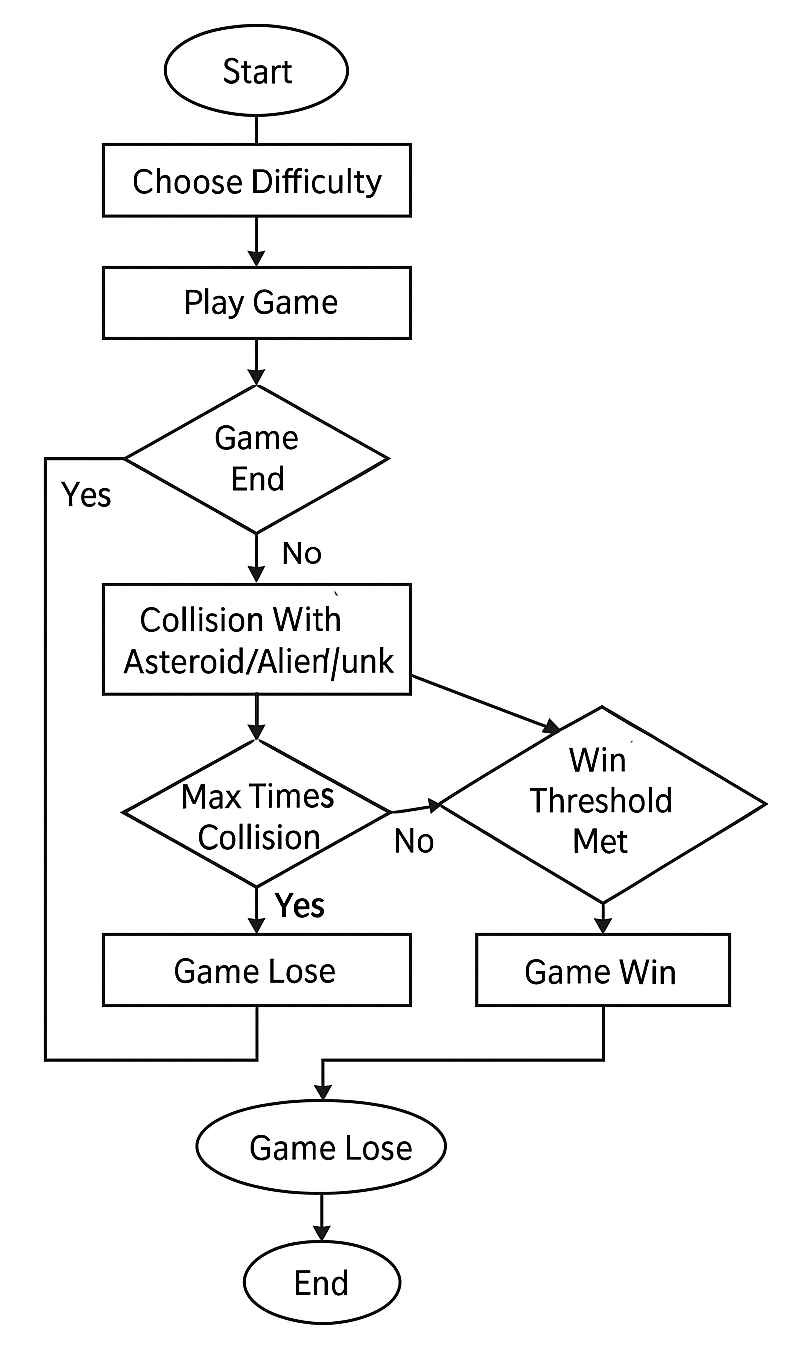


**SPACEXPLORER**

**UFMFGT-15-1 Programming for Engineers**



# **Structure Chart**



# **Source Code**

**// spacexplorer.h**

#ifndef SPACEXPLORER\_H

#define SPACEXPLORER\_H

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <ctype.h>

#define MIN\_GRID 18

#define MAX\_GRID 30

#define MAX\_JUNK 50

#define MAX\_NAME 50

#define MAX\_ALIENS 5

// Player data

typedef struct {

    int x, y;

    int fuel;

    int health;

    int junk\_collected;

    int score;

} Player;

// Asteroid structure

typedef struct {

    int x, y;

    int dx, dy;

} Asteroid;

// Junk structure

typedef struct {

    int x, y;

    int active;

} Junk;

// Alien structure

typedef struct {

    int x, y;

    int active;

} Alien;

// Game data

typedef struct {

    int grid\_size;

    int difficulty;

    char grid[MAX\_GRID][MAX\_GRID];

    Player player;

    Asteroid asteroid;

    Junk junk[MAX\_JUNK];

    int junk\_count;

    Alien aliens[MAX\_ALIENS];

    int win\_threshold;

} Game;

// Core functions

void initialize\_game(Game \*game);

void setup\_difficulty(Game \*game);

void draw\_game(Game \*game);

void move\_player(Game \*game, char input);

void move\_asteroid(Game \*game);

int check\_collision(Game \*game);

void update\_resources(Game \*game);

void check\_game\_state(Game \*game, int \*game\_over, int \*won);

void clear\_input\_buffer();

// Utility

void display\_status(Game \*game);

void show\_health(Game \*game);

void save\_score(Game \*game, char \*name);

void read\_intro();

void display\_end\_message(Game \*game, int won);

// Features

void place\_junk(Game \*game);

void place\_asteroid(Game \*game);

void recycle\_junk(Game \*game);

void handle\_alien\_encounters(Game \*game);

void spawn\_aliens(Game \*game);

// New fixes

void clamp\_fuel(Game \*game);

void handle\_fuel\_starvation(Game \*game);

void reposition\_asteroid\_after\_collision(Game \*game);

#endif // SPACEXPLORER\_H

**// game.c**

#include "spacexplorer.h"

void initialize\_game(Game \*game) {

    srand(time(NULL));

    setup\_difficulty(game);

    game->player.x = game->grid\_size / 2;

    game->player.y = game->grid\_size / 2;

    game->player.fuel = 100;

    game->player.health = 100;

    game->player.junk\_collected = 0;

    game->player.score = 0;

    game->win\_threshold = 10;

    place\_junk(game);

    place\_asteroid(game);

    spawn\_aliens(game);

}

void draw\_game(Game \*game) {

    system("cls"); // Use for Windows

    memset(game->grid, '.', sizeof(game->grid));

    game->grid[game->player.y][game->player.x] = 'P';

    for (int i = 0; i < game->junk\_count; i++) {

        if (game->junk[i].active)

            game->grid[game->junk[i].y][game->junk[i].x] = 'J';

    }

    game->grid[game->asteroid.y][game->asteroid.x] = 'X';

    for (int i = 0; i < MAX\_ALIENS; i++) {

        if (game->aliens[i].active)

            game->grid[game->aliens[i].y][game->aliens[i].x] = 'A';

    }

    for (int y = 0; y < game->grid\_size; y++) {

        for (int x = 0; x < game->grid\_size; x++) {

            printf("%c ", game->grid[y][x]);

        }

        printf("\n");

    }

}

void setup\_difficulty(Game \*game) {

    int choice;

    do {

        printf("Choose difficulty (1=Easy, 2=Medium, 3=Hard): ");

        scanf("%d", &choice);

    } while (choice < 1 || choice > 3);

    switch (choice) {

        case 1: game->grid\_size = 18; game->difficulty = 1; break;

        case 2: game->grid\_size = 24; game->difficulty = 2; break;

        case 3: game->grid\_size = 30; game->difficulty = 3; break;

    }

    game->junk\_count = MAX\_JUNK / game->difficulty;

}

void move\_player(Game \*game, char input) {

    int dx = 0, dy = 0;

    switch (input) {

        case 'W': dy = -1; break;

        case 'S': dy = 1; break;

        case 'A': dx = -1; break;

        case 'D': dx = 1; break;

    }

    int new\_x = game->player.x + dx;

    int new\_y = game->player.y + dy;

    if (new\_x >= 0 && new\_x < game->grid\_size && new\_y >= 0 && new\_y < game->grid\_size) {

        game->player.x = new\_x;

        game->player.y = new\_y;

    }

    game->player.fuel--;

    if (game->player.fuel < 0) {

        game->player.health -= 5;

    }

    for (int i = 0; i < game->junk\_count; i++) {

        if (game->junk[i].active &&

            game->junk[i].x == game->player.x &&

            game->junk[i].y == game->player.y) {

            game->junk[i].active = 0;

            game->player.junk\_collected++;

            game->player.score += 10;

        }

    }

    if (game->player.x == game->asteroid.x &&

        game->player.y == game->asteroid.y) {

        game->player.health -= 20;

        game->asteroid.x = rand() % game->grid\_size;

        game->asteroid.y = rand() % game->grid\_size;

    }

}

void move\_asteroid(Game \*game) {

    game->asteroid.x = rand() % game->grid\_size;

    game->asteroid.y = rand() % game->grid\_size;

}

void check\_game\_state(Game \*game, int \*game\_over, int \*won) {

    if (game->player.health <= 0 || game->player.fuel <= -50) {

        \*game\_over = 1;

        \*won = 0;

    } else if (game->player.junk\_collected >= game->win\_threshold) {

        \*game\_over = 1;

        \*won = 1;

    }

}

**// features.c**

#include "spacexplorer.h"

void place\_junk(Game \*game) {

    for (int i = 0; i < game->junk\_count; i++) {

        game->junk[i].x = rand() % game->grid\_size;

        game->junk[i].y = rand() % game->grid\_size;

        game->junk[i].active = 1;

    }

}

void place\_asteroid(Game \*game) {

    game->asteroid.x = rand() % game->grid\_size;

    game->asteroid.y = rand() % game->grid\_size;

}

void recycle\_junk(Game \*game) {

    if (game->player.junk\_collected > 0) {

        game->player.fuel += 10 \* game->player.junk\_collected;

        game->player.score += 5 \* game->player.junk\_collected;

        printf("Recycled %d junk into fuel and score!\n", game->player.junk\_collected);

        game->player.junk\_collected = 0;

    } else {

        printf("No junk to recycle!\n");

    }

}

void spawn\_aliens(Game \*game) {

    for (int i = 0; i < MAX\_ALIENS; i++) {

        game->aliens[i].x = rand() % game->grid\_size;

        game->aliens[i].y = rand() % game->grid\_size;

        game->aliens[i].active = 1;

    }

}

void handle\_alien\_encounters(Game \*game) {

    for (int i = 0; i < MAX\_ALIENS; i++) {

        if (game->aliens[i].active &&

            game->aliens[i].x == game->player.x &&

            game->aliens[i].y == game->player.y) {

            game->player.health -= 10;

            game->player.score += 20;

            game->aliens[i].active = 0;

            printf("You encountered an alien! Health -10, Score +20\n");

        }

    }

}

**// utils.c**

#include "spacexplorer.h"

void display\_status(Game \*game) {

    printf("Fuel: %d | Health: %d | Junk: %d | Score: %d\n",

           game->player.fuel, game->player.health,

           game->player.junk\_collected, game->player.score);

}

void show\_health(Game \*game) {

    printf("Health Check: %d\n", game->player.health);

}

void save\_score(Game \*game, char \*name) {

    FILE \*file = fopen("scores.txt", "a");

    if (file) {

        fprintf(file, "%s %d\n", name, game->player.score);

        fclose(file);

    } else {

        printf("Failed to save score.\n");

    }

}

void read\_intro() {

    FILE \*file = fopen("intro.txt", "r");

    if (file) {

        char line[256];

        while (fgets(line, sizeof(line), file)) {

            printf("%s", line);

        }

        fclose(file);

        printf("\n\n");

    } else {

        // Fallback message

        printf("Welcome to SpaceXplorer!\nCollect junk, dodge asteroids, and manage your resources!\n\n");

    }

}

void display\_end\_message(Game \*game, int won) {

    if (won) {

        printf("Congratulations! You won with a score of %d!\n", game->player.score);

    } else {

        printf("Game Over. Better luck next time.\n");

    }

}

**// main.c**

#include "spacexplorer.h"

int main() {

    Game game;

    int game\_over = 0, won = 0;

    char input;

    char name[MAX\_NAME];

    read\_intro();

    printf("Enter your name: ");

    fgets(name, MAX\_NAME, stdin);

    name[strcspn(name, "\n")] = 0;

    initialize\_game(&game);

    while (!game\_over) {

        draw\_game(&game);

        display\_status(&game);

        printf("Move (WASD), Check Health (H), Recycle (R), Quit (Q): ");

        scanf(" %c", &input);

        input = toupper(input);

        if (input == 'Q') break;

        switch (input) {

            case 'W':

            case 'A':

            case 'S':

            case 'D':

                move\_player(&game, input);

                move\_asteroid(&game);

                handle\_alien\_encounters(&game);

                break;

            case 'H':

                show\_health(&game);

                break;

            case 'R':

                recycle\_junk(&game);

                break;

            default:

                printf("Invalid input!\n");

        }

        check\_game\_state(&game, &game\_over, &won);

    }

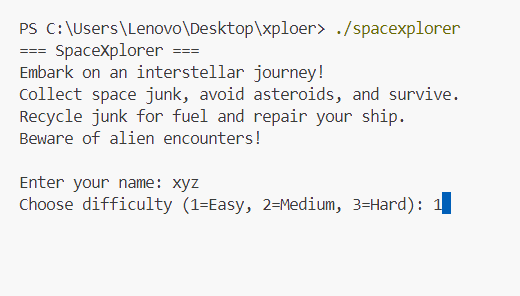
    display\_end\_message(&game, won);

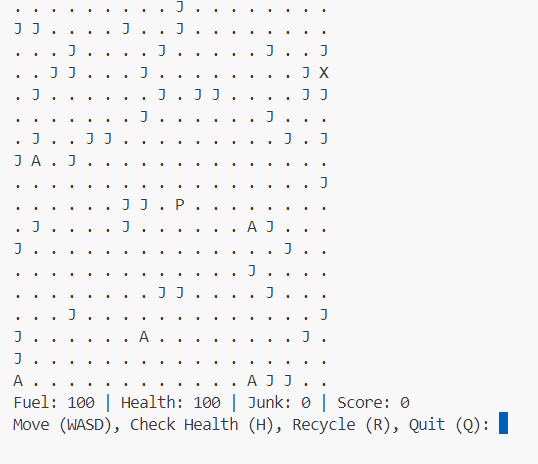
    save\_score(&game, name);

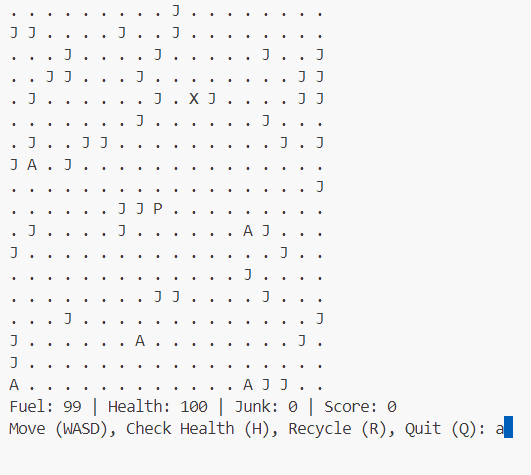
    return 0;

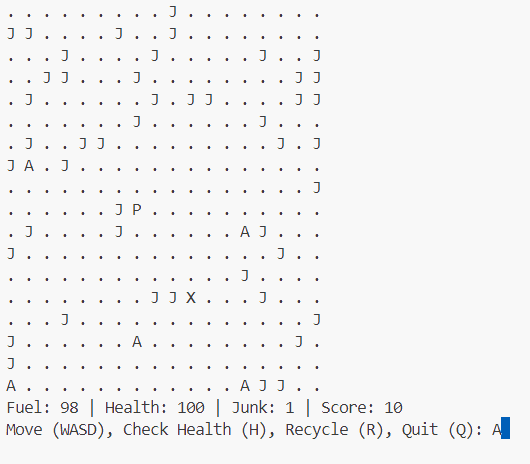
}

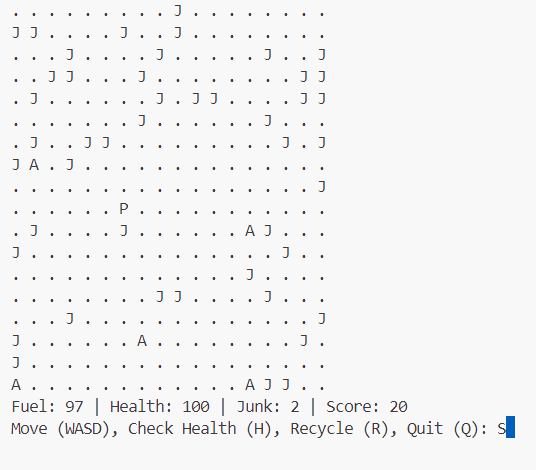
# **Output**

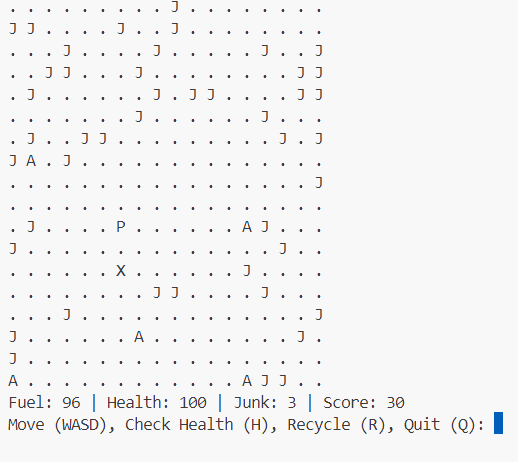


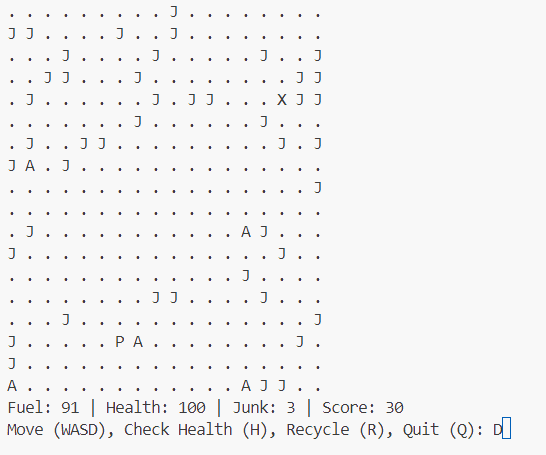


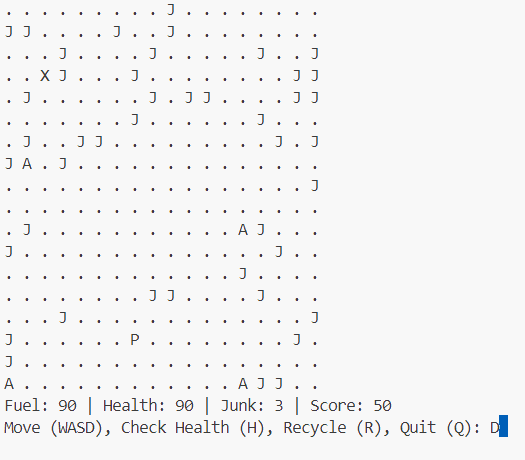


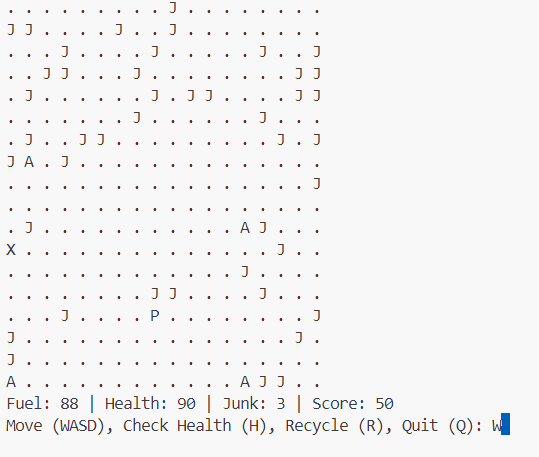


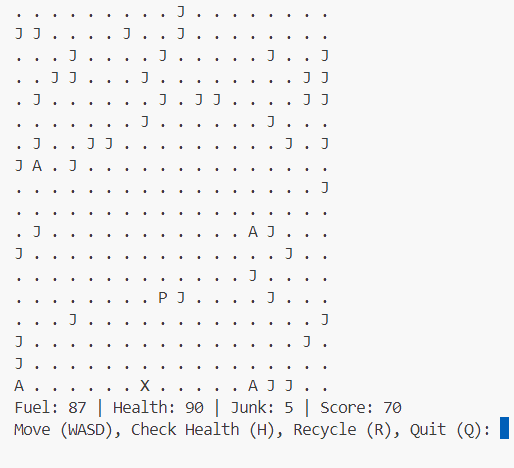


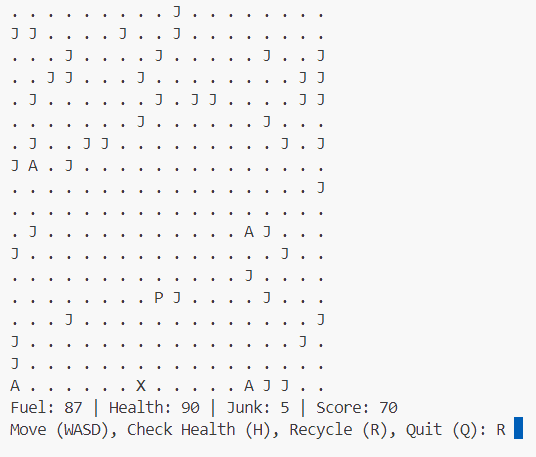


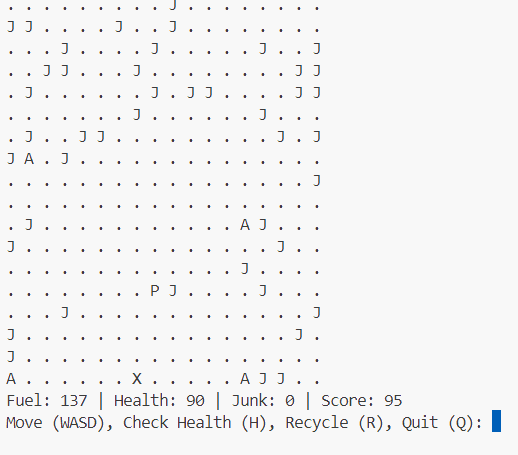


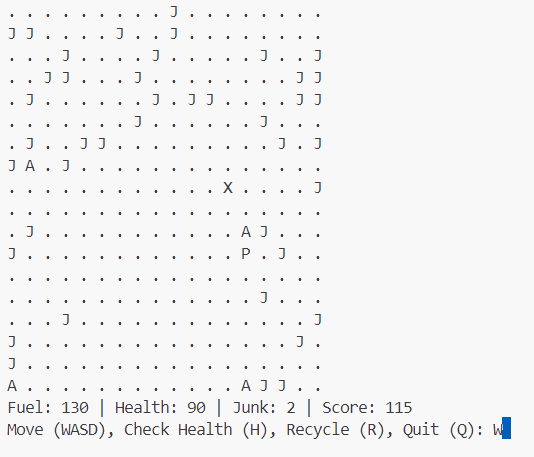


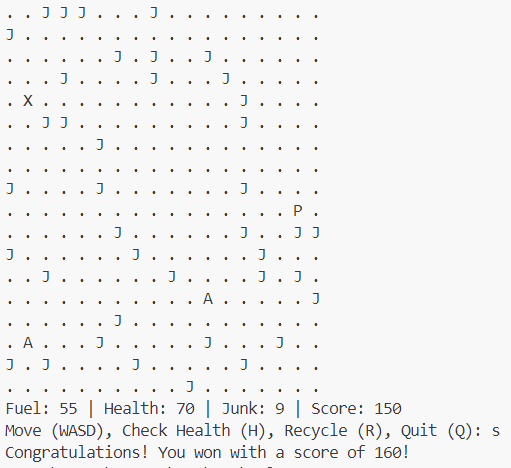












# **Functions**

* **Game Mechanics Functions**

1. **void initialize\_game(Game \*game);**  
   Initializes the game state, including player position, fuel, health, and grid setup.
2. **void setup\_difficulty(Game \*game);**  
   Lets the player select a difficulty level, which affects grid size and win conditions.
3. **void draw\_game(Game \*game);**  
   Renders the current state of the grid, showing the player, junk, asteroids, aliens, and empty spaces.
4. **void move\_player(Game \*game, char input);**  
   Handles movement logic based on WASD input and ensures player stays within bounds.
5. **void move\_asteroid(Game \*game);**  
   Moves the asteroid randomly within the grid to simulate a dynamic obstacle.
6. **int check\_collision(Game \*game);**  
   Checks if the player collides with junk, asteroid, or aliens and applies the appropriate effect.
7. **void update\_resources(Game \*game);**  
   Decreases fuel on every move and reduces health if fuel drops below zero.
8. **void check\_game\_state(Game \*game, int \*game\_over, int \*won);**  
   Determines if the player has won (by collecting enough junk) or lost (by running out of health).

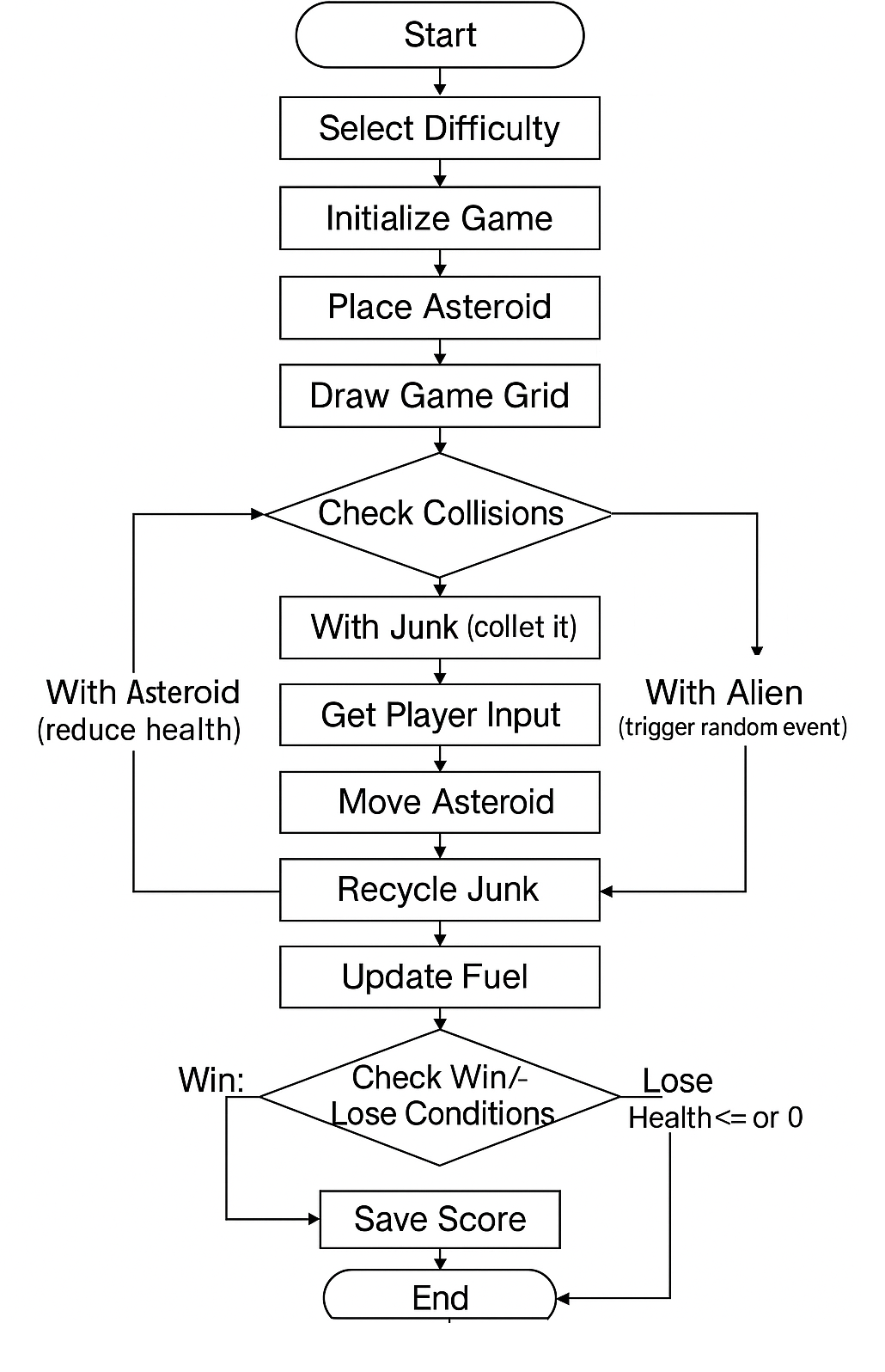
* **Feature and Event Functions**

1. **void place\_junk(Game \*game);**  
   Randomly places collectible junk on the grid at the beginning of the game.
2. **void place\_asteroid(Game \*game);**  
   Randomly positions the asteroid on the grid as a moving obstacle.
3. **void spawn\_aliens(Game \*game);**  
   Spawns alien entities on the grid that can affect the player upon contact.
4. **void recycle\_junk(Game \*game);**  
   Converts collected junk into score, fuel, and possibly health boosts to encourage resource management.
5. **void handle\_alien\_encounters(Game \*game);**  
   Random chance events are triggered when encountering aliens—either beneficial or harmful.

* **Utility & Display Functions**

1. **void display\_status(Game \*game);**  
   Shows real-time status of fuel, health, junk collected, and score at each turn.
2. **void show\_health(Game \*game);**  
   Provides a visual or textual health indicator upon request (pressing H in-game).
3. **void save\_score(Game \*game, char \*name);**  
   Saves the player's score with their name to a file for leaderboard or record-keeping.
4. **void read\_intro();**  
   Displays the game's introductory information and instructions at startup.
5. **void display\_end\_message(Game \*game, int won);**  
   Shows a win or lose message along with stats when the game ends.
6. **void clear\_input\_buffer();**  
   Clears any leftover characters from input buffer to prevent input issues.

# **Full Program Flowchart**



# **Functions Pseudocode**

1. **initialize\_game(Game \*game)**

// Initialize the game state and setup initial parameters

Function initialize\_game(game)

Prompt user to select grid size

Initialize player position, health, fuel, score, and junk count

Place player in center of grid

Call place\_junk(game)

Call place\_asteroid(game)

Call spawn\_aliens(game)

Fill rest of grid with empty symbols

EndFunction

1. **move\_player(Game \*game, char input)**

// Handle player movement based on WASD input

Function move\_player(game, input)

Switch input

Case 'W':

If player can move up

Update player position

Case 'A':

If player can move left

Update player position

Case 'S':

If player can move down

Update player position

Case 'D':

If player can move right

Update player position

Default:

Print "Invalid input"

Subtract 1 from fuel

EndFunction

1. **move\_asteroid(Game \*game)**

// Move asteroid one step in its current direction

Function move\_asteroid(game)

Clear asteroid's old position on grid

Update asteroid position based on dx, dy

If new position is out of bounds

Reverse direction

Clamp position within bounds

Draw asteroid at new position

EndFunction

1. **check\_collision(Game \*game)**

// Check collision between player and other objects

Function check\_collision(game)

If player is on junk

Mark junk as inactive

Increase junk\_collected and score

If player is on asteroid

If health not reduced recently

Reduce health by 20

Mark asteroid recently hit

Return 1

Else

Return 0

If player is on alien

Reduce health by 30

Mark alien as inactive

Return 1

Return 0

EndFunction

1. **recycle\_junk(Game \*game)**

// Convert junk into score and fuel

Function recycle\_junk(game)

If junk\_collected >= 3

Subtract 3 from junk\_collected

Add 20 to fuel

Add 100 to score

Print "Recycling successful"

Else

Print "Not enough junk to recycle"

EndFunction

1. **check\_game\_state(Game \*game, int \*game\_over, int \*won)**

// Determine if game should end based on current stats

Function check\_game\_state(game, game\_over, won)

If health <= 0

Set game\_over = 1

Set won = 0

Return

If fuel <= 0

Set game\_over = 1

Set won = 0

Return

If junk\_collected >= game.win\_threshold

Set game\_over = 1

Set won = 1

EndFunction

1. **setup\_difficulty(Game \*game)**

// Lets player choose difficulty, which sets grid size and win threshold

Function setup\_difficulty(game)

Prompt player to choose difficulty: Easy, Medium, Hard

Switch choice

Case Easy:

Set grid size = 10, win\_threshold = 3

Case Medium:

Set grid size = 15, win\_threshold = 5

Case Hard:

Set grid size = 20, win\_threshold = 8

Allocate and initialize grid based on size

EndFunction

1. **draw\_game(Game \*game)**

// Displays the grid and game stats

Function draw\_game(game)

Clear terminal screen

Print stats: fuel, health, junk collected, score

For each row in grid

For each column in grid

Print character for player, junk, asteroid, alien, or empty

EndFor

Print newline

EndFor

EndFunction

1. **update\_resources(Game \*game)**

// Adjusts fuel and applies health penalty if out of fuel

Function update\_resources(game)

If fuel < 0

Subtract 5 from health

Print "You're burning health due to no fuel!"

Else

Subtract 1 from fuel

EndFunction

1. **check\_game\_state(Game \*game, int \*game\_over, int \*won)**

// Determines win/lose conditions

Function check\_game\_state(game, game\_over, won)

If health <= 0

Set game\_over = 1

Set won = 0

ElseIf fuel <= -50

Set game\_over = 1

Set won = 0

ElseIf junk\_collected >= win\_threshold

Set game\_over = 1

Set won = 1

EndFunction