ROCK
PAPER
SCISSORS
LIZARD
SPOCK

THE ULTIMATE GAME

AI REINFORCEMENT LEARNING PROJECT

presented by...

HAYLEE MCLEAN

QUENTIN O'NEAL

TIA SCOTT

Overview

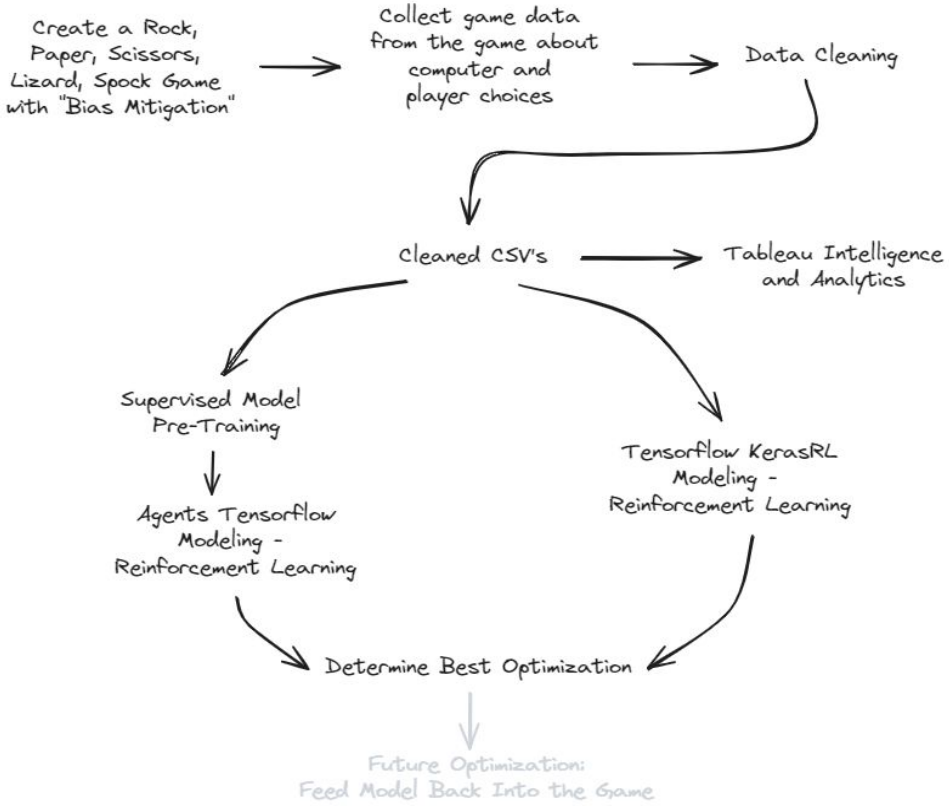
ROCK
PAPER
SCISSORS
LIZARD
SPOCK

Our project was to create a ROCK, PAPER, SCISSORS, LIZARD, SPOCK game, collecting user data from the game, storing this data, and then using the data to build machine learning models.

Our goal was to understand user behavior and potentially develop optimal strategies for the computer increase it’s chance of winning.

Thought Process

ROCK
PAPER
SCISSORS
LIZARD
SPOCK



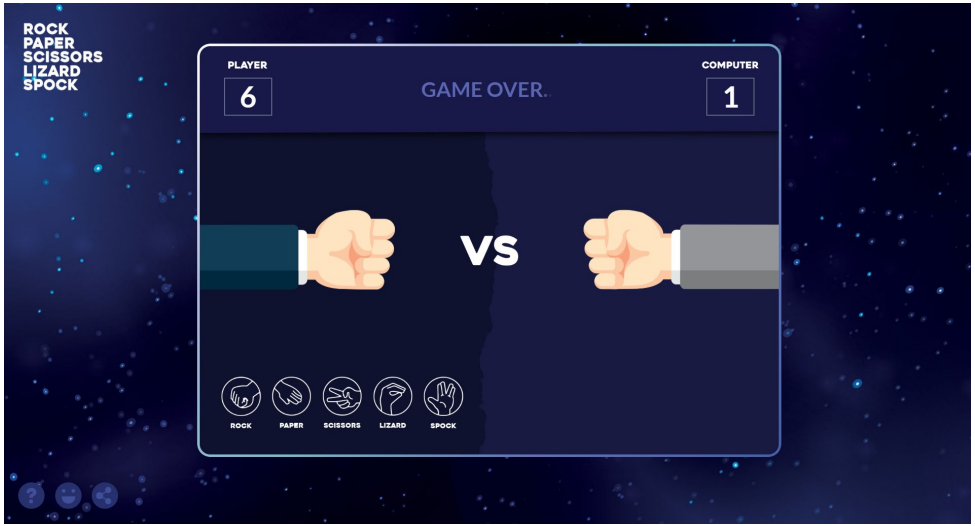
Game Design

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

The website was built using html and css, with javascript integrated into it for various aspects.

The game requirements included:

1. Game Logic
2. Comprehensive Logging
3. I/O Data Aggregation

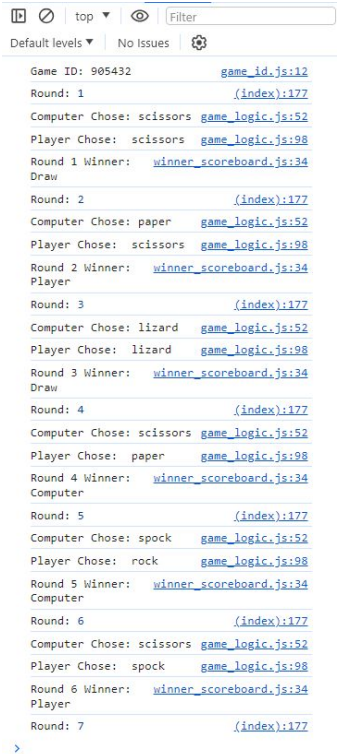


Game Design - Game Logic, Player

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

The player will see a seamless design that allows them to pick a hand (with buttons) and have the action called via javascript and keyframe animation after countdown.

The game was designed to make players “feel” like they are playing they are playing a “similar” opponent elicit real play responses.



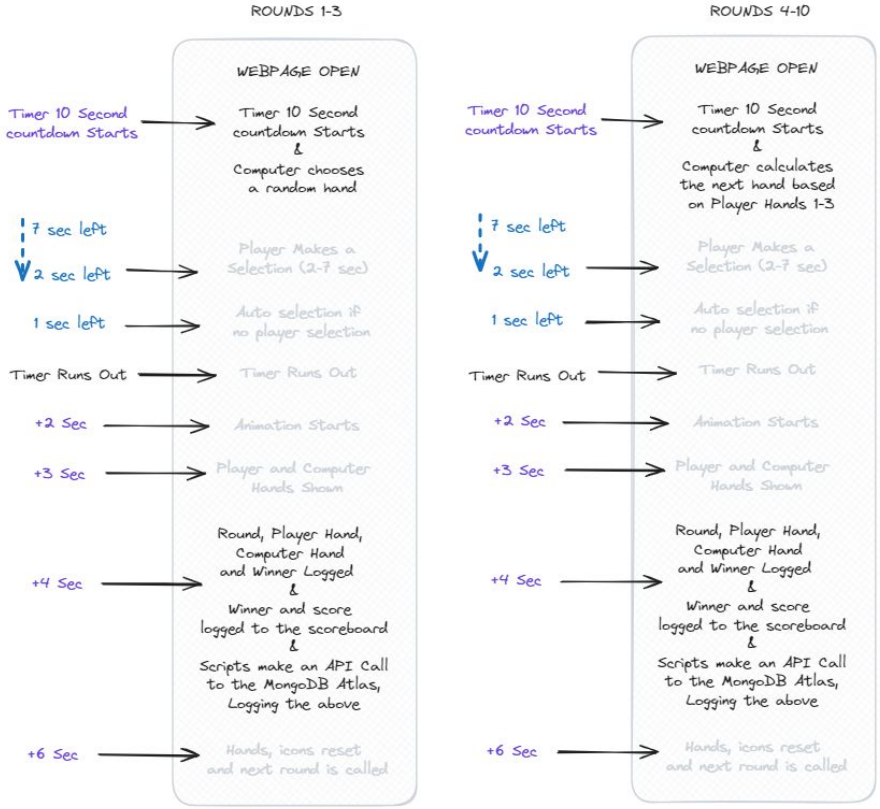
Game Design - Game Logic, Logging

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

The game picks a hand at the initial timer instance to avoid bias choosing against the player.

The game uses “random” hands for games 1-3, then uses an N-gram to predict the next likely probability using the last three hands (n-3 items) for games 4-10.

Scripts log the round, computer choice, player choice, and round winner.

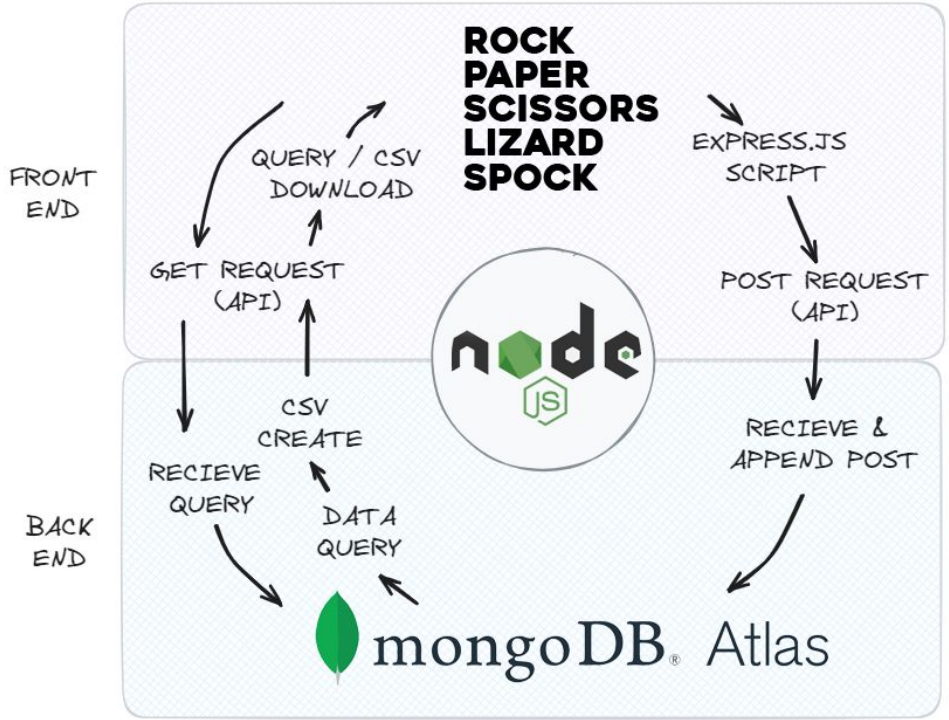


Game Design - I/O Data Aggregation

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

After A LOT of exploration, we decided to use an event-driven non blocking I/O model. This is the heart of the game reporting, which uses Node.js loops to initiate and execute logging operations.

In addition to Node.js, mongoDB Atlas provided the noSQL, cloud-based SQL database for storing and logging the game results and for querying real-time game data.



Data Collection, Cleaning

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

Extract: We pulled a csv from the website where the data was collected and stored using MongoDB Atlas.

Transform: Several parts of transformation included weighting rounds, creating a choice winner column, normalizing the round numbers

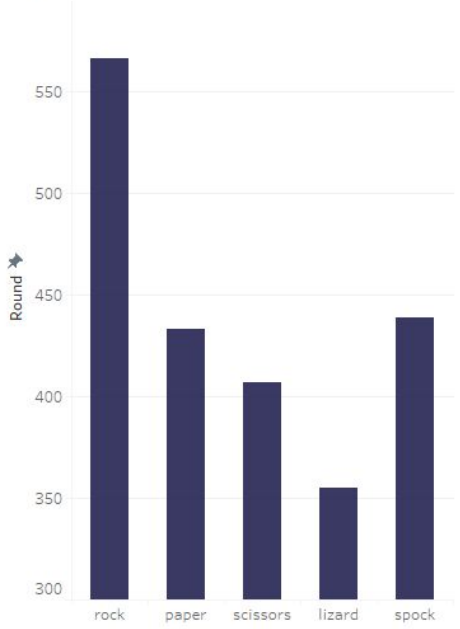
Load: The initial collection of the data from the RPSLS website pushed into our database. After it was downloaded from the database and transformed it was then loaded into the deep learning models.

Tableau Data

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

Computer Choice

Player Choice



Game Winner (N-Gram Model)

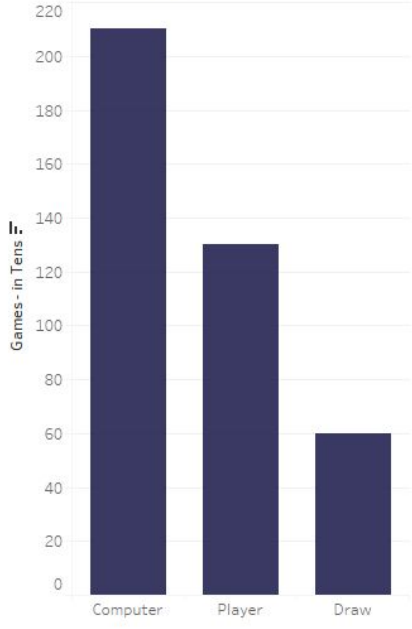
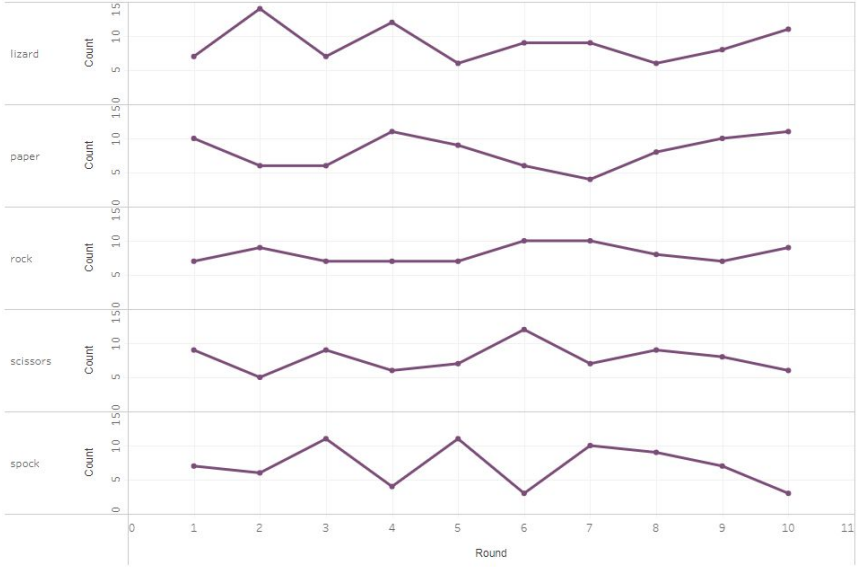


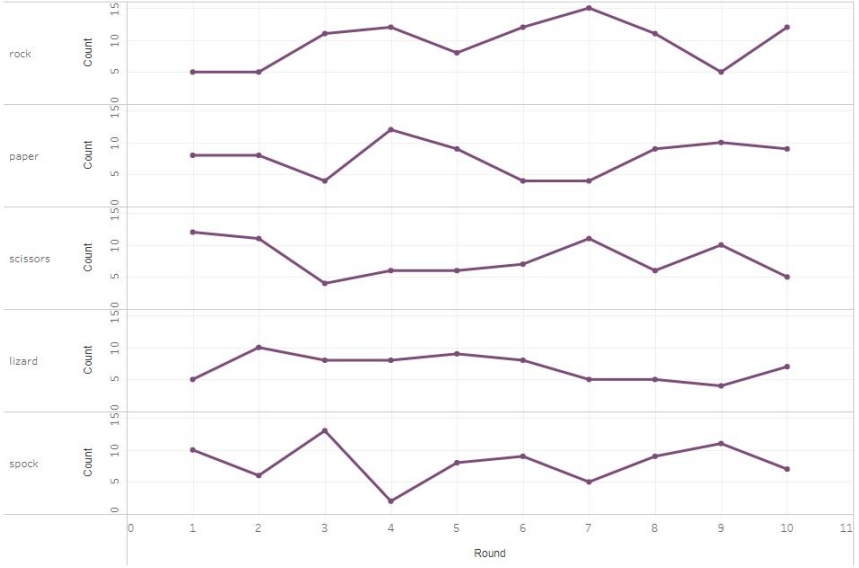
Tableau Data

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

Computer Choice



Player Choice



RL Modeling - Probabilities

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

If the opponent's actions are chosen randomly, the probabilities of winning, losing, and tying:

- **Probability of winning (P_win): 3/5** (3 out of 5 actions result in a win against a random action)
- **Probability of losing (P_loss): 2/5** (2 out of 5 actions result in a loss against a random action)
- **Probability of tying (P_tie): 1/5** (1 out of 5 actions results in a tie against a random action)

With these probabilities, we can calculate the expected value (average score) of each game is:

Expected value (EV) = Pwin × win reward + Ploss × loss reward + Ptie × tie reward.

The question is, what is the baseline for RPSLS if players and opponents were playing randomly?

RL Modeling - Baseline

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

Deep Learning Model Results:

- Accuracy: 40%
- Loss: 0.127

Reinforcement Learning Model Initial Performance:

- Sum: 1
- Mean: 0.01
- Median: 0.0

Enhanced Reinforcement Learning Model:

- Opponent actions incorporate probabilities.
- Sum: 418
- Mean: 4.18
- Median: 4.0

But we can go further with our models by oversampling our data…

Reinforcement Learning

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

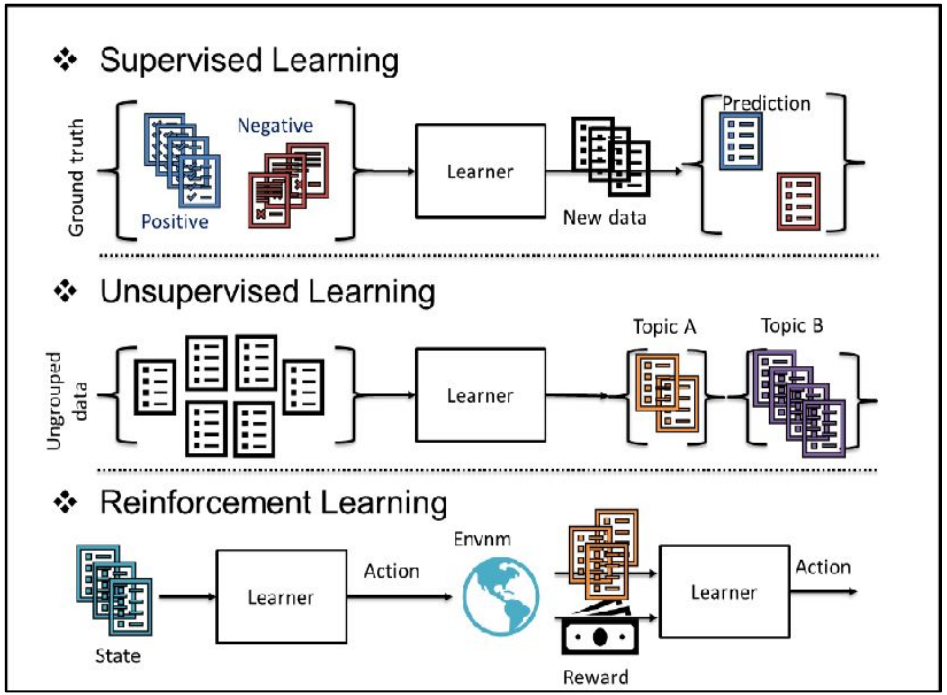
What is Reinforcement Learning?

Reinforcement learning is a type of machine learning where an agent learns to make a decision based on actions in an environment to achieve a goal. Unlike supervised and unsupervised learning, reinforcement learning is about training to make actions for cumulative rewards.

RL Learning v. Supervised & Unsupervised

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

- It involves *learning from interaction* with the "environment."
- The agent learns through trial and error, by receiving feedback in the form of *rewards or penalties*.
- There is no labeled dataset or explicit feedback provided as in supervised learning.
- The *goal is to learn a policy* to maximize cumulative rewards over time.



How Do You “Measure” a RL Model?

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

There are several ways to evaluate the performance of a RL model:

1. **Cumulative Reward:** Track the cumulative reward that the agent receives while interacting with the environment. Plotted over time/episodes to see how the agent's performance improves.
2. **Running Average Reward:** Compute a running average over a certain number of episodes, which can smooth out the plot and make trends more apparent.
3. **Testing Phase:** After training, enter a testing phase with the model's policy fixed (no more learning) and measure the average reward per episode to evaluate the performance.
4. **Win Rate:** Calculate the win rate of the model over a certain number of games. This gives percentage of games won, which is similar to an accuracy metric.

RL Modeling - KerasRL

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

Keras RL (Reinforcement Learning) is a high-level library built on top of Keras, which is a popular deep learning framework in Python. Keras RL provides a set of tools and utilities to implement and train reinforcement learning algorithms efficiently.

Keras RL Components:

- **Environments:** Supports various environments for RL tasks, such as OpenAI Gym environments.
- **Agents:** Provides implementations of popular RL algorithms like Deep Q-Networks (DQN), Policy Gradient methods, Actor-Critic methods, etc.
- **Memory:** Includes memory implementations like replay buffers for DQN.
- **Policy Networks:** Seamlessly integrates with Keras for defining and training these networks.
- **Callbacks and Callback Functions:** Offers callback functions to monitor and control the training process, similar to Keras callbacks.

RL Modeling - Tensorflow Agents

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

The custom environment was created by subclassing the **`py_environment.PyEnvironment`** class from the TensorFlow Agents library, providing a standard interface for RL environments, and the implementation of several methods:

- **`action_spec`**
- **`observation_spec`**
- **`_reset`**
- **`_step`**

The TensorFlow Agents library was also used to load the trained model and use it within the custom environment. The model was trained using a separate script, saved to a file, and then loaded into the environment using the **`tf.keras.models.load_model`** function.

The unique aspects of the TensorFlow Agents library used in this project include:

- **`py_environment.PyEnvironment`**
- **`array_spec`**
- **`trajectory`**

Q-learning method - goal is to train the model to predict Q-values that maximize the expected total reward,. The epsilon-greedy strategy balances exploration (trying out new actions) and exploitation (choosing the best-known actions) to learn an optimal policy

```python
class RPSLSEnvironment(py_environment.PyEnvironment):
    def __init__(self, model):
        self._action_spec = array_spec.BoundedArraySpec(
            shape=(), dtype=np.int32, minimum=0, maximum=4, name='action')
        self._observation_spec = array_spec.BoundedArraySpec(
            shape=(4,), dtype=np.int32, minimum=0, maximum=4, name='observation')
        self._state = np.zeros(4, dtype=np.int32)
        self._episode_ended = False
        self.model = model
        self.num_moves = 0
        self.max_moves = 100  # Set your own maximum number of moves

    def action_spec(self):
        return self._action_spec

    def observation_spec(self):
        return self._observation_spec

    def _reset(self):
        self._state = np.zeros(4, dtype=np.int32)
        self._episode_ended = False
        self.num_moves = 0
        return ts.restart(self._state)

    def _step(self, action):
        if self._episode_ended:
            return self.reset()
```

RL Modeling - Tensorflow Agents Parameters

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

```
# Define the number of episodes to train for, an episode is a complete game
num_episodes = 1000

# Define the discount factor, determines the importance of future rewards
gamma = 0.9

# define intial and minimum epsilon, which is the probability of choosing a random action
# Define the initial epsilon
epsilon = 1.0

# Define the minimum epsilon
min_epsilon = 0.01

# Define the epsilon decay rate, which determines how quickly epsilon decreases
epsilon_decay = 0.995

# Initialize the list to store rewards
rewards = []

# choose optimizer and learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
```

Determine Best Optimization

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

There are almost endless methods to optimize a model.

A few that we used were:

- Creating new features, organizing data by binning.
- Using libraries for optimization, i.e. optimizers like Adam.
- Running different hyperparameters, i.e. loss settings.
- Choosing different policies to determine actions, i.e. exploration versus exploitation.

In the end, our best outcome was data from our game that was encoded, normalized, and oversampled with Tensorflow, trained in a neural network supervised learning model, **achieving 81% accuracy and a 0.31 loss rate**.

Challenges and Considerations

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

This project posed a significant challenge given the creation of our own game, data, and the intricate integration requirements.

- Creating a working game
- Creating game logic that keeps bias at a minimum
- Capturing the data points we needed from the game
- Weighing the best Input/Output options for our game data
- Learning Node.JS as well as Mongoose schema creation
- Learning how to make an RL model and environments
- Using KerasRL and Tensorflow and finding documentation on it
- Determining what the best optimization was for our specific project

Final Thoughts

ROCK
PAPER
SCISSORS
LIZARD
SPOCK

We've been talking about this game a lot… wanna see it?

RPSLSGAME.COM