School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

NeuroNudge: An unsupervised learning algorithm for young learners with ASD

by

Adi Kishore

COMP4121 Major Project

Submitted: November 2023

Supervisor: A/Prof. Aleksandar Ignjatovic

Student ID: z5254569

Contents

Chapter 1

Introduction

1.1 The Problem Space

Autism Spectrum Disorder (also known as ASD) is a condition which has impacted the lives of many. Among children, up to 1 in 36 of them are found to have ASD in the USA [1]. A wide range of impacts have been found from this disorder on the lives of such children, such as a difficulty in socialising or communication, or the difficulty to grasp knowledge and basic concepts - relative to the pace of an average child who does not have this disorder.

The key difference for learners who have ASD is that typically, they require reinforcement of concepts multiple times more than a regular learner. Furthermore, each learner is placed at a different position of this spectrum, meaning the impact on their learning is unique and depends on the unique traits that they have as a result of ASD. These traits may include attention span, communication level, communicability and others.

NeuroNudge is a potential unsupervised algorithm that can automate the experience of learning for individuals with ASD. NeuroNudge aims to work on one key goal: recommend lesson content to learners based on their individual learning progress.

This report intends to introduces NeuroNudge as a promising unsupervised learning algorithm for learners who have ASD.

1.1.1 Unsupervised Learning and ASD

Among the many responsibilities that a parent or caretaker has for a child/individual with ASD, here are a few that are often outlined by most experts for children in particular (paraphrased and summarised from American source, Hope for Healing [2]:

- **Creating a Sensory-Sensitive Environment:** this involves the need to minimise sensory overload by creating a predictable and calming space, which caters to the child's sensory processing needs.
- **Facilitating Therapeutic Support:** this involves consulting health professionals to access beneficial therapies such as speech or education at large.
- **Advocacy in Education and Social Domains:** this requires that the child receives necessary support and accommodations by advocating for their needs in schools and social environments.
- **Fostering Social Skills:** this requires aiding the child in learning social interactions to enhance their communication skills.
- **Behavioral Management:** this includes the development and implementation of strategies often with healthcare providers to address and manage challenging behaviors.
- **Addressing Comorbid Conditions:** this involves working with medical professionals to identify and treat any concurrent medical or mental health conditions in order to improve the child's overall well-being.

Hence, caretakers have a wide expanse of responsibilities that they must handle simultaneously, which can cause a lot of stress. While many of these areas are not things which can be automated due to the need for personal touch and attention to the child or individual, there could be potential in automating the educational aspect.

Instead of relying on a clinical therapist and accruing further costs in this respect, a

hybrid approach may be viable where both the costs of a clinical therapist and caretaker may drop, as a reinforcement algorithm can be used to recommend lesson plans and routines that the child should follow based on their unique traits and progress. In particular, the hybrid act comes into play with experts such as clinical therapists tweaking and adjusting the algorithm to better fit a child. Ultimately, this is certainly an opportunity for making the lives of caretakers easier, and such is the goal of NeuroNudge.

The following sections will now explain the aim of NeuroNudge and detail the algorithm design, particularly across the four different strategies that have been applied. Finally, the report will explain the testing and results of this algorithm before concluding on the potential effectiveness it may have with real learners.

Chapter 2

Solution Aim and Algorithm Design

2.1 What is NeuroNudge?

NeuroNudge is a Q-Learning algorithm that applies 4 different strategies to test the viability of Q-Learning in an educational setting for learners with ASD. It aims to evaluate whether Q-Learning is a viable reinforcement learning technique with which learners with ASD are able to not only grow in their ability, but also get tailored recommended difficulties of content based on their unique traits and progress in a module.

NeuroNudge does not provide a learning environment as such, but it does provide a content data model and assumes a few properties of the learning environment:

- to prioritise positive reinforcement, the learning environment that NeuroNudge would operate with is one where each question gives no negative feedback on an incorrect attempt, but does not move forward until a correct attempt is made.
- a lesson will not be marked complete until all questions are attempted in the lesson. As this report takes a simulated testing approach, this is viable - but for a

real-world user, it is understandable that learners may need an early-exit of some kind. However, for the first assessment of NeuroNudge and its viability, this is not an important feature *yet*.

The following sections will provide key definitions that NeuroNudge employs in its content design and then explain how Q-Learning works, before exploring the application of this algorithm to special education.

2.2 Prior Definitions

These definitions are used heavily for many of the discussions that follow for the rest of this report.

- *ContentModule* (or *Module*) - this is a collection of lessons under some common module of content, such as *Actions* or *Shapes* (i.e. copying/recognising actions, or recognising shapes)
- *Lesson* - this is a collection of questions which can be attempted, under a particular lesson of some *DifficultyLevel*
- *DifficultyLevel* - this is an important concept which at this stage, is in a 1:1 relationship with a Lesson. That is, in NeuroNudge the content system is currently assumed to have 1 lesson designed for each difficulty level, and therefore a ContentModule consists of |DifficultyLevel| lessons. Although this may not be a fully realistic representation of the domain, it provides the ability for NeuroNudge to form a functioning MVP which can be easily expanded. In NeuroNudge, the following difficulties exist (in order of easiest to hardest): *VeryEasy, Easy, Medium, Hard, VeryHard, Expert, Master, GrandMaster*.
- *ASDTraits* - this refers to a collection of traits that are often found with which varying levels of competency are found in individuals who have ASD. In NeuroNudge, the focus is on *attention span, communicability, communication level, motor skills*. Communicability simply refers to the ability of a learner to engage in *NonVerbal* and *Verbal* communication. Communication level and

motor skills are simply qualitative scales (e.g. *Low, Medium* and *High* for communication levels). Once again, there are certainly other traits and more precise scales that could be used, but this subset gives the opportunity to form a functioning MVP that can be easily expanded.

The next section will now define the workings of Q-Learning as an algorithm, and then the following sections detail the underlying algorithmic design for each strategy employed by NeuroNudge.

2.3 Q-Learning

Q-Learning is a model-free reinforcement learning algorithm. It is designed to find the best action to take in a given state by estimating the utility of state-action pairs, denoted as Q(s, a). The core idea is to maximize the expected utility of actions, guiding an actor in the system towards the most rewarding behaviors.

2.3.1 How Q-Learning Works

The core equation of Q-learning, known as the Q-learning update rule, iteratively improves the Q-values in a q-table (a simple matrix of state, action and reward values) based on new experiences. The update rule for the Q-value of a state-action pair, after taking some action a and reaching some state s is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2.1}$$

From the above rule, each component refers to the following concepts:

Current Q-Value $Q(s,a)$: Expected utility or total expected reward for taking action a in state s.

Learning Rate α: A value between 0 and 1, where a higher α allows faster learning, quickly incorporating new information.

Reward R: The immediate reward for an action, guiding the learning by reinforcing beneficial actions.

Discount Factor γ: Also between 0 and 1, it quantifies the importance of future rewards compared to immediate ones.

Maximal Future Reward $\max_{a'} Q(s', a')$**:** The best possible future reward in the next state s', considering the consequences of current actions.

Q-learning starts with an initialization of the Q-table and updates it based on the experience gained from actions taken. Each action update uses the Q-learning rule to gradually converge to optimal Q-values, indicating the best actions in each state.

An important part of Q-Learning is also the **policy**. The policy in any reinforcement learning context like this one determines how an agent (in this case, the learning algorithm) selects actions based on the current state. In particular, when applying Q-Learning to a particular problem the definition of the next action that should be taken by the agent is extremely important to the convergence on optimal Q-values. This is where the concepts of **exploration** and **exploitation** also come into relevance. As mentioned above the parameters α and γ balance how much new information supersedes old information vs. the value of current information, hence a weighted balance of exploration vs exploitation.

2.3.2 Example Scenario: a Lizard

To illustrate these concepts and the algorithm with more clarity, consider a $9x9$ grid where a lizard wishes to reach the square with the most food. The idea with q-learning will be that using a balanced approach of exploration vs exploitation, the lizard will go through squares of the grid by choosing an action (from a set of actions it can take in its current square, e.g. it cannot exit the grid). After a particular action, it will receive

a certain reward on the square. For example, if it lands on a square with 1 fly, then the reward will be relatively higher, but if it lands on a square with a bird, then perhaps the reward will be penalised heavily. In this way, a q-table of values is formed and the optimal path of actions to reach its desired goal is formed, all from applying iterations of Q-learning.

2.3.3 Applying Q-Learning to Education

If we now consider an environment where each state is a lesson at some difficulty level, actions are attempting a lesson, and the reward is the level of success a learner has with a lesson, we can adapt Q-learning to our educational context. The more specific relations and inner workings will be detailed in strategy 1 — but to summarise, NeuroNudge aims to create an environment where a q-table of values represents a learner's progress through a module of content. While this won't have a direct alignment with typical applications such as the lizard finding an optimal path to its food, we can instead observe how over time the values of the q-table change and develop a learning profile through this correlation.

However, one critical point in this context is the policy that is chosen. If a learner is using NeuroNudge at 100% exploration, they will receive a new lesson in almost every single iteration because the algorithm will always explore new states - but this will impact learning with a lack of knowledge consolidation. On the other hand, if NeuroNudge is at 100% exploitation, the reverse extreme occurs where the learner will not move away from their current lesson of difficulty meaning they'll never grow and plateau at one difficulty of content.

Therefore, it is apparent that a delicate balance is necessary between these factors, and an intelligent policy is needed to achieve this. This is where the strategies come into play. The following sections will each highlight strategies that explore how Q-learning is adapted by NeuroNudge for the problem domain of learning with ASD. In particular, the details of how each *component* of Q-Learning will be provided, as well as the adjustments that are made for a more contextually aware algorithm.

2.4 Strategy 1: Basic Q-Learning with modified Reward

2.4.1 Q-Learning in NeuroNudge

In NeuroNudge each component of the Q-Learning algorithm is represented as the following:

- $Q(s, a)$: this represents progress of a learner for a particular difficulty level lesson in a module of content.
- s, a: in our context, s is a lesson from a module of content at some difficulty level, and a is the learner attempting a lesson at that difficulty level.
- α: is the learning rate as mentioned earlier. Here, a higher α values new experiences more, ideal for adaptive learning environments.
- R: this represents the immediate feedback received after an action. Given the diverse learning patterns in ASD, the reward must be nuanced, considering factors beyond mere correctness.
- γ: this balances immediate and future rewards. Crucial for long-term learning strategies, especially relevant for learners with ASD where immediate progress might be slow, but long-term gains are significant.
- $\max_{a'} Q(s', a')$ is the best future reward that can be achieved from the next state, encouraging forward-looking learning strategies.
- $Q(s', a')$ this is just the next difficulty level, but the next two strategies below provide different interpretations of this (see NextDiff throughout).

Along with the components above, NeuroNudge defines a policy which balances exploration and exploitation using probability and randomisation. Firstly, the policy requires a function that can output the next level of difficulty from an existing lesson difficulty that a learner has attempted, as well as a function that can output the best possible action a learner can take w.r.t. the entire Q table (i.e., recommend their current best lesson progress).

$$NextDiff(s) = \text{Lesson with the next higher difficulty level than } s \tag{2.2}$$

$$BestAction(s) = \arg\max_{a \in A(s)} Q(s, a) \tag{2.3}$$

These two functions can then be organised in an ϵ-greedy policy whereby at some probability ϵ, this policy of strategy 1 $\pi_1(s)$ will recommend the next action be the next difficulty up, and at probability $1 - \epsilon$, recommend the next action be the best action instead:

$$\pi_1(s, \epsilon) = \begin{cases} NextDiff(s) & \text{with probability } \epsilon \\ BestAction(s) & \text{with probability } 1 - \epsilon \end{cases} \tag{2.4}$$

2.4.2 Reward Calculation

The reward calculation is tailored to assess not just the correctness of an action but also factors like the time taken and the need for hints. This multi-faceted approach aligns with the diverse challenges faced by learners with ASD. The following metrics are key to the calculation of the rewards.

Difficulty Level Constants

To produce a reward that is sensitive to the difficulty of a lesson, some constant weights are pre-defined (between 0 and 1) for every difficulty level.

$$w_d : \text{The weight assigned to difficulty } d \tag{2.5}$$

In order to make the metrics sensitive to the duration of an attempts that learners have on a lesson, some constant time ranges are also pre-defined for each difficulty. This is vital because when a learner attempts a harder lesson, it is contextually aware to ensure that they are not penalised for taking longer than when they attempt a very easy lesson.

Hence, harder difficulties such as *Grandmaster* are assigned with a larger time range, meaning a learner can take more time for harder content and not be penalised.

$$T_{\text{range},d} : \text{The expected time range for difficulty } d \tag{2.6}$$

$$\min T_{\text{range},d} : \text{The minimum expected time for difficulty } d \tag{2.7}$$

Reward for Time

Making use of (3.6) and (3.7), a reward for the time taken in a lesson by a learner can be computed, relative to the difficulty level. The formula $R_{\text{time}}(t)$ accounts for difficulty by reducing the reward slightly if a learner takes more time than the minimum expected but not penalizing them harshly unless the time taken exceeds the typical range significantly. This approach encourages learners to take the time they need while still aiming for efficiency.

$$R_{\text{time}}(t) = \begin{cases} 1 & t \leq T_{\text{min},d} \\ 1 - \frac{t - T_{\text{min},d}}{T_{\text{range},d}} & \text{otherwise} \end{cases} \tag{2.8}$$

Reward for Incorrect Attempts and Hints Requested

The calculations $R_{\text{incorrect}}(i)$ and $R_{\text{hints}}(h)$ ensure that learners are not discouraged by occasional mistakes or needs for hints. The slight decrease in reward for incorrect attempts or hints encourages improvement but avoids harsh penalties that might demotivate learners, especially important for ASD learners who might have unique challenges in communication or motor skills.

$$R_{\text{incorrect}}(i) = 1 - \frac{i}{\text{total\_attempts}} \tag{2.9}$$

$$R_{\text{hints}}(h) = 1 - \frac{h}{\text{total\_attempts}} \tag{2.10}$$

Total Reward

Finally, with a weighted sum, the reward R is calculated in total as follows:

$$R = w_d \times (w_{\text{time}} \cdot R_{\text{time}} + w_{\text{incorrect}} \cdot R_{\text{incorrect}} + w_{\text{hints}} \cdot R_{\text{hints}}) \tag{2.11}$$

2.5 Strategy 2: Apply Mastery Thresholds

The introduction of mastery thresholds in Strategy 2 allows the algorithm to adjust its recommendations based on the learner's mastery level of each difficulty, which is essential for ASD learners who may have uneven skill profiles. It's also essential to having a more consolidated balance of exploration and exploitation, as the policy from Strategy 1 is now modified to explore lessons in the rest of the Q Table contingent on the mastery level that a learner has on their current lesson. In other words, based on certain success metrics at different thresholds of mastery for a content of lessons, a learner will either progress to the next difficulty, stay where they are, or move down to something easier. The following defines these thresholds:

$$M_f = 0.8 : \text{The threshold value chosen for full mastery} \tag{2.12}$$

$$M_c = 0.65 : \text{The threshold value chosen for competent mastery} \tag{2.13}$$

$$M_b = 0.5 : \text{The threshold value chosen for basic mastery} \tag{2.14}$$

Then, these thresholds are applied in the following functions to produce an adjusted R that takes the mastery concept into account:

$$Mastery(R) = \begin{cases} Full & R \geq M_f \\ Competent & R \geq M_c \\ Basic & R \geq M_b \\ None & \text{otherwise} \end{cases} \tag{2.15}$$

$$R_{\text{mastery}} = \begin{cases} 1.0 & Mastery(R) = Full \\ R + 0.1 & Mastery(R) = Competent \\ R & Mastery(R) = Basic \\ R - 0.1 & \text{otherwise} \end{cases} \tag{2.16}$$

Finally, with the adjusted reward the Q Value update function is also changed:

$$Q_{\text{mastery}}(s, a) = Q(s, a) + \alpha[R_{\text{mastery}} + \gamma \max_{a'} Q(s', a') - Q(s, a)] \tag{2.17}$$

As mentioned above, to enable this behaviour the policy for exploring or exploiting must also be modified. To do this, there are three tiers of lessons that can be considered for any state s in this Q learning environment:

$$L_{\text{hs}} : \text{a lesson with higher difficulty than state s} \tag{2.18}$$

$$L_{\text{es}} : \text{a lesson with equal difficulty to state s} \tag{2.19}$$

$$L_{\text{ls}} : \text{a lesson with lower difficulty than state s} \tag{2.20}$$

Hence, by considering the mastery level m that a learner has with their current attempt at state s, the $NextDiff$ component of the original policy can be modified to take

mastery into account. Note: p_{rand} refers to a random probability:

$$NextDiff(s,m) = \begin{cases} L_{\text{hs}} & \text{m} = full \text{ or } m = competent \text{ at } p_{\text{rand}} = 0.6 \\ L_{\text{es}} & \text{m} = competent \text{ at } p_{\text{rand}} = 0.4 \text{ or } m = basic \\ L_{\text{ls}} & \text{otherwise} \end{cases} \tag{2.21}$$

With this adjustment made to the $NextDiff$ component of the policy, the policy for strategy 2 can now be changed to account for the mastery level m a learner has with their current attempt at state s:

$$\pi_2(s,\epsilon,m) = \begin{cases} NextDiff(s,m) & \text{with probability } \epsilon \\ BestAction(s) & \text{otherwise} \end{cases} \tag{2.22}$$

This way, Strategy 2 now provides a Q-Learning environment where the algorithm prioritises mastery of levels when it triggers exploration, so that a learner cannot jump to a new level without mastery. Such an environment is vital for the robust growth of a learner with ASD as they should not progress to higher difficulties of content before they have mastered easier content.

2.6 Strategy 3: Apply Decay

Strategy 3 addresses the need for revisiting and reinforcing previously learned material using a decay mechanism. This is particularly beneficial for ASD learners, who may need repeated exposure to fully grasp and retain information. In particular, if they have learned a particular skill and not revisited this information for a certain period of time, it is vital that they are guided to revisit and confirm mastery of their skills, before it is completely committed to memory.

In this way, NeuroNudge takes on an exponential backoff strategy with easier levels asking for less reinforcement from the learner, and harder levels asking for more - and

so once a learner has completed easier levels a certain number of times **with a certain time period of gaps in between**, it assumes full mastery of the difficulty level.

To start with, a few constants are predefined for each difficulty level. A decay count is set for each difficulty level d where it should have a certain number of times that the Q value should drop. For example, if the Easy difficulty should drop at least twice so that a learner reinforces their knowledge of the Easy lesson at least twice, then the decay count for Easy starts at 2.

$$D(d) : \text{Decay count for difficulty level } d \tag{2.23}$$

To appropriately reinforce a difficulty level, there should be a minimum amount of time that a learner has spent away from the level before it is decayed. The reasoning behind this is as follows: suppose a learner has completed the Easy difficulty to a full mastery. Then, the algorithm in its current form will not return to this level unless the learner struggles with the next level (Medium) and by the mastery thresholds, is directed to return to the lower difficulty (Easy). So, if there is a situation where the learner never struggles enough to return to VeryEasy, they may never revisit that knowledge ever again.

So, to reinforce and revisit knowledge to a certain degree of competency, the concept that NeuroNudge introduces is *non-attempts*. This simply refers to the number of iterations of lesson attempts since the last time a learner completed a lesson at that difficulty level. The idea is that after a certain number of non-attempts, we expect the difficulty d to be revisited because it has not been attempted again with enough recency.

$$N(d) : \text{Non-attempts counter for difficulty level } d \tag{2.24}$$

$$R_{\text{na}}(d) : \text{Required non-attempts to apply decay for difficulty level } d \tag{2.25}$$

Hence, with these defined functionalities we simply update non-attempts counter $N(d)$ each attempt for all difficulties d at each call of the update function defined

above in Strategy 1:

$$N(d) = \begin{cases} N(d) + 1 & \text{if lesson not attempted} \\ 0 & \text{if lesson attempted} \end{cases} \tag{2.26}$$

Now that the setup is defined for the decay counting and non-attempts counting, it is necessary to define the decay rate $\delta(d)$ for some difficulty level d. In this case, it is simply inversely proportional to the decay counter because the more decays that are still left, the stronger the decay should be - as this will enable the exponential backoff for a learner (the more that knowledge is reinforced, the less significantly it will decay).

$$\delta(d) = \frac{1}{D(d)} \tag{2.27}$$

The final step is to apply this decay rate to each Q value when a specific condition is met. This is when there are enough non-attempts for the difficulty level, and the difficulty level has decays still left to be applied. To apply the decay rate, it is a simple multiplication on top of the existing update Q-value function from Strategy 2 as both the Q-value and decay rate are between 0.0 and 1.0 by design:

$$Q_{\text{decay}}(s,a) = \begin{cases} Q_{\text{mastery}}(s,a) \times \delta(d) & \text{N}(d) \geq R_{\text{na}}(d) \;\; \& \;\; D(d) > 0) \\ Q_{\text{mastery}}(s,a) & otherwise \end{cases} \tag{2.28}$$

Note that if decay has been applied for a difficulty d, then we require to decrement the decay counter $D(d) \leftarrow D(d) - 1$ and reset the non-attempts $N(d) = 0$, because the non-attempts must be re-counted again.

The outcome of applying the defined steps of decay from above is that after a certain pre-defined period of non-attempts for each difficulty level, they will decay to a lower level and "weaken" (i.e. the learner's progress is forcefully decreased). However, if the policy π_2 from Strategy 2 is not modified to redirect exploration towards these weakened

lessons, there is a very high probability that easier difficulties after being decayed will be ignored and the algorithm will never drop down to these levels. Hence, a new policy is necessary that prioritises the weakest levels - and bringing this into context, it is a sensible modification because when a learner has stayed away from content for a while they should revisit that content as immediately as possible and confirm their mastery.

Hence, the first part of introducing a new policy π_3 that accounts for this prioritisation is a simplistic function that finds the current weakest level. Note that we choose the lesson with lowest Q-value **at difficulty levels that have already been attempted** (this can be kept track of with a simple map between difficulty levels and a boolean flag).

$$WeakestLevel() = \begin{cases} \text{Min Q-value from attempted lessons} & \text{if it exists} \\ \text{None} & \text{otherwise} \end{cases} \tag{2.29}$$

Then, the adjusted policy π_3 can be formed where $NextDiff$ will prioritise the weakest level found using $WeakestLevel$:

$$\pi_3(s, \epsilon) = \begin{cases} NextDiff(s) & \text{with probability } \epsilon, \text{ prioritise } WeakestLevel \\ BestAction(s) & \text{otherwise} \end{cases} \tag{2.30}$$

2.7 Strategy 4: ASD Traits Sensitivity

The final strategy of NeuroNudge is to add sensitivity to the unique ASD traits which each learner may have. So far, we have theoretically achieved a Q-learning environment that is able to prioritise mastery of levels, and reinforces content that has not been revisited in a certain time period (based on its difficulty). While this may be adaptive enough and will uniquely treat each learner based on their attempts at different question styles, a small modification should be made in two different contexts:

1. Define thresholds for common ASD traits that each question requires - for example, it may define the minimum level of attention span that the question needs from a learner in order for success to occur
2. Develop a profile on the same ASD traits for each learner so that the impact of their traits on their learning can be analysed along with the Q-values vs time
3. While NeuroNudge does not have access to real learners who have unique ASD traits and potential impacts on their cognitive function, the simulated testing environment must craft different ASD trait profiles and ensure that the simulation of a learner's attempt at questions are affected by their traits. (This will be further addressed later in testing - if real learners are to use the algorithm, then the simulated environment of lesson attempts would not be used.)

Hence, at this stage of NeuroNudge's development Strategy 4 simply requires the addition of thresholds to the question data model and a probabilistic application of simulated lesson attempts that account for the potential effect that ASD traits will have depending on the required competencies a question may have.

This means there are no algorithmic design changes to the Q-value update function, Reward calculation or policy π_3. Therefore this strategy and need for ASD Traits Sensitivity will be explained in the next section where the simulated environment is introduced.

Chapter 3

Testing and Results

Ideally, testing the NeuroNudge algorithm would be best achieved by interacting with real learners who have ASD and monitoring whether the suggestions the algorithm gives is aligned with the opinion of experts in special education. Without access to this, the best possible testing environment for NeuroNudge is to simulate all actors of the system. This includes:

- Learners - creating mock learners with varied ASD traits
- Content - creating mock content modules with lessons that increase in difficulty level
- Learning Experience - creating a mock learning environment where iterations of learning occur for each learner, and probabilities map out likely successes in each lesson delivered at a particular difficulty level

With this in mind, the following section will detail how the environment above was set before summarising the results of the 4 different strategies that have been applied (defined in Chapter 3).

3.1 Testing Methodology

The testing of NeuroNudge was done by the following: first, simulated learners and content was defined based on the data model defined earlier in Important Definitions (alongside other definitions which can be accessed in the Rust project implementation of this algorithm. Please note that the way NeuroNudge has been designed does not specifically require Rust, this language was chosen for the implementation and testing for its speed with optimised builds and runtime). Then, an environment was set up where $N = 5000$ iterations run within which each learner individually attempts a lesson and from the strategy applied in the NeuroNudge algorithm, has a particular set of logic that updates their Q-table and provides a recommendation for their next lesson.

Hence, the following sections will detail how the simulated learners, then simulated content, and finally simulated learning environment were created.

3.1.1 Simulating Learners

In NeuroNudge a Learner has basic personal information such as their age and name, and a mapping of their ASD traits. To simulate tests of NeuroNudge in a manner that provides breadth across different learning profiles, the goal in simulating learners was that each learner created should provide a unique collection of ASD traits. This way, trait sensitivity will have a more detectable impact on the results of the algorithm and an analysis of the algorithm's degree of success can be more easily accomplished.

The following table summarises how 6 learners were simulated with a variation in their traits. Note: for formatting reasons, some of the column names have been abbreviated - AS = Attention Span, Communication Level = CL, Motor Skills = MS:

Name	Age	AS (min)	Communicability	CL	MS
Learner 1	7	5	Non-Verbal	Low	Low
Learner 2	8	6	Non-Verbal	Medium	Low
Learner 3	9	7	Non-Verbal	Medium	Medium
Learner 4	10	9	Non-Verbal, Verbal	Medium	High
Learner 5	11	12	Non-Verbal, Verbal	High	High
Learner 6	12	15	Non-Verbal, Verbal	High	Very High

Table 3.1: Summary of Simulated Learners in NeuroNudge

3.1.2 Simulating Content

To simulate a learning environment for learners, content is also necessary. To do this, ChatGPT [3] was used so that using a template lesson (example code snippet below from the Rust project of a Very Easy lesson of Shapes), lessons in ascending difficulty with different *sensitivities to levels of ASD traits* could be generated.

```
// Very Easy lesson: "Recognising Circles"
let very_easy_lesson = Lesson::new(
    "Recognising Circles".to_string(),
    (0..6)
        .map(|i| {
            let asd_traits = ASDTraits::new(
                "".to_string(), // No learner_id to attach to this
                1,
                vec![Communicability::NonVerbal],
                CommunicationLevel::Low,
                MotorSkills::Low,
            );

            if i < 3 {
                generate_question(
                    "Select the circle!",
                    CIRCLE_IMAGE, // Answer
                    vec![], // Distractors
```

```
                    Some(asd_traits.clone()),
                )
            } else {
                generate_question(
                    "Select the circle!",
                    CIRCLE_IMAGE, // Answer
                    vec![SQUARE_IMAGE], // Distractors
                    Some(asd_traits),
                )
            }
        })
        .collect(),
    DifficultyLevel::VeryEasy,
    "Shapes".to_string(),
);
```

In the calls to the function *generate_question* the second and third parameters also account for a key concept in designing content for learners who have ASD: distractors. While learners with ASD learn content with more repetition than others may require, it is important to note that repeating the same question without changes in the environment will cause a learner to remember answers for the wrong reasons. That is, they will be unable to apply a concept they have effectively rote learnt in one environment, to other environments where the concept is still prevalent.

Hence, the use of distractors - objects that are there to distract from the main answer - allows for learners to acquire knowledge of a particular concept within different environments. Although NeuroNudge does not simulate any UI (user interface) where learners may interact, the idea it proposes is that both the set of distractors and the real answer can be placed on the screen in randomised positions to create new environments for each question. This way, although the above lesson is *VeryEasy*, in such a UI the last 3 questions would use a circle and square with different,

randomised positions to test if the learner can still identify the circle.

To see how the lessons ascended in difficulty, refer to either *simulated_content_shapes.rs* or *simulated_content_actions.rs* in the *engine* crate of the Rust project. In summary, for a module like Shapes each question simply introduces more complex shapes as the difficulty rises and also increases the total number of questions required to complete the lesson.

3.1.3 Simulating Learning

With simulated learners and content, the final step for testing NeuroNudge was to simulate a learning environment. Here, the key step was to define how a lesson attempt can be generated for each lesson by each learner as this occurs for each iteration (as mentioned in the introduction of this section).

Recall from the previous section that for Strategies 1-3, no trait sensitivity is directly built into the simulated environment and this is only done with strategy 4. Hence, the definition of the function *simulate_lesson_attempt* used in the simulated learning environment can be explored via its base case and its adjusted form for strategy 4. To see the full fledged simulation environment constructed to test NeuroNudge, see *simulate.rs* in the *engine* crate.

Base Case Simulation

For the base case simulation, we must define how to calculate the total time taken T for a lesson, and the probability of correctness C for a student:

$$\mathrm{T}(d) = rand\ \mathrm{T}_{\mathrm{range}}(d) \tag{3.1}$$

Where:

- $\mathrm{T}(d)$ is the time taken for difficulty level d.
- $\mathrm{T}_{\mathrm{range}}(d)$ defines the pre-defined expected time ranges for each difficulty level d.
- *rand* selects a random value within the specified range.

The probability of correctness C for each student refers to the probability at which they will successfully attempt a question. Recall that NeuroNudge assumes no question can be progressed without choosing the correct answer - hence, this factor is compared with a sequence of randomly generated probabilities which ends when the generated probability is less than this factor.

Bringing this more specifically into the learner's context, it is important to note that the level of difficulty should be proportional to the rate of success. That is, a simulated learner's chance of getting a correct attempt should be much higher for easier lessons. Hence, some base success for a difficulty level d, $S(d)$, is required for the correctness factor. However, continuing to align with context, it must also be recognised that while a Hard lesson should have a lowered probability of success, a learner who is gaining mastery in this difficulty should not be continually predicted to have a low chance of success, which means their progress should also be accounted for in order to simulate an environment where mastered content will increase the rate of success at completing that content if given again.

This leads to the definition of a weighted sum for the correctness factor C so that the current Q-value at difficulty d, $q(d)$ does not dominate the correctness factor. A small weight $w_{\mathrm{q\_value}}$ is used and when tested with real learners, may be subject to more adjustments that may better recommend lessons:

$$C(d, q) = S(d) + w_{\mathrm{q\_value}} \cdot q(d) \tag{3.2}$$

Now that the correctness factor and total time taken have been simulated, the last step in this environment is to loop through each question in a lesson and simulate its attempt. With the correctness factor we estimate a certain number of incorrect attempts (if any), and with the total time taken for the lesson we simply give an average expected time

per question.

With this approach, a learning environment is simulated where learners are attempting questions with a level of success that is proportional to the difficulty of the lesson. However, as discussed in Strategy 4 in the prior section, this base case does not take the unique traits of each learner into account. This requires the introduction of an adjusted time taken and correctness factor.

Inclusion of Strategy 4

When Strategy 4 is applied, the traits of the learner and how well they *align* with the trait parameters defined within each question is vital to the correctness factor. Moreover, the time calculation is also adjusted based on the attention span of a learner.

That is, if a learner has an attention span that is higher than the expected time that the lesson will take, then the total time taken for a lesson at some difficulty d is unchanged from $T(d)$ defined above. However, if the attention span is smaller, an exponential trend should be applied. In this way, if a lesson is barely larger than the learner's attention span then the penalty on the duration they are likely to take should be minimal. On the other hand, if they are far less focused than is required for the question, the penalty must increase at an increasing rate. This effect is calculated with $ASAdjustment(a, t)$ for some given attention span (in minutes) a and (randomly generated) expected time t.

$$T_{\text{as\_adjusted}}(d, a) = \begin{cases} ASADjustment(a, T(d)) & a \leq \text{AS} \\ T(d) & \text{otherwise} \end{cases} \tag{3.3}$$

To define $ASAdjustment(a, t)$, an exponential factor $\epsilon = 1.2$ is used in the following setup:

$$excess\_time = t - a \tag{3.4}$$

$$ASAdjustment(a,t) = \begin{cases} \text{t} + \epsilon \cdot excess\_time & \text{if } t > a \\ \text{t} & \text{otherwise} \end{cases} \tag{3.5}$$

As such, the more distant a learner's attention span is from a question, the higher the simulated value of the total time they have taken in the question.

For the correctness factor, the key concept to introduce is the use of a trait alignment score. This refers to the calculation of how well a learner's traits align with the trait parameters expected for higher probabilities of success in a question (as we saw in the example lesson from the code snippet in the content simulation section).

This brings forward another weighted sum as certain parameters have more importance than others. Note that while attention span has already been accounted for in the total time taken in the lesson, it has not been accounted for in the correctness factor, so this trait alignment score still uses attention span. Hence, the following weights are defined for each ASD trait that is currently defined in NeuroNudge. The weights sum to a total of 1.0 – signifying complete alignment:

$$w_{\text{as}} : \text{weight for the attention span alignment factor} \tag{3.6}$$

$$w_{\text{c}} : \text{weight for the communicability alignment factor} \tag{3.7}$$

$$w_{\text{cl}} : \text{weight for the communication level alignment factor} \tag{3.8}$$

$$w_{\text{ms}} : \text{weight for the motor skills alignment factor} \tag{3.9}$$

The calculation of the attention span alignment factor $align_{\text{as}}(as_{\text{learner}}, as_{\text{question}})$ is simply done by a percentage that maxes out at 100% - as the goal is not to consider how much better the attention span of a learner is than a question (if that is the case), rather how well it is fit to answer the question:

$$align_{\text{as}}(as_{\text{learner}}, as_{\text{question}}) = \frac{as_{\text{learner}}}{as_{\text{question}}} \tag{3.10}$$

The calculation of the communicability alignment factor is again best done as a percentage. NeuroNudge currently assumes that if a learner has more than no communicability, it must start with having NonVerbal communication and then Verbal. Hence, a simple percentage calculation is made where the total communication levels expected in the question is divided by the total common communication levels of the learner:

$$align_{\text{c}}(c_{\text{learner}}, c_{\text{question}}) = \frac{|c_{\text{learner}} \cap c_{\text{question}}|}{|c_{\text{question}}|} \tag{3.11}$$

Based on 3 defined levels of communication level (Low, Medium, High), the communication level alignment factor is simply evaluating comparisons between these 3 possibilities. Put simply, if the level of the learner is 2 below the level required (such as Basic vs High), the alignment score is given by 0.25. 1 below is given by 0.5, and equal or higher is given by 1.0:

$$align_{\text{cl}}(cl_{\text{learner}}, cl_{\text{question}}) = \begin{cases} 1.0 & cl_{\text{learner}} \geq cl_{\text{question}} \\ 0.5 & cl_{\text{learner}} < cl_{\text{question}} \\ 0.25 & cl_{\text{learner}} << cl_{\text{question}} \end{cases} \tag{3.12}$$

While these do appear as magic numbers, in this simulated environment of testing the goal is to simply provide observable relative differences in traits so that the results can indicate the reaction of NeuroNudge when it is sensitive to the ASD Traits of each learner and each question.

The motor skills alignment factor $align_{\text{ms}}(ms_{\text{learner}}, ms_{\text{question}})$ is calculated in the same fashion, except across 4 different levels of Low, Medium, High and VeryHigh, and differences of 3 levels yields an evaluated factor of 0.1225.

Finally, the overall alignment factor can be calculated using a weighted sum from these values:

$$align_{\text{total}} = w_{\text{as}} \cdot align_{\text{as}} + w_{\text{c}} \cdot w_{\text{cl}} \cdot align_{\text{c}} + w_{\text{ms}} \cdot align_{\text{cl}} + align_{\text{ms}} \tag{3.13}$$

Now that we have $align_{\text{total}}$, the correctness factor calculation can be made ASD trait sensitive in the simulated learning environment, simply by multiplying the correctness factor with the total trait alignment score $align_{\text{total}}$:

$$\text{C}_{\text{adjusted}}(d, q, t_{\text{learner}}) = \text{C}(d, q) \times align_{\text{total}} \tag{3.14}$$

In the simulated tests run with strategy 4, it was observed that this adjusted correctness factor had one minor issue - no sensitivity to the potential for growth after consecutive attempts at the same content. That is, because higher difficulties produce a low $S(d)$ and have a low $q(d)$ in the learner's q table, the original correctness value of $C(d, q)$ defined earlier yields a small decimal which may sometimes be 0 as well. In this situation, the simulation will stall because the learner can never get a question right when the correctness factor becomes so small, especially after $C_{\text{adjusted}}(d, q, t_{\text{learner}})$ applies the alignment factor which is at most 1.

Hence, another counter was introduced into the Q-learning environment whereby consecutive attempts of each difficulty d are managed, and this counter is then normalised when calculating the adjusted correctness to provide a growth factor. That is, after doing a harder level say 100+ times, we can expect that a learner will have learned more content than when they started the level. So, using a normalised consecutive attempts value $n(d)$, the final definition for the correctness factor became:

$$\text{C}_{\text{adjusted}}(d, q, t_{\text{learner}}) = \text{C}(d, q) \times (align_{\text{total}} + n(d)) \tag{3.15}$$

3.2 Results

The results of each strategy is summarised with its ability to accomplish the following core goals for learners with ASD:

Goal 1 Make progress: with a good balance of exploration vs exploitation, learners should eventually make progress through each difficulty level of learning.

Goal 2 Master current concepts: coupled with (1), learners should make progress to higher difficulty levels $\iff$ they have mastered the lower difficulty levels.

Goal 3 Reinforce concepts after time: coupled with (2), NeuroNudge should not assume eternal mastery over knowledge. Once a difficulty has been mastered, a learner should be made to revise that knowledge after an appropriate amount of time has passed.

Goal 4 Develop a trait-sensitive profile: the algorithm should develop a q-table of values that are representative of how the student's traits affect their own learning.

While all figures are provided in the appendix for each learner, the following results sections will only make selective comparisons which represent the overall trend. However, if further visual evidence of this overall trend is desired, then refer to Appendix A. Note that each figure has a left column of plots where each plot is for an individual difficulty level and shows the Q values at that level (i.e. the learner's progress in the level) vs iterations over time, and a right column that shows which difficulty was attempted in each iteration by the learner.

3.2.1 Strategy 1: Summary of Results

To recap, Strategy 1 is the application of basic Q Learning with a specific reward calculation that takes the learner's level of success in attempting each question into account.

Unfortunately for Strategy 1, what happens for all 6 learners is evidenced in figure 3.1. The algorithm simply recommends the next difficulty after very few iterations of

learning and then plateaus at the final difficulty, because there is no higher difficulty level left. Shown are just the results for learner 1 after 5000 iterations of the algorithm. This was done on a value of $\epsilon = 0.3$ for the exploration vs exploitation balance.

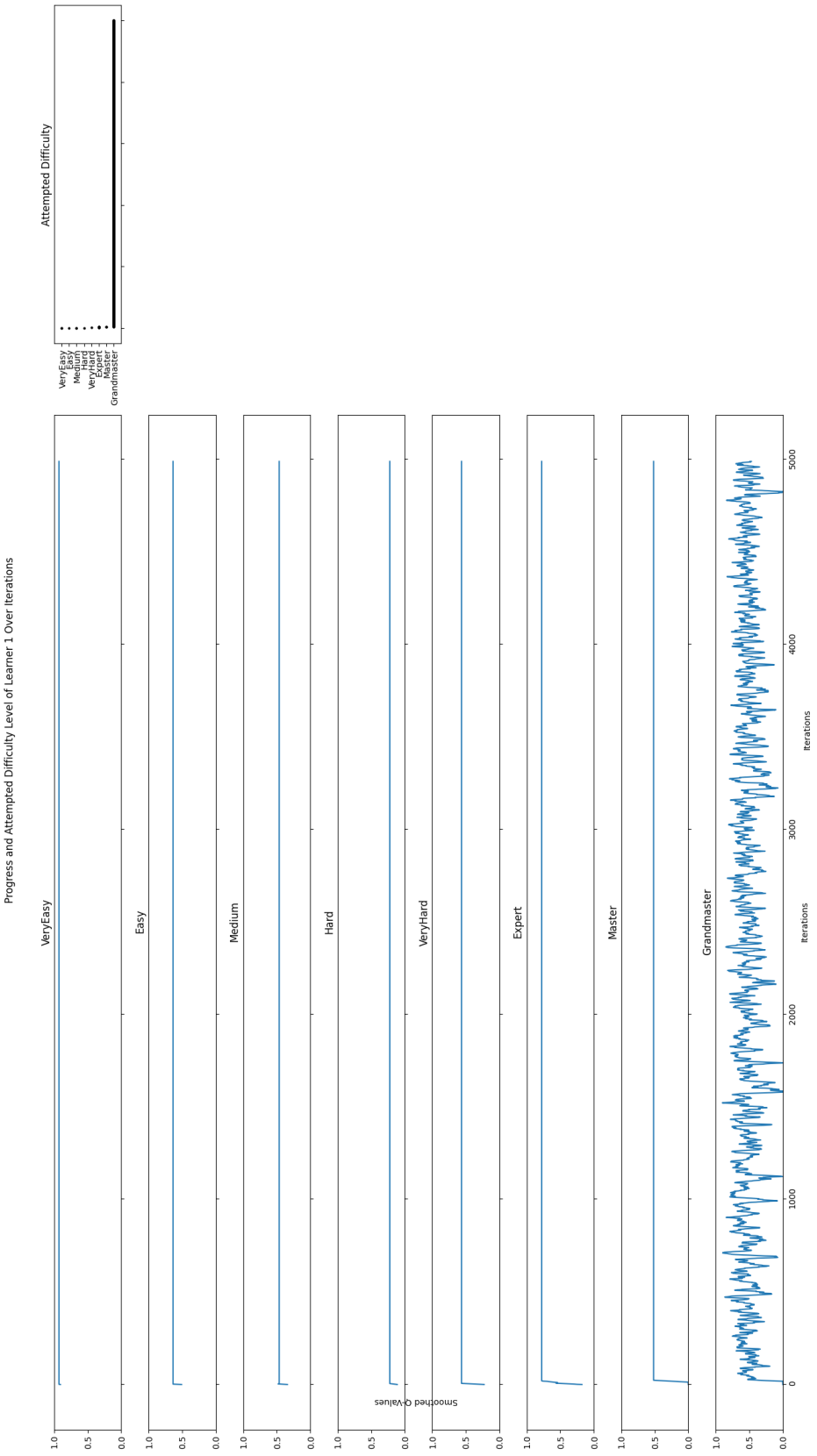

Figure 3.1: Example of a learner's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.3$

This points out that Strategy 1's policy is ineffective in exploiting existing knowledge and too heavily inspires exploration, revealing that the original definition of *NextDiff* where the next highest difficulty is the only priority, is not a good choice for this context.

Based on this being a problem of exploration vs exploitation, if we change ϵ to a higher value such as 0.6, then perhaps exploration will be inspired - however, because *NextDiff*, the exploration case, does not choose a random difficulty and the next highest one instead, the same result is attained (as visible in figure 3.2)

Furthermore, it is not contextually viable to select a random difficulty for a learner as it is not ideal for an inexperienced, young learner to be demotivated with potentially harder content delivered to them very early in their learning experience. So, evaluating strategy 1 based on the goals, the following is understood:

1. Goal 1 - Yes, strictly speaking, the learner does make progress
2. Goal 2 - No, the learner does not master concepts before moving to a higher level. In both figures thus far, it can be observed that some of the q-values are low and yet the algorithm progresses the learner forward. This is not viable for learning. For example, see the Hard difficulty in figure 3.2 where Learner 1 only reaches 0.5 and is recommended to move ahead.
3. Goal 3 - No, the learner never goes back to difficulty levels they have previously attempted and "completed".
4. Goal 4 - No, the algorithm has no sensitivity to the learners' traits.

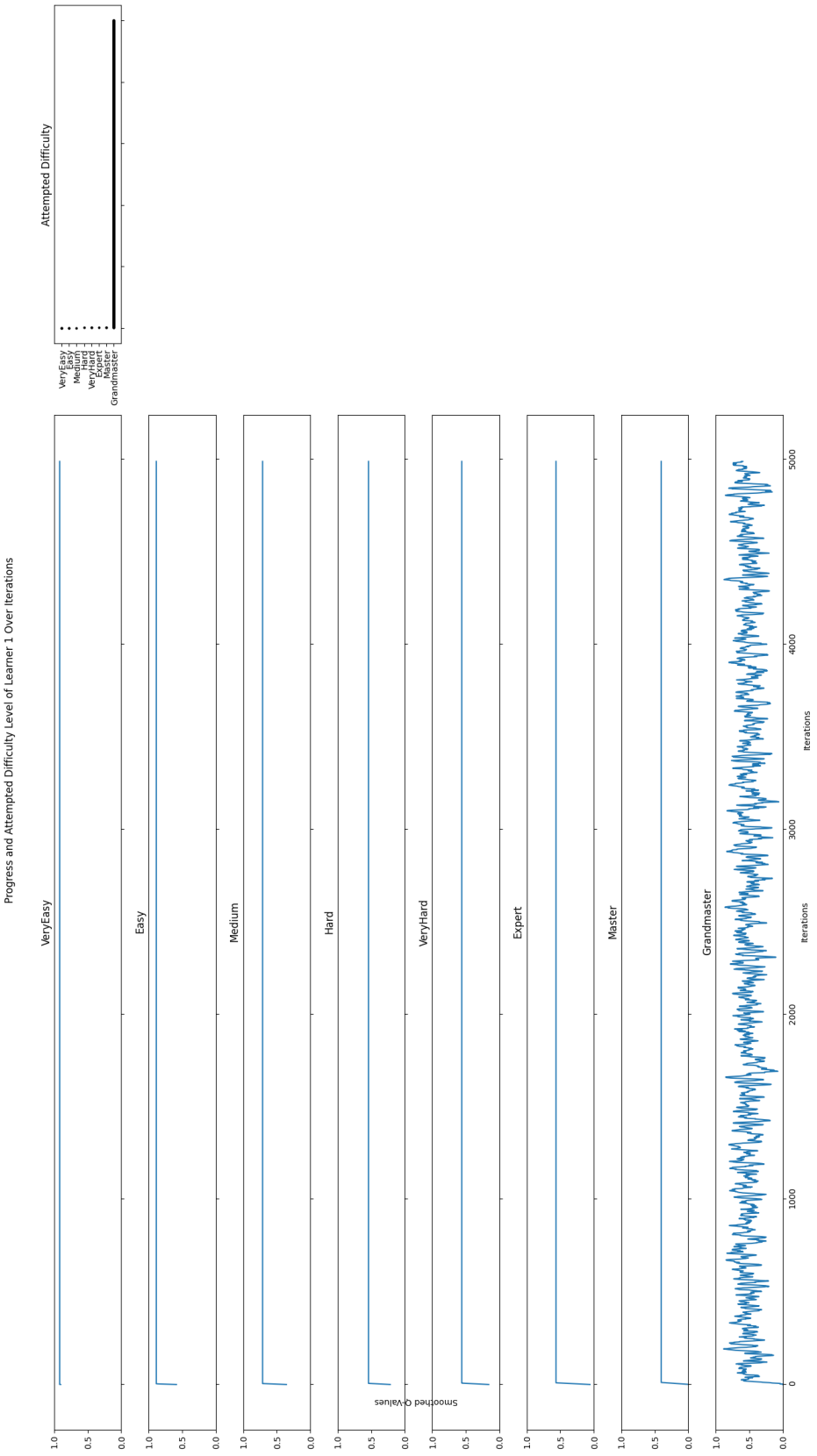

Figure 3.2: Example of a learner's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.6$

Hence, as this is not a viable algorithm for their learning experience, we turn to Strategy 2 next.

3.2.2 Strategy 2: Summary of Results

To recap, strategy 2 is where mastery thresholds are introduced into the policy and reward calculation of the Q-learning algorithm from strategy 1. In this strategy, the results are quite evidently a major improvement where learners stay at existing difficulty levels a lot more, until they achieve mastery, before moving ahead.

Furthermore, as observable from the attempted difficulties graph, it is clear that if a certain difficulty is incompetently attempted by a learner, they are moved back down to the easier level to remaster the easier concepts before moving forward. This is a great result for the context that this algorithm wishes to solve with learners who have ASD. This trend is established across all learners, but again a simple example with just learner 1 is provided below in figure 3.3.

In the figure, it is strongly evident that as the difficulty level rises, the time it takes for mastery is increased and the level of progress (the q-values and their closeness to 1.0) is much more fluctuating. In other words, it is certainly observed that each learner masters the easier content quicker, and fluctuates over time and finds that much more time is needed to master the harder difficulty levels. The most significant difference between this and strategy 1 can be observed by comparing the attempted difficulty plots across both strategies.

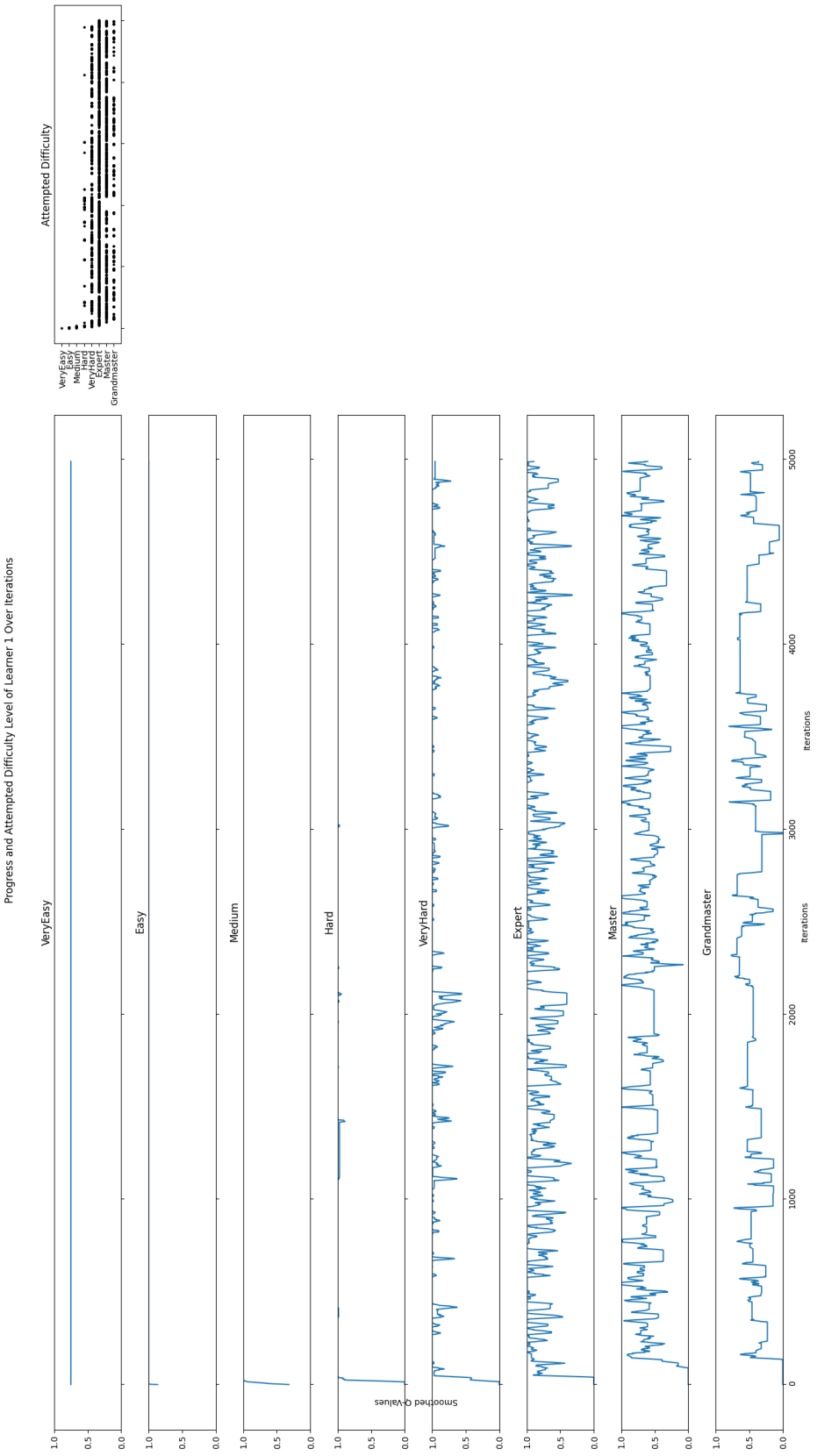

Figure 3.3: Example of a learner's q table and attempted difficulties over 5000 iterations of Strategy 2.

However, it is noticeable that the learner does not revisit previously completed knowledge to reinforce that content, even after a large number of iterations have passed. This means that NeuroNudge still isn't viable for a learner with ASD because there is a higher probability through this model that the learner will not fully remember the older, easier knowledge that they have mastered, after a certain period of time. Furthermore, we still observe the same trend of learning across all 6 learners despite the unique traits that each learner has - which is a highly inaccurate representation of how learning should occur for these learners in reality.

So, evaluating strategy 2 based on the goals, the following is understood:

1. Goal 1 - Yes, strictly speaking, the learner does make progress
2. Goal 2 - Yes, the learner progresses only after mastering a difficulty level
3. Goal 3 - Partially, the learner can sometimes move down to easier levels if they have incompetently attempted harder levels, but they never go back to the easiest levels to reinforce that knowledge
4. Goal 4 - No, the algorithm has no sensitivity to the learners' traits.

Hence, while it is an improvement, this is not a viable algorithm for their learning experience and we turn to Strategy 3 next.

3.2.3 Strategy 3: Summary of Results

To recap, strategy 3 introduces the concept of decay so that previously mastered knowledge can be revisited and reinforced - and it involves a modification of the q learning update function with smart counters that keep track of how much time has passed since a previously mastered difficulty level has been attempted. In this strategy, the results provide major evidence of another improvement to the learning experience.

Paying close attention to both the difficulty q value subplots and the attempted difficulties graph, it is clear to see that dips occur in the q values of the difficulties

with a certain trend. This trend is by design in the strategy, whereby easier levels dip less frequently because they are easier to master, whereas the harder levels dip more frequently because they are not as easy to master - but, these dips only occur if mastery has been achieved in this difficulty level before.

Note that for harder difficulties (VeryHard and higher), it can be deduced that most learners do not achieve mastery for long enough periods to cause decay, and its through the typical balance of exploration and exploration with the policy pi_3 that the learning progress continues to fluctuate due to the lowered probability of successes at higher difficulties. However, it is clearly observable in the VeryEasy to Hard levels for in figure 3.4 that a dip and corresponding revisit (if you cross-reference with the attempted difficulty plot) does occur more frequently as the difficulty levels rise.

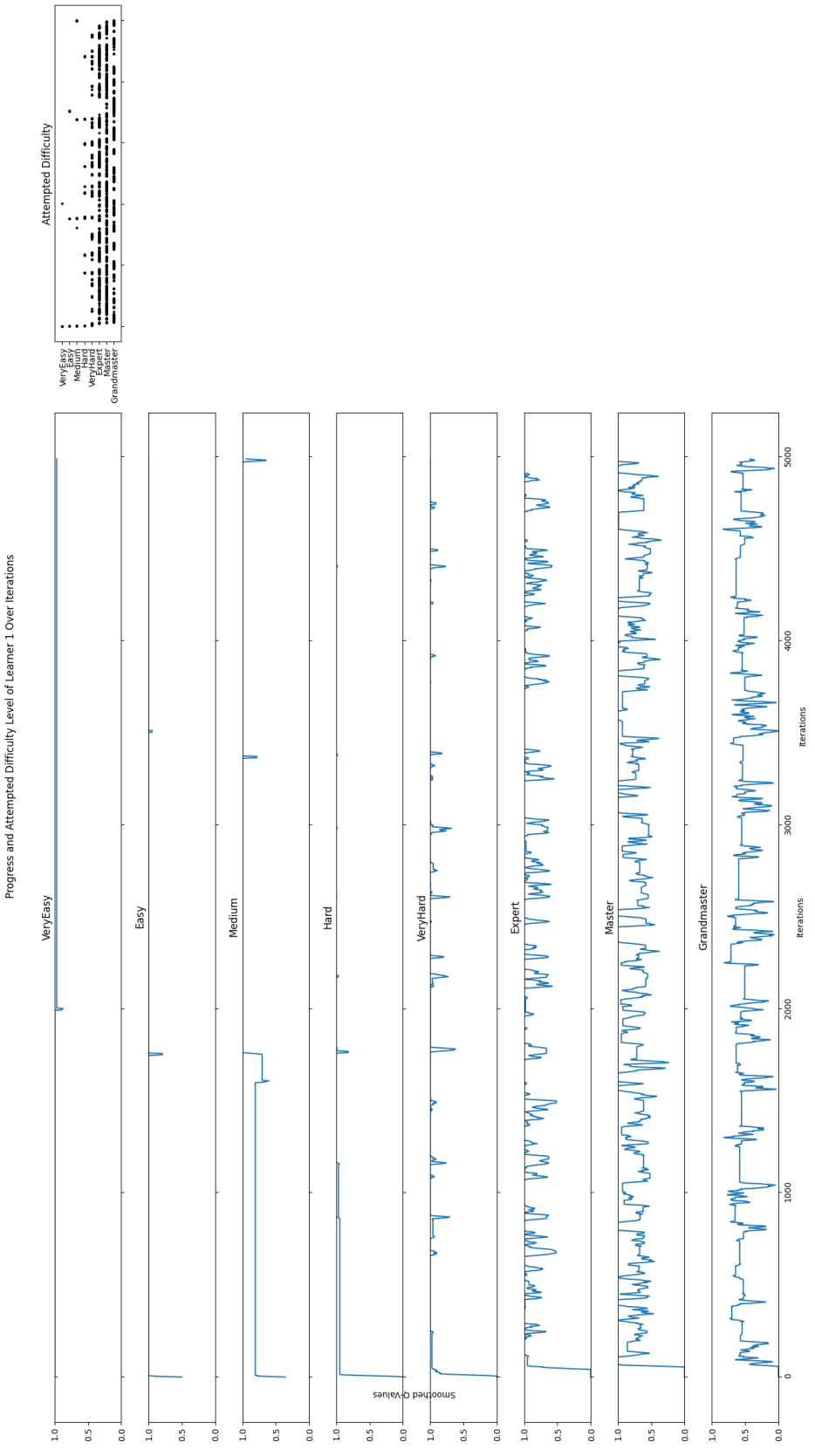

Figure 3.4: Example of a learner's q table and attempted difficulties over 5000 iterations of Strategy 3.

However, there is one final flaw which is noticeable - the fact that all 6 learners have the same trend in this strategy is still not a reflection of the reality that each learner should have a unique profile based on their ASD traits. So, evaluating strategy 3 based on the goals, the following is understood:

1. Goal 1 - Yes, strictly speaking, the learner does make progress
2. Goal 2 - Yes, the learner progresses only after mastering a difficulty level
3. Goal 3 - Yes, the learner revisits and reinforces previously mastered knowledge
4. Goal 4 - No, the algorithm has no sensitivity to the learners' traits.

Hence, while this may be the optimal policy and Q-Learning algorithm design for the problem, the simulation environment itself needs to provide clarity on the sensitivity that the algorithm has to a learner's ASD traits, leading to the need for strategy 4.

3.2.4 Strategy 4: Summary of Results

To recap, strategy 4 does not claim that strategy 3 falls short in its algorithm design for the problem at hand. Instead, due to the inaccessibility of real learners with ASD, the simulated environment must be carefully tweaked to have sensitivity to a learner's unique traits and its alignment with the minimum levels of competency required for each trait in a particular question. Strategy 4 accomplishes this with a redesign of the manner in which a lesson attempt is simulated, and successfully, proves clear differences between a learner with weaker and stronger levels of the ASD traits currently captured in NeuroNudge.

Across figures 3.5 and 3.6, we can observe the differences between Learner 1 and 6 in the simulated learning environment when attempting the *Actions* module. Despite the high degree of random probabilities that are used for generating responses from learners in the simulated learning environment (covered prior in section 4.2.3), there is visual evidence of learner 6 completing lessons with much more ease. While the fluctuations of success still exist for both learners in harder difficulties, they occur at a far lower variability for learner 6 than it does with learner 1, who is clearly struggling a lot more

with higher difficulties.

Why are they struggling more? Well, consider the following requirements of ASD traits for lessons at the harder difficulties in the *Actions* module:

Difficulty	**AS (mins)**	**Communicability**	**CL**	**MS**
Hard	7	Non-Verbal	High	Medium
VeryHard	10	Verbal, Non-Verbal	High	Medium
Expert	12	Verbal, Non-Verbal	High	High
Master	15	Verbal, Non-Verbal	High	Very High
GrandMaster	20	Verbal, Non-Verbal	High	Very High

Table 3.2: ASD Trait Requirements for across Difficulty Levels in *Actions*

Evidently, comparing this to the traits of learners 1 and 6 from table 3.1, learner 6 is much better suited to success for these lessons in *Actions*. This is why the subplots on the left side of figures 3.5 and 3.6 look so different for learners 1 and 6. Note that the remaining fluctuations in easier levels are simply due to the decay that we already saw earlier from the application of strategy 3. In Appendix A, notice the same trend occurs as each plot is compared from Learner 1 all the way to Learner 6.

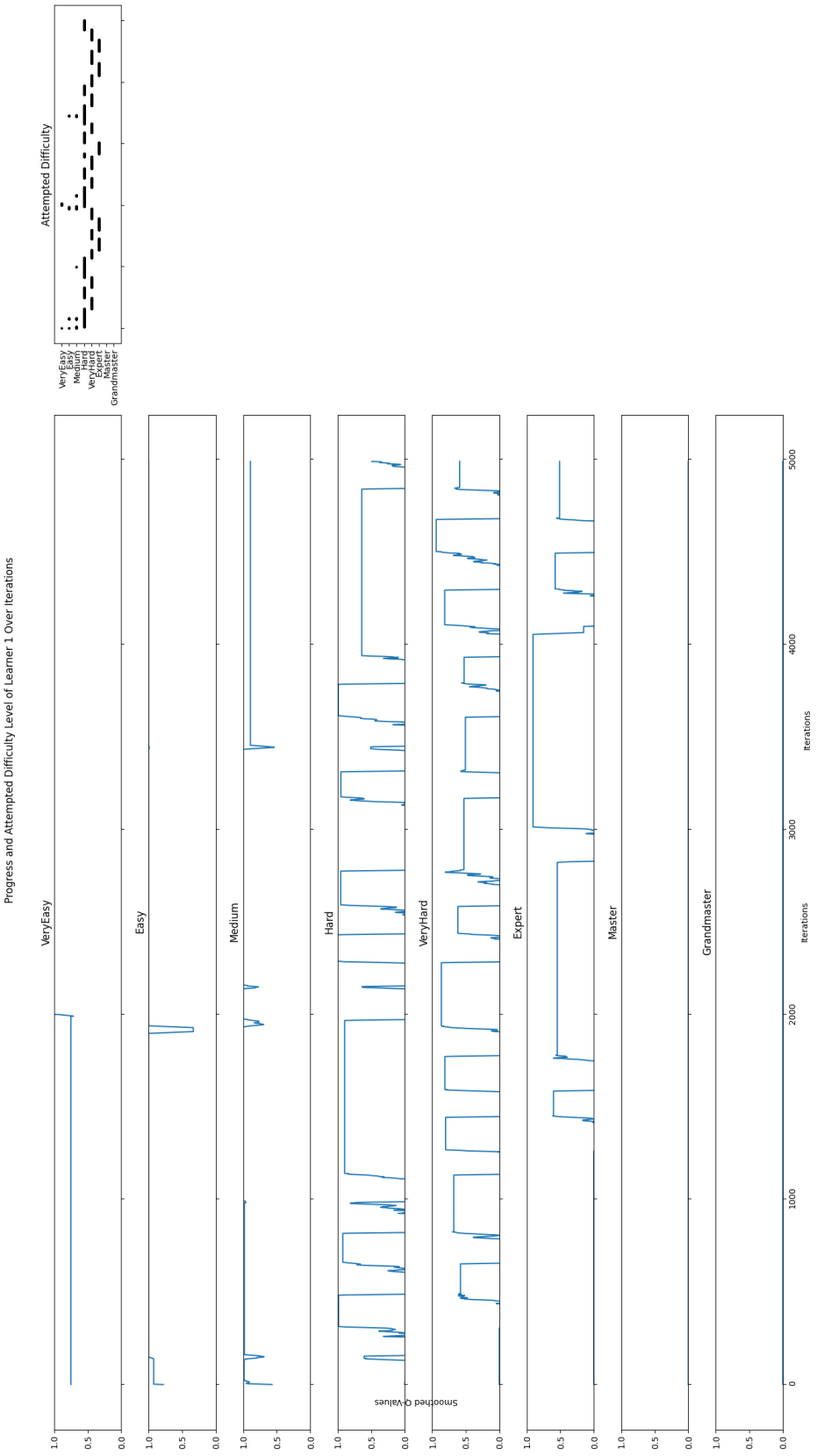

Figure 3.5: Learner 1's q table and attempted difficulties over 5000 iterations of Strategy 4.

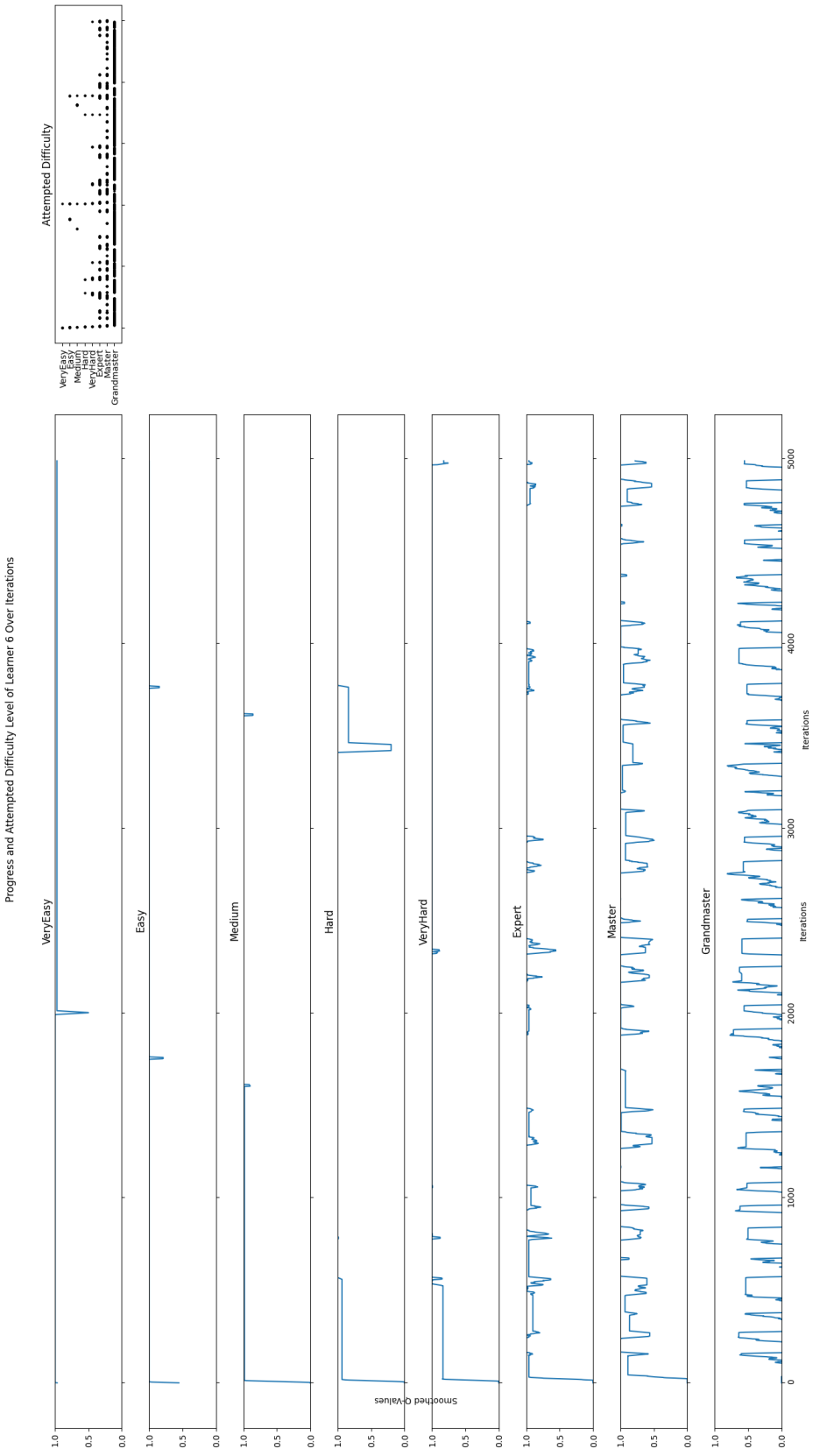

Figure 3.6: Learner 6's q table and attempted difficulties over 5000 iterations of Strategy 4.

Hence, strategy 4 provides a solution to the final missing piece of strategy 3 where trait sensitivity was not observable. So, evaluating strategy 3 based on the goals, the following is understood:

1. Goal 1 - Yes, strictly speaking, the learner does make progress
2. Goal 2 - Yes, the learner progresses only after mastering a difficulty level
3. Goal 3 - Yes, the learner revisits and reinforces previously mastered knowledge
4. Goal 4 - Yes, the algorithm is sensitive to the learners' traits and how well it aligns with a question

3.2.5 Final Analysis and Remarks

Complexity Analysis and Runtime Statistics

To complete the analysis of how NeuroNudge performed across strategies, the time and space complexity will now be explored.

By definition a Q-Learning algorithm iterates over all the states and actions to update the Q-values. In each iteration, the algorithm updates the Q-value for a single state-action pair, which is an $O(1)$ operation. However, since it needs to be done for every possible action in every state, the time complexity for a single update across the entire Q-table is $O(S \cdot A)$. If the algorithm runs for I iterations, the overall time complexity becomes $O(I \cdot S \cdot A)$. Hence, for strategy 1, the algorithm has a time complexity of $O(I \cdot S \cdot A)$ where $N = 5000$ (the simulation testing ran 5000 iterations for each strategy).

To be more specific - the total number of states S is the number of lessons in a particular content module, which is the same as the number of difficulties D in the NeuroNudge content model (at this stage, it was 8). For strategy 1, there's only 2 actions which can occur of staying at the current state s or ascending to the next state s'. More generally, the number of actions that can be taken in the context of NeuroNudge even if it is scaled up, is always a fraction of the total lessons and difficulties which exist -

because the algorithm is not designed to recommend a lesson of random difficulty that may be multiple levels higher or lower in difficulty than the current lesson.

In fact, at best, the algorithm allows the recommendation of 3 different actions: going up a difficulty, going down a difficulty, and staying at the current difficulty. This means that A is maximised by 3 and S is equivalent to the number of difficulties D. Hence, for all strategies if we consider just the update rule and basic Q-Learning behaviour, the time complexity is linear with respect to the number of difficulties, $O(c \cdot D) \approx O(D)$ where $c = A \cdot I : 1 \leq A \leq 3$ and $I = 5000$.

As for space complexity, it is simply $O(D)$ as it requires storing a Q-value for each lesson and difficulty level pair - and this was clarified in the prior definitions section to be in a 1:1 relationship for the current NeuroNudge content model, meaning there are exactly D pairs.

The minor changes to time and space complexity which may require consideration is the layers that are added through each strategy:

- **Strategy 2**: The significant addition is a mastery level calculation in reward calculation, however this is a constant time overhead as it's effectively a large conditional for the computed reward. Hence, the time complexity remains as $O(D)$. No modifications are made to the memory being stored beyond a small addition of local state such as the mastery thresholds, so the space complexity is unchanged from $O(D)$.
- **Strategy 3**: A few significant additions occur. The first is decaying at most $D - 1$ difficulties within a single iteration (because we do not decay the difficulty that is being attempted right now, by definition of the decay mechanism clarified earlier in strategy 3). This means that an additional $D - 1$ operations occur for every update D, effectively making the time complexity quadratic at $O(D^2)$.

  Furthermore, the space complexity changes because the decay mechanism requires an update of $D - 1$ non-attempt counter values, but this is simply an

addition of linear overhead and maintains the space complexity at $O(D)$. As the constant c multiplied to D (that is, $O(c \cdot D)$) is still a small factor, this will not be prone to explosion in real application (which we see in the table shown below of the run times for each strategy across 5000 iterations).

- **Strategy 4**: A few significant additions occur. For each update of some D that was attempted in a single iteration, a trait alignment is now calculated. Trait alignment at this stage operates in linear time $O(T)$ of the number of traits T with a direct comparison of each value. Because this occurs at every update in the attempt of some level D, the time complexity changes to $O(D^2 \cdot T)$. As T is not expected to be a large input size at any stage (that is, the number of traits defined in the system should not explode to large numbers), this is effectively another linear factor that amplifies the time complexity.

  As for the space complexity, the counter of consecutive attempts for each difficulty level D is added to this strategy (as per its definition in the prior section). This will simply have the same effect as above in adding to the space complexity with linear overhead $O(D)$ as each level D has a value for its consecutive attempts.

The following table provides a summary of the time and space complexity for each strategy:

Strategy	**Time Complexity**	**Space Complexity**
Strategy 1	$O(D)$	$O(D)$
Strategy 2	$O(D)$	$O(D)$
Strategy 3	$O(D^2)$	$O(D)$
Strategy 4	$O(D^2 \cdot T)$	$O(D)$

Table 3.3: Summary of time and space complexity for each strategy.

On top of this complexity analysis for each strategy, the following table provides averaged figures of observed runtime for each strategy:

What can be concluded from a cross-examination from the two complexity tables? Strategy by strategy, this is what can be reasoned:

Iterations	Strategy 1	Strategy 2	Strategy 3	Strategy 4
1,000	1029.0 ms	938.8 ms	1167.2 ms	1171.2 ms
5,000	5223.2 ms	4815.0 ms	5905.4 ms	5944.4 ms
10,000	10463.2 ms	9650.4 ms	11812.6 ms	11930.4 ms
20,000	20983.8 ms	19276.4 ms	23712.0 ms	24040.4 ms

Table 3.4: Average runtimes for different strategies across varying iterations on an M1 Pro with Rust's optimised builds.

- **Strategy 1**: The runtime increases linearly with the number of iterations, aligning with the theoretical time complexity of $O(D)$. The linear relationship reflects the direct mapping between lessons and their difficulties without additional computations.
- **Strategy 2**: Similar to Strategy 1, this strategy also shows a linear increase in runtime with iterations. The slight decrease in runtime compared to Strategy 1 that occurs across each tier of iterations can perhaps be attributed to the way its modification of exploration using stricter mastery thresholds rather than an $O(D)$ search for lesson with the next highest difficulty level is quicker on average.
- **Strategy 3**: The runtime increase is more pronounced in Strategy 3, reflecting the quadratic nature of its time complexity $O(D^2)$. The decay mechanism, which updates $D - 1$ difficulties each iteration, adds to the computational load, explaining the higher runtimes compared to Strategies 1 and 2. However, it is not notably larger than the other runtimes and suggests that the algorithm has layered on revisitation of previously mastered content via a decay mechanism in a relatively efficient manner.
- **Strategy 4**: This strategy shows the highest observed runtimes, indicating its more complex nature. The time complexity is $O(D^2 \cdot T)$, where T represents the number of traits. The alignment calculation for ASD traits, done for each lesson update, adds significant overhead. While T is not a large number, its multiplicative effect on the complexity is evident in the increased runtimes. However, it is not notably larger than the other runtimes and suggests that the algorithm has layered on trait sensitivity in a relatively efficient manner.

An interesting statistic is that therefore, running one lesson in each strategy takes

approximately 1.05 ms for Strategy 1, 0.97 ms for Strategy 2, 1.19 ms for Strategy 3, and 1.20 ms for Strategy 4 on an M1 Pro processor (using the 20,000 iteration metrics).

Assessment of Q-Learning Effectiveness

To assess the effectiveness of NeuroNudge with respect to Q-Learning and q tables, the typical points which are examined are as follows:

- **Convergence:** Here it is common to evaluate how effectively the Q-values in the tables converge to stable values. In NeuroNudge, convergence is observed for easier levels, indicating effective learning and decision-making for simpler tasks. However, for scenarios where learner traits are not aligned with the question's requirements, convergence is less straightforward. The fluctuations in Q-values may be attributed to the random probabilities used in simulating attempts (but this wouldn't be a factor with real users). Another reason is when a learner attempts lessons with incompatible ASD trait levels as they have a much lower chance of success. Yet, this lack of convergence is a good thing - what's much more important in this context is to accurately direct the learning path for a student based on their needs. That is, convergence is not so straight forward in an educational context and having the malleability of a relatively more randomised ascent to mastery is vital.
- **Balance Between Exploration and Exploitation:** An important aspect of Q-Learning is balancing the exploration of new states (or lessons in this context) and the exploitation of known rewarding states. NeuroNudge's implementation of strategies like decay counters and mastery thresholds has successfully addressed this balance. The system explores more when the learner's progress in certain areas is weak or when they haven't been attempted for a while (Strategy 3), and in general if it chooses to exploit then it will only direct the learner to their best current difficulty level - effectively giving a learner a reset back to their best content before moving forward again.
- **Responsiveness to Actor of the System:** Particularly relevant in Strategy 4,

where ASD traits are taken into account, the Q-table updates reflect the learner's progress not just based on lesson difficulty but also on their ASD trait alignment with the lesson content. This personalized approach ensures that the learning process is tailored to the individual needs of the learner.

- **Reinforcement of Learning:** The effectiveness of NeuroNudge is also evident in how it reinforces learning with decaying and revisitation. Through strategies like decaying Q-values, learners are encouraged to revisit content that they haven't attempted for a while, ensuring that the knowledge is reinforced and retained.
- **Adaptability and Flexibility:** The biggest point for NeuroNudge is that while it does adapt flexibly to different learning profiles of learners, it could easily be modified to map to traits of any learner at large (instead of just those impacted by ASD). This can simply be done by defining more learner traits.

Final Remarks

The conclusion from these results is that the combination of all 4 strategies provides a version of NeuroNudge that has great potential as a learning algorithm for learners with ASD. With this algorithm and its recommendation of lessons, the learner makes progress and indeed does so with mastery of content, and they also revisit previously mastered content that hasn't been attempted in a while. Furthermore, with some tweaks to the simulated environment it has been proven that the algorithm should be sensitive to the unique ASD traits of learners.

Overall, this is a very positive result for NeuroNudge and cements its potential as an algorithm that can change the space of special education and learning.

Chapter 4

Conclusion

To summarise, NeuroNudge is a promising unsupervised learning algorithm for learners with ASD. By adopting Q-Learning with 4 strategies that adjust its ability to encapsulate the unique needs of such learners, it has shown great potential in creating an environment that dynamically tailors learning to each individual. This can provide great help in reducing stress and also costs for caretakers and clinical therapists in the space of special education and care for individuals with ASD in general.

To grow, NeuroNudge requires further rigorous testing and should be assessed with real learners. Furthermore, with a wider expanse of ASD traits and unique learning profiles from learners who have ASD, the algorithm may benefit by moving towards a Deep Q-Learning approach - so that a trainable model can slowly form around the parameters that are best suited to making appropriate learning recommendations. Other adjustments may include making the content design and system more rich and audited by experts such as clinical therapists.

Ultimately, the future may be bright for NeuroNudge as a solution that can aid many in the space of learning with ASD.

Bibliography

[1] Matthew J. Maenner, Zachary Warren, Ashley Robinson Williams, Esther Amoakohene, Amanda V. Bakian, Deborah A. Bilder, Maureen S. Durkin, Robert T. Fitzgerald, Sarah M. Furnier, Michelle M. Hughes, Christine M. Ladd-Acosta, Dedria McArthur, Elise T. Pas, Angelica Salinas, Alison Vehorn, Susan Williams, Amy Esler, Andrea Grzybowski, Jennifer Hall-Lande, Ruby H.N. Nguyen, Karen Pierce, Walter Zahorodny, Allison Hudson, Libby Hallas, Kristen Clancy Mancilla, Mary Patrick, Josephine Shenouda, Kate Sidwell, Monica DiRienzo, Johanna Gutierrez, Margaret H. Spivey, Maya Lopez, Sydney Pettygrove, Yvette D. Schwenk, Anita Washington, and Kelly A. Shaw. Prevalence and characteristics of autism spectrum disorder among children aged 8 years — autism and developmental disabilities monitoring network, 11 sites, united states, 2020. *MMWR. Surveillance Summaries*, 72(2):1–14, March 2023.

[2] Hope for Healing. Parents and caregivers responsibilities for a child with autism. https://get2theroot.com/parents-and-caregivers-responsibilities-for-a-child-with-autism/.

[3] OpenAI. Chat openai. https://chat.openai.com/.

Appendix A: Extensive Results

The following contains all results for each strategy, for each learner. In strategy 4, results from both modules of *Shapes* and *Actions* are provided.

A.1 Results for Strategy 1

A.1.1 First Set: $\epsilon = 0.3$

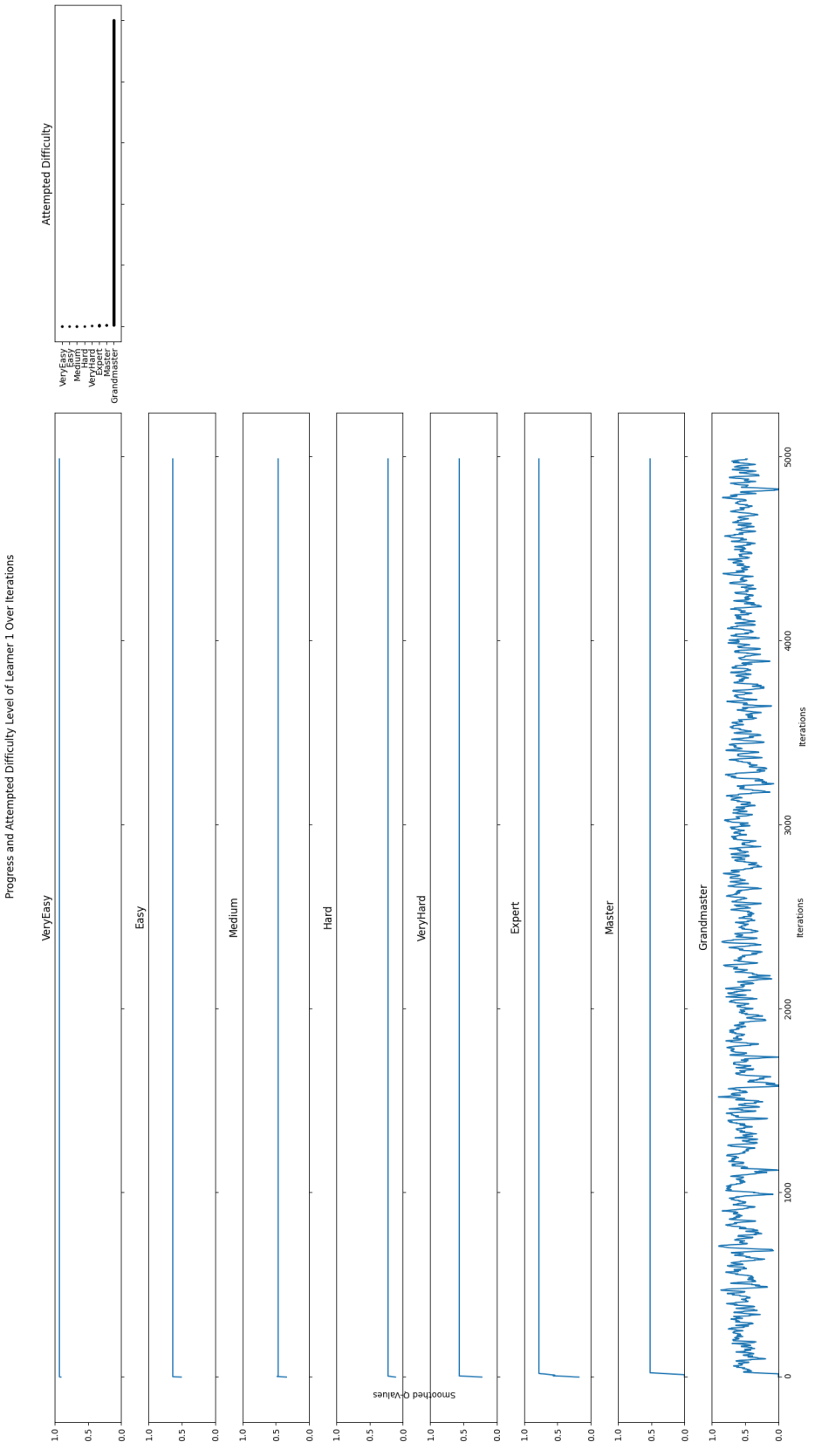

Figure A.1: Learner 1's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.3$

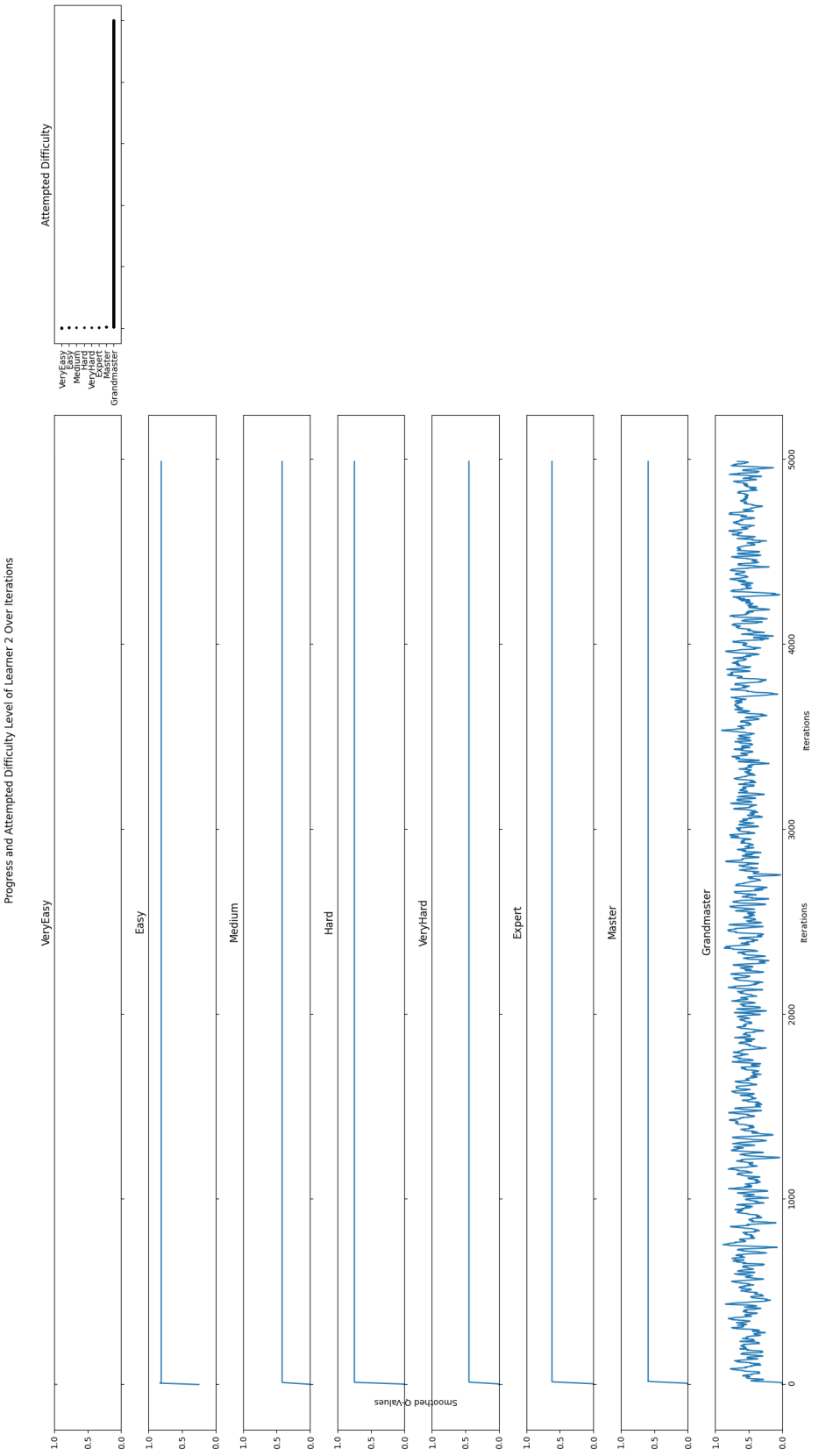

Figure A.2: Learner 2's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.3$

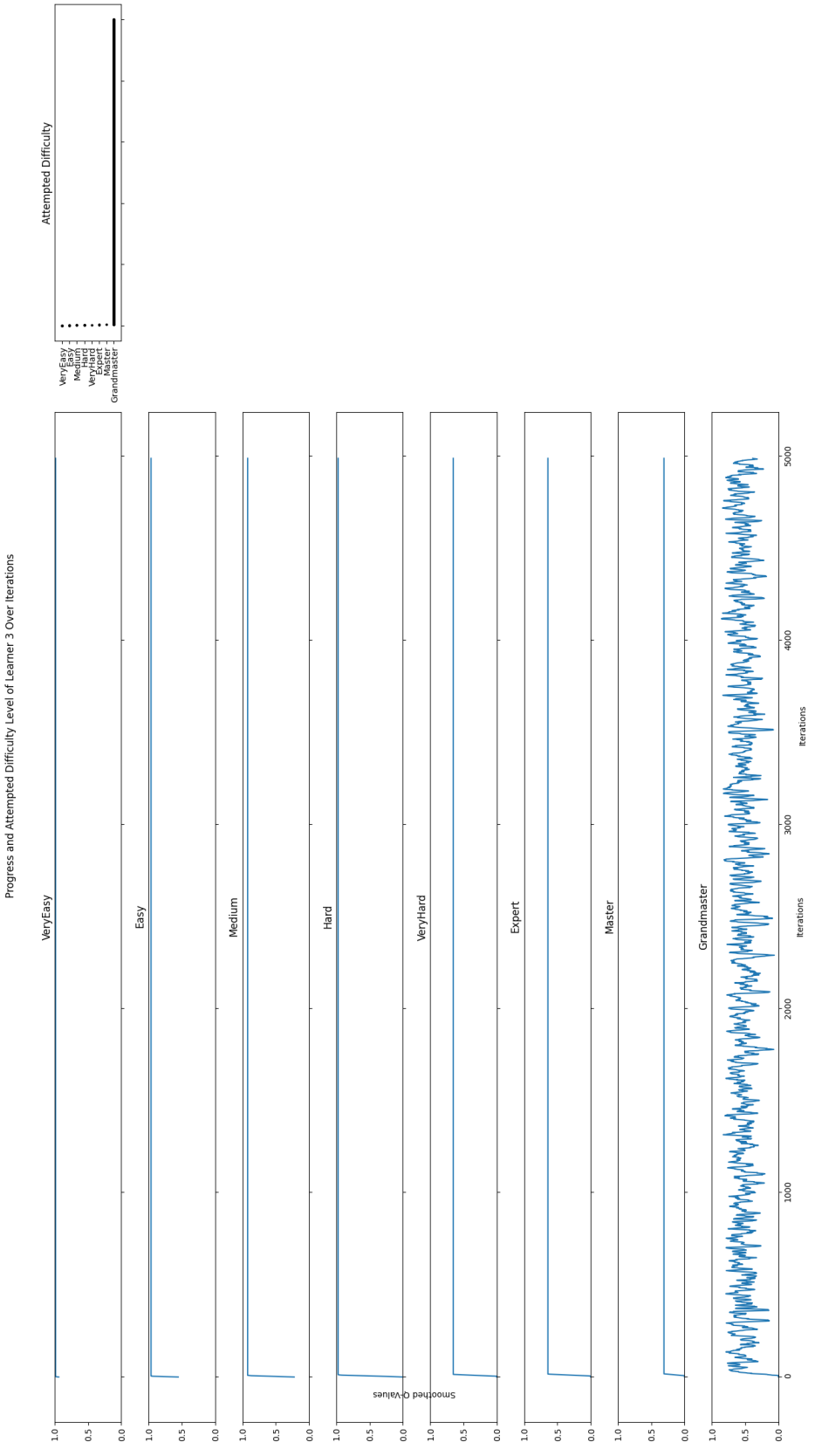

Figure A.3: Learner 3's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.3$

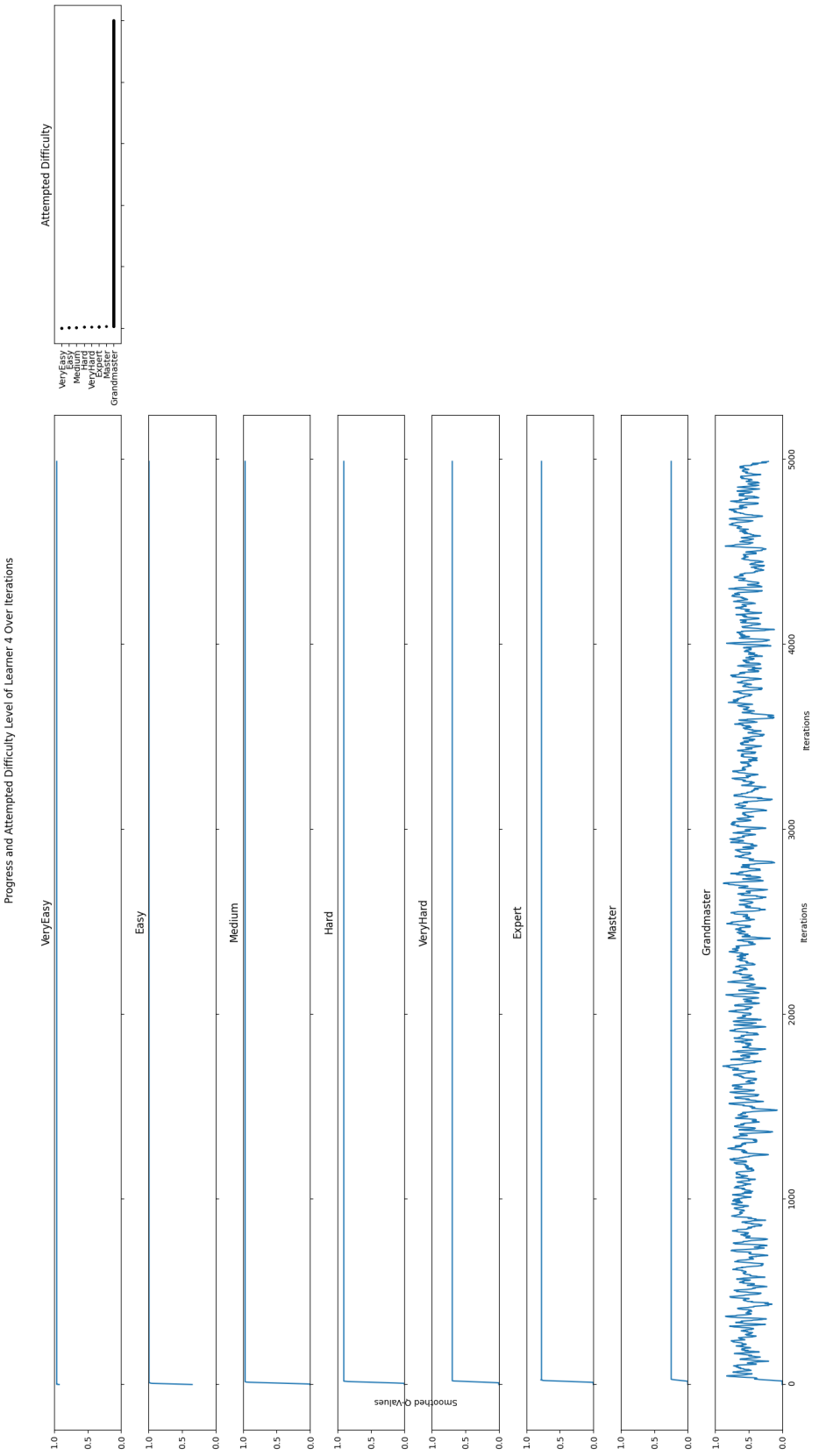

Figure A.4: Learner 4's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.3$

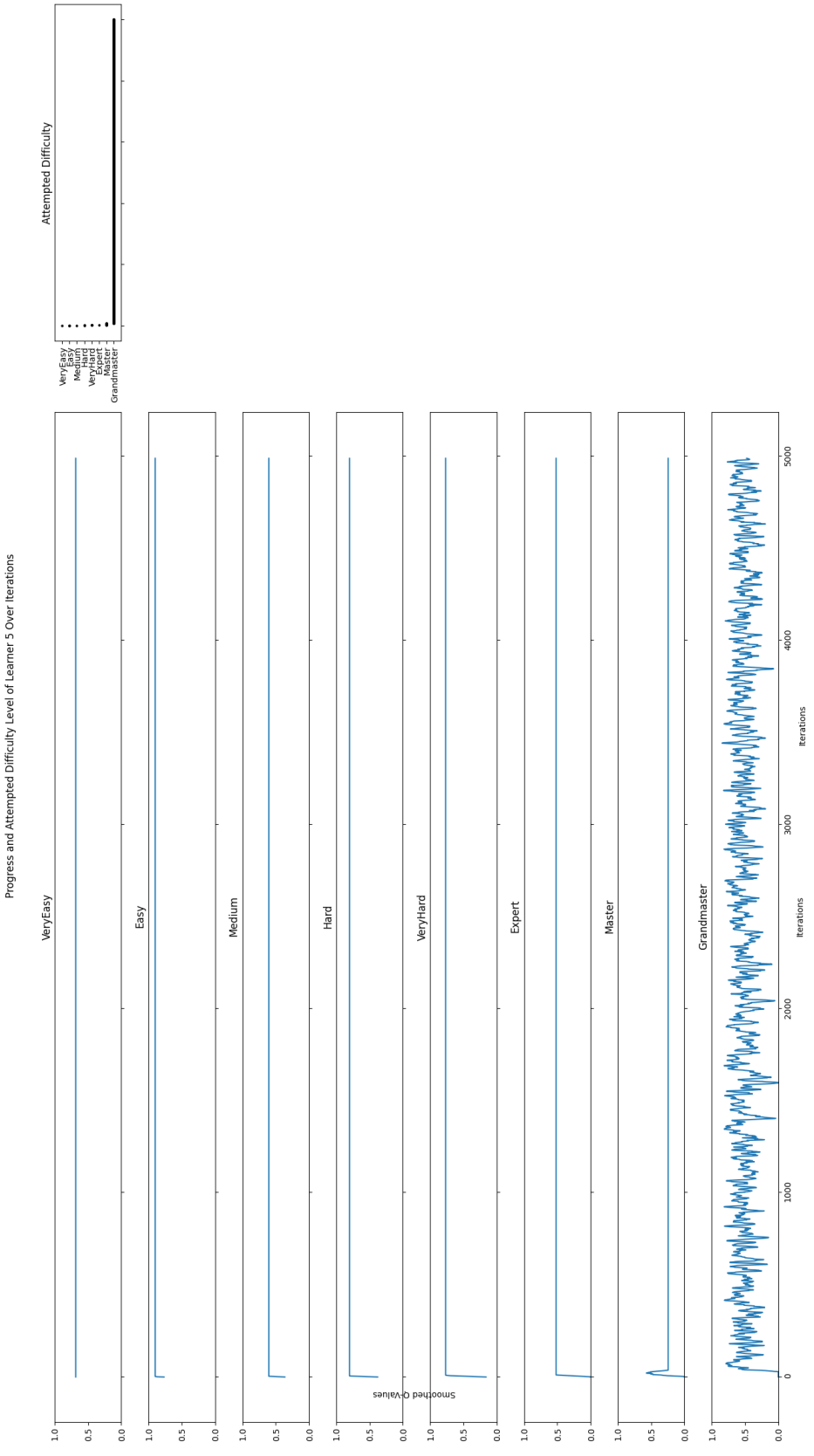

Figure A.5: Learner 5's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.3$

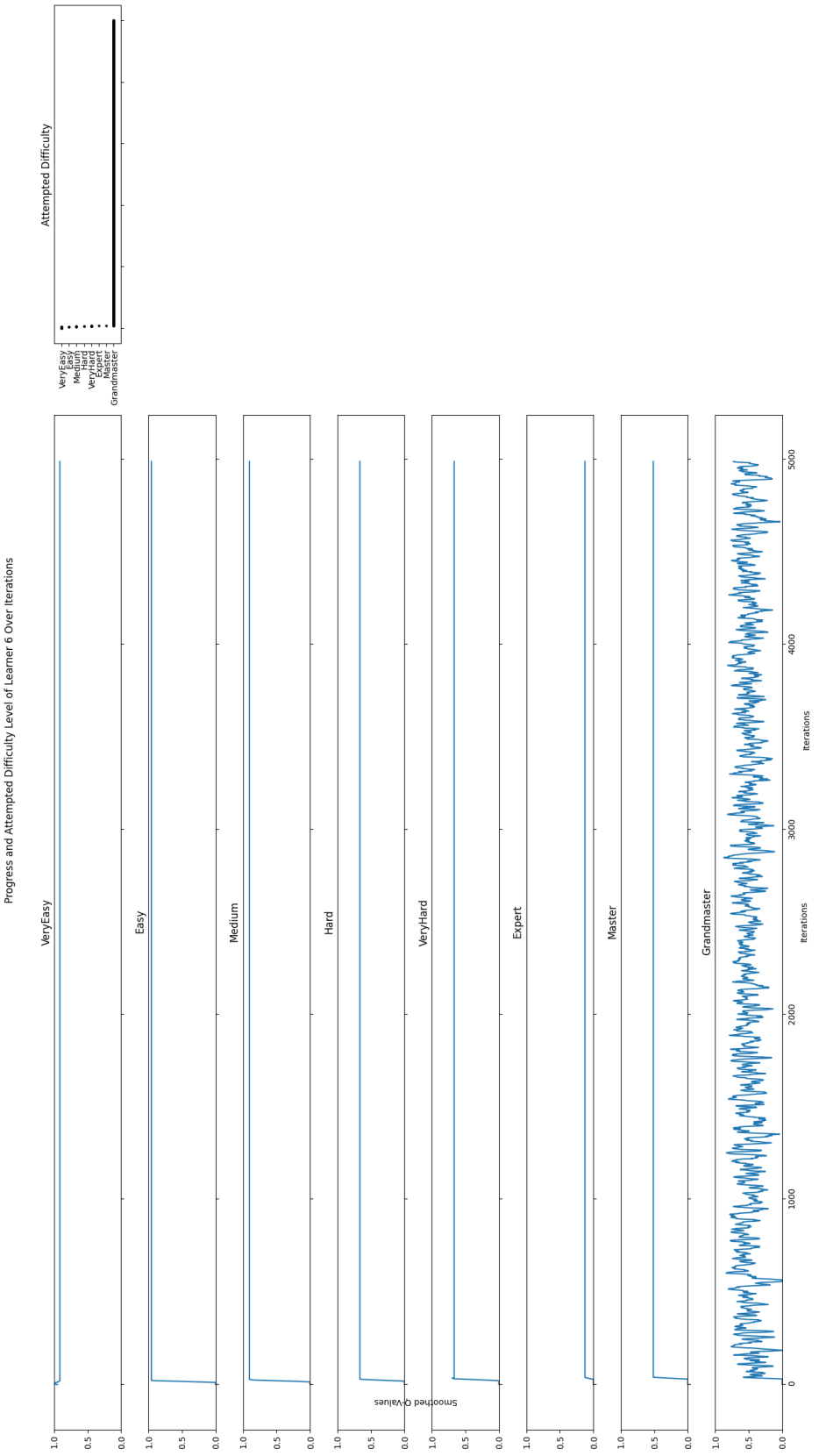

Figure A.6: Learner 6's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.3$

A.1.2 Second Set: $\epsilon = 0.6$

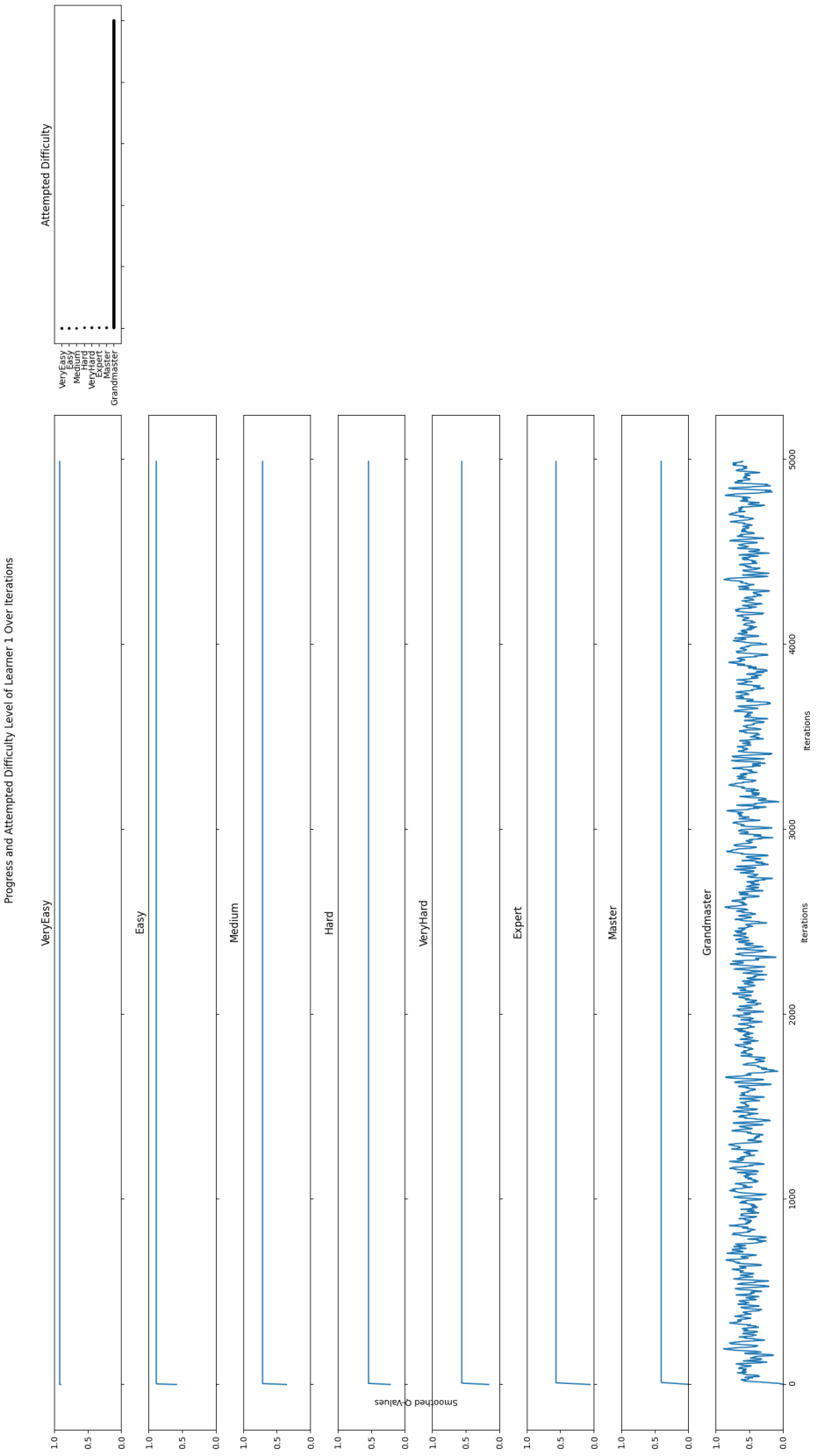

Figure A.7: Learner 1's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.6$

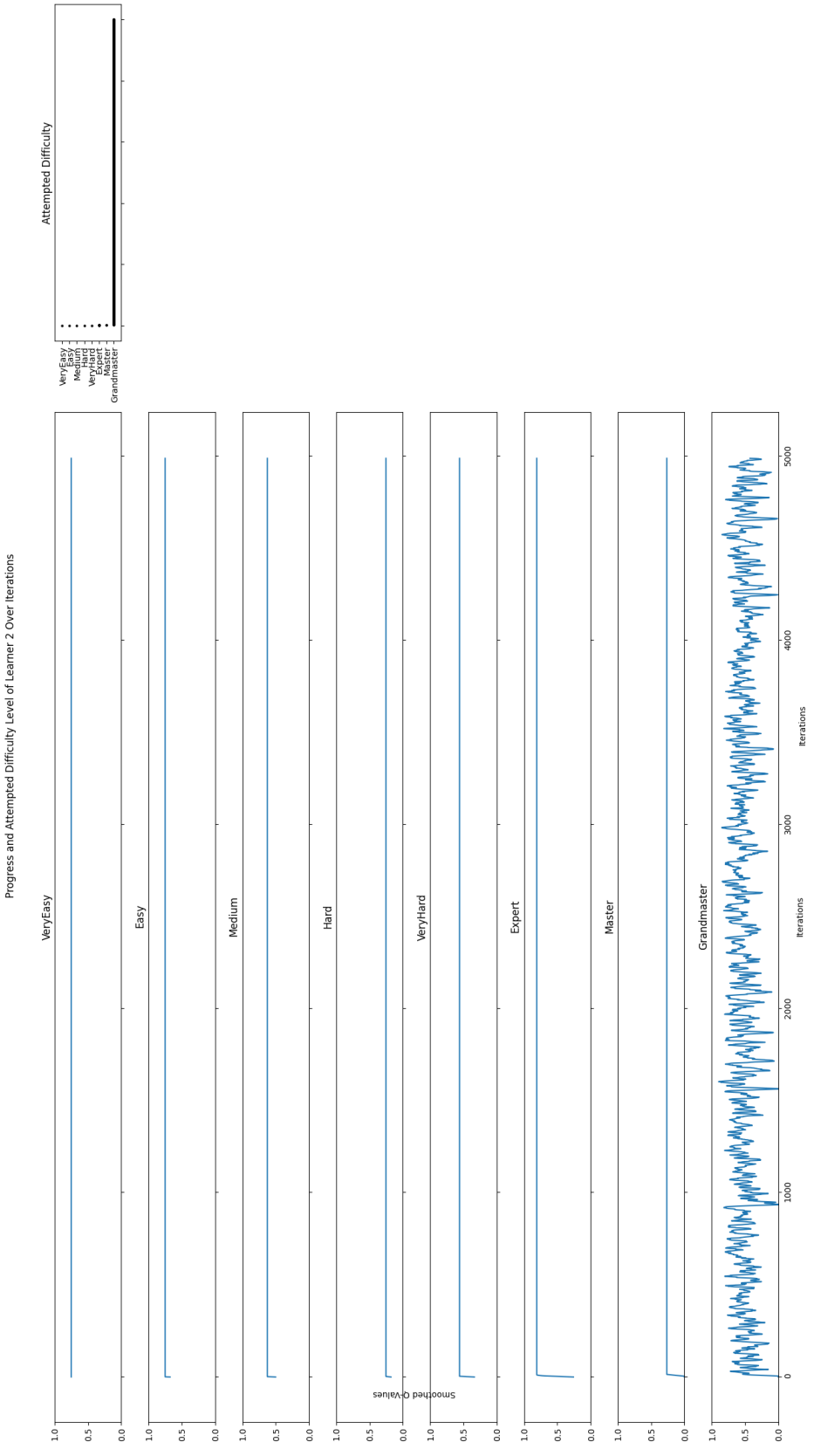

Figure A.8: Learner 2's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.6$

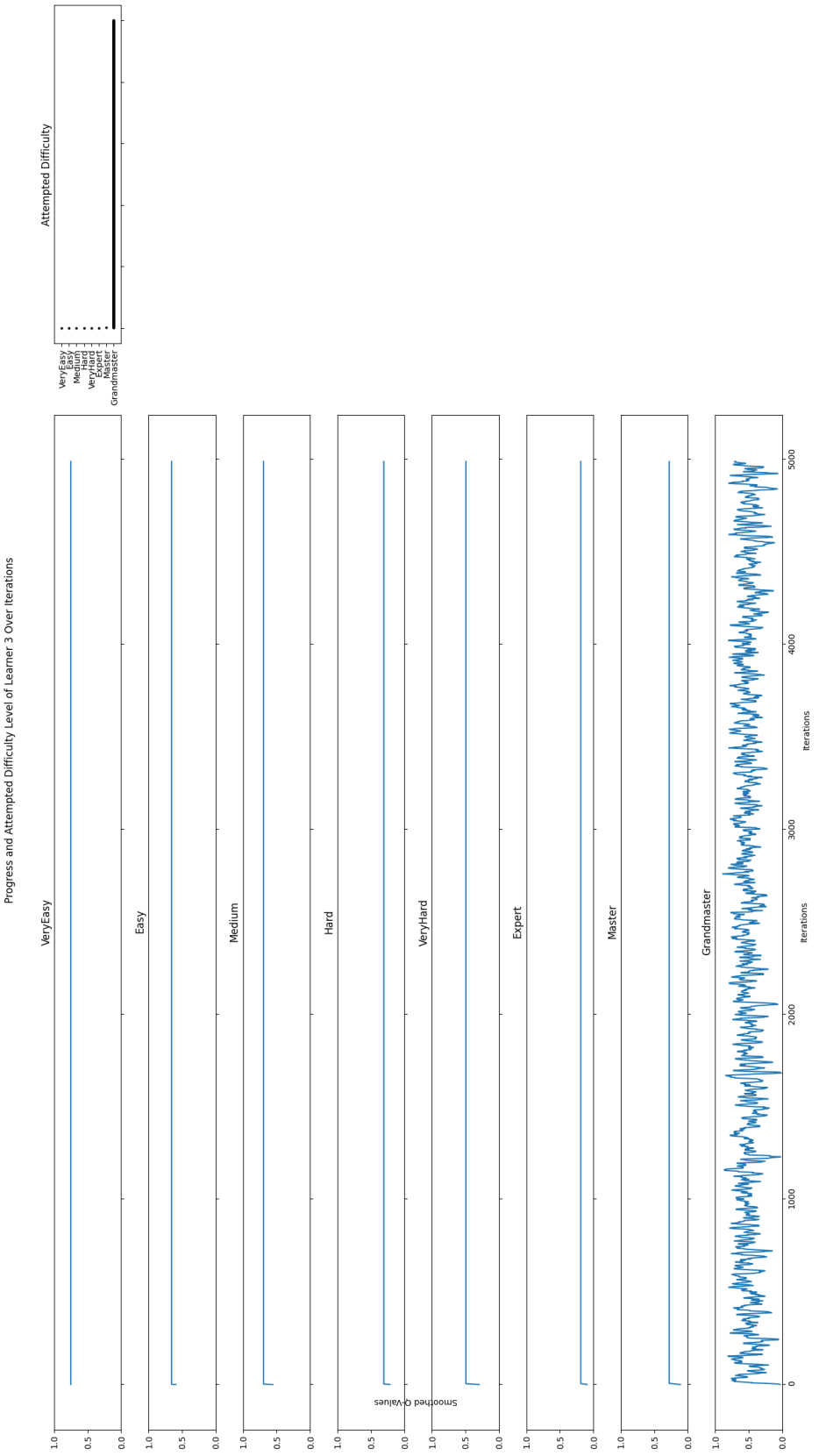

Figure A.9: Learner 3's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.6$

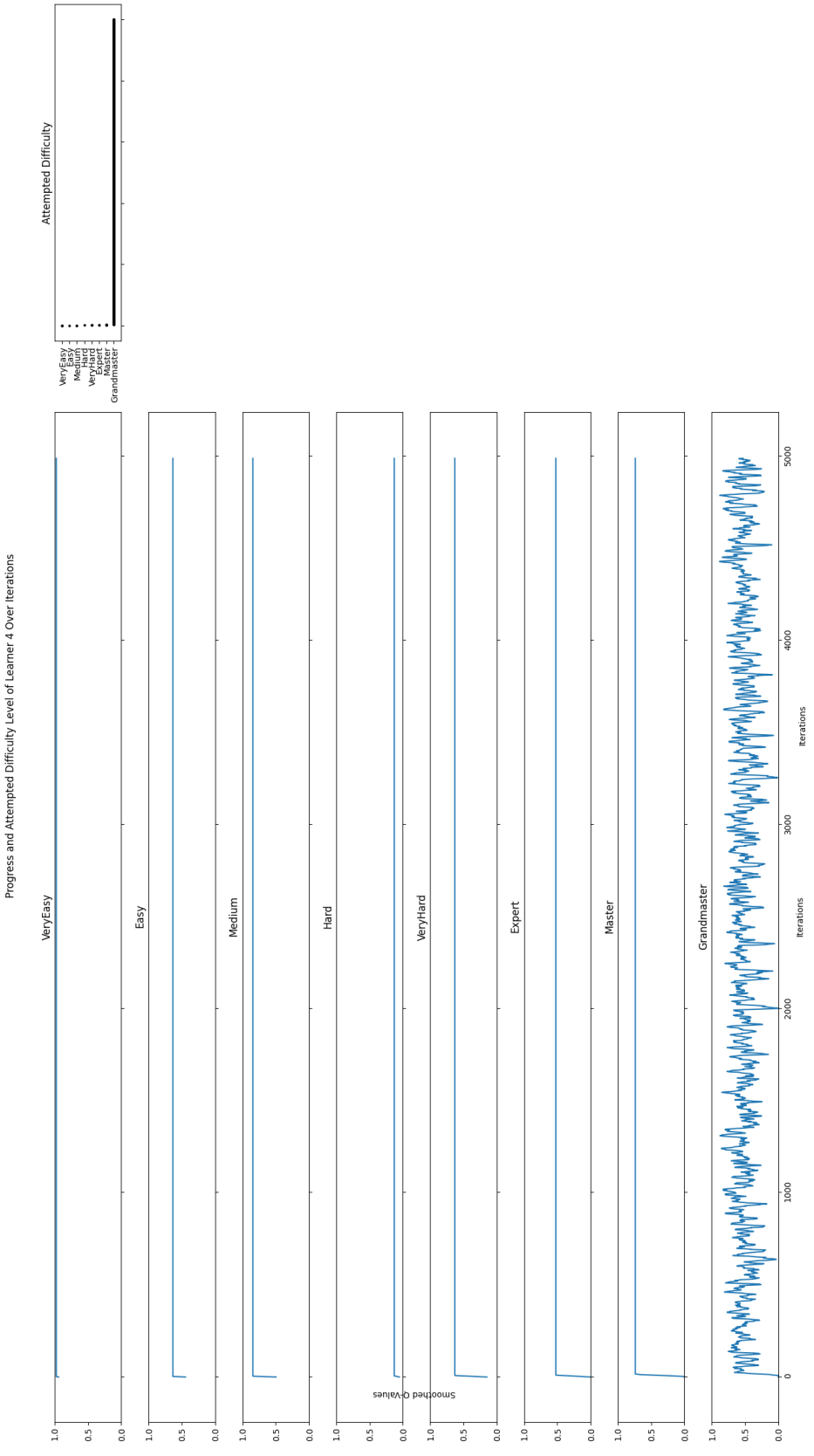

Figure A.10: Learner 4's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.6$

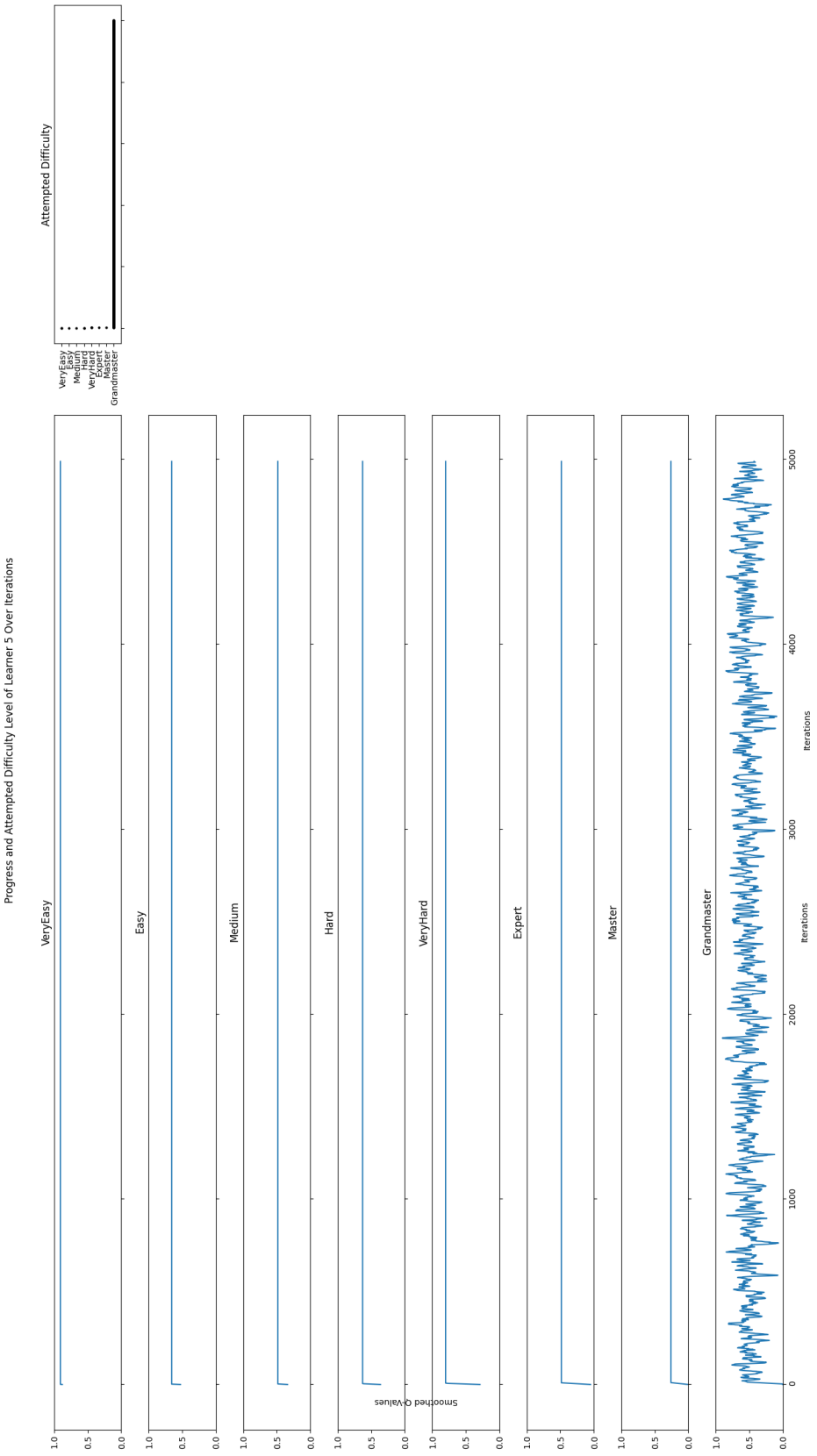

Figure A.11: Learner 5's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.6$

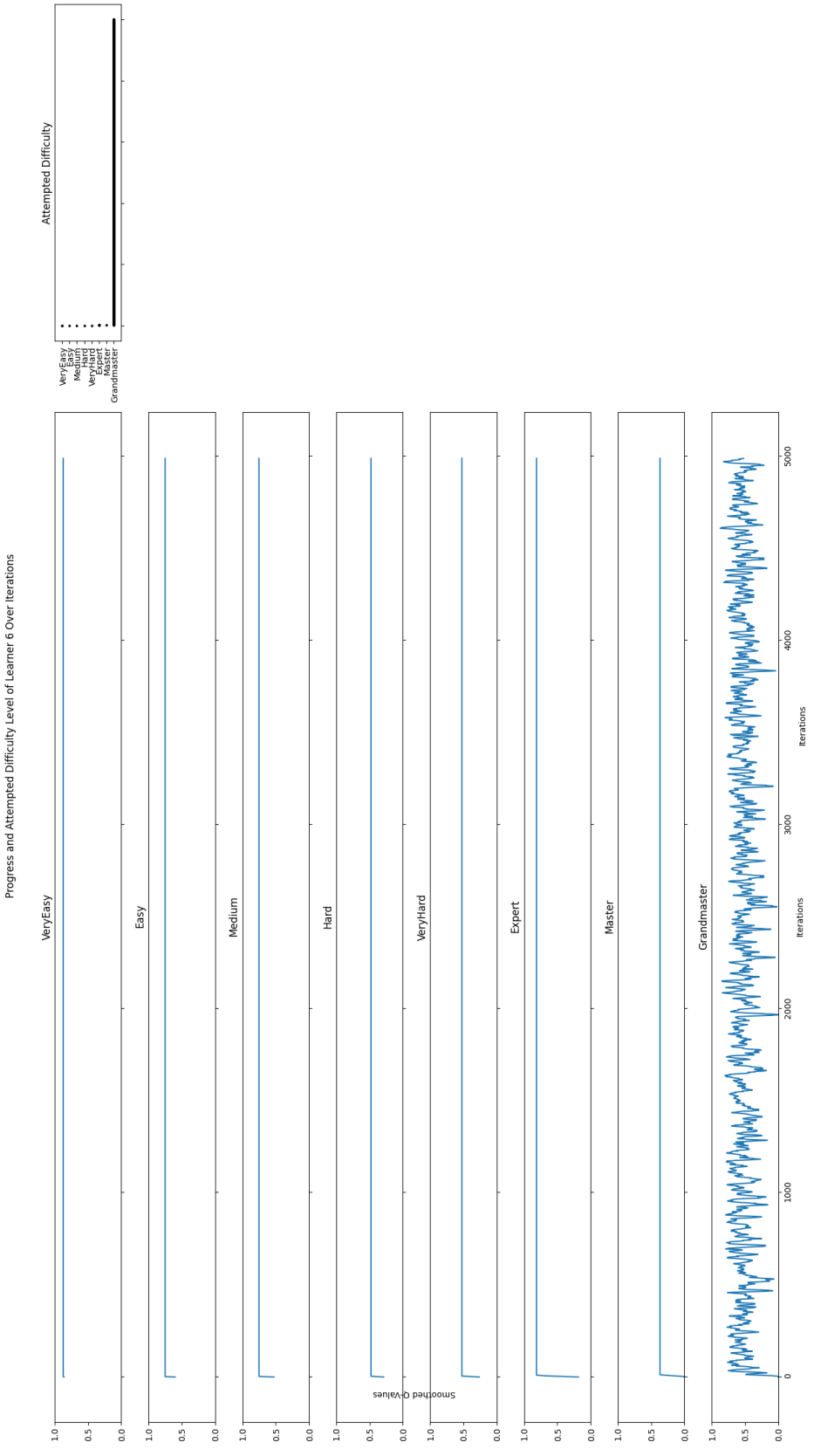

Figure A.12: Learner 6's q table and attempted difficulties over 5000 iterations of Strategy 1. $\epsilon = 0.6$

A.2 Results for Strategy 2

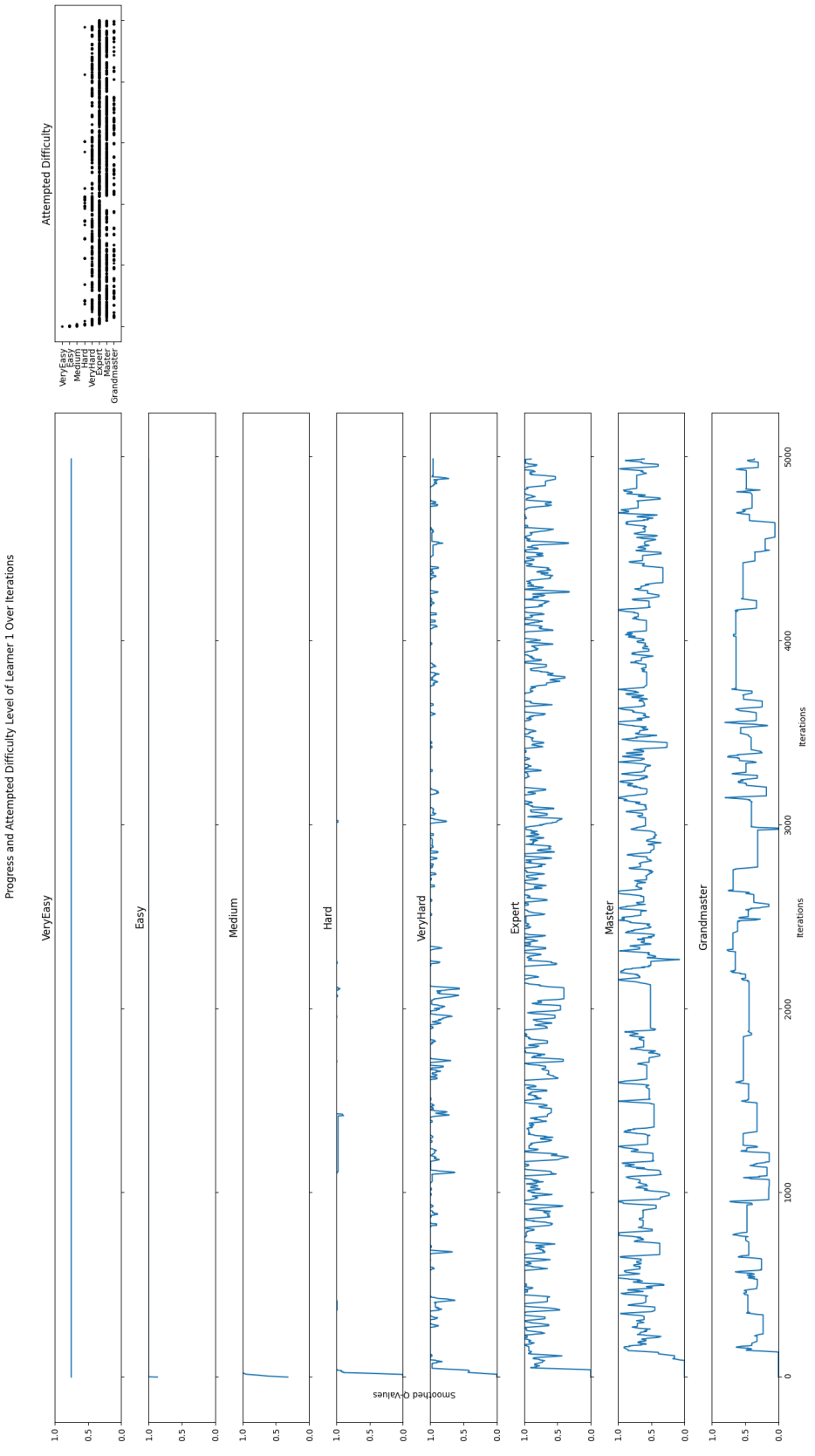

Figure A.13: Learner 1's q table and attempted difficulties over 5000 iterations of Strategy 2.

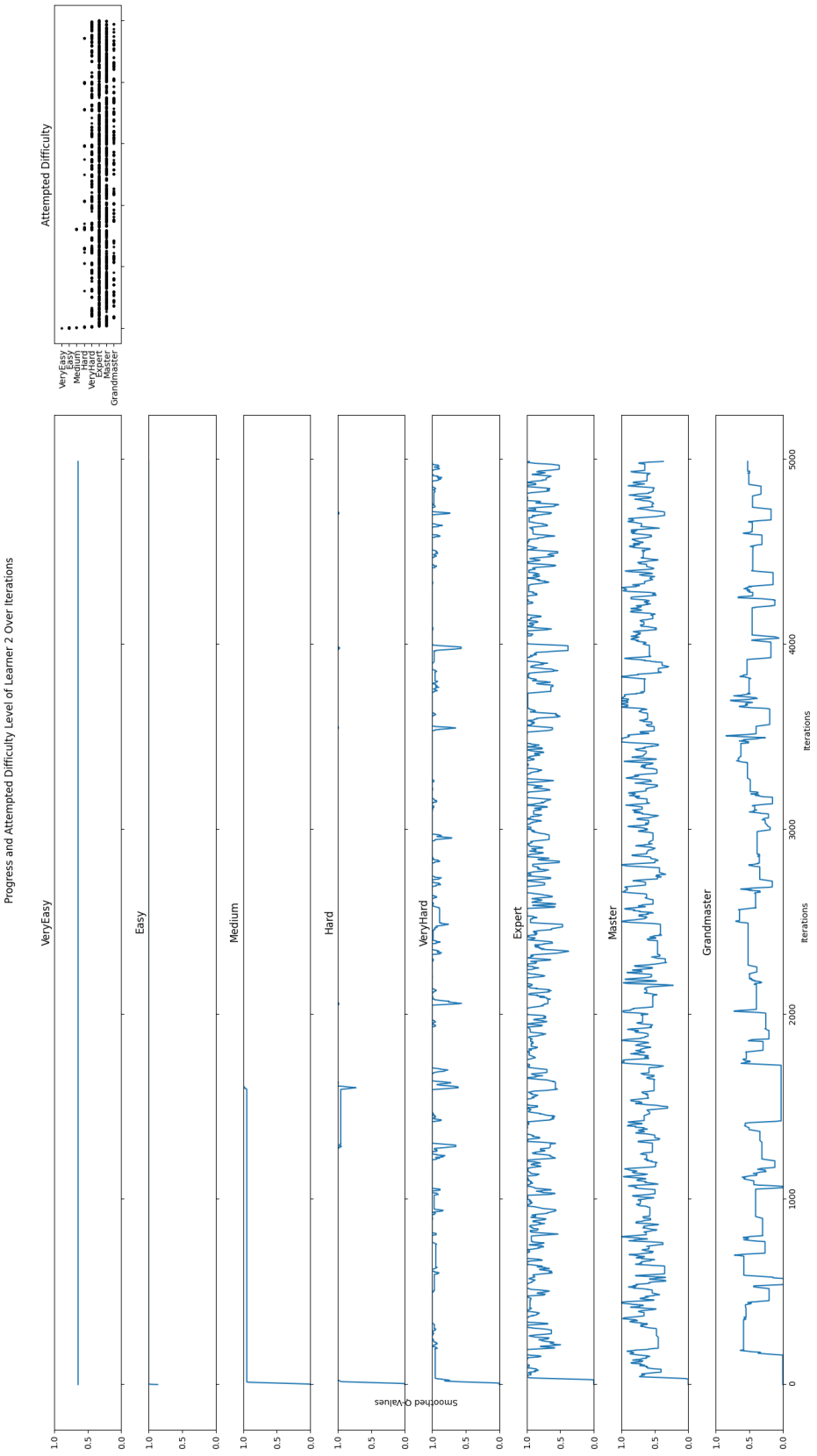

Figure A.14: Learner 2's q table and attempted difficulties over 5000 iterations of Strategy 2.

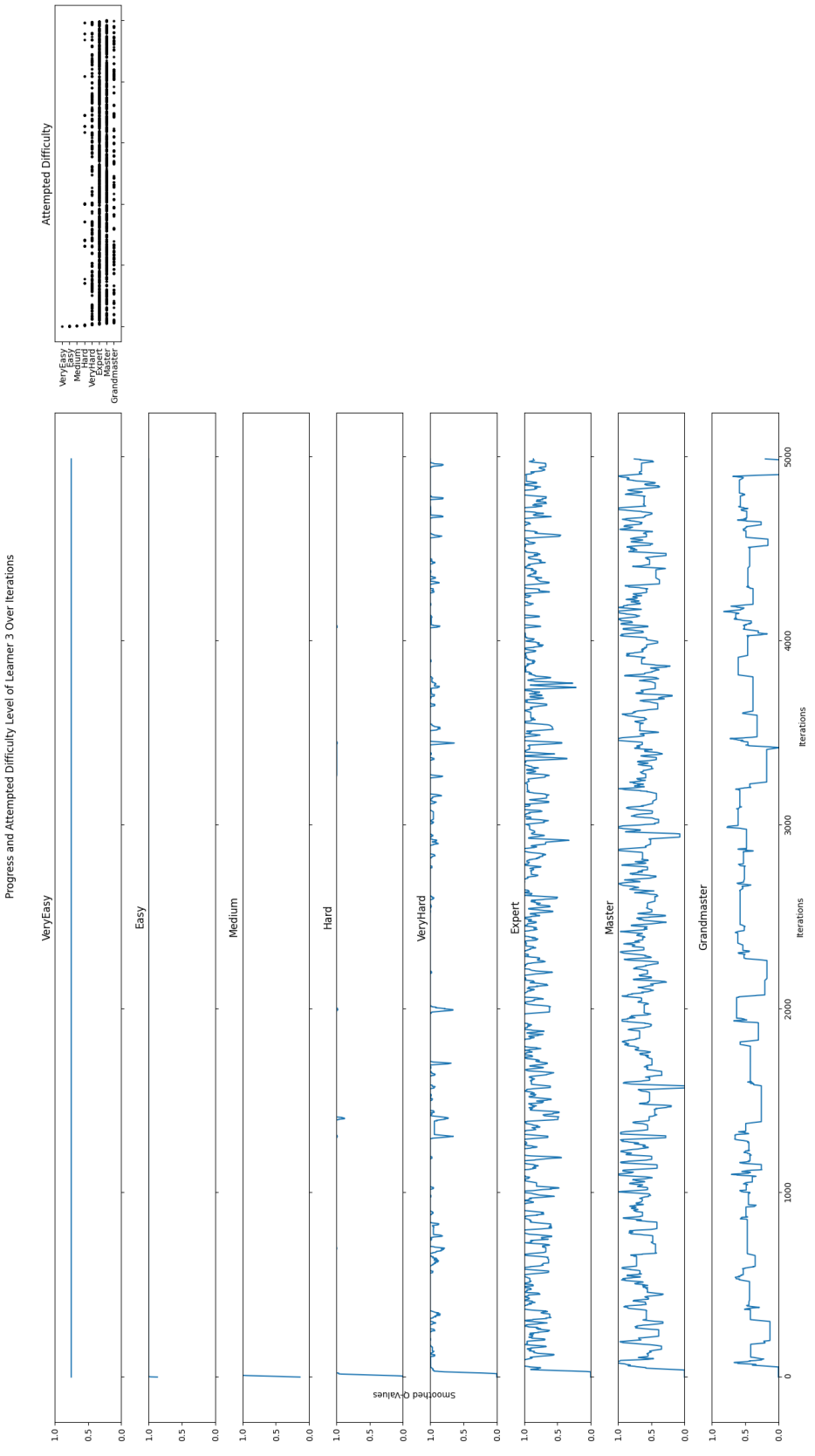

Figure A.15: Learner 3's q table and attempted difficulties over 5000 iterations of Strategy 2.

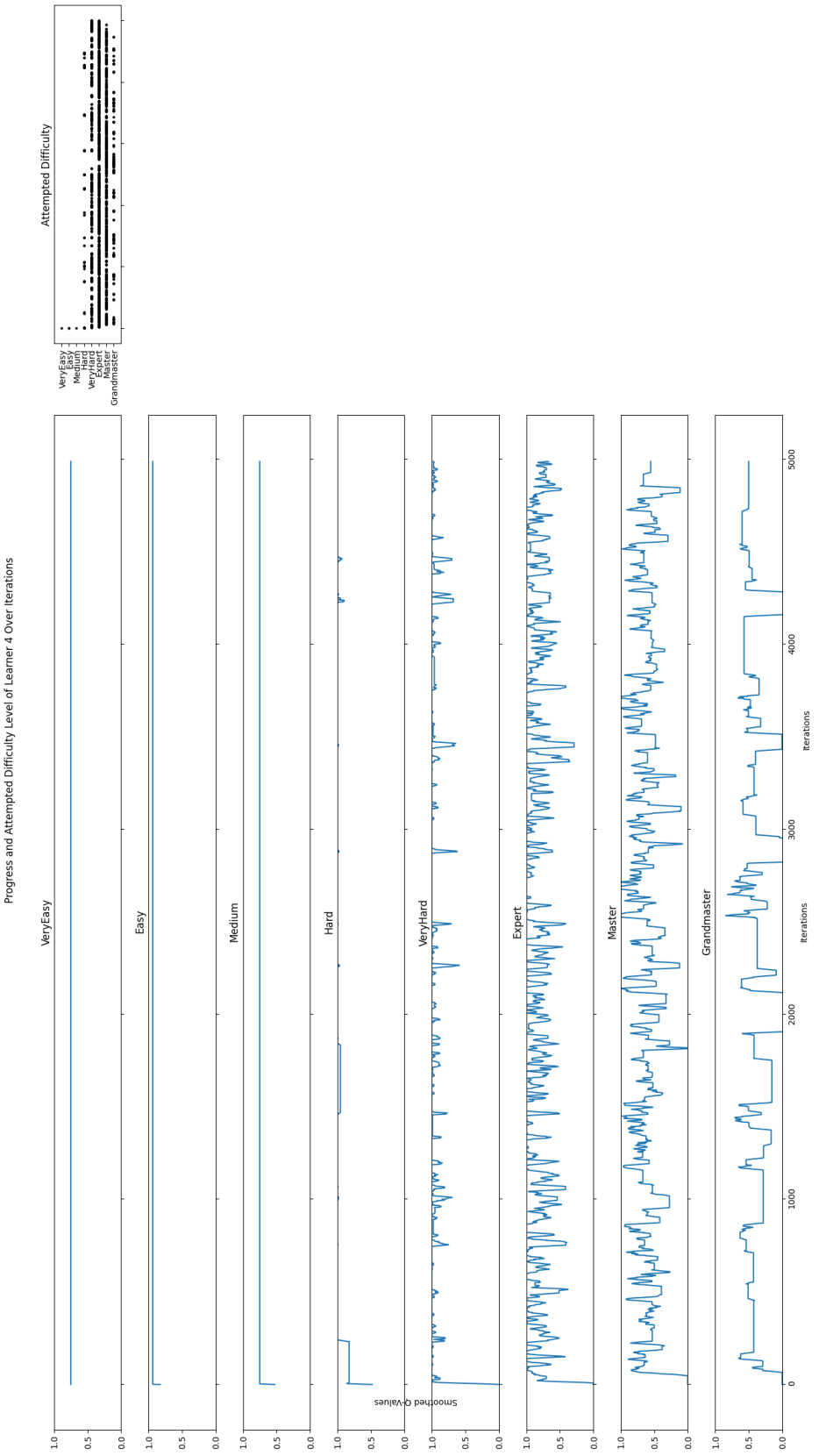

Figure A.16: Learner 4's q table and attempted difficulties over 5000 iterations of Strategy 2.

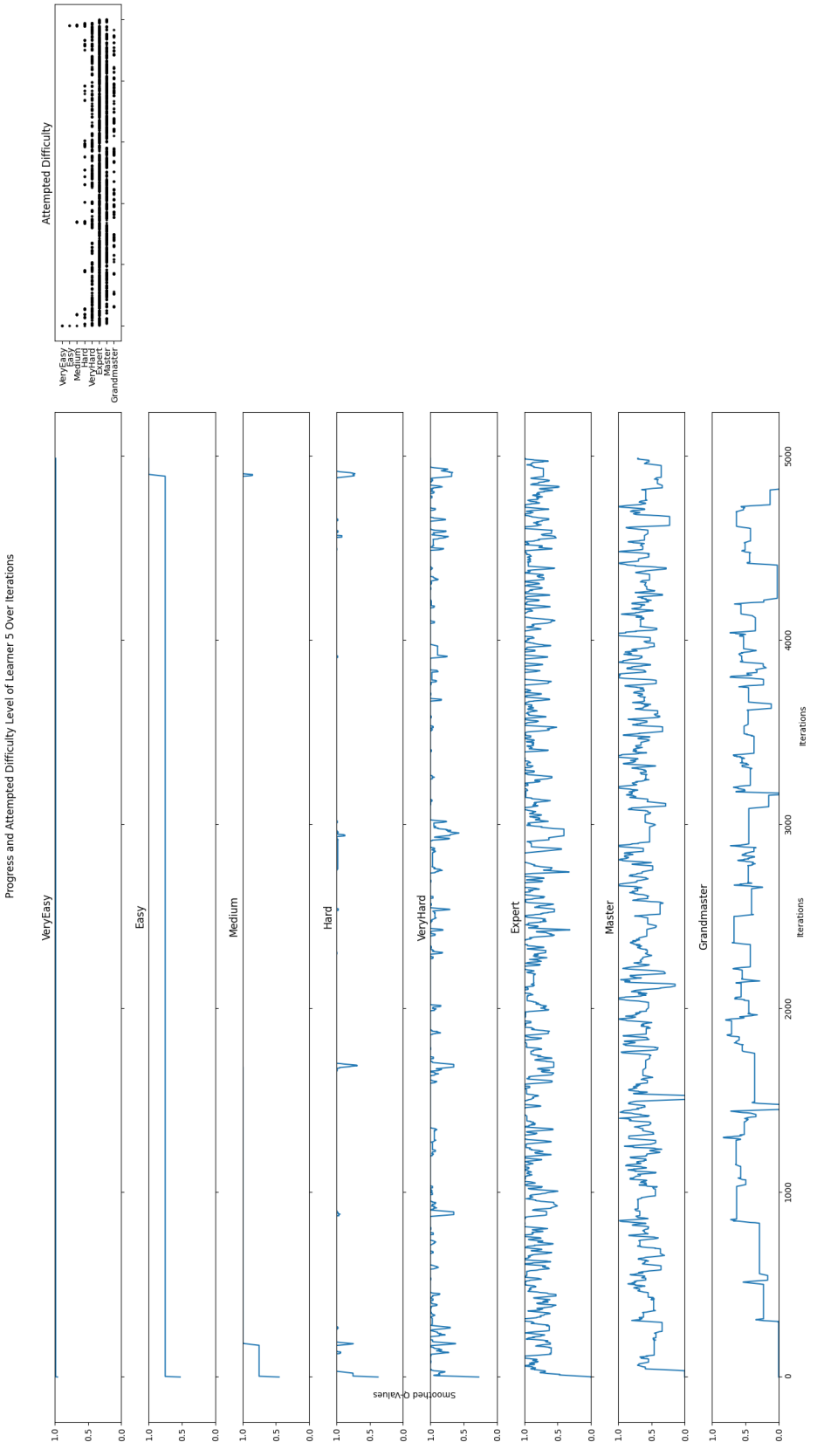

Figure A.17: Learner 5's q table and attempted difficulties over 5000 iterations of Strategy 2.

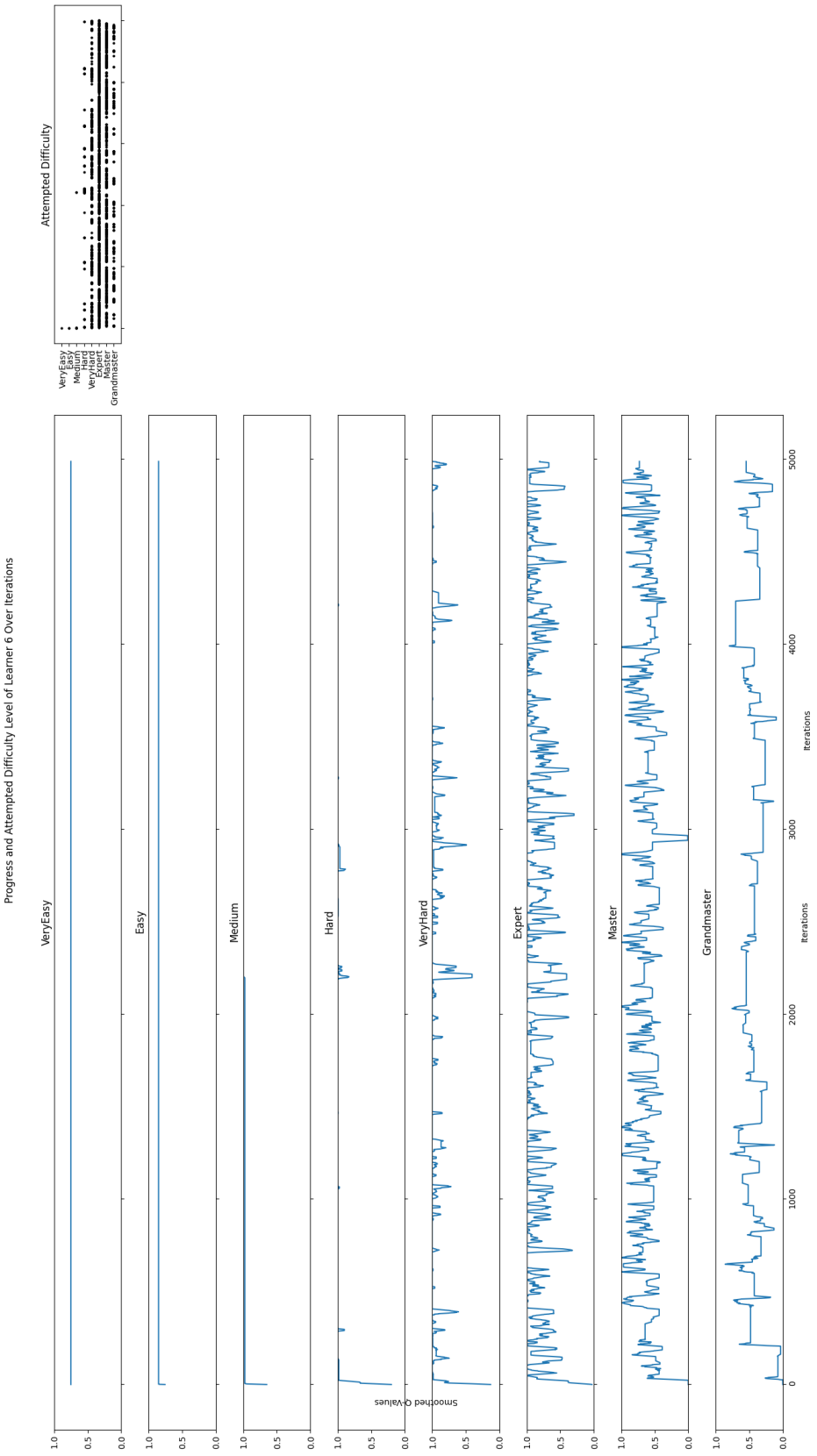

Figure A.18: Learner 6's q table and attempted difficulties over 5000 iterations of Strategy 2.

A.3 Results for Strategy 3

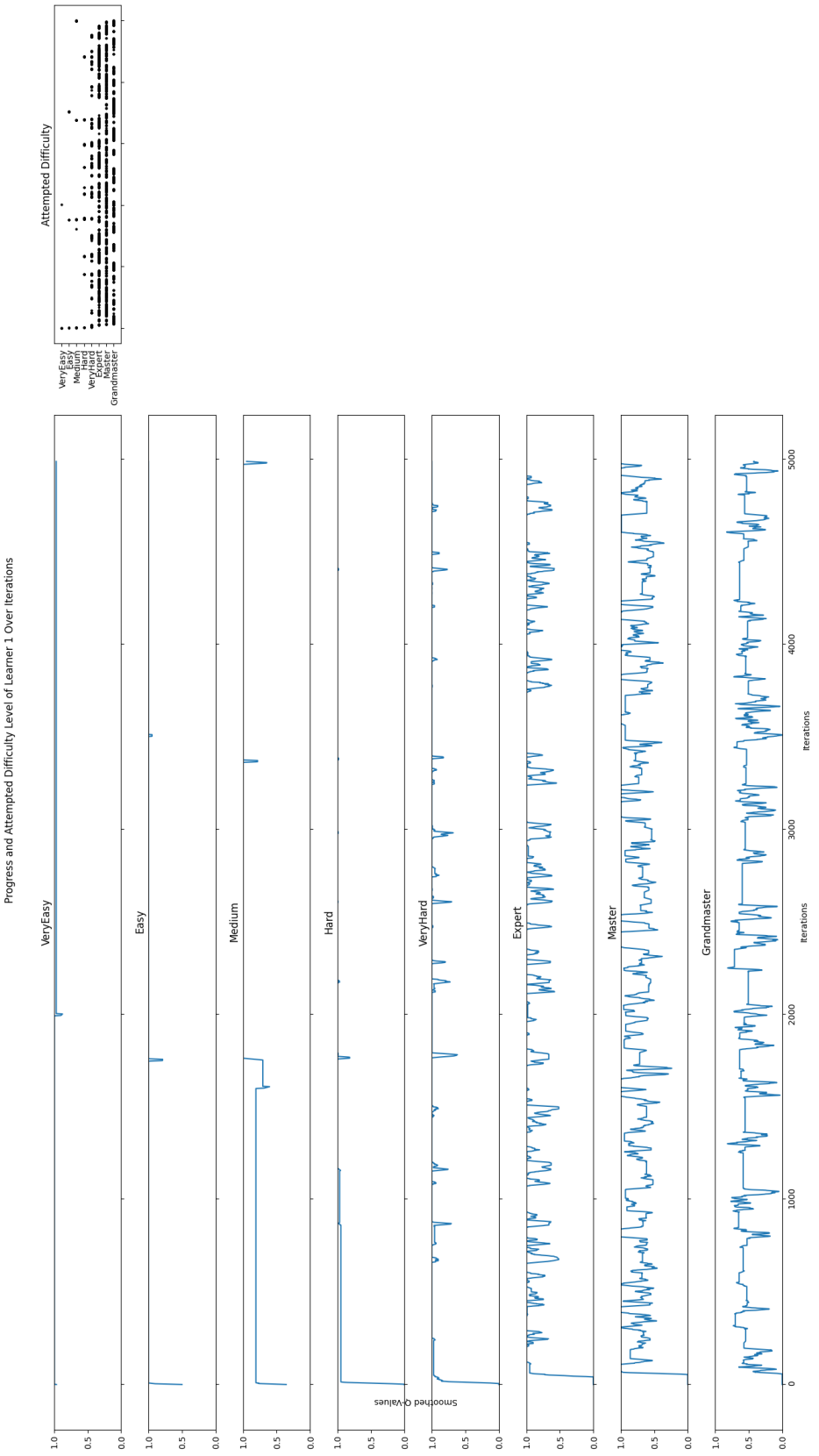

Figure A.19: Learner 1's q table and attempted difficulties over 5000 iterations of Strategy 3.

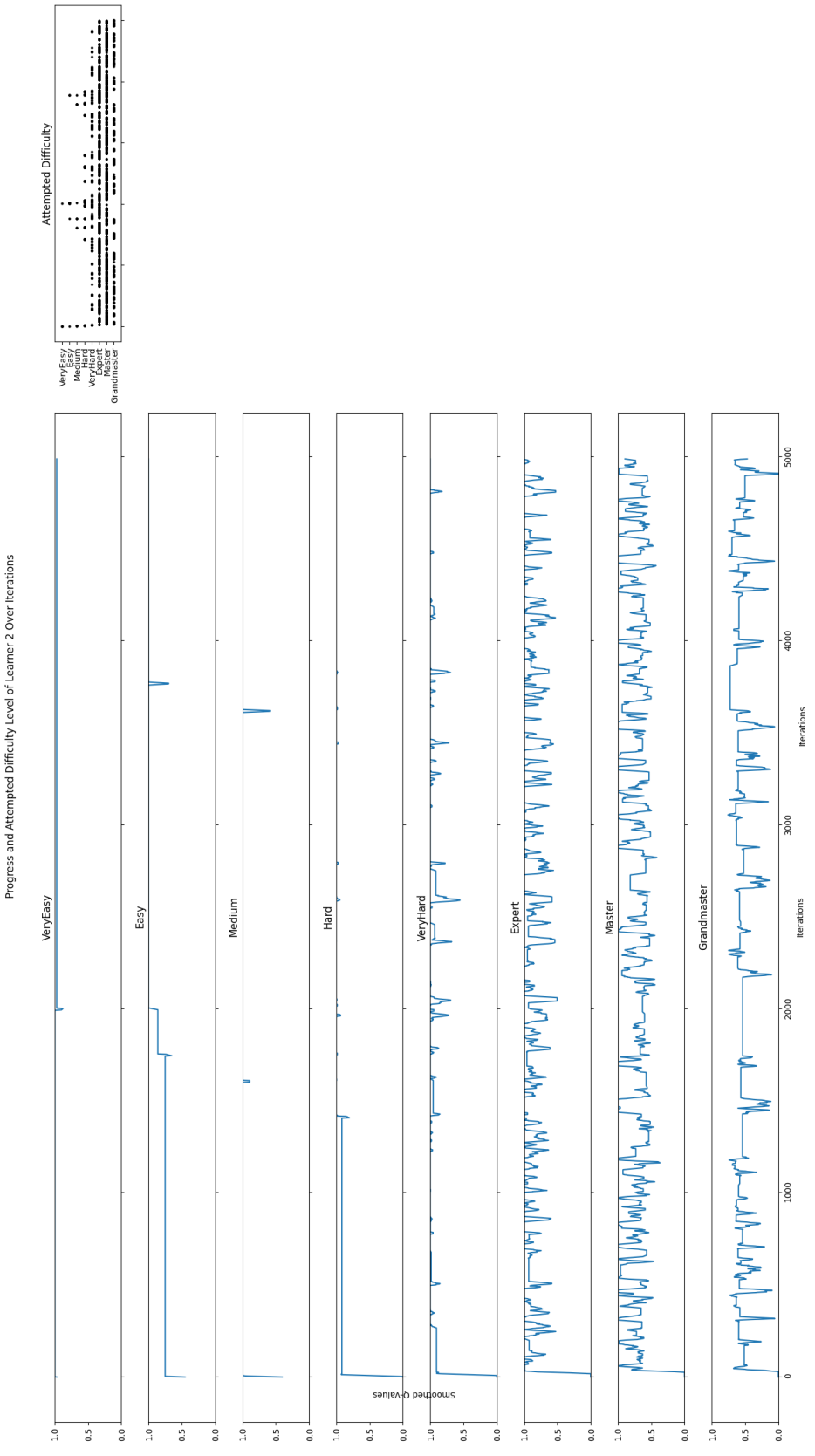

Figure A.20: Learner 2's q table and attempted difficulties over 5000 iterations of Strategy 3.

Figure A.21: Learner 3's q table and attempted difficulties over 5000 iterations of Strategy 3.

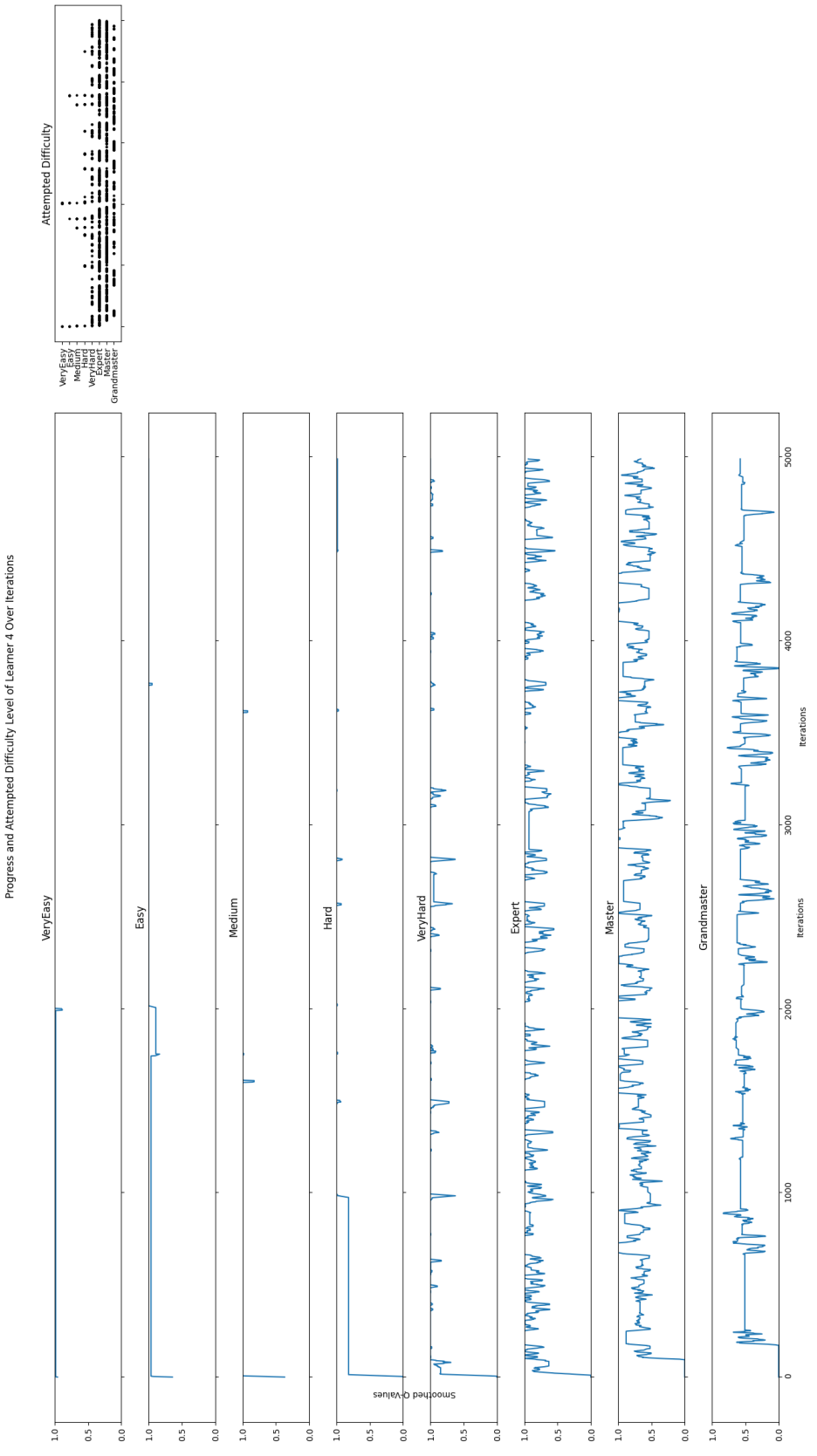

Figure A.22: Learner 4's q table and attempted difficulties over 5000 iterations of Strategy 3.

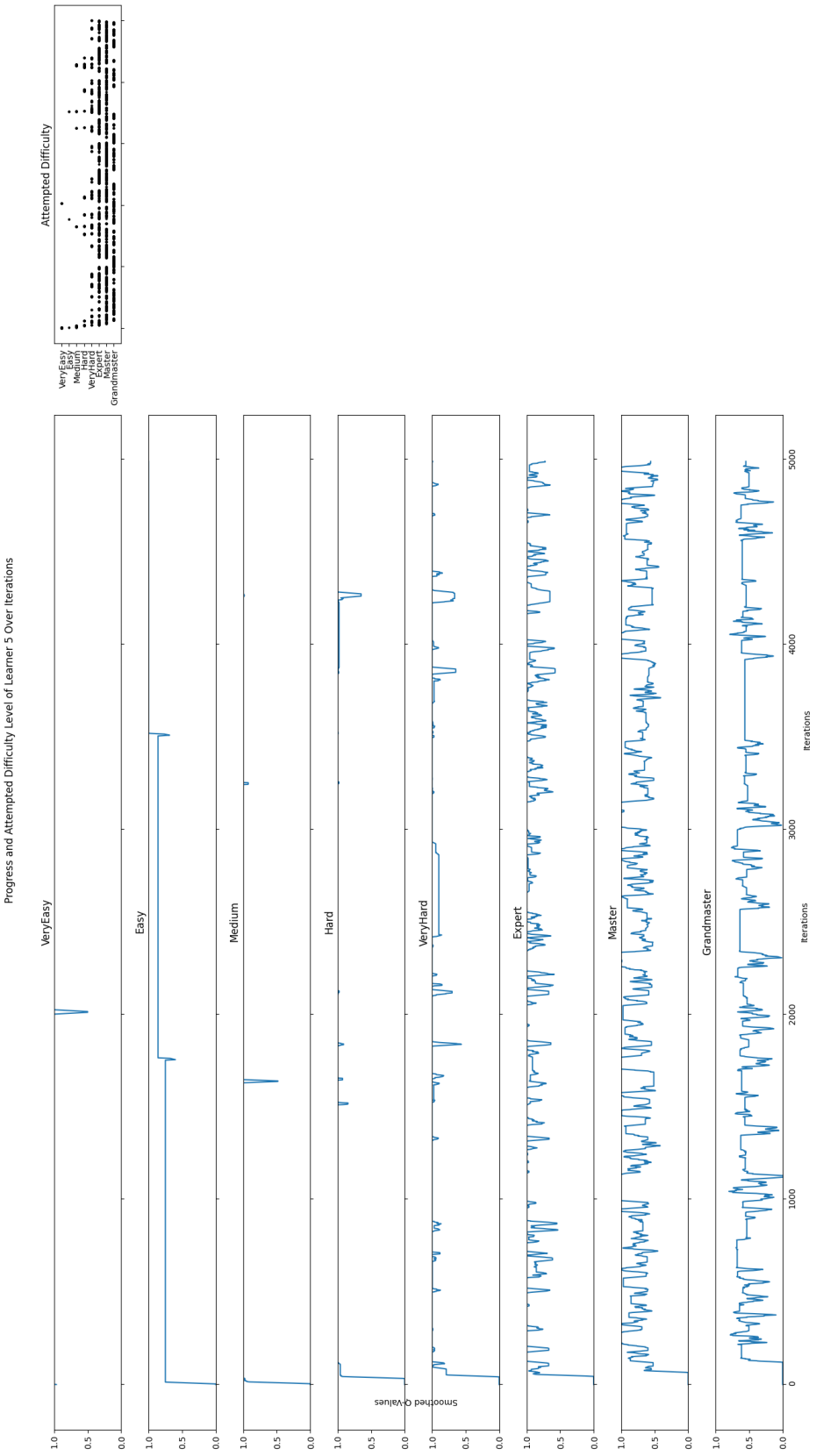

Figure A.23: Learner 5's q table and attempted difficulties over 5000 iterations of Strategy 3.

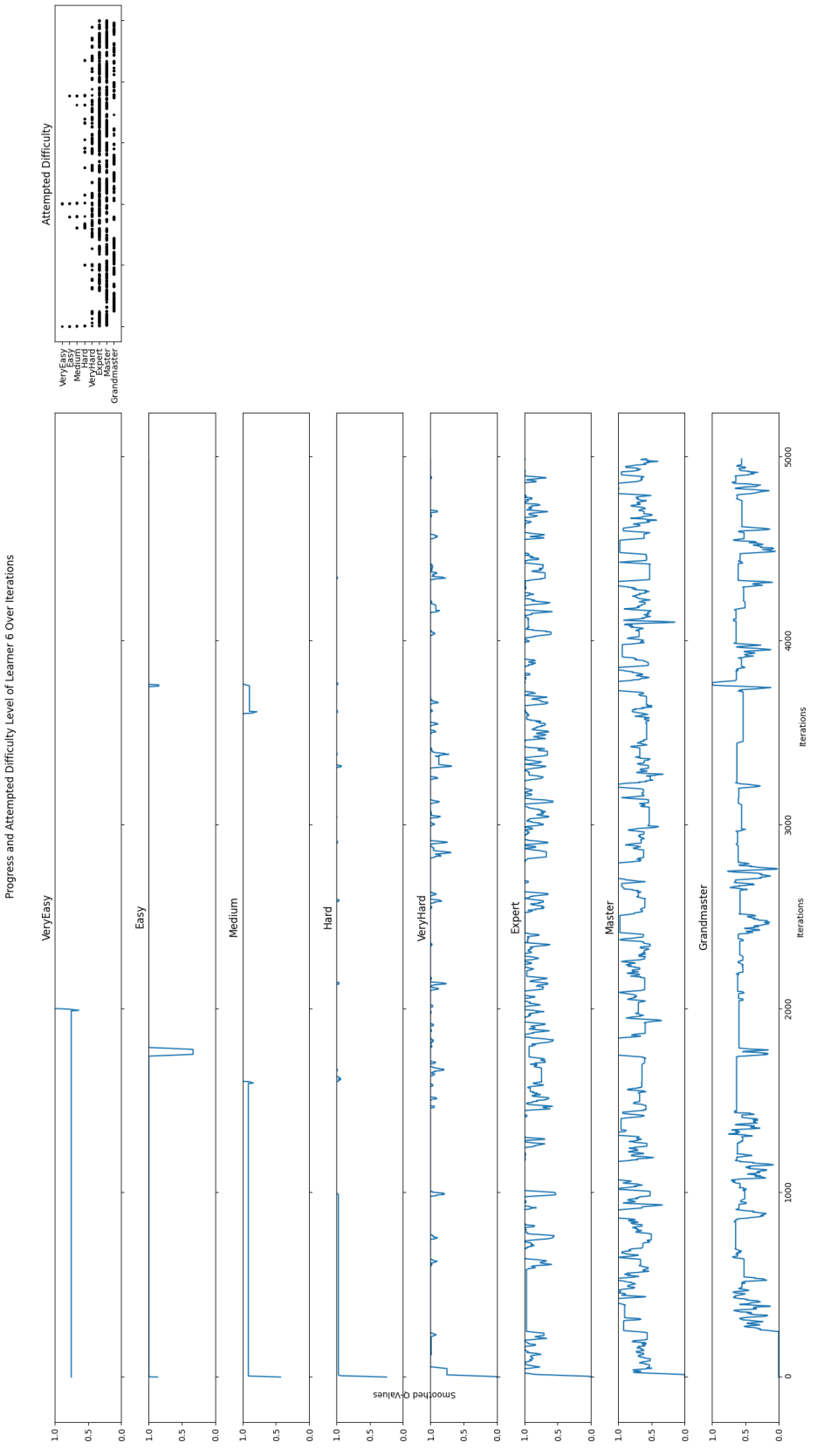

Figure A.24: Learner 6's q table and attempted difficulties over 5000 iterations of Strategy 3.

A.4 Results for Strategy 4

A.4.1 First Set: *Actions* Module

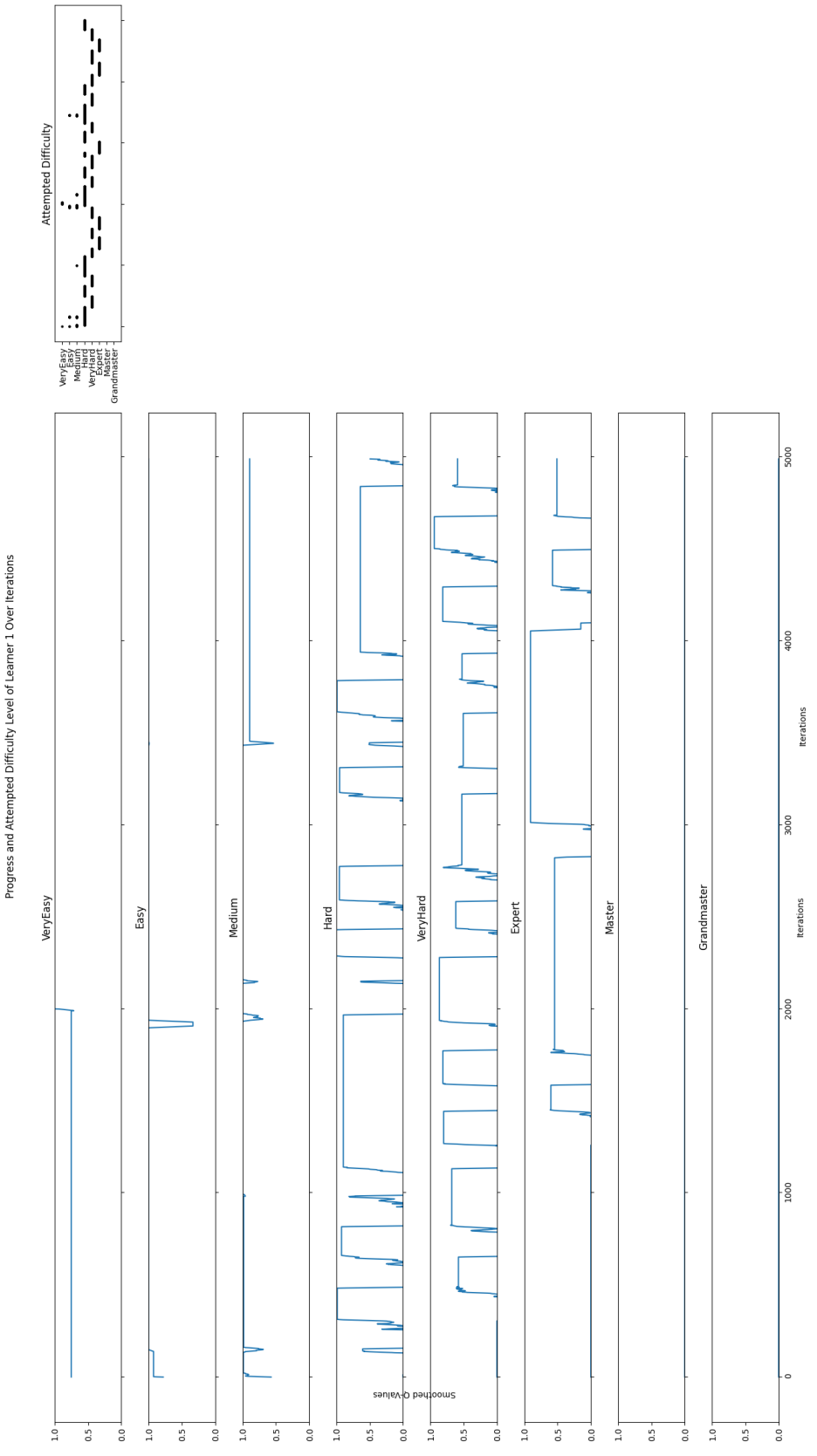

Figure A.25: Learner 1's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Actions*

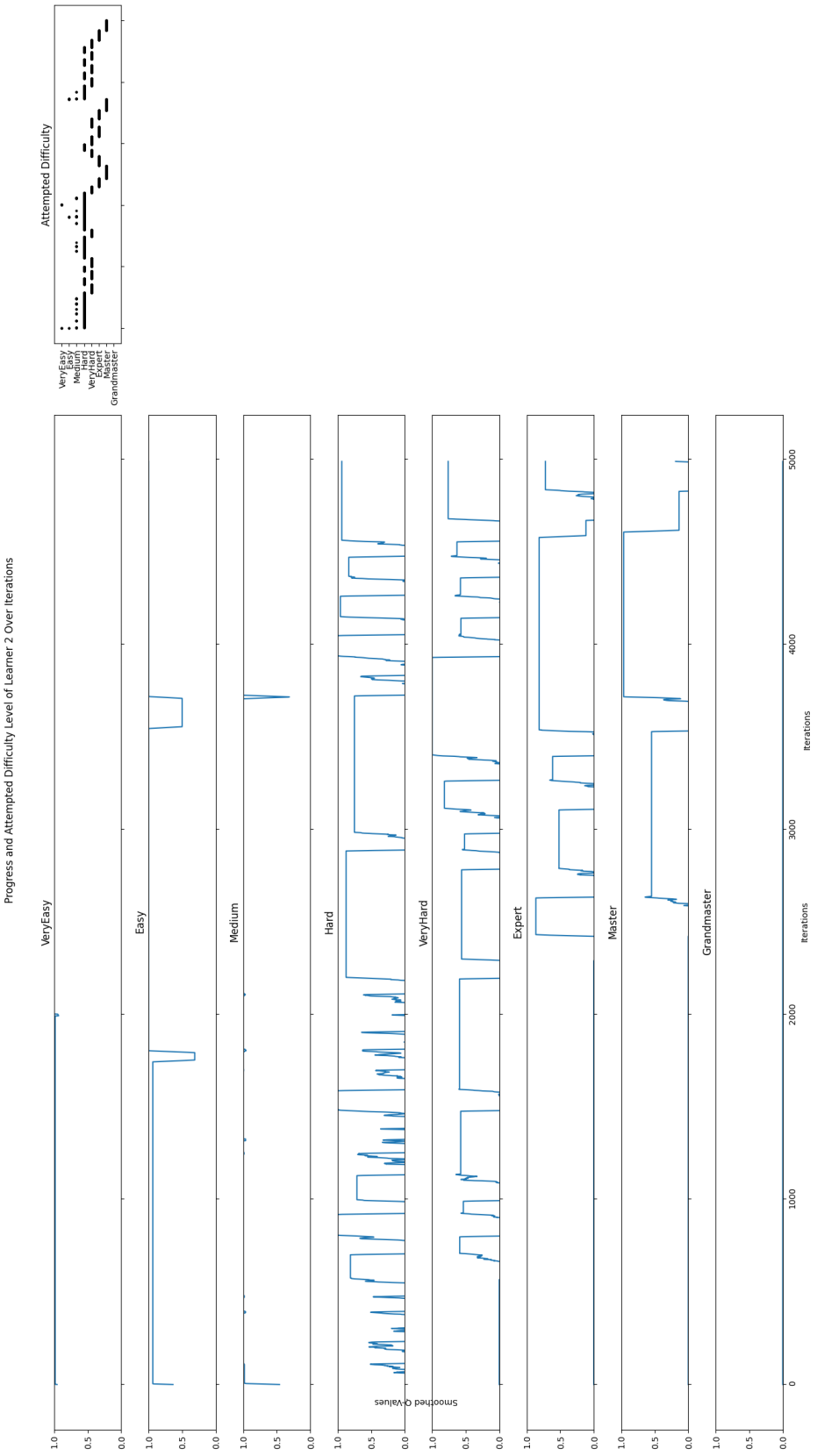

Figure A.26: Learner 2's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Actions*

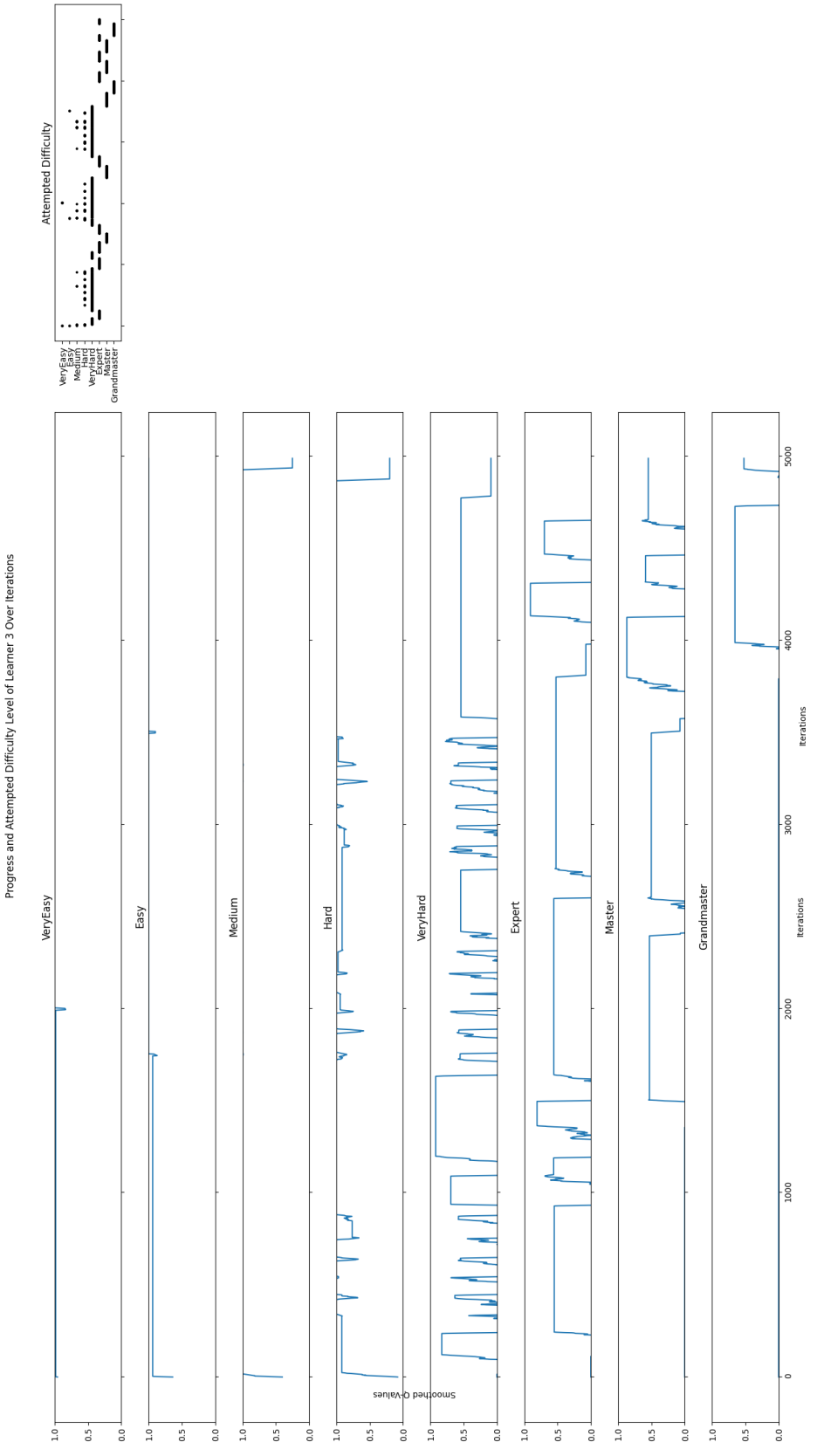

Figure A.27: Learner 3's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Actions*

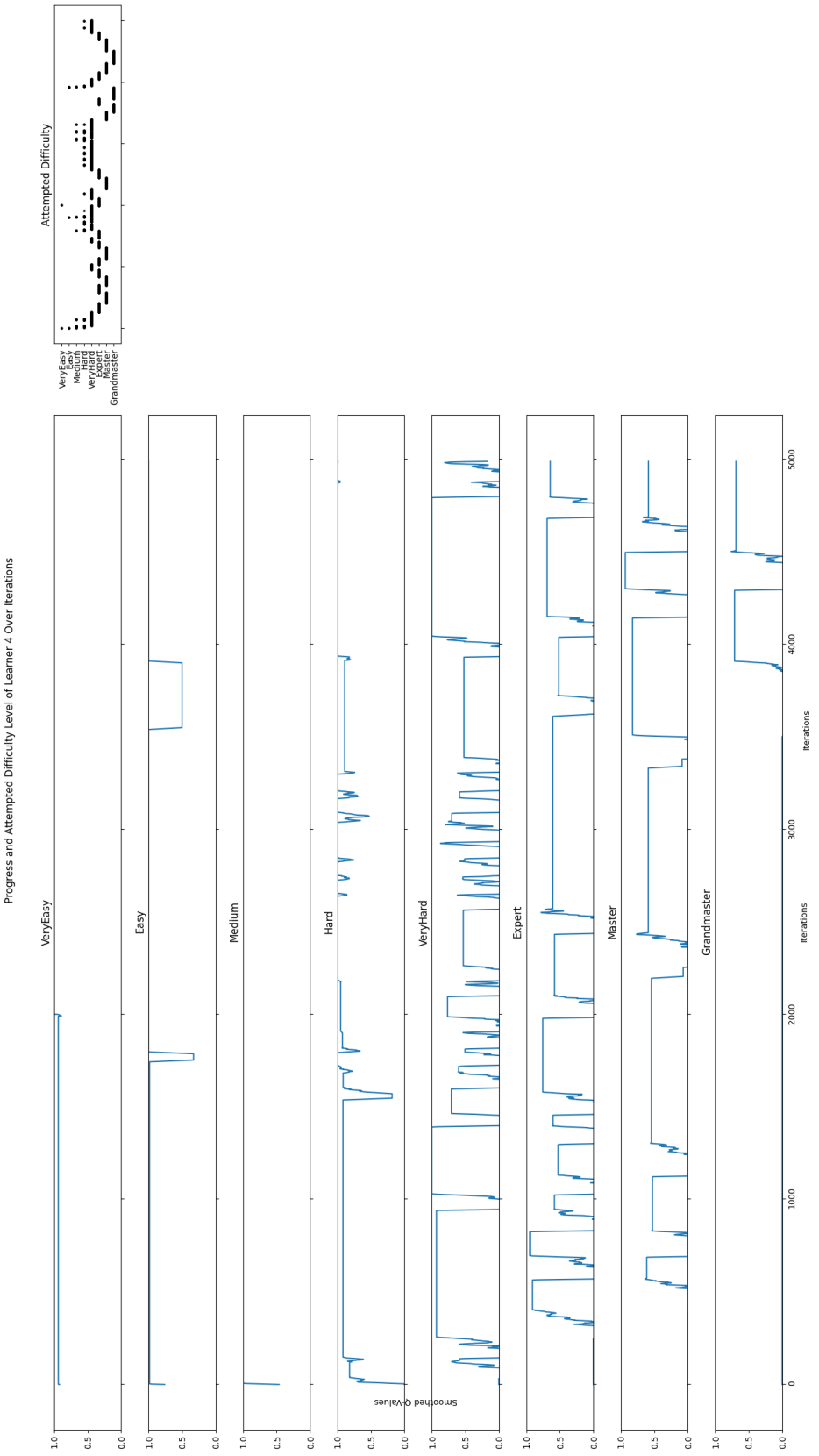

Figure A.28: Learner 4's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Actions*

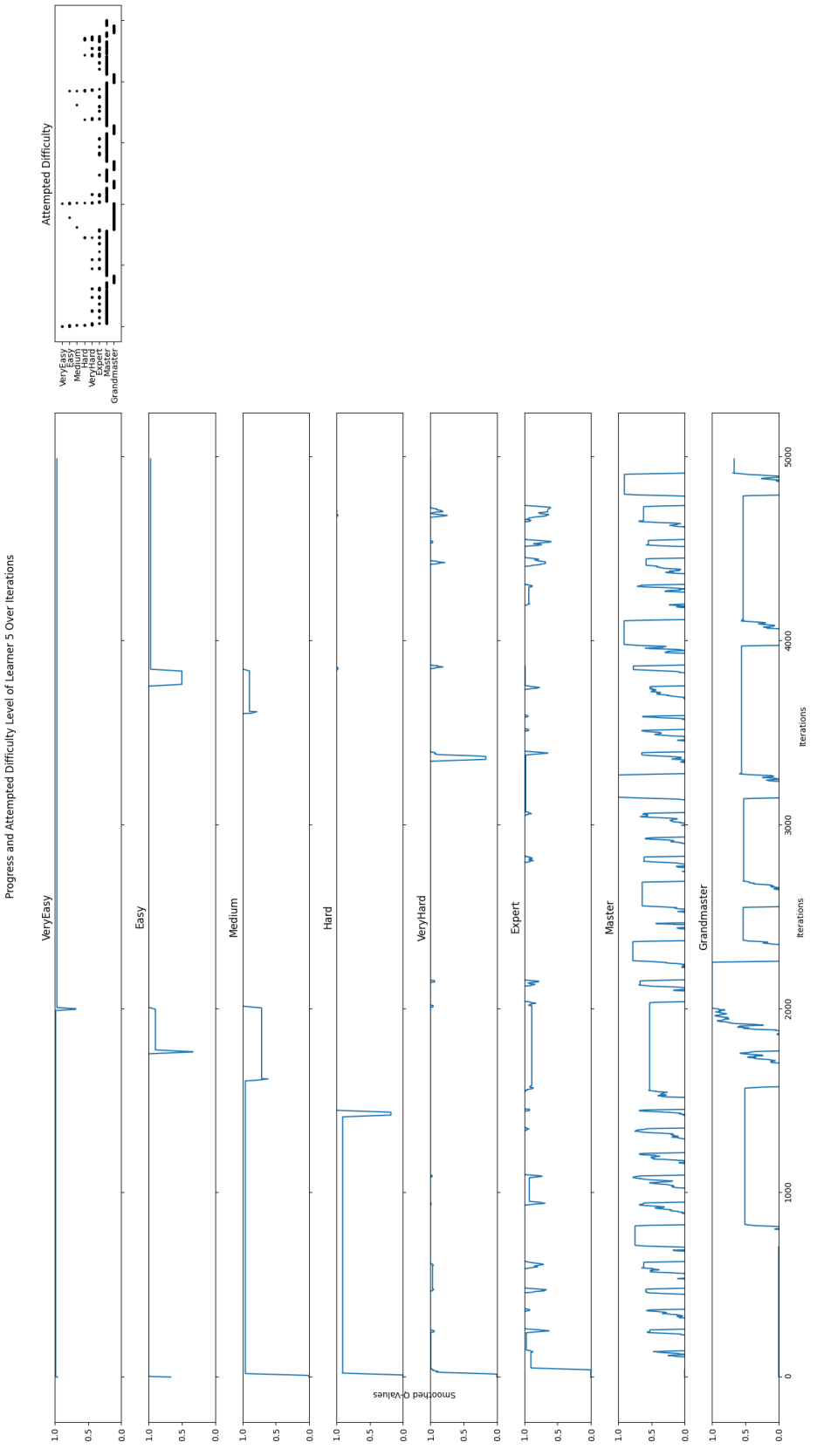

Figure A.29: Learner 5's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Actions*

Figure A.30: Learner 6's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Actions*

A.4.2 Second Set: *Shapes* Module

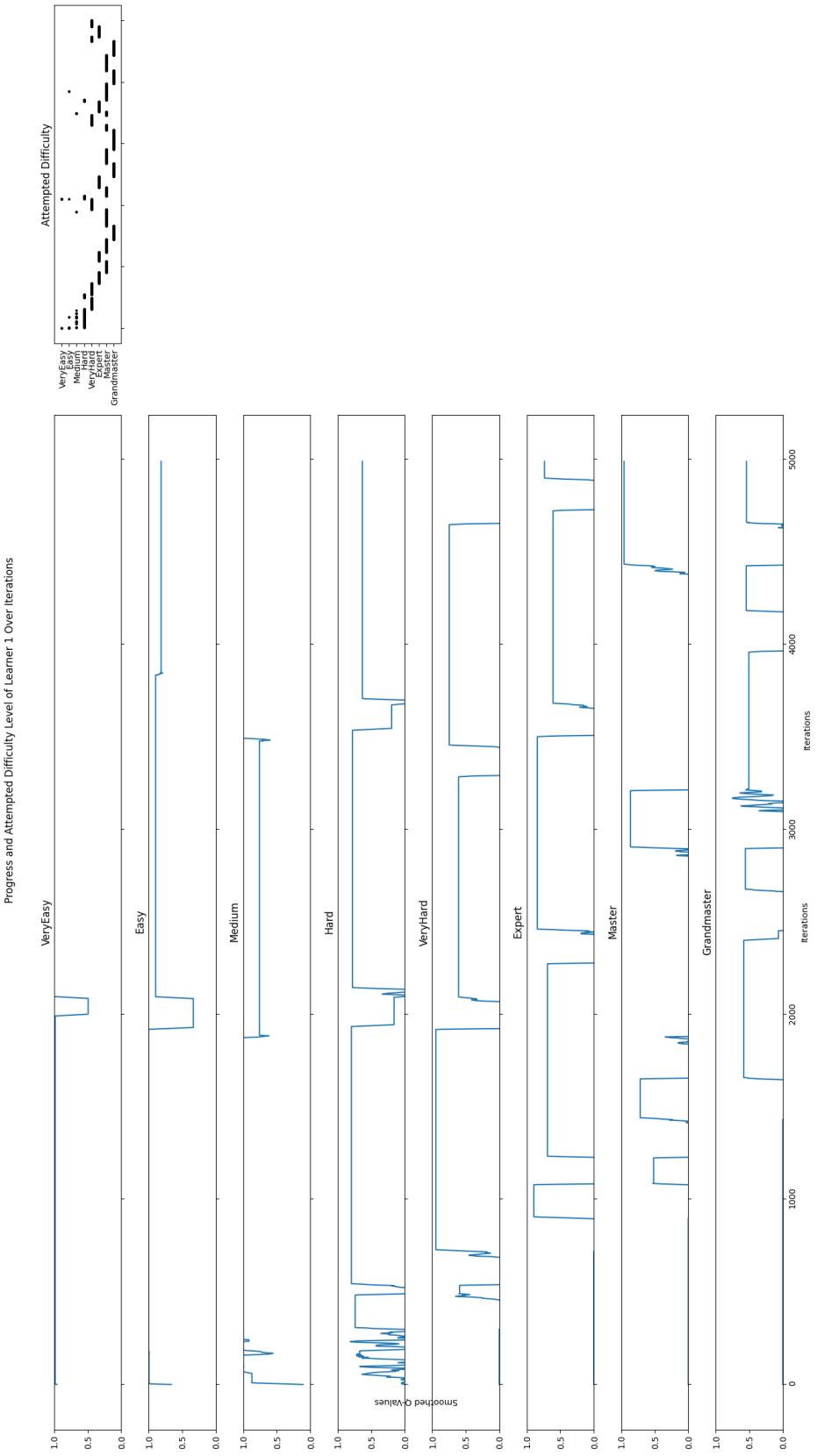

Figure A.31: Learner 1's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Shapes*

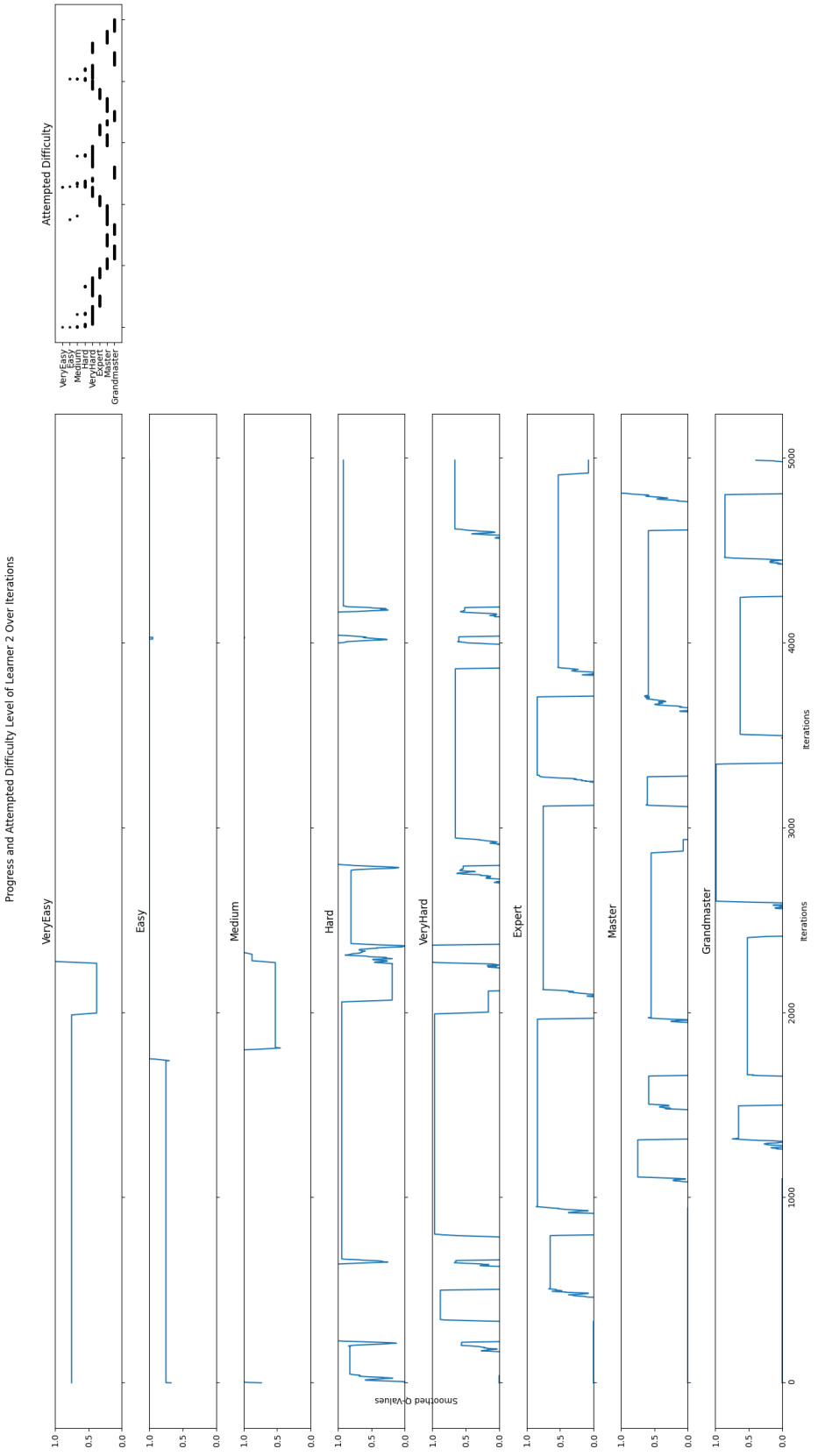

Figure A.32: Learner 2's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Shapes*

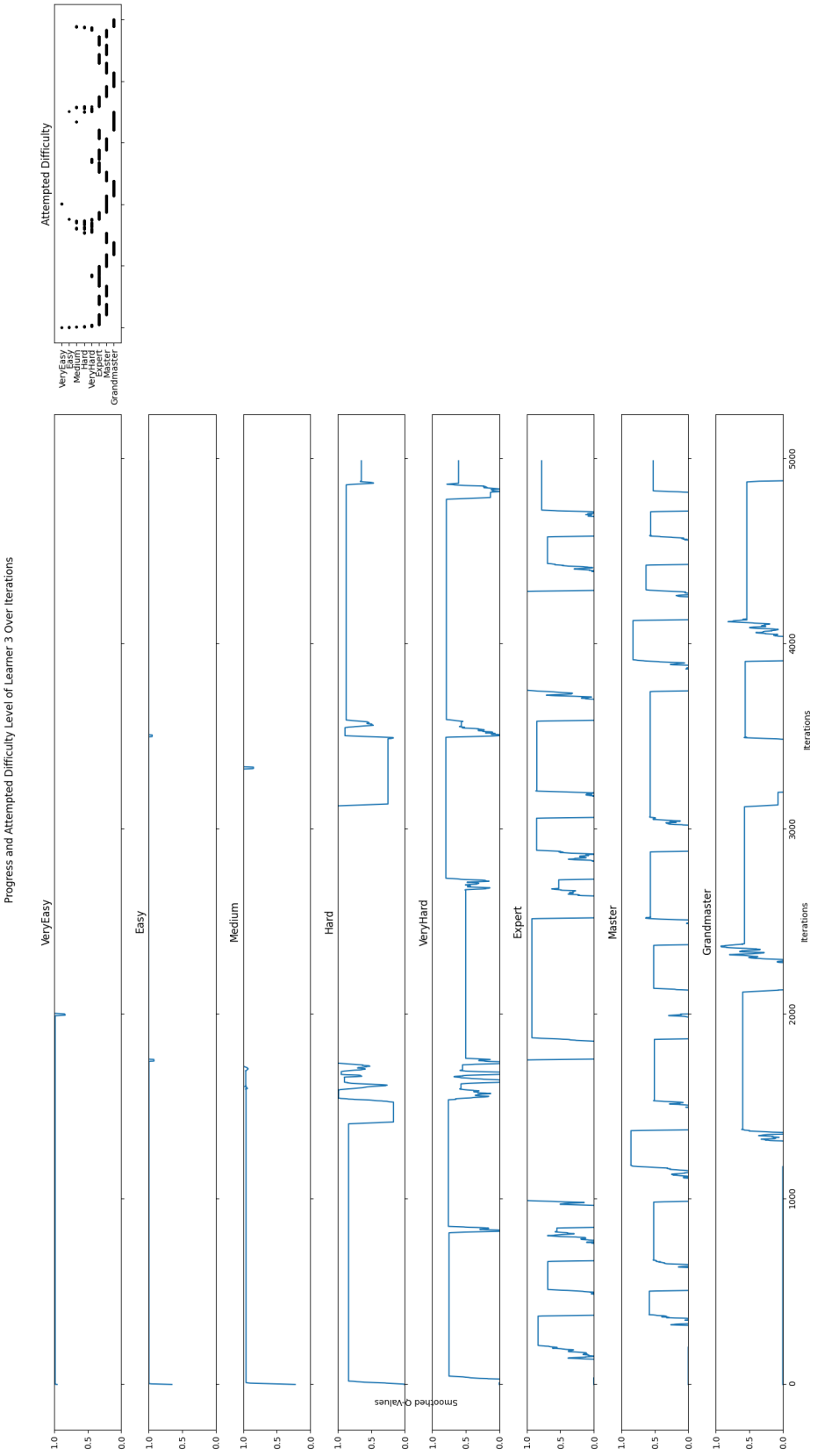

Figure A.33: Learner 3's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Shapes*

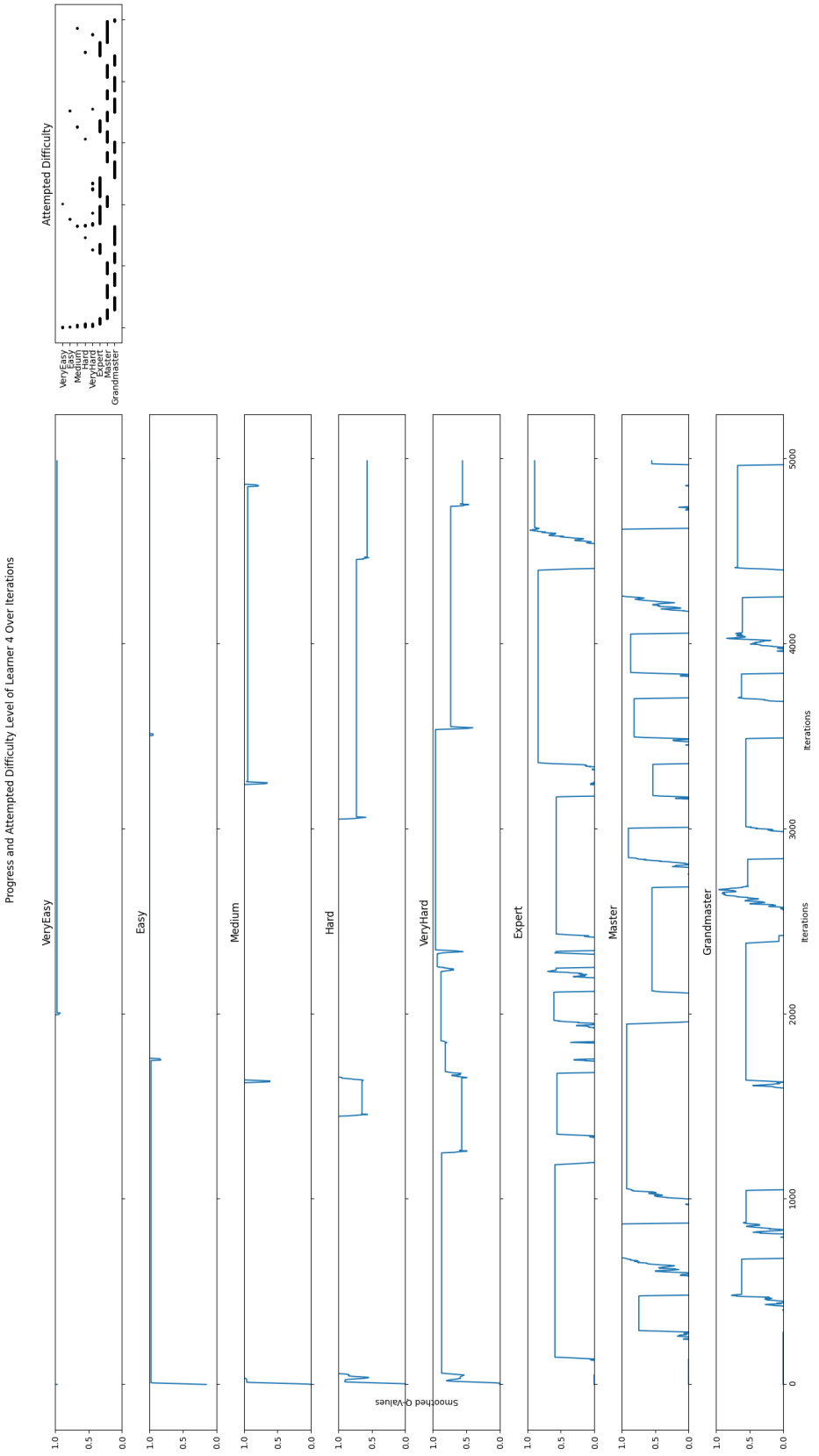

Figure A.34: Learner 4's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Shapes*

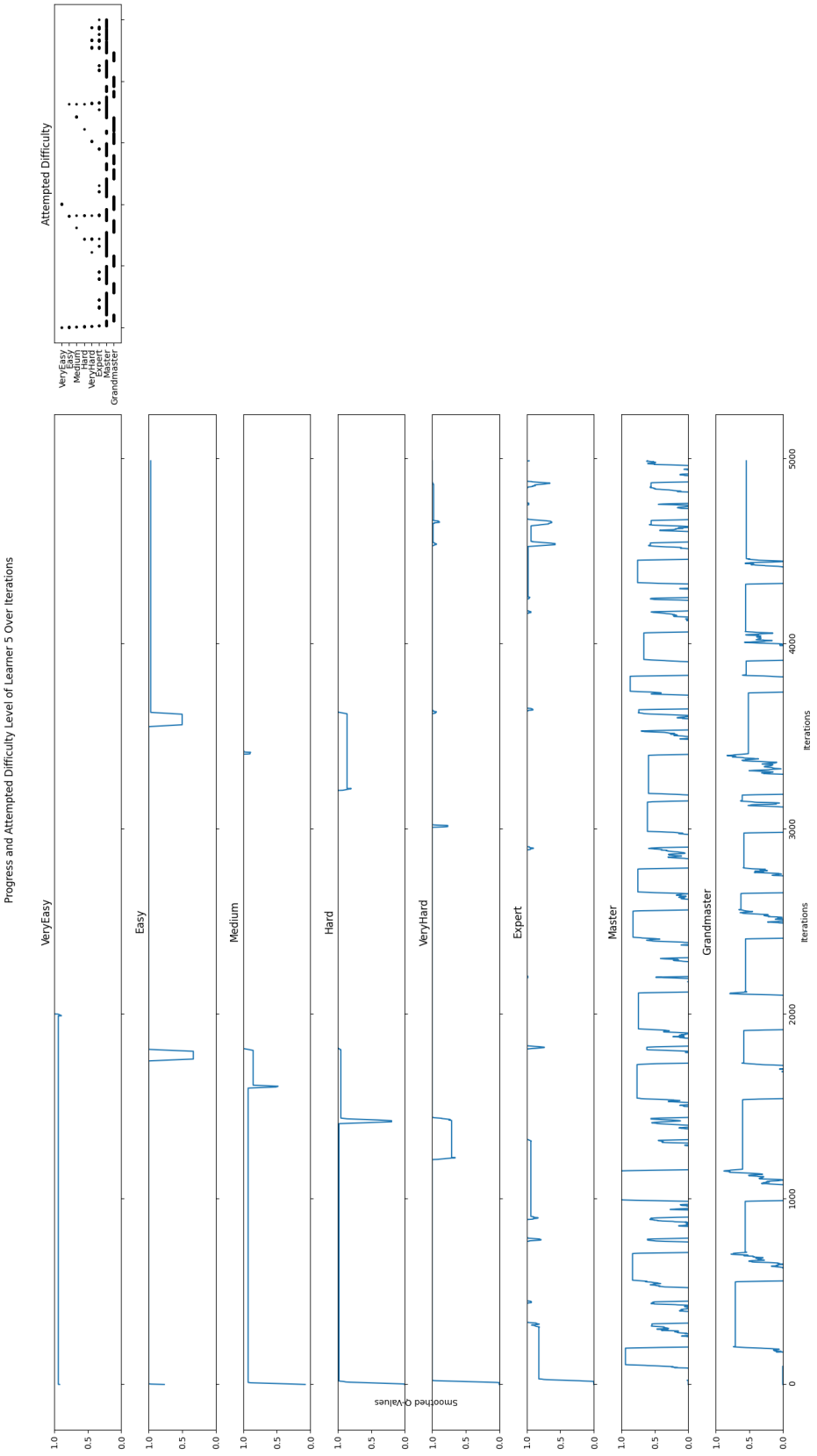

Figure A.35: Learner 5's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Shapes*

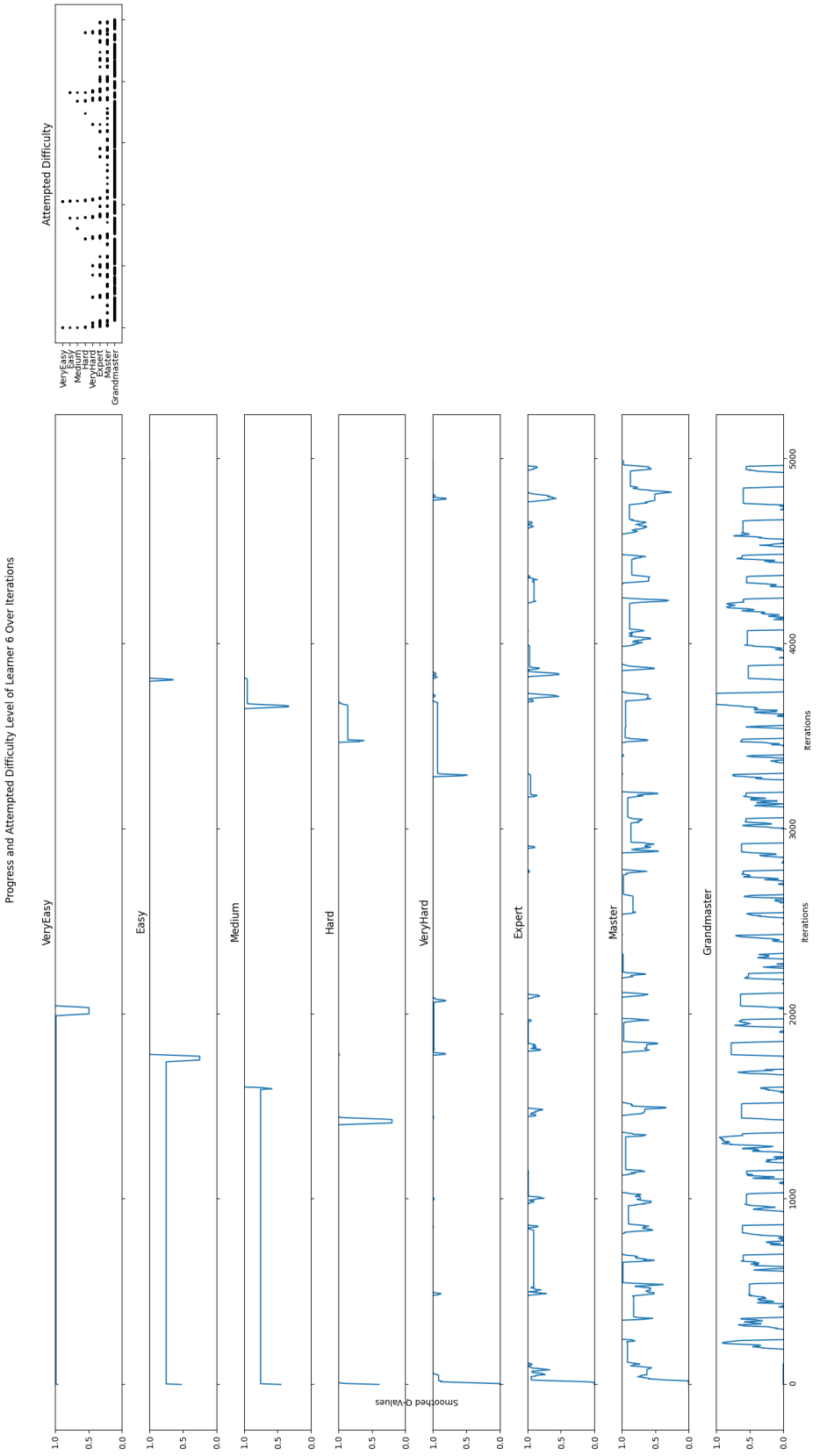

Figure A.36: Learner 6's q table and attempted difficulties over 5000 iterations of Strategy 4 for *Shapes*