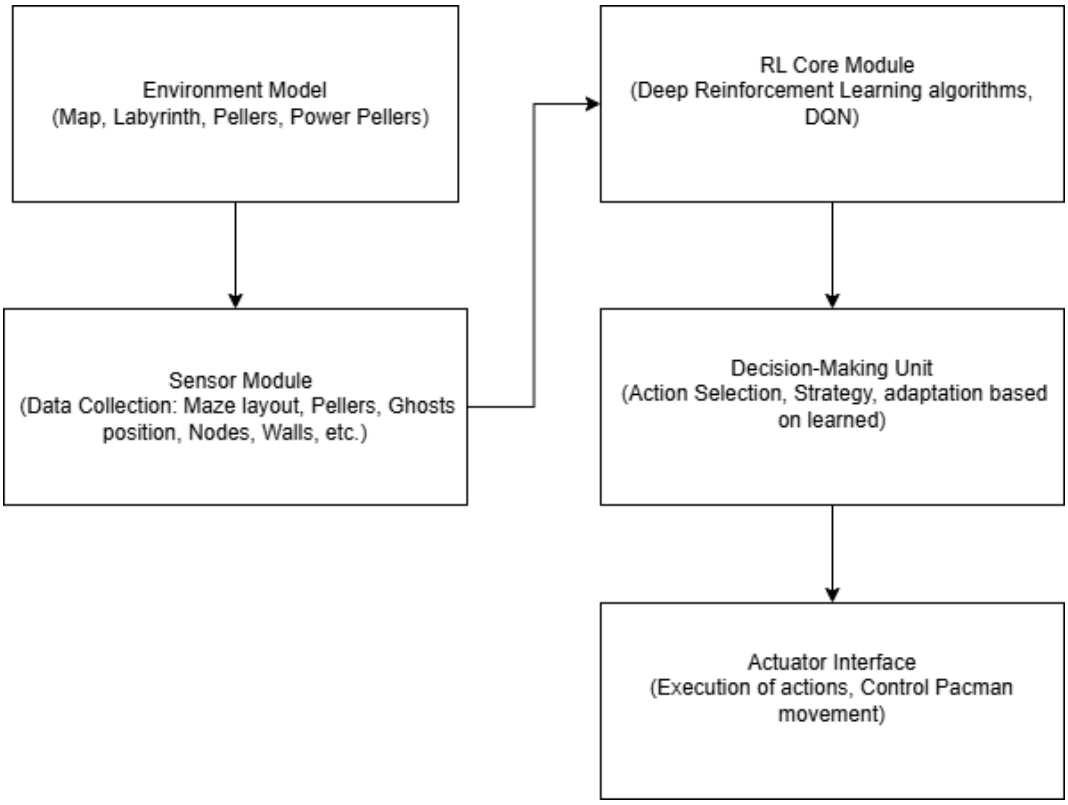

Fig. 1. Component diagram – original illustration by the author.

Cybernetics Feedback Loops

Feedback loops are foundational mechanisms that govern the agent's ability to learn from and adapt to the environment through continuous evaluation of its actions and their consequences. These loops resemble biological systems in which feedback — both positive and negative — helps regulate behavior and ensure survival under dynamic conditions. In the Pacman environment, the feedback loop can be understood as a cyclic process involving sensing, decision-making, acting, and evaluating outcomes, with the goal of optimizing score while minimizing life penalties.

Core Components

Perception (Input Signal Acquisition): Pacman continuously receives information about its surroundings through environmental sensors that capture the state of the maze, the position of Pellers, Power Pellers, ghosts, and other relevant features.

These sensory inputs are transformed into a vectorized representation used by the neural network, serving as the state input in the reinforcement learning pipeline.

Decision-Making (Policy Evaluation): Using its current policy, encoded within a neural network (DQN), the agent evaluates the best action to take in the current state to maximize its expected future reward.

The choice is informed not only by immediate stimuli (e.x., proximity of a ghost or Peller) but also by long-term reward estimations derived from experience replay and Q-value approximation.

Action (Output Signal Deployment): Pacman executes the chosen action (e.x., move up, down, left, right), altering its state within the environment.

This action can have immediate effects, such as consuming a Peller or colliding with a ghost, triggering corresponding environmental changes.

Environment Response (Stimuli and Reward Generation): The environment updates in real time, reflecting the consequences of Pacman's actions.

If a Peller is consumed, a positive reward is issued. If a ghost catches Pacman, a large negative

reward is given. If a Power Peller is consumed, the feedback loop triggers a change in ghost states (to frightened), modifying their behavior and potential reward outcome.

Learning and Adaptation (Policy Update): Based on the reward received and the new state, the agent updates its Q-values or network weights to reinforce successful strategies and penalize poor ones. This adaptation uses the Bellman equation in Q-learning or policy gradients in actor-critic methods, allowing the agent to learn over time which actions yield the highest cumulative rewards.

Types of Feedback Loops Identified in the Pacman Environment

Positive Feedback Loop: Repeated consumption of Pellers reinforces the behavior of navigating toward and prioritizing regions rich in Pellers. Power Peller consumption leads to ghost vulnerability, reinforcing offensive strategies during this temporary phase.

Negative Feedback Loop: Being caught by a ghost discourages risky proximity or ineffective evasion, encouraging avoidance strategies. Prolonged inactivity without consuming Pellers incurs penalties, preventing stagnation and promoting continuous exploration.

Adaptive Feedback Loop: The presence of dynamic ghost behaviors (chase, scatter, frightened) introduces complexity that forces Pacman to adjust strategies in real time. Feedback from these environmental changes modifies the agent's policy to favor flexible, context-aware responses, such as switching from evasion to pursuit upon Power Pellet activation.

Potential Frameworks

In the first stage of the intelligent Pacman agent, in the outline is defined three foundational frameworks that can serve as the core infrastructure for implementing, training, and evaluating the agent: OpenAI Gym, OpenAI Retro Gym, and PyTorch.

OpenAI Gym

OpenAI Gym is one of the most widely used environments for reinforcement learning research. It provides a simple and consistent API for interfacing with a variety of environments, including classic control problems, 2D/3D simulations, and custom grid-based setups like Pacman. For this project, a custom Pacman environment can be created following the Gym environment template, including reset() and step() functions, reward structure, and observation/action spaces. This facilitates compatibility with a wide array of reinforcement learning algorithms and libraries built on Gym standards.

OpenAI Retro Gym

OpenAI Retro Gym extends the capabilities of Gym by enabling reinforcement learning over classic video games using emulated environments. It supports games from consoles such as NES, SNES, Sega Genesis, and more. While there is no official Pacman ROM in the default library, Retro Gym can be configured with supported Pacman-like games (e.g., Ms. Pac-Man for NES) using custom integrations. This allows for a more complex and visually realistic training ground, supporting pixel-based observation and frame skipping, which can be useful for training convolutional neural networks and implementing vision-based agents.

PyTorch

PyTorch is the primary deep learning framework recommended for implementing the neural network components of the agent. Its dynamic computation graph and Pythonic syntax make it especially suitable for modeling feedback loops, step-by-step learning dynamics, and adaptive

policies that change in response to environmental stimuli. PyTorch can be used to implement value-based methods such as Deep Q-Networks (DQN) or policy-based methods like Actor-Critic, which are essential for learning optimal behaviors in the Pacman environment. It also integrates well with Gym through libraries like stable-baselines3 and TorchRL, facilitating rapid experimentation and model iteration.

Roadmap Stages

Stage 1: Reinforcement Learning Fundamentals

In this initial stage, the focus is on understanding the foundational concepts of reinforcement learning (RL), such as agents, environments, rewards, states, and actions. The objective is to grasp the basic loop of policy evaluation and improvement. This stage sets the theoretical groundwork necessary for all subsequent stages.

Stage 2: Q-Learning with Tabular Methods

This stage introduces Q-learning as a model-free, off-policy RL algorithm. The focus lies on tabular implementations where each state-action pair is stored and updated in a Q-table. Agents learn optimal policies through direct interaction with the environment. The OpenAI Gym framework is introduced here to provide a suite of simple RL environments for experimentation.

Stage 3: Identifying Limitations of Tabular Methods

This stage is dedicated to recognizing those limitations and exploring the need for function approximation to generalize across unseen states. It builds the case for using neural networks instead of explicit Q-tables.

Stage 4: Transition to Deep Q-Networks (DQN)

Here, the Q-table is replaced by a neural network that approximates the Q-function. The transition involves integrating PyTorch to build and train deep learning models. At this point, the

Retro Gym framework is introduced, enabling interaction with more complex environments such as Pac-Man, which require visual input and high-dimensional state processing.

Stage 5: Stabilizing DQN

DQN training is sensitive to instability due to correlated data and non-stationary targets. This stage incorporates techniques such as experience replay and target networks to stabilize learning. It also explores reward clipping, frame skipping, and normalization techniques to improve robustness. PyTorch continues to be the framework of choice for implementing these improvements.

Stage 6: Testing and Evaluation

The focus here shifts toward evaluating the performance of the DQN agent. Metrics like cumulative reward, win ratio, and convergence behavior are monitored. Testing is done across various Gym and Retro Gym environments to assess the generalization and reliability of the trained agent.

Stage 7: Advanced DQN Variants

The final stage explores extensions of DQN such as Double DQN, Dueling DQN, and Prioritized Experience Replay. These methods offer improved learning performance and sample efficiency. Implementation continues in PyTorch, and the focus is on modular design and benchmarking against baseline models.

References

- OpenAI. (2023). *Gym Documentation*. https://www.gymlibrary.dev/
- OpenAI. (2023). *Retro Documentation*. https://github.com/openai/retro
- PyTorch Team. (2024). *PyTorch Documentation*. https://pytorch.org/docs/stable/index.html
- Gupta, A. (2020). *Feedback Loops in AI Systems*. Microsoft Research. https://www.microsoft.com/en-us/research/blog/feedback-loops-in-ai-systems/
- Corbett, R. (2021). *Understanding Feedback Loops in Reinforcement Learning*. Towards Data Science. https://towardsdatascience.com/understanding-feedback-loops-in-reinforcement-learning-d1bdbf12ddbd