Artificial Intelligence
Project – Part 2

Deadline: 10th May (Wednesday) 11:55 pm.

Instructions:

1. Work in groups of two members.
2. You have to use python programming language.
3. Plagiarism of any kind (copying from others and copying from the internet, etc.,) is not allowed and can result in zero marks in whole assignment category.
4. Any part of the assignment marked 'AI Generated' by ZEROGPT, DETECTGPT can result zero marks.
5. Your code must be properly commented.
6. Any assignment marked late by google will be considered late.
7. You will have to submit the 'bot' file, all your helper classes and heler files and a README in which you will have to explain to us in detail how to run your code/solution on our machine for the tournament. The README should contain all the steps to set up and install the software dependencies or the libraries which you have used and to run your code. This submission will be done via a Google Form which will be shared with you one or two days before the deadline.
8. No marks will be assigned if any of the following deliverables are missing.
    a. The source code of the program and README as explained in 7.
    b. A pdf or word report containing a brief explanation of the steps involved.
9. Put both source code and report in one folder, ZIP it and submit it. Your folder must be named as ROLLNO01_NAME_ROLLNO02_NAME.ZIP
10. Only one student is required to make submission on GCR. Multiple submissions may lead to plagiarism.

GameBot

Prequisites/Dependencies

- Operating System: Windows 7 or above (64-bit)
- The Python API is made in version 3.6.3 and tested only for this version but it should run on any version above 3. For running it in Python versions below 3, some slight changes are required. Probably you would be able to figure them out yourselves but you can always contact the event head for your problem resolution.
- Install the pre-requisites for the BizHawk emulator from the attached zip

Running the game and executing the API code

1. For running a single bot (your bot vs CPU), open the single-player folder. For running two of your bots at the same time both fighting each other, open the two-players folder.
2. Run EmuHawk.exe.
3. From File drop down, choose Open ROM. (Shortcut: Ctrl+O)
4. From the same single-player folder, choose the Street Fighter II Turbo (U).smc file.
5. From Tools drop down, open the Tool Box. (Shortcut: Shift+T)
6. Once you have performed the above steps, leave the emulator window and tool box open and open the command prompt in the directory of the API and run the following commands:

   For Python API: `python controller.py 1`

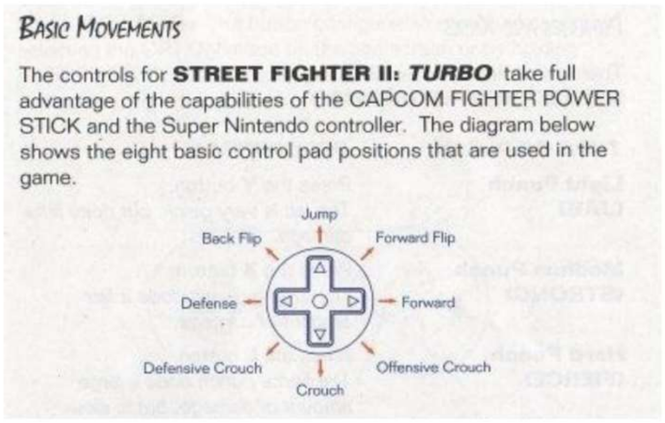
   Note: The '1' at the end of each execution command is a command-line argument. '1' is for controlling player 1 through your bot (left hand side player in the game). '2' if you want to contol player 2 (right hand side player in the game). Any command-line arguments other than '1' or '2' (without the quotes) will cause the code to give an error. (Yes we haven't done exception handling. Deal with it.)
7. After executing the code, go and select your character(s) in the game after choosing normal mode. Controls for selecting players can be set or seen from the controllers option in the config drop down of the emulator.
8. Now click on the second icon in the top row (Gyroscope Bot). This will cause the emulator to establish a connection with the program you ran and you will see "Connected to the game!" or "CONNECTED SUCCESSFULLY" on the terminal.
9. If you have completed all of these steps successfully then you have successfully run the GameBot starter code.
10. The program will stop once a single round is finished. Repeat this process for running the emulator and the code again.
11. To run two bots at the same time. To run two bots at the same time and to make them fight each other, you will have to run both your bots at the same time. One with the command-line argument '1' and the other with the command-line argument '2'. Also be sure to select two players in the game in the VS Battle mode.

API Details

Both the API's contain similar code structure and files. Brief explanation of each file is explained as follows:

- Buttons: Contains the Button class which represents a simple SNES gamepad used for playing Street Fighter. Each gamepad contains twelve buttons. So does the Buttons class which contains 12 boolean members each representing a single button. The function of each button in the game of Street Fighter II can be seen in the below image:

  Street Fighter Controls:

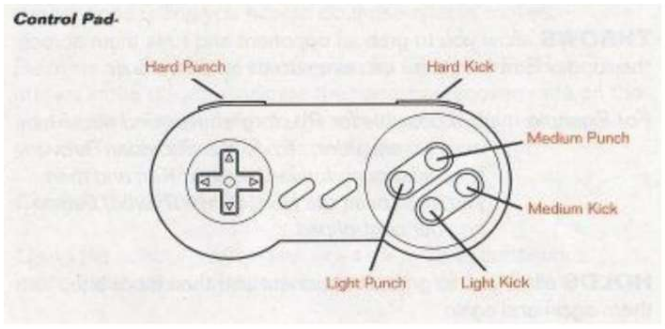


BASIC MOVEMENTS

The controls for **STREET FIGHTER II: *TURBO*** take full advantage of the capabilities of the CAPCOM FIGHTER POWER STICK and the Super Nintendo controller. The diagram below shows the eight basic control pad positions that are used in the game.

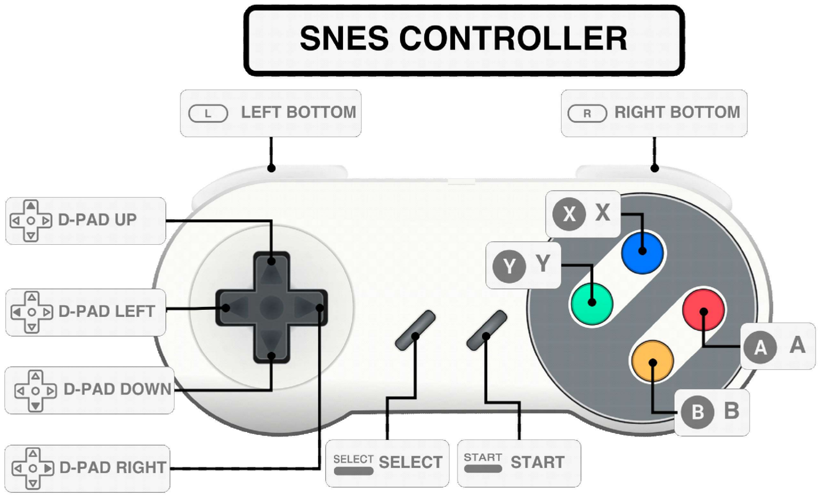
Jump
Back Flip
Forward Flip
Defense
Forward
Defensive Crouch
Offensive Crouch
Crouch

Directional controls (from the SNES SF II Turbo instruction manual)



Button controls (from the SNES SF II Turbo instruction manual)

SNES Gamepad Layout:



Player: Contains the Player class which represents a player. The data members of the Player class are explained as follows:

player_id (integer): The character id of the identifying the character the player has chosen. There are

12 characters present in the game.

health (integer): The remaining health value of the player.

x_coord (integer): The x-coordinate of the player.

y_coord (integer): The y-coordinate of the player.

is_jumping (boolean): Is the player jumping right now?

is_crouching (boolean): Is the player crouching right now?

player_buttons (Buttons Object): The buttons player pressed (set) in the current time instance.

in_move (boolean): Is the player currently performing a move now or is idle?

move_id (integer): What move is the player performing right now?

- GameState: Contains the GameState class which represents all the information we are getting from the game in a single time instance. The data members are explained as follows:

  player1 (Player object): Contains all the information about player-1 for the current time instance.

  player2 (Player object): Contains all the information about player-2 for the current time instance.
  timer (integer): The current time of the round. Each round is of 100 seconds.

  fight_result: What was the result of the fight if the fight is over? Which player won?

  has_round_started (boolean): Has the round started yet?

  is_round_over (boolean): Has the round ended yet?

- Command: Contains the Command class which represents the command which will be passed to the game by the bot for the next time instance. It contains Buttons objects for the respective player from which you are playing. A command to the game will only consist of the buttons you want the bot to press at the next time instance.
- Controller: Contains the socket connection logic, communication logic and the main game loop.

What you have to do?

You have to implement the fight function in the bot.py or Bot.java file. The fight function takes as input a GameState object representing the input from the game at the current time instance and a string indicating which player is playing through this program. You will have to use the information provided in the GameState object, set the buttons accordingly in the Command object and return that Command object from this function. A simple example is provided for your reference in the 'bot' file of both the APIs. In this example, the 'up' button is always being set to true which causes the bot to repeatedly jump in the game.

As a first step you have to generate a dataset by playing single player. You must record all the percepts and actions in a csv. You will need this dataset for the training of the ML/DL algorithm. The solution must be generic we will choose player randomly. **Reinfiorcement learning/ rule base agents are not allowed.**

Good Luck