Cardiff University
School of Computer Science and Informatics
CM3203 One Semester Individual Project
40 Credits

Visualizing Deep Reinforcement Learning

Author
Tian Sou Zen Tan

Supervisor
Dr Jing Wu

Abstract

Reinforcement Learning is an area of machine learning in which an agent learns through experience to make decisions and take suitable actions to maximise long term rewards. Deep Q-Network (DQN) is a reinforcement learning model that is a variant of the Q-learning algorithm that utilises a Convolutional Neural Network (CNN). There has been a lack of visual analytics work targeted at deep reinforcement learning models in recent years. Therefore, this project proposes a visual analytics system that can aid reinforcement learning experts in examining and understanding the evolution of the agent's behaviour to potentially identify capabilities and diagnose problems in the decision-making process. This work will be focusing mainly on DQN and applying it to Atari environments that comprises of a simple action space. The visual analytics system will guide experts from an overview perspective down to an in-depth observation of these agents to extract useful patterns and behaviours. It will also feature experiments and evaluations of the tool from domain experts to obtain valuable feedback to demonstrate the effectiveness of the system.

Contents

Abstract 1
Introduction 3
Background 5
  Existing Solutions 5
  Technical Background 6
Specification & Design 9
  Requirement Analysis 9
  Approach 11
Implementation 13
  Tools 13
  Training & Data Collection 13
  Visualizations 17
  Interface 21
  Challenges 24
Results & Evaluation 26
  Case Study 26
  Expert Feedback 29
Future Work 34
Conclusion 35
Reflection 36
References 37

Introduction

Deep reinforcement learning (DRL) is the combination of both reinforcement learning and deep learning to allow for high dimensional problems using function approximation. Recently, much work has been targeted at improving deep reinforcement learning algorithms but analysing the learning process is still under-explored. These can include the understanding of the agent's predictions and choices or the reasons behind it failing in certain situations. Unlike supervised and unsupervised learning, interpreting the behaviour of these RL agents can be challenging as it requires generating a large number of experiences through a time-consuming training process. These experiences also include randomness as a means of exploring the unknown parts of the environment to potentially improve decision-making and maximising long-term rewards. Typically, when RL experts identify good or bad strategies learned by the agent, they are unable to incorporate such findings to the training for further examination. There are numerous ways of visualizing the behaviour of these agents, but data collected during the training process may vary according to the different RL algorithms applied therefore some visualization techniques may be more suitable than others. This has motivated me to developing my own visualization system and allowed me to learn about RL during the process which I have found to be an interesting direction.

This project introduces a visual analytics system which allows reinforcement learning experts to examine and understand the evolution of the agent's behaviour at different stages of training. These can allow RL experts to recognize the capabilities of the agents and hopefully diagnose problems in the decision-making process for better optimisations. The interface follows a hierarchical structure whereby RL experts can interactively navigate through a large-scale overview of the entire training process down to a more detailed observation of the episodes. Using these types of methods can help to breakdown large amounts of data by identifying patterns and major changes in results to filter out key elements to be examined further. As a secondary objective, the system also provides a slight insight for RL experts to compare agents learned by different RL models to better understand the reason behind these vast result differences and why some models are better suited for certain environments. To further improve and assess my prototype, RL experts have experimented the visual analytics system and provided feedback on some positive aspects as well as areas which can be improved.

One of the main objectives for my project was to train and collect my own data which required me to research on different RL algorithms and environments as the data collected would have an impact in the type of visualization technique constructed.

The environment that I have chosen to utilize my visualization is called Atari Breakout where the objective is to obtain the maximum score by breaking bricks using a ball and catching it with a paddle to avoid life loss. Learning of Atari 2600 games consists of using a reward-based system, images as input, and a set of possible actions.

The RL model I primarily focused on to train and collect my data is DQN which is a variant of the Q-learning algorithm developed by Google's DeepMind using a Convolutional Neural Network (CNN). This RL model was popular for achieving superhuman level performance in playing Atari 2600 games without modifications to the model architecture and using only the game pixels as an input. Another RL model that I have applied is A2C which is part of my

secondary objective to compare different models while incorporating similar visualization techniques.

The use of Stable Baselines3 in PyTorch, an open-source machine learning library, has allowed me to implement and modify these RL algorithms for training and data collection. Alongside SB3, I have also used a toolkit called Gym by OpenAI that provides a range of simulated environments in which RL algorithms can be incorporated for training agents.

D3.js is a JavaScript library which I have found to be suitable for producing these interactive data visualizations on a web browser hosted on GitHub. There are also visualization libraries available in python providing a simpler learning curve, however D3.js is more flexible and includes better features that suits my dynamic visualizations.

The structure of the report following the introduction is divided into 9 sections. The background section composes of two sections broken down into existing solutions and technical background. The next section is specifications and design which contains the requirement analysis, for composing requirements, and my approach to target these requirements. The third main section is implementation to go over the tools I have used, my training and data collection journey, the interactive charts developed, the overall interface of the system, and the challenges I have faced along the way. Results and evaluation consist of my case study such as the interesting visualizations I have found as well as expert feedback to show the overall usefulness of my system. The future work section is where it indicates me secondary objectives and desirable goals that are not completed due to time constraint.

Visual Analytics System: https://z3n-t4n.github.io/IndividualProject/website/dqn.html

Background

Existing Solutions

There are a few attempts made to develop visual analytics to help domain experts explore and interpret the memory and training process of agents. Some examples of these published solutions that I have focused on are DQNViz [1] and DRLViz [2]. I have chosen to concentrate on these research papers as they relate closely to my project in developing appropriate visualization techniques for Deep Reinforcement Learning. It will also help me to identify some important necessary features and insight into what RL experts require in general. For my visualization, I am instead interested in explaining how the different agents behaves at different stages of training.

DQNViz is a system that primarily focuses on Atari games as an environment with a simple action space. Their training and data collection consists of the agent's experience as well as the model losses and network parameters, which assesses the quality of the model and understands the agent's mind at different training stages. They have presented a visual analytics system that breaks the training process down into four levels of visualization to allow domain experts understand the experiences of a DQN agent. These include statistical charts of the overall training level, epoch level, episode level, and segment level.

The overall training level includes line charts, that reveal the trend of different summary statistics, and the stacked area charts demonstrate the distribution of actions and rewards over time. This view allows users to verify whether the training is progressing well and creates a basis to trigger the other views.

The epoch view contains a pie chart, showing the action and reward distribution of the current epoch, and a stacked bar chart presents the action and reward distribution of individual episode in the epoch. This view can help users quickly identify the specific episode that has the maximum number of steps or rewards.

The trajectory view is an overview that aims to fit all the steps in one epoch as well as reveal all the episode's actions and rewards of the epoch. This view has helped to identify and reveal different actions that are being repeated and the movement patterns of the agent. The design is also scalable and can synchronize other types of data along with the action data such as to help indicate random actions. This view also incorporated progress bars for the video clip pop up which they have found to be very useful to reflect the progress of static videos for when the agent has repeated actions.

Finally, there is the segment view which allows users to see what the agent sees and the ways it obtains such vision in the screen states from a trajectory segment. There are quite a few uses in this view for example, the use of filters in the convolutional layers can allow the ball to be extracted from the screen and enabling users to further examine the filters and states they have selected in the other views.

They have found that the agent gradually gets smarter further into training and always attempts to dig a tunnel through the bricks so that the ball can bounce in the top rows for more rewards. Another one of their findings includes the decreasing of repeating actions indicating that the agent has become more flexible in switching among actions. These show that tracking patterns is an effective method towards visualizing the evolution of the agent's behaviour.

They have conducted multiple case studies together with deep learning experts and demonstrated that the system can help to understand, diagnose, and potentially improve DQN models. For example, all the domain experts have confirmed the training was successful by basing off on the observations from the statistics view. Some other highlights from experts include finding abnormal epochs which led to further exploration into why these did not follow the general trend.

In conclusion for DQNViz, all the experts believed that the system is extremely useful in providing insights and has great potential in improving future deep RL models. They also stated that the most useful and time spent component was the trajectory view.

DRLViz on the other hand is also a visual analytics interface used to interpret the internal memory of an agent in deep reinforcement learning. Similarly, it assists experts by providing interactions to overview, filter, and select parts of the memory to investigate decisions and errors. Their focus is on navigation problems which operates on a partially observed environment and this defers from the type of environment I have chosen.

Their first visualization goal is an overview that consists of a memory timeline of the trained agent to check what the agent sees and its decisions using sliders as interactions. It also allows filtering of the timeline for something of interest and selecting elements whose activation behaviour is linked to the decisions.

The memory timeline is also a useful view that helps to expose the memory's internal structure for further analysis. The derived metrics view helps to link memory over time and decisions as well as filter a sub-set of the memory and these metrics support in finding interesting agent behaviours. Similar to DQNViz, a stacked area chart is also used to feature actions probabilities so that users can observe similar sequences of decisions.

They have also conducted frequent meetings with experts in RL and DRL and overall DRLViz received positive feedbacks stating their experience and interest in explaining the agent's behaviour as well as the ease of using the system.

Technical Background

Reinforcement Learning is the process whereby an agent learns by trial-and-error to find and take the best actions in an environment or state to maximise long term rewards. Compared to supervised and unsupervised learning, it utilises the agent and the environment to learn from experience which are dynamically generated over time. An agent is a program that sees the states and interacts with the environment to make decisions on actions that generates rewards whereas an environment is either a real-world or simulated problem to be solved. Model-free RL algorithms are sample inefficient so RL results vary from different runs and therefore having several runs can yield more accurate and quantitative results. Obtaining good results are also usually dependent on tuning appropriate hyperparameters.

Atari Breakout is an environment where the goal of the agent is to maximise rewards by hitting bricks with a ball and catching the ball with a paddle to prevent life loss. It takes in states as an input and outputs the Q-value for each action. With the help of Atari Learning Environment that was introduced in 2013, it allows researchers to easily simulate Atari games and evaluating algorithms.

Formulating RL problems correlates to the Markov Decision Process that consists of five components S, A, P, R, γ which is like the Markov Reward Process but includes an agent that takes actions. These are the states, actions, probability, reward, and discount factor.

The states can either be discrete or continuous and in the case of Atari Breakout, it is a finite discrete state space. The states are be represented as an image of the current frame which is an array of 84x84 values in the range of [0, 255].

The agent can take actions that can transition the environment into another state whereby it can receive a reward after each action. Breakout consists of four possible actions which are no-operation (noop), firing the ball (fire), moving right (right), and moving left (left) and these are represented as a value of 0, 1, 2, or 3. The actions are discrete as there are a fixed number of possibilities to represent the observation and action spaces in the environment. The rewards correspond to the change in score where positive rewards are received as the score increases. There is also a life system which has a value of 1 to 5 indicating the number of lives the agent has left in the current situation.

The probability of the state space is described in the transition function which determines the probability of settling in the different future environments given an action, P = S x A. In the case of Breakout, the number of possible states is $255^{4x210x160}$ which is large but finite.

The reward function R = S x A will provide the agent with a reward based on the action it has taken in a particular state and in the case of Breakout, the reward is based of the change in score.

There are two main methods of RL, namely policy-based methods, or value-based methods. In the value-based method, we only store the optimal value function based on action mapping and its corresponding value. The policy-based method does not use any value functions and instead builds a representation of the optimal policy that is mapped from a state of the agent to the action. Since the environment that I have chosen does not provide a vector of values to the agent, the Convolutional Neural Network (CNN) policy is used instead of the Multi-Layer Perceptron (MLP) policy.

The policies are outputs of RL algorithms that indicate what action to take at a given state and they can either be deterministic or stochastic. A deterministic environment will have no uncertainty and the predicted outcome is always determined based on the unique action. Stochastic on the other hand involves uncertainty with a probability of some randomness whereby you are unable to accurately predict the outcome. In MDP, the policy is always deterministic meaning it will never involve probabilities for deciding the action in any state.

Deep Q-Network (DQN) [3] was introduced in 2013 by Google's DeepMind which is a model-free off-policy algorithm that utilises deep Q-learning and composes of Convolutional Neural Networks (CNN). DQN uses an ε-greedy behaviour for the network's predicted Q-value to compute the loss and learn the optimal predictions. There are two Q-values per state-action pair and it updates the first Q-value by selecting an action that maximises over the second Q-value. Below is a screenshot of the DQN algorithm where it uses the experience replay function to train the model:

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Specification & Design

Requirement Analysis

The system requirements are derived through the perspective of RL experts whereby the features are implemented based on their most pursued fundamental requirements. These are mainly derived from relevant research papers whereby extensive interviews with RL experts has been conducted to obtain requirements that are valuable towards their study. I have summarised the most essential and requested requirements to be applied to my visualisation system.

One of the most important features is providing an overview of the entire training process as this would develop a great foundation for RL experts to see the evolution in the agent's behaviour and results. As the number of agents and data to analyse is large and can get very detailed, this overview can be seen as a primary filter to target specific points that expose major differences in patterns and behaviours as well as exposing anomalies.

Another popular requirement that experts seek is the ability to display the distributions of actions and rewards. They are interested in investigating whether the action distribution becomes stable in the later training stages and how it would affect the reward.

In addition to the overview, demonstrating the patterns of the agent is a strong need from the experts as they would like to efficiently detect and extract these patterns with the help of presenting other types of data to facilitate comprehensive reasoning. As they would be facing large number of experiences, the experts would also need some assistance and guide in pattern explorations.

One of the most common structures that domain experts seek is a hierarchical interactive view. They would want to be able to start exploring from an overview perspective down to a more detailed view through a series of simple interactions. As it has been observed that experts tend to go through a time consuming and inefficient process to analyse these results for example, they would manually add conditions to system prints to look for unusual values. Therefore, with this sought-out structure it would improve the experts experience and motivation towards discovering and diagnosing agent behaviours.

From the interviews, it is also claimed that most experts tend to manually record videos to playback trained agent's episodes and decisions to get a sense of their strategy and point of view. They find this to be quite useful towards reflecting the patterns and being able to look through the eyes of an agent to either envision the new strategy it learned or any unusual behaviours.

Typically, experts already have trained agents available, and they would want to investigate their decision-making without having to go through a time-consuming training process. This means that they would like to have a system that allows them to easily upload their trained agents while using the same visualization techniques for thorough comparison.

From the discussion with my supervisor, who is also an expert in RL, we have added a few requirements that helps to provide more visualizing techniques as well as enhancing the existing requirements. One of the requirements was to be able to transition from the overview to a more sophisticated view demonstrating a single agent at a particular timestep trained. This would allow experts to see the different types of variables displayed together for that

agent, for example to hopefully see the different scores it can achieve and the different factors that contributes to the score such as actions. Another requirement is a follow up to visualize the individual simulations to hopefully find patterns that can reveal the agent's learned strategy or unexpected failures.

Through iterative discussions and refinements with my supervisor on the requirements that I have obtained, I have established the following requirements for my visual analytics system:

Requirement 1
Providing and in-depth model statistics displaying an overview of the training process.
R1.1
Ability to select between different types of overview charts that targets different attributes of the agent.

Requirement 2
Demonstrating the probability distribution for the actions an agent takes.
R2.1
Providing a selection to individually target the different types of actions to display the probability distribution for that specific action.
R2.2
Displaying a combination of every action for all the agents at each training stage in one chart.

Requirement 3
Applying additional features in the overview charts.
R3.1
Ability to toggle additional features as an overlay to the overview charts.
R3.2
Include hover over effects on the data points to display more information.

Requirement 4
Providing more attributes and details on an agent of interest the experts have selected.
R4.1
Visualizing all the attributes and variables in one chart for an agent at a specific timestep.
R4.2
Experts can visualize all the different possible simulations an agent has attained in the same chart.

Requirement 5
Viewing a specific simulation of an agent.
R5.1
Ability to select the different simulations an agent has attained to visualize it in a chart.

Requirement 6
Presenting a video playback of the agent playing the game.
R6.1

Experts can select a specific simulation of the agent to retrieve the video playback.

Requirement 7
Interactively navigate through the system from an overview down to a detailed view.
R7.1
Option to select an agent that is of interest to the experts in the overview charts.
R7.2
Experts can select a specific simulation for the chosen agent.

Approach

Since the first requirement has stood out to be the most important among experts, I have found it ideal to include a range of overviews that target different types of results such as scores, rewards, and actions. These would all be displayed in separate charts to make it less convoluted and improve readability for the experts. This can also make room for additional features to be implemented within the same charts that have the same relatable attributes. The overview charts would all have similar positioning and axis layout to allow experts to easily compare results while selecting between charts.

I have decided to include a scores overview as this would allow for a broader study to target specific behaviours as the scores are generated differently compared to rewards. Each brick comprises of different scores, where bricks further up are worth more. This would target one of the sought-out requirements from experts in efficiently detecting patterns in more detail based on the type of bricks the agent targets.

The rewards overview on the other hand is also calculated every time a ball hits a brick, but the reward is consistent for every brick unlike scores. This can be more beneficial towards visualizing patterns of the agent failing or discovering new strategies by identifying how often the ball hits a brick given a specific number of steps.

For targeting the actions distribution requirement, I have decided to utilize an overview stacked area chart as this would show the full combination of all actions taken. This can help to answer the expert's question on whether the action distribution becomes more stable or how they evolve over time. Having the probability distribution for each action can allow more detailed observations from individual actions. Therefore, employing a 3D chart or separating the individual actions into multiple charts can accomplish this. I would also need to ensure that the colour code for each action is synchronised throughout with the stacked area chart for easy reference.

Having additional features like displaying filters, standard deviation of scores, and the length of simulations at each timestep can target the requirement three to facilitate more comprehensive reasoning. These additional features should also have an option to be toggled on or off to prevent cluttering and enables more features to be easily added in the future. Another important feature is a hover over function applied to the data points as this would provide the experts with information about the specific point such as the exact values.

For requirement four I have decided on the use of parallel coordinates as I found it is the most suitable for representing high dimensional data as a 2D visualization and this view would be a follow up from the overview. This can help experts in targeting a specific agent to analyse

its attributes and how they affect each other as well as exploiting the different scores an agent can achieve in that trained timestep. For example, they would be able to see whether the agent's achieved score increases according to the length of the simulation or if any of the actions are contributing.

Requirement five as discussed would be a follow up of the parallel coordinates chart. In this view it would instead consists of displaying individual simulations for experts to further examine the details of these episodes. The colour code for these plots should also synchronise with the ones in parallel coordinates as this would allow easy detection of the corresponding simulation. I have chosen to do an accumulated rewards chart that will represent data as lines so that experts will be able to easily detect patterns that correspond to the agent's strategy or unusual behaviours. This view would also include an option for experts to toggle between the different simulations available according to the agent's score.

Since most experts rely on video playbacks to see through the eyes of an agent, it would be useful to incorporate this technique into the final stages of the visualization. Requirement six will provide a familiar tool to assist with what they have been doing but also further enhancing the experience providing better details. This view should be a placeholder where the simulation playback can be retrieved and displayed based on the individual selection of simulation that the experts chose in the previous view. This can allow experts to simultaneously examine the patterns they have observed together with what the agent sees for comprehensive analysis.

The last requirement would allow experts to seamlessly navigate through the whole system step-by-step, so it is important to have similar positionings for the interactions. To make it feel more natural, the interactions should be integrated within the charts where experts will be able to click and select their points of interest as well as hover over them. There should also be navigation instructions provided as well as indicators that shows the experts what they have selected or interacted with. For example, I simple change in the colour of data points can suggest to the expert they have selected that agent.

For the overall design of the system, it should be presented in a top-down approach where the overview would be the highlight of the interface while hiding other visualizations and hinting experts to select an agent for further details. There should also be navigation links at the top to move to a visualization of another RL model for the secondary objective.

Implementation

Tools

One of the main tools that has aided my project is Stable Baselines3 [4] which is an open-source implementation for important reinforcement learning algorithms that requires the use of Python and PyTorch. The algorithms provide an easy and consistent interface for a simple process to train and compare the different RL algorithms. It also includes numerous documentations which can allow users to replicate and refine these algorithms making it very flexible to suit their needs. More importantly I have chosen this tool as it allows beginners in RL like myself to experiment and learn without being overcomplicated by implementation details and dependencies.

SB3 allows users to easily incorporate OpenAI Gym [5] if the environments and policies are registered. The documentation provides many useful tips and information about RL experiments such as the type of RL algorithms it recommends depending on the action space or tutorials on how to evaluate a RL algorithm to extract important results for further examination. It also provides some basic examples to demonstrate the use of its library and functions which can be helpful for users to test and ensure the system works as intended.

Focusing on my project, SB3 includes a helper function called make_atari_env makes training RL agent on Atari games straightforward by delivering all the pre-processing and multiprocessing. SB3 also enables users to train their agent on multiple environments at the same time as well as including simple ways for users to tune parameters. The main algorithm I will be focusing on is DQN which on SB3 is built on Fitted-Q-Iteration (FQI) that makes use of different tricks to stabilise the learning with neural networks such as using a replay buffer, target network and gradient clipping. SB3 also supports multiple policies for their algorithms such as CnnPolicy which would allow me to use images as input.

I have chosen to install these on my Windows PC as PyTorch supports NVIDIA CUDA which allows software to make use of GPUs for faster processing. This would be very helpful towards reducing the overall time spent on training agents as it is a very time-consuming procedure with users usually taking weeks or even months to complete their training on less capable machines. Atari-py is a package I have installed that can be integrated into SB3 to allow training on Atari games specifically the Breakout Environment I have chosen.

Another important tool that I have used to aid development of my visual analytics system is D3.js [6] which is a JavaScript library to create dynamic and interactive data visualizations. This framework has provided me with the freedom to manipulate the code and design charts whichever way that suits my needs and data. I found it to have quite a steep learning curve however once I have grasped the basics and understanding it was very useful and flexible. It also provides multiple data loading methods and types such as CSV, JSON, TSV, and XML which allows me to store the data collected however I find suitable. D3.js also supports many types of charts from simple line charts to more advanced interactive and animated charts with documentation providing basic implementation examples and instructions for advanced features.

Training & Data Collection

To start training Atari games, I had to first install a more advanced version of SB3 and Gym as the standard versions did not include the Atari Breakout environment. With SB3 I can

easily initialise the environment using make_atari_env as well as setting no frame skipping and v4 indicating the probability of actions repeating is 0 meaning it will always follow the issued new action instead of using the previous action. Originally Atari Breakout outputs 210x160 RGB arrays therefore make_atari_env helps to provide pre-processing of the images to obtain an 84x84 image to reduce memory usage while keeping important information as well as converting them to grayscale. VecFrameStack is a frame stacking wrapper for vectorised environments that is specifically designed for image observations and this is useful for my Breakout environment as it uses images as input. The agent needs information such as the velocity and direction of the ball which is impossible using only a single frame for these environments that includes movement. Therefore, setting the number of stacks to four would allow the agent to learn the movement of the ball and taking actions based on the last fourth frame. This can however cause some confusion as this would mean that one timestep would be equivalent to four frames. Below is a snippet of code I used to initialise my environment:

```
env = make_atari_env("BreakoutNoFrameskip-v4")
env = VecFrameStack(env, n_stack=4) # Give the agent frame-stacking as input to learn movement of the ball
```

To begin training my agent, SB3 provides a simple structure for users to easily change and try out different RL algorithms just by changing the name of the algorithm in a line of code. The policy which I have chosen is CnnPolicy as this is suitable for my Breakout environment using images as an input. The parameters are hidden and contains default parameters however tuning these hyperparameters can easily be done using the documentation as reference. I have reduced the buffer size and enabled memory usage optimisation mainly because of memory issues with the replay buffer. Some of the other hyperparameters have been tuned based on RL Zoo repository which is a training framework that also provides a collection of optimal hyperparameters. For the overall training of my agent, I have set it to train a total of 5 million timesteps including a callback function. A checkpoint callback allows for saving a model at every set number of timesteps which would be useful to my visualization of different agent behaviours at different training timesteps. After training my agent using the DQN algorithm, I continued to train another agent using A2C algorithm as this would allow me to save time for targeting my secondary objective of using other RL algorithms. Training of A2C consists of the same structure as DQN but with different hyperparameters set. With the help of my GPU, the training time was reduced significantly, and it only took me about 2 days to fully train my agent for 5 million timesteps. However, because of issues such as incorrect parameters and non-increasing rewards due to errors, the time took to properly train my agent was far more as I needed to retrain quite a few times. In previous trainings, I have set the total number of timesteps to be 10 million, however I quickly noticed that after a few million timesteps, the behaviour was not changing much therefore I have found it more beneficial to have less total number of timesteps with a higher callback frequency to capture important details and patterns in the early stages of training. Through experimenting the callback frequency, I have noticed that reducing the callback frequency from 100,000 to 50,000 is a good interval for optimal visualization. Below is a piece of code that showcases the implementation for training my agent:

```
checkpoint_callback = CheckpointCallback(save_freq=int(5e4), save_path="./logs/") # Save a checkpoint every 50,000 steps

model = DQN("CnnPolicy", env, learning_rate=0.0001, buffer_size=100000, learning_starts=100000, batch_size=32, train_freq=4,
gradient_steps=1, optimize_memory_usage=True, target_update_interval=1000, exploration_fraction=0.1, exploration_final_eps=0.05,
verbose=1) # Instantiate the agent
model.learn(total_timesteps=int(5e6), callback=checkpoint_callback) # Train the agent
model.save("breakout_dqn_1e7") # Save the agent
```

Once training was fully completed, I needed to extract and collect data for these trained agents and ensure data collected is in good structure for easy accessibility in visualization. In this environment, the agent has four possible actions which are no-operation (noop), firing the ball (fire), moving right (right), and moving left (left). The agent receives a score based on the type of brick it breaks where bricks further up are worth more points. The agent also carries five lives in which the game simulation will end when all lives are lost. While the agent has lives remaining, the game simulation is endless whereby if all visible bricks are broken, a new set of bricks will be newly formed, and the game continues bringing the scores forward. One episode in the simulation is represented as one life therefore the total number of episodes in one simulation would be five.

To obtain the scores, SB3 has provided an evaluation function that returns the scores and lengths of each game simulation. To get a more accurate representation of the score, I have run the simulation 10 times and calculated the mean score. This would reduce the chances of getting abnormal results by having more common runs being integrated in the pool of scores. To get an estimated metric on the types of scores and to not disregard the highs and lows, I have also calculated the standard deviation for the scores as this would allow experts to see how large the gap is between the different scores an agent achieves depending on its play style. Like the score, I have calculated the mean for the total length or number of steps an agent would require for completing a game simulation. Deterministic means that when given a current state and action, the agent will be able to predict exactly the next state of the environment whereas stochastic would involve some form of randomness as it uses probability to maximise the performance instead. In this evaluation, deterministic is set to true as Atari games have no entropy source meaning they are deterministic environments. These data for all the 100 agents are then exported and stored as a JSON file to be applied in the score overview chart. Below is a bit of code that demonstrates the evaluation function of SB3 being applied:

```
# n_eval_episodes is number of game simulations instead of per episode
# Evaluate the loaded policy
score_list, steps_list = evaluate_policy(model, env, n_eval_episodes=10, render=True, return_episode_rewards=True, deterministic=True)
# print(f"mean_reward={mean_score:.2f} +/- {std_score}")
mean_score = sum(score_list) / len(score_list)
std_score = np.std(score_list)
mean_length = sum(steps_list) / len(steps_list)
```

Collecting the rewards data was straightforward when running the trained agent in each epoch. I have chosen to set each epoch to be 15000 iterations instead of having a set number of game simulations or episodes for more flexibility. This was to expose and exaggerate behaviours and patterns that would not be visible in one simulation. The reward system was also meant to provide another alternative to score as the scores would reset after each simulation whereas I am able to stack rewards indefinitely through multiple simulations. The predict function in DQN overrides the base class predict function to include epsilon-greedy exploration which returns the action that will be passed in the step function. The step function in SB3 allows to step the environment with the given action and in return output one of the

variables that I require called reward with values being either 0 or 1 depending on whether the agent hits a brick in that iteration. Like the scores these are also stored in a JSON file to be used later. Below is a snippet of the code that demonstrates what was explained:

```
for j in range(15000):
    action, _states = model.predict(obs, deterministic=True)
    obs, reward, done, info = env.step(action)
    epoch_reward += int(reward)
    env.render()
env.close()
```

Finally, the actions data would need to be collected to represent them in the overview charts. Like the scores, the total number of episodes to be ran is 50 which equates to 10 game simulations for a more accurate representation. Using the predict function in DQN, I can extract the predicted action an agent takes at each step which returns a value of 0, 1, 2, or 3 to represent noop, fire, right, and left. This would then allow me to collect all the actions for each timestep and calculating the probability distribution for these actions. This data would be stored as a JSON file with their corresponding agent and timestep. Below is a code extract for collected and calculating the probability distribution:

```
episodes = 50
for j in range(episodes):
    obs = env.reset()
    done = False
    while done == False:
        action, _states = model.predict(obs, deterministic=True)
        obs, reward, done, info = env.step(action)
        action_list.append(action)
        env.render()

noop_prob = action_list.count(0) / len(action_list)
fire_prob = action_list.count(1) / len(action_list)
right_prob = action_list.count(2) / len(action_list)
left_prob = action_list.count(3) / len(action_list)
```

Collecting data for parallel coordinates would require a rerun as attributes for the parallel coordinates must all be from the same agent in the same game simulation for accurate results. As the evaluation function does not come with actions, I would need to improvise existing code to extract the scores and lengths which is returned as a dictionary containing all the external information. I have also set to run it for 30 game simulations as this would allow me to see more of the different types of scores the agent is able to achieve. Below showcases the collection of scores, lengths, and actions all in one dataset for each timestep:

```
    while done == False:
        action, _states = model.predict(obs, deterministic=True)
        obs, reward, done, info = env.step(action)
        actions.append(action)
        env.render()
env.close()

# extract episode information
episode_info = (info[0]["episode"])
scores.append(list(episode_info.values())[0])
lengths.append(list(episode_info.values())[1])

noop_prob = actions.count(0) / len(actions)
fire_prob = actions.count(1) / len(actions)
right_prob = actions.count(2) / len(actions)
left_prob = actions.count(3) / len(actions)
```

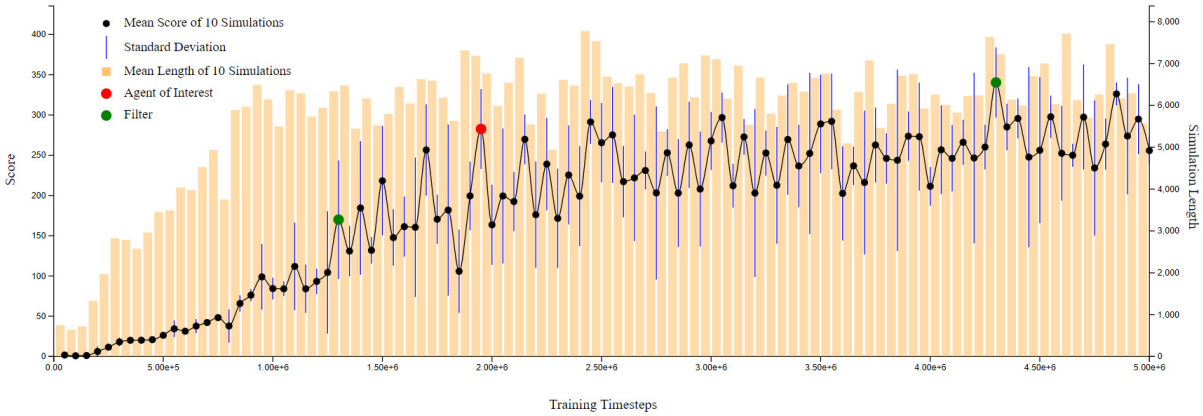
The accumulated rewards data is like the rewards data in the overview. The main difference is the method used to store the data as instead of having a total reward per timestep, the incrementing reward would be linked with its corresponding iteration for one agent and timestep as shown below. This would allow experts to see how frequently the agent breaks a brick to hopefully find patterns of strategies.

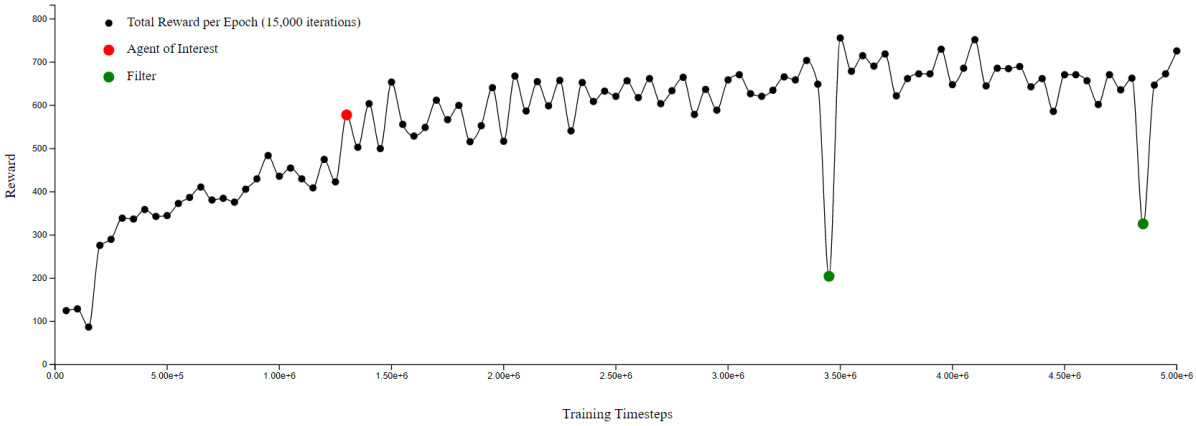
```
while done == False:
    action, _states = model.predict(obs, deterministic=True)
    obs, reward, done, info = env.step(action)
    count += 1
    set_rewards += reward
    reward_freq.append({"reward": int(set_rewards), "iteration": count})
    env.render()
```

Visualizations

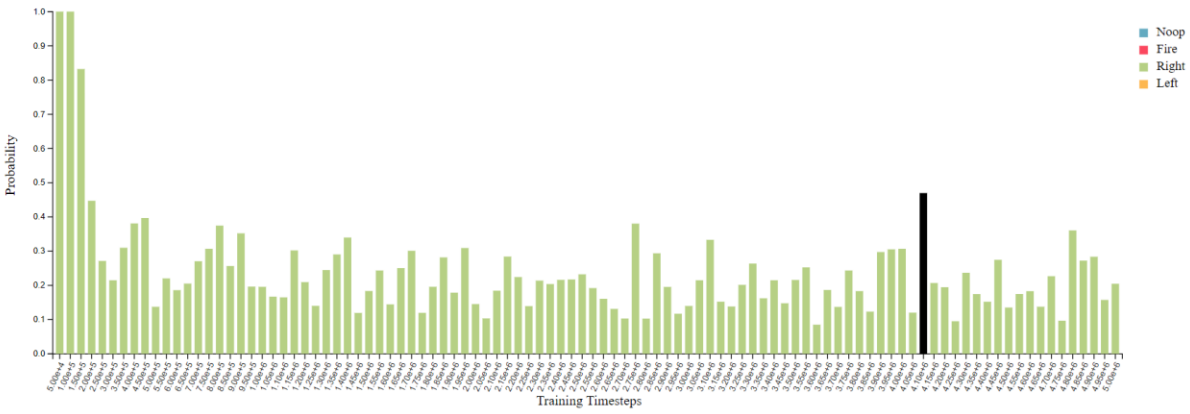
With D3.js I was able to develop quite a few interactive overview charts. All the overview charts follow a consistent design pattern with sufficient information provided such as legends to indicate what the different objects in the charts represent.

For the score and reward chart, I have decided to use a scatter line plot as I found this to be most suitable for pattern identification for a large dataset of experience. The length for each timestep is also displayed in parallel with the data points for easy reference indicating the mean number of steps the agent took to complete the simulation. The standard deviation lines are also displayed in every data point to show the experts the possible scores an agent can achieve for that specific timestep and how they evolve over time. The red points are agents of interest which helps to signal to the user that they have selected an agent for further examination. The green points are filters which I have obtained from using a function to filter out interesting data points. For example, in the score chart, filtering out an increase in the mean score of at least 50 within the next iteration of 50,000 timesteps or also filtering out the highest mean score achieved overall. For the reward chart on the other hand filtering out a drop in reward of at least 300 within the next iteration of 50,000 timesteps. Below showcases an example of the score and reward charts:

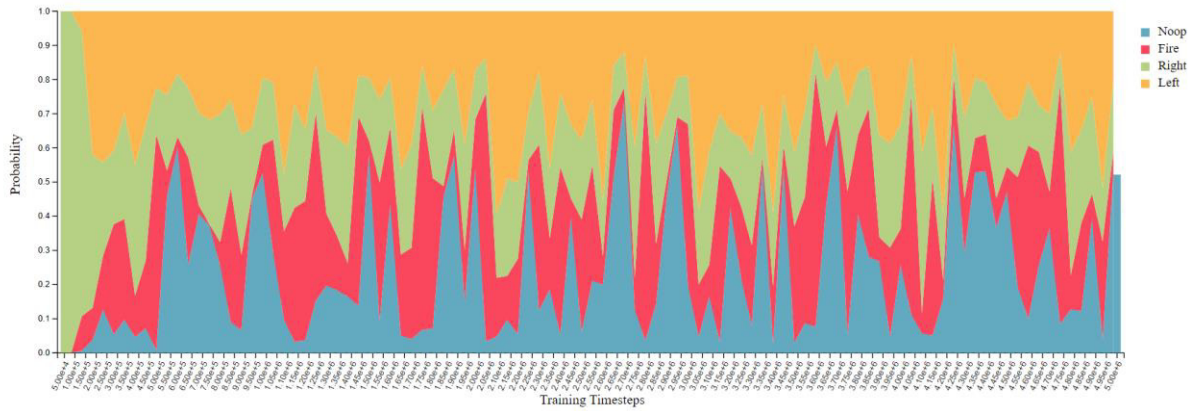




I have used a bar chart for the actions as this is suitable to show how high or low the probability of the specific actions being taken. I have also used a colour scheme to differentiate the types of actions an agent takes, and this information is also displayed on the legend. The black bars in this action chart would be the agents of interest the user has selected for further examination. Below is an example of the action chart displaying the probability distribution for right actions:

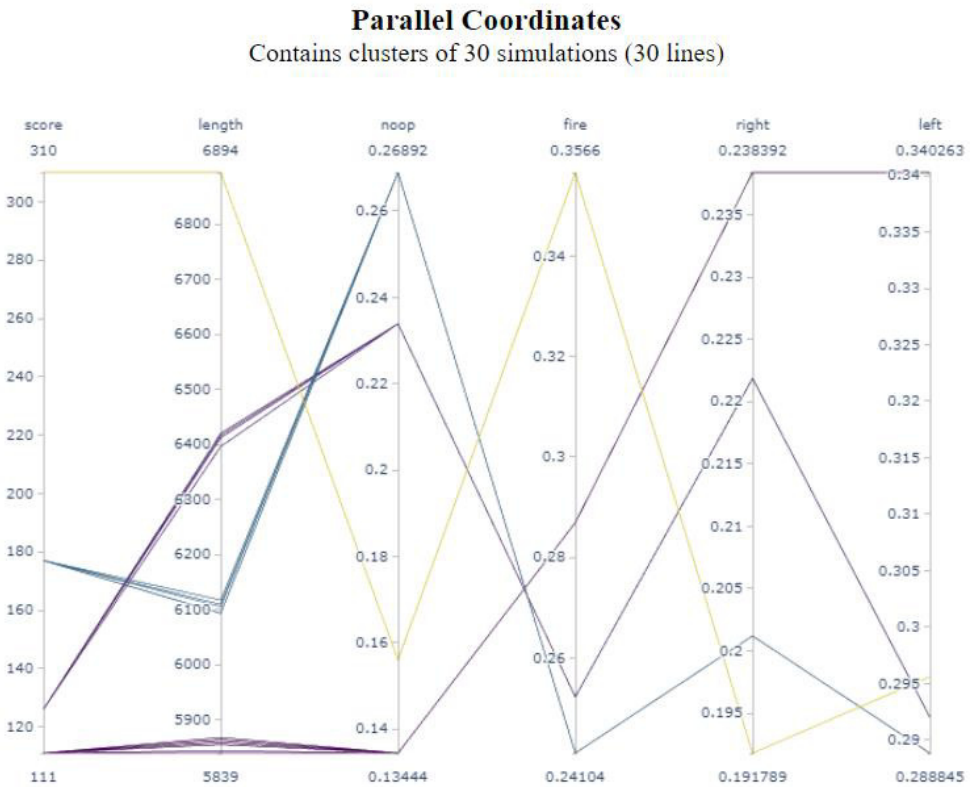


Using D3.js I have also developed a stacked area chart using the same colour code for the individual actions to demonstrate how these actions evolve over time shown below:

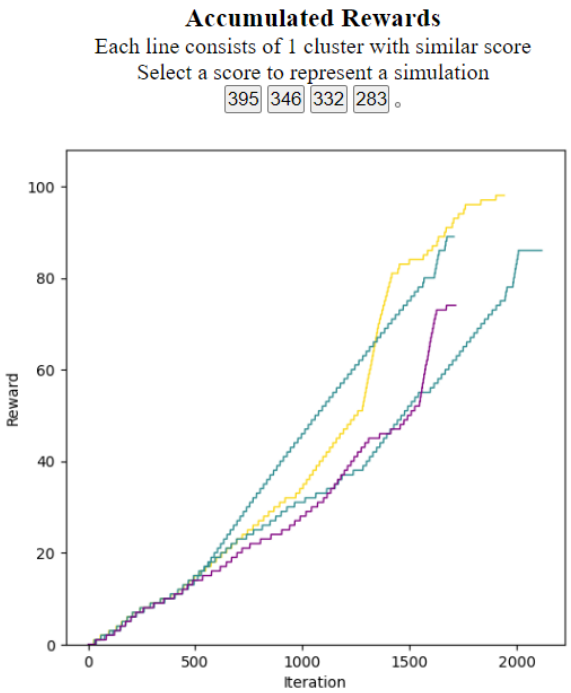


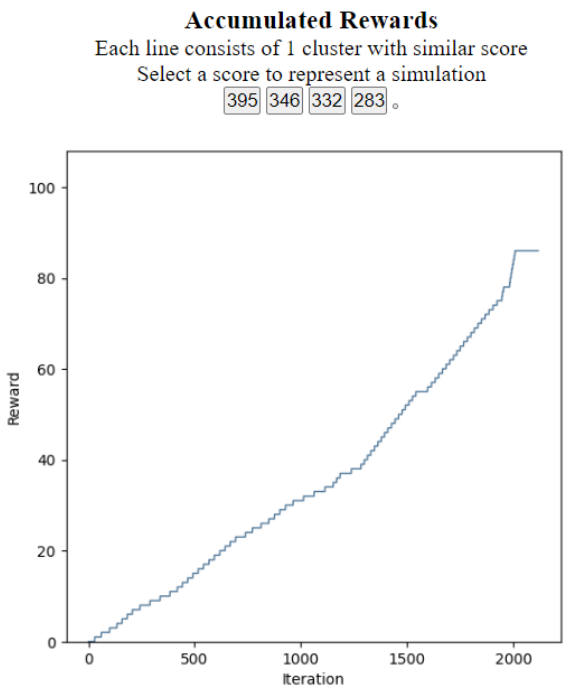
Experimenting with the parallel coordinates data, I have noticed that the data lines are usually represented in clusters meaning that each cluster would have the same score. This would be difficult for experts to individually select the data lines as they would be cluttered together. Therefore, I have decided that it is irrelevant to develop this chart using D3.js as it would not be interactive. Using a python library called plotly would allow me to easily create parallel coordinates chart which can be pre-computed and loaded into the system. I would also need to ensure that the upper and lower bounds for each attribute are precise depending on the dataset's highest and lowest value as this would enlarge the clusters for better visualization showing the anomalies. Below is an example of how I utilise plotly to create parallel coordinates and an example a chart generated:

```
fig = go.Figure(data=
    go.Parcoords(
        line = dict(color = df["score"]),
        dimensions = list([
            dict(range = [score[0],score[-1]],
                label = "score", values = df["score"]),
            dict(range = [length[0],length[-1]],
                label = "length", values = df["length"]),
            dict(range = [noop[0],noop[-1]],
                label = "noop", values = df["noop"]),
            dict(range = [fire[0],fire[-1]],
                label = "fire", values = df["fire"]),
            dict(range = [right[0],right[-1]],
                label = "right", values = df["right"]),
            dict(range = [left[0],left[-1]],
                label = "left", values = df["left"])
        ])
    ),
    layout = layout
)
```



To display the accumulated rewards charts, I have used matplotlib in python, again due to not needing any interactions within the chart itself. I have also ensured that the colour for each simulation represents the colour of the cluster with the same score for easy reference. I will also be using buttons in the accumulated rewards section that can allow experts to target specific clusters which has similar scores which would update the accumulated rewards chart as well as the simulation video playback. Below is an example of the accumulated rewards chart showing all the simulations together or individually:





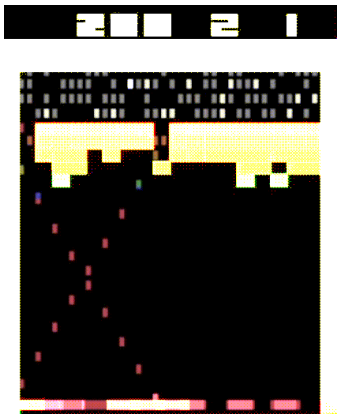
To create the video playback, I have used a video recording software called OBS and a tool called ffmpeg to record and edit these recordings into gifs so that they be pre-computed and retrieved in the system on demand. This would reduce the time spent waiting as computing these data, especially the simulation takes a long time. Below showcases what the simulation would look like with the corresponding score displayed for reference:

Simulation

Playback Score 310

As showing gifs would be impossible in my report, I have developed a method called frame stacking where I can stack frames to show the movement of the ball. This can be useful to display new strategies or anomalies in a static image. Below is a snippet of the code and example of a frame stacked image:

```
for j in range(len(frame_list)):
    background = cv2.imread("./data/filter/" + i + "/frame.png")
    next_frame = cv2.imread(frame_list[j])
    # blend images together
    overlap = cv2.addWeighted(background, 1, next_frame, 1, 0)
    cv2.imwrite("./data/filter/" + i + "/frame.png", overlap)
```

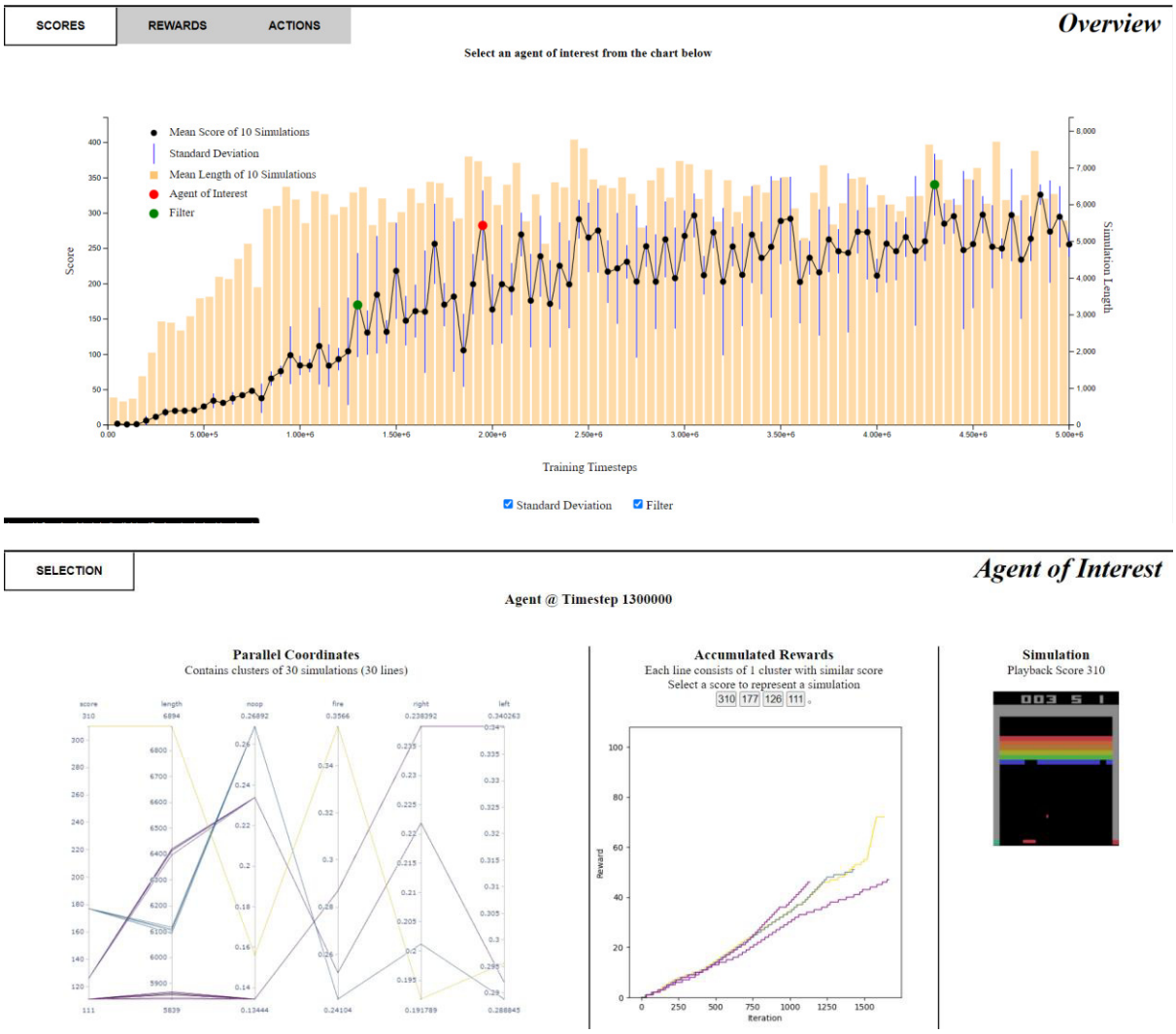


Interface

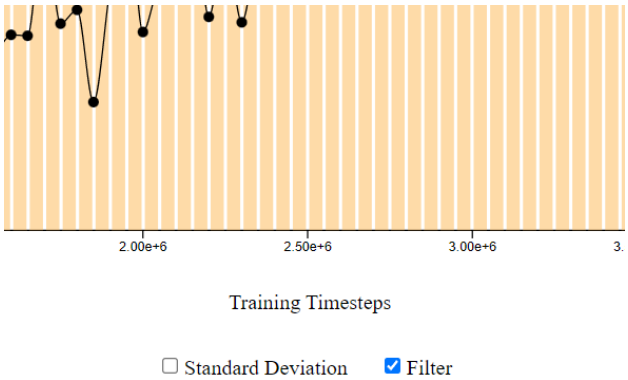
For my visual analytics system, I have decided to host it as a web page on GitHub which supports the integration of JavaScript, HTML, and CSS, therefore making it suitable for my needs. GitHub also provides a place to store my data as well as images to be easily accessed all in one system.

At the very top of the interface, users will be able to select between DQN or A2C visualization as shown below:

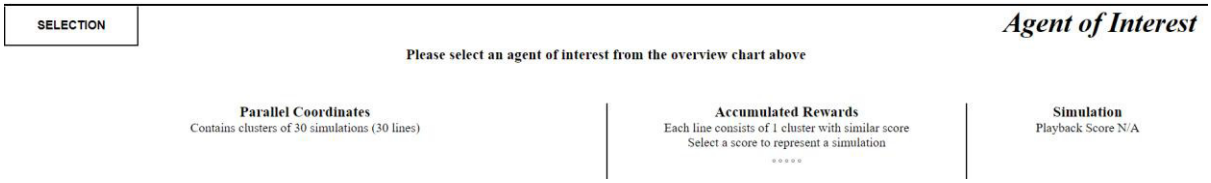
The main focus would be the overview section where users can choose between the tabs to display the different charts for scores, rewards, and actions. The overview chart incorporates a consistent layout such as the axis and positioning of labels for easy navigation. There are also interactions in place like checkboxes to toggle the additional features. The overall procedure for this section allows users to select an agent or data point from this overview chart which would then be highlighted in red indicating their agent of interest. This would then update the agent of interest section to display further information about the agent at the specific timestep. In the agent of interest section, users would then be greeted with the parallel coordinates, accumulated rewards, and simulation information. Experts will be able to select individual simulations to display the updated accumulated rewards chart and video playback of their choice. With the parallel coordinates, accumulated rewards, and simulation all in one row, experts can compare and refer to the data lines easily.



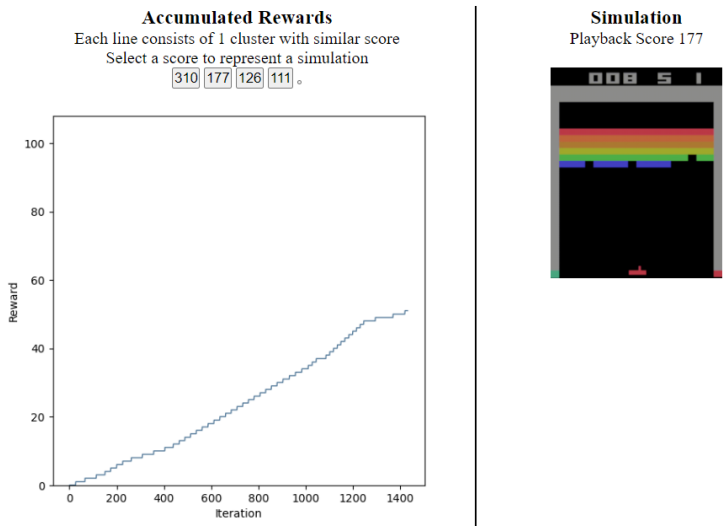
Below showcases an example of the checkbox interactions to toggle off the standard deviation feature so that experts can have a clearer visualization if more features were to be added:



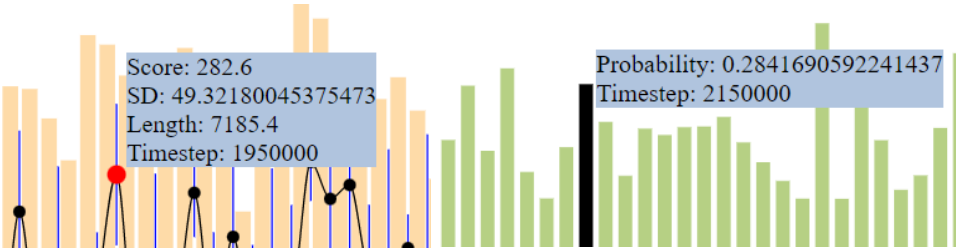
Below showcases what would happen if the user does not select an agent of interest. The agent of interest section would contain no charts and interactions with navigation guidance provided to indicate to the experts that an agent must be selected from the overview chart:



Below showcases the change in accumulated rewards chart and simulation video playback to the correspond the score button selected which in this example is 177:



With assistance from D3.js I was able to create hover over effects using tooltips, for the individual data points to display their specific values and details. This would provide experts with more information especially in the overview charts as the datasets are usually very large. Below are some examples of what they look like:



With the structure of implementing the visualizations in D3.js, changing or applying new datasets is made simple. This would especially make it easy to apply the same visualization on different RL algorithms for comparison. For example, as shown below, it would only require inserting a new url containing the new or updated dataset without having to modify any of the visualization code:

```
// data for dqn scores
var url = "https://z3n-t4n.github.io/IndividualProject/dqn/scores.json";

d3.json(url).then( data => {
  var xScale0 =  d3
    .scaleLinear()
    .range([0, width - spacing])
    .domain([0, d3.max(data, function(d){ return d.timestep; })]);
```

Challenges

There were quite a few problems at the start of the project when installing the necessary tools and libraries due to errors from the compatibility of dependencies for example with PyTorch. Atari-py is meant to be suitable for Linux and Mac but however does not support Windows and this was a problem as I wanted to utilise my GPU and NVIDIA CUDA for faster performance in training. To overcome this, I had to search for alternative ports of Atari-py on GitHub for Windows.

Another major problem that forced me to find other alternative methods was during the development of the actions overview using D3.js. One of my original proposals was to generate the probabilities of actions taken in a 3D interactive chart as this would allow experts to visualize it clearer with all the information in a single chart. However, this was a limitation of D3.js as the library is oriented more towards data visualization rather than scientific visualization which meant that it did not have extensive support for displaying charts in 3D space other than for geographic 3D data which is irrelevant to my dataset.

In the parallel coordinates chart, my original plan was to allow domain experts to individually select the game simulation data lines or at least select the cluster of data lines that achieved the same score. However, these were limited in the Python library plotly, as it only contains very few interactions which were not relevant to my objectives. I had to overcome this challenge by finding another method of visualization accessibility such as making use of a simple button to be generated according to the types of scores achieved. I have also thought about using colours instead of scores but due to the colour gradient of plotly, different scores close to each other may have very similar colours and could be misleading to experts.

One of my objectives in visualization was to retrieve the positions of the ball and the paddle in the game simulation. This was however limited in SB3, which I heavily relied on, as the

documentation did not provide any way for me to accomplish this which meant that I had to think of alternatives such as image processing. Using image processing was the next best solution to obtaining the coordinates of the ball and paddle, but this itself also came with inevitable challenges. The colour scheme of the simulation was mainly the issue as the colours of the ball and paddle were the same as the other objects. In addition, the colour of the ball also constantly changes as it travels. This meant that image processing was either too difficult, if I were to extract it using object recognition, or impossible for this solution.

Creating the video playback for experts to visualize the simulation also posed a great challenge again due to limitations of SB3. SB3 by default supports easy recording and conversion of these simulations but lacks the support specifically for Atari games. This has brought me to using alternative video recording software such as OBS and tools like ffmpeg for conversion and editing which was a very time-consuming manual process. I was also unable to find a video recording software that was advanced enough to automate the process of recording every time a new simulation starts and stop whenever it ends.

One of the last major problems that I faced was from the web hosting platform GitHub as there were upload and storage limitations meaning that I was unable to upload all my video playbacks. This meant that I could only show all the video playbacks for certain important agents instead of having all the possible playbacks for all the 100 agents.

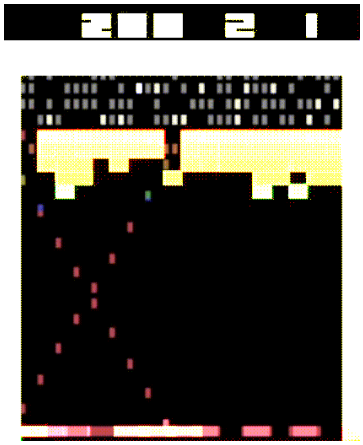
Results & Evaluation

Case Study

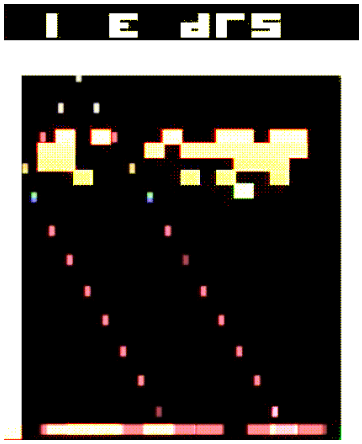
Through the course of development, I have learned much about RL and discovered many interesting results through the process.

When developing the overview charts, I have noticed that the score view has beneficial aspects compared to the rewards view. For example, the score view serves as a raw in-game point system which can potentially be used in comparing different RL models such as demonstrating the highest score a model is able to achieve. It is also able to provide insight into how an agent would learn to target bricks that are worth more points. The reward system on the other hand can also be useful for running a predefined set of iterations while stacking the rewards indefinitely whereas the scores are only limited to five lives each simulation and resets every time it ends.

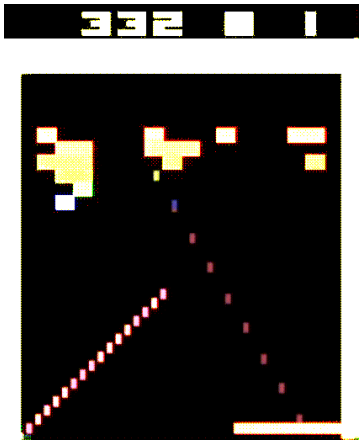
With these visualizations, I have noticed many new aspects and issues with the agent. One of them is the learning of a new strategy where the agent learns to dig through a tunnel to the roof so that the ball can bounce between the top row bricks for more points. I have also noticed that after it has applied this strategy, it fails to catch the ball falling back down. This led to it learning a new method of first hitting the lower blocks as much as possible before using the tunnel strategy to obtain maximum points before it dies. An example of this strategy is shown below:



Another point I have noticed through the course of training, is the agent being stuck in an infinite loop where it bounces the ball between the same corners while refusing to lose any life or obtaining any reward. With some further research I have also found a research paper [7] that encountered this issue calling it the “local stuck”. They have concluded that this situation occurs due to the lack of exploration. It explains that during the early stages of training, there would always be a reward when the agent catches a ball. This is because in the early stages, it is impossible to miss a brick and not get rewarded as the bricks are still new and fully formed on all sides. This has led to the agent focusing on just catching the ball to prevent life loss which would in turn lead to a bad learning outcome in the later training stages. Below is a frame stacked image showing the agent being stuck in a loop:

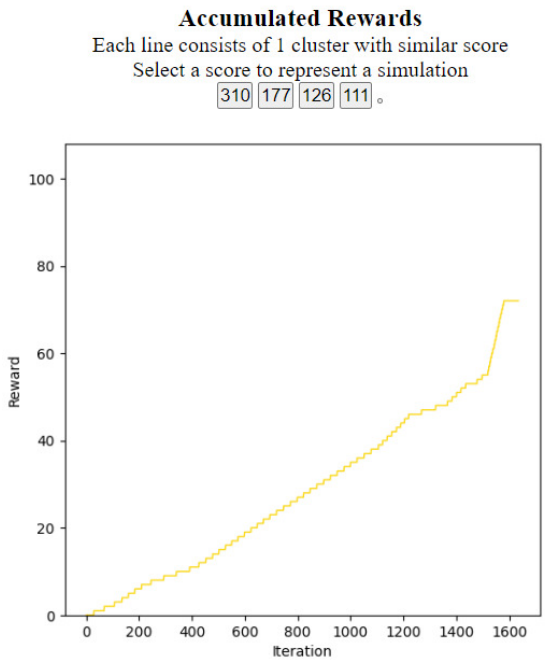


Finally, the last illustration below depicts the agent failing to catch the ball. I have noticed this behaviour in the later stages of training where the agent refuses to catch the ball consistently, losing all lives immediately. Below is an image showing the agent moving to the right instead of left to catch the falling ball:

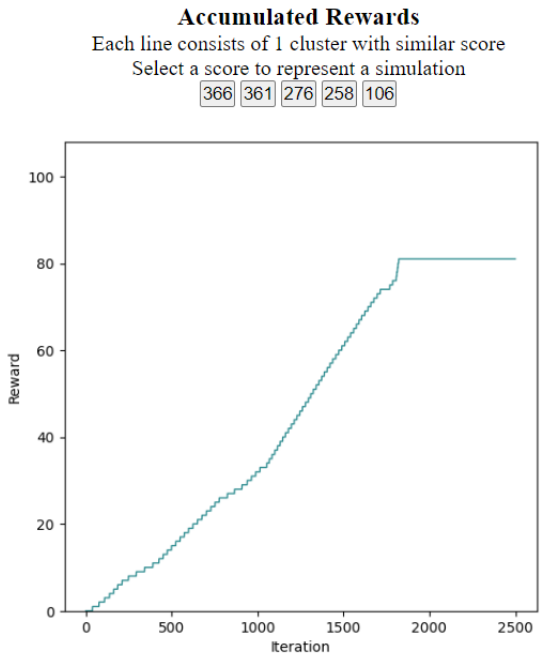


I have also noticed when retraining agents, some do not encounter these situations at all which means that the training outcome can vary drastically based on what they have learned in the early stages. Another example is instead of the agent starting with having a probability of 1 in the right action, they would have it as the left action instead.

The accumulated rewards view was very useful to me in seeing the agent's behaviour which assisted me in finding these interesting points. For example, the learning of the new strategy as shown below is easily detectable by the sudden rise in rewards over a short period of time. With this information, I have noticed that further down the training stages, the agent has learned to use these strategies more often.



It also allows me to identify the agent being stuck due to the non-incrementing reward as shown below:

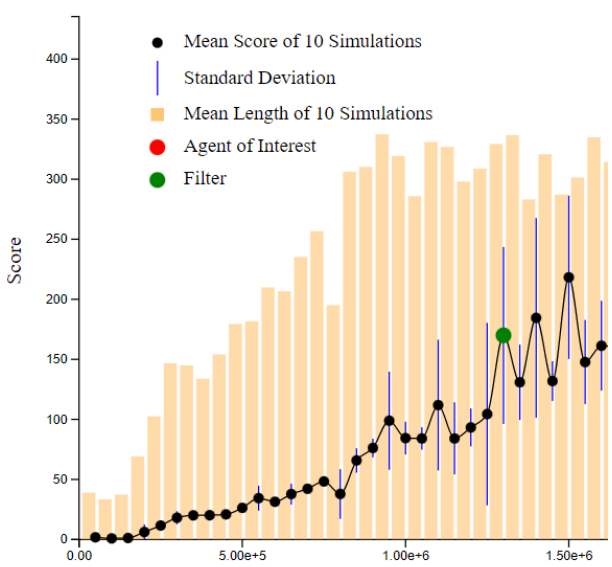


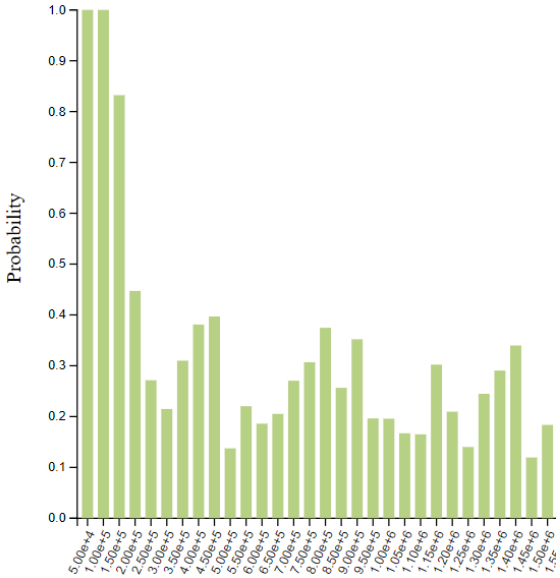
With the parallel coordinates view, one aspect caught my attention where I noticed that the data lines were in clusters with the scores being the same. I believe this is due to the deterministic nature meaning that the agent's prediction of actions is always accurate and based on what they have learned.

There was a rare occurrence in one retraining where the agent completely clears all the bricks and therefore new bricks were formed. With this I have noticed the agent's behaviour suddenly changes where it loses what it has learned and refuses to catch the ball in this newly formed environment. I believe this is because the agent has not learned what to do in situations like this as when new bricks are formed, the score it obtains would also be lower

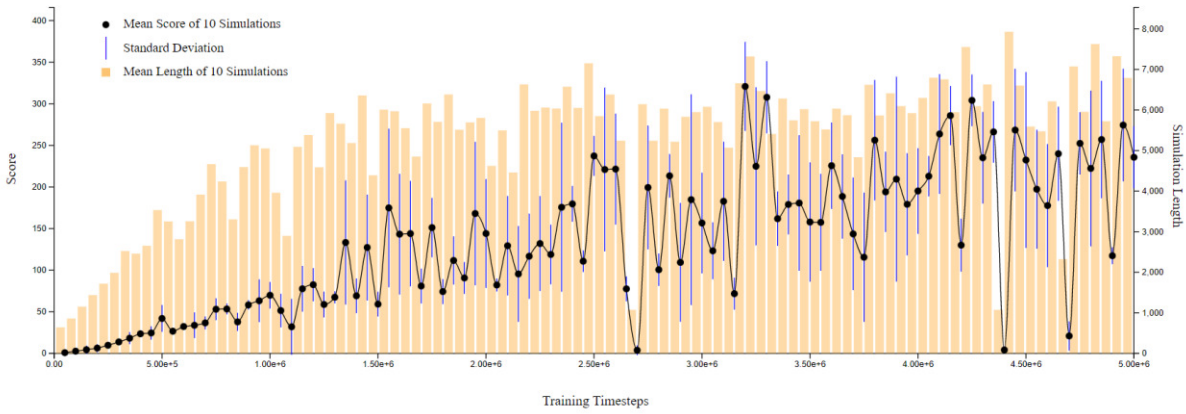
from targeting the lower bricks again. I believe with more training this can be solved by the agent.

Looking at the overview chart, I noticed that the standard deviation at the early stages is small, and I believe that this is due to the agent not being able to achieve different scores using different play styles. To back this, the actions chart indicates in the early stages that only the right action is being utilised meaning that the scores would be very consistent without many changes as shown below:





While targeting my secondary objective of training with the A2C algorithm and comparing the overview charts between DQN and A2C, I have noticed that A2C is less stable compared to DQN and this is mainly because DQN uses a replay buffer instead and therefore produces better and more accurate results. I have also found that both DQN and A2C discovers the strategy of digging a tunnel through the bricks. Below is a representation of A2C score overview chart:



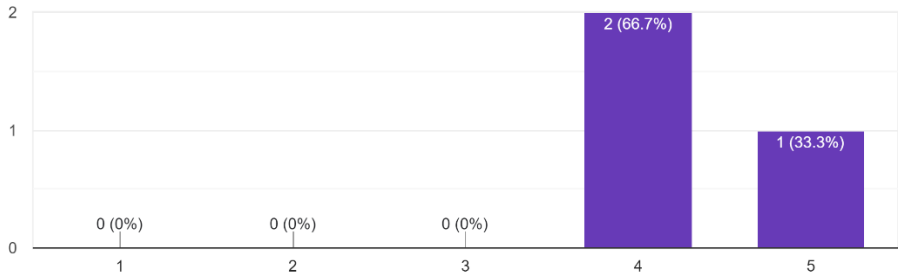
Expert Feedback

The RL experts evaluating my system was organised by my supervisor being her students. My supervisor who is also a RL expert has also provided consistent feedback throughout my development stage. To obtain feedback from these RL experts, I have decided to use Google Forms where I am able to question them on the usefulness and intractability of the system.

The RL experts would be able to answer using a scale of 1 being poor to 5 being excellent. Below consists of the feedback for all eight questions in the form:

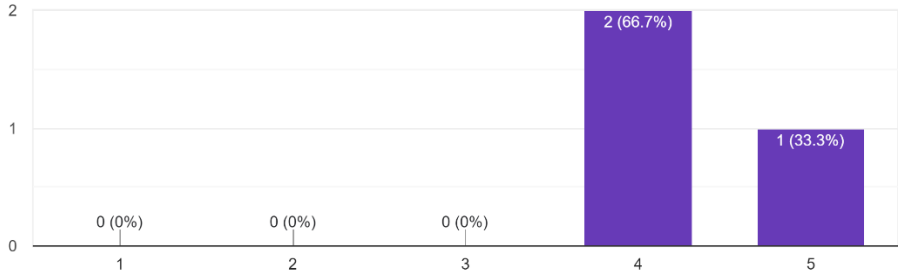
Charts are easy to read and understand

3 responses



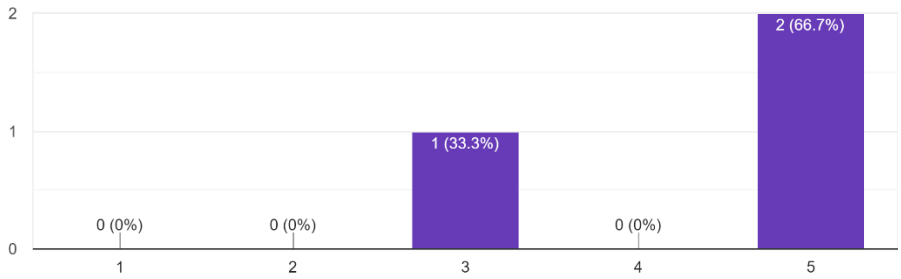
Instructions provided help with navigation

3 responses



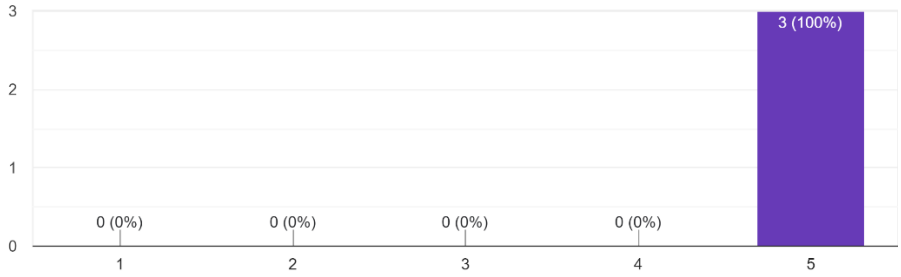
Interactions are easy to detect and simple to use

3 responses



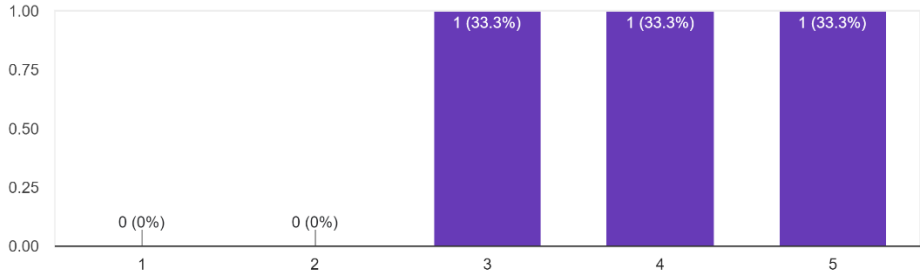
Overview charts displayed are useful

3 responses



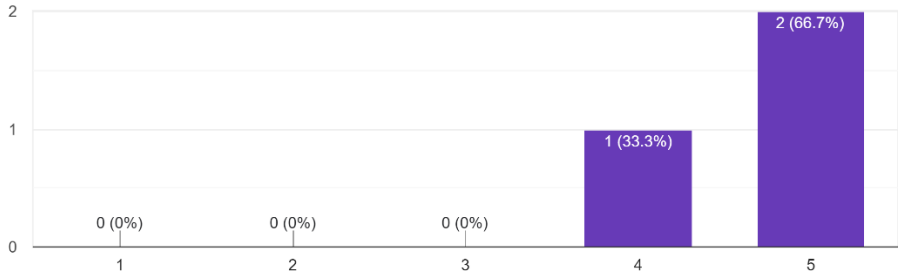
Parallel coordinates chart is useful

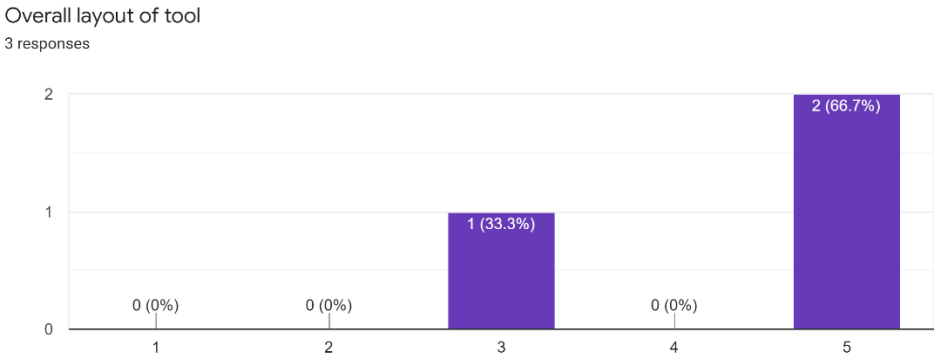
3 responses

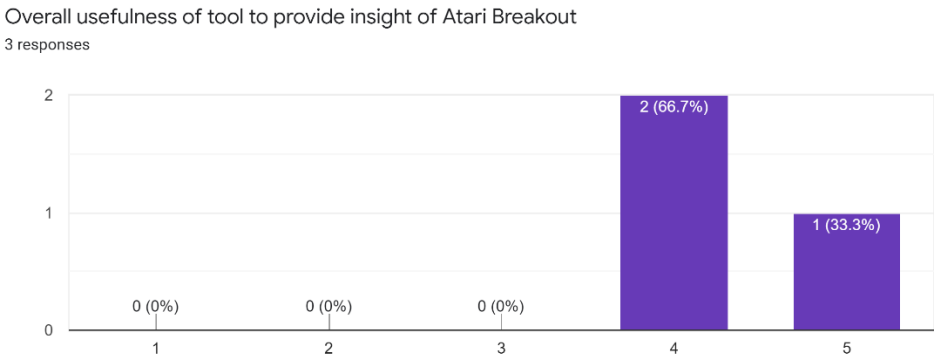


Accumulated rewards is useful

3 responses







With this feedback, it has provided me with valuable insights where I able to see through an expert's perspective on what they found useful. It has shown me that the overall usefulness and information of my system is well established. However, it also shows that there are some improvements to be made with the layout and interactions as well as certain visualization techniques applied.

Besides the forms, the experts have also provided me with some external feedback for further elaboration on their likings, desirables, and issues.

For the first expert feedback, they found it nice to visualize the performance of a single agent being able to click from the overview down to seeing the simulation running. They were also unsure whether the simulation was running I real-time or a recording playback which meant that I would need to include this information on the system as to not cause confusion. One of the first point for improvement is they wanted to know if it was possible to show comparison of different agents in the same chart instead of individually as it would be useful to visualize the different agents with different parameters or algorithms. Another point was that they would prefer a brief documentation or demo on how to use the system as it would help a lot. These gave me some ideas that I believe would further enhance the visualizing capabilities. They also wondered about the difficulty in using the libraries for RL and wanted to know how convenient and customisable it is.

Another short feedback from the second expert focused more on my secondary objective of comparing different RL algorithms. They stated that they would prefer the charts for both the algorithms be placed side-by-side for easier comparison instead of being in another page.

For the feedback of the final expert, they have suggested some visualization techniques for example, having a graph demonstrating the change in trend for reward loss during the training process and to further visualize the sudden increase or decrease as they stated that analytical losses are quite common in RL. He also mentioned that my visualization is too focused and complete whereby I am only able to show sudden changes such as displaying problems where the agent has not received a reward for a long period of time. He also believed that a webpage model is insufficient and difficult to show and identify complex problems. As Atari problems are simple and can fit four sets of reward models, he stated that navigation problems would be more difficult for these techniques to be used. As experts are usually more attentive towards the training time and efficiency, he suggested having a feature that can indicate the training time. With my secondary objective, he suggested that there could be a display of results to see which RL algorithm approach is better as experts are usually confused about the RL models to use. Another external feature is to show the sample efficiency of the algorithm as he indicated that most articles on RL focuses to improve the sample efficiency and diversity of buffers.

Future Work

Looking at the expert feedback, there are quite a few improvements and features that can be implemented in the future where there isn't much time constraint. As stated in one feature, having two different agents being visualized side-by-side can be a useful tool to help in comparing different results to gain more in-depth details on the attributes affecting the change. As the layout was not excellent, providing documentation or demos can help experts to navigate the system quicker and easier. It would also be a useful tool to have a feature that can compare the different RL algorithms and present the algorithm with the highest result that is best suited for the specific environment. I would also want to train more RL models to be integrated into the system in the future as these trainings usually require a very long training process.

In one paper it states that experts usually have trained agents and want to investigate their decision-making process. Therefore, I would also need to ensure that my implementation code must be maintainable for future improvements and additions of other RL models whereby experts would be able to easily upload their trained agents for a simpler visualization process.

For some of my own desirable features, I would want to look further into the agent's behaviour for example touching into the observation space to see through the agent's perspective. I believe these in-depth details would provide even more interesting cases and provide an easier explanation to the agent's learned behaviours.

I feel that the data that I have collected can be more accurate by simply increasing the number of simulations which would require more runtime. I have also investigated some data smoothing methods which I hope can be applied in the future for better results and pattern detection.

Conclusion

The visual analytics system that I have developed has targeted all the defined requirements, but some were incomplete due to limitations. I also believe that there can be more requirements established to target the expert's desires in more detail.

The main objective for my project was to develop my own visualization techniques while studying at existing solutions. As the existing solutions target more towards the training process and memory, my solution differs instead by explaining how the different agents behaves at different training stages, and how they evolve. This to me was an interesting path as I wanted to learn reinforcement learning and it has at the same time given me the opportunity to visual it along the way.

Throughout the process, there were many issues with time management which led to being behind schedule during the later development stages. Most of these issues are tied to the uncertainty of agent training which heavily affected my planning.

As this was the first prototype, having feedback from experts who are experienced in the field, provide positive feedback has allowed me to see the usefulness of my tool and hints towards whether I am heading in the correct direction.

Reflection

There were many difficulties that I have faced over the course of this project due to my own skills and time management. Many of these delayed schedules were affected during training where the learning the SB3 documentation took longer than expected as I started encountering errors and incorrect data. The visualization process also plays a factor in the delay due to the steep learning curve of D3.js and I needed to grasp the basic concept to be able to utilise the advanced features.

One of the major parts of training and data collection that I wished could be different is the structure of my data. I noticed that a lot of issues relating to visualization was due to my data storage structure where it could have been avoided if I had better planning knowledge of what I needed. In the beginning I would have also combined relevant data better so that it can be easily accessible for future work.

Overall, this project has allowed me to learn quite a lot about reinforcement learning and to see how it compares to other forms of machine learning. Learning of D3.js is also a very valuable and transferable skill as it is quite popularly used for data visualization in the programming and development industries.

References

[1] J. Wang, L. Gou, H.-W. Shen and H. Yang, “DQNViz: A Visual Analytics Approach to Understand,” IEEE TVCG. 25(1): 288-298, 2019.

[2] T. Jaunet, R. Vuillemot and C. Wolf, “DRLViz: Understanding Decisions and Memory in Deep,” Computer Graphics Forum, 2020.

[3] V. Mnih, K. Kavukcuoglu, . D. Silver, A. Graves, . I. Antonoglou, D. Wierstra and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” 2013.

[4] A. Hill, A. Raffin, M. Ernestus, A. Gleave and A. Kanervisto, “Stable Baselines3,” 2020. [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/. [Accessed 2022].

[5] OpenAI, “Gym,” [Online]. Available: https://gym.openai.com/. [Accessed 2022].

[6] M. Bostock, “D3.js,” 2021. [Online]. Available: https://d3js.org/. [Accessed 2022].

[7] N. D. Nguyen, S. Nahavandi and T. Nguyen, “A Human Mixed Strategy Approach to Deep Reinforcement Learning,” IEEE, 2018.

[8] D. Amir and O. Amir, “HIGHLIGHTS: Summarizing Agent Behavior to People,” Scholars at Harvard, 2018.