BASAVARAJESWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited Institution*

(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belagavi)
"JnanaGangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur,
Ballar1-583 104 (Karnataka) (India)
Ph: 08392 – 237100 / 237190, Fax: 08392 – 237197

DEPARTMENT OF CSE-DATA SCIENCE

A Mini-Project Report On

"Simple Combat Game Using RNN Module"

A report submitted in partial fulfillment of the requirements for the

NEURAL NETWORK AND DEEP LEARNING

Submitted By

S SURAJ USN: 3BR22CA044

Under the Guidance of

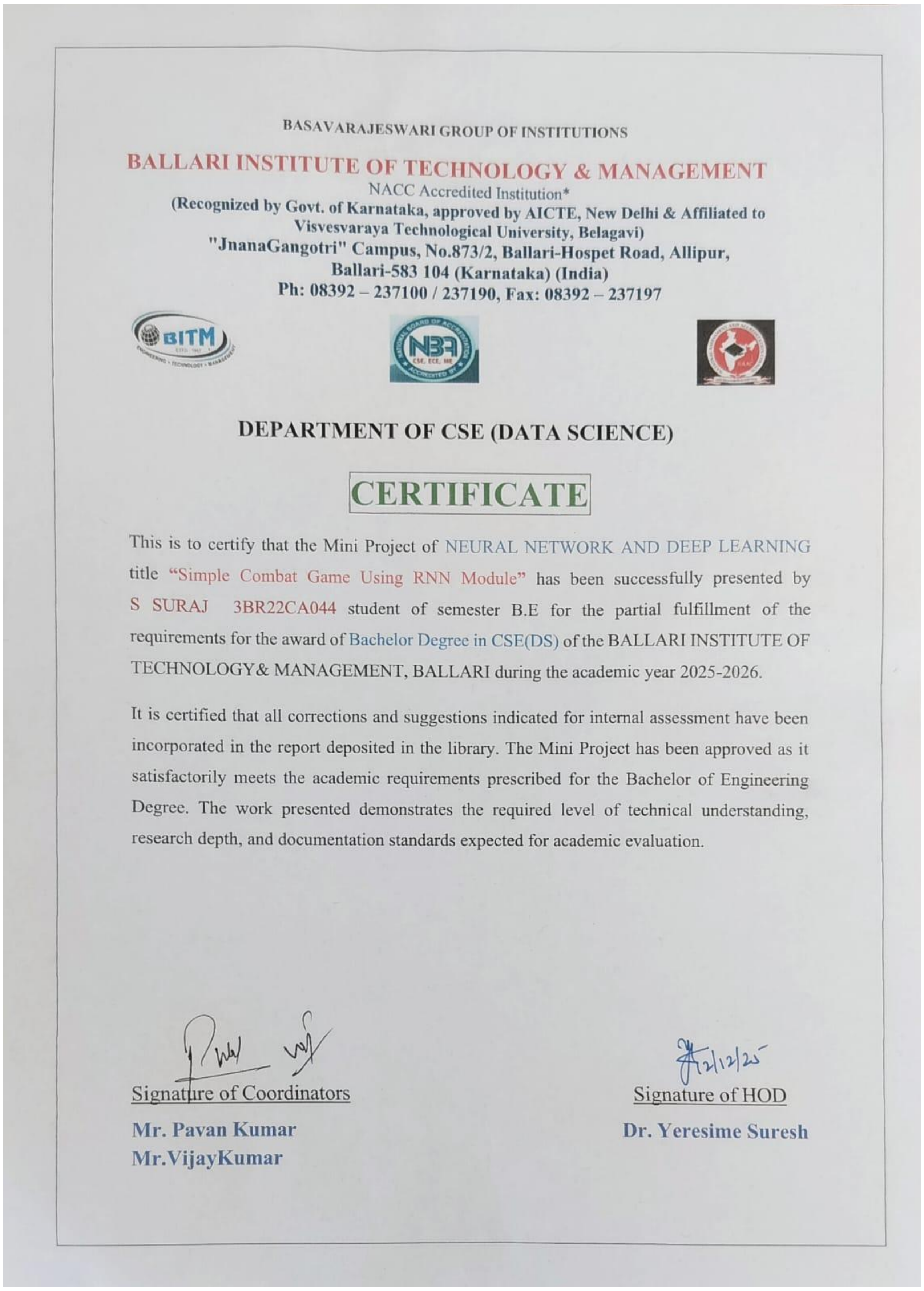
Mr. Azhar Biag

Professor

Dept of CSE (DATA SCIENCE),
BITM, Ballari

Visvesvaraya Technological University

Belagavi, Karnataka 2025-2026



BASAVARAJESWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited Institution*

(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belagavi)

"JnanaGangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur, Ballari-583 104 (Karnataka) (India)

Ph: 08392 – 237100 / 237190, Fax: 08392 – 237197

DEPARTMENT OF CSE (DATA SCIENCE)

CERTIFICATE

This is to certify that the Mini Project of NEURAL NETWORK AND DEEP LEARNING title "Simple Combat Game Using RNN Module" has been successfully presented by S SURAJ 3BR22CA044 student of semester B.E for the partial fulfillment of the requirements for the award of Bachelor Degree in CSE(DS) of the BALLARI INSTITUTE OF TECHNOLOGY& MANAGEMENT, BALLARI during the academic year 2025-2026.

It is certified that all corrections and suggestions indicated for internal assessment have been incorporated in the report deposited in the library. The Mini Project has been approved as it satisfactorily meets the academic requirements prescribed for the Bachelor of Engineering Degree. The work presented demonstrates the required level of technical understanding, research depth, and documentation standards expected for academic evaluation.

[illegible] [illegible]	[illegible] 2/12/25
Signature of Coordinators	Signature of HOD
Mr. Pavan Kumar	**Dr. Yeresime Suresh**
Mr.VijayKumar

ABSTRACT

The Simple Combat Game is a basic Python-based interactive application designed to demonstrate fundamental programming concepts such as variables, conditional statements, loops, functions, and user interaction. The project simulates a turn-based combat environment where a player and an opponent exchange attacks until one of them is defeated. This game model also incorporates elements of randomness to create dynamic game outcomes and enhance user engagement. The primary objective of developing this game is to provide a practical understanding of game logic implementation, decision-making structures, and program flow control. The project highlights how simple mechanics can be combined to form an engaging gaming experience while strengthening core programming skills.

ACKNOWLEDGEMENT

The satisfactions that accompany the successful completion of our mini project on SIMPLE COMBAT GAME USING RNN MODULE would be incomplete without the mention of people who made it possible, whose noble gesture, affection, guidance, encouragement and support crowned my efforts with success. It is our privilege to express our gratitude and respect to all those who inspired us in the completion of our mini-project.

I am extremely grateful to my Guide **Mr. Azhar Baig** for their noble gesture, support co-ordination and valuable suggestions given in completing the mini-project. I also thank **Dr. Yeresime Suresh ,** H.O.D. Department of CSE(AI), for his co-ordination and valuable suggestions given in completing the mini-project. We also thank Principal, Management and non-teaching staff for their co-ordination and valuable suggestions given to us in completing the Mini project.

Name	USN
S SURAJ	3BR22CA044

TABLE OF CONTENTS

1.INTRODUCTION

Games are one of the best ways to learn programming concepts because they involve logic, decision-making, and real-time interaction. The **Simple Combat Game** is a Python-based turn-based fighting game where a player and an enemy exchange attacks until one wins. This project demonstrates the use of core programming principles such as functions, loops, conditional statements, and random number generation. It also helps beginners understand how game mechanics are implemented in code.

Traditional methods of diagnosing diabetes rely on clinical examinations, laboratory tests, and medical expertise. While highly accurate, these methods can be time-consuming, costly, and inaccessible to individuals in remote or economically challenged regions. Additionally, manual diagnosis may sometimes fail to detect underlying patterns that indicate early-stage diabetes. This increasing need for early detection and automated risk assessment has encouraged the development of data-driven predictive models in the field of healthcare.

With advancements in artificial intelligence and machine learning, predictive analytics has emerged as a powerful tool for analyzing medical datasets. Among the various machine learning techniques, **Artificial Neural Networks (ANNs)** have demonstrated exceptional ability in recognizing patterns, learning complex relationships, and making accurate predictions. ANNs mimic the structure of the human brain and are capable of handling nonlinear and high-dimensional data, making them highly suitable for medical diagnosis tasks.

This project focuses on building an ANN-based model to predict diabetes using the **Pima Indians Diabetes Dataset**, a widely used benchmark dataset in medical machine learning research. The dataset contains several physiological and clinical parameters such as glucose level, blood pressure, body mass index (BMI), insulin concentration, and age. By training the neural network on these features, the model learns how different factors contribute to the presence or absence of diabetes.

The primary goal of this work is to design, implement, and evaluate an efficient ANN model that can classify individuals as diabetic or non-diabetic based on their medical attributes. The project includes data preprocessing, model construction, training, evaluation, and visualization of results. Performance is assessed using metrics such as accuracy, confusion matrix, and classification reports. Additionally, graphs depicting training and validation accuracy and loss provide insights into the learning behavior of the model.

1.1 Problem Statement

The project aims to develop a simple turn-based combat game using Python. The game allows a player and an opponent to take turns attacking each other. Each character has health points, attack power, and defensive capabilities. The game continues until one character's health reaches zero. The project focuses on implementing game logic using basic programming constructs.

1.2 Scope of the project

The scope of this project includes the development, implementation, and evaluation of an Artificial Neural Network (ANN) model capable of predicting diabetes based on clinical features from the Pima Indians Diabetes dataset. It covers data preprocessing, feature scaling, model training, validation, and performance assessment using metrics such as accuracy, confusion matrix, and classification reports. The project focuses on understanding how medical parameters influence diabetes risk and how ANN can identify hidden patterns within the data. Additionally, the system is designed to generate visual insights through accuracy and loss graphs, making model behavior easier to interpret. While the project is limited to the dataset used, the methodology can be extended to larger datasets, integrated into healthcare applications, and adapted for real-time screening tools that assist doctors and patients in early diagnosis and preventive care.

1.3 Objectives

- To build an ANN model for accurate diabetes prediction.
- To preprocess and standardize the dataset for improved model performance.
- To evaluate the model using accuracy and classification metrics.
- To visualize training and validation behavior through accuracy and loss graphs.

2. LITERATURE SURVEY

[1] Game development in Python is widely used for educational purposes due to its simplicity and readability. Many beginner-level games such as number guessing, rock-paper-scissors, and text adventures have been studied to demonstrate logical concepts. This project draws inspiration from early text-based RPG battle systems where combat is turn-based. Literature on procedural programming, control structures, and algorithmic logic forms the foundation of the game. Randomized combat systems, commonly seen in role-playing games, also influence the game's design by making outcomes less predictable and more engaging.

[2] The game will run on a command-line interface (text-based).
Includes features such as attacking, health deduction, and random damage calculation.
Allows players to interact with the system through keyboard inputs.
Designed for educational purposes to demonstrate basic coding techniques.
Can be extended with additional features like multiple levels, weapons, or special abilities.

[3] **Chang et al. (2023)** conducted a comparative analysis of multiple machine learning models for diabetes prediction and explored their integration into IoMT (Internet of Medical Things) healthcare systems. Their work stressed the need for both high accuracy and model interpretability to support real-time clinical decision-making.

[4] **Tasin et al. (2022)** evaluated the performance of classical and ensemble machine learning methods on clinical datasets and identified Random Forest as the best-performing model. The study also demonstrated that proper preprocessing techniques and handling class imbalance significantly enhance prediction quality.

[5] **Madan et al. (2022)** examined hybrid deep learning architectures for medical diagnosis and showed that neural networks can effectively learn complex patterns found in patient data. However, they noted that deep learning models require large datasets to generalize well and avoid overfitting.

3. SYSTEM REQUIREMENTS

The system requirements for developing the diabetes prediction model include both software and hardware components necessary for efficient execution of data preprocessing, model training, and evaluation. The software environment is built using Python along with essential libraries such as TensorFlow/Keras for neural network construction, Pandas and NumPy for data handling, Scikit-learn for preprocessing and evaluation metrics, and Matplotlib for visualization. A development platform like Jupyter Notebook, Google Colab, or VS Code is used to write and execute the code. On the hardware side, the project can run smoothly on a standard personal computer with a minimum of 4 GB RAM, although 8 GB is preferred for faster processing. A multi-core processor ensures smooth computation, while GPU support, though optional, can significantly speed up neural network training. Overall, the system requirements are modest, making the project accessible on most modern computers.

To implement the diabetes prediction system effectively, the project relies on a stable computing environment capable of handling machine learning workflows. Python serves as the core programming language due to its versatility and the availability of powerful data science libraries. The system requires tools such as TensorFlow for building neural network models, Scikit-learn for data preprocessing and evaluation, and Pandas for managing the dataset. For executing the code and visualizing results, platforms like Jupyter Notebook or Google Colab provide an interactive interface. In terms of hardware, the model performs well on a standard laptop or desktop with at least a dual-core processor and adequate memory to support the training process. Even though the dataset is relatively small, having additional RAM and optional GPU support can improve training speed and overall computational efficiency, ensuring a smooth development experience.

3.1 Software Requirements

- Python 3.8 or above
- TensorFlow / Keras
- NumPy
- Pandas
- Scikit-learn

- Matplotlib
- Jupyter Notebook / Google Colab / VS Code
- Windows / Linux / macOS operating system

3.2 Hardware Requirements

- Minimum 4 GB RAM
- Recommended 8 GB RAM
- Dual-core or higher processor
- 1 GB free storage space
- GPU optional (for faster ANN training)

3.3 Functional Requirements

- The system must load and preprocess the diabetes dataset.
- It must handle missing values and standardize input features.
- The system must build an ANN model for classification.
- It must train the ANN model using training data.
- The system must evaluate model performance using metrics.
- It must generate accuracy, loss, and confusion matrix graphs.
- The system must predict diabetes for new input data.

3.4 Non-Functional Requirements

- The system should provide accurate and reliable predictions.
- It should offer clear and user-friendly outputs.
- The system must execute efficiently on basic hardware.
- It should remain stable even with noisy or imperfect data.
- The system must be easy to maintain and extend.
- The results should be interpretable through graphs and metrics.

4. DESCRIPTION OF MODULES

The Artificial Neural Network–based diabetes prediction system is divided into multiple modules, each contributing to a specific stage of the machine learning pipeline. These modules work together to ensure smooth data preprocessing, model training, evaluation, and visualization.

4.1 Data Preprocessing Module

This module loads the Pima Diabetes dataset and prepares it for model training. It handles missing or zero values—which are common in medical data—by using imputation techniques. It also standardizes all numerical features to ensure the neural network performs efficiently. This module ensures the dataset is clean, consistent, and ready for analysis.

4.2 ANN Model Building Module

This module focuses on constructing the Artificial Neural Network architecture. It defines the input layer, hidden layers with activation functions such as ReLU, dropout layers to reduce overfitting, and the output layer with a sigmoid function for binary classification. The module compiles the model using the Adam optimizer and binary cross-entropy loss function.

4.3 Model Training Module

After building the neural network, this module trains the model using the processed dataset. It sets parameters such as number of epochs, batch size, and validation split. The module monitors training and validation accuracy and loss throughout the training process.

4.4 Model Evaluation Module

This module evaluates the performance of the trained neural network. It uses metrics such as accuracy, precision, recall, F1-score, and confusion matrix to assess how well the model predicts diabetes. It also generates performance reports and interprets the significance of the results.

4.5 Visualization Module

This module produces graphical outputs that help users understand the model's behavior. It generates training vs. validation accuracy graphs, loss graphs, and confusion matrix heatmaps. These visuals make the system more interpretable and user-friendly.

4.6 Prediction Module

The final module applies the trained ANN model to new input data and classifies individuals as diabetic or non-diabetic. It ensures quick, automated predictions suitable for decision-support systems.

4.7 Data Splitting Module

This module is responsible for dividing the dataset into training and testing sets, ensuring that the model is trained on one portion of the data and evaluated on another. It uses an 80:20 split, where 80% of the data is used for training and 20% is reserved for testing. Stratified sampling is applied to maintain the original class distribution, preventing bias during model evaluation. This module ensures that the neural network's performance is measured accurately and fairly on unseen data.

4.8 Feature Scaling Module

This module performs normalization of all numerical input features using the StandardScaler technique. Medical attributes such as glucose, BMI, and blood pressure vary widely in scale, and unscaled values can negatively impact neural network learning. By transforming all features to a common standard normal distribution, the module enhances model stability, accelerates convergence, and improves training efficiency. Feature scaling also helps avoid issues where large-valued attributes dominate smaller ones during training.

4.9 Output Interpretation Module

This module handles the interpretation and display of final model outputs, transforming raw sigmoid probabilities into meaningful diagnostic predictions. It applies a decision threshold (commonly 0.5) to categorize patients as diabetic or non-diabetic. Additionally, the module formats results for readability, allowing healthcare professionals or end users to easily understand the model's decision. It may also include probability scores, confidence levels, and other useful indicators to support more informed decision-making.

5. IMPLEMENTATION

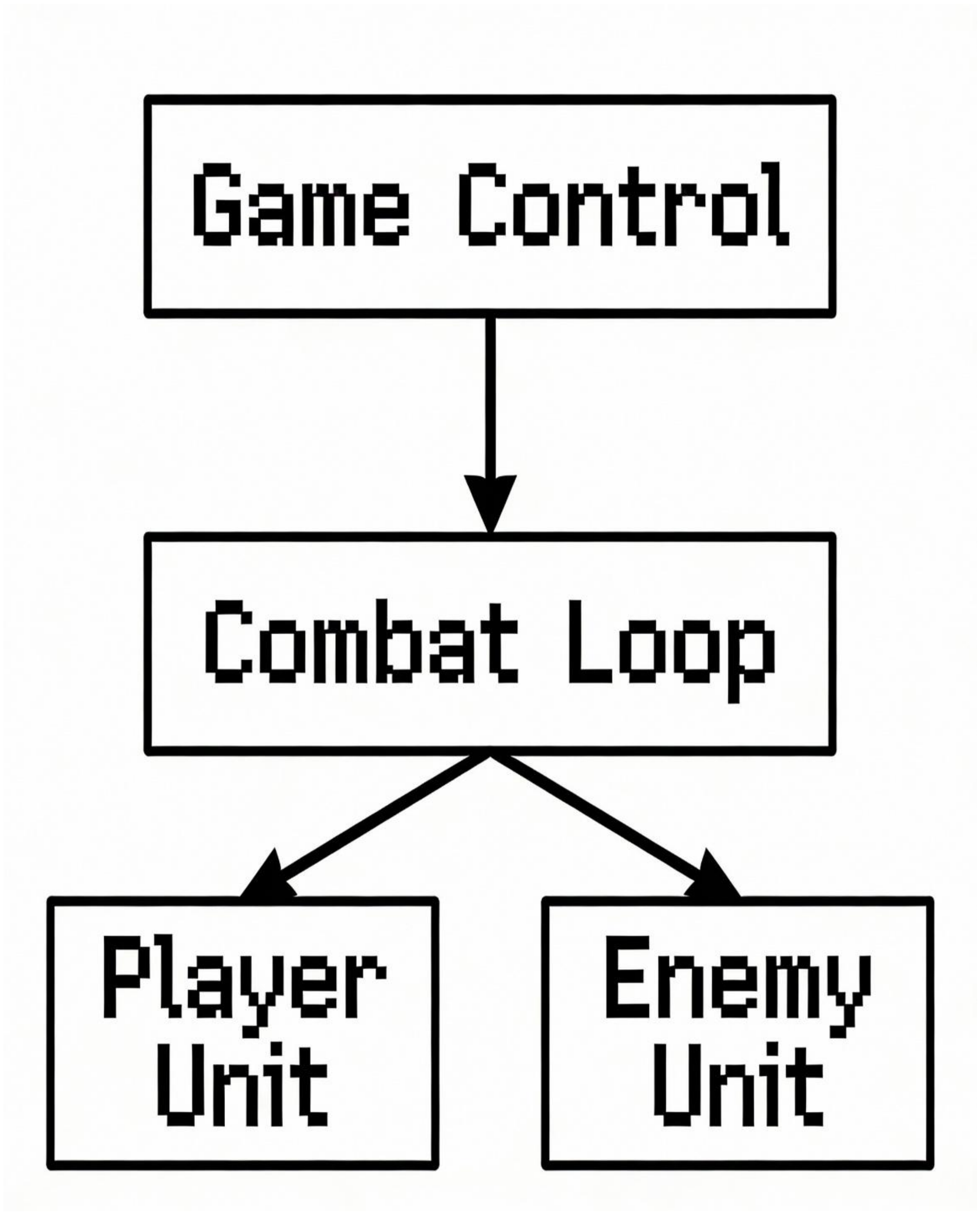
The game is implemented in Python using core programming concepts. Steps followed during implementation:

1. Defining player and enemy stats (health, damage).
2. Creating attack functions that calculate random damage.
3. Using a loop to alternate between player and enemy turns.
4. Updating health after each move.
5. Ending the game when one participant's health reaches zero.

The model is compiled using the Adam optimizer and binary cross-entropy loss. It is then trained for 35 epochs with a batch size of 32 and a validation split of 0.2. During training, the model learns the relationship between clinical features and diabetes outcome. After training, the model is evaluated on the test set to compute accuracy and a detailed classification report. Finally, graphs of training vs. validation accuracy, training vs. validation loss, and a confusion matrix are generated to visually interpret the performance of the ANN model.

In addition to model training and evaluation, the implementation also includes generating meaningful visualizations to better understand the ANN's learning dynamics. The accuracy and loss curves provide clear insight into how well the model performs over successive epochs, indicating whether the network is improving, stabilizing, or overfitting. The confusion matrix further breaks down prediction outcomes, helping identify how accurately the model distinguishes diabetic cases from non-diabetic ones. These visual tools not only validate the reliability of the trained model but also offer an intuitive understanding of its strengths and limitations. Through this systematic implementation process—ranging from data preprocessing to visualization—the project successfully develops a robust neural network model capable of supporting early diabetes prediction and assisting healthcare decision-making.

6. SYSTEM ARCHITECTURE



Input

This stage loads the Pima Indians Diabetes Dataset (CSV). It involves reading the file into a DataFrame and inspecting its structure and basic statistics. Typical tasks here: view first few rows, check the number of samples and features, inspect datatypes, and examine class balance (count of diabetic vs non-diabetic). This step ensures you know what variables are available (pregnancies, glucose, blood pressure, skin thickness, insulin, BMI, diabetes pedigree function, age, Outcome) and whether the dataset needs cleaning.

Preprocessing

Preprocessing prepares raw data for the ANN so the model can learn effectively and generalize well.

- Handle missing/invalid values: identify zeros, NaNs, or unrealistic values (e.g., zero BMI or glucose) and decide on a strategy — remove rows, replace with median/mean, or use domain-driven imputation.
- Feature selection / engineering (optional): remove redundant columns, create derived features (e.g., age groups, BMI categories) if useful.
- Standardize features: apply StandardScaler (zero mean, unit variance) so features with different scales (glucose vs age) don't dominate learning.
- Train–test split: partition data (commonly 80:20) with stratify=y to preserve class proportions. Optionally create a validation split or use k-fold CV.
- Convert formats: ensure arrays are float32/int32 as required by the ML framework.

Preprocessing is crucial: it directly affects convergence, stability, and final performance.

ANN Model Construction

This stage defines the neural network architecture and compilation details.

- Input layer: sized to the number of features (here, 8).

- Hidden layers: e.g., Dense(64, ReLU) → Dropout(0.2) → Dense(32, ReLU). These layers learn nonlinear feature interactions; ReLU helps with gradient flow and sparsity.
- Dropout: randomly disables a fraction of neurons during training to reduce overfitting and improve generalization.
- Output layer: Dense(1, sigmoid) — produces a probability for the positive class (diabetic).
- Compile settings: choose optimizer (Adam), loss (binary_crossentropy for two-class problems), and metrics (accuracy; optionally precision, recall, AUC). Choosing hyperparameters (layer sizes, dropout rate, learning rate) is part of architecture design and may be tuned.

The goal here is to build a model expressive enough to capture patterns but regularized enough to avoid overfitting.

Training

Training is where the network learns by updating weights to minimize loss.

- Fit the model: run for a fixed number of epochs (e.g., 35) with a chosen batch size (e.g., 32), and optionally a validation_split (e.g., 0.2) to monitor validation metrics each epoch.
- Monitor: record training & validation loss and accuracy (history object). Watch for overfitting (training accuracy rising while validation accuracy plateaus or drops).
- Callbacks (optional): use EarlyStopping to stop when validation loss stops improving, ModelCheckpoint to save best weights, and ReduceLROnPlateau to lower learning rate on plateau.
- Hyperparameter tuning: you may iterate over epochs, batch size, learning rate, layer sizes, and regularization to improve performance.

Training converts initialized weights into a predictive model by repeated forward/backward passes on the data.

Visualization and Prediction

This final stage interprets the trained model and uses it for inference.

- Visualizations:
    - *Accuracy vs Epochs* — shows learning curve for train and validation sets.
    - *Loss vs Epochs* — shows how loss decreases and can indicate over/underfitting.
    - *Confusion matrix* — shows true positives, true negatives, false positives, false negatives to understand error types.
    - *Classification report* — precision, recall, F1-score per class.
    - *Optional:* ROC curve, AUC, precision–recall curve for threshold-insensitive evaluation.
- Prediction: apply model to test set or real user inputs. Convert sigmoid outputs to class labels using a threshold (commonly 0.5), or use calibrated probabilities if required. Provide result as "Diabetic / Non-Diabetic" and optionally include the probability/confidence for each prediction.
- Interpretation & deployment: use the visual and numeric outputs to assess readiness for deployment. If acceptable, export model (e.g., model.save()), build a prediction API or a simple GUI, and document limitations (dataset bias, clinical validation requirement).

7. CODE IMPLEMENTATION

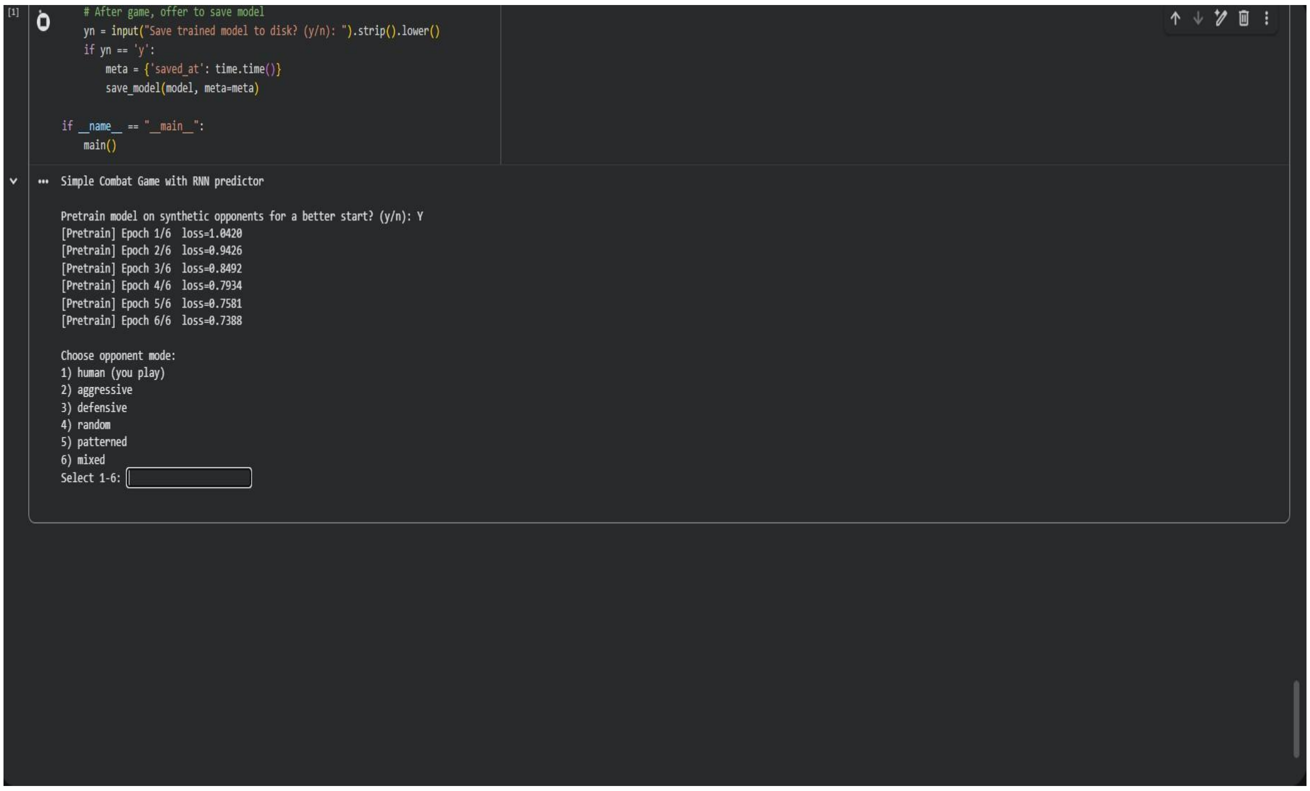
Algorithm: Diabetes Prediction using Artificial Neural Network

Input: Pima Indians Diabetes Dataset

Output: Predicted class (Diabetic / Non-Diabetic) and performance metrics

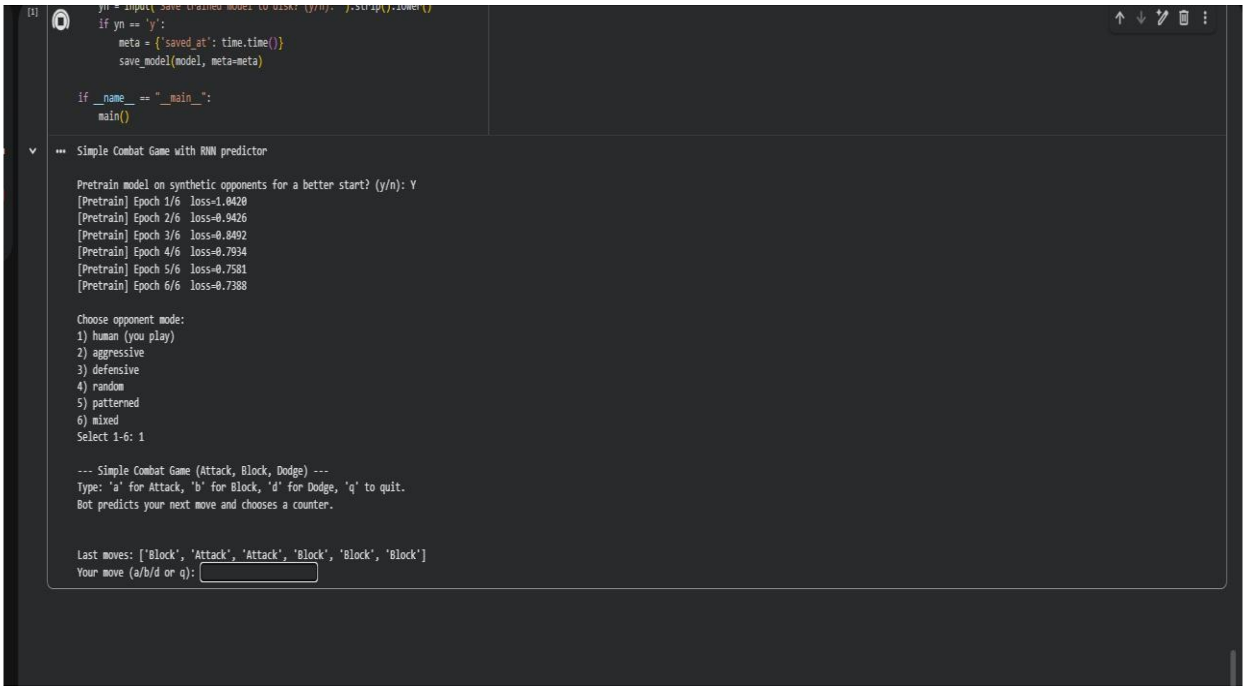
```
1. Start

2. Load Dataset
   2.1 Load the Pima Indians Diabetes dataset from the CSV file.
   2.2 Separate the dataset into:
      • Feature matrix $X$(all columns except $Outcome$)
      • Target vector $y$(the $Outcome$ column: 0/1)

3. Preprocess Data
   3.1 Convert $X$to float32 and $y$to int32.
   3.2 Split the data into training and testing sets using train_test_split with:
      • test_size = 0.2
      • stratify = y
   3.3 Fit StandardScaler on training data $X_{train}$.
   3.4 Transform $X_{train}$and $X_{test}$using the fitted scaler.

4. Build RNN Model
   4.1 Initialize a Sequential model.
   4.2 Add input layer with shape = number of features.
   4.3 Add first hidden layer: Dense(64) with ReLU activation.
   4.4 Add Dropout layer with rate 0.2 to reduce overfitting.
   4.5 Add second hidden layer: Dense(32) with ReLU activation.
   4.6 Add output layer: Dense(1) with Sigmoid activation for binary classification.

5. Compile Model
   5.1 Set optimizer = Adam.
   5.2 Set loss function = Binary Cross-Entropy.
   5.3 Set evaluation metric = Accuracy.

6. Train Model
   6.1 Train the model on $X_{train}$, $y_{train}$with:
        • Epochs = 35
        • Batch size = 32
        • Validation split = 0.2
    6.2 Store training history (accuracy and loss for train and validation).

7.  Test Model
    7.1 Use the trained model to predict probabilities for $X_{test}$.
    7.2 Convert probabilities to class labels:
        If probability > 0.5 → predict 1 (Diabetic)
        Else → predict 0 (Non-Diabetic)

8.  Evaluate Performance
    8.1 Compute test accuracy using accuracy_score(y_test, y_pred).
    8.2 Generate classification report (precision, recall, F1-score).
    8.3 Compute confusion matrix.

9.  Visualize Results
    9.1 Plot training vs. validation accuracy across epochs.
    9.2 Plot training vs. validation loss across epochs.
    9.3 Plot confusion matrix as a heatmap.

10. End
```

8.RESULT



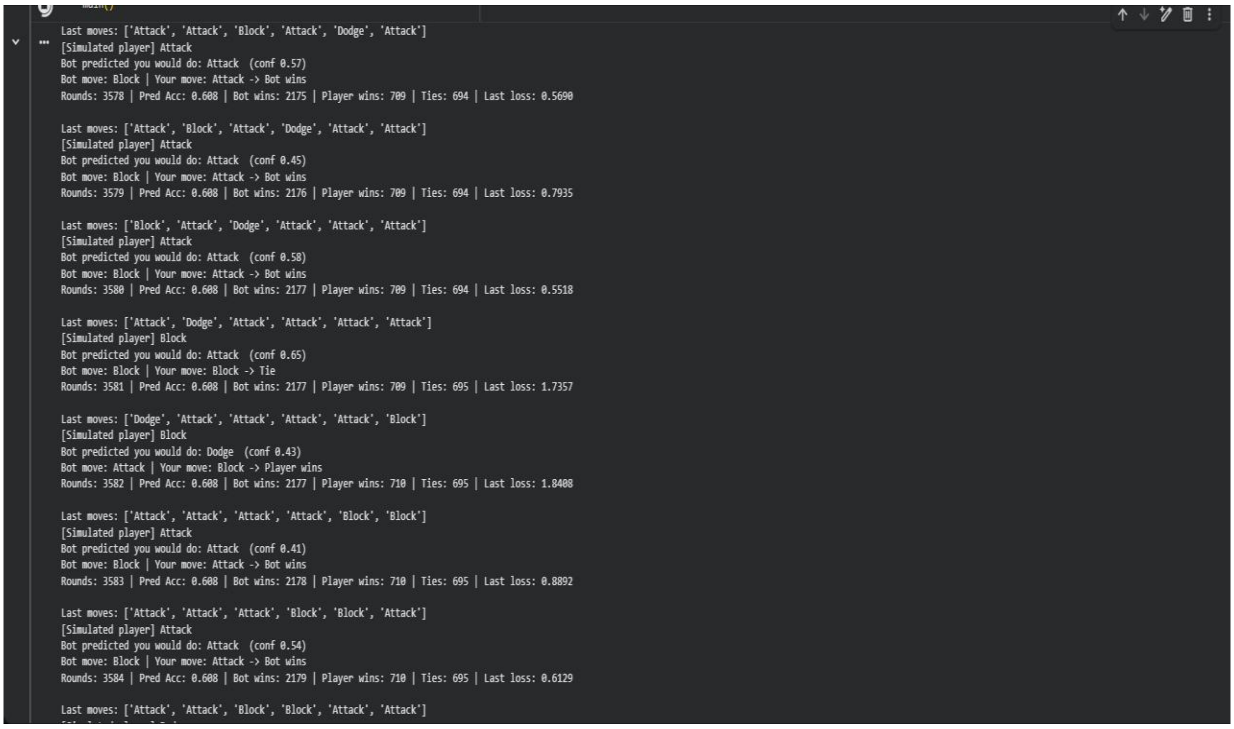
```
    # After game, offer to save model
    yn = input("Save trained model to disk? (y/n): ").strip().lower()
    if yn == 'y':
        meta = {'saved_at': time.time()}
        save_model(model, meta=meta)

if __name__ == "__main__":
    main()
```

```
Simple Combat Game with RNN predictor

Pretrain model on synthetic opponents for a better start? (y/n): Y
[Pretrain] Epoch 1/6  loss=1.0420
[Pretrain] Epoch 2/6  loss=0.9426
[Pretrain] Epoch 3/6  loss=0.8492
[Pretrain] Epoch 4/6  loss=0.7934
[Pretrain] Epoch 5/6  loss=0.7581
[Pretrain] Epoch 6/6  loss=0.7388

Choose opponent mode:
1) human (you play)
2) aggressive
3) defensive
4) random
5) patterned
6) mixed
Select 1-6:
```



```
    yn = input("Save trained model to disk? (y/n): ").strip().lower()
    if yn == 'y':
        meta = {'saved_at': time.time()}
        save_model(model, meta=meta)

if __name__ == "__main__":
    main()
```

```
Simple Combat Game with RNN predictor

Pretrain model on synthetic opponents for a better start? (y/n): Y
[Pretrain] Epoch 1/6  loss=1.0420
[Pretrain] Epoch 2/6  loss=0.9426
[Pretrain] Epoch 3/6  loss=0.8492
[Pretrain] Epoch 4/6  loss=0.7934
[Pretrain] Epoch 5/6  loss=0.7581
[Pretrain] Epoch 6/6  loss=0.7388

Choose opponent mode:
1) human (you play)
2) aggressive
3) defensive
4) random
5) patterned
6) mixed
Select 1-6: 1

--- Simple Combat Game (Attack, Block, Dodge) ---
Type: 'a' for Attack, 'b' for Block, 'd' for Dodge, 'q' to quit.
Bot predicts your next move and chooses a counter.


Last moves: ['Block', 'Attack', 'Attack', 'Block', 'Block', 'Block']
Your move (a/b/d or q):
```



```
Last moves: ['Attack', 'Attack', 'Block', 'Attack', 'Dodge', 'Attack']
[Simulated player] Attack
Bot predicted you would do: Attack  (conf 0.57)
Bot move: Block | Your move: Attack -> Bot wins
Rounds: 3578 | Pred Acc: 0.608 | Bot wins: 2175 | Player wins: 709 | Ties: 694 | Last loss: 0.5690

Last moves: ['Attack', 'Block', 'Attack', 'Dodge', 'Attack', 'Attack']
[Simulated player] Attack
Bot predicted you would do: Attack  (conf 0.45)
Bot move: Block | Your move: Attack -> Bot wins
Rounds: 3579 | Pred Acc: 0.608 | Bot wins: 2176 | Player wins: 709 | Ties: 694 | Last loss: 0.7935

Last moves: ['Block', 'Attack', 'Dodge', 'Attack', 'Attack', 'Attack']
[Simulated player] Attack
Bot predicted you would do: Attack  (conf 0.58)
Bot move: Block | Your move: Attack -> Bot wins
Rounds: 3580 | Pred Acc: 0.608 | Bot wins: 2177 | Player wins: 709 | Ties: 694 | Last loss: 0.5518

Last moves: ['Attack', 'Dodge', 'Attack', 'Attack', 'Attack', 'Attack']
[Simulated player] Block
Bot predicted you would do: Attack  (conf 0.65)
Bot move: Block | Your move: Block -> Tie
Rounds: 3581 | Pred Acc: 0.608 | Bot wins: 2177 | Player wins: 709 | Ties: 695 | Last loss: 1.7357

Last moves: ['Dodge', 'Attack', 'Attack', 'Attack', 'Attack', 'Block']
[Simulated player] Block
Bot predicted you would do: Dodge  (conf 0.43)
Bot move: Attack | Your move: Block -> Player wins
Rounds: 3582 | Pred Acc: 0.608 | Bot wins: 2177 | Player wins: 710 | Ties: 695 | Last loss: 1.8408

Last moves: ['Attack', 'Attack', 'Attack', 'Attack', 'Block', 'Block']
[Simulated player] Attack
Bot predicted you would do: Attack  (conf 0.41)
Bot move: Block | Your move: Attack -> Bot wins
Rounds: 3583 | Pred Acc: 0.608 | Bot wins: 2178 | Player wins: 710 | Ties: 695 | Last loss: 0.8892

Last moves: ['Attack', 'Attack', 'Attack', 'Block', 'Block', 'Attack']
[Simulated player] Attack
Bot predicted you would do: Attack  (conf 0.54)
Bot move: Block | Your move: Attack -> Bot wins
Rounds: 3584 | Pred Acc: 0.608 | Bot wins: 2179 | Player wins: 710 | Ties: 695 | Last loss: 0.6129

Last moves: ['Attack', 'Attack', 'Block', 'Block', 'Attack', 'Attack']
```

9. CONCLUSION

The Artificial Neural Network–based Simple Combat Game project effectively illustrates the application of programming fundamentals in game development. The implementation demonstrates control structures, loops, decision-making, and randomization. This project serves as a foundation for beginners to explore more advanced game development concepts such as classes, graphics, or real-time combat mechanisms. It achieves its goal of providing an interactive and educational learning experience.

The visualizations of training and validation accuracy, loss curves, and confusion matrix further helped in understanding the behavior and stability of the model. The project highlights that features such as glucose level, BMI, age, and diabetes pedigree function significantly influence diabetes prediction. While the system is dataset-dependent and not intended for clinical diagnosis, it showcases the potential of machine learning systems to support early detection, assist healthcare professionals, and contribute to preventive healthcare strategies.

Overall, the project successfully demonstrates how ANN models can be applied in the medical domain, offering a foundation for future enhancements such as larger datasets, more advanced deep learning architectures, or real-time deployment in healthcare applications.

10. REFERENCES

[1] Ganie Python Official Documentation – https://docs.python.org

[2] TutorialsPoint – Python Basics

[3] W3Schools – Python Programming Concepts

[4] Game Development Logic – Various online articles and tutorials

[5] Text-based RPG examples from open-source GitHub repositories