Artificial Intelligence Project Report
Game Bot for Street Fighter II Turbo

Maryam Khalid (22i-1917)
Maryam Amjad (22i-1924)
Amina Anjum (22i-2065)

Spring 2025

Project Overview

This project involves developing an AI-powered game bot for *Street Fighter II Turbo* using the BizHawk emulator and a Python API. The objective was to replace the rule-based logic in the provided `bot.py` with a machine learning (ML) model to control the bot's actions. The bot is designed to be generic, capable of playing with any character, and avoids using reinforcement learning or rule-based agents. We generated a dataset by playing the game, trained an MLP (Multi-Layer Perceptron) model, and integrated it into the bot to predict actions based on the game state.

Steps Involved in the Project

1. **Understanding the Starter Code**: We analyzed the provided Python API files (`controller.py`, `bot.py`, `game_state.py`, `command.py`, `buttons.py`). The `GameState` object provides player data (e.g., health, coordinates, move status) and game status (e.g., timer, round status). The `fight` function in `bot.py` originally used rule-based logic, which we replaced with an ML model.

2. **Dataset Generation**: We played the game in single-player mode to collect data, running `controller.py` with argument `1` to control Player 1. The script was modified to log game state features and Player 1's button presses into `manual_game_data.csv`. Features included: player health, coordinates, jumping/crouching status, move ID, relative distance, velocities, and health difference. Actions recorded: `up`, `down`, `right`, `left` (boolean values). The dataset contained 20 columns for features and 4 columns for actions, covering multiple rounds with different characters.

3. **Feature Engineering**: Extracted features from `GameState` in `bot.py` using the `extract_features` method. Added derived features like `relative_distance` (absolute difference in x-coordinates between players) and velocities (change in coordinates between frames). Handled initial states (`has_round_started: False`) by setting default values (e.g., health = 176, positions at 205 and 359).

4. **Model Training**: Used the `manual_game_data.csv` dataset to train an MLP model with `scikit-learn`. Preprocessed the data by scaling features using `StandardScaler`. Trained the model to predict four binary outputs (`up`, `down`, `right`, `left`) based on 20 input features. Saved the trained model as `mlp_model_tuned.pkl` and the scaler as `scaler_tuned.pkl`.

5. **Integration into `bot.py`**: Modified the `Bot` class in `bot.py` to load the trained model and scaler. The `fight` method extracts features from the current `GameState`, scales them, predicts actions using the model, and sets the `Command` object's buttons accordingly. Ensured the bot works for both Player 1 and Player 2 by using the `player_id` argument.

6. **Testing and Debugging**: Ran the bot in single-player mode (`python controller.py 1`) and two-player mode (`python controller.py 1` and `python controller.py 2`). Observed that the model initially predicted all zeros (`[0 0 0 0]`), indicating no actions. Debugged by hardcoding actions to confirm the command pipeline worked (both players moved when hardcoded to jump and move right). Identified the model training as the root cause—likely due to insufficient or unbalanced training data.

7. **Two-Player Mode**: Configured BizHawk for two-player mode (VS Battle) and ensured both ports (9999 and 10000) were active. Ran both bots simultaneously, but they remained idle due to the model's zero predictions.

Challenges Faced

- **Model Predictions**: The MLP model consistently predicted all zeros, likely because the training dataset lacked diversity in actions or features didn't correlate well with button presses.
- **Version Compatibility**: Initially used Python 3.11.9, which caused compatibility issues with the model. Switched to Python 3.6.3 as specified.
- **Two-Player Mode**: Ensuring both bots connected to BizHawk without port conflicts required careful timing and configuration.

Future Improvements

- **Retrain Model**: Collect a larger, more diverse dataset with active gameplay (e.g., jumps, attacks) to improve predictions.
- **Feature Enhancement**: Add more derived features (e.g., opponent's move prediction, combo detection).
- **Model Exploration**: Experiment with other ML algorithms like decision trees or neural networks for better performance.