PROJECT REPORT

PACMAN

Team Members:

Sana Munir Alam (24K-0573) {Group Leader}

Adeena Asif (24K-0628)

Sanjna Kataria (24K-0738)

BSCS-1E

Submitted To:

Ms.Khadija Tul Kubra

INTRODUCTION

What is your Project?

This project is a console-based implementation of the classic **PAC-MAN** game with added features and functionality. The game is designed to provide an engaging gaming experience.

What does it do?

It has a continious moving PACMAN and 4 Ghost. (2 ghost moves Horizontally and 2 ghost moves Vertically)

It has coins which can be collected and replaced with empty space.

It has a timer that calculates how much time has passed since game started

It has a High Score and Player Score counter

It has W,A,S,D Keys as control keys (Made sure that both Upper Case and Small Case Letter are accepted)

It has Power Coins (@) that allows PACMAN to eat ghost temporaly for 8 seconds.

It has 3 Lives of PACMAN

It has a functioning maze

Invalid Enteries are ignored by the program

Game Ends in two Cases when all lives are lost or when all Coins Collected

When Gamover it shows options of NEW GAME and PLAY AGAIN, both these options work as they should (!Retain HighScore, Retain HighScore) Respectively

The game gives the user visual and audio feedback

Additional Thing:

This code gives colored output.

This code includes filing.

It asks user to create account and store it in a file.

it asks user to login to play

The game play when user has logged in.

The user than quits the game to save their score if it's higher than their previous score.

The user can view every player high score by choosing display option

Why did you choose it?

This project looked a bit exciting and challenging to do. Our aim was to test our boundaries on how much profecient we are in coding. Furthermore, the project is based upon the concepts taught to us in our classes hence it was a perfect project to chose to polish our skills.

DESIGN & IMPLEMENTATION:

The PACMAN game shows the User 4 options to choose from:

1- **Create Account:** If the User is new they choose option 1 to create an account.

2. **Login and Play:** After the account creation/ Or if the user already has an account. The user choose option 2 to login and play.

3- **Login and Play:** The login Username and Password is validated, if it's true the game starts by calling the function Pacman_Game(). Else the User is shown the Invalid Input message.

5- **Display Score:** The User can view all the players highscore through this option.

4- **Exit and Save Score:** Through this option the game ends and the user score are saved.

```
int main() {
    struct Player Players[MAX_Players];
    int PlayerCount = LoadHighscores(Players);
    int choice, score;
    char Username[MAX_NAME_LEN], Password[MAX_PASS_LEN];

    while (1) {
        // Initial Game Instructions and Start Page
        printf("\033[1;33m\n");
        printf(" ____   _    ____ __  __    _    _   _\n");
        printf("|  _ \\ / \\  / ___|  \\/  |  / \\  | \\ | |\n");
        printf("| |_) / _ \\| |   | |\\/| | / _ \\ |  \\| |\n");
        printf("|  __/ ___ | |___| |  | |/ ___ \\| |\\  |\n");
        printf("|_| /_/   \\_\\____|_|  |_/_/   \\_|_| \\_|\n");
        printf("\033[0m\n");
        printf("\n\n\n");
        printf("\033[0;36mINSTRUCTIONS:\033[0m\n");
        printf("1. Create New Account\n");
        printf("2. Login and Play\n");
        printf("3. Show High Scores of All Players in File\n");
        printf("4. Exit and Save Score in File\n");

        // Validate the choice input
        printf("Enter your choice: ");
        if (scanf("%d", &choice) != 1 || choice < 1 || choice > 4) {
            printf("Invalid input. Please enter a number between 1 and 4.\n");
            system("cls");
            clearInputBuffer(); // Clear the invalid input from the buffer
            continue;
        }
```

```
    // Process the valid choice
    switch (choice) {
        case 1:
            CreateAccount(Players, &PlayerCount);
            break;
        case 2:
            system("cls");
            getchar();
            printf("\033[1;33m\n");
            printf(" _      ___   ____   ___ _    _\n");
            printf("| |    / _ \\ / ___| |_ _| \\ | |\n");
            printf("| |   | | | | |  _   | ||  \\| |\n");
            printf("| |__| |_| | |_| |  | || |\\  |\n");
            printf("|_____\\___/ \\____| |___|_| \\_|\n");
            printf("\033[0m\n");
            printf("\nEnter your Username: ");
            fgets(Username, 30, stdin);
            Username[strcspn(Username, "\n")]='\0';
            printf("Enter your Password: ");
            fgets(Password, 30, stdin);
            Password[strcspn(Password, "\n")]='\0';

            int playerIndex = Login(Players, PlayerCount, Username, Password);
            if (playerIndex != -1) {
                system("cls");
                // Player logged in successfully, now start the game and get score
                score = Pacman_Game();
                system("cls");
                printf("\033[1;33m\n");
                printf(" ____  _____  _    _____ _   _ ____  \n");
                printf("/ ___|_   _|/ \\|_   _| | | / ___| \n");
                printf("\\___ \\ | | / _ \\ | | | | | \\___ \\ \n");
                printf(" ___) || |/ ___ \\| | | |_| |___) |\n");
                printf("|____/ |_/_/   \\_|_|  \\___/|____/ \n");
                printf("\033[0m\n");
                // Update the player's high score
                UpdatePlayerHighscore(Players, PlayerCount, playerIndex, score);
                SaveHighscores(Players, PlayerCount);
            } else {
                printf("Invalid Username or Password!\n");
            }
            break;
        case 3:
            system("cls");
            printf("\033[1;33m\n");
            printf(" ____    ____ ___  ____  _____    ____    ___     _     ____  ____  \n");
            printf("/ ___|  / ___/ _ \\|  _ \\| ____|  | __ )  / _ \\   / \\   |  _ \\|  _ \\ \n");
            printf("\\___ \\ | |  | | | | |_) |  _|    |  _ \\ | | | | / _ \\  | |_) | | | |\n");
            printf(" ___) | |__| |_| |  _ <| |___   | |_) | |_| |/ ___ \\ |  _ <| |_| |\n");
            printf("|____/ \\____\\___/|_| \\_|_____| |____/ \\___/_/   \\_|_| \\_|____/ \n");
            printf("\033[0m\n\n");
            DisplayHighscores(Players, PlayerCount);
            break;
        case 4:
            SaveHighscores(Players, PlayerCount);
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice. Please enter a number between 1 and 5.\n");
            break;
    }
}
return 0;
}
```

Program uses swicth statements to decide which option has been selected.

Each options calls the respective functions written in the cases.

Pre-processor Diretives

```c
#include <stdio.h>
#include <conio.h>      // For kbhit() and getch()
#include <windows.h>    // For Sleep()
#include <stdbool.h>
#include <time.h>       // To call time related functions
#include <string.h>
```

Macro Definations

These are used to define constants/symbolic names for values and expression throughout the code

```c
// Pacman Define
#define Width 60
#define Height 26
#define Wall '|'
#define Pacman 'C'
#define GhostGang 'X'
#define GhostGang2 'Y'
#define GhostGang3 'Z'
#define GhostGang4 'W'
#define Coins '.'
#define Empty ' '
#define PowerCoin '@'

// Filing Define
#define MAX_NAME_LEN 30
#define MAX_PASS_LEN 20
#define MAX_Players 100
```

Global Variables.

These variables are defined globally so that it doesn't have to be called/initialised in each separate function.

```c
//Global Variable
int Food = 0;
int Score = 0;
int HighScore = 0;
bool powerUpActive = false;
time_t powerUpStart;
```

STRUCT

```c
struct Player{
    char Username[MAX_NAME_LEN];
    char Password[MAX_PASS_LEN];
    int Highscore;
};
```

GAME START:

As the game start the first thing that happens is that it calls the function:

LoadHighScores(struct Player *Players)

This function is used to load every data existing in the file Highscore2.txt; and updates PlayerCount by 1 each time it iterates through the file until all data has been read. It returns the Total Player Count in the file.

If no data it returns 0.

```
// Load Players and their high scores from the file
int LoadHighscores(struct Player *Players) {
    FILE *file = fopen("Highscores2.txt", "r");  // Open for reading
    int PlayerCount = 0;
    if (file == NULL) {
        return 0; // No saved data, so no Players
    }
    // Loop through the file and read each player's data
    // fscanf reads the username, password, and highscore in a specific format
    while (fscanf(file, "Username: %s\n", Players[PlayerCount].Username) == 1) {
        fscanf(file, "Password: %s\n", Players[PlayerCount].Password);  // Read the
        fscanf(file, "Highscore: %d\n", &Players[PlayerCount].Highscore);   // Read
        PlayerCount++;  //PlayerCount is incremented by 1 to reflect that one more p
    }
    fclose(file);
    return PlayerCount;
}
```

OPTION 1: CreateAccount

CreateAccount(struct Player *Players, int *PlayerCount)

The program asks for User to enter a USERNAME and PASSWORD, which is stored in an array, and the Player Count is increased by 1 to keep track of Player quantity.

The User is displayed a visual feedback of Account Created Succesfully.

```
// Create a New Account For the Player
void CreateAccount(struct Player *Players, int *PlayerCount) {
    getchar();

    printf("Enter Username: ");
    fgets(Players[*PlayerCount].Username, MAX_NAME_LEN, stdin);
    Players[*PlayerCount].Username[strcspn(Players[*PlayerCount].Username, "\n")] = '\0';
    printf("Enter Password: ");
    fgets(Players[*PlayerCount].Password, MAX_PASS_LEN, stdin);
    Players[*PlayerCount].Password[strcspn(Players[*PlayerCount].Password, "\n")] = '\0';

    Players[*PlayerCount].Highscore = 0; // New Players start with 0 score
    (*PlayerCount)++;   // Increase the player Counter by 1
    printf("Account created successfully!\n");
}
```

OPTION 2: Login-Play

Login(struct Player *Players, int PlayerCount, char *Username, char *Password)

The program ask user to enter their USERNAME and PASSWORD. The program than compares the Input with the stored data by iterating through the array based upon Player Count.

If the User Input matches with Stored Value the Function returns player index and the game starts by calling **Pacman_Game()** function.

```
// Check Login credentials for an existing player
int Login(struct Player *Players, int PlayerCount, char *Username, char *Password) {
    for (int i = 0; i < PlayerCount; i++) {
        if (strcmp(Players[i].Username, Username) == 0 && strcmp(Players[i].Password, Password) == 0) {
            return i; // Player Found (Index)
        }
    }
    return -1; // Player not found
}
```

OPTION 3: Display HighScore

DisplayHighScores(struct Player *Players, int PlayerCount)

This function is called when User chooses this option in the **Main**. Th function iterates using For loop till the PlayerCOunt and prints:

-Username

-Password

-HighScore

```
// Display the High Scores For All Players
void DisplayHighscores(struct Player *Players, int PlayerCount) {
    printf("\nHigh Scores:\n");
    for (int i = 0; i < PlayerCount; i++) {
        printf("Username: %s, Highscore: %d\n", Players[i].Username, Players[i].Highscore);
    }
}
```

OPTION 4: Exit

The game ends by returning 0.

UpdatePlayerHighScore(struct Player *Players, int PlayerCount, int playerIndex, int score)

This Function is called jus as the **Pacman_Game()** returns either Score or 0.

It firstly checks if the Player Index is valid, than checks if the Player Score is greater than the Score saved in the file.

If true the High Score is updated and Showed to the User with a prompt that his highscore has updated.

If false: The current Score is displayed back.

If the Player Index is not valid it prints “Invalid Player Index”.

```
// Update the player's Highscore if necessary
void UpdatePlayerHighscore(struct Player *Players, int PlayerCount, int playerIndex, int score) {
    // Make sure the player index is valid
    if ((playerIndex >= 0 && playerIndex < PlayerCount)) {
        if(score > Players[playerIndex].Highscore){
            Players[playerIndex].Highscore = score;
            printf("\n\nHighscore updated successfully!\n");
            printf("Your New Highscore: %d\n", score);
        }else{
            printf("\nYour Score: %d\n", score);
        }
        return;
    } else {
        printf("Invalid player index.\n");
    }
}
```

SaveHighScore(struct Player *Players, int PlayerCount)

This function is called after **UpdateHighScore()** has been called after the game ends.

It opens the HighScore2.txt file and starts to overwrite each data as it iterates through PlayerCount one by one.

After All the data has been rewritten the file is closed.

```
// Save all player data, including high scores, to a file
void SaveHighscores(struct Player *Players, int PlayerCount) {
    FILE *file = fopen("Highscores2.txt", "w");  // Use "w" mode to overwrite and save the data
    if (file == NULL) {
        printf("Error opening file!\n");
        return;
    }
    for (int i = 0; i < PlayerCount; i++) {
        fprintf(file, "Username: %s\n", Players[i].Username);
        fprintf(file, "Password: %s\n", Players[i].Password);
        fprintf(file, "Highscore: %d\n\n", Players[i].Highscore);
    }
    fclose(file);
}
```

CURSOR FUNCTIONS:

1- move_cursor(int x, int y)

This functions are called in **Pacman_Game()** and **Render()**. It moves the cursor to the coordinates determined by the parameter.

2- hide_cursor()

This function is called in **Pacman_Game()** just after the **InitialiseGame()** function has been called.

 Retrieves the current console cursor settings using GetConsoleCursorInfo and modifies the visibility attribute (bVisible) to FALSE.

 Updates the cursor settings with SetConsoleCursorInfo to hide the cursor in the console window during gameplay.

3- show_cursor()

This function is called in **Pacman_Game()** just after the game ends

Retrieves the current console cursor settings using GetConsoleCursorInfo and modifies the visibility attribute (bVisible) to TRUE.

Updates the cursor settings with SetConsoleCursorInfo to make the cursor visible again, usually after gameplay ends.

```
// Move cursor to a specific position in the terminal
void move_cursor(int x, int y) {
    COORD coord;
    coord.X = x;
    coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

// Hide the cursor
void hide_cursor() {
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
    cursorInfo.bVisible = FALSE; // Hide the cursor
    SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
}

// Show the cursor
void show_cursor() {
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
    cursorInfo.bVisible = TRUE; // Show the cursor
    SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
}
```

ClearInputBuffer()

This function is called in **Main** finction in order to clear the input Buffer untial a newline is found.

```
// Function to avoid Run Time Error in Filing Area
void clearInputBuffer() {
    while (getchar() != '\n'); // Clears the input buffer until a newline is found
}
```

PACMAN FUNCTIONS

GetElapsedTime()

This function is called in the **Pacman_Game()**.

This function simply calculates the difference in time since the Game started and the Current Time.

```
// Time Function
double GetElapsedTime(time_t StartTime) {
    return difftime(time(NULL), StartTime);
}
```

IsWALL(int x, int y)

The function is boolean and is called in **Render()** function when it is running through nested loop.

The i and j values in the Render function is sent to this function to check whether a wall exists.

Multiple Conditions are checked for the parameters (int x, int y). If the condition matches the function returns TRUE to **Render()** and the Wall (**|**) is printed.

```
// Check if next position is wall (This is used to print the maze wall structure)
bool IsWall(int x, int y) {
    if(y == 0 || x == Width -1 || x == 0 || y == Height -1) {   //This creates the main boundary of the wall
        return true;
    }else if((y==1 || y==2 || y==3 || y==4) && (x == 29 || x == 30)){ // Row 1 to Row 4 walls
        return true;
    }else if((y==2 || y==3 || y==4)&& ((x>=2 && x <= 13)||(x>=16 && x<=27)||(x>=32 && x<=43)||(x>=46 && x<=57))){
        return true;
    }else if((y==6)&& ((x>=2 && x<=13) || (x>=16 && x<=18) || (x>=20 && x<=39)|| (x>=41 && x<=43)|| (x>=46 && x<=57))){ // Row 6
        return true;
    }else if((y==7)&& ((x>=16 && x<=18) || (x>=29 && x<=30) || (x>=41 && x<=43))){ // Row 7
        return true;
    }else if((y==8)&& ((x>=1 && x<=13) || (x>=16 && x<=26) || (x>=29 && x<=30)|| (x>=33 && x<=43)|| (x>=46 && x<=58))){ // Row 8
        return true;
    }else if((y==9)&& ((x>=1 && x<=13) || (x>=16 && x<=18) || (x>=41 && x<=43)|| (x>=46 && x<=58))){ // Row 9
        return true;
    }else if((y==10)&& ((x>=1 && x<=13) || (x>=16 && x<=18) || (x>=20 && x<=28)|| (x>=31 && x<=39)|| (x>=41 && x<=43)|| (x>=46 && x<=58))){ // Row 10
        return true;
    }else if((y == 11) &&((x>=20 && x<=22)||(x>=37 && x<=39))){ // Row 11
        return true;
    }else if((y == 12) &&((x>=1 && x<=13)||(x>=16 && x<=18)||(x>=20 && x<=28)||(x>=31 && x<=33)||(x>=34 && x<=39)||(x>=41 && x<=43)||(x>=46 && x<=58))){ // Row 12
        return true;
    }else if((y == 13) &&((x>=1 && x<=13)||(x>=16 && x<=18)||(x>=41 && x<=43)||(x>=41 && x<=43)||(x>=46 && x<=58))){ // Row 13
        return true;
    }else if((y == 14)&&((x>=1 && x<=13)||(x>=16 && x<=18)||(x>=20 && x<=22)||(x>=23  && x<=39)||(x>=41 && x<=43)||(x>=46 && x<=58))){ // Row 14
        return true;
    }else if((y == 15)&&((x==29 || x ==30))){ // Row 15
        return true;
    }else if ((y==16)&& ((x>=2 && x<= 13) || (x>=16 && x<=26) || (x>=29 && x<= 30) || (x>=33 && x<= 43) || (x>=46 && x<=57))) { // Row 16
        return true;
    }else if ((y==17)&& ((x>=8 && x<= 13) || (x>=46 && x<=51))) { // Row 17
        return true;
    }else if ((y==18)&& ((x>=1 && x<= 6) || (x>=8 && x<=13)|| (x>=16 && x<=18)|| (x>=20 && x<=39) || (x>=41 && x<=43) || (x>=46 && x<=51) || (x>=53 && x<=58))) { // Row 18
        return true;
    }else if ((y==19)&& ((x>=16 && x<= 18) || (x>=29 && x<=30)|| (x>=41 && x<=43))) { // Row 19
        return true;
    }else if ((y==20)&& ((x>=2 && x<= 26) || (x>=29 && x<=30)|| (x>=33 && x<=57))) { // Row 20
        return true;
    }else if ((y==21)&& ((x>=2 && x<= 10) || (x>=28 && x<=31)|| (x>=49 && x<=57))) { // Row 21
        return true;
    }else if((y==22)&&((x>=2&&x<=7)||(x>=11&&x<=26)||(x>=33&&x<=54))){ // Row 22
        return true;
    }else if((y == 23)&&((x>=2&&x<=9)||(x>=11&&x<=15)||(x>=17&&x<=26)||(x>=33&&x<=57))){ // Row 23
        return true;
    }else if((y == 24)&&(x>=28&&x<=31)){ // Row 24
        return true;
    }
    return false;
}
```

InitializaGame(char Map[Height][Width], int *x, int *y, int *Lives, bool resetScore, bool resetHighScore)

This Function is called through **Pacman_Game()** just as the user press **Y** to start game or when the game has ended and the user presses **P** or **N**. It basically prints the very first layout of the game and resets the variables.

A FOR loop is used to play a beep sound 3 times to show that the game has started.

The PACMAN position is being reset at the determined position of (29,17).

The Lives counter is set to 3.

A check is made based upon the TRUE/FALSE status of resetHighscore to set Current Score to 0 and Highscore to 0 or to retain High Score from LastSession.

If resetHighScore is TRUE, HighScore is set to 0.

If resetHighScore is FALSE, HighScore is retained from Last Session

A nested FOR loop is used to print PACMAN/ GHOST-GANG/ and POWER-COINS on fixed Location Initially. The rest of the spaces is filled with COINS.

PACMAN(29,17)GHOSTS(14,5),(23,11),(45,19),(14,1) POWERCOIN(58,11),(1,11)

```
// Initializes or resets the game variables and map
void InitializeGame(char Map[Height][Width], int *x, int *y, int *Lives, bool resetScore, bool resetHighScore) {
    int k = 100;
    for(int i = 0; i < 4; i++){
        k = k + 110;
        Beep(k, 100);
    }
    // Reset PACMAN position, lives, and score
    *x = 29;
    *y = 17;
    *Lives = 3;
    Score = resetHighScore? 0 : 0;
    HighScore = resetHighScore ? 0 : HighScore;
    // Reset the map with coins and clear previous characters positions
    for (int i = 0; i < Height; ++i) {
        for (int j = 0; j < Width; ++j) {
            if (i == 17 && j == 29) {
                Map[i][j] = Pacman;
            } else if (i == 5 && j == 14) {
                Map[i][j] = GhostGang;
            } else if ( i == 11 && j == 23){
                Map[i][j] = GhostGang2;
            } else if (i == 19 && j ==45){
                Map[i][j] = GhostGang3;
            } else if(i == 1 && j == 14){
                Map[i][j] = GhostGang4;
            } else if ((i == 11 && j == 1) || (i == 11 && j == 58)){
                Map[i][j] = PowerCoin;
            }else {
                Map[i][j] = Coins;
            }
        }
    }
}
```

Render(int x, int y, int Ghost1X, int Ghost1Y, int Ghost2X, int Ghost2Y, int Ghost3X, int Ghost3Y, int Ghost4X, int Ghost4Y, int Lives, int Score, int HighScore, double ElapsedTime, char Map[Height][Width])

The function is called every time the while loop itereate in **Pacman_Game()**.

It sets the cursor at (0,0) by calling the function **move_cursor()**. For smooth display.

Nested For Loop is used to Draw the Maze and characters.

As the iteration occurs the i and j acts as y and x coordinates respectively of maze.

Conditins are checked on each coordinate whether the wall is to be printed by calling **IsWall(j,i)** function or the characters are to be printed based on the parameters the Render recieves.

The Render function recieves the updated coordinates of Pacman(**int x, int y)** and Ghosts(**Ghost1X, Ghost1Y….)** through **Pacman_Game()** funcion and based upon those paramaters it prints the WALLS/ PACMAN/ GHOST on those new coordinates if it is qual to j and i values.

For ghost it checks the powerUpActive status if it is TRUE or FALSE and based upon that it prints the ghost style Smaller Case (**x**) indicates PACMAN can eat it, Upper Case(**X**) indicates Ghost can eat PACMAN respectively.

If j and i which is equal to COIN coordinates it will print (**.**) in that location. Else Blank space is left behind(EMPTY).

The function gives visual feedback to user by printing current status of gameplay:

-Lives

-Current Score

-High Score

-Time Elapsed

```
// Render the box and character (every iteration)
void Render(int x, int y, int Ghost1X, int Ghost1Y, int Ghost2X, int Ghost2Y, int Ghost3X, int Ghost3Y,
int Ghost4X, int Ghost4Y, int Lives, int Score, int HighScore, double ElapsedTime, char Map[Height][Width]) {
    move_cursor(0, 0);  // Move cursor to the top left corner

    // Draw the Maze and Characters
    for (int i = 0; i < Height; ++i) {
        for (int j = 0; j < Width; ++j) {
            if (IsWall(j, i)) {
                printf("%c", Wall);
            } else if (i == y && j == x) {
                printf("%c", Pacman);  // The Pacman
            } else if (i == Ghost1Y && j == Ghost1X) {
                printf(powerUpActive ? "x" : "X");    // Ghost1
            } else if (i == Ghost2Y && j == Ghost2X){
                printf(powerUpActive ? "y" : "Y");   // Ghost2
            } else if (i == Ghost3Y && j == Ghost3X){
                printf(powerUpActive ? "z" : "Z");   // Ghost3
            } else if (i == Ghost4Y && j == Ghost4X){
                printf(powerUpActive ? "w" : "W");   // Ghost4
            }else if (Map[i][j] == PowerCoin) {
                printf("%c", PowerCoin);  // Power Coin
            } else if (Map[i][j] == '.') {
                printf("%c", Coins);  // Coin
            } else {
                printf("%c", Empty);  // Empty space
            }
        }
        printf("\n");
    }
    printf("Lives: %d\n", Lives); // Display Remaining Lives
    printf("Score: %d\n", Score); // Display the Player Score
    printf("High Score: %d\n", HighScore); // Display the High Score
    printf("Time: %.0f seconds\n", ElapsedTime); // Display Elapsed Time
    // printf("Pacman(%d,%d)\n", x,y);      THESE LINES COMMENTED SO THAT THE USER DOESNT SEE THIS
    // printf("Demon 1 (%d,%d)\n", Ghost1X, Ghost1Y);
    // printf("Demon 2 (%d,%d)\n", Ghost2X, Ghost2Y);
    // printf("Demon 3 (%d,%d)\n", Ghost3X, Ghost3Y);
    // printf("Demon 4 (%d,%d)\n", Ghost4X, Ghost4Y);
    fflush(stdout);  // Ensure everything is printed immediately
}
```

AllCoinsCollected(int Food)

The function checks if the **Food** Counter is not equal to 613.

If yes it returns FALSE, else it returns TRUE.

```
// Function to check if all coins have been collected
bool AllCoinsCollected(int Food) {
    if (Food != 613){
        Beep(550, 80);
        return false;   // Coins Left
    }
    return true;  // No coins left
}
```

THE MAIN FUNCTION OF PACMAN GAME

Pacman_Game()

```
// Main game function
int Pacman_Game() {
    int x = 29;          // Initial x position of Pacman
    int y = 17;          // Initial y position of Pacman
    int dx = 0;          // No x movement initially
    int dy = 0;          // No y movement initially
    int Ghost1X = 14;    // Initial Ghost1 x position
    int Ghost1Y = 5;     // Fixed Ghost1 y position
    int Ghost1DX = 1;    // Ghost1 moving right initially
    int Ghost2X = 23;    // Initial Ghost2 x position
    int Ghost2Y = 11;    // Fixed Ghost2 y position
    int Ghost2DX = 1;    // Ghost2 moving right initially
    int Ghost3X = 45;    // Fixed Ghost3 x position
    int Ghost3Y = 19;    // Initial Ghost3 y position
    int Ghost3DY = -1;   // Ghost3 Moving up initially
    int Ghost4X = 14;    // Fixed Ghost4 x position
    int Ghost4Y = 1;     // Initial Ghost4 y position
    int Ghost4DY = 1;    // Ghost4 moving down initially
    int Lives = 3;       // Pacman Total Lives
    int Result = 0;      // Holds the AllCoinsCollected Status
    bool Stunned = false;
    bool Running = true;
    char Map[Height][Width];

    time_t StartTime;
    const double frameDelay = 0.03; // Frame delay for movement (the smaller the delay the faster the movement)

    // Initial Game Instructions and Start Page
    printf("\033[1;33m\n");
    printf(" ____    _    ____ __  __    _    _   _\n");
    printf("|  _ \\ / \\  / ___|  \\/  |  / \\  | \\ | |\n");
    printf("| |_) / _ \\| |   | |\\/| | / _ \\ |  \\| |\n");
    printf("|  __/ ___ | |___| |  | |/ ___ \\| |\\  |\n");
    printf("|_| /_/   \\_\\____|_|  |_/_/   \\_|_| \\_|\n");
    printf("\033[0m\n");
    printf("\n\n\n");
    printf("\033[0;36mINSTRUCTIONS:\033[0m\n");
    printf("To Move Up Press W\n");
    printf("To Move Down Press S\n");
    printf("To Move Right Press D\n");
    printf("To Move Left Press A\n");
    printf("To Quit Game Press Q\n");
    printf("\n\n");
    printf("\033[0;36m");
    printf("CAUTION-------!!!-------CAUTION-------!!!-------CAUTION\n\n");
    printf("\033[0m");
    printf("Caution: Pressing Q During Game WILL NOT SAVE YOUR SCORE instead wil send you back to Main Page!\n");
    printf("ALL PROGRESS DURING THAT SESSION WILL BE LOST.\n");
    printf("THINK BEFORE PRESSING \"Q\"");
    printf("\n\n");
    printf("To Start Game Press Y\n");


    // Wait for user input using getch
    char pf = _getch();
    if (pf != 'Y' && pf != 'y') {
        return 0; // Exit if not 'Y'/'y'
    }
    system("cls");  // Clearing the screen so that only map is shown

    // Game Initialization
    InitializeGame(Map, &x, &y, &Lives, true, true); // Start with a clean state
    hide_cursor();
```

This dunction is called from the **Main()** function when the user has logged in.

The first thing this function does is it sets the variables values. The variables indicates the initial position of PACMAN, all four GHOSTS, the continue movement variable, LIVES.

A variable StartTime is initialised. A boolean variable Running is set to 1, this variable is what that runs the game repeatedly. A boolean variable Stinned is set to false.

On the screen PACMAN is printed out along with the Instructions on how to play the game.

The Function asks User to enter Y to Start the game.

Based upon the Input it is checked if the Input matches with Y/y the game starts. Else the game ends and the user is showed the Main Page.

The **InitialiseGame()** function is called to set up the mazze and characters.

The **hide_cursor()** is called to hide the cursor.

```
while (Running) {
    StartTime = time(NULL);  // Reset the timer

    while (Running) {
        // Check if power-up time is over
        if (powerUpActive) {
            double elapsed = difftime(time(NULL), powerUpStart);
            if (elapsed >= 8) {
                powerUpActive = false; // Deactivate power-up after 8 seconds
            }
        }
        if (_kbhit()) { // Check if a key is pressed
            char ch = _getch(); // Get the pressed key
            if (Stunned) {      // This Condition Runs when Pacman has hit Ghost Gang
                Stunned = false;
                x = 29; // Original x coordinate
                y = 17; // Original y coordinate
                dx = 0; // Reset x movement direction
                dy = 0; // Reset y movement direction
            } else {
                switch (ch) {
                case 'w': case 'W': // Move up
                    dx = 0;
                    dy = -1;
                    break;
                case 's': case 'S': // Move down
                    dx = 0;
                    dy = 1;
                    break;
                case 'a': case 'A': // Move left
                    dx = -1;
                    dy = 0;
                    break;
                case 'd': case 'D': // Move right
                    dx = 1;
                    dy = 0;
                    break;
                case 'q': case 'Q': // Quit the game
                    system("cls");
                    Running = false;
                    return 0;
                    break;
                }
            }
        }
```

A while loop runs till the variable RUNNING is true.

The StartTime variable is set to 0.

Another nested while loop iterates till variable RUNNING is true.

A condition is checkedif powerUpActive is TRUE. If true the time count will start and a nothercondition will be checked:

- if the count time has passed 8seconds. If it's false the loop continues to run and the poweUpActive status remains true. If the innner condition is FALSE the powerUpActive status is set to False and the condition is exited.

A variable **ch** takes the user key input

Another check is placed to check is the variable Stunned is true? If true the PACMAn is set back to its starting position of (29,17) and its continious movement variable **(dx,dy)** is set to 0.

Else a switch case checks the variable **ch** and based upon it, it checks whether the pacman has ti move Up/Down/Right/Left or the game has to end if Q is pressed.

In case Q is pressed the **Pacman_Game()** function ends and we go back to the **Main()** function.

```
// This Condition Runs when Pacman has not hit Ghost Gang
if (!Stunned) {
    int newX = x + dx;
    int newY = y + dy;

    if (IsWall(newX, newY)) {
        continue;
    } else if (Map[y][x] == PowerCoin) {
        Beep(700, 900);
            powerUpActive = true;
            powerUpStart = time(NULL);
            Map[y][x] = ' ';
    } else if (((newX + 1) == Ghost1X && newY == Ghost1Y) || (Ghost1X == (newX - 1) && Ghost1Y == newY) || (Ghost1X == newX && Ghost1Y == newY)) {
        if (powerUpActive) {
            Beep(700, 900);
            Score += 10;
            printf("\033[0;36mCongratulations!! You ate a ghost!!      \033[0m");
        }else{
            Beep(690, 200);
            Sleep(150);
            Beep(610, 200);
            Sleep(150);
            Beep(690, 200);
            Lives--;    // Decrease Lives by 1
            Stunned = true; // Set stunned state to true
            printf("\033[1;31mOuch! You hit a ghost! Lives left: %d\033[0m\n", Lives); //red color text to inform user that a life is lost

            // Reset Pacman's position
            x = 29; // Original x position
            y = 17; // Original y position

            // Check if the player is out of lives
            if (Lives <= 0) {
                break; // Break if no lives are left
            }
        }
    } else if(((newX + 1) == Ghost2X && newY == Ghost2Y) || (Ghost2X == (newX - 1) && Ghost2Y == newY) || (Ghost2X == newX && Ghost2Y == newY)) {
    } else if((newX == Ghost3X && newY == Ghost3Y) || (Ghost3X == newX && Ghost3Y == newY - 1)|| (Ghost3X == newX && Ghost3Y == newY + 1)) {   // C
        if (powerUpActive) {
            Beep(700, 900);
            Score += 10;
            printf("\033[0;36mCongratulations!! You ate a ghost!!      \033[0m");
        }else{
            Beep(690, 200);
            Sleep(10);
            Beep(610, 200);
            Sleep(10);
            Beep(690, 200);
            Lives--;    // Decrease Lives by 1
            Stunned = true; // Set stunned state to true
            printf("\033[1;31mOuch! You hit a ghost! Lives left: %d\033[0m\n", Lives);  //red color text to inform user that a life is lost

            // Reset Pacman's position
            x = 29; // Original x position
            y = 17; // Original y position

            // Check if the player is out of lives
            if (Lives <= 0) {
                break; // Break if no lives are left
            }
        }
    } else if(newX == Ghost4X && newY == Ghost4Y || (Ghost4X == newX && Ghost4Y == newY - 1)|| (Ghost4X == newX && Ghost4Y == newY - 1)){   // Coll
```

If the variable stunned is false. The program than checks:

It continue to update the Pacman direcion based upon (dx,dy) values.

Than a condition is checked by calling **IsWall()** function and sending Pacman coordinates as paramters.

If the function returns true the next iteration is skipped. Else we move to next condition which checks if the Map[x][y] is equal to PowerCoin.

If true, the **powerUpActive** status is set to true, and the time variable powerUpStart is set to 0 and that location is now set to Empty.

The loop itertaes again and we reach back to these conditions.

If the next location was not of PowerCoin we move to the next condition check which checks whether the PACMAN coordinate is equal to the GHOSTS coordinate.

If true it checks if the variable **powerUpActive** is true. If true, it increases the score by 10 points and prints a message to User “Congratulations!! You ate a ghost!”

If the **powerUpActive** is false a beep is played and the **Lives** counter is decreased by 1 and the variable **Stunned** is set to true. A message is printed to the user that tells thay have been eaten by Ghost and their current ives status.

The Pacman is sent back to its starting position. An inner check is made to confirm if Lives <= 0. If true the program breaks out of the condition. If false the program continues to run.

These checks are made for all 4 ghosts.

```
    } else {
        // Only update Pacman's position if not stunned
        x = newX;
        y = newY;

        // Check for Coin Collection
        if (Map[y][x] == Coins) {
            Map[y][x] = Empty; // Remove Coin from Map
            Score++;    // Increase Score by 1
            Food++;
            if (Score > HighScore) {
                HighScore = Score; // Update High Score
            }
            // Check if All Coins Are Collected
            if (AllCoinsCollected(Food)) {
                Result = 1;
                break;
            }
        }
    }
}
```

If none of the above conditions are true the else block executes by updating only Pacman’s position.

In this block a condition is checked by checking if the Map[y][x] is equal to Coins. If true, the COIN is replaced by EMPTY, the Score is iincreased by 1 and Food is increased by1, A condition check to update HighSCore to current is the Score is greater than previous HighScore.

A condition check by calling **AllCoinsCollected()** function by sending Food count as parameter to check whether the maze is empty of food. If yes the variable Result is set to 1 and the inner conditions are broken off.

```
        // Ghost 1 movement // Confirmed Deaths of PACMAN if hit
        Ghost1X += Ghost1DX;
        if (Ghost1X < 1 || Ghost1X >= Width - 1) { // Change direction at boundaries
            Ghost1DX = -Ghost1DX;
            Ghost1X += Ghost1DX; // Move Ghost1 in the new direction
        }
        // Ghost 2 movement // Confirmed Deaths of PACMAN if hit
        Ghost2X += Ghost2DX;
        if (Ghost2X < 23 || Ghost2X > 36) { // Change direction at boundaries
            Ghost2DX = -Ghost2DX;
            Ghost2X += Ghost2DX; // Move Ghost2 in the new direction
        }
        // Ghost 3 movement // Confirmed Deaths of PACMAN if hit
        Ghost3Y += Ghost3DY;
        if (Ghost3Y < 1 || Ghost3Y > 19) { // Change direction at boundaries
            Ghost3DY = -Ghost3DY;
            Ghost3Y += Ghost3DY; // Move Ghost3 in the new direction
        }
        // Ghost 4 movement // Confirmed Deaths of PACMAN if hit
        Ghost4Y += Ghost4DY;
        if (Ghost4Y < 1 || Ghost4Y > 19) { // Change direction at boundaries
            Ghost4DY = -Ghost4DY;
            Ghost4Y += Ghost4DY; // Move Ghost4 in the new direction
        }

        double ElapsedTime = GetElapsedTime(StartTime);
        Render(x, y, Ghost1X, Ghost1Y, Ghost2X, Ghost2Y, Ghost3X, Ghost3Y, Ghost4X, Ghost4Y, Lives, Score, HighScore, ElapsedTime, Map);

        // End Game EITHER due to All Lives Lost OR All Coins Collected / Option Menu
        if ((Lives <= 0) || (Result == 1)) {
            //music when the game is over
            int k = 700;
            for(int i = 0; i < 3; i++){
                k = k - 100;
                Beep(k, 100);
            }
            system("cls");

            printf("\033[1;33m");
            printf("  ____    _    __  __ _____     _____   _____ ____\n");
            printf(" / ___|  / \\  |  \\/  | ____|   / _ \\ \\ / / | ____|  _ \\\n");
            printf("| |  _  / _ \\ | |\\/| |  _|    | | | \\ V /  | |_) |  \n");
            printf("| |_| |/ ___ \\| |  | | |___   | |_| |\\ V / | |___|  _ <\n");
            printf(" \\____/_/   \\_|_|  |_|_____|   \\___/  \\_/  |_____|_| \\_\\\n");
            printf("\033[0m");     // Reset text formatting
            printf("\n\n\n");
            printf("Your Score: %d\n", Score);
            printf("High Score: %d\n", HighScore);
            printf("\n\n\n");
            printf("Press Q to Quit and Save your Score in File.\n");
            printf("Press N for New Game (Reset High Score of this SESSION, \033[1;31mCaution: Score is not saved on File\033[0m)\n");
            printf("Press P for Play Again (Retain High Score of this SESSION, \033[1;31mCaution: Score is not saved on File\033[0m)\n");

            pf = _getch();
            if (pf == 'Q' || pf == 'q') {
                Running = false;
                system("cls");
                return Score;
            } else if (pf == 'N' || pf == 'n') {    // New Game (High Score is set to 0)
                system("cls");
                Result = 0;
                InitializeGame(Map, &x, &y, &Lives, true, true);
            } else if (pf == 'P' || pf == 'p') {    // Play Again (High Score is Retained)
                system("cls");
                Result = 0;
                InitializeGame(Map, &x, &y, &Lives, true, false);
            }
        }
        Sleep((DWORD)(frameDelay * 1000)); // Frame rate control
    }
}
show_cursor();
return 0;
}
```

After each iteration the Ghost coordinates are incremented and decremented by 1 based on the condition check, if the ghost move right or down the bariable for their continious movement adds 1.

If their cordinates is eaqual to the coordinate of the boundary -1 length the ghost direction changes and they start to move in. the opposite direction (Left, Up). Due to their continious movement variable now decrementing by 1.

The function **GetElapsedTime()** is called to claculate how much time has passed since game started. And is stored in varaiable ElapsedTime

The **Render()** function is called and is passed the PACMAN, GHOST-GANG coordinates, Lives, Score, HighScore, ElapsedTime and Map as Paramater.

A condition is checked if Lives <= 0 OR Result == 1. If either of the condition is true a beef is sounded and the Game Over message is displayed to user.

Aling with their Score and HighScore.

The User is displayed an option of whether they want to Quit game (**Press Q**) or want to start a New Game with the High Score set to 0 (**Press N**), or Play Again with High Score retained from the current session (**Press P**).

The user Input is checked If it's **Q** the Runnig Vvariable is set to False and the **Pacman_Game()** returns Score to the **Main()** function.

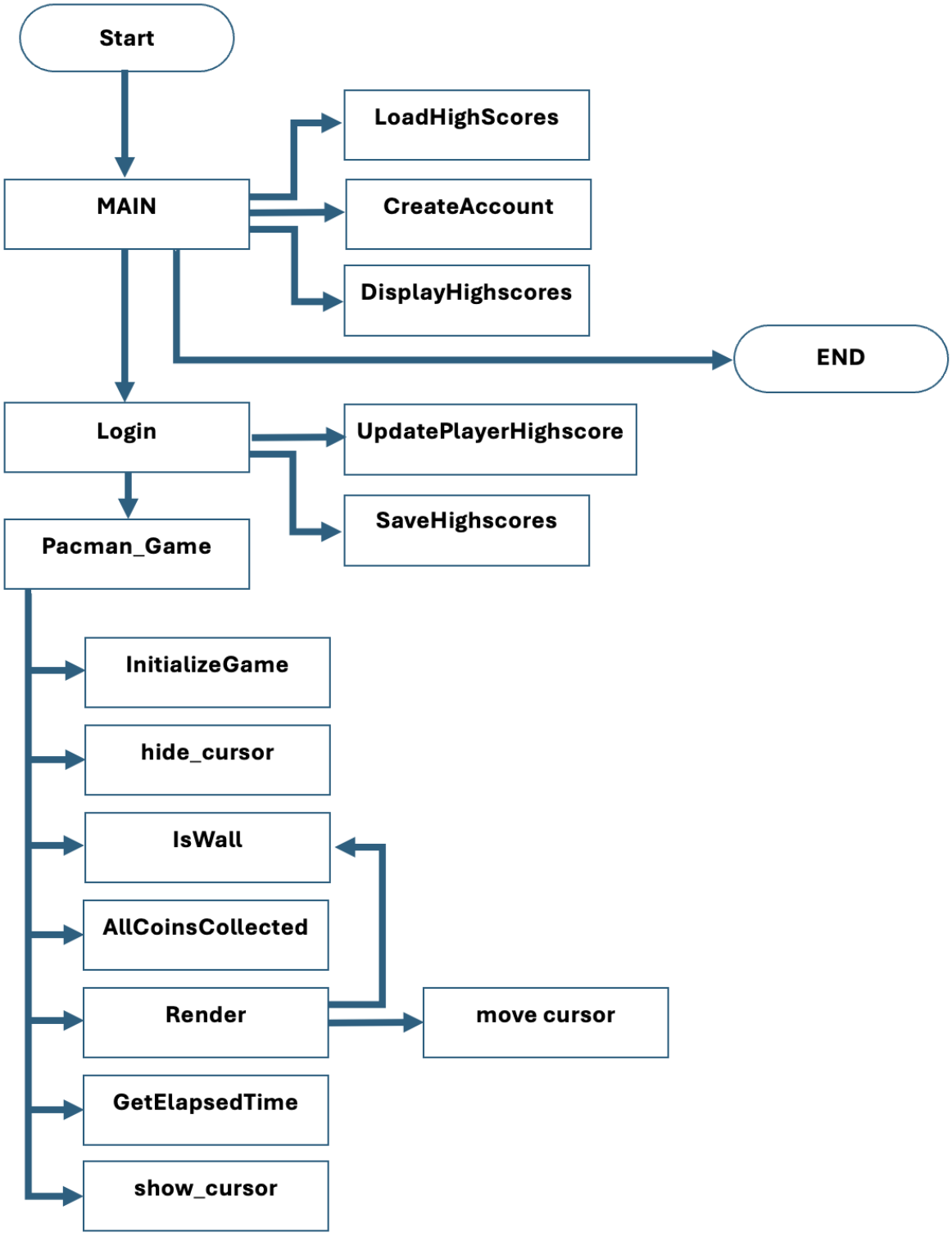
If the player choose **N** the Result is set to 0 and the function **InitialiseGame(Map, &x, &y, &Lives, true, true)** is called.

If the [;ayer chooses **P** the Result is set to 0 and the function **InitialiseGame(Map, &x, &y, &Lives, true, false)** is called.

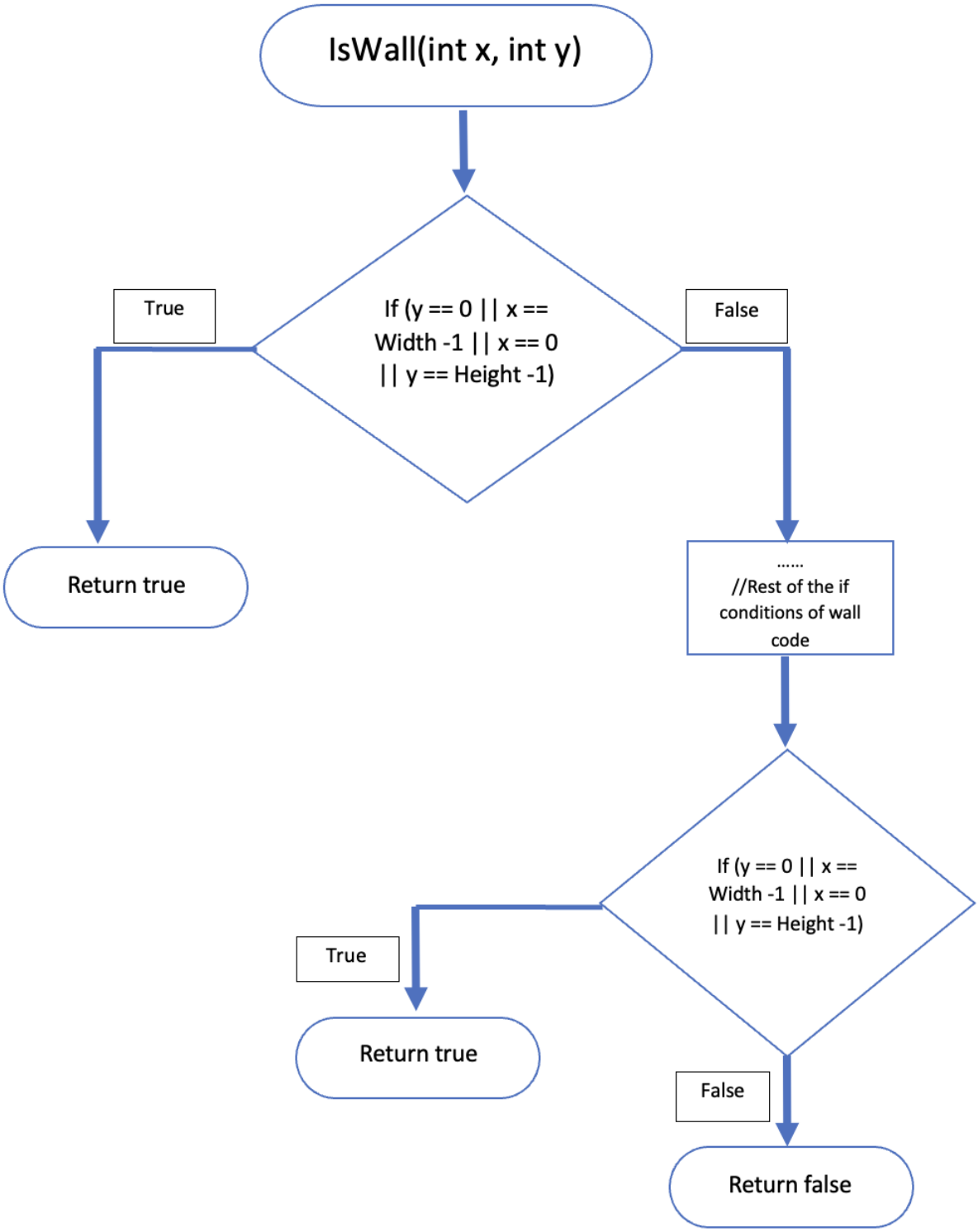
After exiting the while loop the function **show_cursor()** is called.

FLOWCHARTS

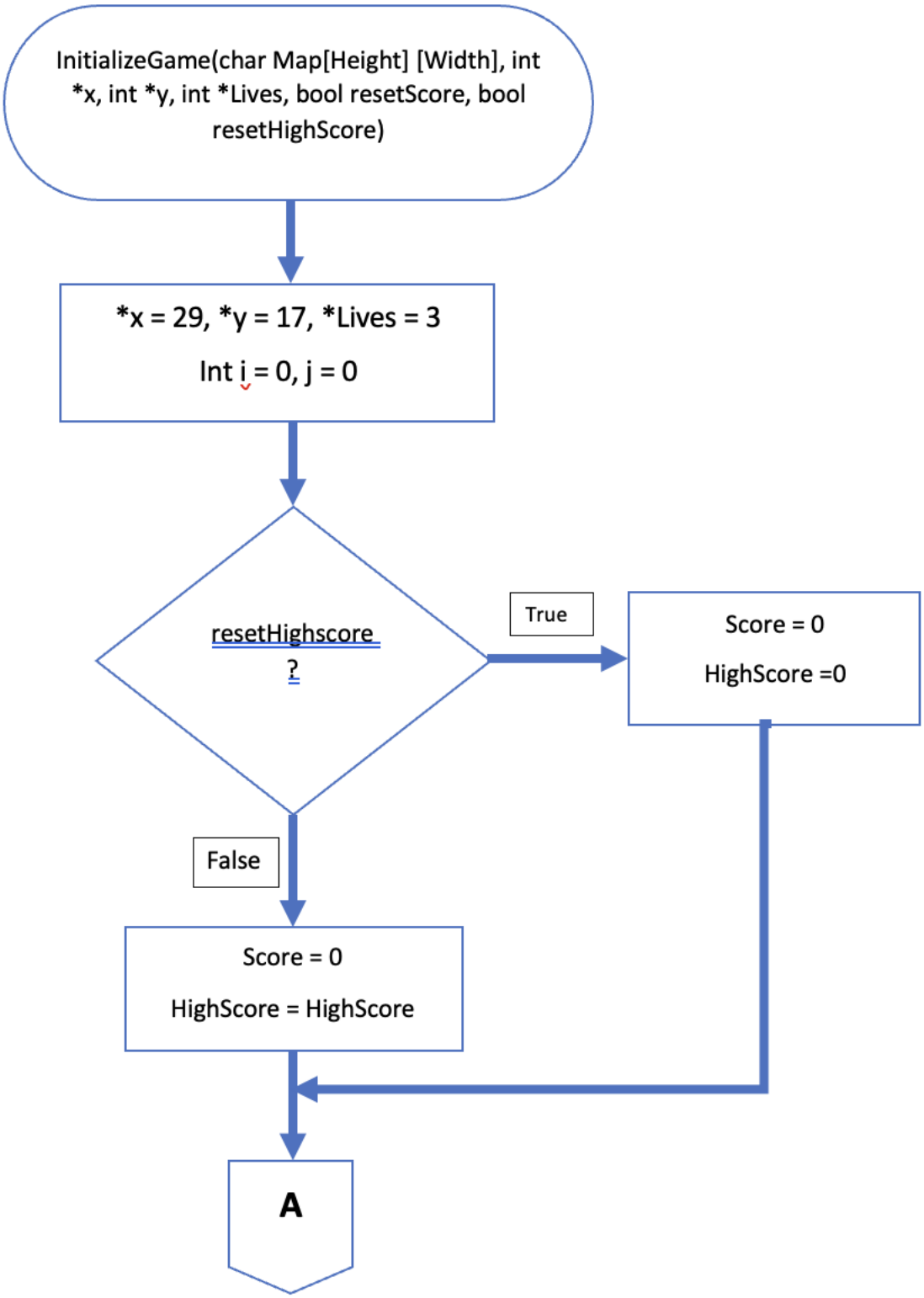
GENERAL MAIN FLOWCHART

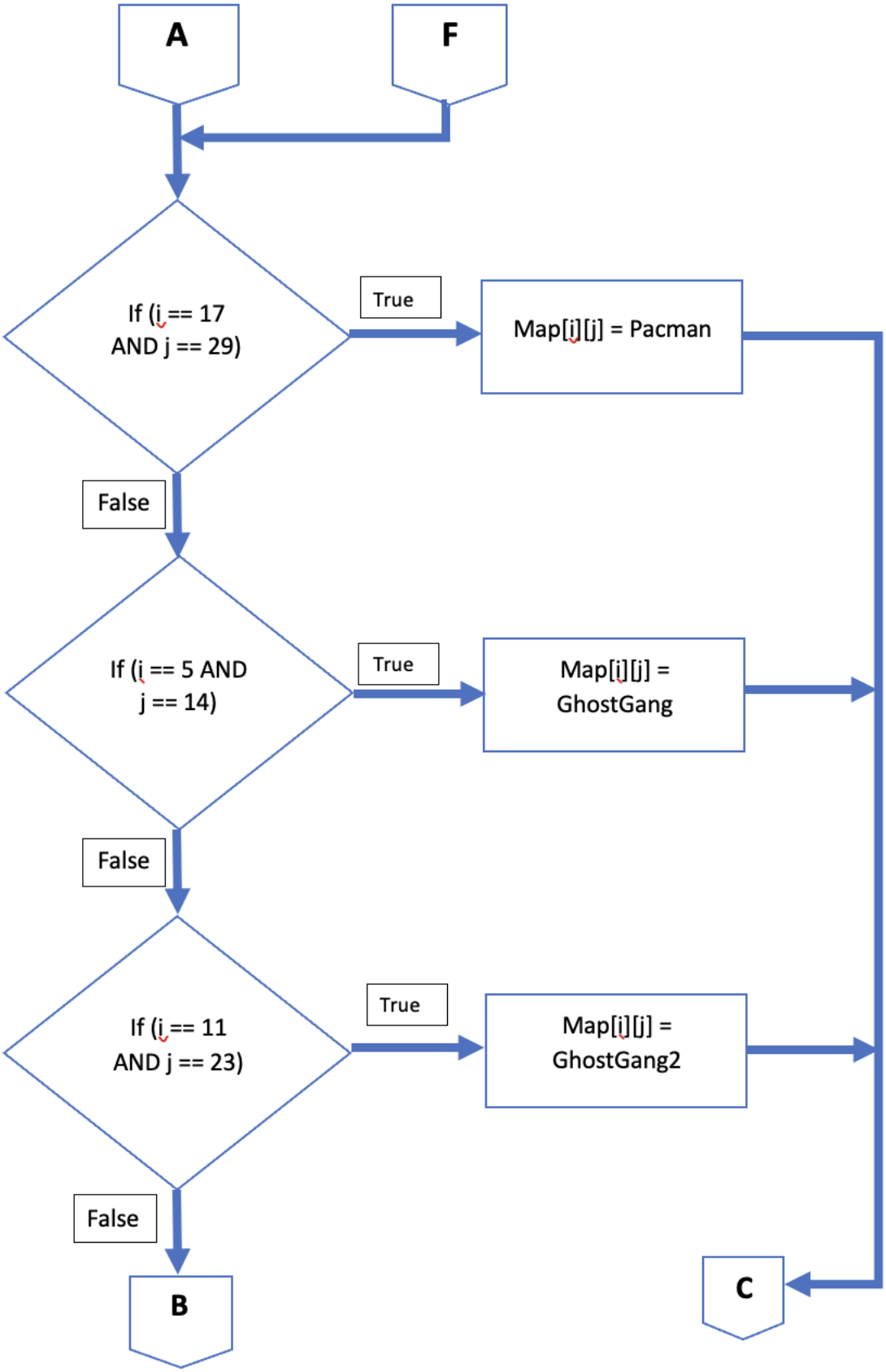


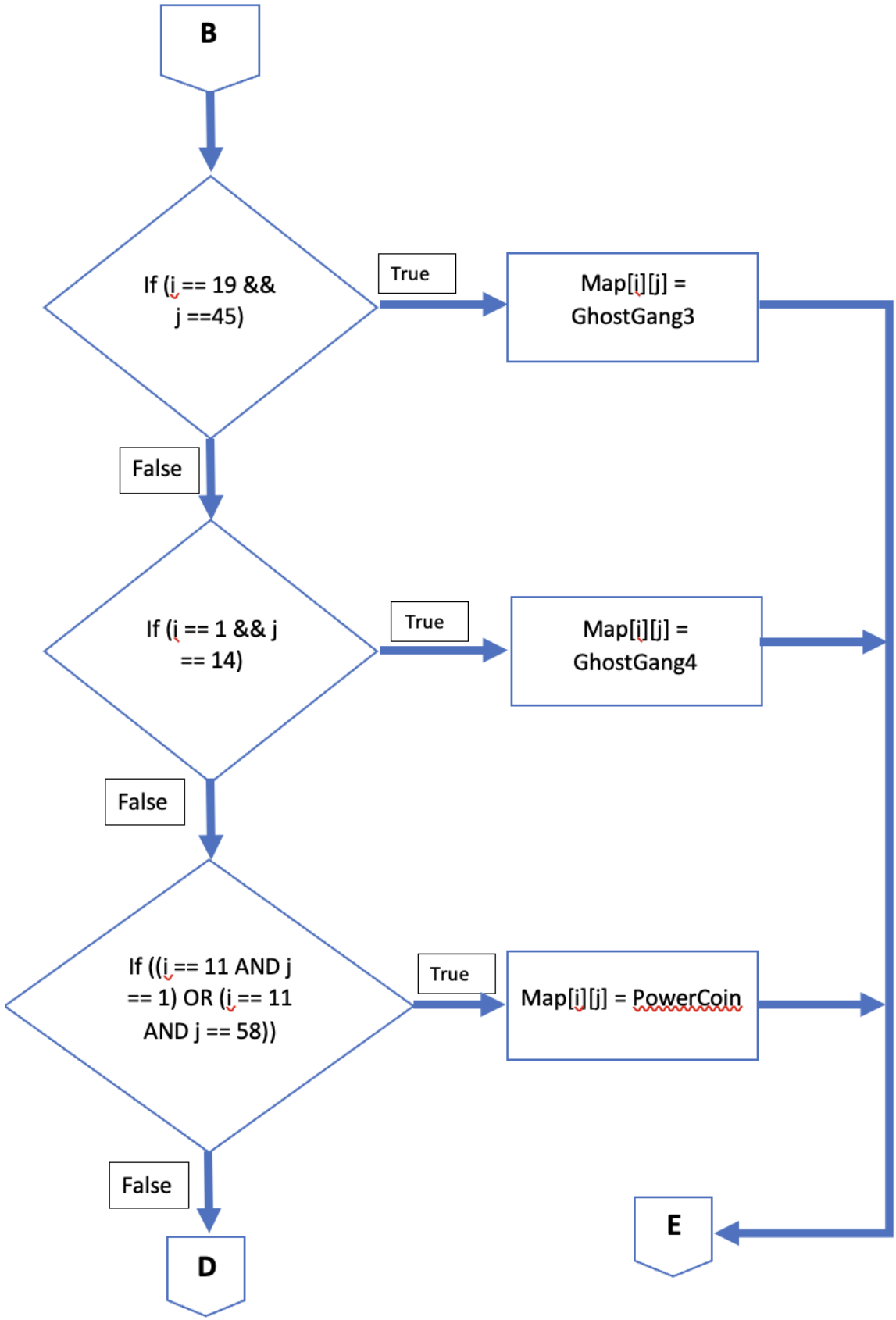
IsWall Function:

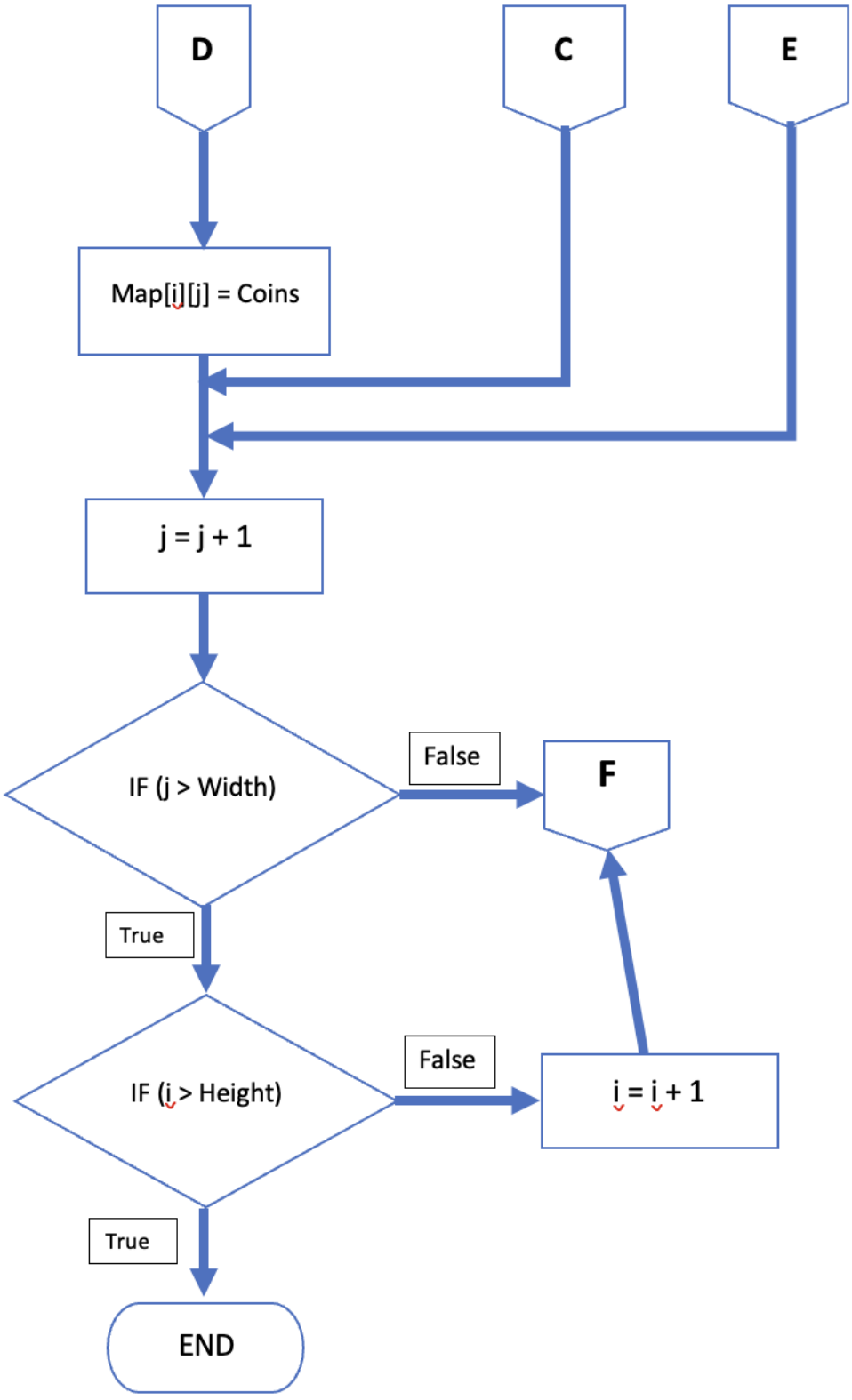


InitializeGame Function

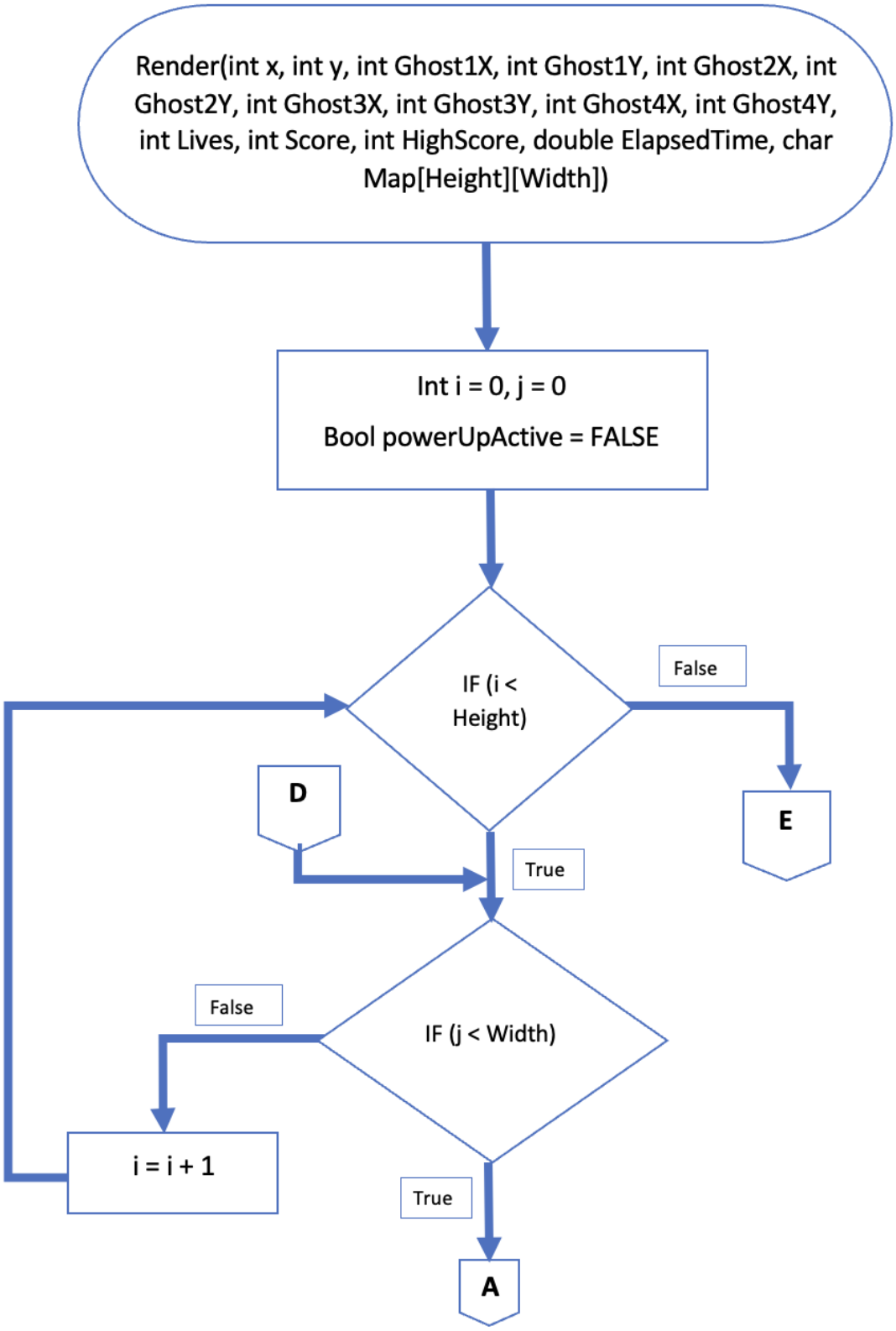


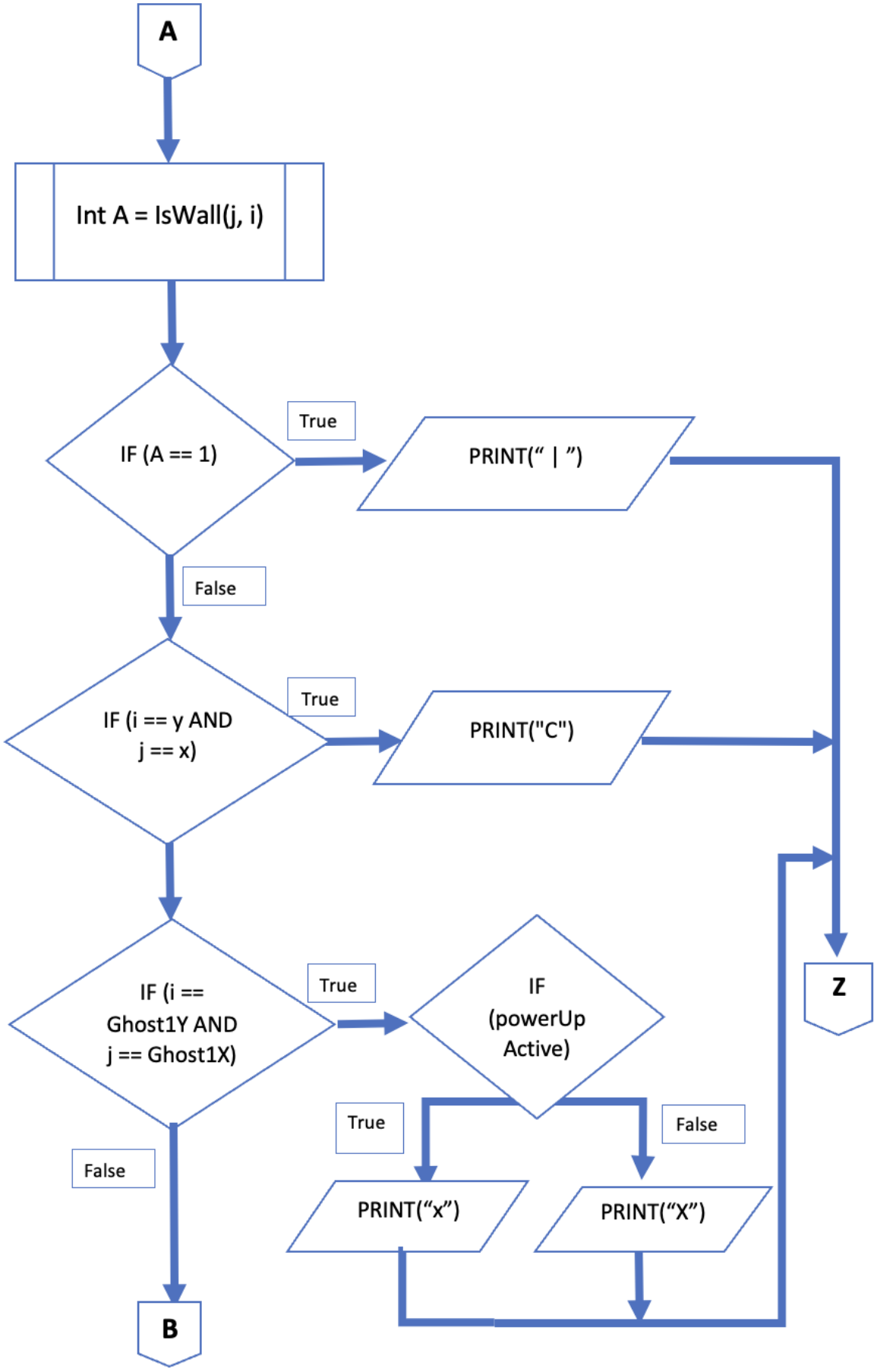

A
F
If (i == 17
AND j == 29)
True
Map[i][j] = Pacman
False
If (i == 5 AND
j == 14)
True
Map[i][j] =
GhostGang
False
If (i == 11
AND j == 23)
True
Map[i][j] =
GhostGang2
False
B
C

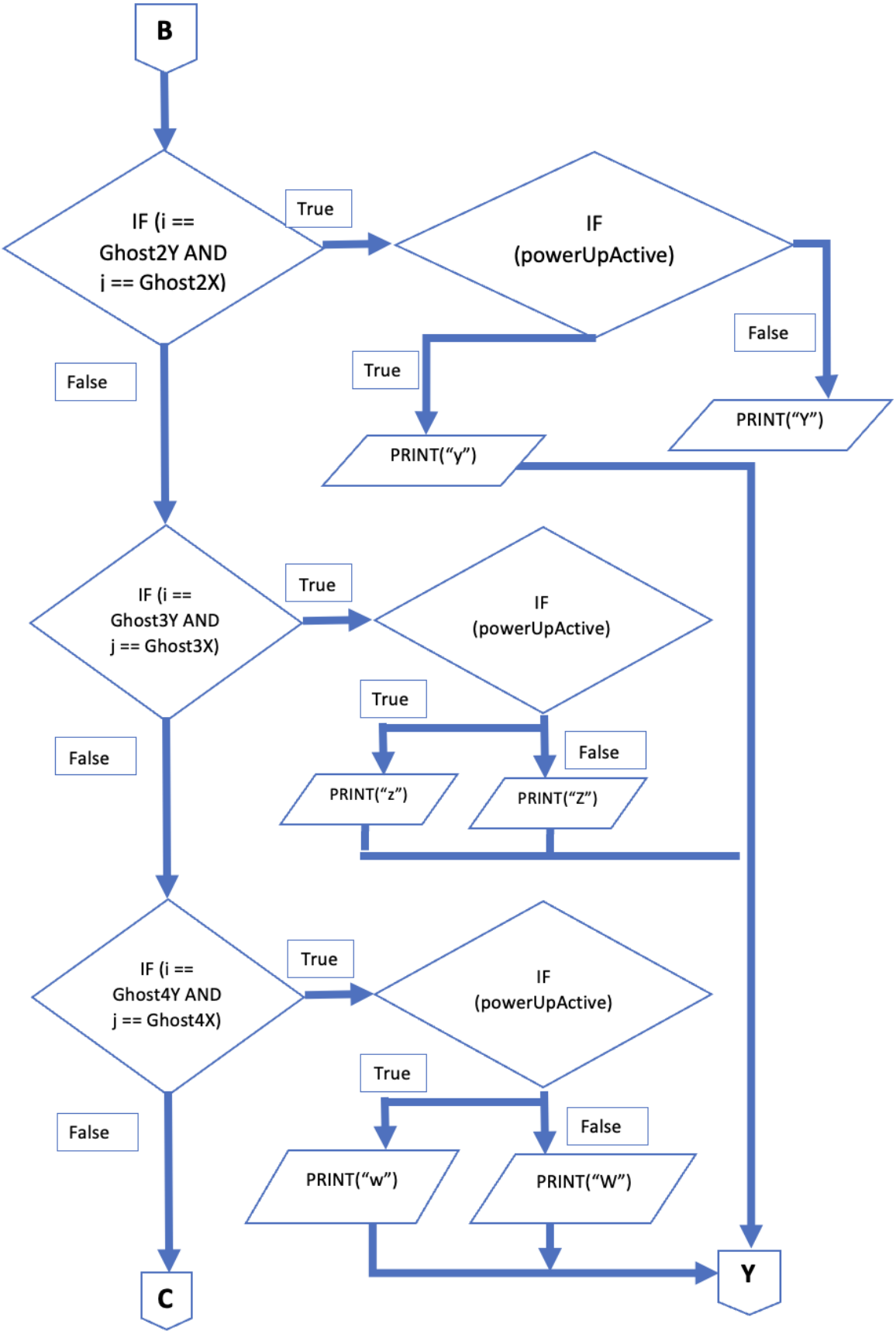

B
If (i == 19 && j ==45)
True
Map[i][j] = GhostGang3
False
If (i == 1 && j == 14)
True
Map[i][j] = GhostGang4
False
If ((i == 11 AND j == 1) OR (i == 11 AND j == 58))
True
Map[i][j] = PowerCoin
False
D
E

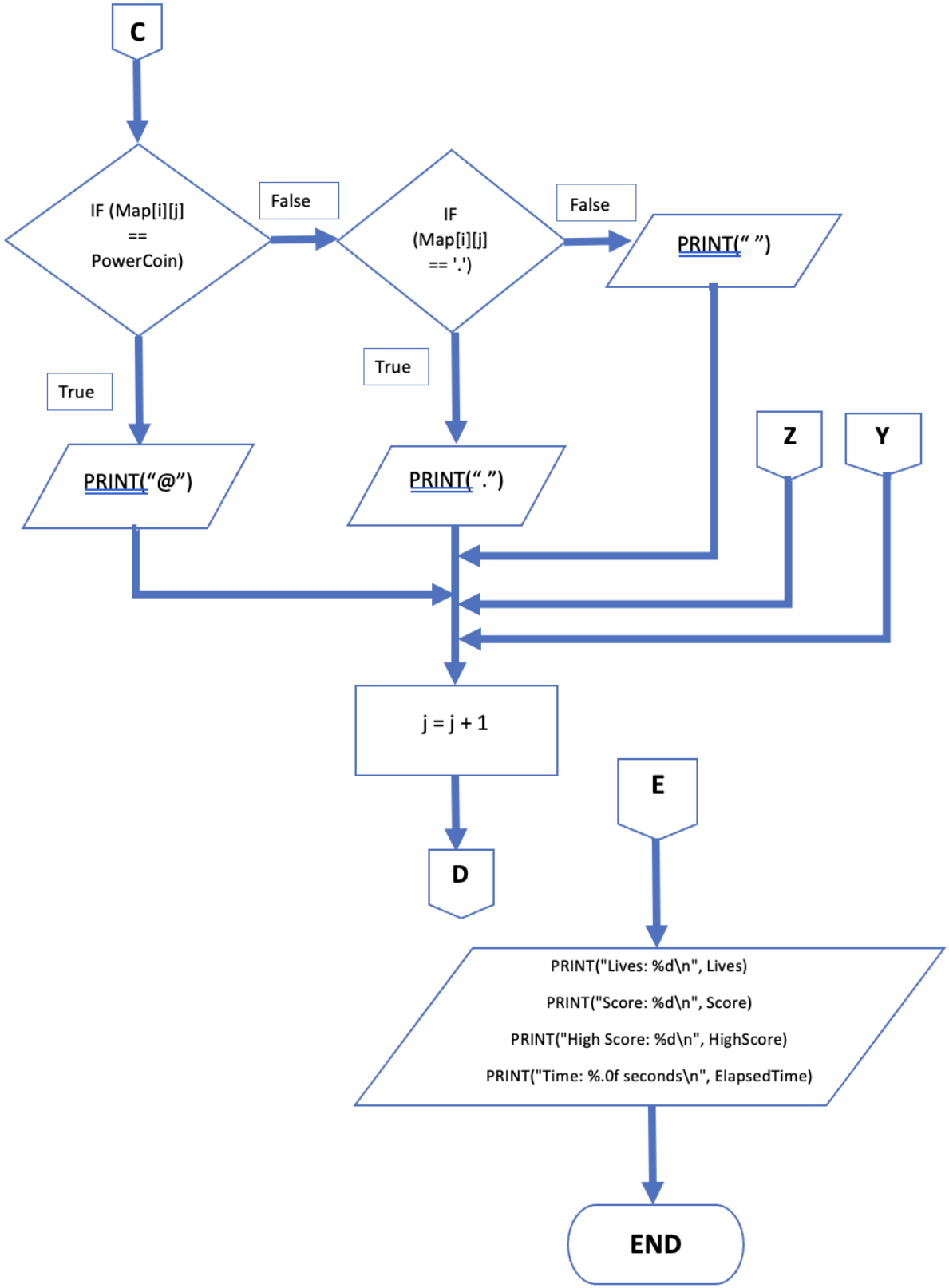

D
C
E
Map[i][j] = Coins
j = j + 1
IF (j > Width)
False
F
True
IF (i > Height)
False
i = i + 1
True
END

Render Function

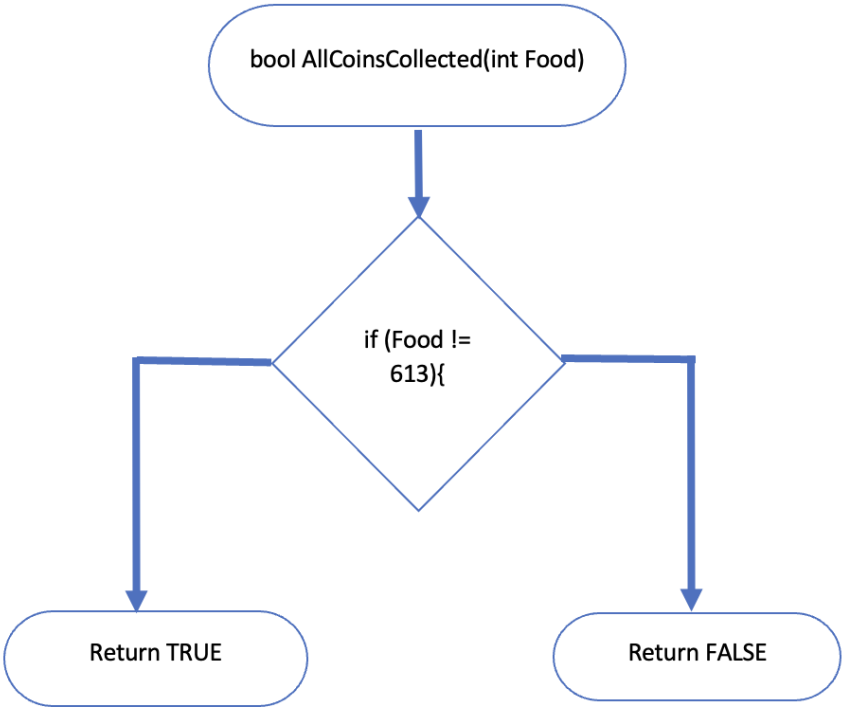



A
Int A = IsWall(j, i)
IF (A == 1)
True
PRINT(“ | ”)
False
IF (i == y AND j == x)
True
PRINT("C")
IF (i == Ghost1Y AND j == Ghost1X)
True
IF (powerUp Active)
Z
True
False
PRINT(“x”)
PRINT(“X”)
False
B

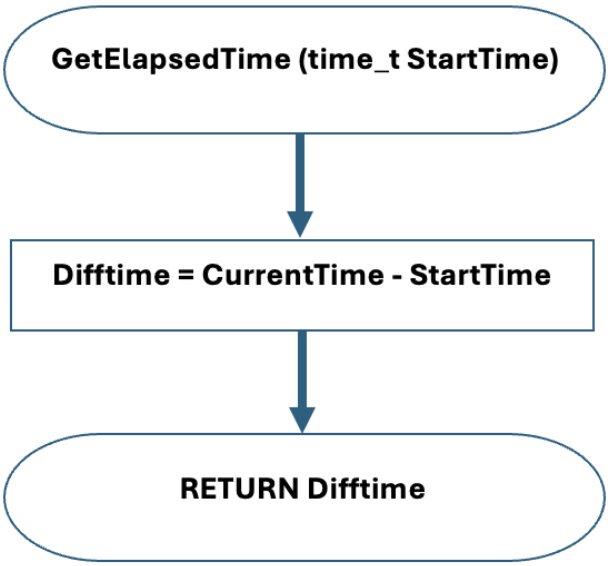

B
IF (i == Ghost2Y AND j == Ghost2X)
True
False
IF (powerUpActive)
True
False
PRINT("y")
PRINT("Y")
IF (i == Ghost3Y AND j == Ghost3X)
True
False
IF (powerUpActive)
True
False
PRINT("z")
PRINT("Z")
IF (i == Ghost4Y AND j == Ghost4X)
True
False
IF (powerUpActive)
True
False
PRINT("w")
PRINT("W")
C
Y


C
IF (Map[i][j] == PowerCoin)
False
IF (Map[i][j] == '.')
False
PRINT(" ")
True
True
PRINT("@")
PRINT(".")
Z
Y
j = j + 1
D
E
PRINT("Lives: %d\n", Lives)
PRINT("Score: %d\n", Score)
PRINT("High Score: %d\n", HighScore)
PRINT("Time: %.0f seconds\n", ElapsedTime)
END

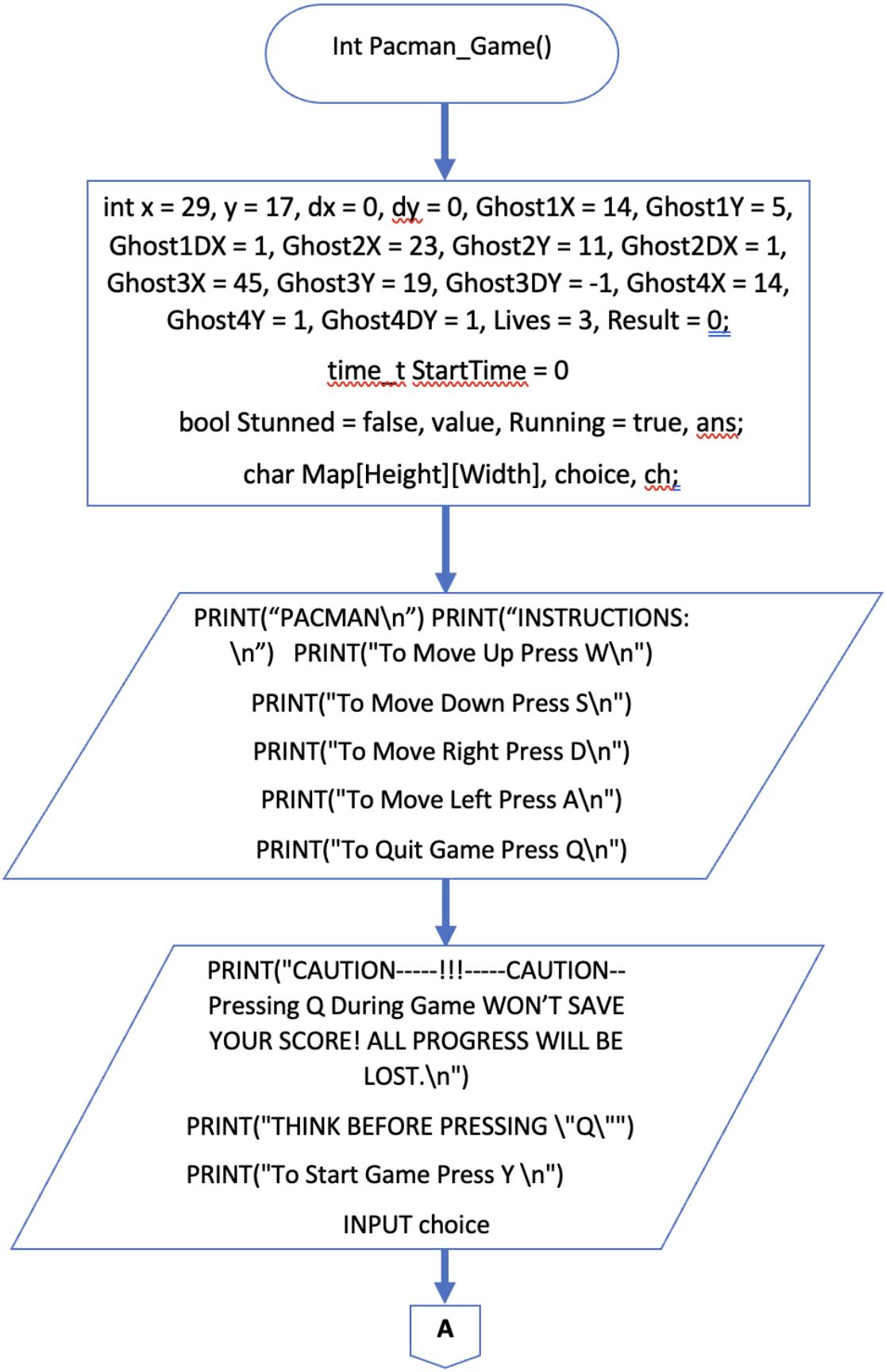
AllCoinsCollected Function

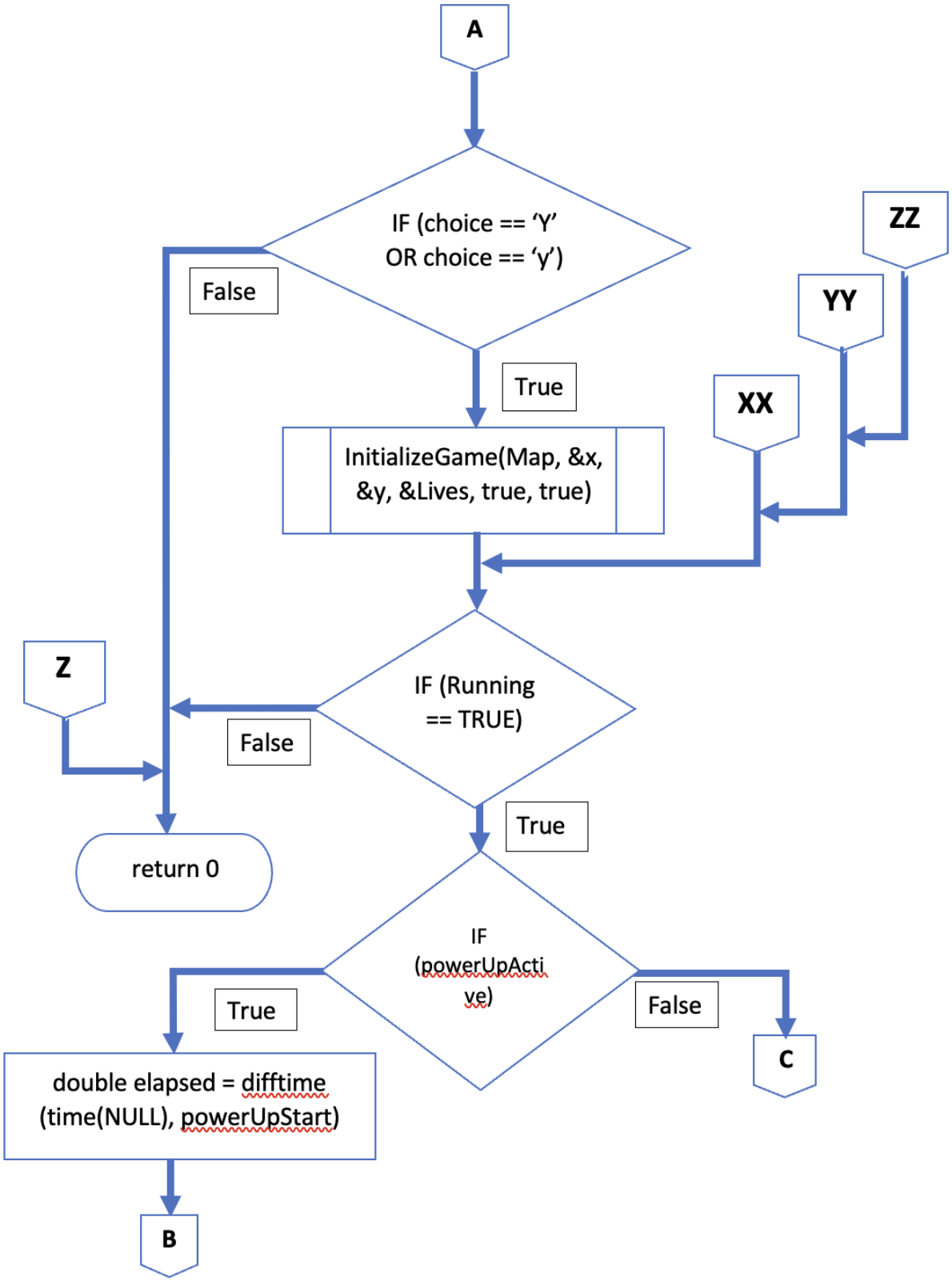


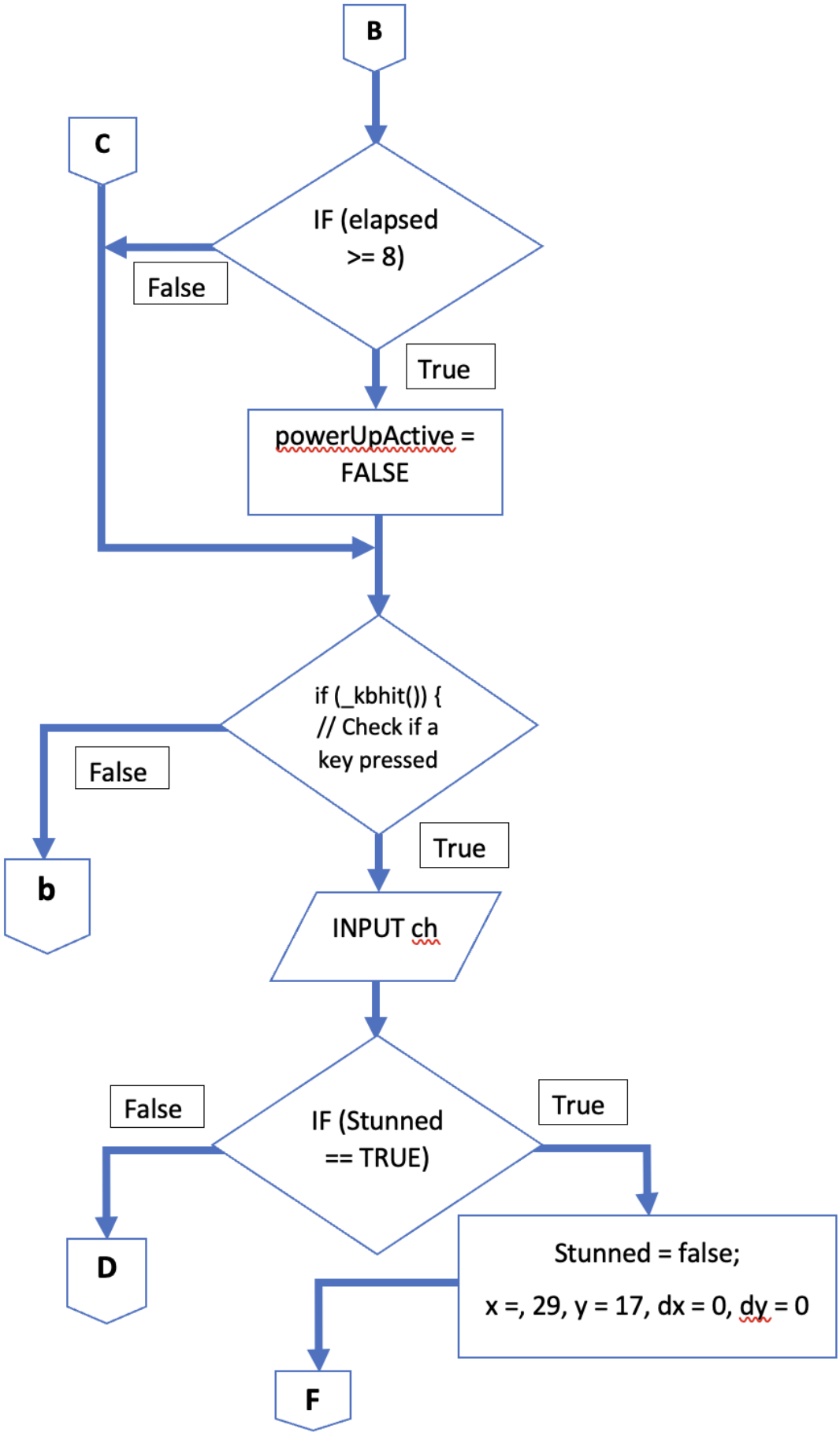
GetElapsedTime Function

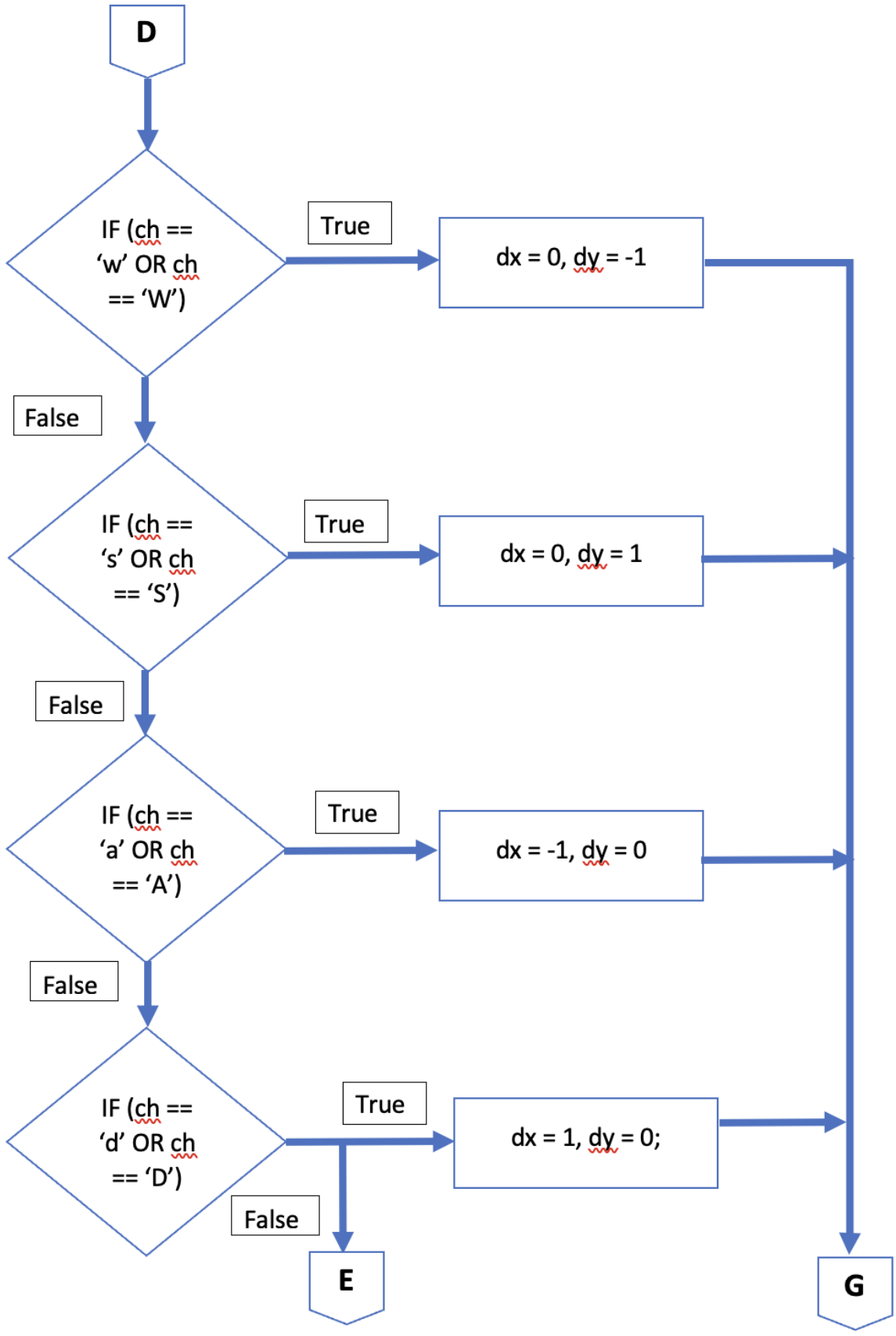


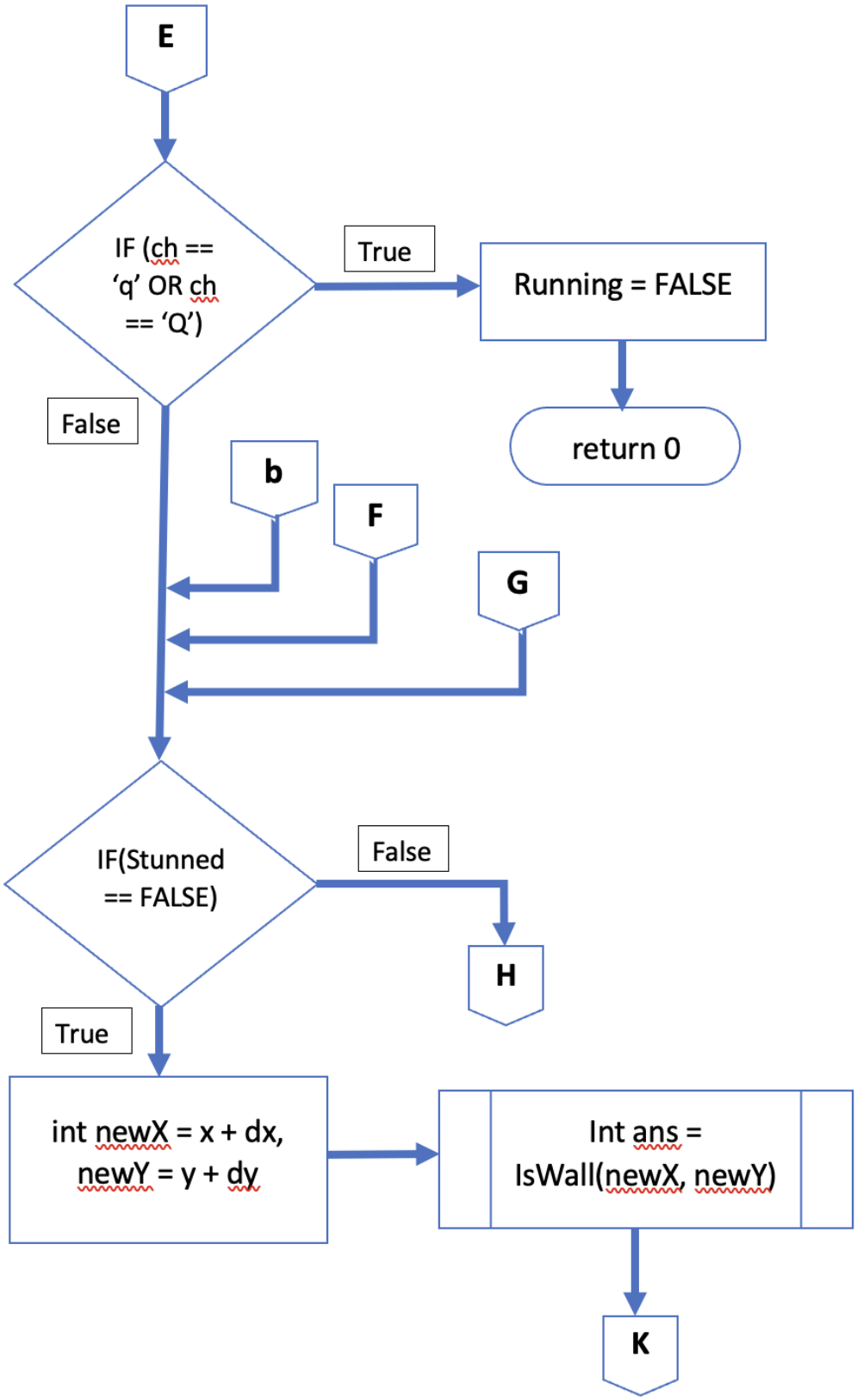
Pacman_Game Function

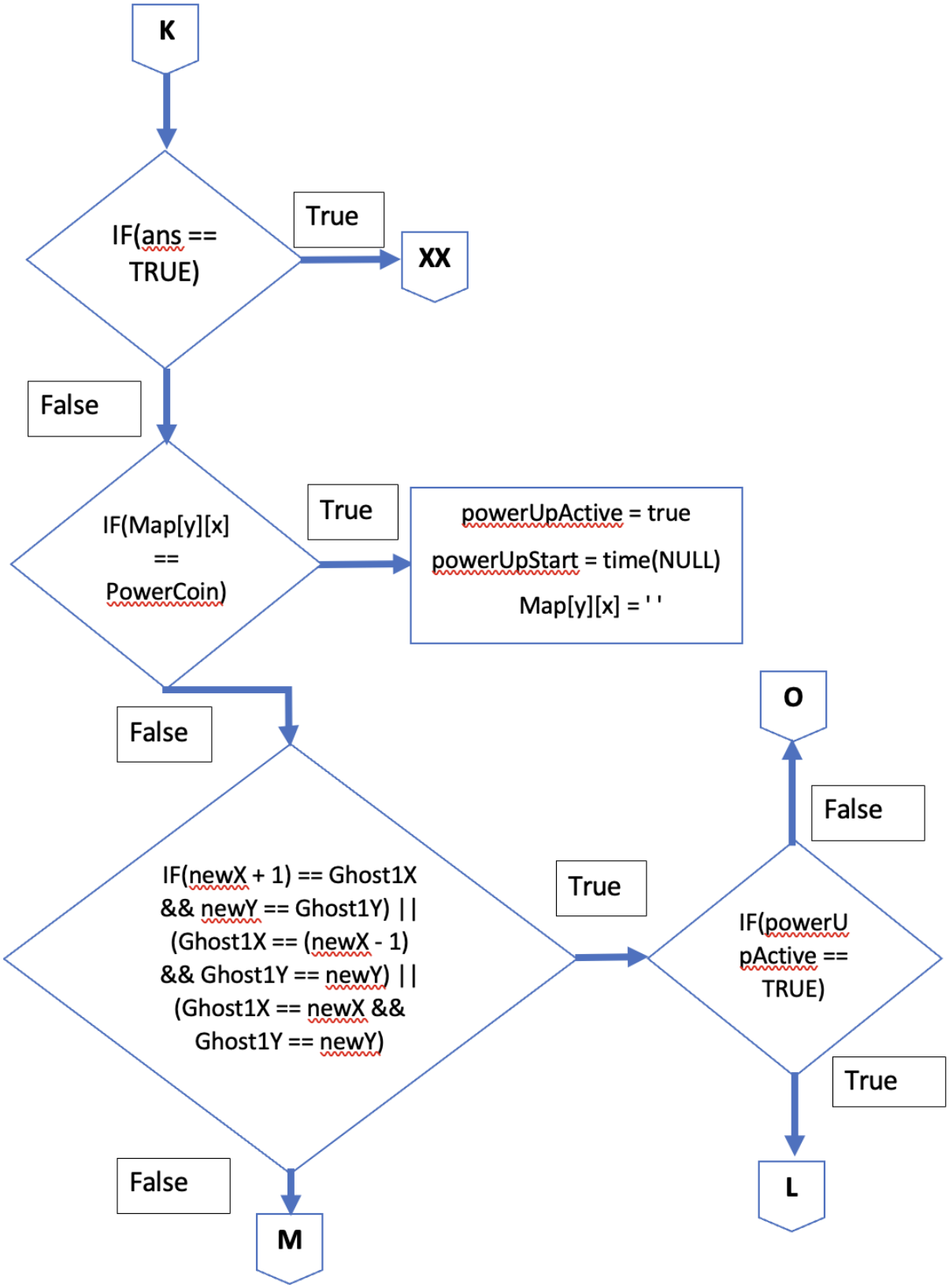


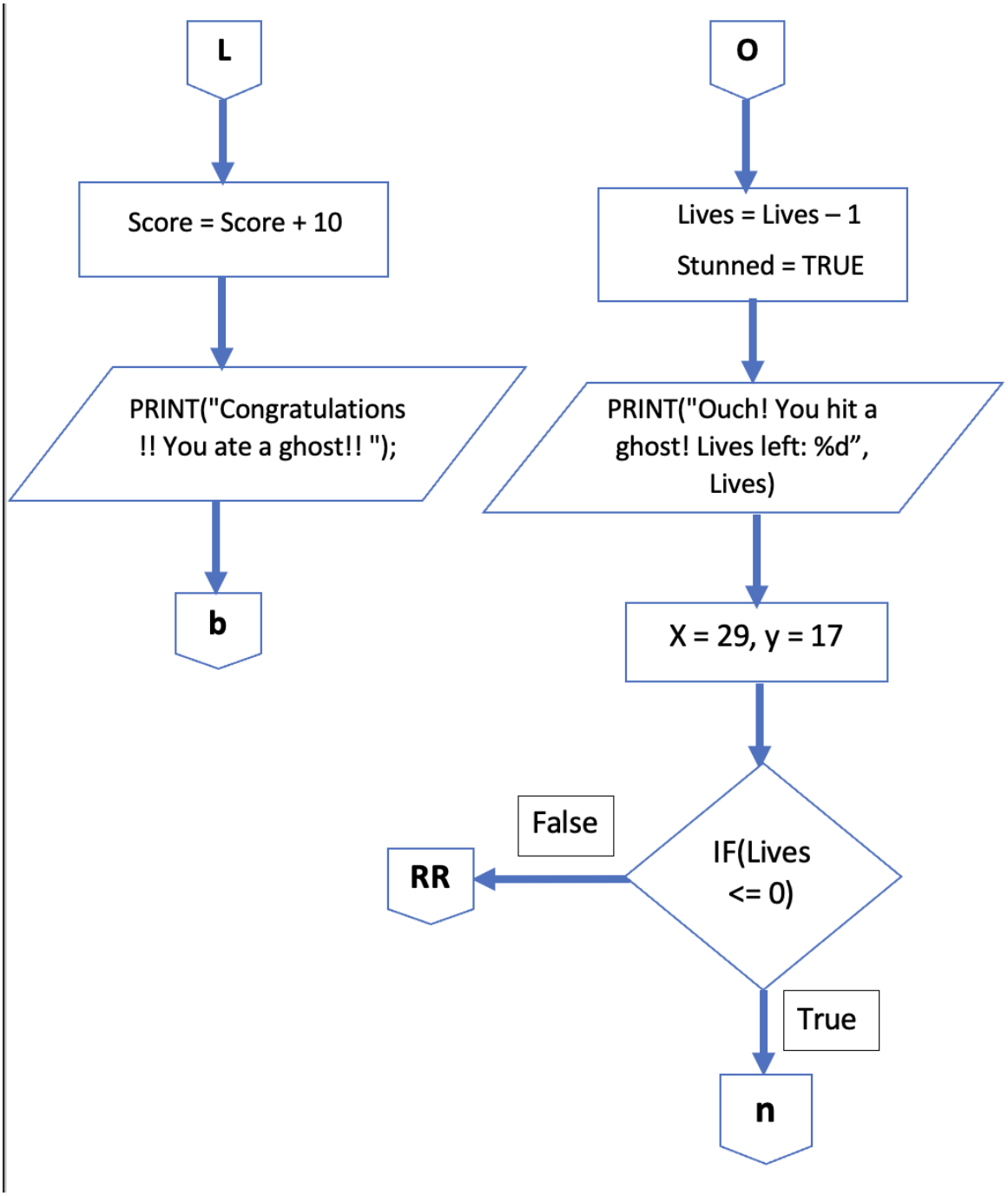

A
IF (choice == 'Y' OR choice == 'y')
False
True
InitializeGame(Map, &x, &y, &Lives, true, true)
ZZ
YY
XX
Z
IF (Running == TRUE)
False
True
return 0
IF (powerUpActive)
True
False
double elapsed = difftime (time(NULL), powerUpStart)
C
B

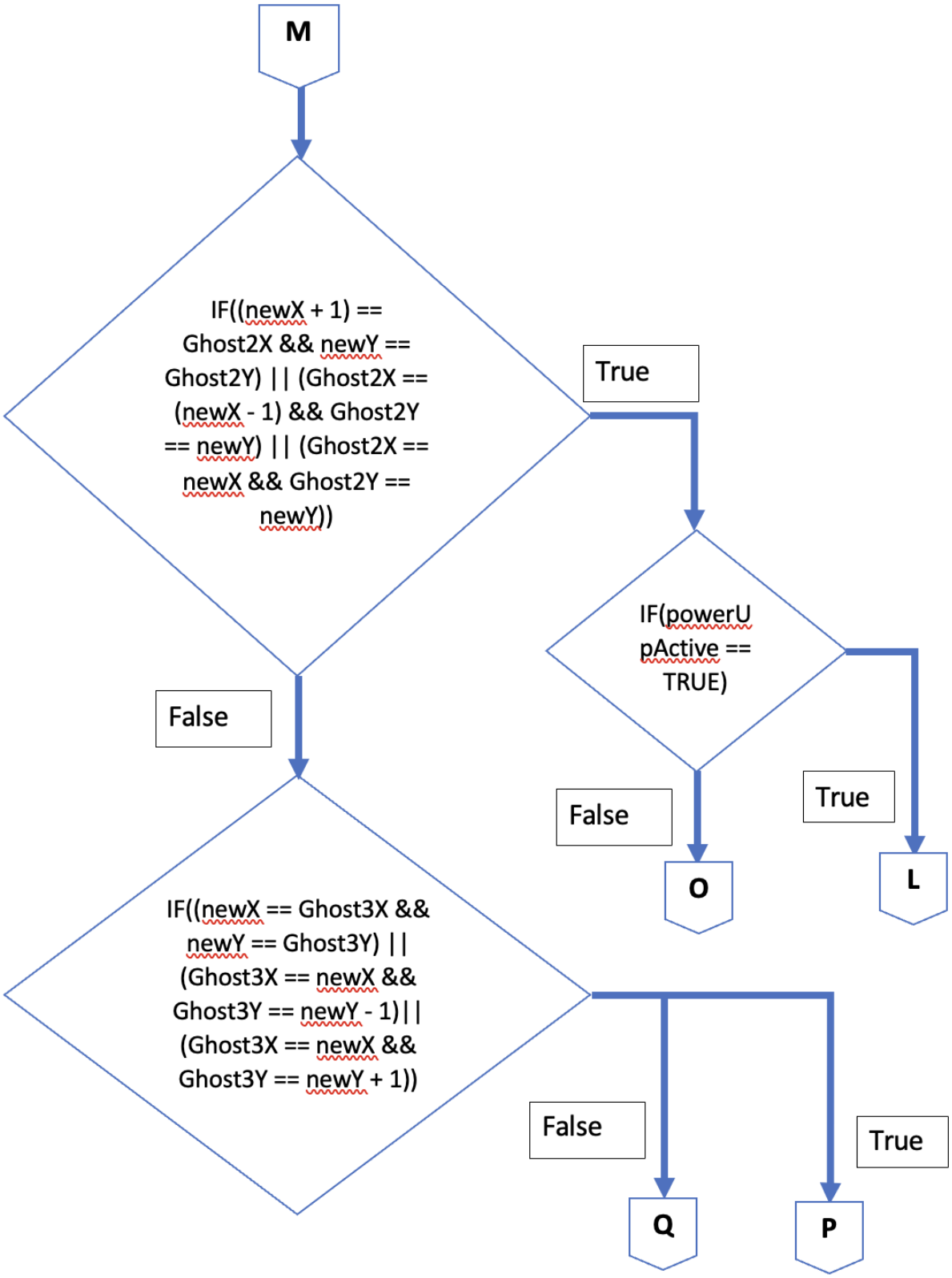

B
C
IF (elapsed >= 8)
False
True
powerUpActive = FALSE
if (_kbhit()) { // Check if a key pressed
False
True
b
INPUT ch
False
IF (Stunned == TRUE)
True
D
Stunned = false;
x =, 29, y = 17, dx = 0, dy = 0
F

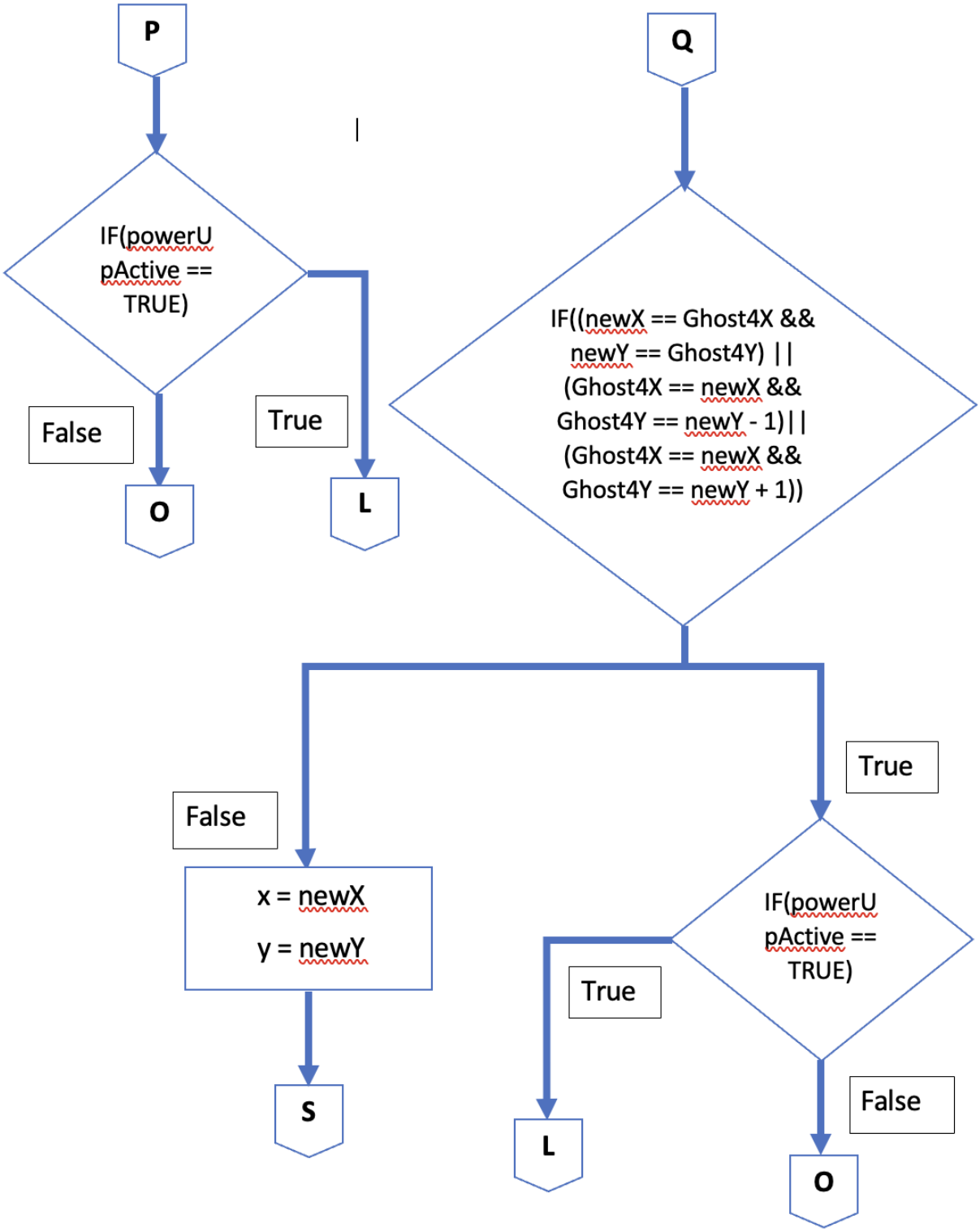

D
IF (ch == 'w' OR ch == 'W')
True
dx = 0, dy = -1
False
IF (ch == 's' OR ch == 'S')
True
dx = 0, dy = 1
False
IF (ch == 'a' OR ch == 'A')
True
dx = -1, dy = 0
False
IF (ch == 'd' OR ch == 'D')
True
dx = 1, dy = 0;
False
E
G

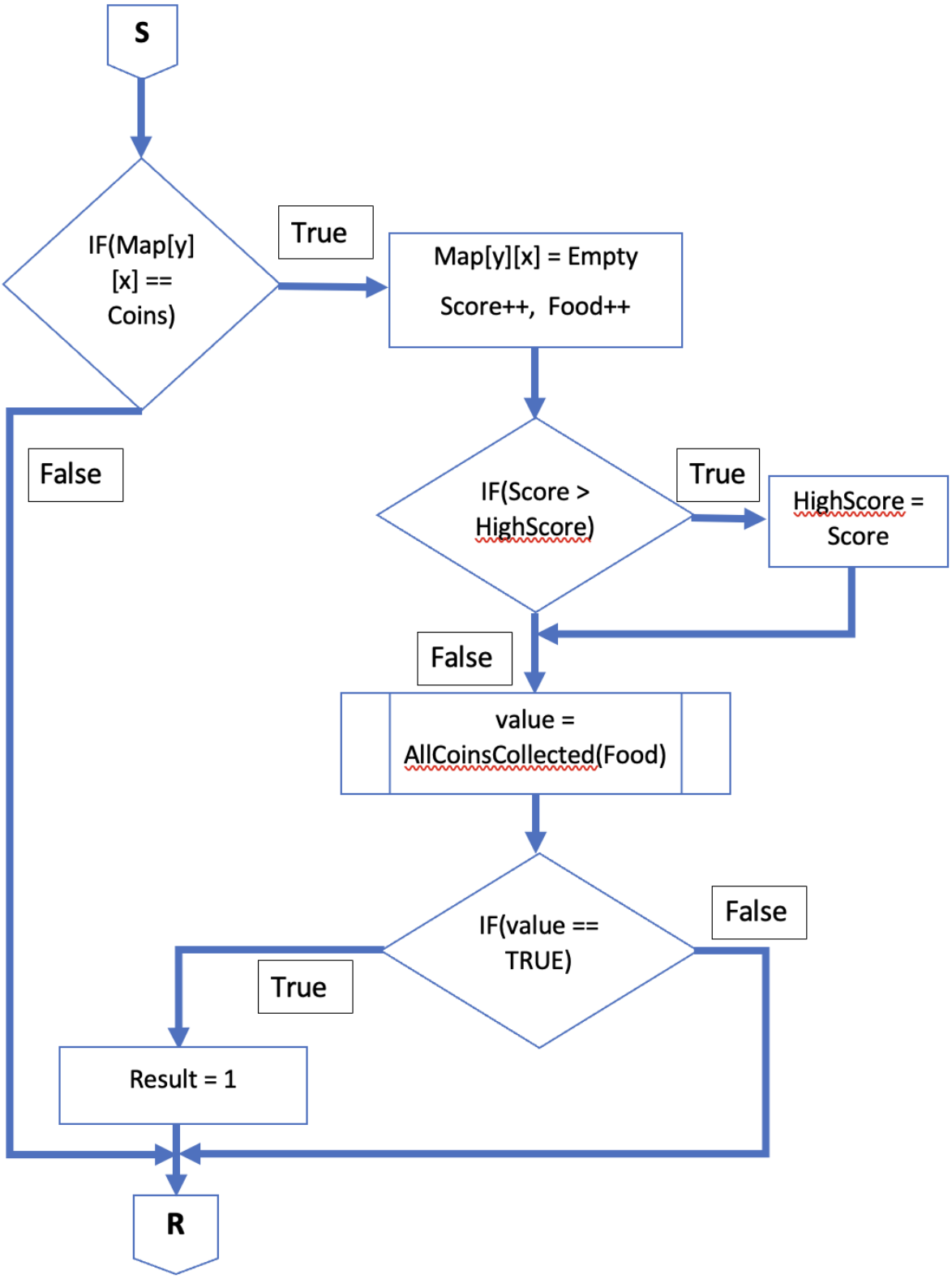

E
IF (ch == 'q' OR ch == 'Q')
True
Running = FALSE
return 0
False
b
F
G
IF(Stunned == FALSE)
False
H
True
int newX = x + dx, newY = y + dy
Int ans = IsWall(newX, newY)
K

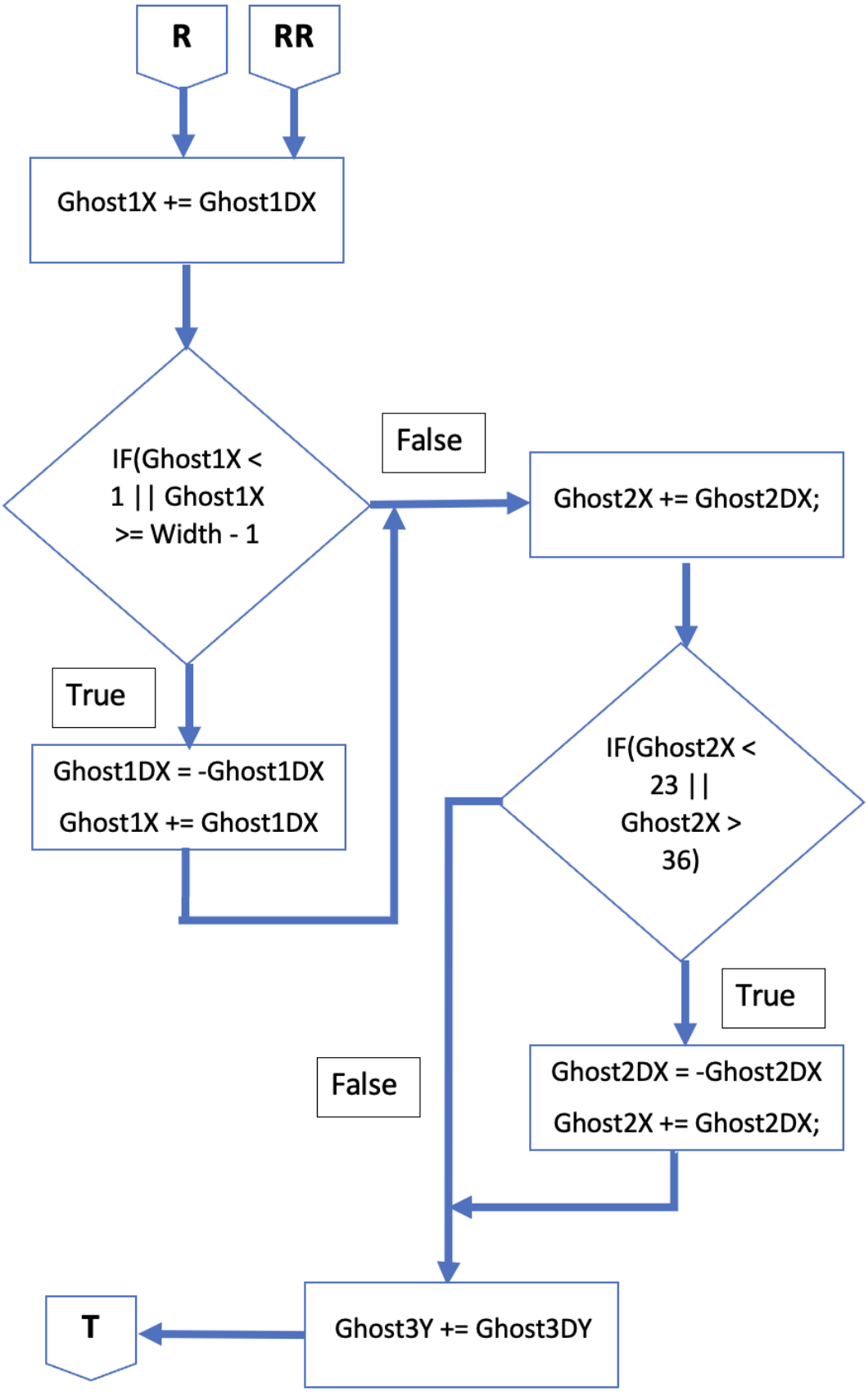

K
IF(ans == TRUE)
True
XX
False
IF(Map[y][x] == PowerCoin)
True
powerUpActive = true
powerUpStart = time(NULL)
Map[y][x] = ' '
False
IF(newX + 1) == Ghost1X && newY == Ghost1Y) || (Ghost1X == (newX - 1) && Ghost1Y == newY) || (Ghost1X == newX && Ghost1Y == newY)
True
IF(powerUpActive == TRUE)
O
False
True
L
False
M

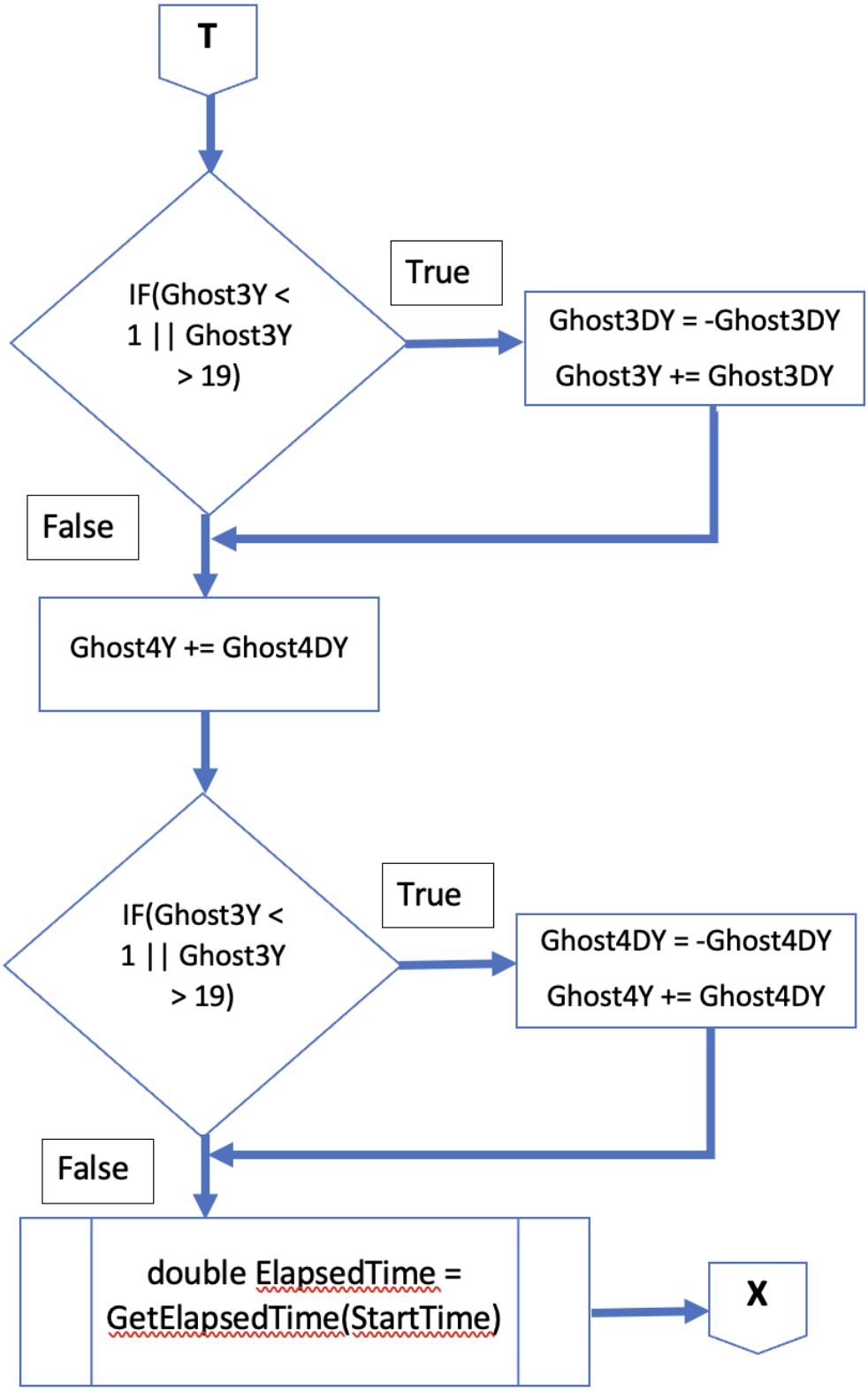

L
Score = Score + 10
PRINT("Congratulations !! You ate a ghost!! ");
b
O
Lives = Lives – 1
Stunned = TRUE
PRINT("Ouch! You hit a ghost! Lives left: %d", Lives)
X = 29, y = 17
IF(Lives <= 0)
False
RR
True
n

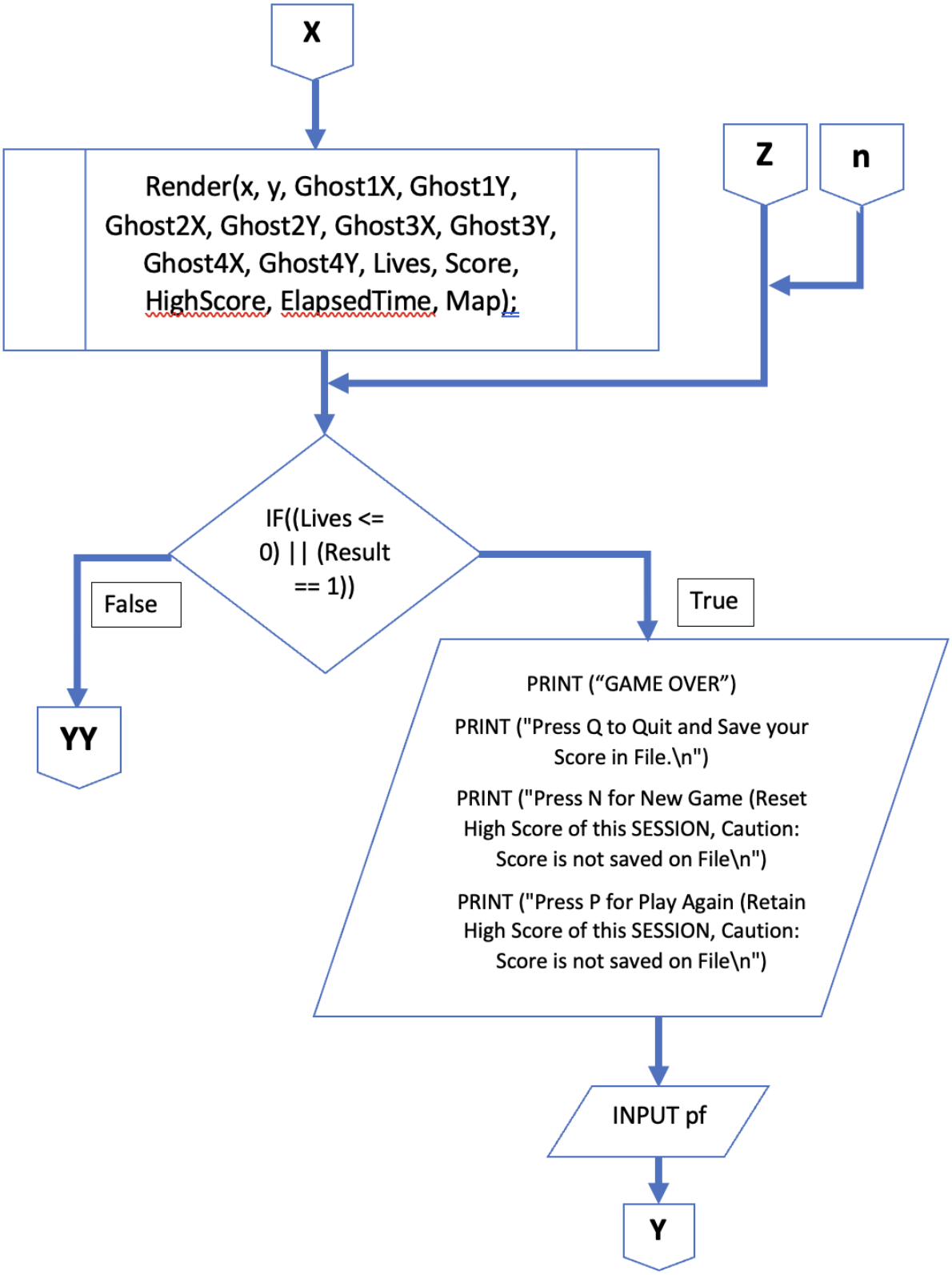

M
IF((newX + 1) == Ghost2X && newY == Ghost2Y) || (Ghost2X == (newX - 1) && Ghost2Y == newY) || (Ghost2X == newX && Ghost2Y == newY))
True
False
IF(powerUpActive == TRUE)
False
True
O
L
IF((newX == Ghost3X && newY == Ghost3Y) || (Ghost3X == newX && Ghost3Y == newY - 1)|| (Ghost3X == newX && Ghost3Y == newY + 1))
False
True
Q
P

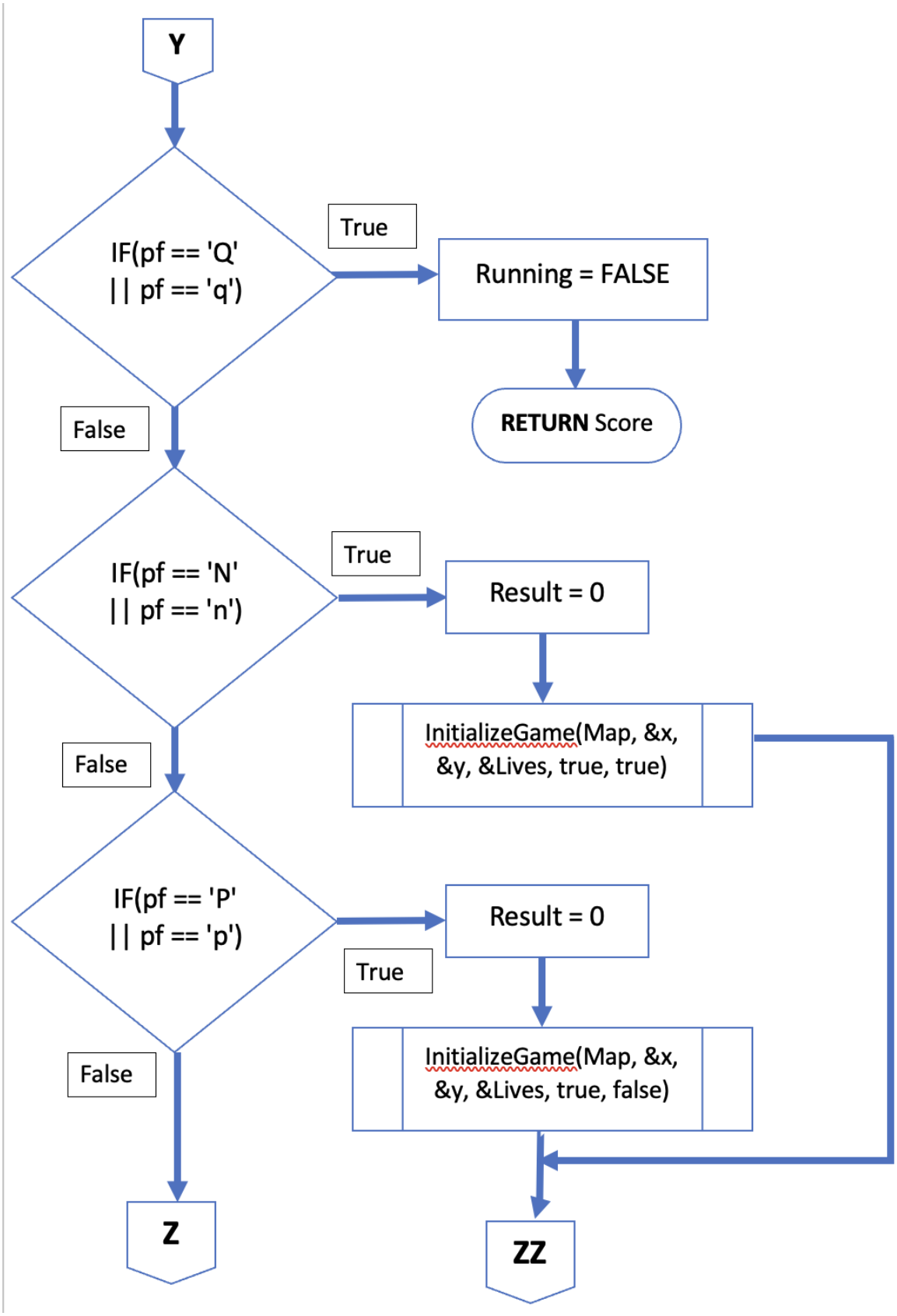

P
IF(powerU pActive == TRUE)
False
O
True
L
Q
IF((newX == Ghost4X && newY == Ghost4Y) || (Ghost4X == newX && Ghost4Y == newY - 1)|| (Ghost4X == newX && Ghost4Y == newY + 1))
False
x = newX
y = newY
S
True
IF(powerU pActive == TRUE)
True
L
False
O


S
IF(Map[y][x] == Coins)
True
Map[y][x] = Empty
Score++, Food++
False
IF(Score > HighScore)
True
HighScore = Score
False
value = AllCoinsCollected(Food)
IF(value == TRUE)
False
True
Result = 1
R

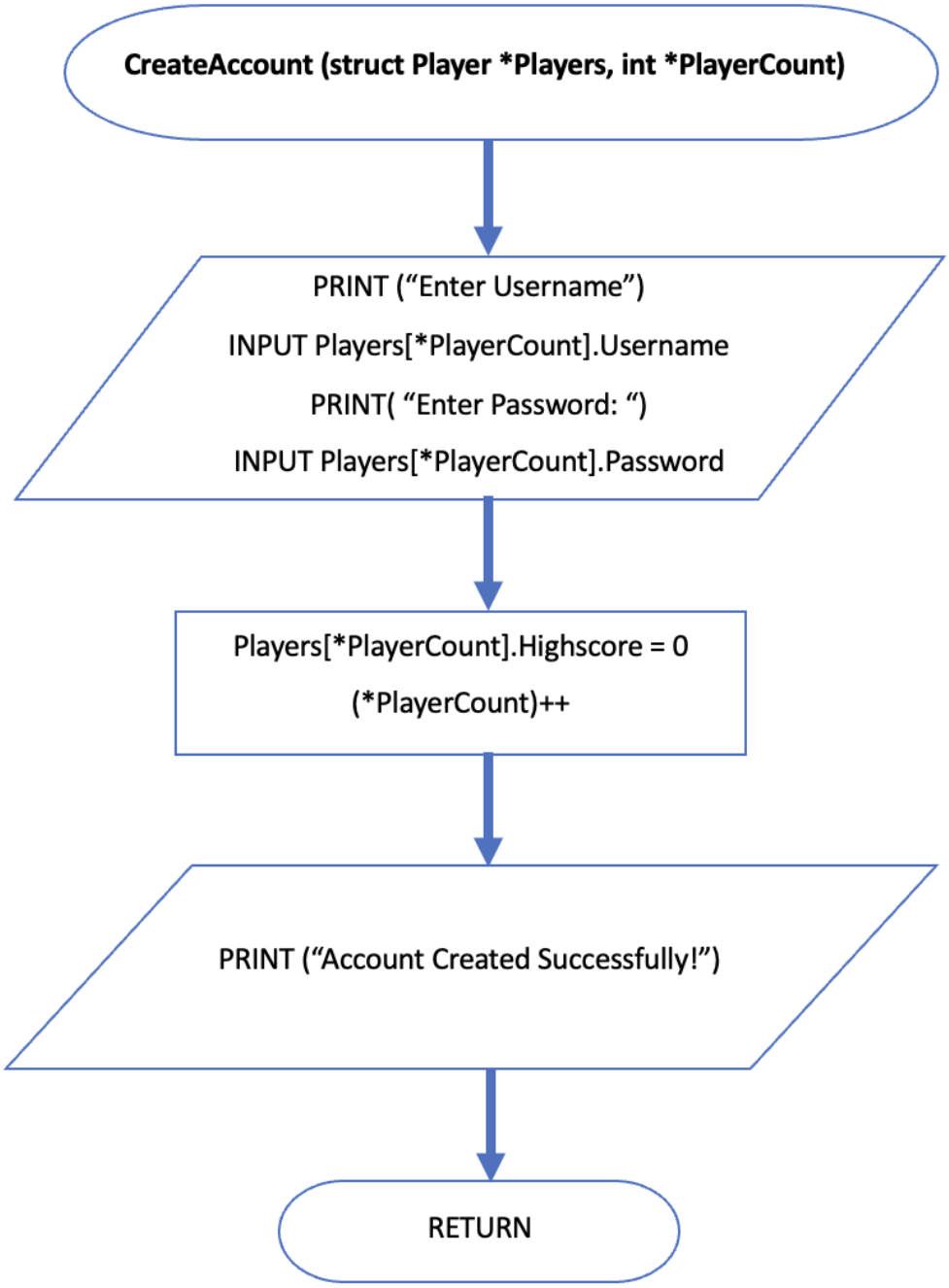

R
RR
Ghost1X += Ghost1DX
IF(Ghost1X < 1 || Ghost1X >= Width - 1
False
Ghost2X += Ghost2DX;
True
Ghost1DX = -Ghost1DX
Ghost1X += Ghost1DX
IF(Ghost2X < 23 || Ghost2X > 36)
True
Ghost2DX = -Ghost2DX
Ghost2X += Ghost2DX;
False
T
Ghost3Y += Ghost3DY


T
IF(Ghost3Y < 1 || Ghost3Y > 19)
True
Ghost3DY = -Ghost3DY
Ghost3Y += Ghost3DY
False
Ghost4Y += Ghost4DY
IF(Ghost3Y < 1 || Ghost3Y > 19)
True
Ghost4DY = -Ghost4DY
Ghost4Y += Ghost4DY
False
double ElapsedTime = GetElapsedTime(StartTime)
X

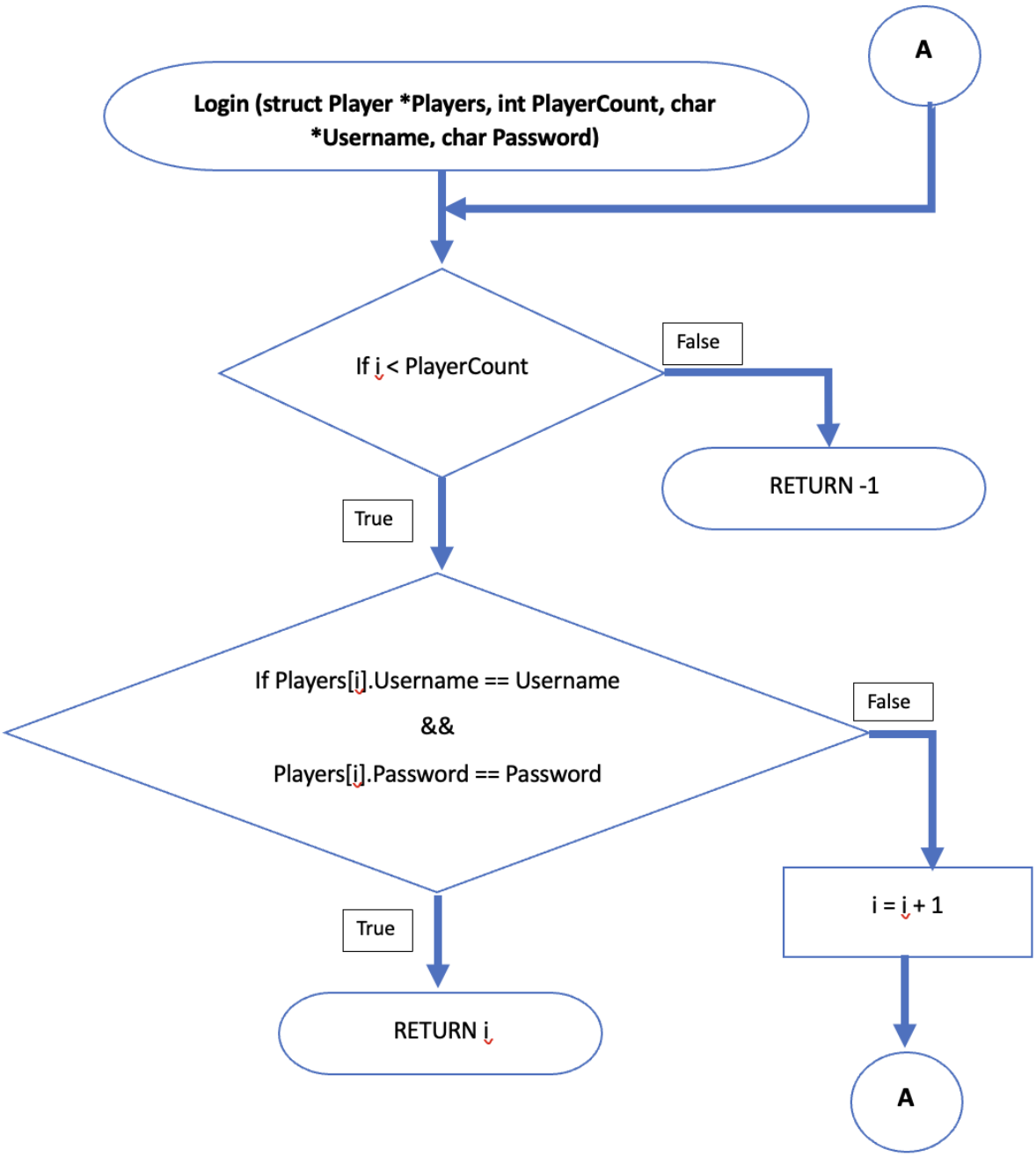

X
Render(x, y, Ghost1X, Ghost1Y, Ghost2X, Ghost2Y, Ghost3X, Ghost3Y, Ghost4X, Ghost4Y, Lives, Score, HighScore, ElapsedTime, Map);
Z
n
IF((Lives <= 0) || (Result == 1))
False
True
YY
PRINT ("GAME OVER")
PRINT ("Press Q to Quit and Save your Score in File.\n")
PRINT ("Press N for New Game (Reset High Score of this SESSION, Caution: Score is not saved on File\n")
PRINT ("Press P for Play Again (Retain High Score of this SESSION, Caution: Score is not saved on File\n")
INPUT pf
Y


Y
IF(pf == 'Q'
|| pf == 'q')
True
Running = FALSE
RETURN Score
False
IF(pf == 'N'
|| pf == 'n')
True
Result = 0
InitializeGame(Map, &x,
&y, &Lives, true, true)
False
IF(pf == 'P'
|| pf == 'p')
Result = 0
True
InitializeGame(Map, &x,
&y, &Lives, true, false)
False
Z
ZZ

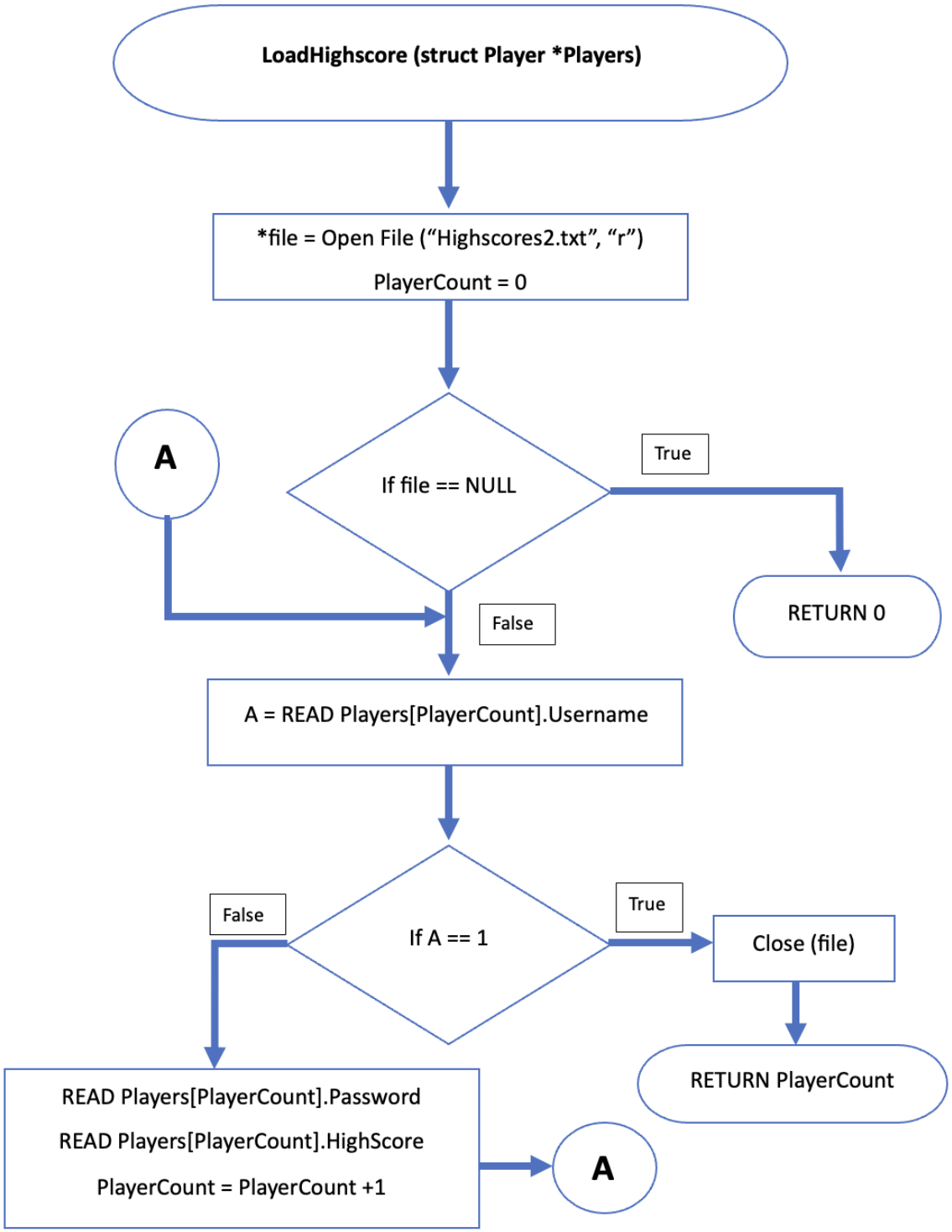
CreateAccount Function



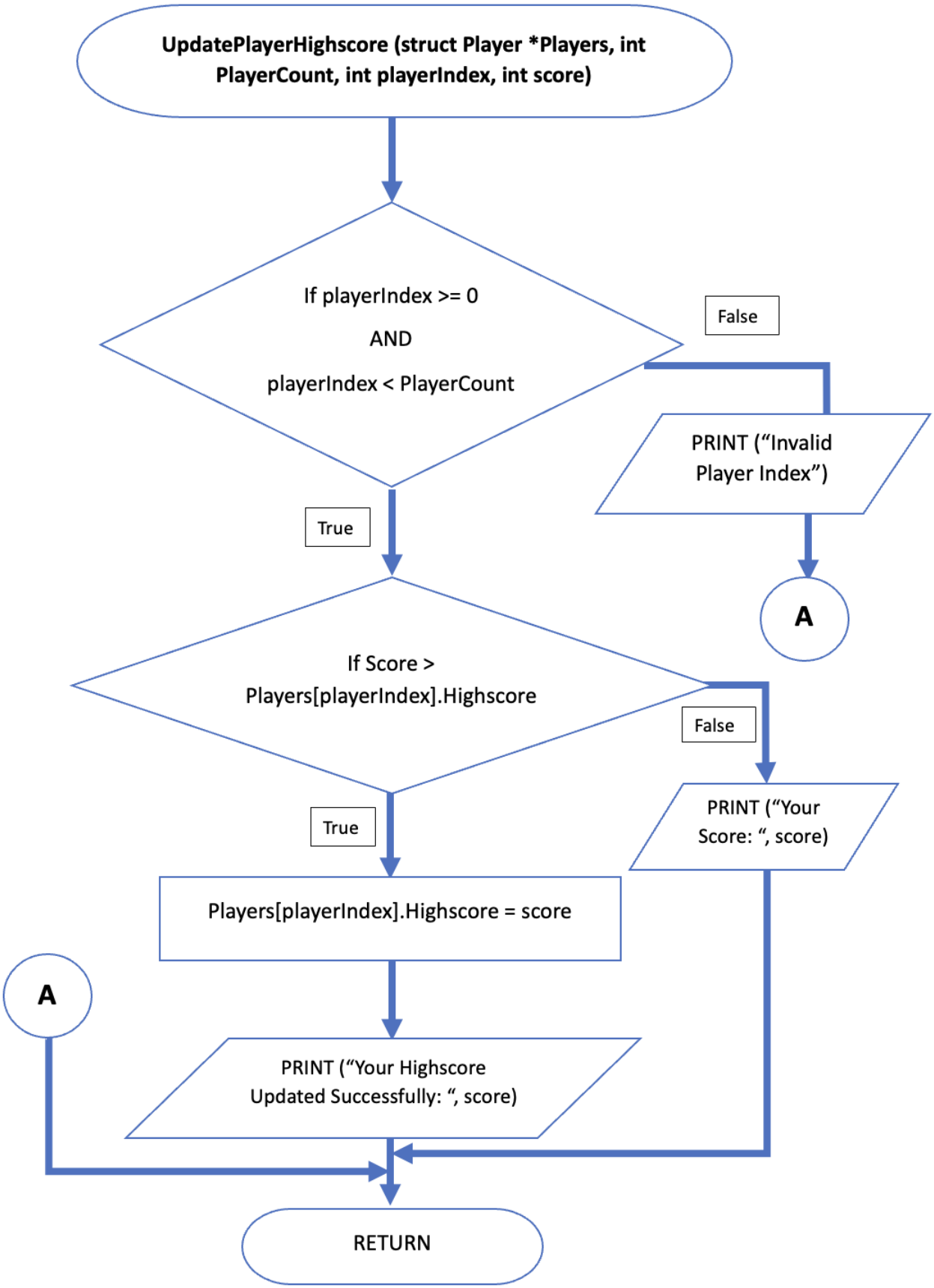
Login Function



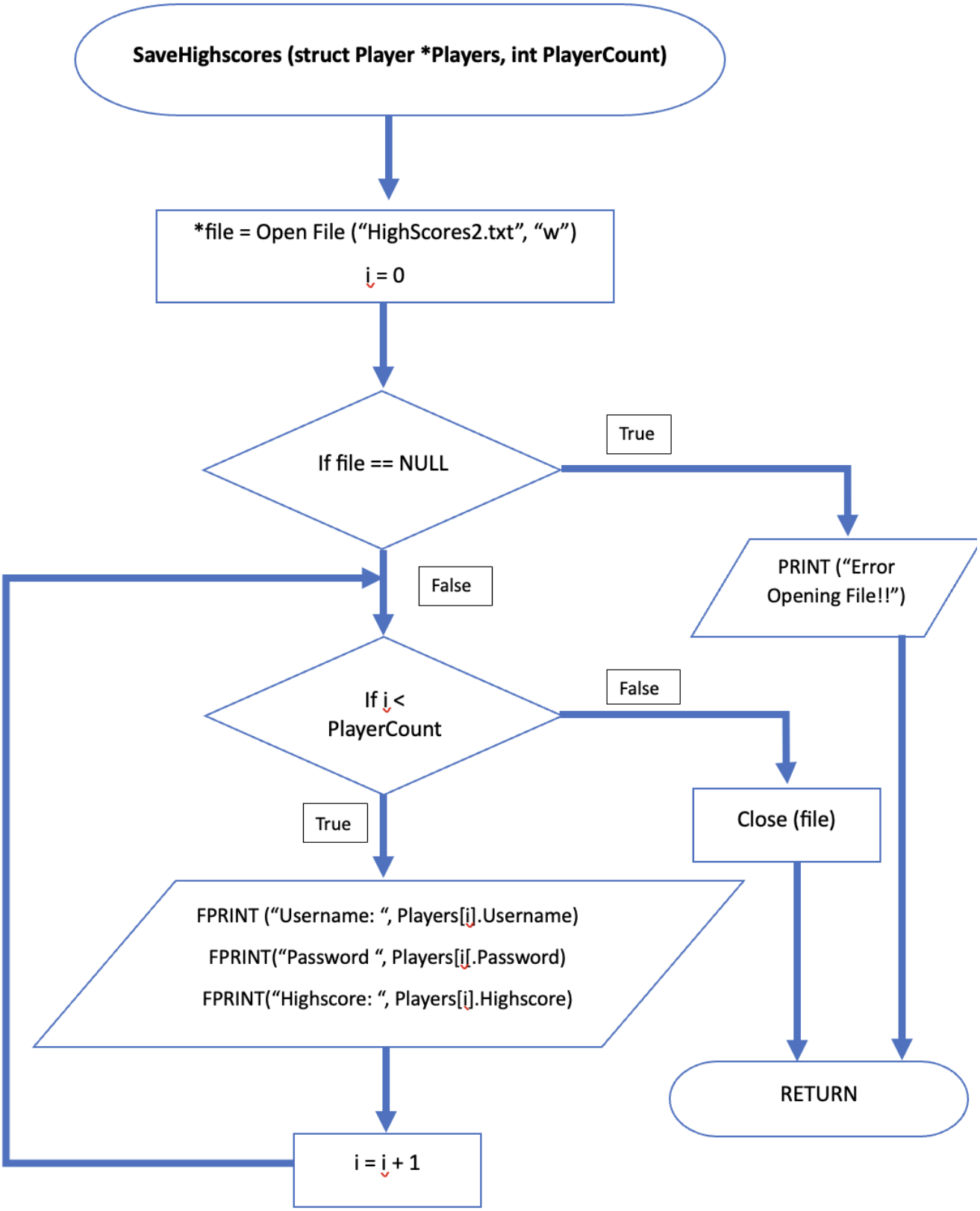
LoadHighscores Function



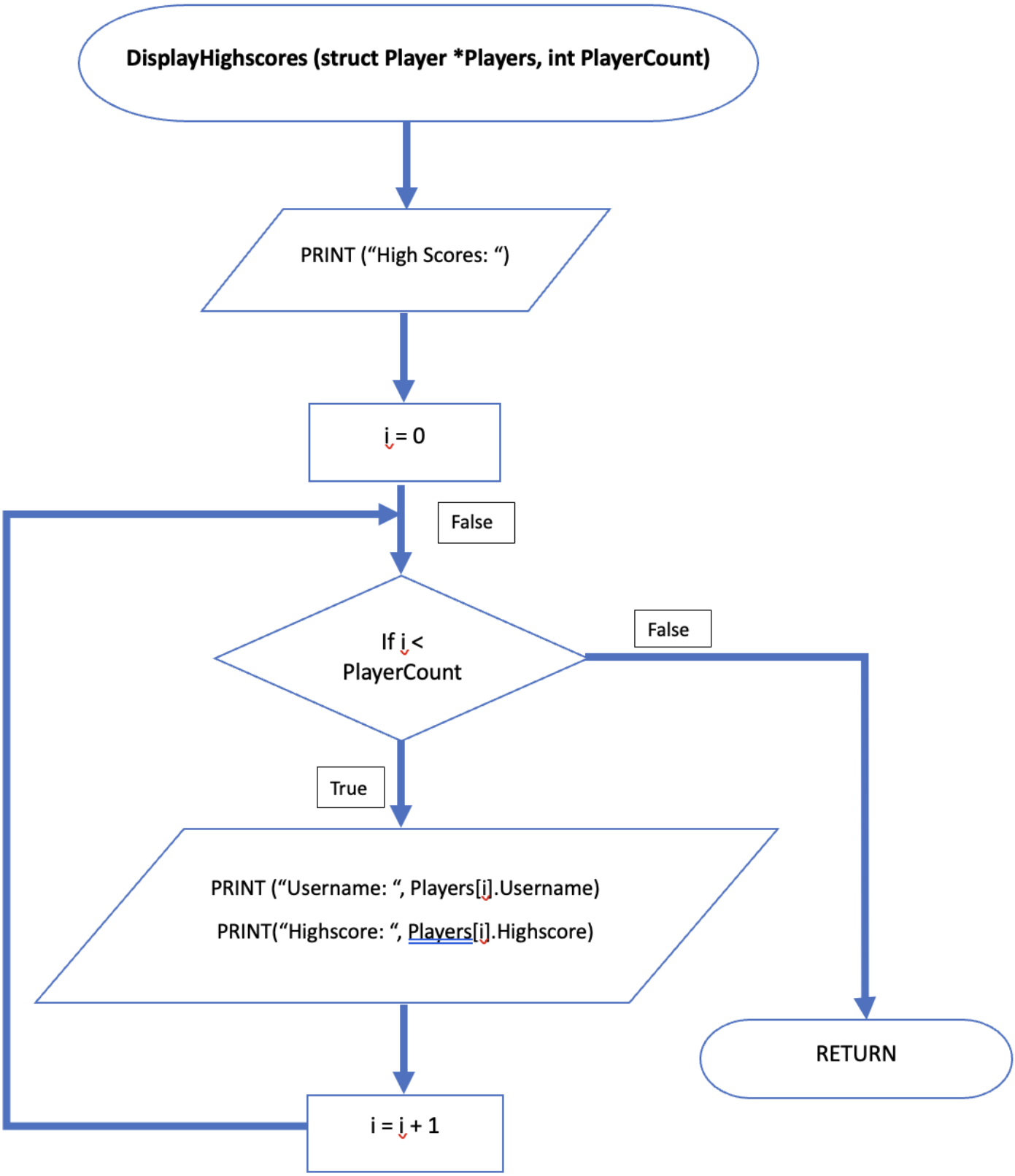
UpdatePlayerHIghscore



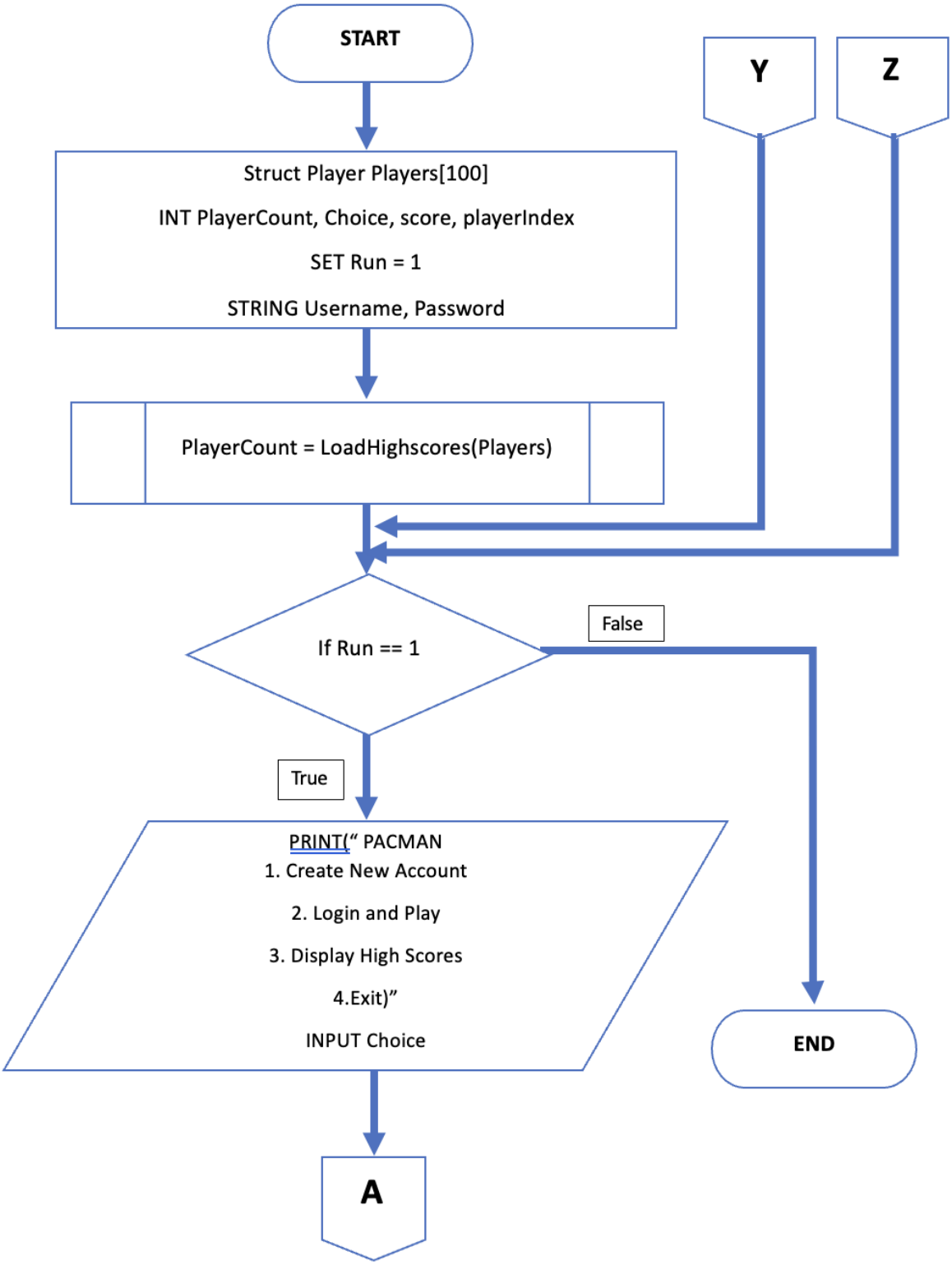
SaveHighscores Function

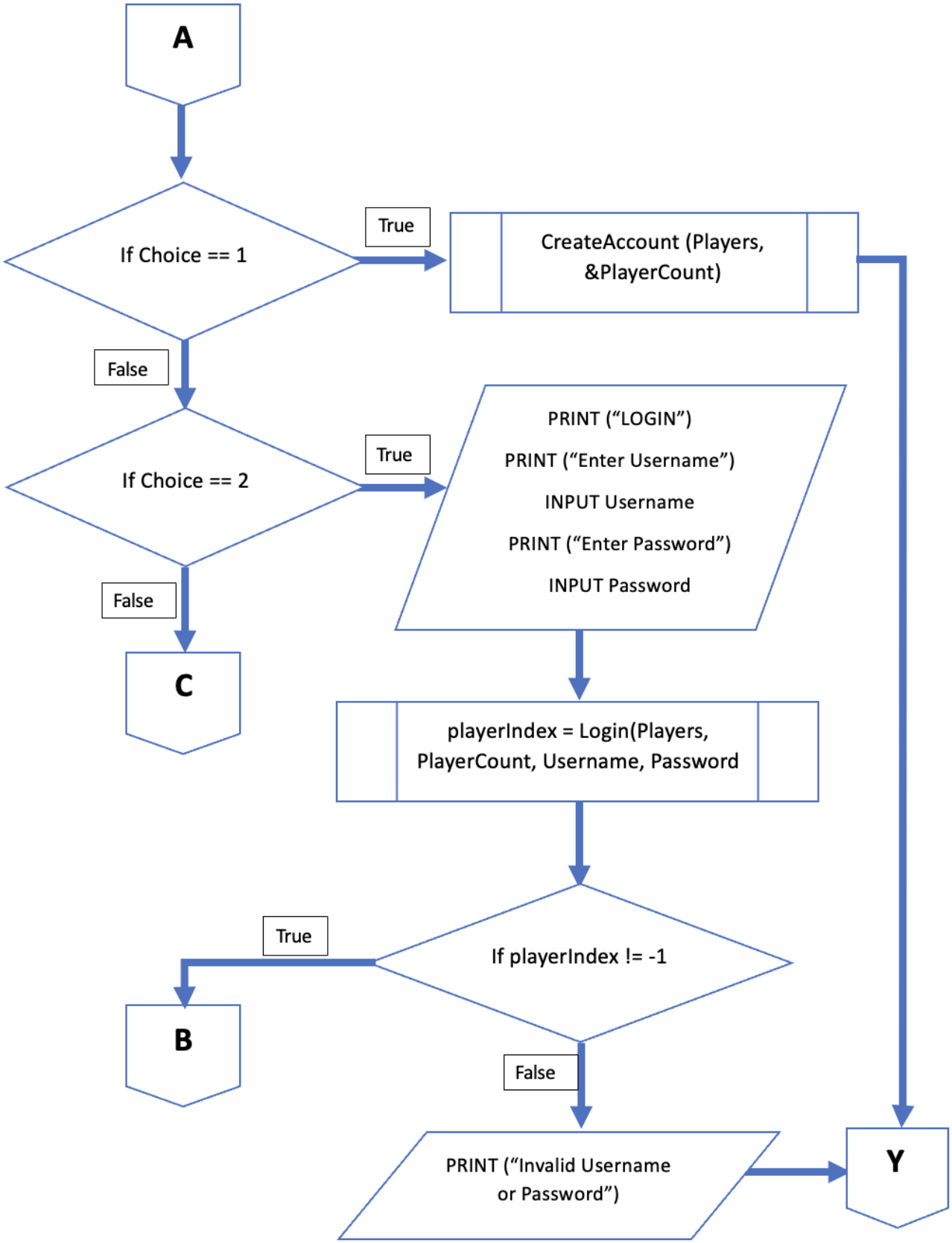


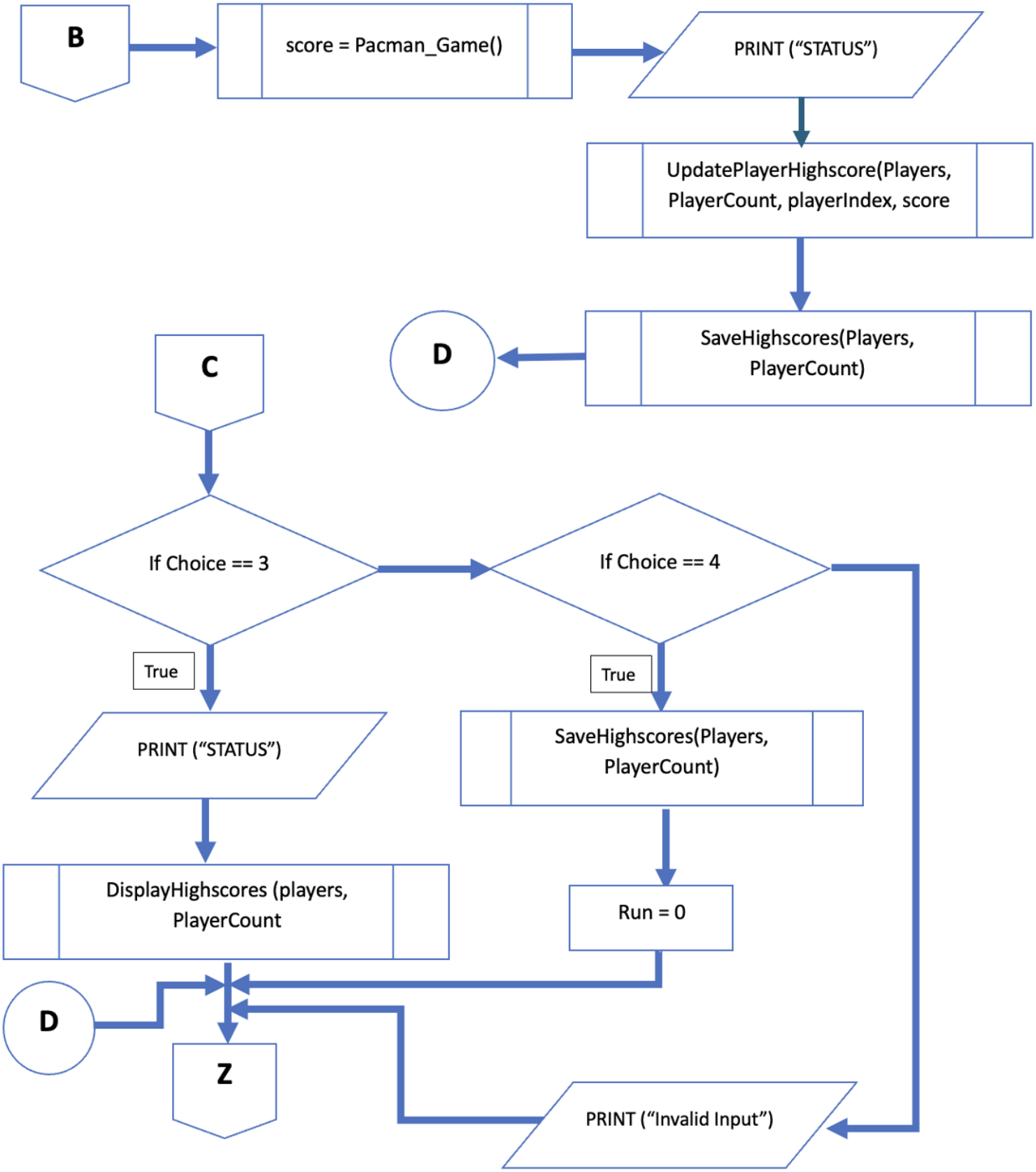
DisplayHighscores Function



Main Function



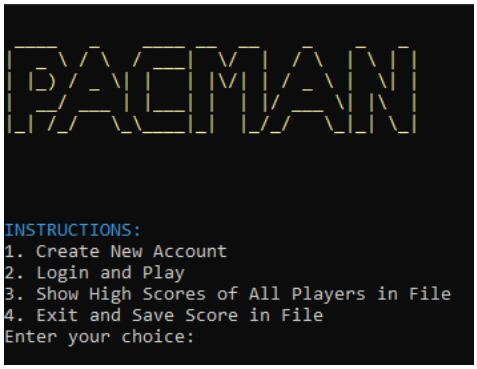

A
If Choice == 1
True
CreateAccount (Players, &PlayerCount)
False
If Choice == 2
True
PRINT ("LOGIN")
PRINT ("Enter Username")
INPUT Username
PRINT ("Enter Password")
INPUT Password
False
C
playerIndex = Login(Players, PlayerCount, Username, Password
If playerIndex != -1
True
B
False
PRINT ("Invalid Username or Password")
Y


B
score = Pacman_Game()
PRINT ("STATUS")
UpdatePlayerHighscore(Players, PlayerCount, playerIndex, score
SaveHighscores(Players, PlayerCount)
D
C
If Choice == 3
If Choice == 4
True
True
PRINT ("STATUS")
SaveHighscores(Players, PlayerCount)
DisplayHighscores (players, PlayerCount
Run = 0
D
Z
PRINT ("Invalid Input")

OUTPUT

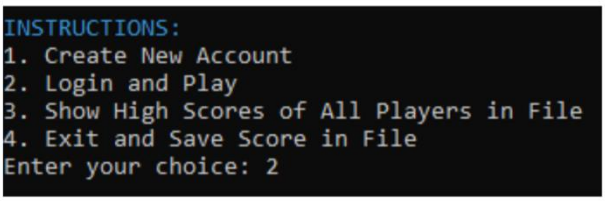
User is shown 4 options

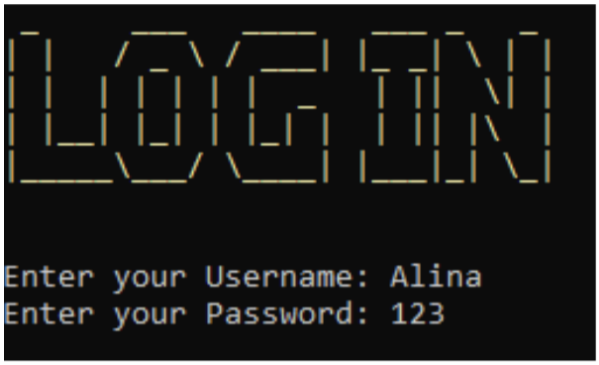
1- Create Account
2- Login and Play
3- Display Highscore
4- Exit



User Choose option 1:

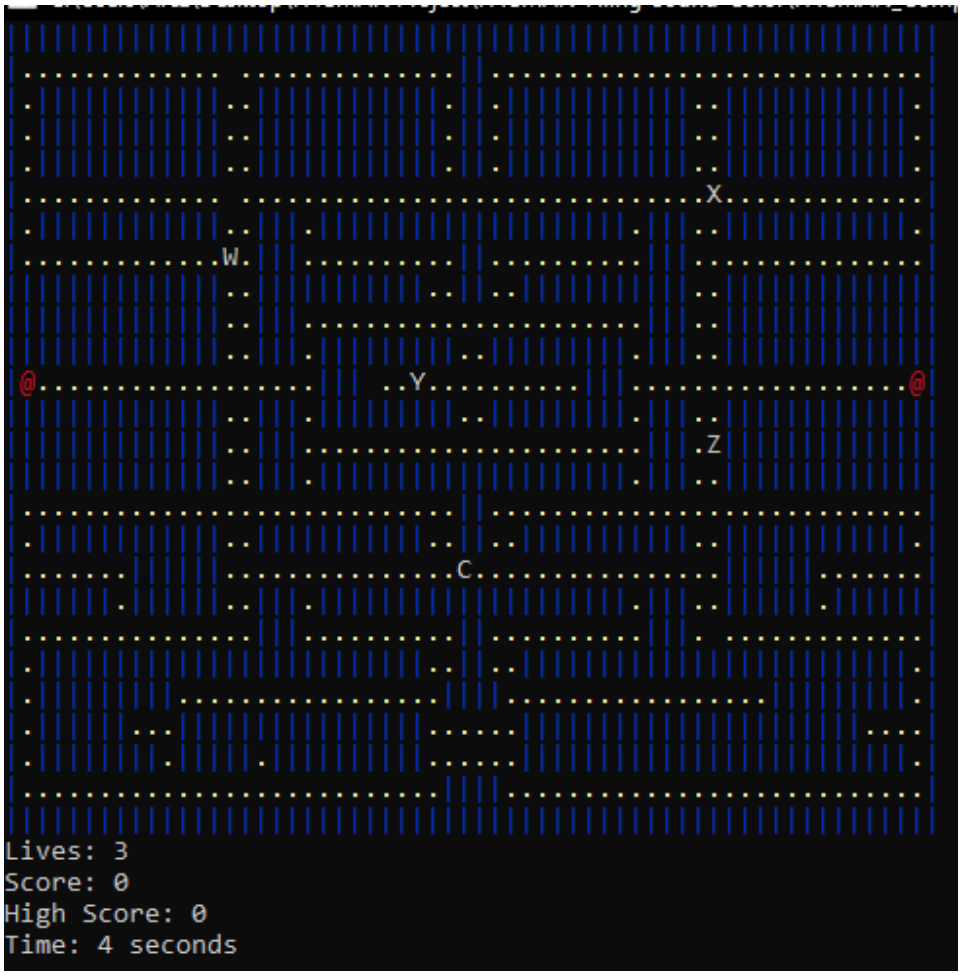
User Choose Option 2:



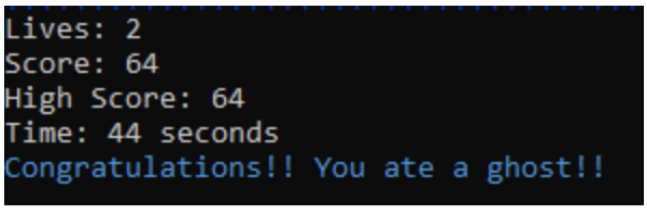


Game Starts:

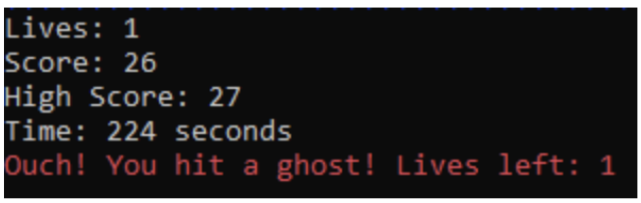
Maze:



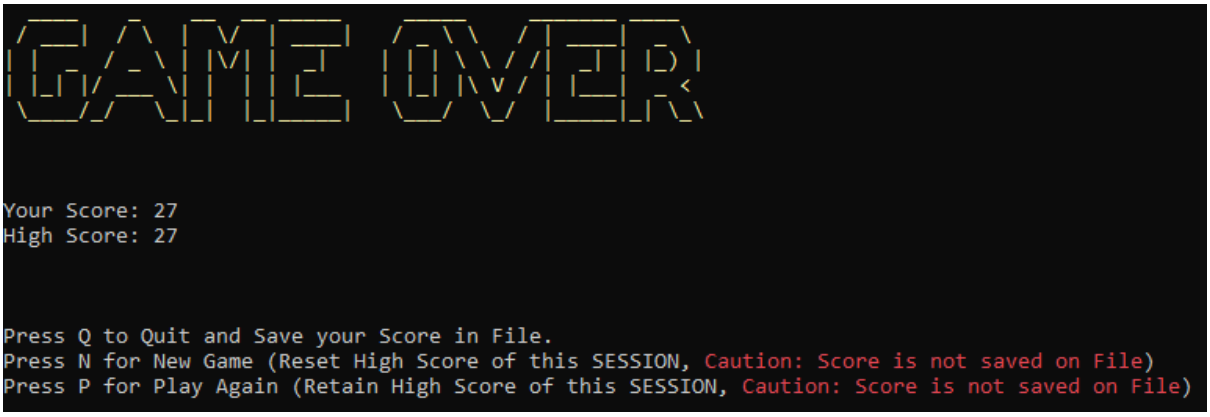
PACMAN ate Powercoin and than ate a ghost



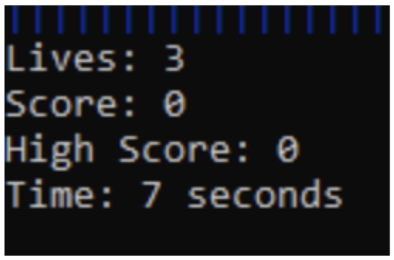
PACMAN got eaten by Ghost, lost lives



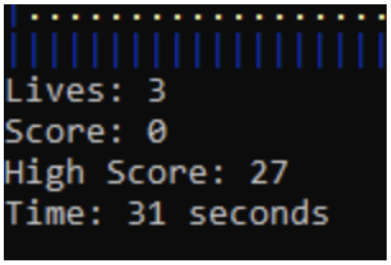
GAME OVER screen:



User Choose N



User Choose P



User chose Q:

User Choose Option 3:

User Choose Option 4:

Game Exit….

REFERENCES

W3 School:

www.w3schools.com

Geek for Geek

https://www.geeksforgeeks.org/pacman-game-in-c/

Stack Overflow

https://stackoverflow.com/questions/30126490/how-to-hide-console-cursor-in-c