### **Project Overview: Fall Guys AI**

This project involves developing an AI agent capable of playing the game Fall Guys autonomously. The core components include data collection, model training, and real-time game interaction.

Initially, CreateData.py is used to capture and store gameplay screen images and corresponding player actions, creating a dataset. This dataset is then utilized in Training.py to fine-tune a pre-trained neural network (ResNet), adapting it to recognize and predict the most suitable actions (move left, right, jump, etc) based on the current game screen. By default, the agent will move forward.

The trained model is implemented in TrainedAgent.py, where it processes real-time screen captures and decides on in-game actions. Complementary utility scripts such as grabscreen.py, getkeys.py, and directkeys.py assist in screen capturing, key logging, and simulating key presses, respectively.

Additionally, CreateImages.py and ViewData.py are employed for data verification and inspection, ensuring the integrity and appropriateness of the training data. The project integrates these components to create an intelligent agent that interprets visual input from the game and responds with strategic movements, aiming to play Fall Guys effectively.

### Skeleton Code:

### **File Structure**

To complete:

* RandomAgent.py
  + Provides a baseline agent that performs random actions in a game-like environment. The agent should continuously perform actions until manually stopped. This will serve as a useful baseline to make sure the pipeline is working properly, and to compare our AI model against.
* CreateImages.py
  + Processes the collected data and extracts individual images, saving them as separate files. This is useful for verifying and visually inspecting the data before training or for analysis purposes.
* CreateData.py
  + Collects and preprocesses data for training the AI model. It captures game screen images and the corresponding key presses (actions) and saves them for later use.
* Training.py
  + Utilizes the collected data to train an AI model. It involves loading the data, setting up a model (using a pre-trained network like ResNet), and fine-tuning it on the game-specific data.
* TrainedAgent.py
  + Implements the trained AI model to make real-time decisions in the game. It processes live game images, uses the model to predict actions, and simulates key presses based on these predictions.

### Already completed for you to use: **Utility Scripts in utils:**

* grabscreen.py
  + Captures a screenshot of the game or a specified region of the screen. This is typically used for collecting training data or real-time decision-making in TrainedAgent.py.
* getkeys.py
  + Detects and returns the current state of specified keyboard keys. This script is useful for logging which keys are pressed during data collection (CreateData.py) and for termination conditions in various scripts.
* directkeys.py
  + Simulates key presses and releases using Windows API calls. It's essential for controlling the game in RandomAgent.py and TrainedAgent.py, simulating real key presses based on the AI model's decisions.
* ViewData.py
  + Loads and displays images and their corresponding target actions. This script is useful for visually inspecting the training data, allowing you to check the images and their labels. It serves a similar purpose to CreateImages.py but focuses more on direct viewing rather than saving the images as files.

### **Instructions**

### **RandomAgent.py**

#### 1. **Understanding the Skeleton Code**

* Examine the provided skeleton code to understand its structure and functionalities.
* Identify the parts of the code that are incomplete or placeholders for future logic.

#### 2. **Implementing Random Action Selection**

* Modify the while loop to include a mechanism for choosing actions randomly.
* Use random.randint(0, 3) to select an action randomly.
* Understand that each number from 0 to 3 represents a different action (0: do nothing, 1: move left, 2: move right, 3: jump).

#### 3. **Coding the Action Mechanics**

* Implement the logic for each action inside the loop.
  + For action 0 (do nothing), release all keys except 'W'.
  + For action 1 (move left), press the 'A' key and release 'D' and 'Space'.
  + For action 2 (move right), press the 'D' key and release 'A' and 'Space'.
  + For action 3 (jump), press the 'Space' bar and release 'A' and 'D'.
* Add print statements for each action to provide feedback on what action is being taken.
* Ensure time.sleep(sleepy) is called after each action to maintain a consistent action rate.

#### 4. **Termination Condition**

* Keep the existing mechanism to end the simulation (if keys == "H": break).

#### 5. **Testing and Validation**

* Test the code to ensure it runs without errors and performs actions as expected.
* Validate that the agent can be started with 'B' and stopped with 'H'.

### **CreateData.py**

Your file paths should look something like this:

file\_name = "C:/Users/Algo/Desktop/testing/Fall-Guys-AI/data/training\_data.npy"

file\_name2 = "C:/Users/Algo/Desktop/testing/Fall-Guys-AI/data/target\_data.npy"

#### 1. **Understanding the Skeleton Code**

* Analyze the skeleton code to understand its structure, especially the areas marked for student implementation.
* Note the import statements and recognize their purposes (e.g., cv2 for image processing).

#### 2. **Data Loading and Checking**

* Implement the get\_data function to load existing data if available, or initialize empty lists for new data.
* Use os.path.isfile to check if the data files exist.
* If files exist, load the data using np.load; otherwise, initialize image\_data and targets as empty lists.
* Understand that image\_data will hold the processed screen captures and targets will hold the corresponding key presses.

#### 3. **Data Collection Trigger**

* In the first while loop, implement a key press check to start data collection (e.g., press 'B' to start).
* Use key\_check to listen for the specific key press.

#### 4. **Screen Capture and Image Processing**

* In the second while loop, capture the screen using grab\_screen with a specified region.
* Process the captured image using OpenCV functions (cvtColor, Canny, resize) to prepare it for model input.
* Convert the processed image into a numpy array and append it to image\_data.

#### 5. **Key Press Recording**

* Record the current key presses using key\_check and append them to targets.

#### 6. **Termination Condition**

* Implement a key press check (e.g., 'H' to halt) to stop data collection.

#### 7. **Data Saving**

* Use the save\_data function to save image\_data and targets to the specified file paths using np.save.

#### 8. **Testing and Validation**

* Test the script to ensure it collects and saves data correctly.
* Validate the data format and consistency.

#### 9. **Documentation and Comments (Optional)**

* Add comments explaining each section of the code.
* Ensure the code is clear and well-structured for educational purposes.

#### 10. **Debugging Tool (Optional)**

* Include a debug line (e.g., cv2.imshow) to display the processed image during data collection for real-time feedback.

### **CreateImages.py**

Your file paths should look something like this:

data = np.load("C:/Users/Algo/Desktop/testing/Fall-Guys-AI/data/training\_data.npy", allow\_pickle=True)

targets = np.load("C:/Users/Algo/Desktop/testing/Fall-Guys-AI/data/target\_data.npy", allow\_pickle=True)

#### 1. **Understanding the Skeleton Code**

* Examine the provided skeleton code to understand its structure and purpose.
* Note the use of numpy for data manipulation and cv2 for image processing and writing.

#### 2. **Data Loading**

* Implement code to load the image data and targets using np.load, specifying the correct file paths.

#### 3. **Data Analysis**

* Use np.unique to analyze the distribution of different actions in the target data.
* Print out the shapes of the data and targets arrays for verification.

#### 4. **Data-Target Pairing**

* Create a list (holder\_list) to store pairs of image data and corresponding targets.
* Iterate through the data array to append each image-target pair to holder\_list.

#### 5. **Image Writing**

* Implement a loop to go through each image-target pair in holder\_list.
* Use cv2.imwrite to save each image as a PNG file.
* Name the files based on their corresponding action and a count (e.g., 1.png, 2.png for 'Up' action).

#### 6. **Action Counters**

* Initialize and update counters for each action type (count\_up, count\_left, count\_right, count\_jump) as images are saved.
* Ensure the naming convention reflects the action type and the count

#### 7. **Testing and Validation**

* Test the script to ensure it correctly processes and saves images.
* Validate the saved images to ensure they match their labels and are correctly formatted.

#### 7. **Error Handling (Optional)**

* Optionally, include error handling or logging for unexpected cases in the target data (e.g., unknown actions).

#### 8. **Documentation and Comments (Optional)**

* Add comments explaining the purpose and functionality of each code block.
* Ensure the code is clear and educational for students to understand.

### 

### **training.py**

Your file path should look something like this:

path = Path("C:/Users/Algo/Desktop/testing/Fall-Guys-AI/data/")

#### 1. **Understanding the Skeleton Code**

* Review the provided skeleton code to identify sections marked for implementation.
* Note the FastAI and OpenCV import statements, indicating the libraries used for model training and image processing.

#### 2. **Data Path and File Names**

* Implement code to set the path variable to the directory containing the training data.
* Use get\_image\_files to retrieve filenames of images in the data directory and store them in fnames.

#### 3. **Creating DataLoaders**

* Create an ImageDataLoaders instance using ImageDataLoaders.from\_path\_func.
* Pass in the path, fnames, and label\_func to organize the data into a format suitable for training.
* Set appropriate batch size (bs) and number of workers (num\_workers) for efficient data loading.

#### 4. **Model Initialization and Training**

* Initialize a CNN learner using cnn\_learner, specifying the data loaders (dls), pre-trained model (resnet18), and evaluation metrics (error\_rate).
* Fine-tune the model using learn.fine\_tune with specified epochs and learning rate.

#### 5. **Model Export**

* Export the trained model using learn.export for later use in prediction or further training.

#### 7. **Testing the Script**

* Run the script to ensure it trains the model correctly and exports the trained model.
* Validate that the model performs reasonably on test images if included.

#### 7. **Testing and Validation (Optional)**

* Optionally, include code to test the model on a sample image after training.
* Use grab\_screen, OpenCV processing (e.g., cvtColor, Canny, resize), and learn.predict to test the model's prediction capabilities.

#### 8. **Documentation and Comments (Optional)**

* Add comments to explain each part of the training process.
* Ensure the code is well-documented for educational purposes.

### **TrainedAgent.py**

Your file path should look something like this:

learn\_inf = load\_learner("C:/Users/Algo/Desktop/testing/Fall-Guys-AI/data/export.pkl")

#### 1. **Understanding the Skeleton Code**

* Review the skeleton code to identify the sections where implementation is required.
* Recognize the use of pydirectinput, keyboard, and cv2 alongside FastAI for model prediction and keyboard simulation.

#### 2. **Model Loading**

* Load the trained model using load\_learner and specify the path to the exported model file (e.g., export.pkl).

#### 3. **Gameplay Initialization**

* Implement a trigger to start the AI agent (e.g., pressing 'B').
* Hold down the 'W' key to simulate continuous forward movement in the game.

#### 4. **Real-Time Image Capture and Processing**

* In the while loop, continuously capture the game screen using grab\_screen.
* Process the captured image (e.g., convert to grayscale, apply Canny edge detection, resize) to match the training data format.
* Optionally, display the processed image using cv2.imshow for debugging.

#### 5. **Model Prediction and Action Implementation**

* Use learn\_inf.predict to get the model's prediction for the processed image.
* Implement logic to convert the prediction into an action (e.g., press 'Jump', move 'Left', move 'Right', or 'Do Nothing').
* Use keyboard library functions to simulate the corresponding key presses.

#### 6. **Termination Condition**

* Implement a mechanism to end the simulation (e.g., pressing 'H').

#### 7. **Release Key Presses**

* Ensure all keys are released at the end of the simulation.

#### 8. **Testing and Debugging**

* Test the script to ensure it interacts with the game as expected.
* Debug any issues related to model prediction or key simulation.

#### 9. **Documentation and Comments (Optional)**

* Add explanatory comments throughout the code.
* Document the purpose and functionality of each section for educational clarity.

Acknowledgements:

Kevin Zhu - Algoverse Coding Academy. Clarity Coders.