Introduction

WarMind is a turn-based battle game developed in Python, where an AI opponent decides its actions using a trained XGBoost model. Players can select one of three characters — Knight, Giant, or Musketeer — each with distinct health, damage, and ability stats.

Character Classes (Knight.py, Giant.py, Musketeer.py)

The game includes three main character types: **Knight**, **Giant**, and **Musketeer**.
Each of them inherits from the Character base class defined in CharacterBase.py, which provides shared functionality such as attacking, defending, and using super abilities.

CharacterBase.py (Character class):

This file defines the base class for all characters (Knight, Giant, Musketeer).
It includes shared attributes such as **name, HP, damage, shield uses, and super uses**, and provides fundamental methods for game actions:

- attack() performs a standard attack, returning the character's base damage.
- super_attack() performs a stronger attack if a super use is available.
- enable_defense() activates a defensive stance, reducing incoming damage
- defend(incoming_damage) calculates and applies the damage taken.
- is_alive() checks whether the character still has remaining HP.

AIBotTraining.py (XGBoost Classifier)

The AI model determines the bot's next move (**attack**, **defend**, or **super attack)** based on the current game state.
It applies a **machine learning approach** (XGBoost Classifier) to make dynamic, data-driven decisions.

Data Preparation:

Training data (ai_smart_behavior.csv) includes features such as AI and player HP, damage, shield/super uses, round number, HP difference, and player's previous action.
These features help the model learn how different situations influence the optimal move.

Preprocessing:

Categorical variables (AI_Name, Player_Action) are encoded using **OneHotEncoder**, and the target variable (Action) is encoded using **LabelEncoder**.
The dataset is divided into **training**, **validation**, and **test** subsets for balanced evaluation.

Model Training:

An **XGBoost Classifier** is trained to recognize patterns between the game state and the best action.
Hyperparameters such as learning rate, tree depth, and regularization are adjusted for higher accuracy.
After training, the model's performance is evaluated and saved as ai_bot_model.pkl.

In-Game Decision Making:

During gameplay, the saved model is loaded through the ai_action() function.
It takes the current state of both characters as input and predicts the most suitable action.
This enables the AI to respond intelligently and strategically, making its behavior adaptive and realistic.

Main.py:

This is the core game script that controls the **game flow** and **player-AI interaction**.
It handles:

- Character selection for both the player and the AI opponent.
- Turn-based logic where each round, the player chooses an action (Attack, Defend, or Super Attack).
- Calling the AI's decision logic, which uses the **trained XGBoost model** (ai_bot_model.pkl) to predict the optimal move.
- Displaying current HP values and announcing round outcomes until one character is defeated.