**Project Report**

1. **Introduction**

The goal of this project is to build an AI bot that can play a video game using an emulator. The objective is to train a machine learning model that can detect the current game state like player positions and buttons, and determine what move to make.

We used the BizHawk emulator to play the game and collected data to train our agent. This project focused on using a Multilayer Perceptron (MLP), a type of Artificial Neural Network, trained on game state data. The bot acts based on predictions for each game frame and controls the character accordingly.

1. **Tools and Technologies Used**

|  |  |
| --- | --- |
| **Tool/Technology** | **Use** |
| Python | Used as programming language for ML algorithms |
| BizHawk Emulator | To play the game for training our agent |
| Artificial Neural Network | Used as Machine Learning technique for pattern learning. |
| Multilayer Perceptron | A specific Artificial Neural Network model used for learning in the bot. |
| TensorFlow Keras | A library used to build, train and evaluate the MLP model. |

1. **Project Objectives:**

Along with learning and practice the objectives of this project include:

* Build an AI that can play video games
* Collect data and process it for training
* Train a model that can predict the next move
* Test the bot on an emulator to evaluate its performance.
* Explore how AI can be applied in learning environments.

**Working Methodology**

**Game Setup:** We played Street Fighter II Turbo (U) using the BizHawk emulator. We downloaded the API and connected the emulator to a Python controller script.

**Generating Dataset:** We played the game manually and recorded each frame’s game state and button presses using the Bot class in bot.py. We created a keyboard.py file in which we mapped Player 1 buttons according to our keyboard for correct dataset generation and to player without a controller using our keyboard. We stored player coordinates, health, move IDs and button states frame-by-frame into GameData.csv. Each row was a snapshot of the game environment and the corresponding button actions.

**Data Preprocessing:**

* We removed unnecessary columns from the raw dataset, buttons and metadata not needed for prediction.
* Created new features x\_diff and y\_diff to measure distance between players.
* Used StandardScaler to normalize inputs.
* Split the dataset into training and testing subsets (80/20 split).

**Model Training:**

We implemented a Multilayer Perceptron (MLP) with TensorFlow/Keras:

* Input: Normalized feature vectors.
* Hidden Layers: Two dense layers with 64 neurons each and ReLU activation.
* Output Layer: 10 sigmoid neurons for each Player 2 button.
* Loss Function: Mean Squared Error (MSE), with the Adam optimizer.
* Training: 1000 epochs with batch size 32.

The model was saved as BotModel.h5 and the scaler as scaler.save

**Challenges Faced:**

**Data Collection:** Initially the dataset was not diverse, many repetitive scenarios. This led to overfitting and the model couldn’t generalize to different game conditions.

**Model Training:** Initially our bot was not making any movement. While training our model our bot was not predicting the move based on the situation. Sometimes it was getting stuck in jumping up and down.

**Solutions:**

We figured out the dataset was not diverse. So, we regenerated the dataset by playing multiple rounds and capturing varied states like attacking, jumping, blocking and retreating through our keyboard. We also improved feature engineering by including distance between players and removed redundant data. After retraining on this improved dataset, the model started responding more accurately to the in-game conditions and producing logical and strategic moves.

**Results and Observations**

* The bot performed much better after retraining.
* It was able to react to player positions by jumping, attacking or moving back.
* The final model predicted multi-button combinations and somewhat mimicked human-like gameplay.
* x\_diff, y\_diff, and player state features helped the MLP to generalize across different fight scenarios.
* Test data accuracy improved, and real-time predictions matched the in-game needs.

**Conclusion**

The bot mimicked human behavior in the game by using game percepts and actions. Despite the challenges we faced, the model worked well and played the game smartly enough.