



PROJECT REPORT CCP

CODEBREAKER
DISCIPLINE : CSIT SECTION "B"
TEACHER: SIR ABDULLAH

PRESENTED BY:
BINARY BREAKERS (CT-175)

ZUHA AZHAR
M. ASHAR HUSSAIN

CT-25055
CT-25087

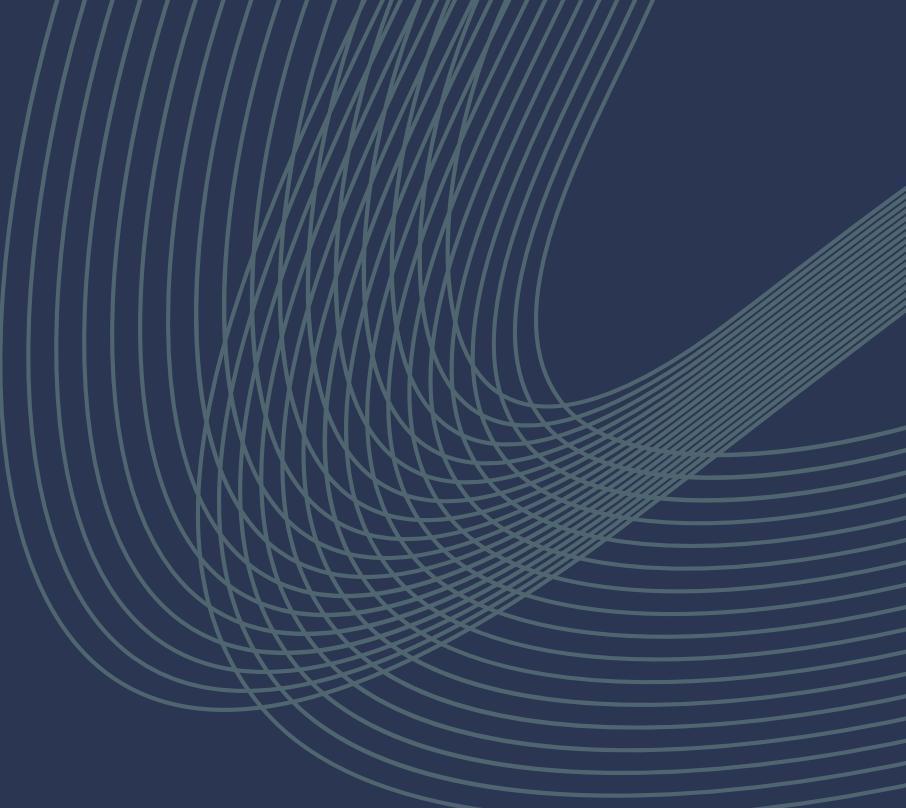


TABLE OF CONTENTS

Introduction	3
Key Features	4
Game Overview & Objectives	5
Code	6-10
Implementation Breakdown & Code Explaination	11-14
Flowchart & Control Flow	15-17
Challenges & Solutions	18
Architecture Diagram	19
Examples	20
Conclusion	21

INTRODUCTION TO CODEBREAKER

Week # 6 - Week # 12



CODEBREAKER is a logical and engaging Number Guessing Game built in C language. It allows users to test their intuition, deduction, and quick thinking by guessing a randomly generated secret number within limited attempts. The game combines simplicity with strategy, offering hints after each guess to guide the player. Its design emphasizes real-time interaction, modular coding, and the practical use of loops, conditionals, and random number generation.



CODEBREAKER

A MIND BENDING GAME

Binary Breakers

Zuha Azhar & M. Ashar Hussain
CT-25055 CT-25087

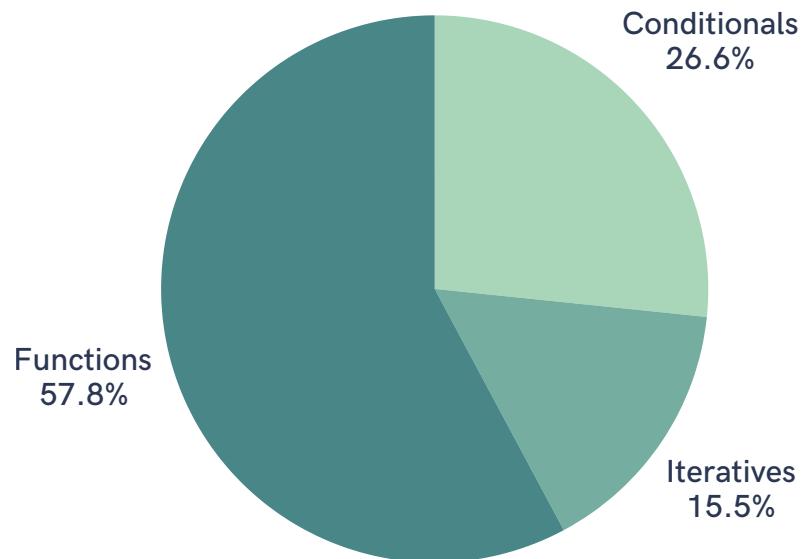
Key Features



The primary objective was to create a fully functional console-based game using core programming fundamentals.



The system supports both Single Player and VS. mode, ensuring a competitive yet educational environment for users



GAME OVERVIEW & OBJECTIVES

The game begins with a welcome screen and prompts the user to choose between two modes — Single Player and VS mode.

SINGLE PLAYER MODE

In **Single Player Mode**, the player selects a difficulty level (Easy, Medium, or Hard). Each level increases the number of digits in the secret code (3, 4, or 5 digits respectively). The player has five attempts to guess the secret code correctly. The score depletes as the number of attempts increase.

VS. MODE

In *VS. Mode*, two players compete head-to-head. Each player receives their own randomly generated 4-digit number. Players take turns guessing their secret codes, and their completion times are recorded. The player who guesses correctly in the least time wins

After each attempt, feedback is given using symbols:

- # indicates the digit and position are correct.
- ~ means the digit exists but is in the wrong position.
- X signifies the digit does not exist in the secret code.



```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main(){
    int mode;
    printf("\t===== WELCOME TO CODEBREAKER =====\n");

    printf("Choose Mode:\n1.Single Player Mode\n2.Multiplayer Mode\n");
    scanf("%d",&mode);

    switch(mode){
        case 1: {
            srand(time(0));
            int i, j, k=0, z=0, attempt, score=115, level, lower=0, upper=9;
            time_t start, end;

            printf("\t=====WELCOME TO SINGLE PLAYER MODE=====");

            printf("\n\nGuess the secret number within 5 attempts!\nChoose a mode: Easy (3 digits),
Normal (4 digits), or Hard (5 digits).\nAfter each guess, you will see hints\n\n
Don't forget to add space in-between digits\n\n # for correct digit & position\n
~ for correct digit wrong position\n X for wrong digit.\n");

            printf("\nChoose difficulty Level:\n1. Easy\n2. Medium\n3. Hard\n");
            scanf("%d", &level);

            if(level != 1 || level !=2 || level != 3){
                printf("\nInvalid Option.");
                return 0;
            }

            int randomNum1[3];
            for(i=0;i<3;i++){
                randomNum1[i]=(rand()%(upper-lower+1))+lower;
            }
            int randomNum2[4];
            for(i=0;i<4;i++){
                randomNum2[i]=(rand()%(upper-lower+1))+lower;
            }
            int randomNum3[5];
            for(i=0;i<5;i++){
                randomNum3[i]=(rand()%(upper-lower+1))+lower;
            }

            time(&start);

            for(attempt=1;attempt<=5;attempt++){
                printf("\nAttempt %d: ", attempt);

                switch (level)
                {
                case 1:{

                    int num[3];
                    scanf("%d%d%d",&num[0],&num[1],&num[2]);
                    z=0;
                    for(i=0;i<3;i++){
                        if(num[i]==randomNum1[i]){
                            printf( " # ");
                            z++;
                            if(z==3)
                                break;
                            continue;
                        }
                    }
                }
                }
            }
        }
    }
}
```

```

for(j=0;j<3;j++){
    if(num[i]==randomNum1[j]){
        k=1;
        break;
    }
}
if(k!=0)
    printf(" ~ ");
else
    printf(" X ");
k=0;
}
score-=15;
if(z==3){
    break;
}
break;
}

```

case 2:{

```

int num[4];
scanf("%d%d%d%d",&num[0],&num[1],&num[2],&num[3]);
z=0;
for(i=0;i<4;i++){
    if(num[i]==randomNum2[i]){
        printf( " # ");
        z++;
        if(z==4)
            break;
        continue;
    }
    for(j=0;j<4;j++){
        if(num[i]==randomNum2[j]){
            k=1;
            break;
        }
    }
    if(k!=0)
        printf(" ~ ");
    else
        printf(" X ");
    k=0;
}
score-=15;
if(z==4){
    break;
}
break;
}

```

case 3:{

```

int num[5];
scanf("%d%d%d%d%d",&num[0],&num[1],&num[2],&num[3],&num[4]);
z=0;
for(i=0;i<5;i++){
    if(num[i]==randomNum3[i]){
        printf( " # ");
        z++;
        if(z==5)
            break;
        continue;
    }
}

```

```

for(j=0;j<5;j++){
    if(num[i]==randomNum3[j]){
        k=1;
        break;
    }
}
if(k!=0)
    printf(" ~ ");
else
    printf(" X ");
k=0;
}
score-=15;
if(z==5){
    break;
}
break;
}
}
if (level == 1 && z == 3) break;
else if (level == 2 && z == 4) break;
else if (level == 3 && z == 5) break;
}
time(&end);

if (level == 1 && z == 3) printf("\nYou Won!");
else if (level == 2 && z == 4) printf("\nYou Won!");
else if (level == 3 && z == 5) printf("\nYou Won!");
else {
    printf("\nYou Lost!");
    score = 0;
    switch(level){
        case 1:
            printf("\nThe number is ");
            for(int x=0;x<3;x++){
                printf("%d", randomNum1[x]);
            }
            break;
        case 2:
            printf("\nThe number is ");
            for(int x=0;x<4;x++){
                printf("%d", randomNum2[x]);
            }
            break;
        case 3:
            printf("\nThe number is ");
            for(int x=0;x<5;x++){
                printf("%d", randomNum3[x]);
            }
            break;
    }
}
double time_taken=difftime(end, start);

printf("\n Your score is %d\n Time taken = %.2fsec", score, time_taken);

break;
}
case 2: {
    srand(time(0));
    int i, j, k=0, z=0, level, lower=0, upper=9, attempt=1;
    char begin1[6],begin2[6];
    int num1[4], num2[4];
    time_t start1, start2, end1, end2;
}

```

```

printf("\t===== WELCOME TO TWO PLAYER MODE =====");
printf("\nBoth players take turns guessing a 4-digit secret number.\n"
     "Unlimited attempts , faster player wins!\n\n# = correct digit & position\n"
     "~ = correct digit, wrong position\nX = wrong digit\n");
int randomNum1[4];
for(i=0;i<4;i++){
    randomNum1[i]=(rand()%(upper-lower+1))+lower;
}
int randomNum2[4];
for(i=0;i<4;i++){
    randomNum2[i]=(rand()%(upper-lower+1))+lower;
}

printf("\nPlayer one's turn:");
printf("\nType 'start' to Begin: ");
scanf("%s",begin1);
time(&start1);

do{
    printf("\nAttempt %d: ", attempt);
    scanf("%d%d%d%d",&num1[0],&num1[1],&num1[2],&num1[3]);
    z=0;
    for(i=0;i<4;i++){
        if(num1[i]==randomNum1[i]){
            printf( "# ");
            z++;
            if(z==4)
                break;
            continue;
        }
        for(j=0;j<4;j++){
            if(num1[i]==randomNum1[j]){
                k=1;
                break;
            }
        }
        if(k!=0)
            printf(" ~ ");
        else
            printf(" X ");
        k=0;
    }
    attempt++;
}
while(z!=4);
time(&end1);
double time_taken1=difftime(end1, start1);

printf("\nTime taken by 1st player: %.2f seconds", time_taken1);
attempt=1;

printf("\nPlayer one's turn:");
printf("\nType 'start' to Begin: ");
scanf("%s",begin2);
time(&start2);

do{
    printf("\nAttempt %d: ", attempt);
    scanf("%d%d%d%d",&num2[0],&num2[1],&num2[2],&num2[3]);
    z=0;
}

```

```

for(i=0;i<4;i++){
    if(num2[i]==randomNum2[i]){
        printf(" # ");
        z++;
        if(z==4)
            break;
        continue;
    }
    for(j=0;j<4;j++){
        if(num2[i]==randomNum2[j]){
            k=1;
            break;
        }
    }
    if(k!=0)
        printf(" ~ ");
    else
        printf(" X ");
    k=0;
}
attempt++;
}
while(z!=4);
time(&end2);
double time_taken2=difftime(end2, start2);

printf("\nTime taken by 2nd player: %.2f seconds", time_taken2);

if(time_taken1<time_taken2)
printf("\n1ST PLAYER WON!");
else
printf("\n2ND PLAYER WON!");
break;
}

default:printf("Wrong option");
}
return 0;
}

```

---X---X---X---

IMPLEMENTATION BREAKDOWN

Single Player Mode

- The random number generator (rand()) & srand(time(0)) creates secret numbers with a digit count based on the chosen difficulty.
- The player has five attempts to guess the number.
- Each guess is compared digit-by-digit to the secret code.
- Feedback is displayed for every digit based on correctness.
- The score starts at 115 and decreases by 15 with every wrong attempt.
- Time is tracked using time_t start and end variables.

If the player correctly guesses all digits, the game ends with a congratulatory message and the total score is displayed.

```
Attempt 1: 1 2 3 4 5
X ~ X X X
Attempt 2: 2 6 7 8 9
# # X ~ #
Attempt 3: 2 6 8 0 9
# # # X #
Attempt 4: 2 6 8 2 9
# # # # #
You Won!
Your score is 55
Time taken = 46.00sec
```



CODE EXPLANATION

Single Player Mode

Mode Selection and System Initialization:

The program begins by displaying a mode selection menu, controlled through a `switch(mode)`, that directs program flow into Single Player or Multiplayer routines. It initializes core variables, sets up `time_t` timers, and prepares the execution context for gameplay through structured control logic.

Random Code Generation Algorithm

Using `srand(time(0))` as a random seed initializer, the program invokes `rand()` to generate the secret numeric array. The digit count dynamically adapts to the chosen difficulty level (3, 4, or 5 digits)

User Input Parsing:

Employs `scanf()` for multi-digit parsing, followed by indexed array comparison using nested for loops. Each element is checked for positional correctness, establishing a deterministic validation system that maps user input to logic outcomes.

Iterative Attempt Control and Scoring Mechanism

The core game loop `for(attempt = 1; attempt <= 5; attempt++)` enforces attempt limits while managing score updates in real time. Each failed iteration triggers a `score -= 15` operation, and success activates a termination flag. The `diffTime(end, start)` function computes player performance, merging iteration control, timing, and evaluation metrics into a cohesive loop structure.

```
printf("Choose Mode:\n1.Single Player Mode\n2.Multiplayer Mode\n");
scanf("%d",&mode);
switch(mode){
case 1: {
    srand(time(0));
    int i, j, k=0, z=0, attempt, score=115, level, lower=0, upper=9, y, l, m;
    time_t start, end;
```

Mode Selection & Initialization

```
int randomNum2[4];
for(l=0;l<4;l++){
    randomNum2[l]=(rand()%(upper-lower+1))+lower;
}
```

Random Number Generation

```
if (level == 1 && z == 3) printf("\nYou Won!");
else if (level == 2 && z == 4) printf("\nYou Won!");
else if (level == 3 && z == 5) printf("\nYou Won!");
else {
    printf("\nYou Lost!");
    score = 0;
```

Output Validation

```
int num[3];
scanf("%d%d%d",&num[0],&num[1],&num[2]);
z=0;
for(i=0;i<3;i++){
    if(num[i]==randomNum1[i]){
        printf( " #");
        z++;
        if(z==3)
            break;
        continue;
    }
    for(j=0;j<3;j++){
        if(num[i]==randomNum1[j]){
            k=1;
            break;
        }
    }
    if(k!=0)
        printf(" ~ ");
    else
        printf(" X ");
    k=0;
}
score-=15;
if(z==3){
    break;
    break;
```

Attempt Loop



IMPLEMENTATION BREAKDOWN

VS. Mode

The VS. mode introduces competition by timing both players separately.

- Player 1 and Player 2 each have unique secret numbers.
- Both begin by typing 'start' to initiate their turn.
- The program tracks their time using time(&start) and time(&end).
- Each player continues guessing until they correctly identify all four digits.
- The player with the lesser completion time wins.

This section highlights the use of loops, time functions, and string handling to ensure accurate timing and fair play

Player one's turn:
Type 'start' to Begin: start

```
Attempt 1: 1 2 3 4
X X ~ ~
Attempt 2: 1 2 4 3
X X ~ #
Attempt 3: 4 5 6 3
~ ~ X #
Attempt 4: 5 4 7 3
~ # X #
Attempt 5: 8 4 5 3
# # # #
Time taken by 1st player: 57.00 seconds
```

Player two's turn:
Type 'start' to Begin: start

```
Attempt 1: 1 2 3 4
X X X #
Attempt 2: 5 6 7 4
# X X #
Attempt 3: 5 8 9 4
# X X #
Attempt 4: 5 0 5 4
# X ~ #
Attempt 5: 5 5 4 4
# # # #
Time taken by 2nd player: 38.00 seconds
2ND PLAYER WON!
```



CODE EXPLANATION

VS. Mode

Dual-Instance Initialization and Mode Activation

Upon selecting VS. Mode, the program initializes two independent player instances, each with unique secret arrays generated via `rand()`.

Timers, variables, and input buffers are declared separately to maintain state isolation and ensure fair concurrent gameplay simulation.

Random Code Generation Algorithm

Using `srand(time(0))`, the program generates unique 4-digit codes for both players. This ensures equitable randomness and prevents code duplication.

Turn-Based Execution and Timing Logic

Each player types "`start`" to trigger `time(&startX)`, beginning their timer.

A do-while loop manages guesses until all digits match, ensuring accurate time tracking.

Comparative Evaluation and Winner Declaration

After both iterations conclude, the program calculates `difftime(end1, start1)` and `difftime(end2, start2)` to determine individual performance.

A simple conditional check identifies the player with the shorter completion time and declares the winner, ensuring an objective and time-based victory condition.

```
printf("\t===== WELCOME TO TWO PLAYER MODE =====");  
  
int randomNum1[4];  
    for(i=0;i<4;i++){  
        randomNum1[i]=(rand()%(upper-lower+1))+lower;  
    }  
int randomNum2[4];  
    for(i=0;i<4;i++){  
        randomNum2[i]=(rand()%(upper-lower+1))+lower;  
    }
```

Random Number Generation

```
time(&end2);  
double time_taken2=difftime(end2, start2);  
  
printf("\nTime taken by 2nd player: %.2f seconds", time_taken2);  
  
if(time_taken1<time_taken2)  
printf("\n1ST PLAYER WON!");  
else  
printf("\n2ND PLAYER WON!");  
break;  
}
```

Comparative Evaluation

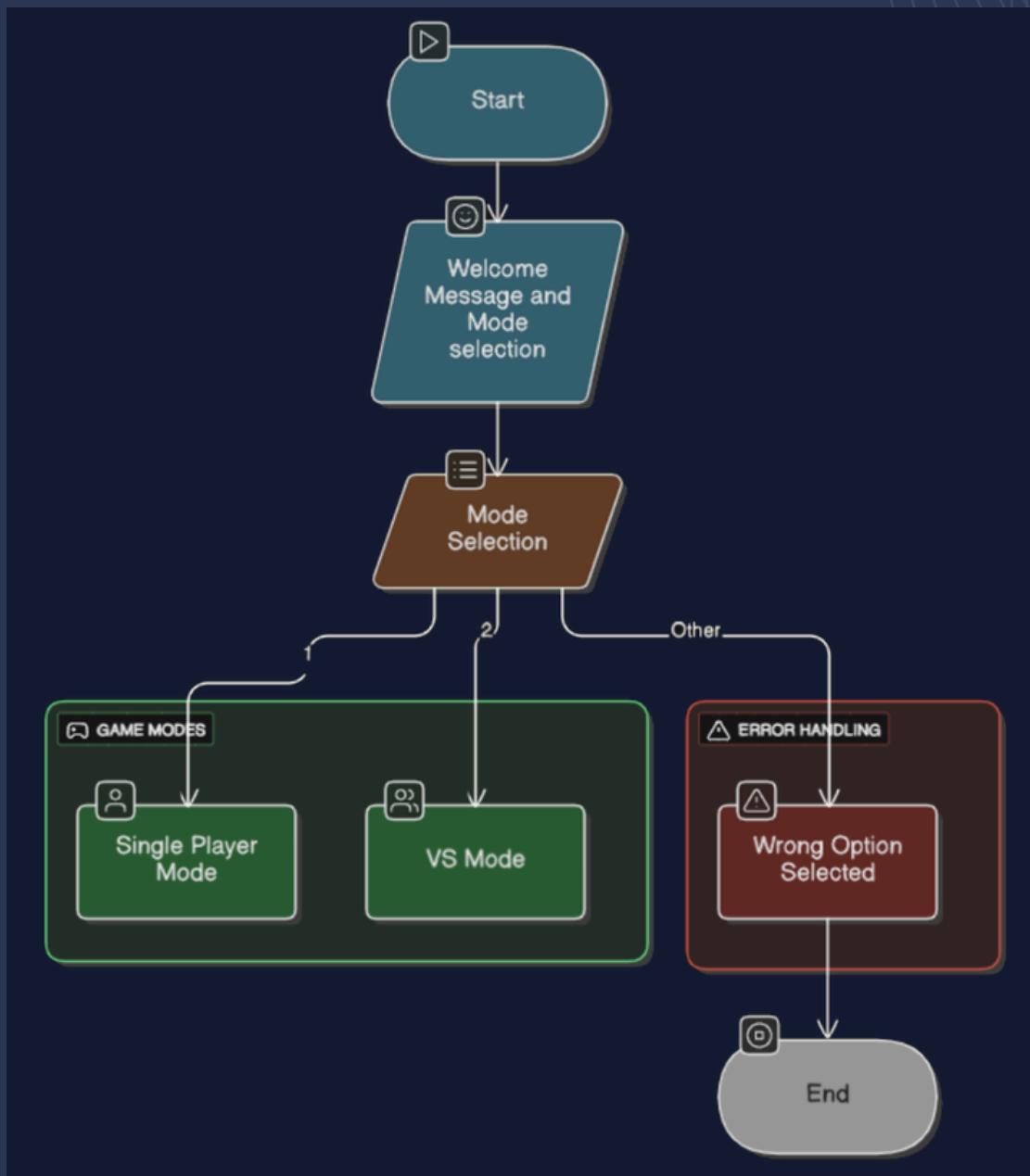
```
do{  
    printf("\nAttempt %d: ", attempt);  
    scanf("%d%d%d%d", &num1[0], &num1[1], &num1[2], &num1[3]);  
    z=0;  
    for(i=0;i<4;i++){  
        if(num1[i]==randomNum1[i]){  
            printf(" # ");  
            z++;  
            if(z==4)  
                break;  
            continue;  
        }  
        for(j=0;j<4;j++){  
            if(num1[i]==randomNum1[j]){  
                k=1;  
                break;  
            }  
        }  
        if(k!=0)  
            printf(" ~ ");  
        else  
            printf(" X ");  
        k=0;  
    }  
    attempt++;  
}
```

Attempt Loop



FLOWCHART AND CONTROL FLOW

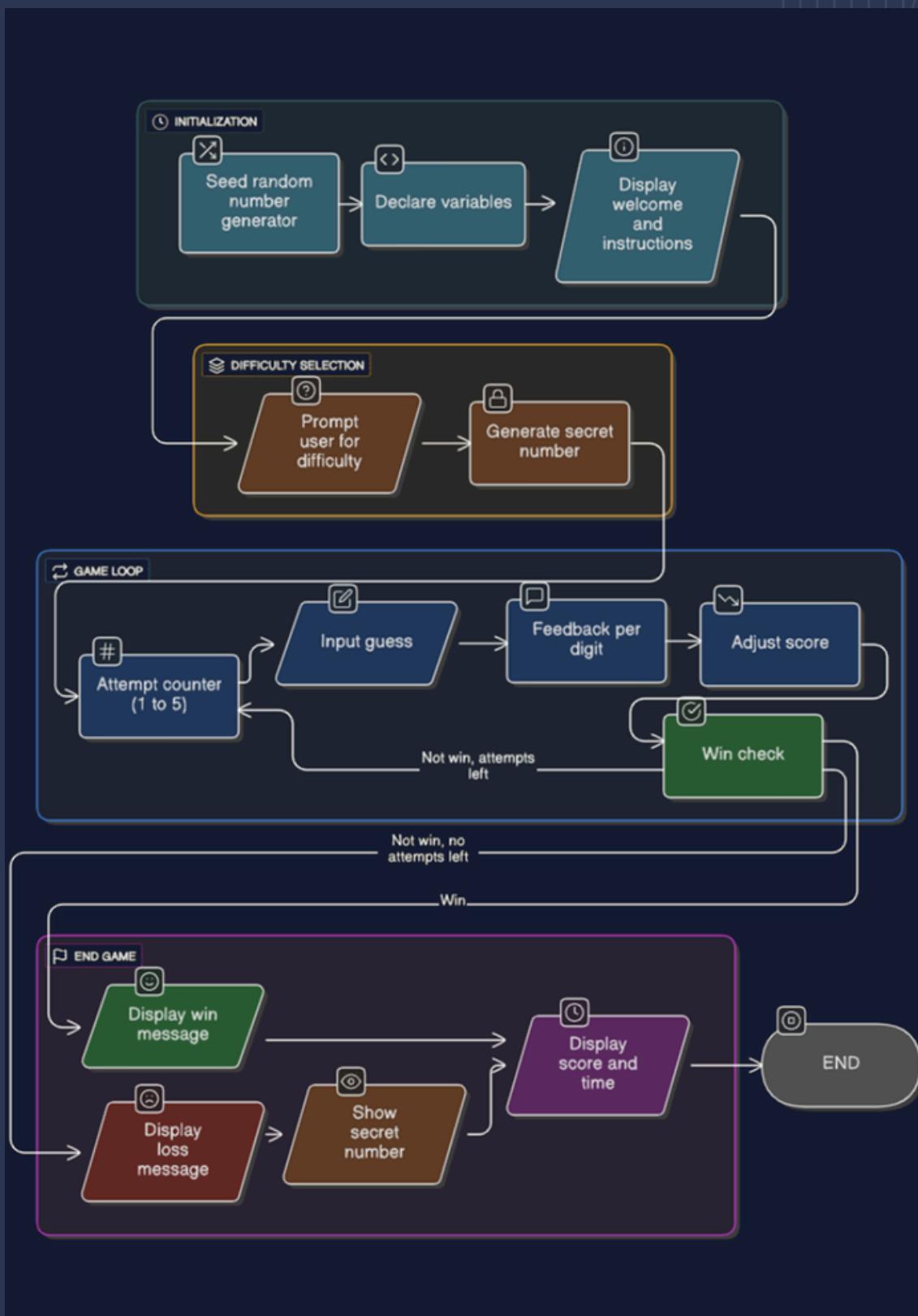
Represents the game's overall logic and execution flow.



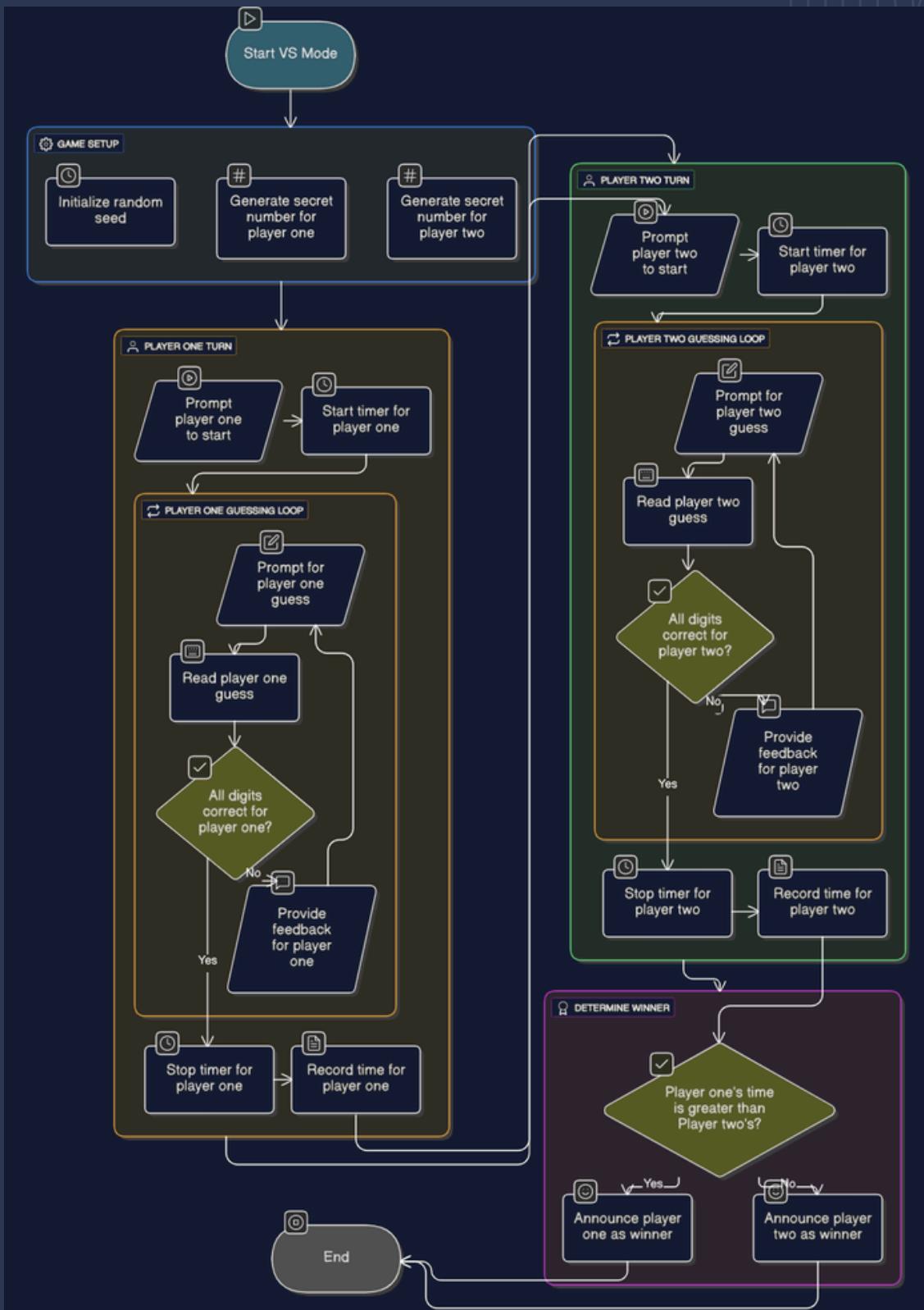
THE FLOWCHART OUTLINES THE COMPLETE LOGIC AND PROGRESSION OF THE CODEBREAKER GAME, STARTING FROM THE WELCOME SCREEN, THEN BRANCHING INTO DISTINCT SINGLE PLAYER AND MULTIPLAYER PATHS.



SINGLE PLAYER MODE FLOWCHART



VS. MODE FLOWCHART

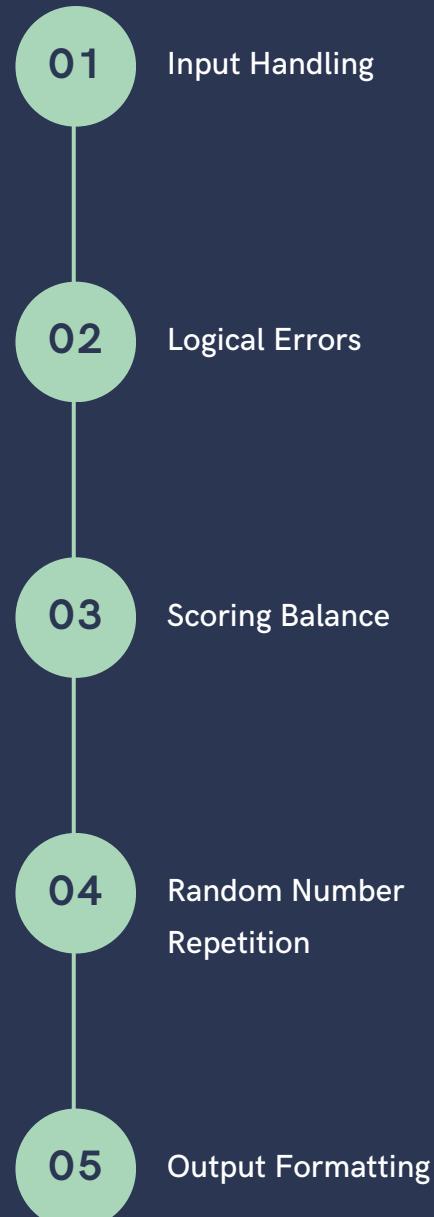


CHALLENGES & SOLUTIONS

Encountered During Development of CCP

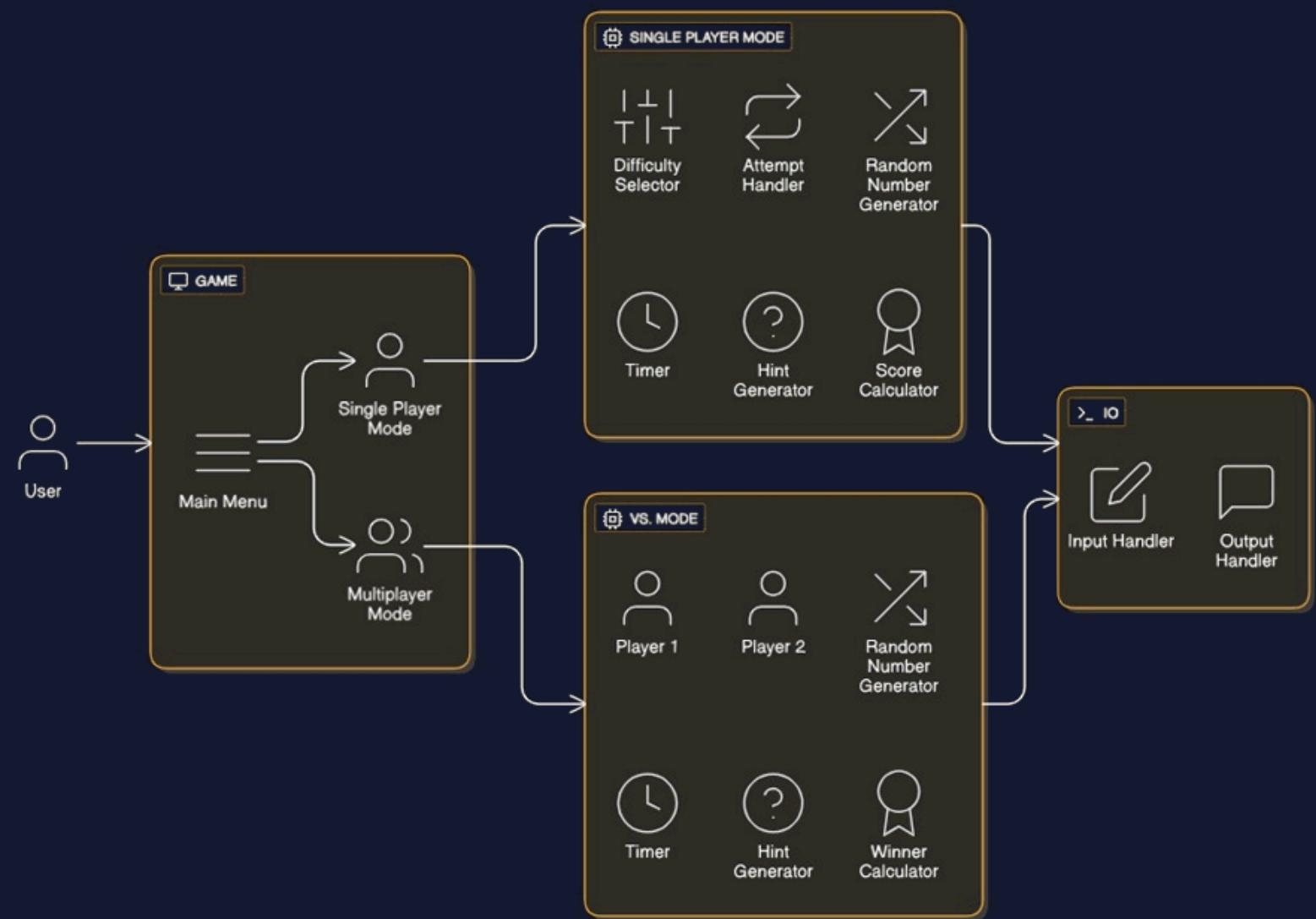
During the development of CODEBREAKER, several technical challenges were addressed to improve performance and user experience. Input overlap from multiple `scanf()` calls was fixed by properly separating integer and string inputs. Random number repetition was resolved by seeding `rand()` once per mode for better randomness. Timing issues in multiplayer mode were corrected so the timer started only after the player's input, ensuring accuracy and fairness.

The scoring system was standardized across all difficulty levels to maintain balanced gameplay, and output formatting was refined with better spacing and symbol separation for clearer feedback. These improvements made the game smoother, more reliable, and enjoyable to play.



ARCHITECTURE DIAGRAM

Program Structure Overview



SINGLE PLAYER MODE

```
===== WELCOME TO CODEBREAKER =====
Choose Mode:
1.Single Player Mode
2.VS. Mode
1
=====WELCOME TO SINGLE PLAYER MODE=====

Guess the secret number within 5 attempts!
Choose a mode: Easy (3 digits), Normal (4 digits), or Hard (5 digits).
After each guess, you will see hints

Don't forget to add space in-between digits

# for correct digit & position
~ for correct digit wrong position
X for wrong digit.

Choose difficulty Level:
1. Easy
2. Medium
3. Hard
```

VS. MODE

```
===== WELCOME TO CODEBREAKER =====
Choose Mode:
1.Single Player Mode
2.VS. Mode
2
===== WELCOME TO VS. MODE =====
Both players take turns guessing a 4-digit secret number.
Unlimited attempts , faster player wins!

# = correct digit & position
~ = correct digit, wrong position
X = wrong digit

Player one's turn:
Type 'start' to Begin: |
```



CONCLUSION

The CODEBREAKER project demonstrates how fundamental programming principles can be used to create an entertaining and educational application. The modular structure, dynamic gameplay, and logical design reinforce the importance of well-organized code. Future improvements may include adding a GUI, sound effects, and adaptive difficulty to make the game even more engaging.

Overall, the project achieved its goals by combining core programming concepts with teamwork, creativity, and strong problem-solving skills.

