**Artificial Intelligence  
Game Bot for Street Fighter II Turbo  
22I2066 AyaanKhan | 22I-1915\_Ahmed Mehmood | 22I-2066\_AhmedLuqman | 22I-1977 M Afnan  
Submission Date: May 9, 2025**

**Table of Contents**

1. Introduction
2. Objectives
3. Project Setup and Dependencies
4. Code Architecture
   * 4.1 controller.py
   * 4.2 bot.py
   * 4.3 command.py and buttons.py
   * 4.4 game\_state.py and player.py
   * 4.5 project.py (Data Merging)
5. Dataset Generation
6. Feature Extraction
7. Model Development and Training
8. Integration and Execution
9. Usage Instructions
10. Results and Observations
11. Conclusion
12. References

**1. Introduction**

This report documents the design, implementation, and testing of an AI-driven game bot for Street Fighter II Turbo. The project applies deep learning to replace rule-based bots, enabling the program to learn optimal button-press strategies based on historical gameplay data.

**2. Objectives**

* Replace the starter rule-based logic with a trained Neural Network model.
* Create a generic bot capable of playing any character.
* Automate data collection and logging for training.
* Provide clear instructions for reproducibility.

**3. Project Setup and Dependencies**

* **Operating System:** Windows 7 or above (64-bit)
* **Python Version:** 3.6.3 (tested), compatible with Python ≥ 3.7
* **Libraries:**
  + TensorFlow
  + NumPy
  + pandas
  + scikit-learn
* **Emulator:** BizHawk (EmuHawk.exe) with pre-requisites installed from provided zip.
* **API Download:** model.h5, controller.py, bot.py, helper classes – obtained from provided Google Drive link.

**Installation Steps**

1. Install Python 3.6+ and pip.
2. Install dependencies:
3. pip install tensorflow numpy pandas scikit-learn
4. Extract BizHawk pre-requisites and place EmuHawk.exe in a known directory.
5. Download the API zip and extract into your project folder.

**4. Code Architecture**

**4.1 controller.py**

Handles socket communication with the emulator, logging game states to CSV (game\_data.csv), and orchestrating the game loop:

* **connect()**: Opens socket on 127.0.0.1:9999 (player1) or :10000 (player2).
* **receive()**: Reads JSON from emulator, parses into GameState.
* **send()**: Sends bot’s Command object to emulator.
* **DataLogger**: Writes game state + chosen actions to CSV for dataset generation.

**4.2 bot.py**

Defines class Bot which:

* Loads model.h5 at startup (or errors if missing).
* Implements extract\_features() to map GameState to numeric feature vector.
* Implements fight(), feeding features through the Keras model to generate button probabilities, applies thresholding, and returns a Command.

**4.3 command.py and buttons.py**

* **Buttons**: Encapsulates 12 SNES buttons (up, down, left, right, Y, B, X, A, L, R, select, start) with boolean flags and conversion to/from dict for JSON.
* **Command**: Holds two Buttons objects for players, serializes into JSON structure sent to emulator.

**4.4 game\_state.py and player.py**

* **GameState**: Parses the emulator’s JSON into two Player objects plus metadata (timer, fight\_result, round flags).
* **Player**: Converts per-player data (health, coordinates, current move, button states) into attributes for feature extraction.

**4.5 project.py (Data Merging)**

Simple script using pandas to merge multiple recorded CSV datasets (sfdataset\*.csv), remove duplicates, and save sfmerged.csv for training.

**5. Dataset Generation**

1. Run emulator in single-player mode.
2. Execute python controller.py 1 to launch the bot and logger.
3. Play a few rounds manually to seed the dataset (logging human actions).
4. Repeat for all characters to ensure generalization.
5. Collect CSV files (sfdataset.csv, sfdataset2.csv, …) and merge via project.py.

**6. Feature Extraction**

For each time frame, extract:

* Round timer, fight status flags
* Player and opponent: character ID, health, x/y coordinates, jumping/crouching flags, move status and ID
* Raw button states (in data logging only)

These 17 features per perspective form the input vector to the Neural Network.

**7. Model Development and Training**

1. **Data Preparation**:
   * Convert boolean columns to integers.
   * Split sfmerged.csv into features (17 inputs) and labels (10 button outputs) and into training/testing sets (80/20 split).
2. **Architecture**:
   * Input layer: size 17
   * Dense(128, relu)
   * Dense(64, relu)
   * Dense(10, sigmoid) (multi-label)
3. **Compilation**:
   * Optimizer: Adam
   * Loss: Binary Crossentropy
   * Metric: Accuracy
4. **Training**:
   * 100 epochs, batch size 32, validation split 0.1
5. **Model Saving**:
   * Save final model to model.h5

**8. Integration and Execution**

1. Place model.h5 in project root.
2. Launch BizHawk and load ROM.
3. Open Tool Box (Shift+T) and connect via emulator UI.
4. Run python controller.py [1|2] to control player.
5. Observe bot playing; logs generated in game\_data.csv.

**9. Usage Instructions**

1. Zip file, containing:
   * Source files: controller.py, bot.py, command.py, buttons.py, game\_state.py, player.py, project.py
   * model.h5
   * README.md (this report converted to PDF/Word and run instructions)

**10. Results and Observations**

* Plays coherent combos, but occasional misfires under high timer pressure.
* Future work: tune thresholding, explore recurrent architectures for temporal context.

**11. Conclusion**

We successfully replaced rule-based logic with a deep learning model, enabling a generic bot for any character. The project pipeline—from data collection through model training to integration—is fully reproducible as documented.

**12. References**

1. BizHawk Emulator API documentation.
2. TensorFlow Keras API guide.
3. Kaggle Street Fighter dataset examples.