Volume 9, Issue 7, July – 2024 International Journal of Innovative Science and Research Technology
ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

A Machine Learning Model for Training Your AI

Akaninyene Udoeyop

Abstract:- Artificial Intelligence is playing an increasing role in solving some of the world's biggest problems. Machine Learning Models, within the context of reinforcement learning, define and structure a problem in a format that can be used to learn about an environment in order to find an optimal solution. This includes the states, actions, rewards, and other elements in a learning environment. This also includes the logic and policies that guide learning agents to an optimal or nearly optimal solution to the problem. This paper outlines a process for developing machine learning models. The process is extensible and can be applied to solve various problems. This includes a process for implementing data models using multi-dimensional arrays for efficient data processing. We include an evaluation of learning policies, assessing their performance relative to manual and automated approaches.

I. INTRODUCTION

Artificial Intelligence (AI) is a field of study that utilizes computer systems and data to solve problems. The term "Artificial Intelligence" is often used interchangeably with the term "Machine Learning". However, it is important to distinguish between the two concepts. Machine learning is a type of AI that involves training computer systems to solve problems through learning and adapting from experience.

There are several approaches to machine learning, including Supervised and Unsupervised Learning. This paper will focus on reinforcement learning. Reinforcement learning is a machine learning technique that involves utilizing a reward-based model to train a learning agent, then enabling the agent to make decisions based on the data accumulated during training.

This paper will outline a process to develop and train data models using reinforcement learning.

A. Reinforcement Learning

Reinforcement learning is a machine learning method that involves rewarding desired behavior and punishing undesired behavior. This involves learning agents interacting with an environment and observing the results or rewards from their actions. The objective in reinforcement learning is to learn the optimal behavior based on experience.

This section outlines reinforcement learning policies and the Q-learning learning algorithm that will be utilized in this paper.

➢ *Policies*

A policy is a function that maps states to actions. It is essentially a strategy that guides the learning agent's actions while it is interacting with the environment. Policies can be either deterministic or stochastic:

- **Deterministic** - Maps states to actions with an expected reward value.
- **Stochastic** - Maps states to a probabilistic distribution over actions.

The difference between a deterministic and stochastic policy is in the way that they choose actions. A deterministic policy will choose the optimal action that maximizes the reward value. This is considered a "greedy" policy. A stochastic policy will choose a random action some percentage of the time, otherwise will follow a greedy policy. This is known as the Epsilon-greedy policy.

➢ *Q-Learning Algorithm*

Q-Learning is a machine learning algorithm that assigns reward values to state-action combinations. The algorithm estimates a function that is closely related to the policy. This function is called the value function. With Q-Learning, the reward values are numerical and are produced by the value function and indicate how "good" or "bad" an outcome of an action is. With Q-Learning, the Q-Value is the metric used to measure an action at a particular state.

$$Q^{new}(S_t, A_t) \leftarrow (1 - \underbrace{\alpha}_{\text{learning rate}}) \cdot \underbrace{Q(S_t, A_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big(\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(S_{t+1}, a)}_{\text{estimate of optimal future value}} \Big)$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

➢ *Machine Learning Models*

A machine learning model is a framework that represents the environmental components and policies used to train learning agents to detect patterns and solve problems. They are trained using techniques such as reinforcement learning. Developing a machine learning model involves defining the states, actions, rewards, and other elements in the environment. Then, selecting learning policies that guide learning agents to an optimal or nearly optimal solution to a problem.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

II. RELATED WORK

Approaches to developing machine learning models are evolving. There are various theories, algorithms, and policies that are available for solving problems using machine learning. Data scientists, for example, can build tens to hundreds of models before arriving at one that meets some acceptance criteria (e.g. AUC cutoff, accuracy threshold)[1]. The process for building a machine learning model is not an exact science. Emily Sullivan, in her research, outlines considerations and rationale behind selecting a learning model and mentions some of these challenges using Neural Networks[2]. MIT researchers Manasi Vartak et al. describe the challenges in building models that adequately correlate to the problem being solved[1].

There are some efforts to organize and standardize machine learning models. In their work, Manasi Vartak et al. are developing a system for the management of machine learning models called ModelDB. While tools like ModelDB can be useful for managing models, this paper focuses on the process of taking the problem, deconstructing the problem into components that are used to build a learning model. Then designing, training, and testing the learning model. James Wexler et al. in their research, developed a tool called What-If to evaluate the performance of several machine learning models[3]. In their study, they show how selecting the appropriate model, algorithm, and criteria can significantly impact the efficiency and the accuracy of results produced by the data model.

The advantages of modeling have been known for many years. In 2013, Christopher M. Bishop touched on some of these in his research on model-based approaches to machine learning[4]. These advantages included the opportunity to create highly tailored models for specific scenarios, as well as rapid prototyping and comparison of a range of alternative models.

III. MACHINE LEARNING MODEL FOR REINFORCEMENT LEARNING

A machine learning model is a framework that defines the states, actions, rewards, logic, data models, and learning policies required to identify patterns or make predictions using machine learning.

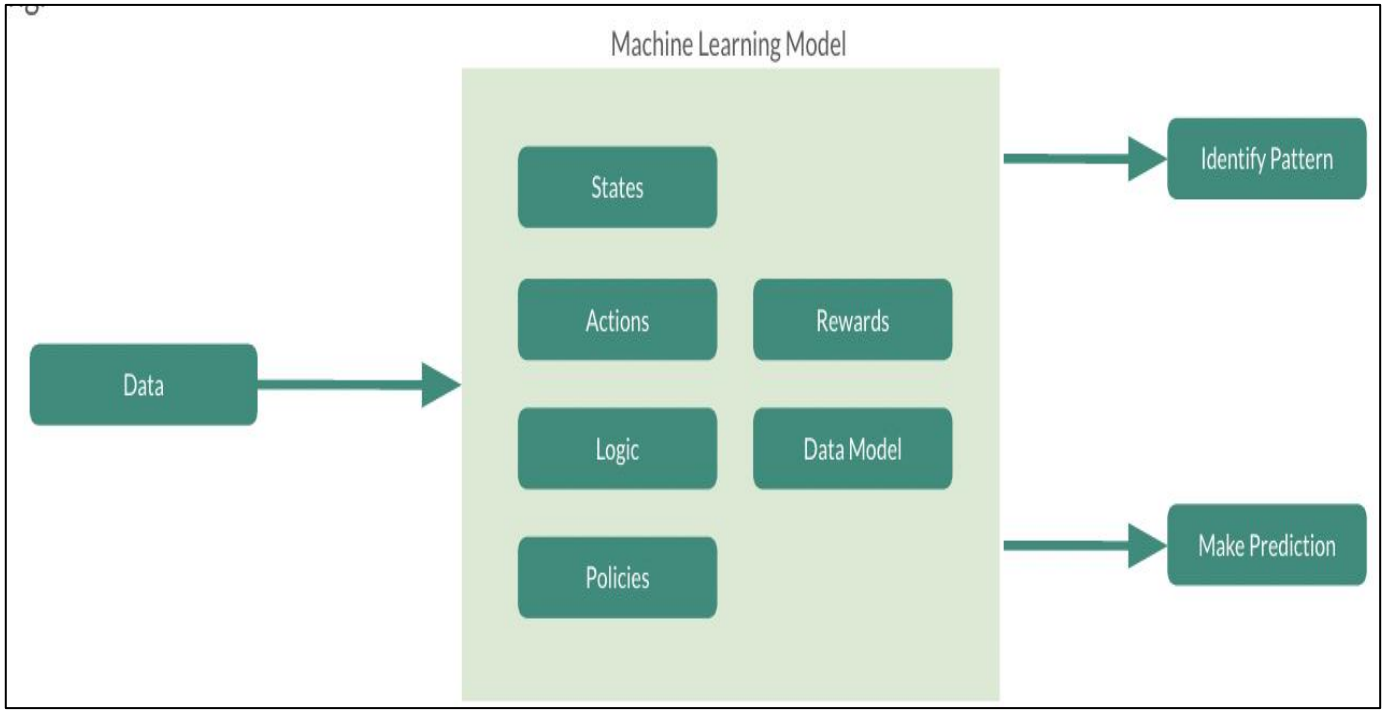


Fig 1: Machine Learning Model

The following outlines a process for deriving machine learning models. The process can be used along with learning policies to solve problems. This includes an approach for implementing the data model used for reinforcement learning.

A. The Learning Egg Problem

To assist in explaining the process for developing a machine learning model, we will use the example of The Learning Egg problem. The problem consists of an egg sitting on a platform that rotates left and right, where the egg can lean to the left and the right to shift the platform in either direction. The objective is for the platform to be balanced and not moving, or as close to being balanced as possible.

B. Define Problems and Goals

The objective in reinforcement learning is to solve a problem through learning from experience. In order to solve the problem, it should be broken down into sub-problems and sub-goals that can be mapped into a machine learning model. Sub-problems and sub-goals should include elements that are quantifiable and limited to factors that affect the overall objective.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

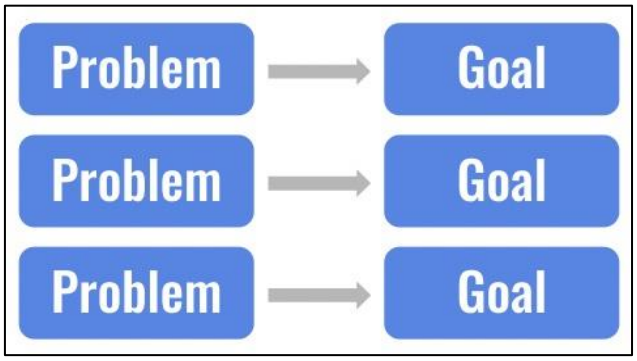


Fig 2: Sub-Problem to Sub-Goal Mapping

For the Learning Egg, the problems and goals are defined in the following sections.

➢ *Problems*

- The platform is unbalanced when it is not positioned at 0°.
- The platform rotates to the left and to the right.

Note that each component of the problem affects the objective of the platform being balanced. The platform's rotation position and the egg's position can be quantified. The objective of the problem is for the platform to also not be moving. This can be quantified by measuring the platform position over time. The egg's movement left and right can also be quantified.

➢ *Goals*

- For the platform to be balanced, or as close to being balanced as possible.
- For the platform to not be moving

Note that the state of the platform being balanced or unbalanced is based on the platform position and speed, which are quantifiable. Being balanced is not based on the position of the egg, although the egg's position influences the position of the platform.

C. Define Environmental Components

After defining the problem and the goal, the environment should be defined. This section outlines environmental components that should be derived, followed by their application in The Learning Egg problem. These elements will provide the foundation for the machine Learning Model.

➢ *States*

A state consists of a combination of quantifiable factors in the environment that affect the problem and the goal. A state factor is defined as a measurable and quantifiable value, also called a **state value**, that reflects an environmental condition at a point in time. Each state value should have an **objective** that is aligned with the overall goal. A distinct state is essentially a snapshot of the set of state values at a point in time.

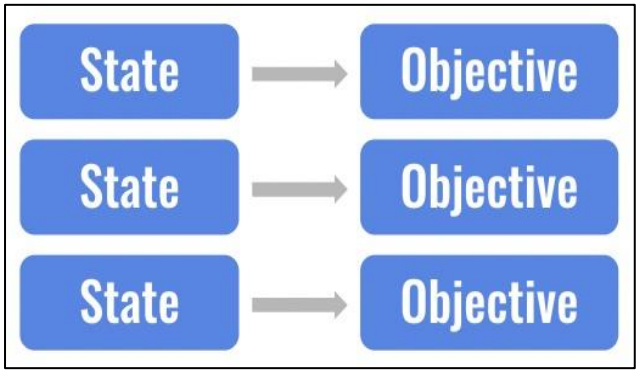


Fig 3: State to Objective Mapping

For The Learning Egg problem, the goal is defined "For the platform to be balanced and not moving, or as close to being balanced as possible". This goal can be quantified by the platform position and speed. The state for this problem can be defined by the following state values:

- **Platform Position** - The number of degrees that the platform has rotated away from a balanced position of 0°. The objective of this state value is for the platform position to be 0°, or as close to 0° as possible.
- **Platform Speed** - The change in the platform position over time. As the platform rotates left and right, it increases and decreases in speed. The objective of this state value is to be at a speed of 0, which indicates that the platform is not moving.

$$platform\ speed = (previous\ platform\ position - current\ platform\ position) / time$$

The objective for this state value is to be zero, which indicates that the platform is not moving.

Note that the factors associated with the egg are not included in the state as the goal only references the platform and not the egg.

➢ *Actions*

An action is an operation that the learning agent, which is the egg in this example, can take to change the state of the environment. Actions are directly enacted by the learning agent and result in a change in one or more state values. Actions should be defined along with a reward metric to gauge how "good" or "bad" the result of an action is. A **reward metric** is a numerical value that represents how positive or negative the result of an action is. An optimal action is the action that will result in the highest reward or the "best" outcome from the action.

For The Learning Egg problem, the only action that can be taken is for the egg to lean to the left or right, or to stand up-right. This can be quantified by the egg's position.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

- **Egg Position** - A number representing the egg's position.

The resulting platform position and speed after the egg moves can be measured as the reward for the action. Thus, the egg position represents the action and the combination of the platform position and speed represents the reward metric. The closer the platform is to being balanced without moving, the better the reward.

➢ *Rewards*

Rewards are quantifiable observations of the environment that indicate how positive or negative that the result of an action is. They need to be quantifiable because they need to indicate a state of positivity or negativity to integrate into a learning model. Rewards are observed after an action is taken and gives the machine learning model feedback that is used to decide on future actions. Each reward should map to a state objective and should be able to measure the result of an action within the context of the objective.

For The Learning Egg problem, the sum of the platform position and the platform speed were used as the reward metric. The objective of the platform being balanced can be measured by the platform angle position. The objective that the platform not be moving while balanced can be measured by the platform speed. The goal for the reward is for both the platform position and speed to be zero, which indicates that the platform is balanced and not moving.

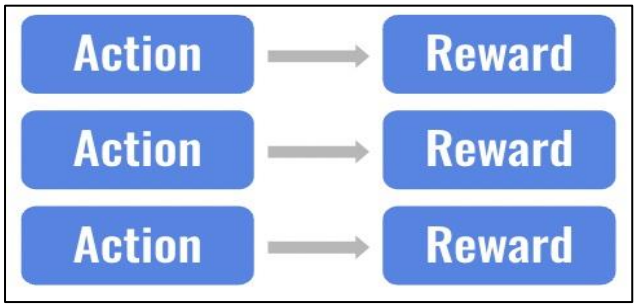


Fig 4: State to Objective Mapping

D. Define Environment Rules and Constraints

Once the environment's components have been defined, rules and constraints should be defined for them. Rules should describe how the environment behaves as it relates to state values and actions. Constraints define what the environment's components can and cannot do. Constraints should be defined as limitations for the state values and actions. This can be implemented by defining a numerical range, scale, and limits for the environment's components. Environment rules and constraints for The Learning Egg problem are defined in the following sections.

➢ *Platform Position*

The platform can rotate from -50° to 50°. Thus, the **platform position** value will range from -50 to 50, with the value 0 meaning that the platform is balanced. Negative values indicate that the platform has rotated left. Positive values mean that the platform has rotated right.

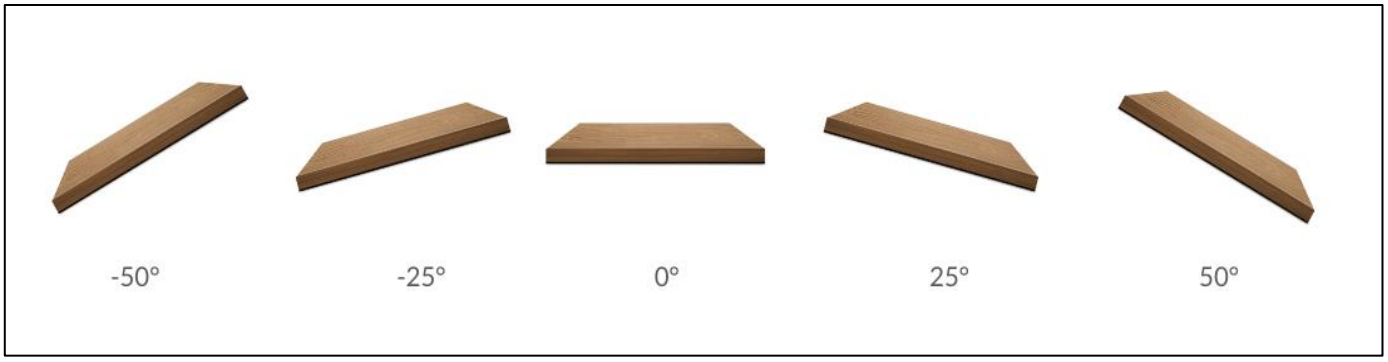


Fig 5: Platform Position

➢ *Egg Position*

The egg can move into the following 5 positions:

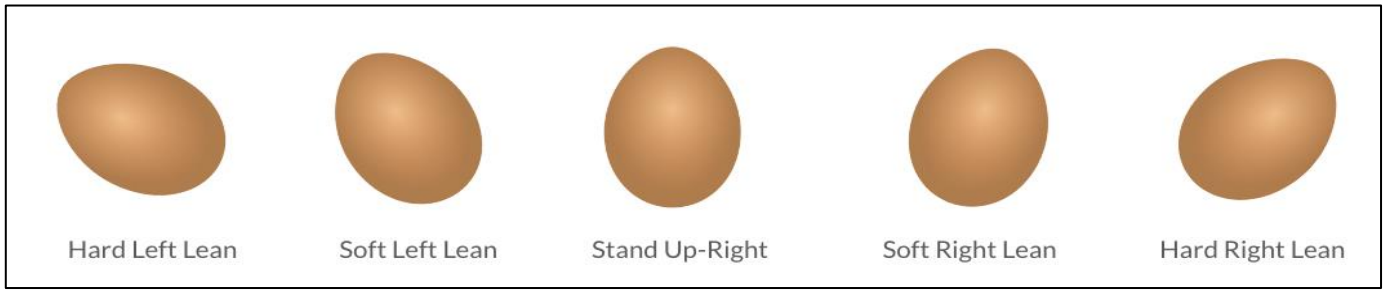


Fig 6: Egg Position

The **egg position** will be represented by the numbers -2,-1,0,1, and 2 with each number denoting the respective position above.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

➢ *Platform Speed*

The platform speed is defined as the change in the platform position over time:

$$platform\ speed\ =\ (previous\ platform\ position\ -\ current\ platform\ position)\ /\ time$$

Since the platform position value is a representation of degrees, the platform speed will be denoted in degrees/second. The **platform speed** can range from -2 to 2, with negative values indicating the speed going in the left direction, and positive values indicating the speed going the right direction. A platform speed of 0 means that the platform is not moving.

The following table outlines how the egg position affects the platform speed:

Table 1: Egg Position Effect on Platform Speed

Egg Position	Platform Speed Change
Hard Left Lean	Shifts platform speed left at -2 degrees/second
Soft left lean	Shifts platform speed left at -1 degree/second
Stand Up-Right	Platform speed does not change
Soft Right Lean	Shifts platform speed right at 1 degree/second
Hard Right Lean	Shifts platform speed right at 2 degrees/second

This table essentially shows that, if the egg leans left, the platform speed will shift to the left. Also, if the egg leans right, then the platform speed will shift to the right. The harder the lean of the egg, the greater the change in speed for the platform. Note that the platform speed will not change when the egg is standing up-right. This simply means that if the platform is in motion, it will remain at the same speed. The egg standing up-right does not mean that the platform is not moving.

E. Define Data Model

A Data Model is a data structure and format for representing data. In machine learning, data models are used to store the information learned from interacting with the environment. There are several types of data models. In Q-Learning, A Q-table is the data model used to store the reward values, or Q-values, for actions taken at a specific state. Q-tables format the states as the rows in the table and the actions as the columns in the table.

Table 2: Q-table Example

	Action 1	Action 2
State 1	QScore(State1, Action1)	QScore(State1, Action2)
State 2	QScore(State2, Action1)	QScore(State2, Action2)
State 3	QScore(State3, Action1)	QScore(State3, Action2)

The Q-table serves as an effective lookup table for scenarios where a single action is taken in a single state. Many problems however, have multiple state factors and multiple actions that affect the environment at a single point in time. The Q-table operates in a 2-dimensional space, where the state is the y-axis, and the action is the x-axis.

Because of this limitation, we derived a data model design to support multiple state factors and multiple actions. Instead of the 2-dimensional space that the Q-table operates in, this data model design takes a multi-dimensional approach. Like the Q-table, this data model design, as a representation of the environment, should include the states, actions, and rewards defined for the environment. These are mapped into the columns of the table. The table should be structured with the first columns representing state values, followed by columns that represent the actions, then followed by a reward column. This approach is extensible and can be applied to a diverse set of problems. This approach is flexible in that the number of states and actions can vary.

Fig 7: State Action Reward Table Column Structure

➢ *Based on this Format, the Table for The Learning Egg Example will be Structured as Follows:*

- **Platform Position** - Number of degrees that the platform has leaned away from a balanced position or 0° before egg movement
- **Platform Speed** - Degrees per second that the platform is moving before egg movement
- **Action** - The position that the egg is moving to
- **Reward** - Sum of the platform position and speed.

Fig 8: Platform/Egg State Action Reward Table Column Structure

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

As the data model is populated with data, the learning agent will gain an understanding of potential outcomes from its actions. The following is a sample of what the data model can look like as it is being populated:

Table 3: Data Model Trained for Platform Position and Speed.

Platform Position	Platform Speed	Egg Position	Reward	Resulting Platform Position	Resulting Platform Speed
25°	1	Hard Left Lean	23	24°	-1
25°	1	Soft Left Lean	25	25°	0
25°	1	Stand Up-Right	27	26°	1
25°	1	Soft Right Lean	28	27°	2
25°	1	Hard Right Lean	28	27°	2

The data model would be populated when the platform has experienced the position of 25° while moving at a speed of 1 degree/second multiple times. When the egg makes a movement from that platform position and speed, it observes the resulting platform and speed, then calculates their sum as the reward. Based on the information in the data table above, we can infer that at a platform position of 25° ,and moving at a speed of 1 degree/second, the optimal action is a Hard Left Lean because that results in the reward of 23. Since the goal of The Learning Egg problem is to get to a platform position and speed of 0, the reward value closest to 0 is considered optimal. Thus, the reward value of 23 being the closest value to 0 for this platform position and speed would be considered optimal. This would lead the learning agent to take a Hard Left Lean if it were pursuing the optimal action.

Note that the data table represents a small subset of a larger dataset. The data model can store data for every combination of platform position, platform speed, egg action, and reward.

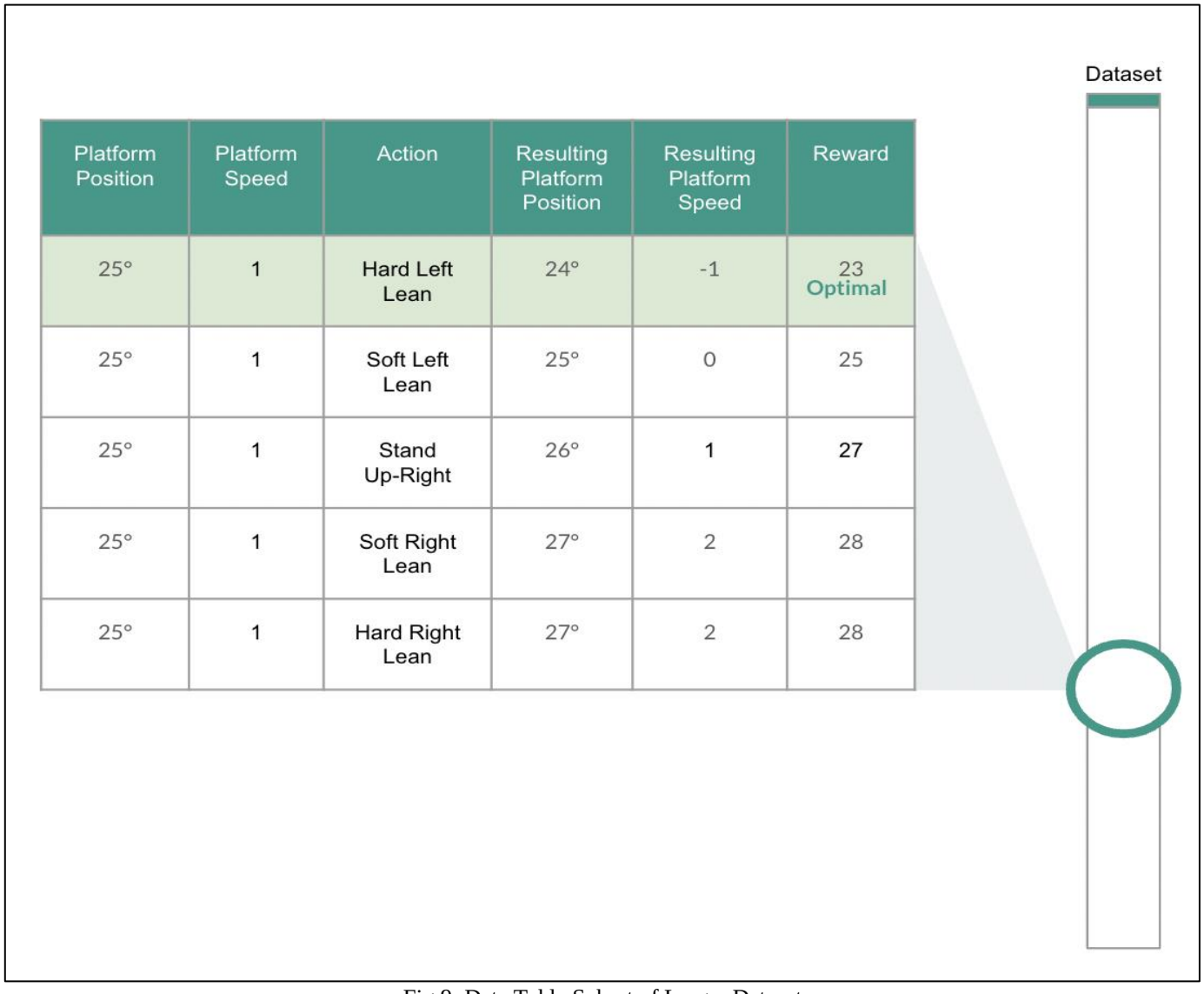


Fig 9: Data Table Subset of Larger Dataset

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

F. Select Data Structure for Data Model

In Computer Science, data structures are software components that are formatted for organizing, processing, retrieving and storing data. These can be used for a variety of purposes, including representing a data model in reinforcement learning. The data structure utilized for this approach is a multi-dimensional array. In Computer Science, an array is essentially a list of values with indexes that point to specific locations on the list. A multi-dimensional array is essentially a list of lists. The choice to use nested arrays was made due to arrays' fast lookup complexity of O(1). This would allow the learning agent to efficiently lookup and update rewards as it learns. Multi-dimensional arrays also support the Q-table inspired data model described in Section 2.5.

➢ *Multi-Dimensional Array*

The multi-dimensional array represents the data from the data model. Thus, each column of the data table, should be mapped to indexes of the multi-dimensional array. The rewards should be stored in the array with the indexes used to lookup and update their respective rewards.

From the data model format described in Section 2.5, we can derive the multi-dimensional array below by mapping each of the columns of the table to an index of the array. This approach allows the lookup and updating of reward values to occur at an efficiency of O(1).

State val 1	State val 2	State val 3	...	Action 1	Action 2	...	Reward

array[state_val_1][state_val_2][state_val_3][action_2][action_2] = reward

Fig 10: Multi-Dimensional Array State Action Reward Data Model

This approach used for The Learning Egg problem produces the following multi-dimensional array:

Platform Position	Platform Speed	Egg Position	Reward

array[platform_position][platform_speed][egg_position] = reward

Fig 11: Multi-Dimensional Array Platform/Egg State Action Reward Data Model

With this array, reward values can be looked up and updated by using the platform position, platform speed, and egg position to index. This allows managing the data via indexing, not searching.

G. Environmental Logic and Policy

A machine learning model involves deriving policies that can be used by a learning agent to guide its future actions. In order to implement a policy, each rule and constraint should be implemented within the logic of the environment in software. For The Learning Egg problem, the following rules and constraints were integrated into the environmental logic of the program:

- The platform can rotate from -50° to 50°. Thus, the **platform position** can range from -50 to 50.
- The following are the possibilities for **egg position**:
  - ✓ Hard Left Lean
  - ✓ Soft Left Lean
  - ✓ Stand Up-Right
  - ✓ Soft Right Lean
  - ✓ Hard Right Lean
- The **egg position** will be represented by the numbers -2,-1,0,1, and 2 with each number denoting the respective position above.
- The **platform speed** can range from -2 to 2, with negative values indicating the speed going in the left direction, and positive values indicating the speed going the right direction.
- The following table outlines how the egg position affects the platform speed:

Table 4: Egg Position Effect on Platform Speed

Egg Position	Platform Speed Change
Hard Left Lean	Decreases platform speed at -2 degrees/second
Soft left lean	Decreases platform speed at -1 degree/second
Stand Up-Right	Platform speed does not change
Soft Right Lean	Increases platform speed at 1 degree/second
Hard Right Lean	Increases platform speed at 2 degrees/second

IV. TRAINING PHASE

Reinforcement learning involves an iterative learning process that involves training the learning agent by running simulations of the problem. During these simulations, the learning agent will interact with the environment and populate the data model, thus learning about the environment. There are various approaches and factors that go into training a learning agent. We will evaluate deterministic, stochastic, manual, automated, and other approaches. For the approaches that involve reinforcement learning, the data model is populated with reward data.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

- *Learning Policy and Approach*

Now that we have developed a data model, we are faced with the challenge of "how" to train our data model. Are some approaches more efficient than others? Do some approaches adapt better to changing environments? Why choose one approach over another? How do these approaches compare to manual or automated solutions? In this section, we will cover the rationale behind why policies and approaches to training were chosen.

➢ *Manual Approach*

A manual approach involves a human being controlling the actions of the agent in the environment. This approach does not use automation or machine learning. For The Learning egg problem, a person would control the egg movement with a computer keyboard. The person would move the egg left and right with the goal of balancing the platform.

➢ *Automated Approach*

An automated approach involves the agent following a pre-programed policy where actions and conditions are hard-coded to achieve a goal. No machine learning is used for this approach as the behavior is predetermined. For The Learning Egg problem, the following outlines the behavior of the egg for an automated approach used for this research:

- If the platform angle is greater than 0°, move the egg one position to the left until it is in a Hard Lean Left position.
- If the platform angle is less than 0°, move the egg one position to the right until it is in a Hard Lean Right position.
- Otherwise, move the egg to a Stand Up-Right position.

➢ *Deterministic Policy*

A deterministic policy selects actions that are projected to yield the highest reward. This type of policy is considered "greedy". For The Learning Egg problem, the following outlines the behavior for the deterministic approach used in this research:

- When the platform is at an angle that has not been experienced yet, the egg should make a random move from the set of all possible moves.
- When the platform is at a previously experienced angle, and has made moves from that angle, and learned from the rewards, make the move that has yielded the highest reward based on experience.

➢ *Stochastic Policy*

In reinforcement learning, stochastic policies will choose a random action from a state some percentage of the time, otherwise will follow a greedy policy. The notion of choosing a random action is referred to as "exploring". The **exploration rate** is the percentage of actions that should be taken by the learning agent in an exploring fashion vs a greedy fashion.

- *For The Learning Egg Problem, the Following Outlines the Behavior for the Stochastic Approach used in this Research:*

- For a percentage of the time that matches the exploration rate, the egg should make a random move from the set of all possible moves.
- The rest of the time, the egg should make the move that has yielded the highest reward based on experience.

➢ *Exploratory Policy*

The learning approaches referenced earlier have their pros and cons, which we will discuss later in the document. For this research, we are implementing and testing an approach called an Exploratory Policy. The policy prioritizes exploring new states and actions when little is known about the environment. Once enough information has been learned about the environment, the policy then guides the learning agent using a deterministic policy.

- *For The Learning Egg Problem, the Following Outlines Behavior Using an Exploratory Policy:*

✓ When the platform is at an angle that has not been experienced yet, the egg should make a random move from the set of all possible moves.
✓ When the platform is at a previously experienced angle, and the egg has made a moves and learned from actions from that angle:

- If the egg has learned from one or some actions, but has not learned from all of the actions to take from that platform angle, the egg makes a random move from the set of actions that have not been taken from that state.
- If the egg has learned from every move from that angle, make the move that has yielded the highest reward based on experience.

This approach prioritizes exploring early in the learning process, then prioritizes maximizing rewards once information has been learned.

➢ *Enumeration Policy*

There are other approaches to training in reinforcement learning. One of these approaches will be referred to as an enumeration policy. This type of policy explicitly iterates through every combination of the states and actions in the environment until the data model is filled with information.

- *For The Learning Egg Problem, the Following Outlines Behavior Using an Enumeration Policy:*

✓ The platform is forced into every position iteratively one-by-one.
✓ At each position, the egg makes every possible move and learns from the rewards from its actions.
✓ This is continued until every action has been taken from every state.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

Note that there are both software and physical limitations for this type of approach. In a physical setting, physics, cost, and safety can render this policy not feasible. This policy is also inefficient from a software complexity standpoint, and does not scale well, which limits its use for software based machine learning.

➢ *Sampling*

Sampling is when a learning agent takes an action from a state and observes the result. In some learning environments, sampling produces relatively consistent results or rewards. This is common in many software-based simulations. However, some learning environments have factors that can lead to varied results when the same action is taken from the same state. This can occur in many physical environments where factors such as weather, turbulence, and air resistance can affect the consistency of the rewards from sampling. When this is the case, multiple samples may need to be taken in order to derive an average or a distribution of rewards. The Exploratory Policy allows for a number of samples in order for a state-action combination to be considered "learned". This will allow the learning agent to explore unknown states until all of the actions from a state have been sampled an adequate number of times. After all of the actions from a state are learned, a learning agent following the Exploratory Policy will then take the greedy action.

- *Learning Egg Simulation*

The Learning Egg is simulated via a web application that utilizes JavaScript to implement the logic, physics, learning model, and policies for solving the problem. The egg can be moved manually, via automation, and by several reinforcement learning techniques. Moving the egg will affect the position and speed of the platform based on the rules and constraints defined in Section 3.4. The egg can be manually moved left and right by a user pressing left and right keys from the computer keyboard.

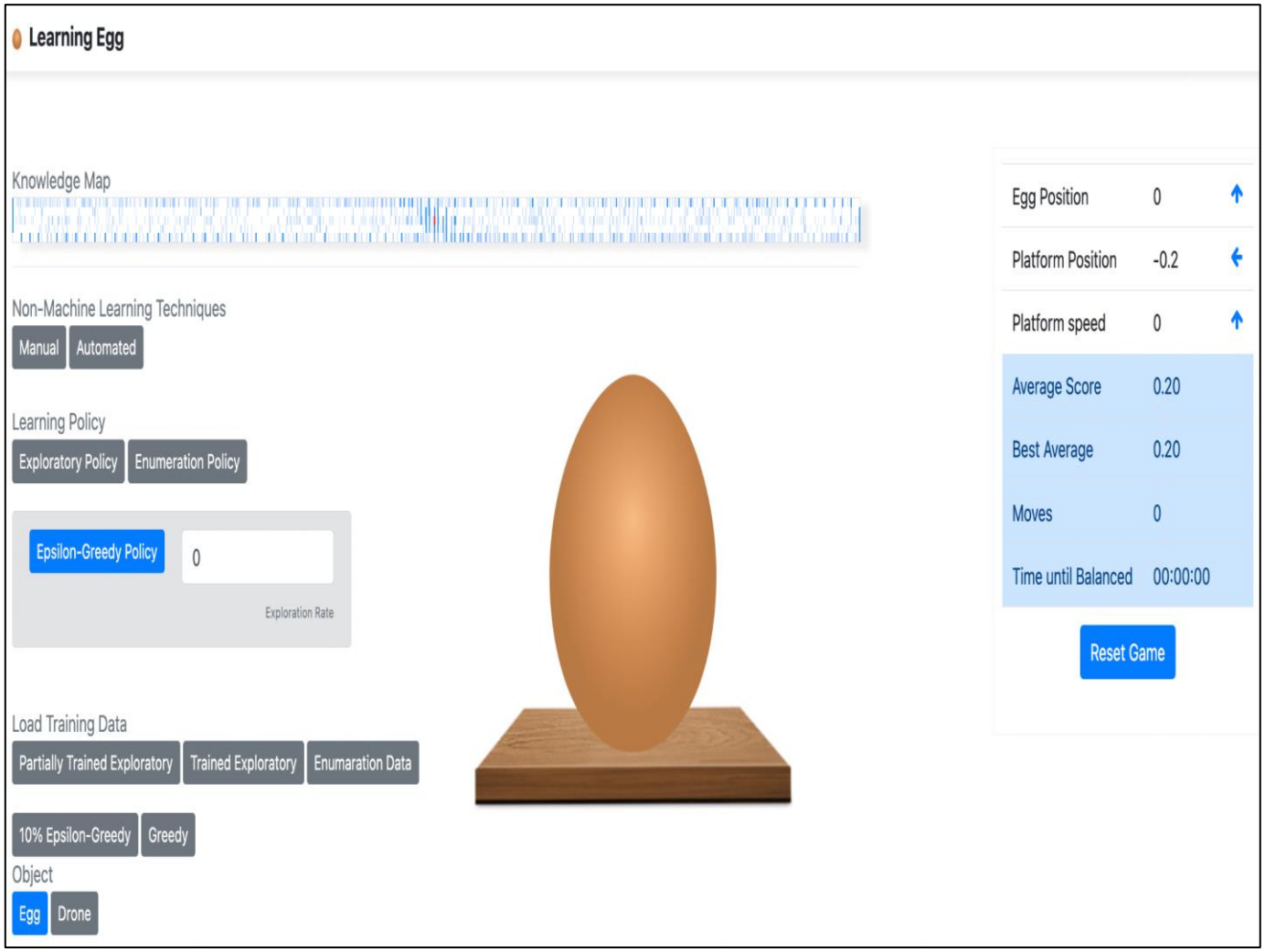


Fig 12: The Learning Egg Web Application

➢ *The Balancing Game*

Performance for each learning approach is measured by simulating The Balancing Game. The goal of the game is to balance the platform for 5 seconds. The platform is considered balanced when its position is between -1° and 1° for 5 seconds. Some approaches did not result in a balanced egg, however a rolling 5 second average of the platform's angle is tracked. The best rolling 5 second average would measure how close the platform came to being balanced during the simulation.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

➢ *Training Machine Learning Model*

Approaches that utilize reinforcement learning will require training. This occurs by the egg being guided by a learning policy to make movements while populating the data model with learned data. This can result in the egg balancing dring training, however some approaches did not result in the egg balancing.

After training the data model, we tested the quality of the learned data by introducing "turbulence", or instability, into the environment and testing how quickly the egg would rebalance the platform by following a greedy policy. This was executed by shifting the egg and the platform all the way to the right, then continuously pressing the right arrow key on the keyboard to constantly manually move the egg to the right. This is all done while the egg tries to rebalance by following a greedy policy. This introduction of instability into the environment will test how the learning policies react to unforeseen changes to the environment.

➢ *Performance and Metrics*

The following metrics will be used to measure the performance of the manual, automated, and machine learning approaches:

- **Best Angle From Balanced** - The best rolling 5 second average of the platform's angle. This would measure how close the platform came to being balanced during the simulation.
- **Number of Moves Until Balanced** - The number of movements that the egg has to make in order to balance. This would measure the efficiency of the egg movements, where balancing using less moves would be considered more efficient.
- **Time Until Balanced** - The amount of time that the egg took to balance. This would measure efficiency in terms of time, with a faster time considered more efficient.
- **Time Until Rebalanced** - The amount of time that the egg took to balance after instability is introduced into the environment and a greedy policy is followed. This would measure how quickly the learning policies react to unforeseen changes to the environment, with a faster time considered more efficient.

V. RESULTS

The Learning Egg problem was simulated using the manual, automated, and machine learning techniques discussed in this document. The table below shows the average result of running 10 simulations for each approach:

Table 5: Results from Simulations

Learning Policy	Best Angle From Balanced	Number of Moves Until Balanced	Time Until Balanced	Time to Rebalance
Exploratory	0°	11,992.3	00:18:56.8	00:00:55.3
Greedy	0°	6268.7	00:15:22.5	00:01:31.9
Epsilon-Greedy (10%)	0°	13,802.1	00:18:46.9	00:01:20.7
Enumeration	0°	25,025.9	01:25:46.4	00:00:07.1
Automated	3.08°	Does Not Balance	Does Not Balance	N/A
Manual	5.77°	Does Not Balance	Does Not Balance	N/A
Epsilon-Greedy (50%)	33.10°	Does Not Balance	Does Not Balance	Does Not Rebalance
Epsilon-Greedy (90%)	37.64°	Does Not Balance	Does Not Balance	Does Not Rebalance

The Epsilon-Greedy policies are denoted with the exploration rate percentage. Note that non-machine-learning approaches did not utilize a machine learning data model, thus could not be used to rebalance the platform. The "Time to Rebalance " column for manual and automated approaches use "N/A" to indicate this.

A. Balancing Performance

From these results, we see that only the Exploratory, Greedy, Epsilon-Greedy (10%), and Enumeration policies resulted in the platform being balanced. The Automated approach outperformed the Manual approach with best 5 second rolling averages of 3.08° and 5.77° respectively. Epsilon-Greedy policies with moderate to high exploration rates performed the worst. Policies with exploration rates of 50% and 90% yielded best 5 second rolling averages of 33.10° and 37.64° respectively.

For the machine learning approaches that resulted in a balanced platform during training, the Greedy policy achieved this in the least number of moves from the egg on average, which was 581.7. The Exploratory and Epsilon-Greedy (10%) policies required an average of 11,992.3 and 13,802.1 moves respectively, with the former slightly outperforming the latter. The Enumeration policy performed the worst in that it required an average of 25,025.9 moves. This is due to the policy iterating through every combination of state and action.

- Note: The following times are formatted as follows: HH:MM:SS
- HH - 2 digits representing the number of hours
- MM - 2 digits representing the number of minutes
- SS - 2 digits representing the number of seconds

The Greedy policy also held the best average performance of 00:03:12.5, with respect to the time that it took to balance the platform during training. The Exploratory and Epsilon-Greedy (10%) policies required an average time of 00:16:46.8 and 00:18:46.9 respectively for the platform to balance, with the former slightly outperforming the latter. The Enumeration policy performed the worst with an average time of 01:25:46.4.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

B. Rebalancing Test

After training, we tested the quality of the learned data by introducing turbulence and observing how quickly a greedy policy can use the learned data to rebalance. This tested how well the learning policies react to unforeseen changes to the environment. Interestingly, The Greedy policy performed poorly with an average rebalancing time of 59.9 seconds. This was due to the learning agent not exploring and learning enough of the environment to know what to do in the states introduced by instability. The Epsilon-Greedy (10%) performed even worse at an average time of 1 minute and 0.7 seconds. Although the learning agent is able to balance during training, the learning agent would require significantly more training to reduce the rebalancing time. The trade-off there would be that more training would increase the amount of time required for adequate training. The Exploratory and Enumeration policies performed the best averaging 18.3 and 7.1 seconds respectively. The exploratory policy prioritizes exploring states and actions that have not been visited before. This adds some range to the set of states that get explored, while minimizing re-visiting states. The Exploratory policy's trait of switching to a greedy policy after enough information is known about a state, allowed the learning agent to move closer to the goal faster than Epsilon-Greedy policies. The Enumeration policy yielded a comprehensive dataset of every state and action. This produced a data model that was robust enough to handle turbulence well.

C. Knowledge Map Visualization

The Learning Egg web application includes a knowledge map, which is a 2-dimensional grid that shows a visualization of the coverage in knowledge. The map shows which platform position/speed states that the egg has been trained for. The x-axis represents the platform position, and the y-axis represents the platform speed.

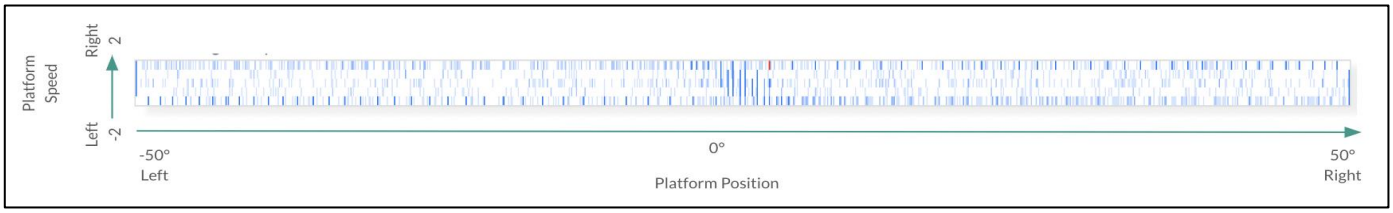


Fig 13: Knowledge Map Visualization

Each grid cell represents the number of actions, at a specific platform position/speed state, that the learning agent has been trained for.

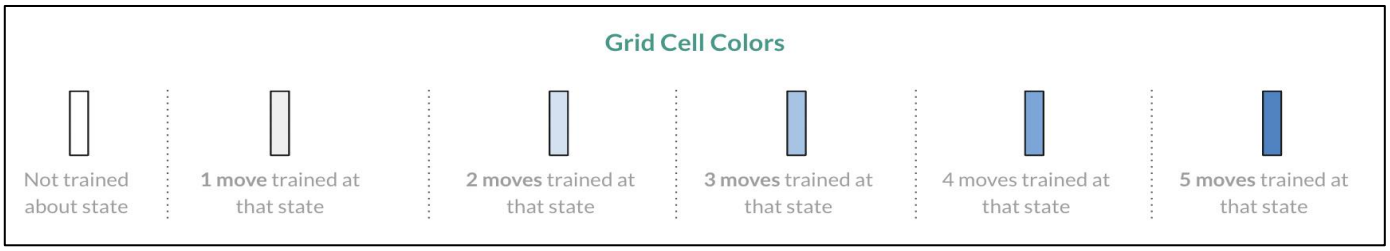
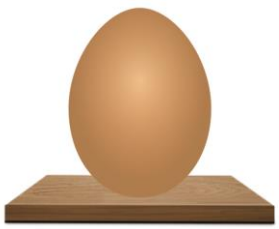


Fig 14: Knowledge Map Grid Cell Color Mappings

D. Extensibility

The process for developing machine learning models that is outlined in this document was designed to be extensible. This means that it can be applied to other problems. The same principles used for The Learning Egg problem can be used to define the environment and learning policies for other problems. One example of how this approach can be applied would be a drone.

The Learning Egg problem used the following data model to represent its environment.



Platform Position	Platform Speed	Egg Position	Reward

Fig 15: Learning Egg Platform/Egg State Action Reward Data Model

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL769

Instead of balancing a platform, the goal for a drone would be to balance the drone based on position and speed. Instead of an egg moving, the drone's propellers are moving. The drone could use a similar data model to The Learning Egg.

Drone Position	Drone Speed	Propeller Position	Reward

Fig 16: Drone/Propeller State Action Reward Data Model

➢ *The following Outlines a Process for Developing a Machine Learning Model to Solve Problems Using Reinforcement Learning:*

- Define the problem and the goal
- Define environment components (states, actions, rewards)
- Define environment rules and constraints
- Define a data model
- Develop a data structure for the data model
- Implement the environmental logic and policies in software
- Train the learning agent to learn an optimal solution

VI. CONCLUSION

Machine learning is a useful tool for solving problems that manual or purely automated solutions cannot. In order to solve these problems, they will need to be mapped to a machine learning model. We have outlined an approach to building machine learning models using reinforcement learning that is extensible. Through simulations, we have observed the performance of various learning policies and approaches. We observed that the Exploratory policy performed well across the board. Due to its directive to explore the unknown, and maximize the reward once enough information is known, the policy allowed for adequate exploration, minimizing revisiting states. The policy also took a greedy approach once enough information was known about a state, which moved the egg and platform closer to the goal of being balanced, faster. It is important to select a data structure that optimizes performance and facilitates learning. This is why multi-dimensional arrays proved useful due to their fast lookup complexity of O(1). The Learning Egg problem provides an example of how using a learning policy that explores states and actions that have not been explored before, then switching to a deterministic policy can be effective. This paper gives insight into how to build a machine learning model, not only for The Learning Egg problem, but to solve a variety of problems.

REFERENCES

[1]. Manasi Vartak , Harihar Subramanyam , Wei-En Lee , Srinidhi Viswanathan , Saadiyah Husnoo , Samuel Madden , Matei Zaharia, 2016. ModelDB: A System for Machine Learning Model Management.

[2]. Emily Sullivan, 2022. Understanding from Machine Learning Models: The British Journal for the Philosophy of Science, Volume 73, Number 1.

[3]. James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viegas, and Jimbo Wilson, 2020. The What-If Tool: Interactive Probing of Machine Learning Models: IEEE Transactions On Visualization And Computer Graphics, Vol. 26, No. 1.

[4]. Christopher M. Bishop, 2013, Model-Based Machine Learning: Phil Trans R, Soc A 371: 20120222