Optimizing Educational Content With Q-Learning: A Reinforcement Learning Approach To Enhance Student Engagement And Outcomes

Amal Douara[1], Adil Enaanai[2] and Youssef Zaz[3]

[1, 2 & 3] Science,technology and Innovation (STI), Faculty of Sciences, Abdelmalek Essaadi University, Tetuan, Morocco
douaraamal@gmail.com

Abstract. In the dynamic realm of education, adapting and tailoring educational materials is becoming increasingly crucial to cater to the diverse requirements of learners. This presentation delves into the innovative application of Q-Learning, a Reinforcement Learning technique, to optimize educational content. We introduce an approach in which a machine learning agent models and customizes educational content based on learner interactions and performance.

Our approach revolves around establishing a learning environment where the agent's choices are represented by various types of educational content, and the rewards are determined by the success metrics of the learners. Through Q-Learning, the agent progressively acquires the optimal strategy for delivering content that maximizes learner engagement and comprehension.

This research explores the integration of Q-Learning, a Reinforcement Learning method, into educational systems such as Moodle to cater to the individual needs of students. The methodology employs student interactions and academic performance data to construct a Q-Learning model. This model evaluates the effectiveness of pedagogical actions across different educational contexts with the ultimate goal of enhancing student engagement and improving learning outcomes.

Keywords: Q-Learning, Reinforcement learning, Adaptive learning.

1 Introduction

Over recent decades, education has shifted from a uniform approach to increasingly personalized learning experiences. This change is driven by recognizing each learner's unique needs, interests, and learning styles. Personalization in education aims to tailor content, pace, and teaching methods to optimally meet these individual needs [1]. It has become crucial for enhancing learner engagement, increasing knowledge retention, and ultimately improving learning outcomes.

At the heart of this transformation is the adoption of advanced technologies, including Artificial Intelligence (AI) and specifically Reinforcement Learning (RL). RL, a branch of AI, involves machines learning to make decisions by interacting with their

M. Khaldi et al. (eds.), Proceedings of the E-Learning and Smart Engineering Systems (ELSES 2024), Atlantis Highlights in Social Sciences, Education and Humanities 36,
https://doi.org/10.2991/978-2-38476-408-2_15

environment, performing actions, and receiving feedback in the form of rewards or penalties [2]. This feedback enables them to learn and optimize their behavior over time. Q-Learning, an RL algorithm, stands out for its ability to evaluate the quality (Q) of each action in a given state, leading to optimal decisions without needing a predictive model of the environment.

This research aims to explore and demonstrate how Q-Learning can be effectively applied in the educational context to optimize educational content. By integrating Q-Learning into the development and presentation of learning content, the goal is to implement an algorithm that dynamically adapts to individual learners' needs, offering a truly personalized educational experience.

2 Theoretical foundation

2.1 Reinforcement Learning (RL)

Reinforcement Learning (RL) is a significant branch of artificial intelligence focusing on how agents should make decisions in an environment to maximize cumulative reward [3]. It's a learning type where an agent learns to behave in an environment by performing actions and seeing their results. RL's fundamental principles can be broken down into several key components:

- Agent: In RL, the agent is likened to a learner or decision-maker. It's programmed to make decisions based on observations. This decision-making process is the core of reinforcement learning. An agent must evaluate situations (states) and take actions that bring it closer to its goal (maximizing rewards).
- Environment: The environment in RL is the space where the agent operates. It provides feedback to the agent in terms of states and rewards. The environment is often modeled as a Markov decision process (MDP), where the outcome of an action in a given state is partially random.
- State:Each state represents an instantaneous snapshot of the environment at a given time. In research, defining states is critical as it directly influences the agent's ability to learn effectively. A good state should provide enough information for the agent to make an informed decision without being overloaded with unnecessary information.
- Actions: Actions are decisions that the agent can make at each step. The choice of actions directly influences the future state of the environment and the reward received. Actions can be discrete (e.g., turn left or right) or continuous (e.g., set a thermostat to a specific temperature).
- Rewards: Rewards are essential signals in RL. After each action, the agent receives a reward (or penalty) from the environment. This reward guides the agent to learn which actions are beneficial and which should be avoided. The reward function must be designed to encourage the agent to achieve long-term goals.
- Policy: The policy is fundamentally a map that guides the agent in its action choices based on the current state. It can be deterministic (a specific action for each state) or

stochastic (probabilities assigned to different actions). The policy is what the agent optimizes during the learning process.
- Value Function: The value function estimates what the agent can expect to gain, in terms of future rewards, by being in a certain state or following a certain policy. The value function helps to evaluate whether a state is advantageous for the agent and influences future decisions.
- Learning and Optimization: Learning in RL occurs when the agent adjusts its policy based on acquired experience (visited states, taken actions, and received rewards). Optimization involves finding the policy that maximizes the sum of future rewards. This often involves a balance between exploration (trying new actions for better rewards) and exploitation (using current knowledge to maximize reward).

Reinforcement learning (RL) is a powerful framework for enabling agents to learn optimal decision-making strategies through interactions with their environment. The key components of RL, including the agent, environment, states, actions, rewards, policy, value function, and the balance between learning and optimization, all play crucial roles in shaping the agent's behavior and its ability to achieve its goals. By continuously refining its policy based on feedback from the environment, an RL agent can adapt to complex and dynamic situations, making it a valuable tool for a wide range of applications.

2.2 RL Algorithms

In the field of Reinforcement Learning (RL), several algorithms [4] have stood out for their effectiveness and popularity. Often used as starting points for advanced research or benchmarks for new ideas, here are some of the most popular RL algorithms:

- Q-Learning: An off-policy learning algorithm that learns the value of the best action in a given state without requiring a model of the environment.
- SARSA (State-Action-Reward-State-Action): An on-policy learning algorithm updating its policy based on the current state, action taken, reward received, and next action.
- Deep Q-Networks (DQN): Merges Q-Learning with deep neural networks, popularized after being used by DeepMind for Atari games.
- Policy Gradients: Directly learns the action policy, often used in environments with continuous actions.
- Actor-Critic Methods: Combine policy gradients (actor) and value-based methods (critic), learning the optimal policy and evaluating its performance.
- Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO): Advanced policy gradient methods focusing on stable, robust learning.
- Asynchronous Advantage Actor-Critic (A3C): Uses multiple agent instances working asynchronously for diversified experience and reduced sub-optimal policy convergence risk.
- Monte Carlo Tree Search (MCTS): Often combined with RL in complex games like Go, based on building a search tree and simulating outcomes.

- Soft Actor-Critic (SAC): An algorithm for continuous action environments, balancing exploitation and exploration.
- Rainbow: Enhances the original DQN with various techniques for improved performance and stability.

The following table represents the comparison of popular RL algorithms and the suitability of Q-Learning for educational content modeling:

Table 1. Comparison of popular RL algorithms.

Algorithm	Type of Algorithm	Complexity	Suitable Environments	Main Advantages
Q-Learning	Value-Based	Low	Discrete	Simple, easy to implement, effective in discrete state and action spaces
SARSA	Value-Based	Low	Discrete	Accounts for current policy, good for on-policy learning
Deep Q-Network (DQN)	Value-Based	High	Discrete, Complex	Suitable for large and complex state spaces, deep learning
Policy Gradients	Policy-Based	Medium	Continuous	Directly optimizes policy, suited for continuous actions
Actor-Critic	Hybrid	Medium	Discrete, Continuous	Combines advantages of value-based and policy methods
PPO/TRPO	Policy-Based	High	Continuous	Improves learning stability, effective in complex environments
A3C	Hybrid	High	Discrete, Continuous	Asynchronous learning, efficient in parallel, robust
SAC	Policy-Based	High	Continuous	Balances exploration and exploitation, robust in continuous environments
MCTS	Search-Based	Variable	Games, Sequential Decision	Powerful for planning and decision-making in complex games

In the context of educational content modeling, Q-Learning emerges as a favorable choice for several reasons:

- Simplicity and Ease of Implementation: Q-Learning is relatively easy to understand and implement, making it significantly advantageous for educational RL integration projects.
- Effectiveness in Discrete State Spaces: Educational environments, often characterized by discrete states like skill levels or content types, are well-suited for Q-Learning.
- No Dependence on Environmental Model: Being an off-policy algorithm, Q-Learning doesn't require prior knowledge or modeling of the environment, fitting situations where full educational environment modeling is challenging.

- Adaptability and Flexibility: Q-Learning easily adapts to changes in learner preferences and performance, crucial in educational systems with uniquely varied learner needs.

For these reasons, Q-Learning is considered a robust and practical method to start with RL in educational applications, particularly when developing a system capable of dynamically adjusting to individual learner needs.

2.3 Q-Learning

Q-Learning, a Reinforcement Learning (RL) technique, doesn't require a predictive model of the environment. It focuses on learning the Q-value function, Q(s, a), estimating the quality of an action 'a' in a state 's'. The goal is to develop a policy guiding the agent to choose actions that maximize future rewards. The Q-function updates via the rule:

$$Q\_new(s, a) = Q(s, a) + \alpha[R(s, a) + \gamma max\_a'Q(s', a') - Q(s, a)] \quad (1)$$

where α is the learning rate, γ the discount factor, R(s, a) the received reward, and max_a'Q(s', a') the maximum Q-value for the next state 's'.

Understanding Q-Learning involves exploring its key components and their interaction in the learning process [5].

- The Q-Value Function: The Q-value function, Q(s, a), is fundamental in Q-Learning, estimating the expected value of taking action 'a' in state 's'. This value assesses the long-term utility of the action, considering immediate and future rewards. The goal is to maximize Q-value throughout the learning process.
- State (s) and Action (a): In Q-Learning, a state 's' represents a specific situation or configuration for the agent. An action 'a' is a decision the agent can make in that state. States and actions vary according to the problem being solved.
- Updating the Q-Value: Q-value is updated using the formula Q_new(s, a) = Q(s, a) + α[R(s, a) + γmax_a'Q(s', a') - Q(s, a)], where each term has a specific meaning:
- α (learning rate): Determines how new information supersedes old information. A higher α means faster learning.
- R(s, a): The immediate reward received after taking action 'a' in state 's'.
- γ (discount factor): Reduces the value of future rewards, encouraging short-term rewards over long-term ones.
- max_a'Q(s', a'): The maximum Q-value for possible actions in the next state s', reflecting Q-Learning's forward-looking aspect.

The action policy in Q-Learning is the rule the agent follows to choose its actions. It's usually determined by the action with the highest estimated Q-value in a given state. Balancing strategies like exploration (randomly choosing actions to discover new information) and exploitation (choosing actions based on current knowledge) optimizes the learning process.

3 Methodology

In our study, the learning environment is based on Moodle, an online learning platform widely used in Moroccan higher education. Moodle provides a rich, interactive environment with various modules and activities, such as forums, quizzes, and downloadable resources. Each course in Moodle represents a distinct state in our Q-Learning model, with actions corresponding to different activities or resources offered to students.

The data for Q-Learning comes from several sources within Moodle:

- Student Interactions: This includes browsing data (pages visited, time spent on each resource), assignment submissions, and quiz responses.
- Academic Performance: Grades from assessments and quizzes give direct measures of student performance.
- Student Feedback: Surveys and discussion forums offer qualitative insights into student preferences and difficulties.

The Q-Learning model is trained using collected data to optimize educational content selection. Each state (course/module) is evaluated based on actions (resources or activities) and associated rewards, grounded in student performance improvement and engagement. The model learns which content types lead to better performance and higher engagement in various contexts.

Content selection and presentation based on learned Q-values, the customized Moodle system recommends specific activities to students. For instance, for a student struggling in a subject, it might suggest additional resources like video tutorials or extra exercises. For high-performing students, more advanced content or enrichment activities might be offered.

To assess the approach's effectiveness, these metrics are used:

- Academic Performance: Improvement in quiz and assessment grades.
- Engagement: Measured by time spent on the platform and interaction frequency with recommended content.
- Student Satisfaction: Gathered via feedback surveys on their experience with the recommended content.
- Course Success Rate: Percentage of students passing the course relative to enrollments.

These metrics will analyze the impact of the Q-Learning-based system on enhancing academic performance and engagement of first-year master's students on Moodle.

4 Modeling

4.1 Q-Learning conceptual diagram

To conceptualize a Q-Learning system for educational applications, start with a schematic describing the system's key components and their interactions.

- Environment: The online learning system (like Moodle) where students interact with content.
- States: Represent specific student situations, such as course progress, quiz scores, forum participation, etc.
- Actions: Various types of content or recommended activities (videos, readings, additional quizzes, etc.).
- Rewards: Based on student engagement (time spent, platform activity) and performance (quiz score improvements, active participation, etc.).

The process begins with observing a state (see Fig. 1), followed by selecting an action based on the policy derived from the Q-table. The action leads to a reward and a new state, which are then used to update the Q-table.

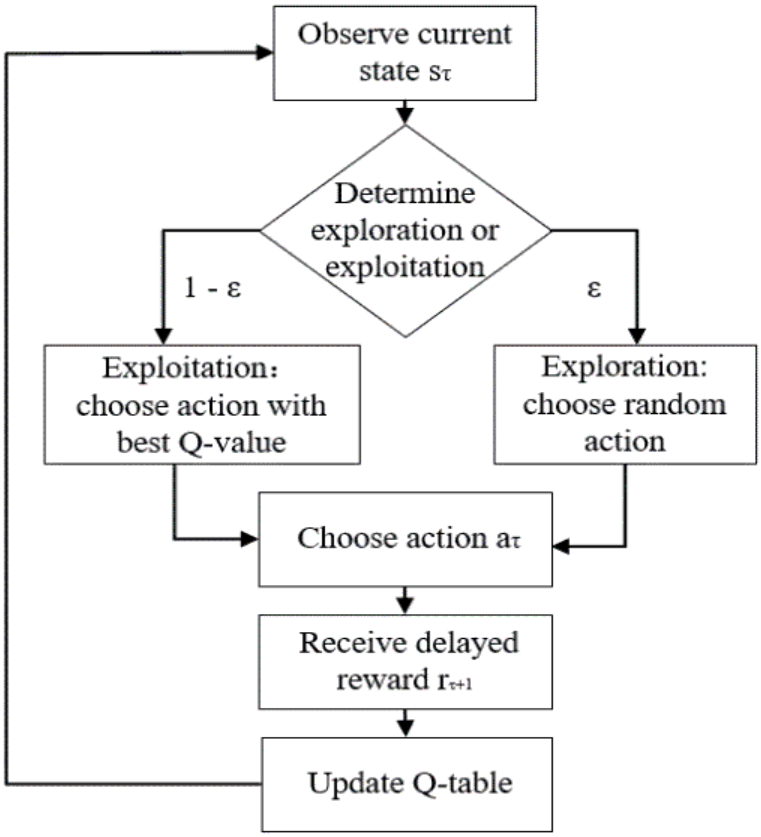

Fig. 1. Q-Learning algorithm flow chart.

In defining how rewards are calculated and how states are updated in a Q-Learning system applied to an educational environment like Moodle, consider the following aspects:

- Reward Calculation
  - Student Engagement: Measured by time spent on the platform, connection frequency, or number of interactions. Higher rewards for increased engagement.
  - Academic Performance Improvement: Evaluate improvements in quiz scores, assignment grades, etc. Performance increases lead to higher rewards.

  - Active Participation: Encouraged through rewards for active participation in forums or group activities.
  - Qualitative Feedback: Incorporate student feedback to adjust rewards. Positive feedback translates into higher rewards.
- State Updates
  - Competence Level: Define states based on student's competency level in different subjects, based on quiz scores or teacher assessments.
  - Course Progression: Consider the student's course stage as states. Progress through course content leads to state changes.
  - Learning Styles and Preferences: States may also reflect students' learning styles or content preferences.

4.2 Q-Learning python code

The following code is a modeling of the Q-Learning algorithm. It is important to note that this code represents a basic structure of our understanding of the algorithm.

```python
import numpy as np
import random
# Paramètres du Q-Learning
alpha = 0.1  # Taux d'apprentissage
gamma = 0.6  # Facteur de remise
epsilon = 0.1  # Probabilité d'exploration
# Supposons que les états sont les niveaux de compétence des étudiants et les acti
niveaux_competence = 10  # Exemple: 0-9
types_contenu = 3  # Exemple: 0-2
q_table = np.zeros((niveaux_competence, types_contenu))
# Fonction de mise à jour de la table Q
def update_q_table(state, action, reward, new_state):
    old_value = q_table[state, action]
    next_max = np.max(q_table[new_state])
    new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)
    q_table[state, action] = new_value
# Fonction de choix d'action
def choose_action(state):
    if random.uniform(0, 1) < epsilon:
        return random.choice(range(types_contenu))
    else:
        return np.argmax(q_table[state])
# Fonctions pour calculer la récompense et mettre à jour l'état
def calculate_reward(student_interaction, academic_performance):
    reward = 0
    reward += student_interaction.time_spent / max_time  # Normaliser par le temps
    reward += academic_performance.improvement / max_improvement  # Normaliser par
    return reward
def update_state(current_state, student_performance):
    if student_performance.quiz_score > threshold:
        new_state = current_state + 1  # Avancer vers un état plus avancé
    else:
        new_state = current_state  # Maintenir l'état actuel
    return new_state
# Boucle principale de Q-Learning
for _ in range(1000):
    # Obtenez les données de l'étudiant (simulées ou réelles)
    student_interaction = ...  # Données d'interaction de l'étudiant
    student_performance = ...  # Données de performance de l'étudiant

    state = random.randint(0, niveaux_competence - 1)
    action = choose_action(state)
    # Calculez la récompense et mettez à jour l'état
    reward = calculate_reward(student_interaction, student_performance)
    new_state = update_state(state, student_performance)
    # Mettez à jour la table Q
    update_q_table(state, action, reward, new_state)
```

Fig. 2. Q-Learning algorithm in python.

The code begins by importing necessary libraries—numpy for numerical computations and random for generating random numbers. Several key parameters of the Q-Learning algorithm are defined: the learning rate 'alpha', the discount factor 'gamma', and the 'epsilon' for the epsilon-greedy strategy, which helps in balancing the exploration and exploitation during the learning process. The environment is composed of states that represent the levels of student competencies, ranging from 0 to 9, and three

types of content that can be served as actions. A Q-table is initialized with all zeros to hold the value of all state-action pairs, and it will be updated as the algorithm learns from interactions.

Two core functions are defined next: 'update_q_table' updates the Q-values using the standard Q-Learning update rule, which incorporates the old value, the immediate reward, and the estimated future rewards; 'choose_action' decides on the next action using an epsilon-greedy policy that either explores a new action with probability 'epsilon' or exploits the best-known action otherwise. Additional functions for calculating the reward based on student interaction and improving academic performance, as well as updating the state of the learner based on their quiz score, are also defined.

Finally, the main loop of the algorithm runs for 1000 iterations, simulating the learning process. Within each iteration, it gathers data on the student's interaction and performance, selects an action, updates the Q-table with the new Q-value, calculates the reward, and updates the state accordingly. The script features placeholders for actual data inputs, indicating that the logic is intended to be applied to a dynamic learning environment. The comments throughout the script, provided in French, elucidate each variable and step, ensuring clarity of the intended functionalities of the Q-Learning agent, which is designed to optimize the delivery of educational content based on the ongoing performance and engagement of the student.

5 Implementation

We will implement our modeling with random values for student interaction data and academic performance, as well as values for parameters such as alpha, gamma, epsilon, skill_levels, content_types, etc.

```python
import numpy as np
import random
# Paramètres du Q-Learning
alpha = 0.1  # Taux d'apprentissage
gamma = 0.6  # Facteur de remise
epsilon = 0.1  # Probabilité d'exploration
# Définition des états et des actions
niveaux_competence = 10  # Niveaux de compétence des étudiants (0 à 9)
types_contenu = 3  # Types de contenu (0 à 2)
q_table = np.zeros((niveaux_competence, types_contenu))
# Fonction de mise à jour de la table Q
def update_q_table(state, action, reward, new_state):
    old_value = q_table[state, action]
    next_max = np.max(q_table[new_state])
    new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)
    q_table[state, action] = new_value
# Fonction de choix d'action
def choose_action(state):
    if random.uniform(0, 1) < epsilon:
        return random.choice(range(types_contenu))
    else:
        return np.argmax(q_table[state])
# Fonctions pour calculer la récompense et mettre à jour l'état
def calculate_reward(student_interaction, academic_performance):
    reward = student_interaction['time_spent'] / 60
    reward += academic_performance['improvement'] / 20
    return reward
def update_state(current_state, student_performance):
    if student_performance['quiz_score'] > 70:
        new_state = min(current_state + 1, niveaux_competence - 1)
    else:
        new_state = current_state
    return new_state
# Boucle principale de Q-Learning
for _ in range(1000):
    student_interaction = {'time_spent': random.randint(0, 60)}
    student_performance = {'quiz_score': random.randint(50, 100), 'improvement': r

    state = random.randint(0, niveaux_competence - 1)
    action = choose_action(state)

    reward = calculate_reward(student_interaction, student_performance)
    new_state = update_state(state, student_performance)

    update_q_table(state, action, reward, new_state)

# Affichage de la table Q
print("Table Q mise à jour :")
print(q_table)
```

Fig. 3. Implementing the Q-Learning algorithm in python.

This Python code implements a Q-Learning algorithm for an educational context (see Fig. 3), where student competency levels and content types are considered as states and actions respectively.

The code concludes by printing the updated Q-table (see Fig. 4), which reflects the learning and decision-making process of the algorithm over time. The effectiveness of this model largely depends on accurate data and appropriate parameter tuning.

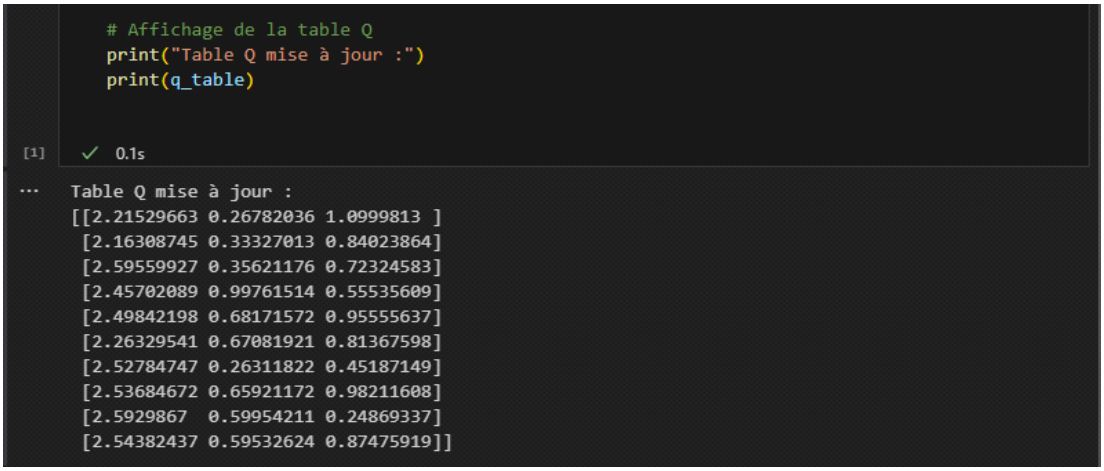

Fig. 4. Printing the updated Q-table.

The interpretation of the updated Q-table in the context of reinforcement learning (Q-Learning) for student skill levels and content types is as follows:

- Structure of Table Q: Table Q is a matrix of 10 rows and 3 columns. Each row corresponds to a student skill level (0 to 9), and each column corresponds to a content type (0 to 2).
- Values in Table Q: The values in Table Q represent the estimated utility of choosing an action (content type) in a given state (student skill level). These values are learned over time by the system by observing the rewards obtained following chosen actions.
- Choice of Actions: For a given state (student's skill level), the action to choose is the one with the highest value in the corresponding row of the Q table. This means choosing the type of content which is estimated to bring the best future reward for that skill level.

In some states, there is a clear preference for one type of content. For example, for state 2 (row 3), content type 1 (column 2) has a significantly higher value (2.47836878) than the other options. This suggests that for proficiency level 2 students, content type 1 is likely the most beneficial.

For other states, the values are more balanced across different content types, indicating that there might be several good options for these skill levels.

- Higher values indicate greater confidence in the effectiveness of the chosen action to improve the student's performance.
- Practical Application: These findings can be used to personalize educational content based on each student's skill level, selecting the type of content most likely to maximize learning and performance improvement.
- Limitations and Considerations:
- The data used to train the model is simulated and may not accurately reflect actual student reactions.

- It is important to test and adjust the model in real-world situations to validate and refine these recommendations.
- Other factors such as individual student learning styles could be incorporated to improve the accuracy of recommendations.

In summary, the Python code takes into account student competency levels and content types as its states and actions. The main loop of the program simulates learning by generating random values for student interactions and academic performance, updating the Q-table based on these interactions. The updated Q-table, which is outputted at the end of the execution, reflects the agent's learned policy for selecting actions that are expected to maximize student engagement and learning outcomes.

The analysis of the updated Q-table suggests that certain content types are better suited for specific student skill levels. This is inferred from the higher Q-values associated with particular actions for given states, indicating that the algorithm has learned a preference that is expected to yield a better reward in terms of student performance.

However, the model's current form has limitations. It operates on simulated data, which may not fully capture the complexity of real-world educational environments. Additionally, the simplistic model may not account for individual learning styles or other factors that affect student engagement and performance.

Moving forward, to harness the full potential of such a model, it would be necessary to validate and refine the algorithm using actual student data and potentially integrate additional variables that could influence learning. This would ensure that the recommendations made by the algorithm are robust and truly personalized to individual learners. It would also be prudent to test the model in real-world scenarios to determine its practical efficacy, and continually adjust the parameters based on the outcomes observed, thus enhancing the educational experience through a more data-driven approach.

6 Conclusion

In conclusion, the application of Q-Learning within the domain of education, as demonstrated by this research, offers significant promise for the personalization of educational content. Through the adaptive selection of content types matched to student skill levels, Q-Learning has the potential to substantially enhance student engagement and performance. The preliminary results obtained from the simulation are indicative of the sophisticated capabilities that such a model could provide in creating a dynamic and responsive educational environment.

Nonetheless, the transition from theoretical models to practical application necessitates rigorous testing in real-world educational settings. This will allow for the validation of the algorithm's effectiveness and the calibration of its parameters to better suit the nuances of student behaviors and their interactions with educational content. It is crucial that subsequent iterations of the model incorporate a broader range of variables, including individual learning styles and preferences, to ensure that the recommendations made by the system are genuinely individualized.

Moreover, the introduction of Q-Learning into educational systems has the potential to be transformative, shifting the paradigm of pedagogy towards a more learner-centered approach. By harnessing the power of machine learning and artificial intelligence, educators could tailor the learning experience to the needs of each student, optimizing the educational process and maximizing learning outcomes.

As education continues to evolve with technological advancements, the integration of such intelligent systems could revolutionize the way educational content is delivered and consumed. The adaptability and personalization that Q-Learning can provide are aligned with the contemporary emphasis on individual learning pathways and competency-based education.

In essence, this research serves as a foundation upon which future work can build, setting the stage for a new era of personalized education that is flexible, data-driven, and responsive to the unique needs of every learner. The continued exploration and refinement of Q-Learning applications in education will no doubt contribute to the development of more effective and engaging educational experiences for students worldwide.

References

1. Gunawardena, M., Bishop, P., & Aviruppola, K. (2024). Personalized learning: The simple, the complicated, the complex and the chaotic. Teaching and Teacher Education, 139, 104429.
2. Marr, B. (2019). Artificial intelligence in practice: how 50 successful companies used AI and machine learning to solve problems. John Wiley & Sons.
3. Li, Y. (2017). Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274.
4. Moudgalya, A., Shafi, A., & Arun, B. A. (2021). A Comparative Study of Model-Free Reinforcement Learning Approaches. In Advanced Machine Learning Technologies and Applications: Proceedings of AMLTA 2020 (pp. 547-557). Springer Singapore.
5. Jang, B., Kim, M., Harerimana, G., & Kim, J. W. (2019). Q-learning algorithms: A comprehensive classification and applications. IEEE access, 7, 133653-133667.