CS-499 Milestone Three  
Enhancement Two: Algorithms and Data Structures

Shawn Henly

06/01/25

### **Brief Description of the Artifact**

The artifact selected for this milestone is the qtrain() function, originally developed in CS-370: Artificial Intelligence. This function drives the reinforcement learning process for a deep Q-learning agent navigating a maze environment. It executes episodes in which the agent selects actions, receives feedback, stores experiences, and updates its Q-values via a neural network. While the original implementation was functional, it lacked modularity, used hardcoded parameters, and offered little in the way of configurability or traceability—making it difficult to scale, test, or maintain.

### **Justification for Inclusion in the ePortfolio**

I chose this artifact because it demonstrates my ability to transform a complex but rigid machine learning function into a more maintainable and professional piece of software. The original qtrain() function was bloated and procedural, with tightly coupled logic and minimal code reuse. Enhancing this artifact allowed me to show how I can elevate algorithmic implementations by applying core software engineering practices such as modularization, configuration management, and logging.

These improvements not only clarify the underlying algorithmic logic but also make the training process easier to tune, debug, and extend. This artifact, therefore, represents a bridge between foundational machine learning knowledge and real-world engineering discipline—key for any aspiring developer in artificial intelligence or data science.

### **Key Components of the Enhancement**

* **Modular Function Design:** I refactored large, repeated logic blocks into helper functions such as choose\_action() and log\_progress(), improving readability and testability while enforcing separation of concerns.
* **Hyperparameter Structuring:** Instead of using scattered or global variables, I clearly declared all major hyperparameters—such as the number of epochs, memory size, and exploration rate—at the top of the training function. This makes it easy to adjust settings without digging through nested logic.
* **Logging Integration:** I introduced structured logging using Python’s logging module. Logs now capture real-time progress updates, win rates, and training metrics. This replaces ad hoc print statements and adds professional observability.
* **Defensive Programming:** I introduced default values and safety checks for edge cases like missing free cells. This ensures the training loop can handle unusual scenarios more gracefully and improves the program's overall robustness.
* **Cleaner Training Loop:** The refactored main loop removes reliance on global variables and hardcoded constants, simplifying state management and enabling the code to scale to other environments or agents with minimal rework.

### **Course Outcomes Achieved**

This enhancement aligns with **Outcome 3**, which involves designing and evaluating computing solutions using algorithmic principles. I restructured a monolithic function into a maintainable, modular system that still solves the original problem—training a reinforcement learning agent. This required balancing performance with flexibility and maintainability.

It also supports **Outcome 4**, which emphasizes using well-founded and innovative techniques and tools. I applied best practices such as logging, modularization, configuration abstraction, and defensive programming—all of which are essential in industry-standard software. These improvements also promote testing and long-term evolution of the codebase.

Since the final result matches the expectations and intent of my original outcome coverage plan from Module One, no updates to that plan are necessary at this time.

### **Reflection on the Enhancement Process**

Enhancing the qtrain() function challenged me to think beyond functionality and focus on the structure and long-term usability of my code. Breaking down logic that was once lumped into a single training loop gave me a greater appreciation for reusable design and clearer control flow. Working with logging also opened my eyes to how essential traceability is in complex systems, especially those involving machine learning, where silent bugs can persist undetected.

This experience not only strengthened my technical toolkit for building algorithmic solutions, but also reinforced the mindset required to write code that is built to last and ready for real-world use.