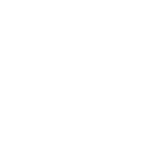
**B.M.S COLLEGE OF ENGINEERING BENGALURU**

Autonomous Institute, Affiliated to VTU



**SPC AAT Report on**

Free Fire Console Game Using C

*Submitted in partial fulfillment of the requirements for AAT*

Bachelor of Engineering

in

Artificial intelligence and machine learning

*Submitted by:*

**DEEPIKA**

**KEERTI Y J**

Department of Artificial intelligence and machine learning

B.M.S College of Engineering

Bull Temple Road, Basavanagudi, Bangalore 560 019

2025-2026

**B.M.S COLLEGE OF ENGINEERING**

**DEPARTMENT ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**

****

***DECLARATION***

We, Deepika and keerti y j, students of 1st Semester, B.E, Department of AIML BMS College of Engineering, Bangalore, hereby declare that, this AAT Project entitled " Free Fire Console Game Using C" has been carried out in Department of CSE, BMS College of Engineering, Bangalore during the academic semester Sep 2025 – Jan 2026. We also declare that to the best of our knowledge and belief, the AAT Project report is not from part of any other report by any other students.

**Student Name Student Signature**

**1.DEEPIKA**

**2.KEERTI Y J**

**BMS COLLEGE OF ENGINEERING**

**DEPARTMENT ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**

****

***CERTIFICATE***

This is to certify that the AAT Project titled “FREE FIRE**”** has been carried out by DEEPIKA (1BM25AI013) and KEERTI Y J (1BM25AI209) during the academic year 2025-2026.

Shruti H

Signature of the Faculty in Charge

Table of Contents

|  |  |  |
| --- | --- | --- |
| Sl. No. | Title | Page no. |
| 1 | Introduction | 05-07 |
| 2 | Algorithm | 08-09 |
| 3 | Flowchart | 10-11 |
| 4 | Source code | 12-18 |
| 5 | Results (screenshots) | 19 |
| 6 | References | 20 |

1.introduction

**Introduction**

The **Free Fire Console Game using C Programming** is a text-based simulation inspired by the popular battle royale game Free Fire. This project is designed mainly for **academic and learning purposes**, focusing on the practical implementation of core **C programming concepts** rather than graphics. The game runs in a console environment and allows the player to experience basic gameplay features such as attacking enemies, managing health and ammunition, progressing through levels, and scoring points.

The primary objective of this project is to help students understand how **structured programming** works in real-world applications. Concepts like **structures** are used to store player and enemy information, **arrays** are used to manage multiple enemies, and **pointers** are used to efficiently pass data between functions. The program is modular in nature, making extensive use of **functions** to improve code readability and reusability. Control statements such as **loops and conditional statements** handle game flow and decision-making.

Additionally, the game includes a **level-based system** that increases difficulty by raising the number of enemies and their attack power. **File handling** is implemented to store game results permanently, demonstrating data persistence. Overall, this console-based Free Fire game serves as an effective mini-project for C programming students to practice logic building, problem-solving, and application of theoretical concepts in a simple yet engaging manner.

Introduction to coding

**🎯 Levels**

* **Level 1** → 3 enemies (easy)
* **Level 2** → 4 enemies (medium)
* **Level 3** → 5 enemies (hard)
* Enemy damage increases with level

**💾 File Handling**

* Saves **player name, level completed, score, result**
* File name: game\_record.txt
* Uses FILE \*, fprintf(), fopen(), fclose()

✔ Uses **structures, arrays, pointers**  
✔ Demonstrates **file handling**  
✔ Menu-based & logical

**🧾 4. IMPORTANT FUNCTIONS USED**

| **Function Name** | **Purpose** |
| --- | --- |
| shootEnemy() | Attack enemies |
| enemyAttack() | Enemy damages player |
| saveGame() | Save data to file |
| reloadGun() | Reload ammo |
| healPlayer() | Restore health |

**Concepts Used in the Free Fire Console Game Using C**

The following **C programming concepts** are used in the above coding:

1. **Structures (struct)**  
   Structures are used to group related data members.  
   Example: Player details (name, health, ammo, score, level) and Enemy details (health, alive status).
2. **Arrays**  
   Arrays are used to store multiple enemy objects and manage them efficiently using index values.
3. **Pointers**  
   Pointers are used to pass structures to functions so that changes made inside functions affect the original data.
4. **Functions**  
   Functions are used to divide the program into small, reusable modules such as shooting, healing, reloading, and saving game data.
5. **Conditional Statements (if, else, switch)**  
   These are used for decision-making like menu selection, checking health, ammo availability, and game results.
6. **Looping Statements (for, while)**  
   Loops are used to:
   * Run the main game continuously
   * Traverse enemy arrays
   * Count alive enemies
7. **File Handling**  
   File handling functions (fopen(), fprintf(), fclose()) are used to store game results in a file.
8. **Random Number Generation**  
   rand() and srand() functions are used to generate random attack damage values.
9. **Input and Output Functions**  
   scanf() and printf() are used to interact with the user.
10. **Header Files**
    * stdio.h for input/output
    * stdlib.h for random numbers and exit
    * time.h for seeding random numbers
11. **Menu-Driven Programming**  
    A menu-based approach is used to allow user interaction and control gameplay.

2. algorithm

 **Start**

 Declare two structures:

* Player (name, health, ammo, score, level)
* Enemy (health, alive status)

 Declare required functions for:

* Shooting enemy
* Enemy attack
* Reloading gun
* Healing player
* Displaying status
* Counting alive enemies
* Saving game data to file

 Initialize random number generator using srand(time(NULL)).

 Read player name from user.

 Initialize player health to 100, ammo to 10, and score to 0.

 Ask user to select game level (1, 2, or 3).

 Set number of enemies based on selected level:

* Level 1 → 3 enemies
* Level 2 → 4 enemies
* Level 3 → 5 enemies

 Initialize each enemy’s health and alive status.

 **Repeat** the following steps while:

* Player health is greater than 0
* At least one enemy is alive

 Display game menu:

* Shoot Enemy
* Reload Gun
* Heal Player
* Show Status
* Exit Game

 Read user choice.

 **If choice is Shoot:**

* Check if ammo is available
* Reduce enemy health by random damage
* Reduce ammo by 1
* Increase score if enemy is defeated
* Enemy attacks the player

 **Else if choice is Reload:**

* Set ammo to full (10)

 **Else if choice is Heal:**

* Increase player health (maximum 100)

 **Else if choice is Show Status:**

* Display player name, health, ammo, and score

 **Else if choice is Exit:**

* Save game details to file
* Terminate program

 End loop when:

* Player health becomes 0 or less
* All enemies are defeated

 If player health ≤ 0:

* Display “Game Over”
* Save result as “Lost” in file

 Else:

* Display “Level Cleared”
* Increase score
* Save result as “Won” in file

 Close file.

 **Stop**

3. flowchart

START

|

v

Enter Player Name

|

Select Level

|

Initialize Player & Enemies

|

WHILE (Player Health > 0 AND Enemies Alive)

|

v

Display Menu

|

User Choice?

|-----------------------------|

| | | | |

Shoot Reload Heal Status Exit

| | | | |

v v v v v

Damage Ammo=10 Health+20 Show Save Game

Enemy (max 100) Status End

|

Enemy Attacks Player

|

Update Health & Score

|

END WHILE

|

Check Result (Win/Lose)

|

Save Game to File

|

END

4. source code

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

/\* Structures \*/

struct Player {

char name[30];

int health;

int ammo;

int score;

int level;

};

struct Enemy {

int health;

int alive;

};

/\* Function Declarations \*/

void showStatus(struct Player \*p);

void shootEnemy(struct Player \*p, struct Enemy enemies[], int count);

void enemyAttack(struct Player \*p, struct Enemy enemies[], int count);

void reloadGun(struct Player \*p);

void healPlayer(struct Player \*p);

int enemiesAlive(struct Enemy enemies[], int count);

void saveGame(struct Player \*p, char result[]);

int main() {

struct Player player;

struct Enemy enemies[5];

int choice, i, enemyCount;

srand(time(NULL));

printf("Enter Player Name: ");

scanf("%s", player.name);

player.health = 100;

player.ammo = 10;

player.score = 0;

/\* Level Selection \*/

printf("\nSelect Level (1-Easy | 2-Medium | 3-Hard): ");

scanf("%d", &player.level);

if (player.level == 1)

enemyCount = 3;

else if (player.level == 2)

enemyCount = 4;

else

enemyCount = 5;

/\* Initialize Enemies \*/

for (i = 0; i < enemyCount; i++) {

enemies[i].health = 40 + (player.level \* 10);

enemies[i].alive = 1;

}

/\* Game Loop \*/

while (player.health > 0 && enemiesAlive(enemies, enemyCount) > 0) {

printf("\n===== FREE FIRE LEVEL %d =====\n", player.level);

printf("1. Shoot Enemy\n");

printf("2. Reload Gun\n");

printf("3. Heal Player\n");

printf("4. Show Status\n");

printf("5. Exit Game\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

shootEnemy(&player, enemies, enemyCount);

enemyAttack(&player, enemies, enemyCount);

break;

case 2:

reloadGun(&player);

break;

case 3:

healPlayer(&player);

break;

case 4:

showStatus(&player);

break;

case 5:

saveGame(&player, "Quit");

exit(0);

default:

printf("Invalid Choice!\n");

}

}

/\* Result \*/

if (player.health <= 0) {

printf("\n💀 GAME OVER!\n");

saveGame(&player, "Lost");

} else {

printf("\n🏆 LEVEL CLEARED!\n");

player.score += 50;

saveGame(&player, "Won");

}

return 0;

}

/\* Functions \*/

void showStatus(struct Player \*p) {

printf("\n--- Player Status ---\n");

printf("Name : %s\n", p->name);

printf("Health : %d\n", p->health);

printf("Ammo : %d\n", p->ammo);

printf("Score : %d\n", p->score);

}

void shootEnemy(struct Player \*p, struct Enemy enemies[], int count) {

int i;

if (p->ammo <= 0) {

printf("❌ No ammo! Reload first.\n");

return;

}

for (i = 0; i < count; i++) {

if (enemies[i].alive) {

int damage = rand() % 20 + 10;

enemies[i].health -= damage;

p->ammo--;

printf("🔫 Shot enemy %d for %d damage\n", i + 1, damage);

if (enemies[i].health <= 0) {

enemies[i].alive = 0;

p->score += 10;

printf("☠ Enemy %d defeated!\n", i + 1);

}

break;

}

}

}

void enemyAttack(struct Player \*p, struct Enemy enemies[], int count) {

int i;

for (i = 0; i < count; i++) {

if (enemies[i].alive) {

int damage = rand() % 10 + (5 \* p->level);

p->health -= damage;

printf("⚠ Enemy attacked you for %d damage!\n", damage);

break;

}

}

}

void reloadGun(struct Player \*p) {

p->ammo = 10;

printf("🔄 Gun Reloaded\n");

}

void healPlayer(struct Player \*p) {

if (p->health < 100) {

p->health += 20;

if (p->health > 100)

p->health = 100;

printf("💉 Health Restored\n");

} else {

printf("❤️ Health Already Full\n");

}

}

int enemiesAlive(struct Enemy enemies[], int count) {

int i, alive = 0;

for (i = 0; i < count; i++) {

if (enemies[i].alive)

alive++;

}

return alive;

}

/\* File Handling \*/

void saveGame(struct Player \*p, char result[]) {

FILE \*fp;

fp = fopen("game\_record.txt", "a");

if (fp == NULL) {

printf("File Error!\n");

return;

}

fprintf(fp,

"Player: %s | Level: %d | Score: %d | Result: %s\n",

p->name, p->level, p->score, result);

fclose(fp);

}

5. RESULTS

Enter Player Name: Rahul

Select Level (1-Easy | 2-Medium | 3-Hard): 2

===== FREE FIRE LEVEL 2 =====

1. Shoot Enemy

2. Reload Gun

3. Heal Player

4. Show Status

5. Exit Game

Enter choice: 1

🔫 Shot enemy 1 for 18 damage

⚠ Enemy attacked you for 14 damage

===== FREE FIRE LEVEL 2 =====

Enter choice: 4

--- Player Status ---

Name : Rahul

Health : 86

Ammo : 9

Score : 0

🔫 Shot enemy 1 for 22 damage

☠ Enemy 1 defeated!

🏆 LEVEL CLEARED!

REFERENCES

**1.Kernighan & Ritchie**, *The C Programming Language*, 2nd Edition, Prentice Hall.  
– Classic book for understanding core C concepts used in this project (structures, pointers, loops, functions).

**1.TutorialsPointC Programming**

**Tutorial**  
–<https://www.tutorialspoint.com/cprogramming/index.htm>  
– Online guide covering all language basics including structures, arrays, file handling, and menu-driven programs.

3. **Balagurusamy, E.**, *Programming in ANSI C*, Tata McGraw Hill.  
– A widely used textbook that explains arrays, structures, file handling, and other basics

**4.GeeksforGeeks – C Programming Language**  
– <https://www.geeksforgeeks.org/c-programming-language/>  
– Helpful resource for examples of pointers, structures, file I/O, and random number generation.

5. **Official GNU C Library Documentation**  
– <https://www.gnu.org/software/libc/documentation.html>  
– Reference for standard functions like fopen, fprintf, rand, etc.