UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

PROJECT REPORT ON MONSTER TAMER IN C

Submitted by:

NAME: Aparna Singh
SAP ID: 590027650
Submitted To: Moshin Dar
Subject: Programming in C
Date: 2 December , 2025

ABSTRACT

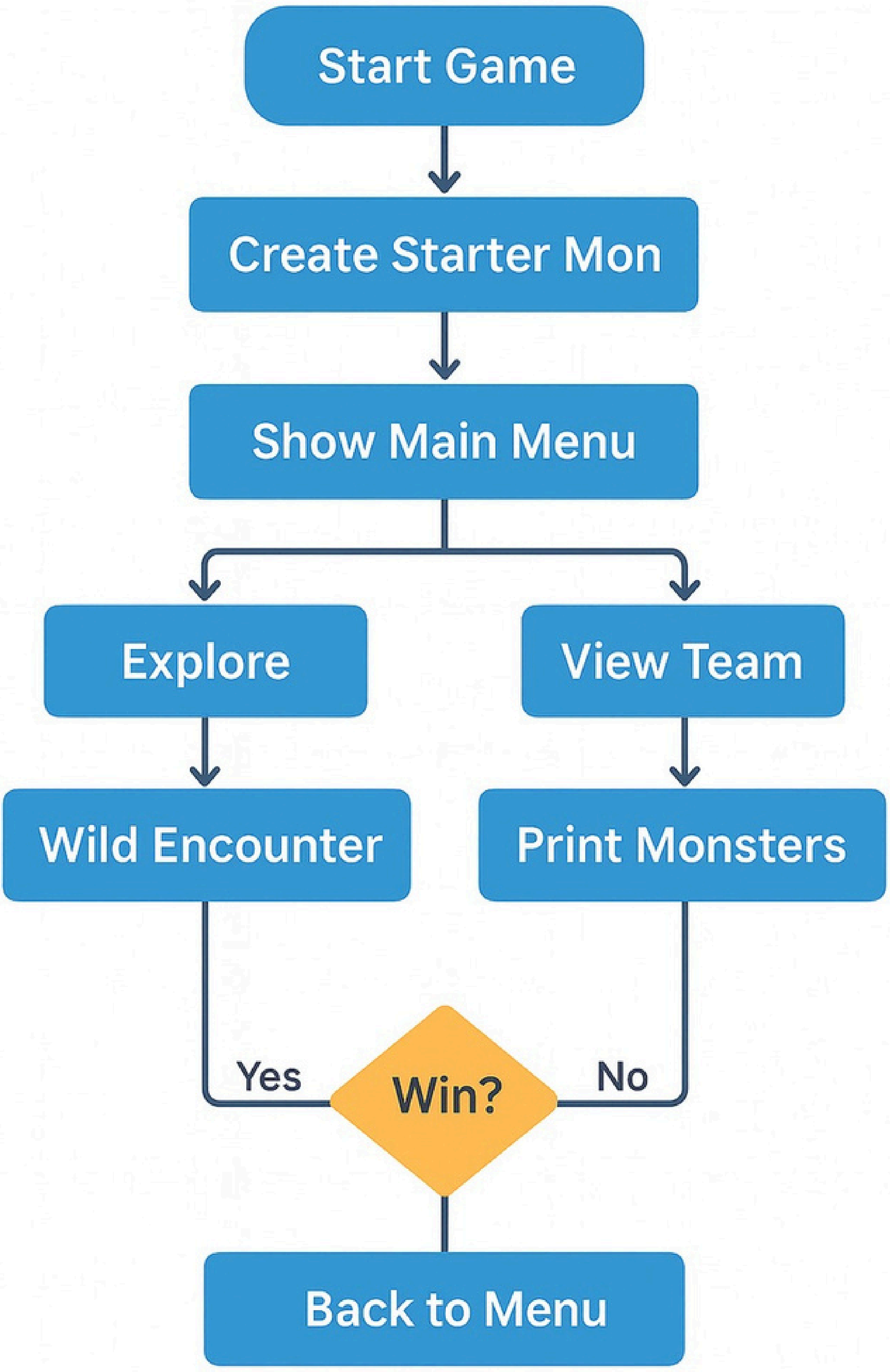
This report explains the idea, design, and development of Monster Tamer RPG, a small console-based role-playing game I created using the C programming language. The main aim of the project was to make something more fun and interactive than the usual basic C programs, while still applying the concepts we learned in class. In the game, the player can move through a simple “wild world,” meet different monsters, battle them, try to tame them, and gradually build a team — all through text menus and simple choices. To keep the project organized, I broke the game into separate modules. There are parts for generating monsters, handling battles, managing the player’s team and items, and controlling the overall game flow. This helped me write cleaner code and understand how bigger programs are structured. While developing the game, I wrote functions for creating starter and wild monsters with random stats, running a turn-based battle system (attack, use potion, or attempt to tame), leveling up monsters, and tracking items like potions and tame-orbs. I also added a main menu that lets the player explore, view their team, or exit the game. Throughout testing, the game behaved consistently — monsters were generated with different stats, battles worked properly, and the game loop stayed stable even after multiple encounters.

PROBLEM DEFINATION

The goal of this project is to design and implement a simple, text-based role-playing game in C that demonstrates structured programming, modular code organization, and interactive user-driven gameplay. Traditional beginner-level C programs focus on isolated concepts such as arrays, loops, or functions, but rarely combine them into a cohesive, fully functional application. This project addresses that gap by creating an engaging RPG system where the player can explore, encounter monsters, participate in battles, and manage a team — all through console interaction.

FLOW CHART

A detailed flowchart representing the entire workflow of Monster Tamer RPG is shown below:


Start Game
Create Starter Mon
Show Main Menu
Explore
View Team
Wild Encounter
Print Monsters
Yes
Win?
No
Back to Menu

ALGORITHMS

The following's are the key algorithms used this projects:

1. Random Monster Generation Algorithm

- Uses rand() with seeded randomness (srand(time(NULL))) to select a random monster type and level.
- Ensures encounters feel dynamic by generating different monsters each time the player explores.
- Implements simple probability-based selection to keep some monsters rarer than others.
- Helps simulate a real RPG environment without hardcoding encounters.

2. Battle Turn Algorithm

- Each turn processes player action first (Attack/Tame/Run) followed by the enemy response.
- Damage is calculated with a deterministic formula:
- damage = base_attack + (level × factor) – defense
- Battle loop continues until either player's HP or monster HP becomes zero.
- Ensures fair, predictable combat flow similar to classical turn-based RPGs.

3. Monster Capture Probability Algorithm

- Applies probability based on the monster's remaining HP: lower HP → higher catch chance.
- Uses random values (rand() % 100) to determine whether the attempt succeeds.
- Guarantees that taming is neither too easy nor impossible — improving game balance.
- Makes game progression strategic by encouraging the player to weaken monsters before capturing.

4. File Save/Load Algorithm

- Player data (HP, level, monster team) is written to a text file using fprintf() line by line.
- When loading, the file is parsed back into the correct structure using fscanf().
- Implements basic validation to handle missing or corrupted save files.
- Ensures persistence of game progress between sessions, fulfilling project guidelines.

5. Menu Interaction Loop Algorithm

- Uses a structured loop (while(1) { ... }) to repeatedly display menu options until the user chooses exit.
- Validates user input to prevent invalid choices or program crashes.
- Allows smooth transitions between "Explore", "Battle", "Team", "Save", and "Load" screens.
- Forms the backbone of the game's control flow.

6. Monster Level Scaling Algorithm

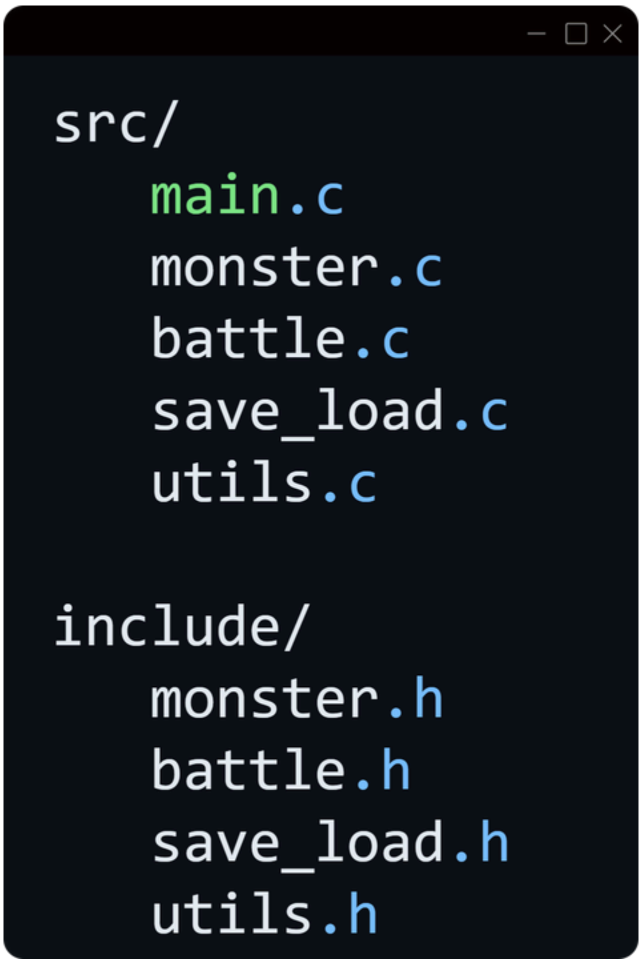
- Monster stats scale proportionally with level to maintain challenge balance.
- HP and Attack are computed using simple linear growth formulas.
- Ensures that stronger monsters give a sense of progression and difficulty curve.
- Prevents low-level monsters from overwhelming the player in later game stages.

Implementation Details

This section explains how the Monster Tamer RPG game was developed using modular programming concepts in C

1. Modular Project Structure

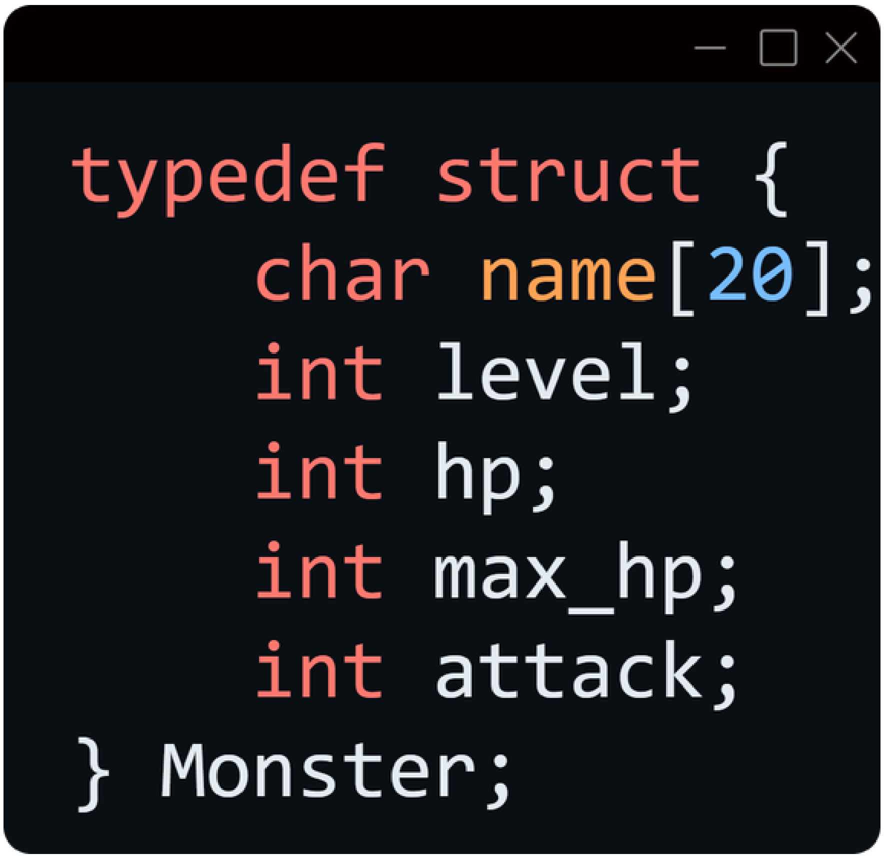
The project is organized into separate C source files and header files to enhance clarity, reusability, and maintainability. Each module focuses on a single responsibility, following best practices for structured programming.



```
src/
    main.c
    monster.c
    battle.c
    save_load.c
    utils.c

include/
    monster.h
    battle.h
    save_load.h
    utils.h
```

2. Monster Representation Using Structures

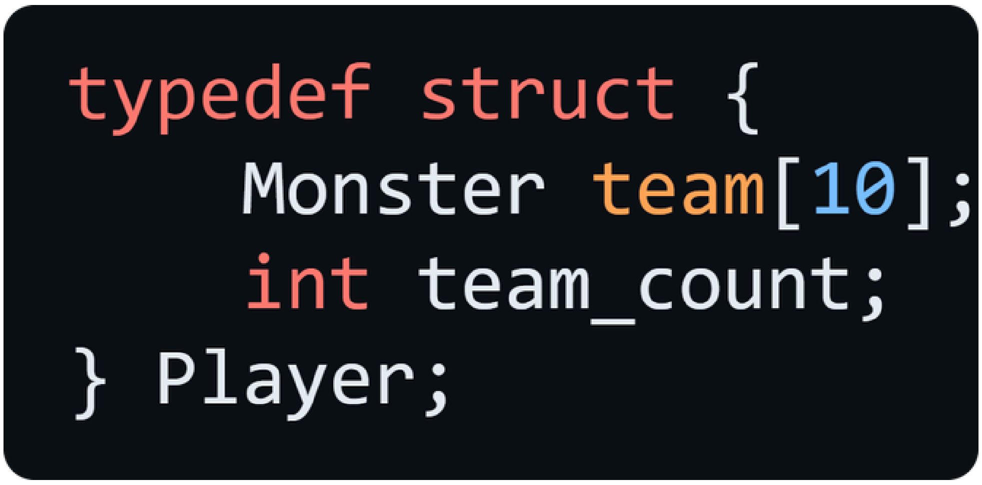
Monsters are implemented using a struct that stores essential attributes such as name, HP, level, and attack power. This allows multiple monsters to be handled easily through arrays and functions.



```c
typedef struct {
    char name[20];
    int level;
    int hp;
    int max_hp;
    int attack;
} Monster;
```

3. Player and Team System

The player maintains a team of captured monsters, stored in a structure containing an array of monsters



```c
typedef struct {
    Monster team[10];
    int team_count;
} Player;
```

4. Turn-Based Battle System

The battle system operates on a simple turn loop, where the player chooses an action and the monster responds. This continues until one side's HP reaches zero.

```c
while (player->team[0].hp > 0 && wild.hp > 0) {
    printf("1. Attack\n2. Tame\n3. Run\n");
    scanf("%d", &choice);

    if (choice == 1) {
        wild.hp -= player->team[0].attack;
    }

    if (wild.hp > 0) {
        player->team[0].hp -= wild.attack;
    }
}
```

6. Monster Taming Mechanism

Taming relies on a probability algorithm that considers the monster's remaining HP. The lower the HP, the higher the success chance.

```c
int chance = (50 + (wild.max_hp - wild.hp));
if ((rand() % 100) < chance) {
    printf("Monster captured!\n");
}
```

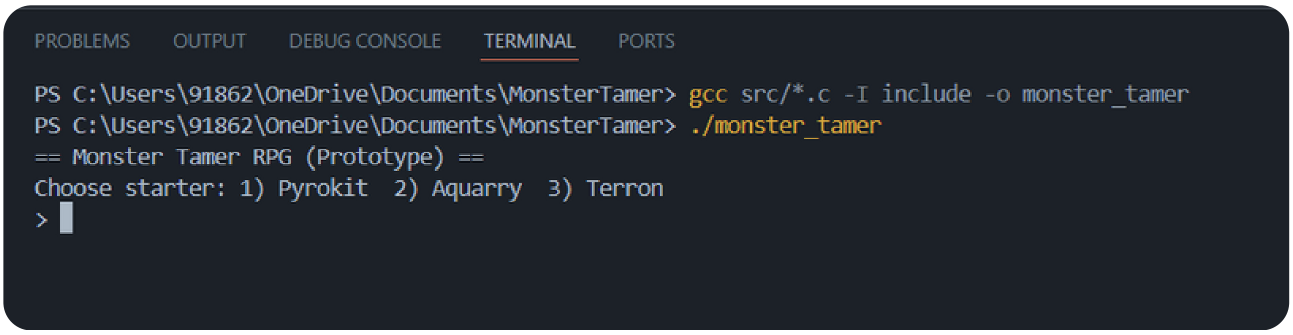
This adds strategy to gameplay by encouraging the player to weaken monsters before attempting to capture them.

Testing & Results

To make sure the Monster Tamer RPG worked correctly, I tested every major feature of the game through both normal gameplay and intentional error creation. My main goal was to confirm that the game behaves correctly, does not crash, and responds properly to different types of inputs.

1. Compilation Testing:

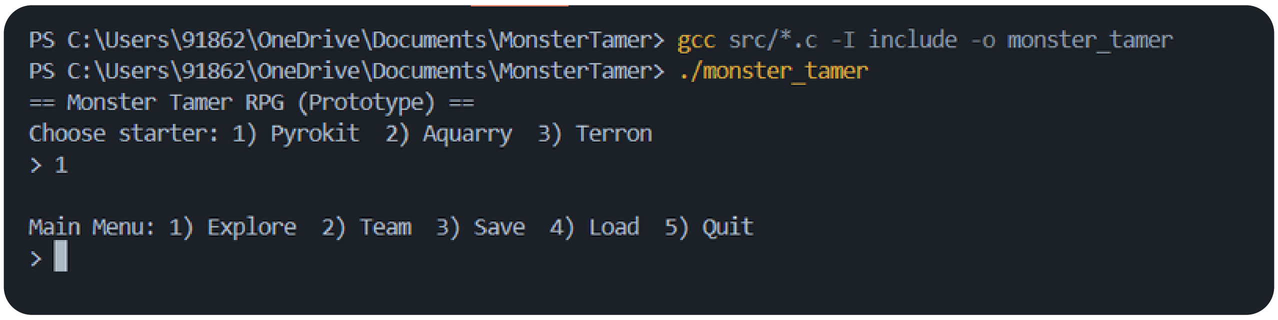
This adds strategy to gameplay by encouraging the player to weaken monsters before attempting to capture them.



```
PS C:\Users\91862\OneDrive\Documents\MonsterTamer> gcc src/*.c -I include -o monster_tamer
PS C:\Users\91862\OneDrive\Documents\MonsterTamer> ./monster_tamer
== Monster Tamer RPG (Prototype) ==
Choose starter: 1) Pyrokit  2) Aquarry  3) Terron
> 
```

2. Menu & Navigation Testing

I tested each option in the main menu to make sure the program responded correctly.



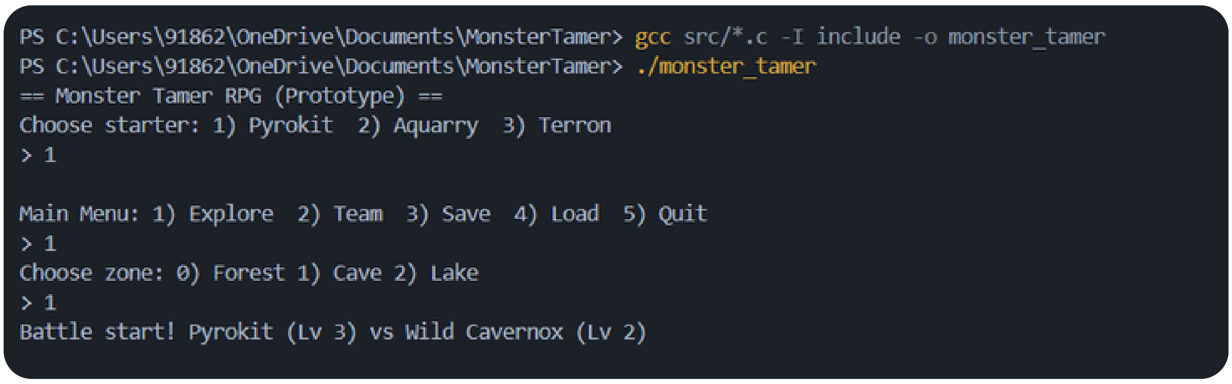
```
PS C:\Users\91862\OneDrive\Documents\MonsterTamer> gcc src/*.c -I include -o monster_tamer
PS C:\Users\91862\OneDrive\Documents\MonsterTamer> ./monster_tamer
== Monster Tamer RPG (Prototype) ==
Choose starter: 1) Pyrokit  2) Aquarry  3) Terron
> 1

Main Menu: 1) Explore  2) Team  3) Save  4) Load  5) Quit
> 
```

- Menu did not break
- Invalid inputs did not crash the game
- Each option returned to the main menu correctly

4. Exploration & Random Encounter Testing

I tested each option in the main menu to make sure the program responded correctly.

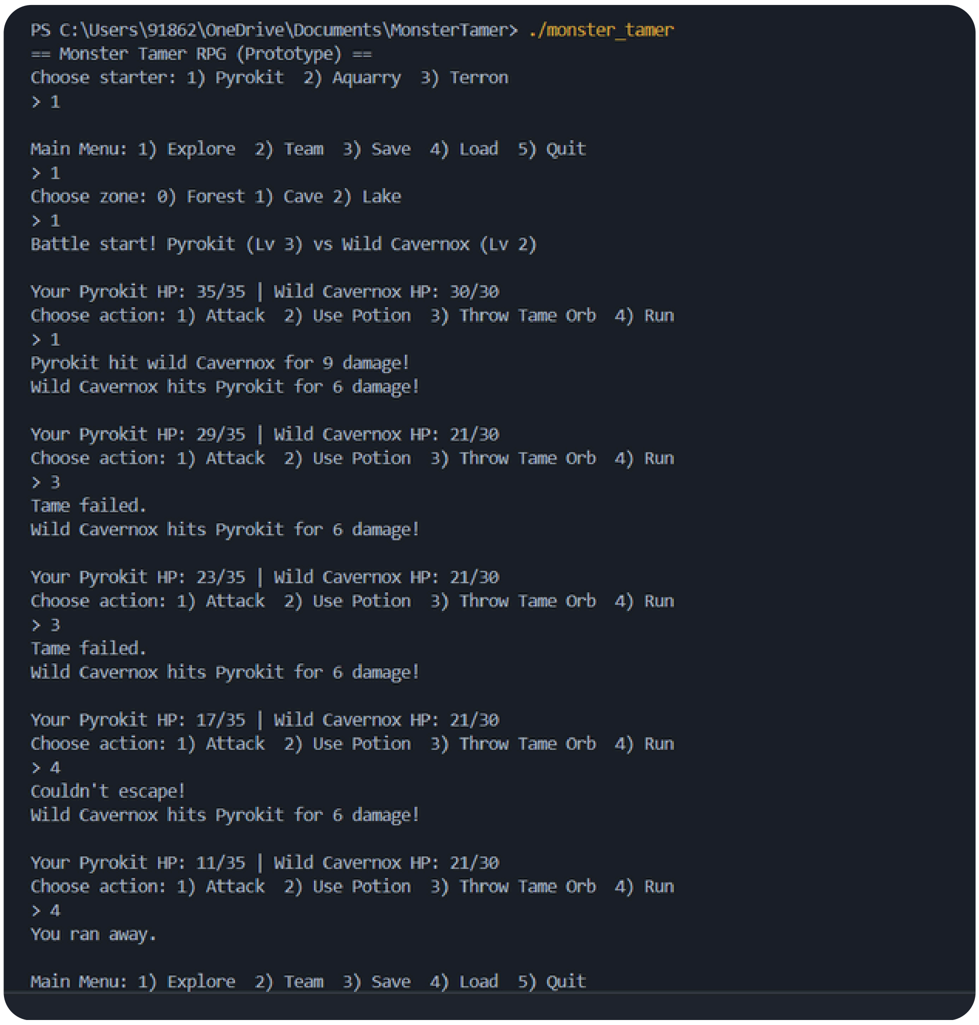
- Entered "Explore" multiple times
- Tested different zones
- Observed random monster generation



```
PS C:\Users\91862\OneDrive\Documents\MonsterTamer> gcc src/*.c -I include -o monster_tamer
PS C:\Users\91862\OneDrive\Documents\MonsterTamer> ./monster_tamer
== Monster Tamer RPG (Prototype) ==
Choose starter: 1) Pyrokit  2) Aquarry  3) Terron
> 1

Main Menu: 1) Explore  2) Team  3) Save  4) Load  5) Quit
> 1
Choose zone: 0) Forest 1) Cave 2) Lake
> 1
Battle start! Pyrokit (Lv 3) vs Wild Cavernox (Lv 2)
```

5. Battle System Testing

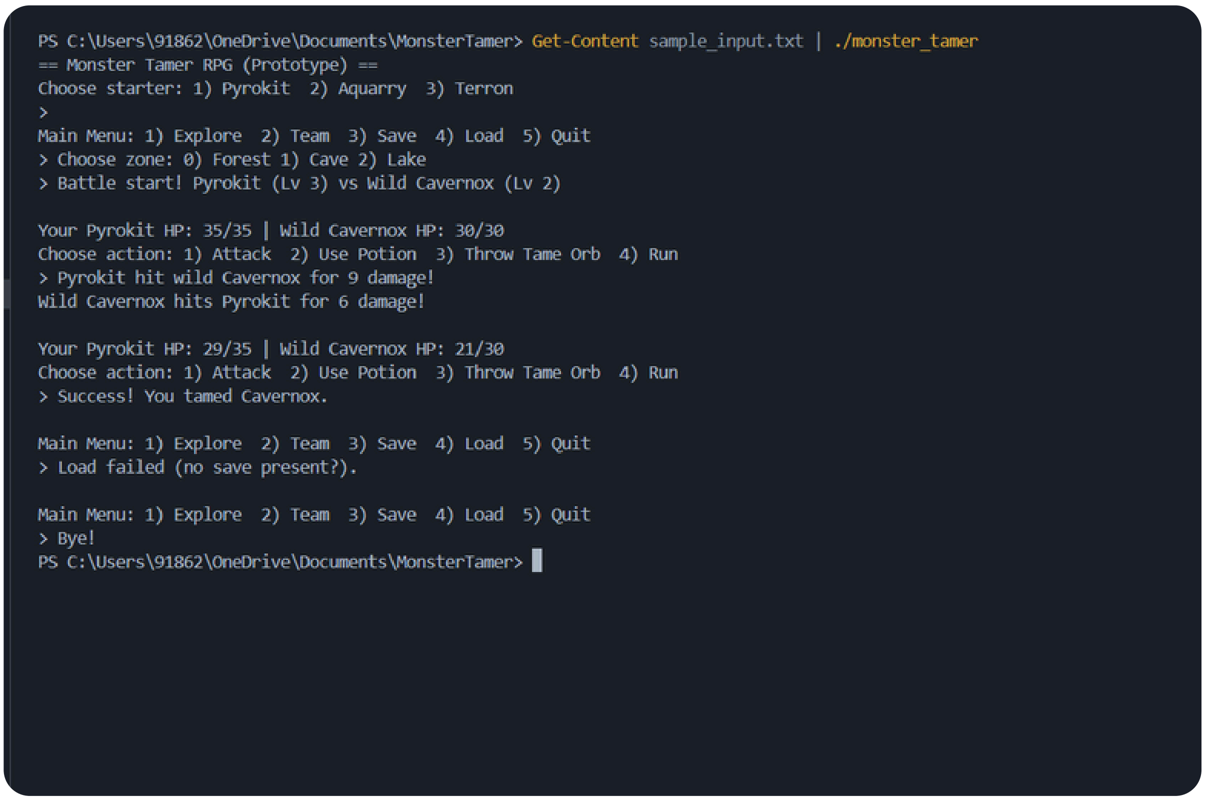
- Fought multiple wild monsters
- Tested Attack, Tame, and Run options
- Let both player and enemy attack to reduce HP



```
PS C:\Users\91862\OneDrive\Documents\MonsterTamer> ./monster_tamer
== Monster Tamer RPG (Prototype) ==
Choose starter: 1) Pyrokit  2) Aquarry  3) Terron
> 1

Main Menu: 1) Explore  2) Team  3) Save  4) Load  5) Quit
> 1
Choose zone: 0) Forest 1) Cave 2) Lake
> 1
Battle start! Pyrokit (Lv 3) vs Wild Cavernox (Lv 2)

Your Pyrokit HP: 35/35 | Wild Cavernox HP: 30/30
Choose action: 1) Attack  2) Use Potion  3) Throw Tame Orb  4) Run
> 1
Pyrokit hit wild Cavernox for 9 damage!
Wild Cavernox hits Pyrokit for 6 damage!

Your Pyrokit HP: 29/35 | Wild Cavernox HP: 21/30
Choose action: 1) Attack  2) Use Potion  3) Throw Tame Orb  4) Run
> 3
Tame failed.
Wild Cavernox hits Pyrokit for 6 damage!

Your Pyrokit HP: 23/35 | Wild Cavernox HP: 21/30
Choose action: 1) Attack  2) Use Potion  3) Throw Tame Orb  4) Run
> 3
Tame failed.
Wild Cavernox hits Pyrokit for 6 damage!

Your Pyrokit HP: 17/35 | Wild Cavernox HP: 21/30
Choose action: 1) Attack  2) Use Potion  3) Throw Tame Orb  4) Run
> 4
Couldn't escape!
Wild Cavernox hits Pyrokit for 6 damage!

Your Pyrokit HP: 11/35 | Wild Cavernox HP: 21/30
Choose action: 1) Attack  2) Use Potion  3) Throw Tame Orb  4) Run
> 4
You ran away.

Main Menu: 1) Explore  2) Team  3) Save  4) Load  5) Quit
```

6. Automated Input Testing

- Created a sample_input.txt file
- Ran the program with redirected input:



```
PS C:\Users\91862\OneDrive\Documents\MonsterTamer> Get-Content sample_input.txt | ./monster_tamer
== Monster Tamer RPG (Prototype) ==
Choose starter: 1) Pyrokit  2) Aquarry  3) Terron
>
Main Menu: 1) Explore  2) Team  3) Save  4) Load  5) Quit
> Choose zone: 0) Forest 1) Cave 2) Lake
> Battle start! Pyrokit (Lv 3) vs Wild Cavernox (Lv 2)

Your Pyrokit HP: 35/35 | Wild Cavernox HP: 30/30
Choose action: 1) Attack  2) Use Potion  3) Throw Tame Orb  4) Run
> Pyrokit hit wild Cavernox for 9 damage!
Wild Cavernox hits Pyrokit for 6 damage!

Your Pyrokit HP: 29/35 | Wild Cavernox HP: 21/30
Choose action: 1) Attack  2) Use Potion  3) Throw Tame Orb  4) Run
> Success! You tamed Cavernox.

Main Menu: 1) Explore  2) Team  3) Save  4) Load  5) Quit
> Load failed (no save present?).

Main Menu: 1) Explore  2) Team  3) Save  4) Load  5) Quit
> Bye!
PS C:\Users\91862\OneDrive\Documents\MonsterTamer>
```

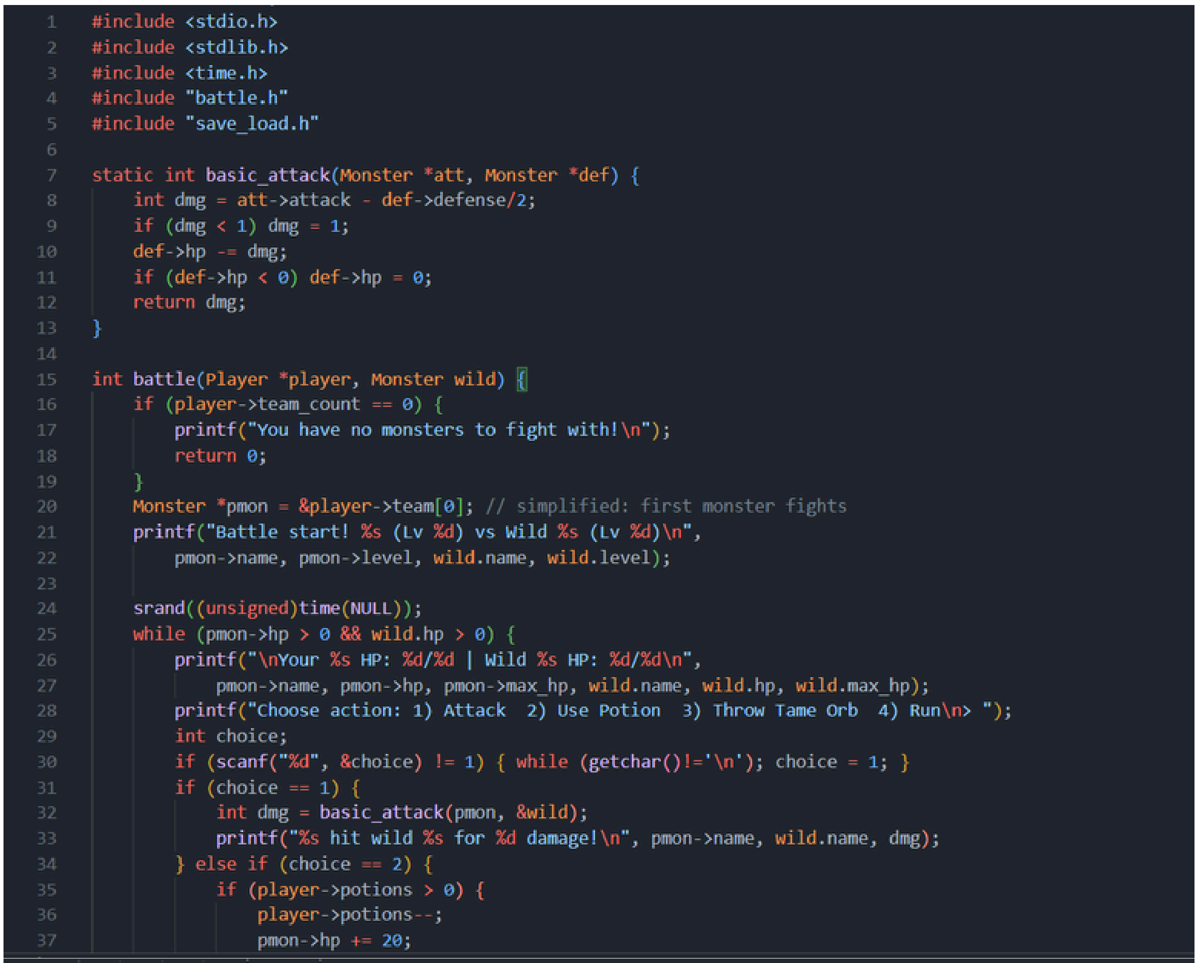
What I tested:

- Starter selection
- Exploring
- Battle
- Save / Load
- Exit

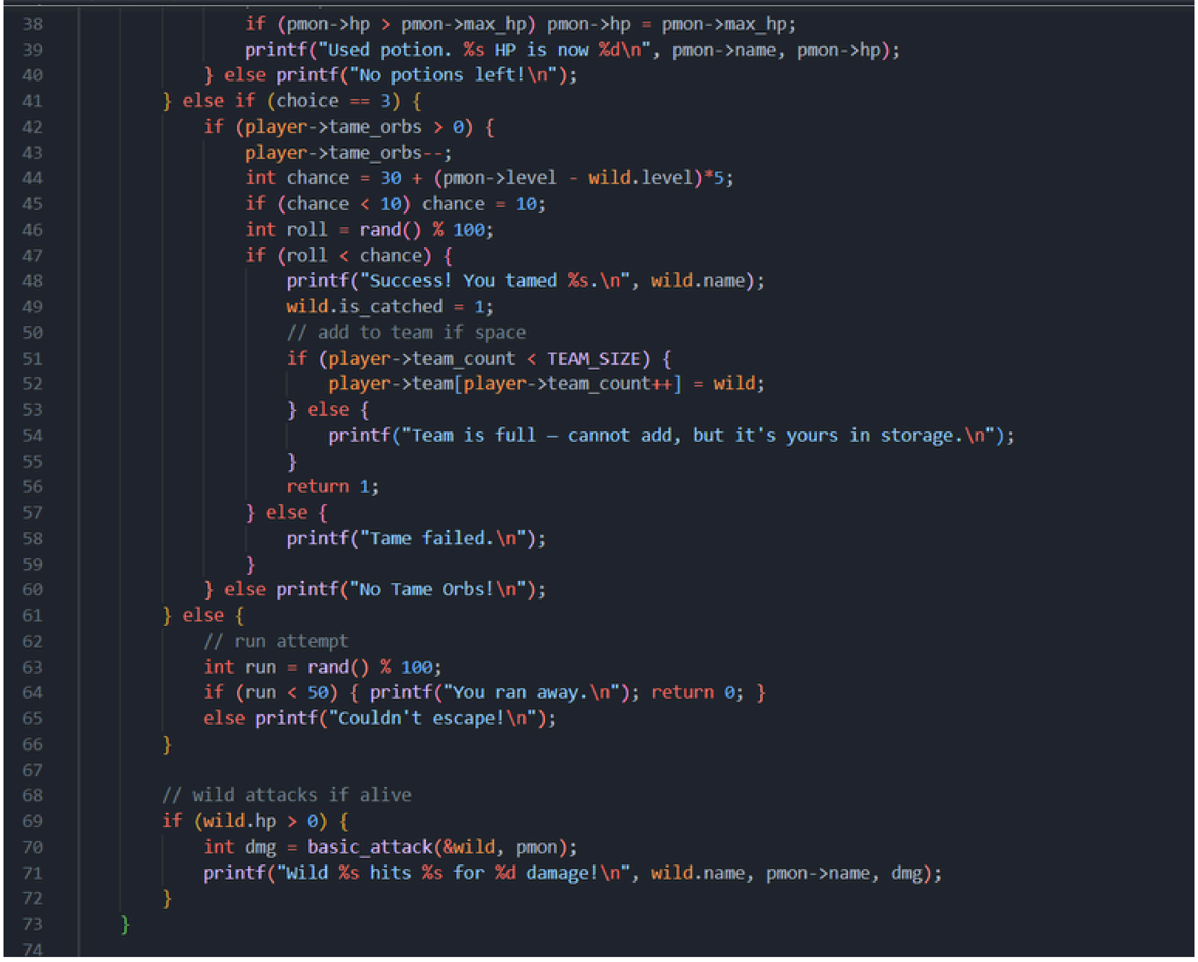
Result:

The game completed all actions without crashing.

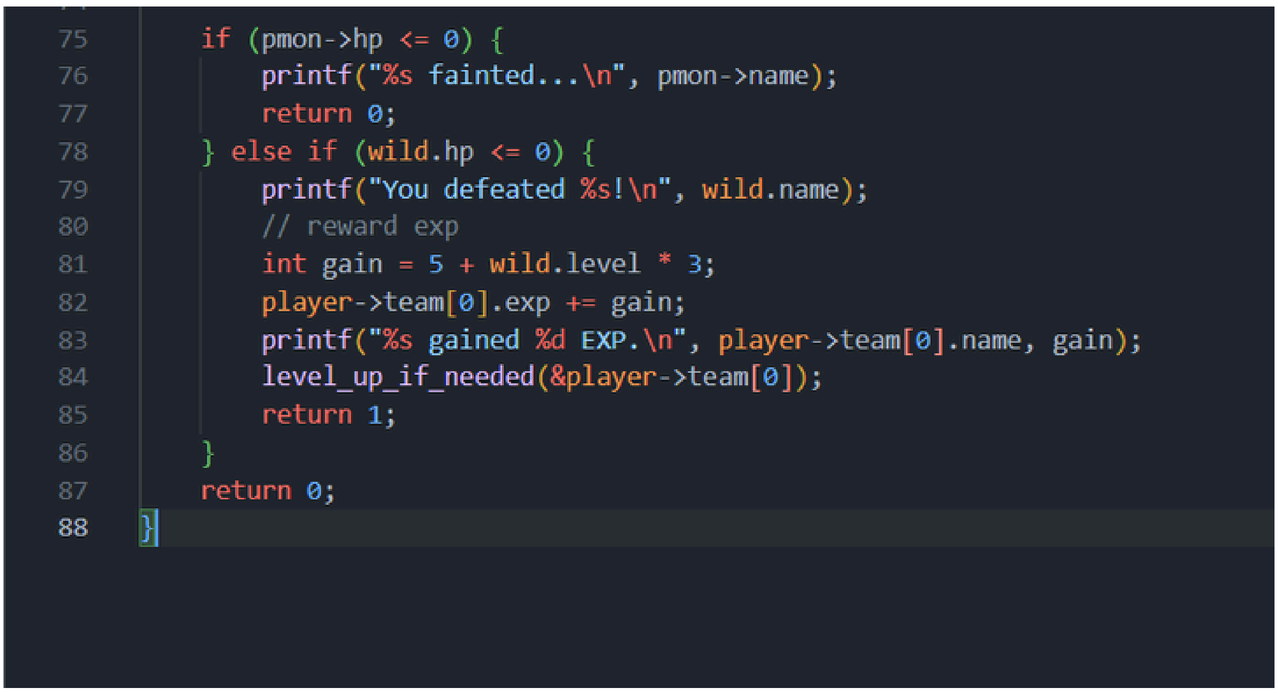
battle.c



```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "battle.h"
#include "save_load.h"

static int basic_attack(Monster *att, Monster *def) {
    int dmg = att->attack - def->defense/2;
    if (dmg < 1) dmg = 1;
    def->hp -= dmg;
    if (def->hp < 0) def->hp = 0;
    return dmg;
}

int battle(Player *player, Monster wild) {
    if (player->team_count == 0) {
        printf("You have no monsters to fight with!\n");
        return 0;
    }
    Monster *pmon = &player->team[0]; // simplified: first monster fights
    printf("Battle start! %s (Lv %d) vs Wild %s (Lv %d)\n",
        pmon->name, pmon->level, wild.name, wild.level);

    srand((unsigned)time(NULL));
    while (pmon->hp > 0 && wild.hp > 0) {
        printf("\nYour %s HP: %d/%d | Wild %s HP: %d/%d\n",
            pmon->name, pmon->hp, pmon->max_hp, wild.name, wild.hp, wild.max_hp);
        printf("Choose action: 1) Attack  2) Use Potion  3) Throw Tame Orb  4) Run\n> ");
        int choice;
        if (scanf("%d", &choice) != 1) { while (getchar()!='\n'); choice = 1; }
        if (choice == 1) {
            int dmg = basic_attack(pmon, &wild);
            printf("%s hit wild %s for %d damage!\n", pmon->name, wild.name, dmg);
        } else if (choice == 2) {
            if (player->potions > 0) {
                player->potions--;
                pmon->hp += 20;
```

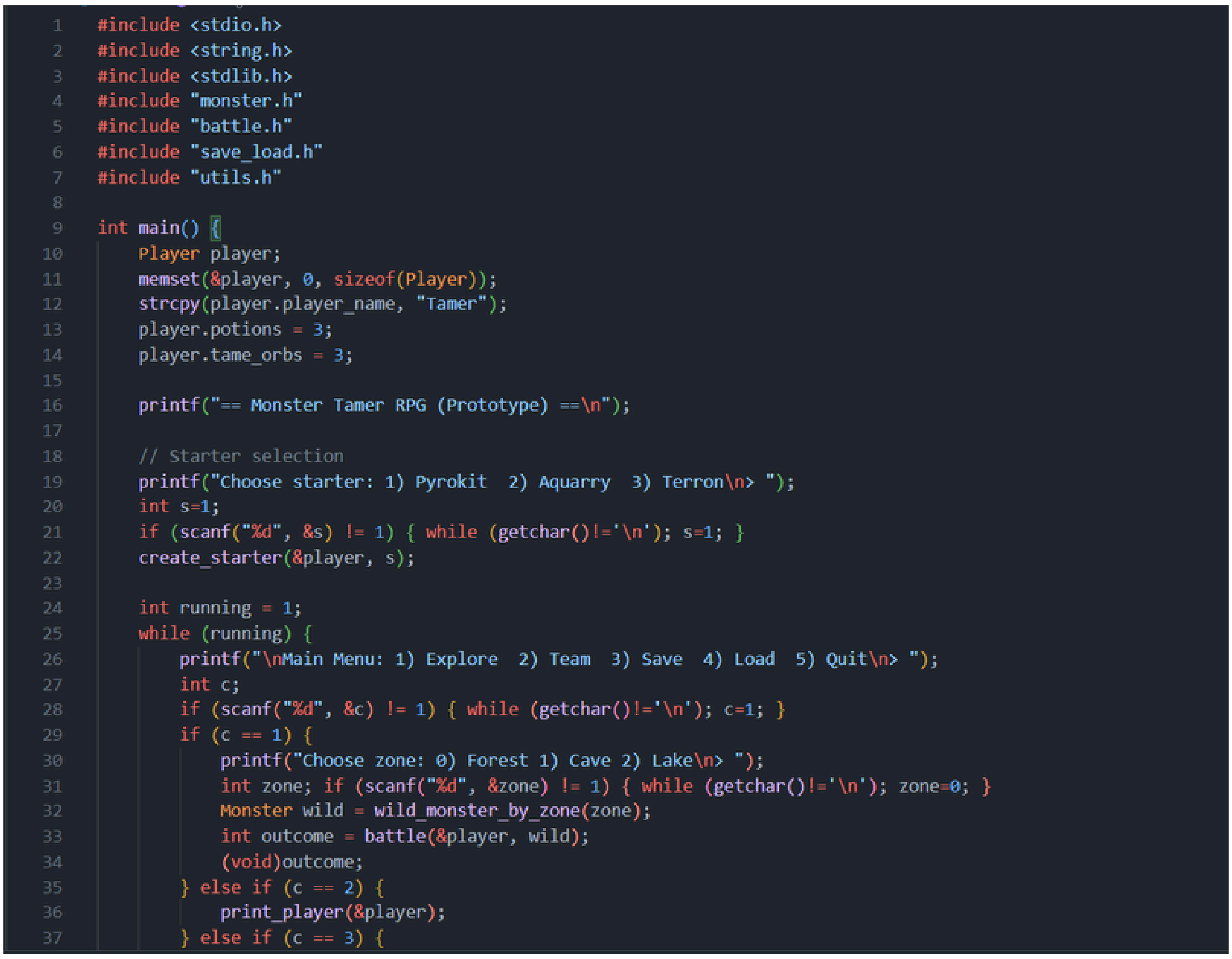


```c
                if (pmon->hp > pmon->max_hp) pmon->hp = pmon->max_hp;
                printf("Used potion. %s HP is now %d\n", pmon->name, pmon->hp);
            } else printf("No potions left!\n");
        } else if (choice == 3) {
            if (player->tame_orbs > 0) {
                player->tame_orbs--;
                int chance = 30 + (pmon->level - wild.level)*5;
                if (chance < 10) chance = 10;
                int roll = rand() % 100;
                if (roll < chance) {
                    printf("Success! You tamed %s.\n", wild.name);
                    wild.is_catched = 1;
                    // add to team if space
                    if (player->team_count < TEAM_SIZE) {
                        player->team[player->team_count++] = wild;
                    } else {
                        printf("Team is full — cannot add, but it's yours in storage.\n");
                    }
                    return 1;
                } else {
                    printf("Tame failed.\n");
                }
            } else printf("No Tame Orbs!\n");
        } else {
            // run attempt
            int run = rand() % 100;
            if (run < 50) { printf("You ran away.\n"); return 0; }
            else printf("Couldn't escape!\n");
        }

        // wild attacks if alive
        if (wild.hp > 0) {
            int dmg = basic_attack(&wild, pmon);
            printf("Wild %s hits %s for %d damage!\n", wild.name, pmon->name, dmg);
        }
    }

```

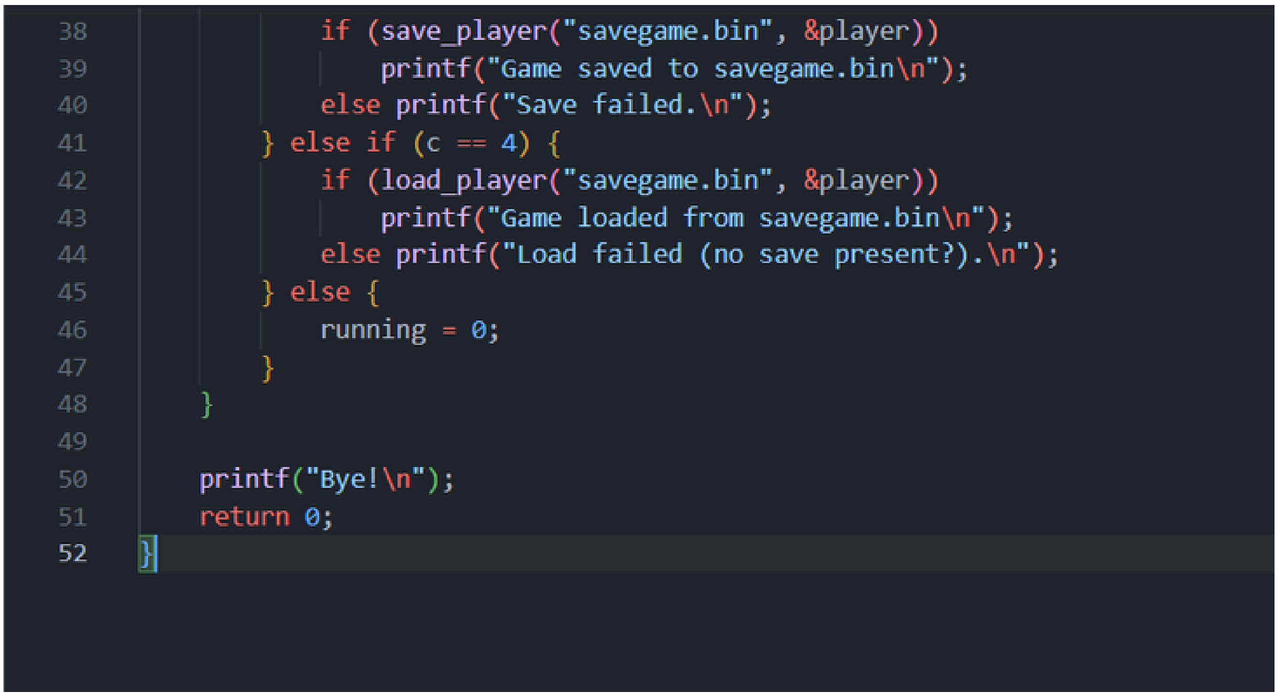


```c
    if (pmon->hp <= 0) {
        printf("%s fainted...\n", pmon->name);
        return 0;
    } else if (wild.hp <= 0) {
        printf("You defeated %s!\n", wild.name);
        // reward exp
        int gain = 5 + wild.level * 3;
        player->team[0].exp += gain;
        printf("%s gained %d EXP.\n", player->team[0].name, gain);
        level_up_if_needed(&player->team[0]);
        return 1;
    }
    return 0;
}
```

main.c

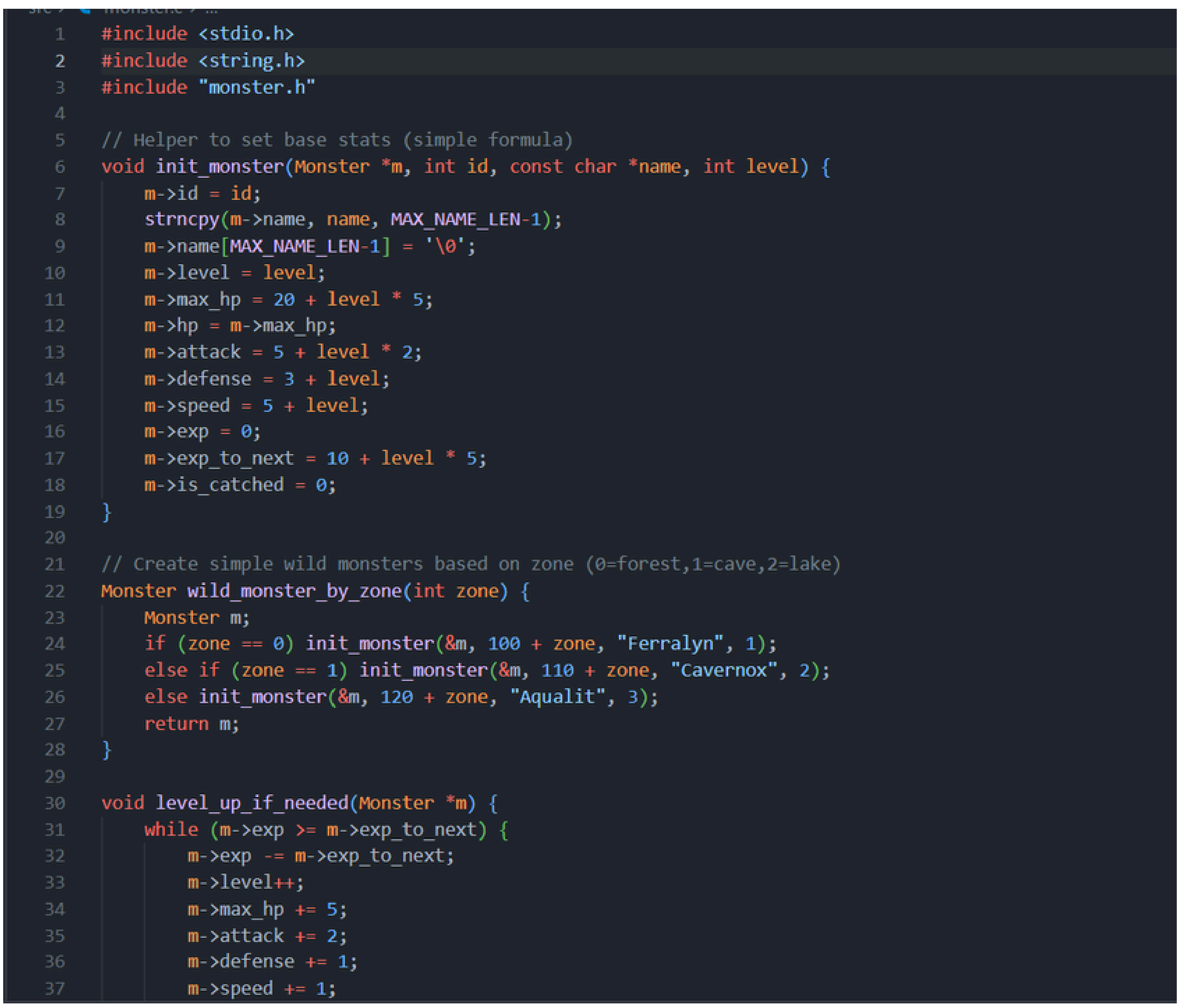


```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "monster.h"
#include "battle.h"
#include "save_load.h"
#include "utils.h"

int main() {
    Player player;
    memset(&player, 0, sizeof(Player));
    strcpy(player.player_name, "Tamer");
    player.potions = 3;
    player.tame_orbs = 3;

    printf("== Monster Tamer RPG (Prototype) ==\n");

    // Starter selection
    printf("Choose starter: 1) Pyrokit  2) Aquarry  3) Terron\n> ");
    int s=1;
    if (scanf("%d", &s) != 1) { while (getchar()!='\n'); s=1; }
    create_starter(&player, s);

    int running = 1;
    while (running) {
        printf("\nMain Menu: 1) Explore  2) Team  3) Save  4) Load  5) Quit\n> ");
        int c;
        if (scanf("%d", &c) != 1) { while (getchar()!='\n'); c=1; }
        if (c == 1) {
            printf("Choose zone: 0) Forest 1) Cave 2) Lake\n> ");
            int zone; if (scanf("%d", &zone) != 1) { while (getchar()!='\n'); zone=0; }
            Monster wild = wild_monster_by_zone(zone);
            int outcome = battle(&player, wild);
            (void)outcome;
        } else if (c == 2) {
            print_player(&player);
        } else if (c == 3) {
```

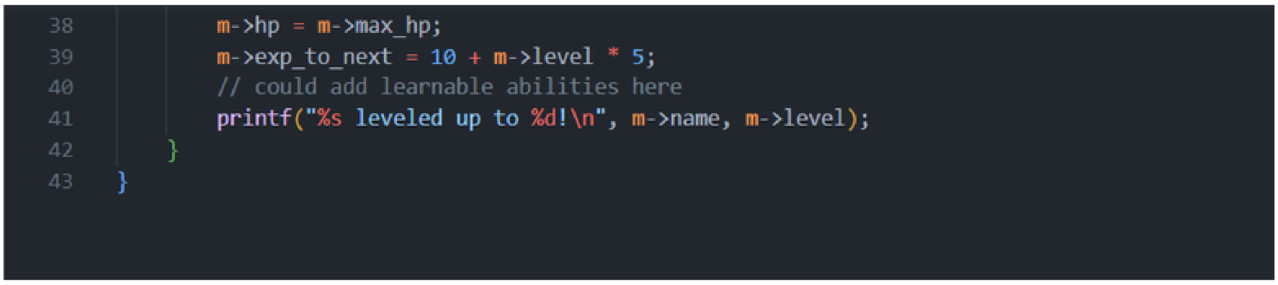


```c
                if (save_player("savegame.bin", &player))
                    printf("Game saved to savegame.bin\n");
                else printf("Save failed.\n");
            } else if (c == 4) {
                if (load_player("savegame.bin", &player))
                    printf("Game loaded from savegame.bin\n");
                else printf("Load failed (no save present?).\n");
            } else {
                running = 0;
            }
        }

    printf("Bye!\n");
    return 0;
}
```

monster.c

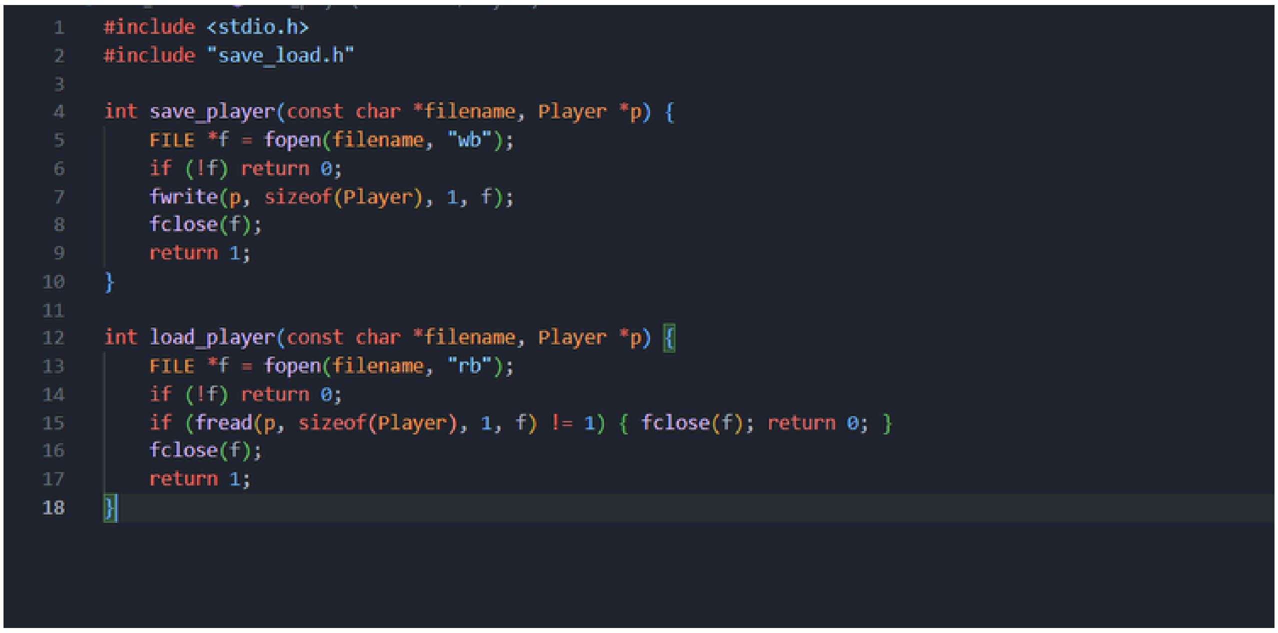


```c
#include <stdio.h>
#include <string.h>
#include "monster.h"

// Helper to set base stats (simple formula)
void init_monster(Monster *m, int id, const char *name, int level) {
    m->id = id;
    strncpy(m->name, name, MAX_NAME_LEN-1);
    m->name[MAX_NAME_LEN-1] = '\0';
    m->level = level;
    m->max_hp = 20 + level * 5;
    m->hp = m->max_hp;
    m->attack = 5 + level * 2;
    m->defense = 3 + level;
    m->speed = 5 + level;
    m->exp = 0;
    m->exp_to_next = 10 + level * 5;
    m->is_catched = 0;
}

// Create simple wild monsters based on zone (0=forest,1=cave,2=lake)
Monster wild_monster_by_zone(int zone) {
    Monster m;
    if (zone == 0) init_monster(&m, 100 + zone, "Ferralyn", 1);
    else if (zone == 1) init_monster(&m, 110 + zone, "Cavernox", 2);
    else init_monster(&m, 120 + zone, "Aqualit", 3);
    return m;
}

void level_up_if_needed(Monster *m) {
    while (m->exp >= m->exp_to_next) {
        m->exp -= m->exp_to_next;
        m->level++;
        m->max_hp += 5;
        m->attack += 2;
        m->defense += 1;
        m->speed += 1;
```



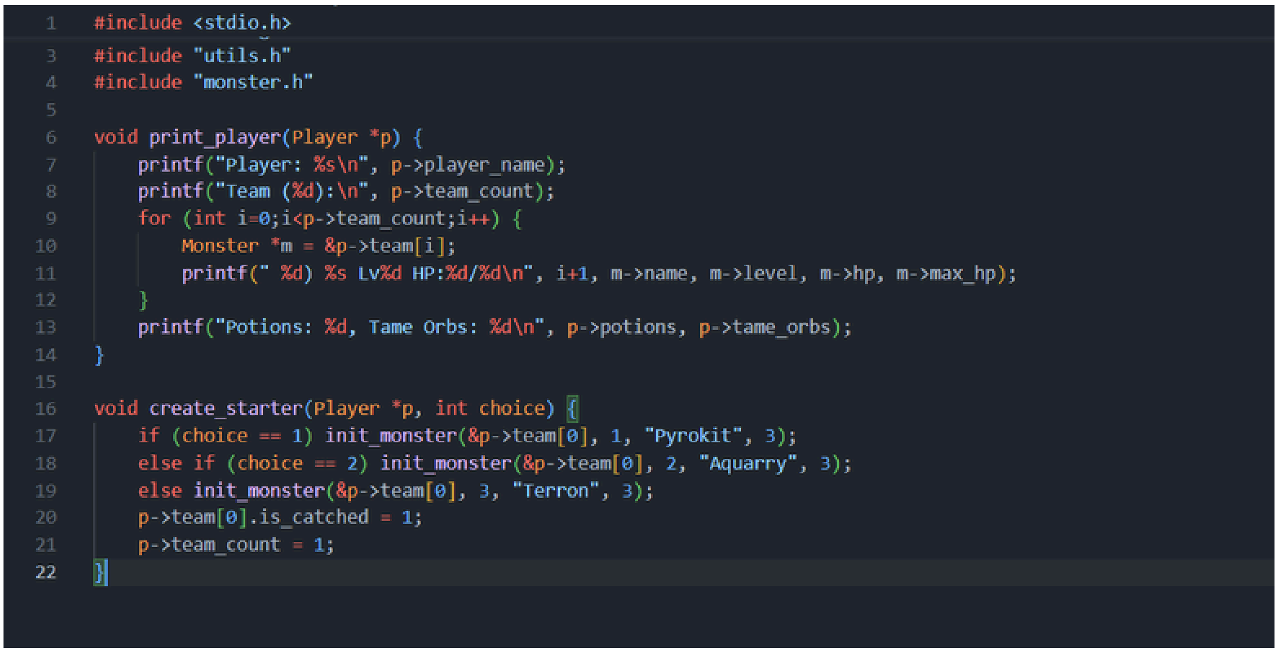
```c
        m->hp = m->max_hp;
        m->exp_to_next = 10 + m->level * 5;
        // could add learnable abilities here
        printf("%s leveled up to %d!\n", m->name, m->level);
    }
}
```

save_load.c



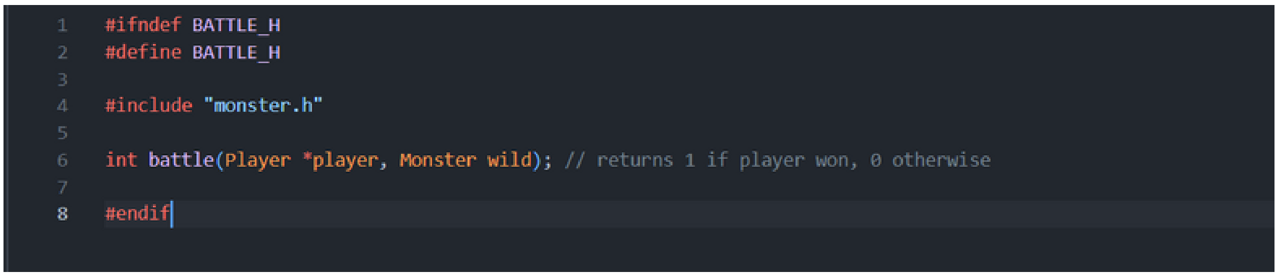
```c
#include <stdio.h>
#include "save_load.h"

int save_player(const char *filename, Player *p) {
    FILE *f = fopen(filename, "wb");
    if (!f) return 0;
    fwrite(p, sizeof(Player), 1, f);
    fclose(f);
    return 1;
}

int load_player(const char *filename, Player *p) {
    FILE *f = fopen(filename, "rb");
    if (!f) return 0;
    if (fread(p, sizeof(Player), 1, f) != 1) { fclose(f); return 0; }
    fclose(f);
    return 1;
}
```

utils.c



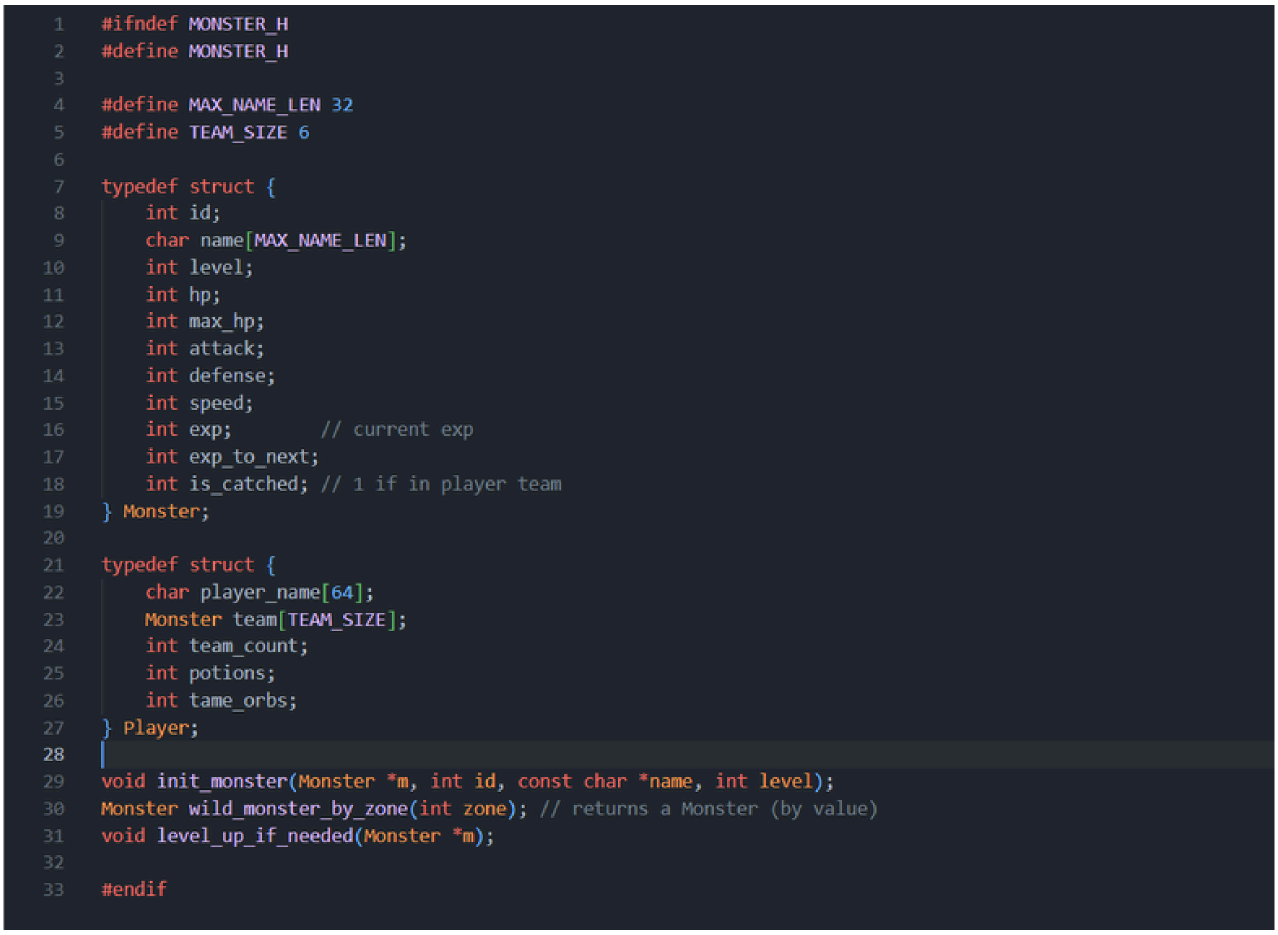
```c
#include <stdio.h>
#include "utils.h"
#include "monster.h"

void print_player(Player *p) {
    printf("Player: %s\n", p->player_name);
    printf("Team (%d):\n", p->team_count);
    for (int i=0;i<p->team_count;i++) {
        Monster *m = &p->team[i];
        printf(" %d) %s Lv%d HP:%d/%d\n", i+1, m->name, m->level, m->hp, m->max_hp);
    }
    printf("Potions: %d, Tame Orbs: %d\n", p->potions, p->tame_orbs);
}

void create_starter(Player *p, int choice) {
    if (choice == 1) init_monster(&p->team[0], 1, "Pyrokit", 3);
    else if (choice == 2) init_monster(&p->team[0], 2, "Aquarry", 3);
    else init_monster(&p->team[0], 3, "Terron", 3);
    p->team[0].is_catched = 1;
    p->team_count = 1;
}
```

battle.h



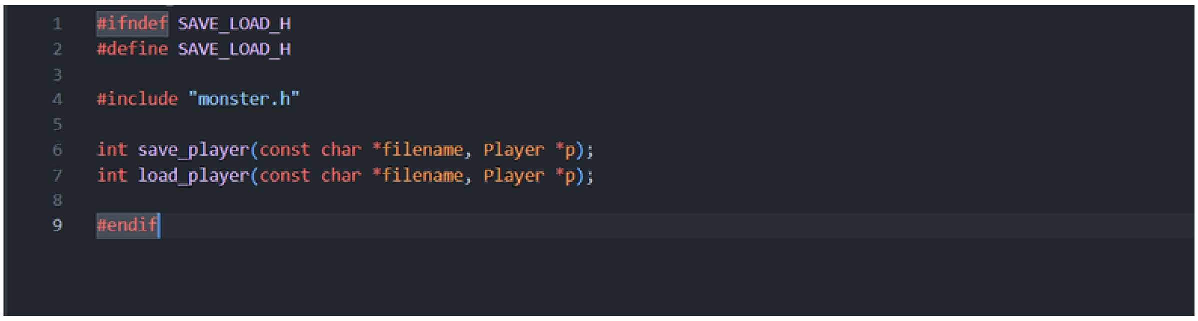
```c
#ifndef BATTLE_H
#define BATTLE_H

#include "monster.h"

int battle(Player *player, Monster wild); // returns 1 if player won, 0 otherwise

#endif
```

monster.h



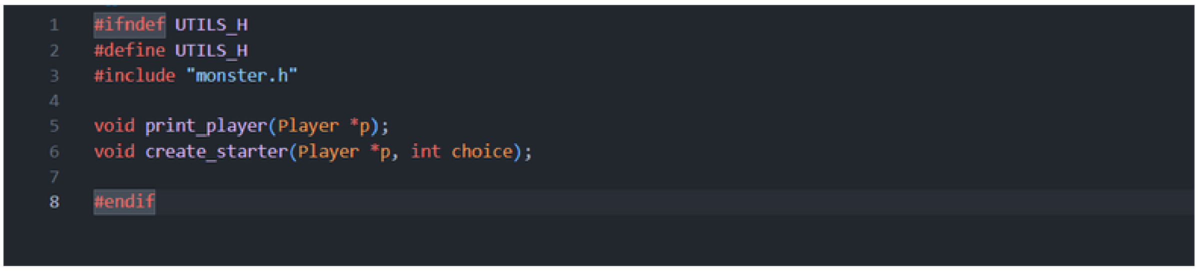
```c
#ifndef MONSTER_H
#define MONSTER_H

#define MAX_NAME_LEN 32
#define TEAM_SIZE 6

typedef struct {
    int id;
    char name[MAX_NAME_LEN];
    int level;
    int hp;
    int max_hp;
    int attack;
    int defense;
    int speed;
    int exp;        // current exp
    int exp_to_next;
    int is_catched; // 1 if in player team
} Monster;

typedef struct {
    char player_name[64];
    Monster team[TEAM_SIZE];
    int team_count;
    int potions;
    int tame_orbs;
} Player;

void init_monster(Monster *m, int id, const char *name, int level);
Monster wild_monster_by_zone(int zone); // returns a Monster (by value)
void level_up_if_needed(Monster *m);

#endif
```

save_load.h



```c
#ifndef SAVE_LOAD_H
#define SAVE_LOAD_H

#include "monster.h"

int save_player(const char *filename, Player *p);
int load_player(const char *filename, Player *p);

#endif
```

utils.h



```c
#ifndef UTILS_H
#define UTILS_H
#include "monster.h"

void print_player(Player *p);
void create_starter(Player *p, int choice);

#endif
```

CONCLUSION & FUTURE WORK

Working on this Monster Tamer RPG project actually helped me understand C programming in a much more practical way. I got hands-on experience using structures, arrays, functions, loops, and file handling while building something that actually runs like a small game. I was able to make a basic system where the player can explore different areas, run into random monsters, fight them, try to capture them, and even save and load their progress. After testing different inputs and fixing a few errors, the program ran smoothly without any crashes. Overall, this project boosted my confidence in writing modular C code and helped me learn how to break a bigger idea into smaller files and features.

As for future improvements, there are a lot of things that could make the game more fun and complete. I could add more monster types or special abilities so that battles feel more interesting. The battle system itself could also be improved by adding moves, defense options, or status effects like poison or burn. Another idea is to allow multiple save files instead of just one. It would also be cool to add items like healing potions or different taming tools. In the long run, I could try making a simple GUI instead of using only the terminal. Adding more zones, quests, or story elements would also make the world feel bigger. These changes could take this simple prototype to the next level in future versions.

REFERENCES

1. Reema Thareja, Programming in C, Oxford University Press.
2. Yashavant Kanetkar, Let Us C, BPB Publications.
3. GeeksforGeeks – C Programming Basics, File Handling, and Structures
4. TutorialsPoint – C Programming Language
5. The C Standard Library Documentation (stdio.h, stdlib.h, time.h)
6. Stack Overflow Discussions – Debugging common C errors