

A project

ANNEX PAY



CSE 100 : Software Development Project I

SUBMITTED BY

Name	ID	Intake
Md. Monirul Hasan	20255103087	55
Ahana Islam	20255103118	55
Hafsa Akter	20255103119	55
Bithi Rani Debi	20255103086	55
Arpita Deb Aishee	20255103083	55

SUPERVISED BY:

Ashfia Jannat Keya
Email: ashfiaj@bubt.edu.bd
Assistant Professor,
Department of CSE

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY
(BUBT)**

15 December, 2025

Abstract

ANNEX PAY is an advanced console-based banking management system developed entirely in C programming language that uniquely integrates traditional banking operations with an entertainment module. This comprehensive system simulates real-world banking workflows including secure account management, PIN-based authentication, financial transactions (deposits, withdrawals, transfers), bill payments, mobile recharges, loan processing, and a novel feature – an integrated Rock, Paper, Scissors game for user engagement during banking sessions. The system implements persistent data storage through binary files for structured data and text files for transaction logs. It features a dedicated admin panel for system management, cross-platform compatibility (Windows/Linux), and a user-friendly color-coded interface. The integrated entertainment module provides users with a refreshing break option while maintaining banking session continuity. ANNEX PAY demonstrates advanced C programming concepts including modular design, file I/O, secure input handling, and software integration, effectively addressing both banking management needs and user experience enhancement through entertainment integration.

List of Figures

	Title	Page
1	System Architecture Diagram	11
2	Use Case Diagram	13
3	Context Level Diagram (DFD-0)	15
4	Level-1 Data Flow Diagram	16
5	Database Schema	17
6	Login Sequence Diagram	19
7	Main Menu Interface	24
8	User Dashboard	25
9	Admin Panel	25
10	Account Creation Screen	26
11	Secure PIN Entry	27
12	User Login Interface	27
13	Deposit Operation	28

14	Withdrawal Operation	28
15	Fund Transfer Screen	28
16	Bill Payment Menu	29
17	Mobile Recharge Interface	30
18	Fund Request Screen	30
19	Loan Application	31
20	Mini Statement Display	31
21	Transaction History	32
22	Search Transactions	32
23	Security Questions Setup	33
24	Checkbook Request	33
25	Messaging System	34
26	Integrated Game Interface	34
27	Referral Program	35
28	Admin Login	36
29	Admin Account Management	36
30	Loan Approval Process	36

Table of Contents

Abstract.....	1
List of Figures.....	2
Chapter 1: Introduction.....	6
Chapter 2: Background.....	8
Chapter 3: System Analysis & Design.....	10
Chapter 4: Implementation.....	20
Chapter 5: User Manual.....	24
Chapter 6: Conclusion.....	39
References	42
Appendix A: Source Code	43

Chapter 1: Introduction

1.1 Problem Specification/Statement

In the digital era, while large financial institutions have sophisticated online banking systems, educational institutions and small organizations often lack integrated solutions that combine financial management with user engagement features. Traditional banking systems focus solely on financial transactions, overlooking the importance of user experience and engagement. ANNEX PAY addresses this gap by integrating a comprehensive banking system with an entertainment module, creating a unique platform that serves both practical banking needs and user engagement requirements.

Existing C-based banking projects typically offer basic financial functionalities but lack interactive features that enhance user experience during banking sessions. This limitation becomes apparent during extended banking activities where users might appreciate brief diversions. ANNEX PAY solves this by providing a sophisticated banking system with an integrated game, allowing users to take refreshing breaks without leaving their banking session.

1.2 Objectives

The primary objectives of the ANNEX PAY project are:

1. Dual-Function System Development: To design and implement a secure banking system with integrated entertainment features using C programming language.
2. User Experience Enhancement: To improve banking session engagement through strategic entertainment integration without compromising security or functionality.
3. Seamless Integration: To ensure smooth transition between banking operations and entertainment features while maintaining session continuity.
4. Data Persistence: To establish reliable file-based storage for both banking data and game statistics across multiple sessions.
5. Security Implementation: To maintain robust security measures while providing accessible entertainment options.
6. Cross-Platform Compatibility: To ensure consistent operation across Windows and Linux environments for both banking and gaming modules.

1.3 Scope

The ANNEX PAY project encompasses:

Included Functionalities:

- Complete banking operations (account management, transactions, services)
- Integrated Rock, Paper, Scissors game with score tracking
- Secure session management between banking and gaming modes
- Comprehensive administrative controls
- User engagement features (referral program, messaging system)
- Detailed reporting and search capabilities

Technical Specifications:

- Support for 1000+ user accounts
- Real-time game processing with visual animations
- Daily banking limits enforcement (20,000 BDT)
- Cross-platform implementation with consistent user experience
- Modular architecture for both banking and gaming components

Limitations:

- Console-based interface (no graphical UI)
- Single-user access mode
- File-based storage system

Deliverables:

- Complete C source code with integrated gaming module
- Comprehensive project documentation
- User manual covering both banking and gaming features
- Installation and configuration guide

1.4 Organization of Project Report

This report is organized to provide comprehensive understanding:

Chapter 2: Background examines existing systems and literature informing the integrated approach.

Chapter 3: System Analysis & Design details the architectural models and integration strategies.

Chapter 4: Implementation explains the development process of both banking and gaming modules.

Chapter 5: User Manual provides complete guidance for system operation including gaming features.

Chapter 6: Conclusion summarizes achievements, discusses limitations, and proposes enhancements.

Chapter 2: Background

2.1 Existing System Analysis

Current banking systems can be categorized into three tiers, each with specific characteristics:

Traditional Banking Systems:

Focus exclusively on financial operations with no consideration for user engagement. These systems, while functionally complete, often lead to

user fatigue during extended sessions and lack features that make banking more enjoyable.

Educational Banking Projects:

Basic C/C++ implementations demonstrating core concepts but lacking real-world features and user engagement elements. These typically serve as academic exercises rather than practical applications.

Entertainment Systems:

Standalone games or entertainment applications with no integration with practical systems. These lack the utility of banking systems while demonstrating engagement strategies.

ANNEX PAY Innovation:

By integrating banking operations with entertainment features, ANNEX PAY creates a unique hybrid system that addresses the limitations of existing solutions. This approach recognizes that modern users expect engaging experiences even in utility applications.

2.2 Supporting Literatures

Technical Integration:

Software engineering literature on modular design and system integration informed the approach to combining banking and gaming functionalities.

Research on user experience design provided insights into maintaining engagement during utility tasks.

Game Development Principles:

Basic game design principles from interactive computing literature guided the implementation of the Rock, Paper, Scissors game with appropriate feedback mechanisms, score tracking, and visual elements.

User Engagement Strategies:

Studies on user engagement in software applications influenced the decision to integrate entertainment as a value-added feature rather than a distraction from core functionality.

Methodological Approach:

The incremental development model supported the phased implementation of banking features followed by seamless integration of gaming elements. This approach ensured stable banking operations before adding entertainment features.

Chapter 3: System Analysis & Design

3.1 Technology & Tools

Programming Environment:

- Primary Language: ANSI C (C99 standard)
- Compilers: GCC for Linux, MinGW for Windows
- Development Tools: Code::Blocks, Visual Studio Code

Libraries:

- Standard C Libraries: <stdio.h>, <stdlib.h>, <string.h>, <time.h>, <ctype.h>
- Platform-Specific: <conio.h> (Windows), <termios.h> (Linux)

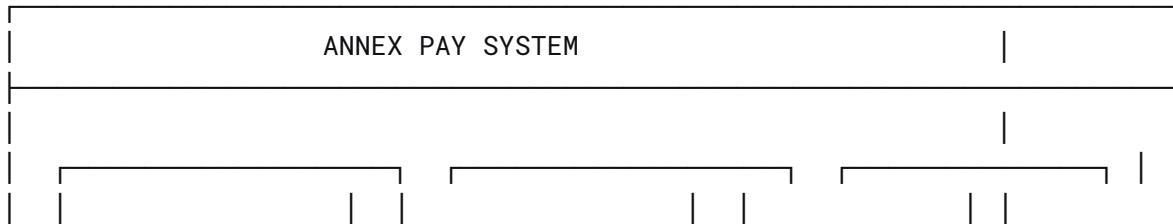
Storage Architecture:

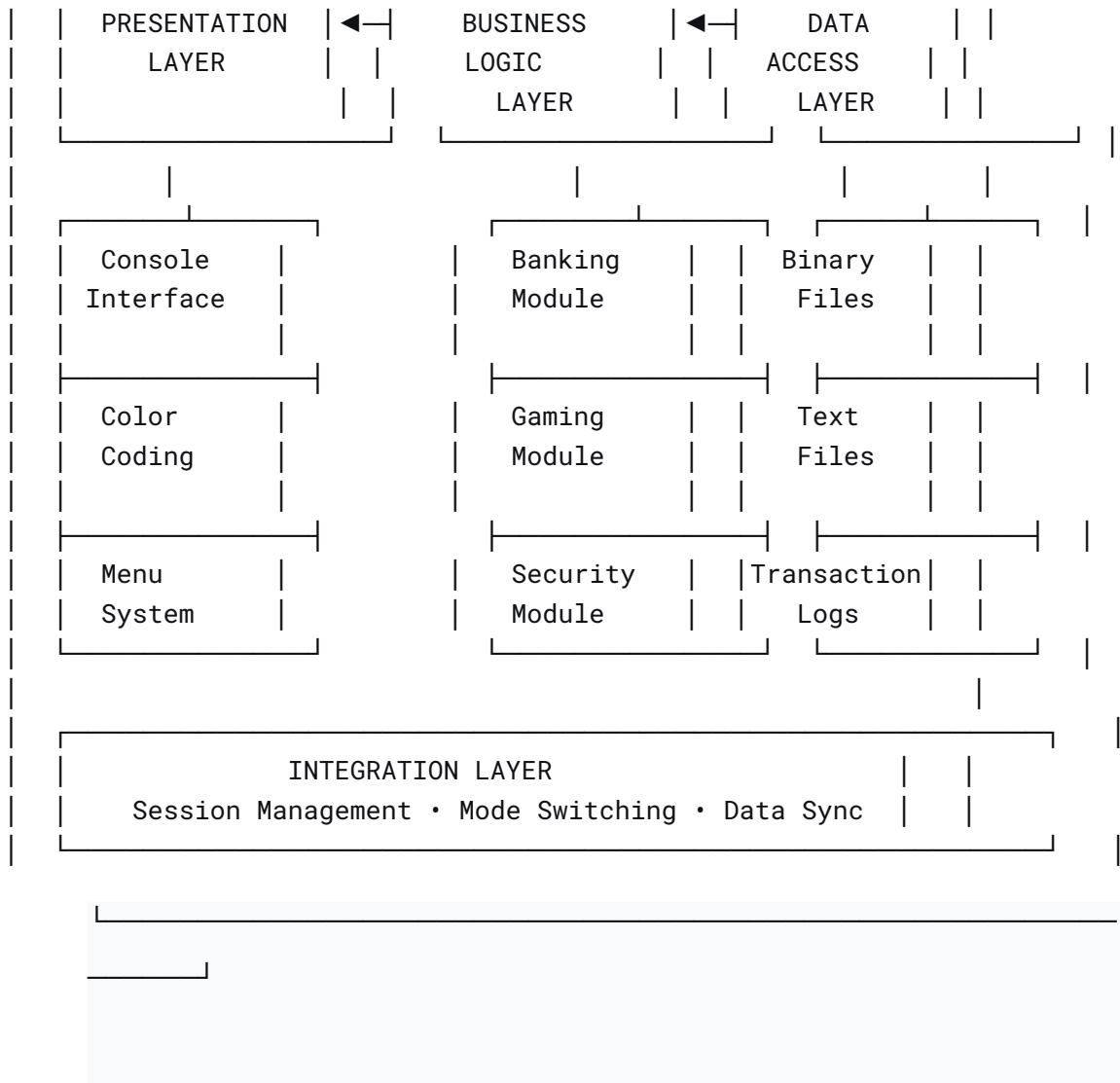
- Binary files for structured data
- Text files for transaction logs
- Hierarchical directory structure

3.2 System Architecture

Figure 1: System Architecture Diagram

text





The system employs a modular architecture with three primary layers:

Presentation Layer:

- Console interface with ANSI color coding
- Integrated menu system (banking + gaming)
- Consistent navigation across modules

Business Logic Layer:

- Banking operations module
- Gaming module with score management
- Session management for mode switching
- Security and validation systems

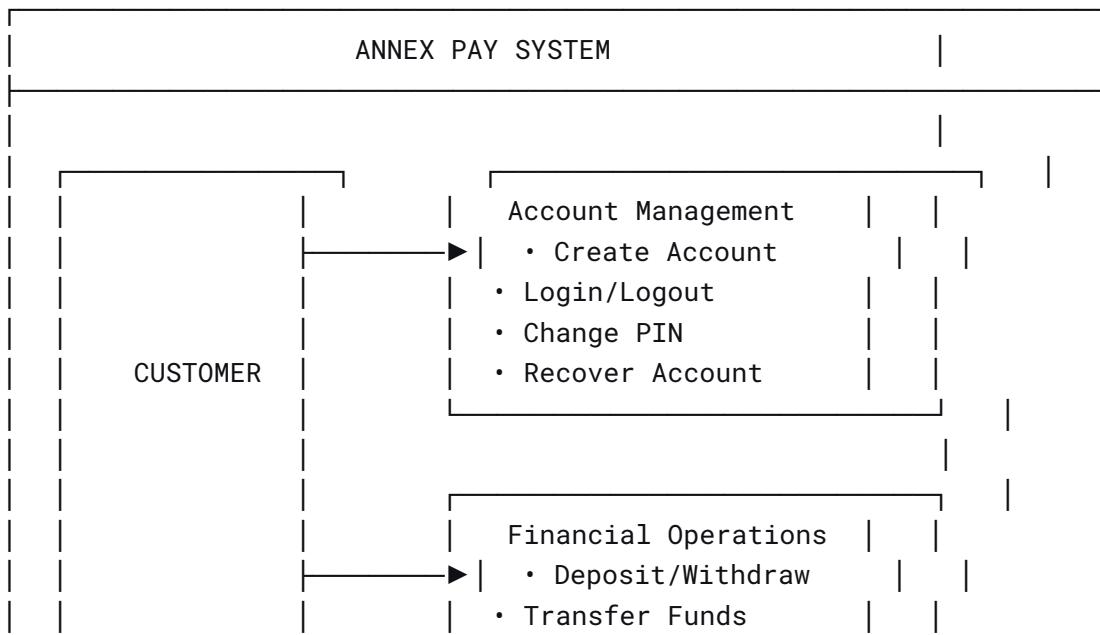
Data Access Layer:

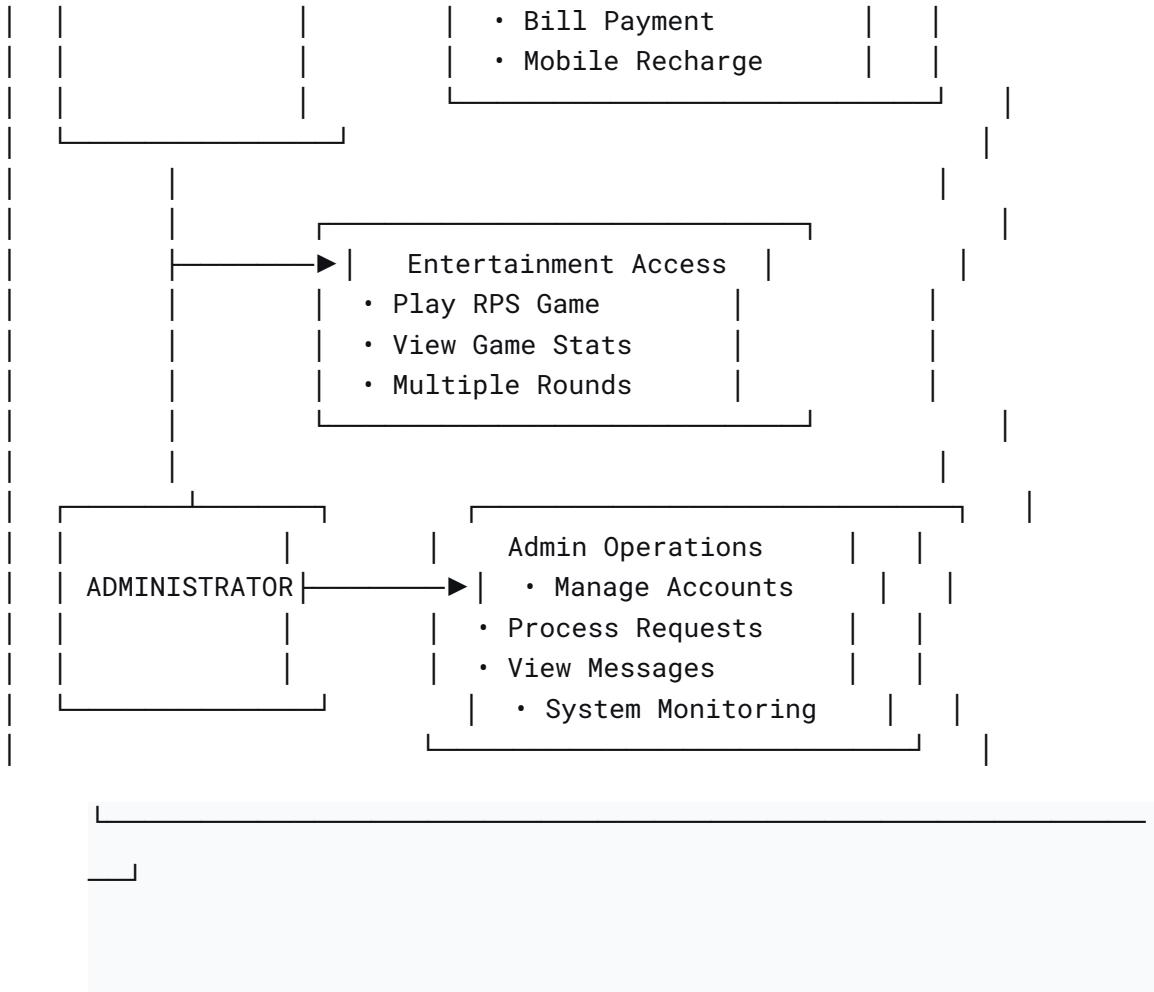
- File I/O for banking data
- Game statistics tracking
- Transaction logging
- Cross-module data consistency

3.3 Use Case Diagram

Figure 2: Use Case Diagram

text





Primary Actors:

Customer:

- Banking Operations (all standard features)
- Entertainment Access:
 - Launch Rock, Paper, Scissors game
 - Play multiple rounds
 - View game statistics
 - Return to banking operations

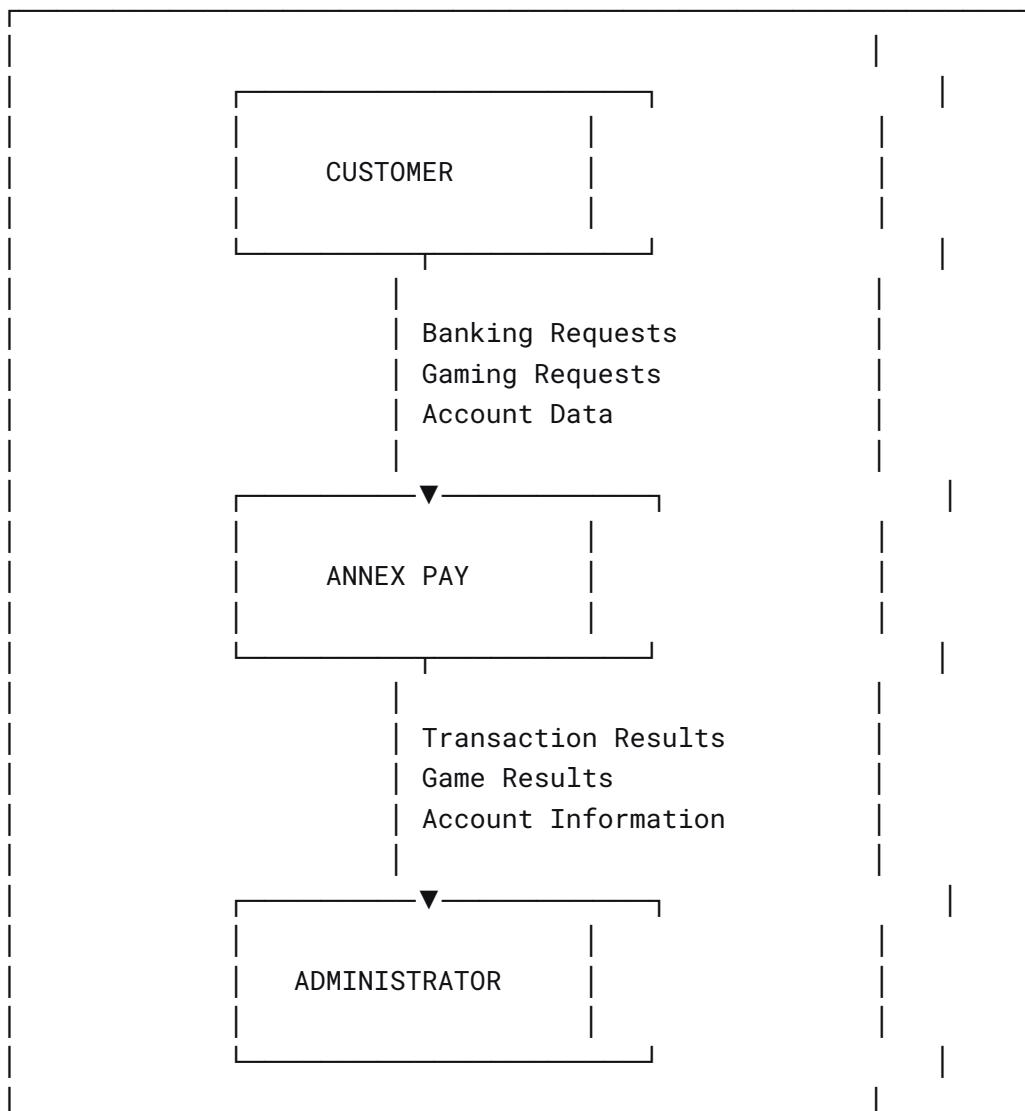
Administrator:

- Standard administrative functions
- System oversight including usage monitoring

3.4 Data Flow Diagrams

Figure 3: Context Level Diagram (DFD-0)

text

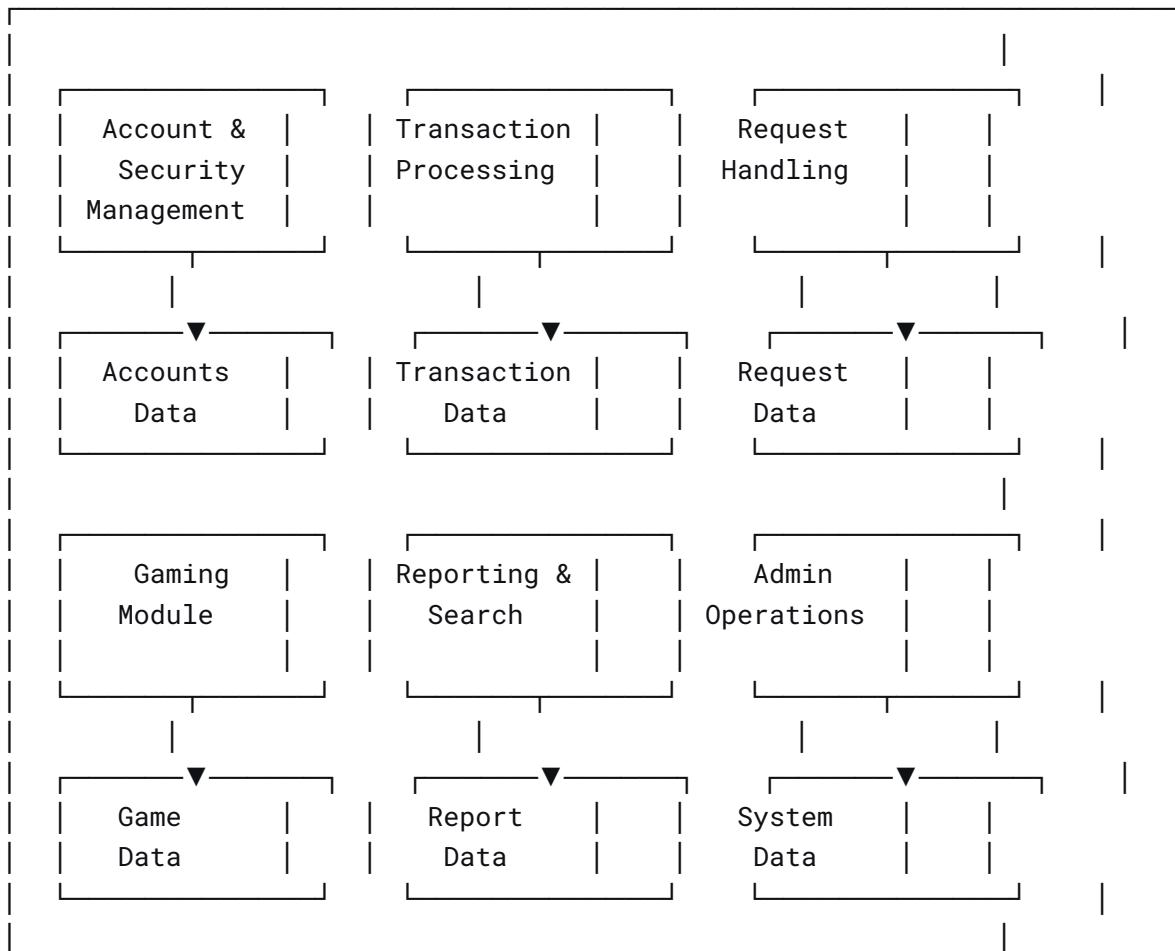


Shows ANNEX PAY as an integrated system interacting with:

- Customer (provides banking/gaming inputs)
- Administrator (system management)

Figure 4: Level-1 Data Flow Diagram

text



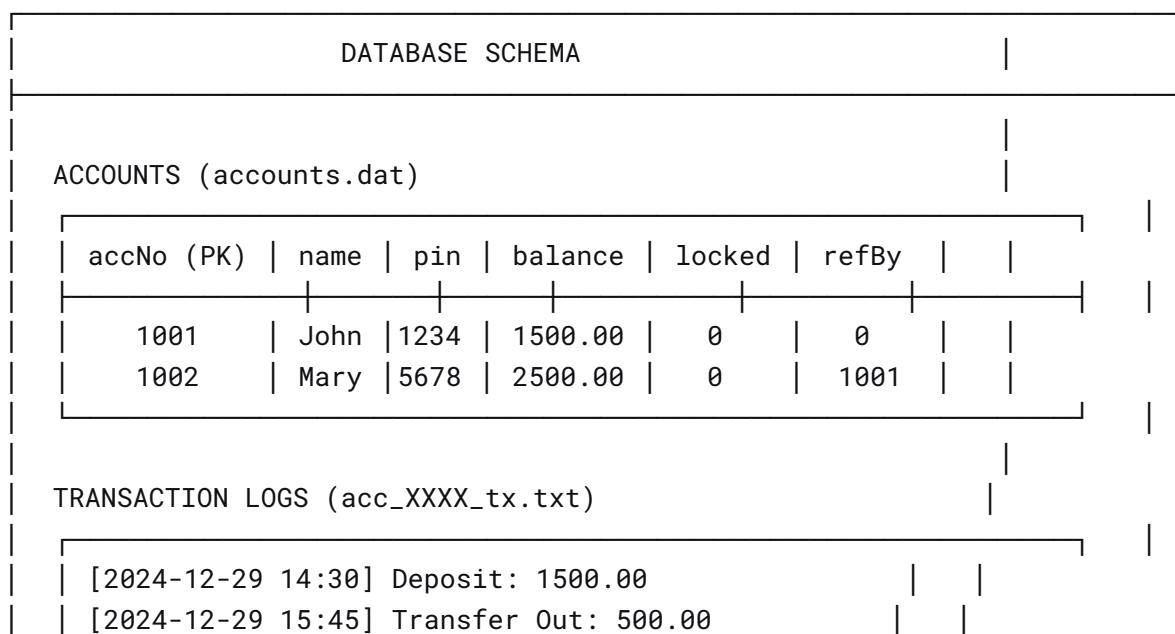
Decomposes system into six processes:

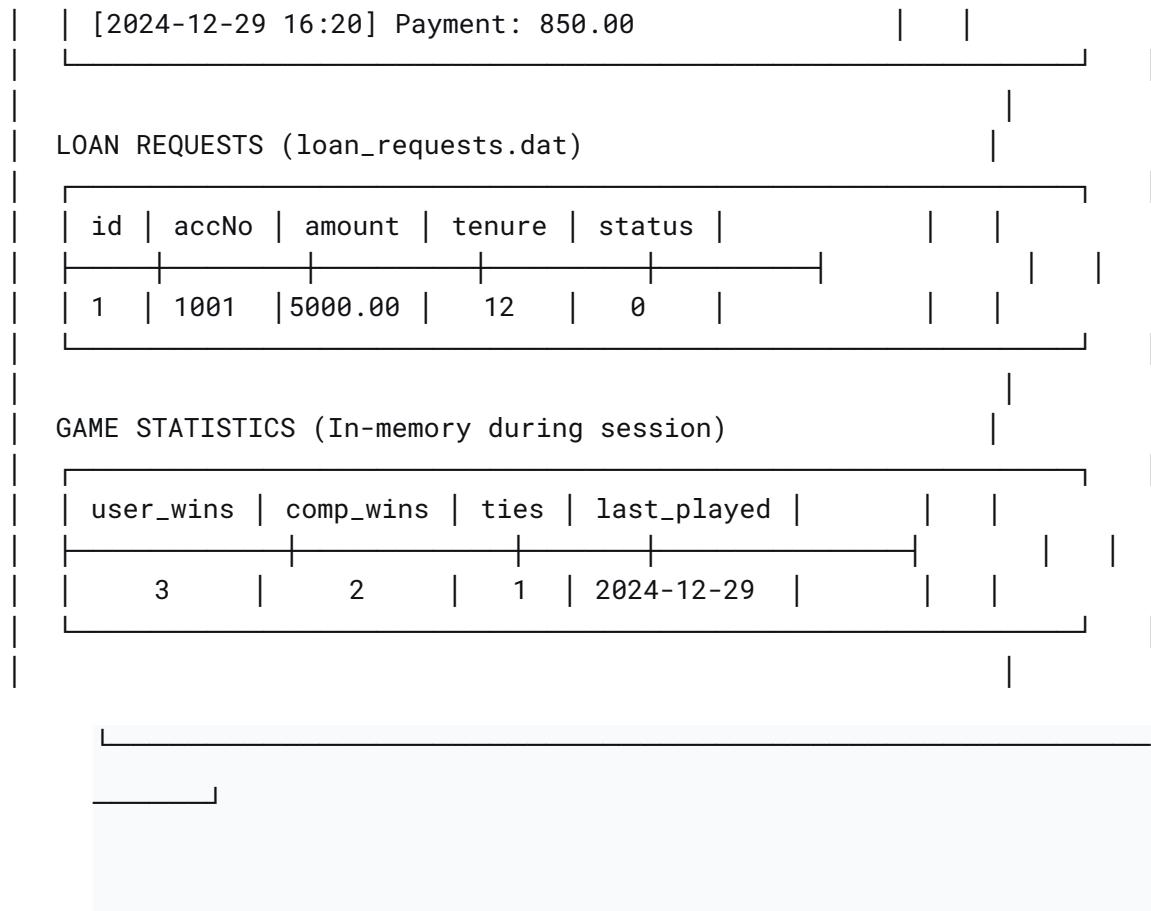
1. Account & Security Management
2. Transaction Processing
3. Request Handling
4. Gaming Module
5. Reporting & Search
6. Admin Operations

3.5 Database Schema

Figure 5: Database Schema

text





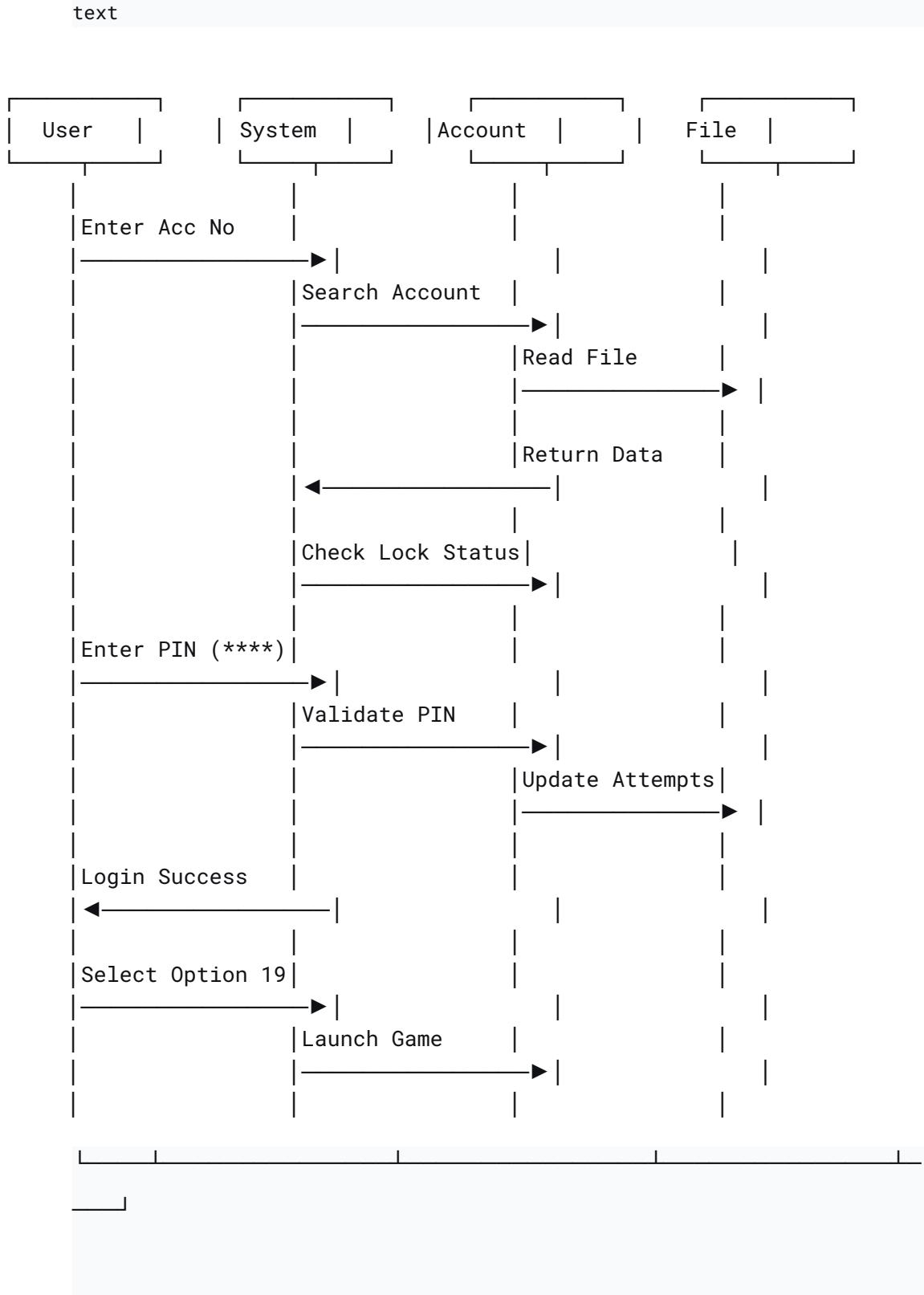
Enhanced Structures:

c

```

// Gaming statistics (integrated with session)
typedef struct {
    int user_wins;
    int annex_pay_wins;
    int ties;
    char last_played[20];
} GameStats;
  
```

3.6 Behavioral Diagrams

Figure 6: Login Sequence Diagram

Enhanced to show gaming option access after authentication.

Game Integration Sequence:

text

User → System: Select "Play Game" (Option 19)
System → Gaming Module: Initialize game environment
Gaming Module → User: Display game interface
User → Gaming Module: Make game choice
Gaming Module: Process game logic
Gaming Module → User: Display results
User → System: Choose to play again or return

System: Restore banking session context

Chapter 4: Implementation

4.1 Interface Design/Front-End

Enhanced Visual Design:

- Extended color scheme for gaming elements:
 - Bright Green (COLOR_GAME_WIN): User wins
 - Bright Red (COLOR_GAME_LOSE): Annex Pay wins
 - Bright Yellow (COLOR_GAME_TIE): Ties
 - Bright Cyan (COLOR_GAME_APP_NAME): Branding

Navigation Architecture:

- Enhanced user dashboard with 20 options:
 - Options 1-18: Banking operations
 - Option 19: Play Game (new integration)
 - Option 20: Logout (relocated)

Gaming Interface Components:

- Animated choice display with frame sequencing
- Score tracking interface with color coding
- Seamless transition animations
- Play-again prompt with flexible input

4.2 Back-End Implementation

Gaming Module Architecture:

Game Initialization:

c

```
void playRockPaperScissors() {
    // Initialize game variables
    int user_wins = 0, annex_pay_wins = 0, ties = 0;
    srand(time(NULL)); // Seed random generator
    // Display welcome message with branding
```

}

Animation System:

```
c

void game_animate_choice() {
    // Display thinking animation
    const char* frames[] = {rock_frame, paper_frame, scissors_frame};
    for (int i = 0; i < 6; i++) {
        printf("%s", frames[i % 3]);
        sleep_ms(300);
        printf("\033[F\033[F\033[F\033[F"); // Move cursor up
    }
}
```

Game Logic Processing:

```
c

void game_determine_winner(char user_choice, char computer_choice,
                           int *user_wins, int *annex_pay_wins, int *ties) {
    // Enhanced game logic with score updating
    if (user_choice == computer_choice) {
        (*ties)++;
        display_tie_message();
    } else if ((user_choice == 'R' && computer_choice == 'S') ||
               (user_choice == 'P' && computer_choice == 'R') ||
               (user_choice == 'S' && computer_choice == 'P')) {
        (*user_wins)++;
        display_win_message();
    } else {
        (*annex_pay_wins)++;
        display_lose_message();
    }
}
```

Session Management:

- Maintain banking session context during gameplay
- Preserve account information for return transition
- Clear game state after session completion

4.3 System Modules

Enhanced Module Structure:

Module 9: Entertainment Integration

- `playRockPaperScissors()`: Main game function with complete gameplay loop
- `game_animate_choice()`: Visual animation for computer "thinking"
- `game_determine_winner()`: Game logic and outcome processing
- `game_display_score()`: Real-time score tracking display
- `game_get_user_choice()`: Validated user input for game choices

Integration Points:

- Seamless calling from user dashboard (option 19)
- Consistent input/output handling across modules
- Shared utility functions (`sleep_ms`, `clearScreen`)
- Unified color management system

Cross-Module Consistency:

- Banking security maintained during gameplay
 - Session continuity preserved
 - Consistent user experience standards
 - Shared platform compatibility layer
-

Chapter 5: User Manual

5.1 System Requirements

Hardware Specifications:

- Processor: 1 GHz or faster
- RAM: 512 MB minimum (1 GB recommended)
- Storage: 50 MB available space
- Display: Console supporting ANSI color sequences

Software Prerequisites:

- Operating Systems: Windows 7+, Linux distributions, macOS
- Compilers: GCC or MinGW
- Terminal applications with color support

5.2 User Interfaces

Figure 7: Main Menu Interface

```

Start here × project_annex_pay.c ×
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using Annex Pay. Goodbye.%s\n", COLOR_GREEN, COLOR_RED);
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s", COLOR_RED, COLOR_RESET);
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

```

ex_pay.c C/C++ Windows

```

" E:\New folder\project_annex × + ▾
== Annex Pay Banking System ==
1) Create Account
2) Login
3) Admin Panel
4) Account Recovery
5) About Annex Pay
6) Exit
Choice:

```

Figure 8: User Dashboard

```

Start here × project_annex_pay.c ×
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using Annex Pay. Goodbye.%s\n", COLOR_GREEN, COLOR_RED);
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s", COLOR_RED, COLOR_RESET);
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

```

C/C++ Windows

```

" E:\New folder\project_annex × + ▾
== User Menu (Acc 1005) ==
1) View Profile
2) Change PIN
3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice:

```

Figure 9: Admin Panel

The screenshot shows a terminal window titled "E:\New folder\project_annex" displaying an admin menu with choices from 1 to 8. Below the menu, the text "Choice:" is followed by a blank line. To the left of the terminal, a code editor window titled "Start here x project_annex_pay.c x" shows C/C++ code. The code includes a switch statement handling choices 1 through 6, and a return statement at the end.

```

2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using Annex Pay. Goodbye.%s\n", COLOR_GREEN, COLOR_RED);
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s\n", COLOR_RED, COLOR_RESET);
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

```

5.3 Screenshot Gallery

Account Management Screenshots:

Figure 10: Account Creation Screen

The screenshot shows a terminal window titled "E:\New folder\project_annex" performing an account creation. It prompts for an account number (1006), full name (Monirul Hasan Mithu), and a referral code (No). It then asks for a 4-digit PIN and creates the account. The terminal also displays a message to press Enter twice to return to the menu. To the left of the terminal, a code editor window titled "Start here x project_annex_pay.c x" shows the same C/C++ code as in Figure 10, specifically the section for creating a new account.

```

2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using Annex Pay. Goodbye.%s\n", COLOR_GREEN, COLOR_RED);
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s\n", COLOR_RED, COLOR_RESET);
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

```

Figure 11: Secure PIN Entry

The screenshot shows a terminal window titled "E:\New folder\project_annex" with the command "C:\Windows\system32\cmd.exe". The window displays the following text:

```
Start here x project_annex_pay.c x
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using Annex Pay. Goodbye.%s\n", COLOR_GREEN, COLOR_RESET);
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s\n", COLOR_RED, COLOR_RESET);
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

E:\New folder\project_annex x + v
== Login ==
Enter account number: 1006
Enter PIN: ****
```

Figure 12: User Login Interface

The screenshot shows a terminal window titled "E:\New folder\project_annex" with the command "C:\Windows\system32\cmd.exe". The window displays the following text:

```
Start here x project_annex_pay.c x
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using Annex Pay. Goodbye.%s\n", COLOR_GREEN, COLOR_RESET);
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s\n", COLOR_RED, COLOR_RESET);
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

E:\New folder\project_annex x + v
== User Menu (Acc 1006) ==
1) View Profile
2) Change PIN
3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice:
```

Transaction Operations Screenshots:

Figure 13: Deposit Operation

```

Start here × project_annex_pay.c ×
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using\n");
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s\n");
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

== User Menu (Acc 1006) ==
1) View Profile
2) Change PIN
3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice: 6
Enter deposit amount: 5000

```

Figure 14: Withdrawal Operation

```

Start here × project_annex_pay.c ×
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using\n");
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s\n");
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

== User Menu (Acc 1006) ==
1) View Profile
2) Change PIN
3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice: 7
Enter withdrawal amount: 2000

```

Figure 15: Fund Transfer Screen

The screenshot shows a C++ IDE interface with two windows. The left window displays the code for 'project_annex_pay.c' containing a switch statement for handling user choices. The right window shows the execution of the program, displaying a user menu with 20 options from View Profile to Logout. The user selects choice 8 (Transfer) and is prompted to enter a receiver account number (1001) and an amount (1000).

```

Start here project_annex_pay.c
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using\n", AnnexPay);
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s\n", AnnexPay);
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

== User Menu (Acc 1006) ==
1) View Profile
2) Change PIN
3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice: 8
Enter receiver account number: 1001
Enter amount to transfer: 1000

```

Banking Services Screenshots:

Figure 16: Bill Payment Menu

The screenshot shows a C++ IDE interface with two windows. The left window displays the code for 'project_annex_pay.c' containing a switch statement for handling user choices. The right window shows the execution of the program, displaying a user menu with 20 options from View Profile to Logout. The user selects choice 9 (Bill Payment) and is prompted to select a bill type. The bill types listed are Electricity, Water, Internet, University Fees, and Other Fees.

```

Start here project_annex_pay.c
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using\n", AnnexPay);
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s\n", AnnexPay);
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

== User Menu (Acc 1006) ==
1) View Profile
2) Change PIN
3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice: 9
Bill types:
1) Electricity
2) Water
3) Internet
4) University Fees
5) Other Fees
Select bill type:

```

Figure 17: Mobile Recharge Interface

```

Start here x project_annex_pay.c x
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using A\n", "AnnexPay");
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s\n", "\033[1;31m", "\033[0m");
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

== E:\New folder\project_annex x +
== User Menu (Acc 1006) ==
1) View Profile
2) Change PIN
3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice: 10
Enter mobile number: 01521796217

```

Request System Screenshots:

Figure 18: Fund Request Screen

```

Start here x project_annex_pay.c x
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using A\n", "AnnexPay");
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice.%s\n", "\033[1;31m", "\033[0m");
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

== E:\New folder\project_annex x +
== User Menu (Acc 1006) ==
1) View Profile
2) Change PIN
3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice: 11
Enter payer account number (who will send funds): 1001
Enter amount to request: 200
Enter note (optional): lunch money

```

Figure 19: Loan Application

The screenshot shows a C/C++ IDE interface with two panes. The left pane displays the source code for `project_annex_pay.c`, specifically the `main()` function which handles user input for a loan application. The right pane shows the terminal window where the program is running, displaying the user menu and the user's choice of 13, followed by prompts for loan amount and tenure.

```

2640     adminPanel();
2641     } else if (ch == 4) {
2642         accountRecovery();
2643     } else if (ch == 5) {
2644         aboutAnnexPay();
2645     } else if (ch == 6) {
2646         clearScreen();
2647         printf("%sThank you for using\n", sleep_ms(1000);
2648         break;
2649     } else {
2650         printf("%sInvalid choice.%s\n", sleep_ms(1000);
2651     }
2652 }
2653
2654
2655
2656
2657
}
return 0;

```

== User Menu (Acc 1006) ==
1) View Profile
2) Change PIN
3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice: 13
Enter loan amount (max 100000.00): 50000
Enter tenure in months (max 60): 12

Reporting & Search Screenshots:

Figure 20: Mini Statement Display

The screenshot shows a C/C++ IDE interface with two panes. The left pane displays the source code for `project_annex_pay.c`, and the right pane shows the terminal window where the program is running, displaying the user menu and the user's choice of 3, followed by the mini statement for the last 5 transactions.

```

1) View Profile
2) Change PIN
3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice: 3
--- Mini Statement (Last 5) ---
[2025-12-01 16:27:27] Deposit: 5000.00
[2025-12-01 16:30:29] Withdraw: 2000.00 (ATM/Cash)
[2025-12-01 16:32:44] Transfer Out: 1000.00 (To 1001)
[2025-12-01 16:33:50] Payment: 1000.00 (Bill Payment: University Fees - Ref: 087)
[2025-12-01 16:35:34] Recharge: 40.00 (Mobile Recharge: 01521796217)
Press Enter twice to return to menu...

```

```

<global>      main(): int
Start here  project_annex.c

3) Mini Statement
4) Transaction History
5) Download Statement
6) Deposit
7) Withdraw
8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice: 4

--- Transaction History ---
[2025-12-01 16:25:58] Deposit: 7000.00
[2025-12-01 16:26:11] Deposit: 10000.00
[2025-12-01 16:27:27] Deposit: 5000.00
[2025-12-01 16:30:29] Withdraw: 2000.00 (ATM/Cash)
[2025-12-01 16:32:44] Transfer Out: 1000.00 (To 1001)
[2025-12-01 16:33:50] Payment: 1000.00 (Bill Payment: University Fees - Ref: 087)
[2025-12-01 16:35:34] Recharge: 40.00 (Mobile Recharge: 01521796217)

Press Enter twice to return to menu...

```

Redo the last editing action

Figure 22: Search Transactions

```

<global>      main(): int
Start here  project_annex.c

8) Transfer
9) Bill Payment
10) Mobile Recharge
11) Fund Request
12) Process Requests
13) Request Loan
14) Security Questions
15) Checkbook Request
16) Messages
17) Refer a User
18) Search Transactions
19) Bored? Play A Game!
20) Logout
Choice: 18
1) Search by Date Range
2) Search by Amount Range
Choice: 1
Enter start date (YYYY-MM-DD): 2025-12-01
Enter end date (YYYY-MM-DD): 2025-12-01

--- Transactions from 2025-12-01 to 2025-12-01 ---
[2025-12-01 16:25:58] Deposit: 7000.00
[2025-12-01 16:26:11] Deposit: 10000.00
[2025-12-01 16:27:27] Deposit: 5000.00
[2025-12-01 16:30:29] Withdraw: 2000.00 (ATM/Cash)
[2025-12-01 16:32:44] Transfer Out: 1000.00 (To 1001)
[2025-12-01 16:33:50] Payment: 1000.00 (Bill Payment: University Fees - Ref: 087)
[2025-12-01 16:35:34] Recharge: 40.00 (Mobile Recharge: 01521796217)

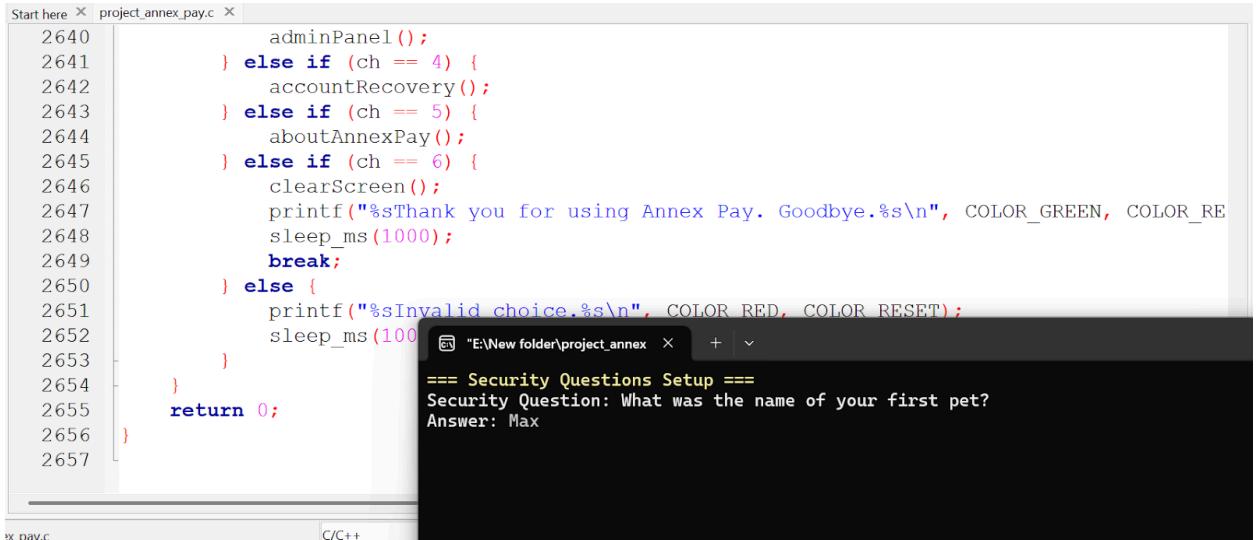
Press Enter twice to return to menu...

```

dipboard

Security & Support Screenshots:

Figure 23: Security Questions Setup

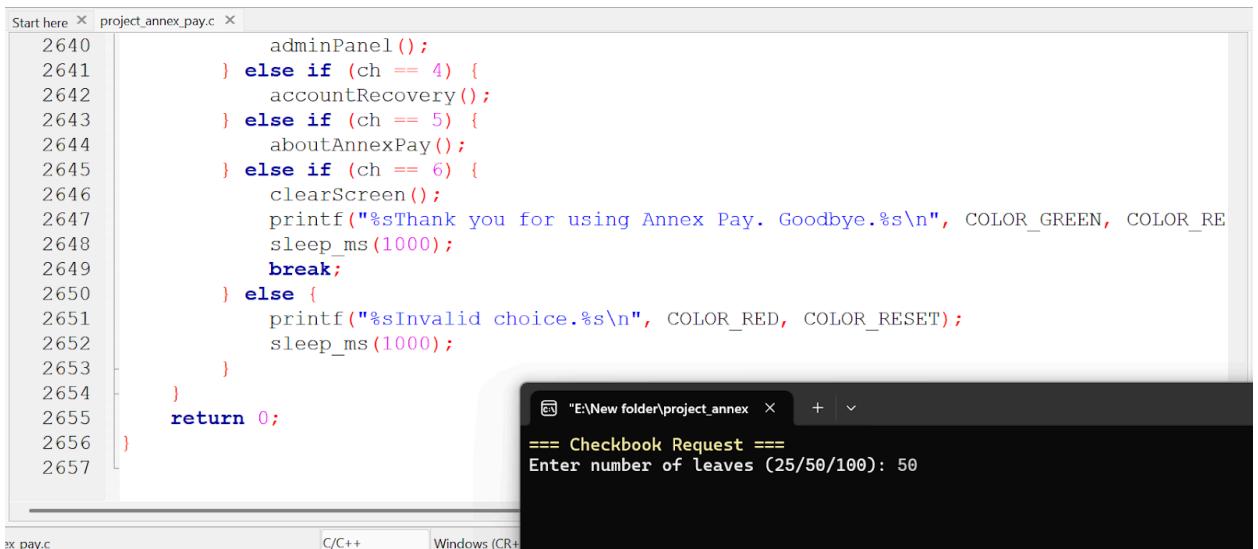


A screenshot of a terminal window titled "E:\New folder\project_annex" showing a security question setup. The terminal output is as follows:

```
== Security Questions Setup ==
Security Question: What was the name of your first pet?
Answer: Max
```

The terminal window is positioned over a code editor window for "project_annex_pay.c". The code editor shows C/C++ code with line numbers from 2640 to 2657. The code includes several `else if` statements for different menu options, including one for a security question.

Figure 24: Checkbook Request



A screenshot of a terminal window titled "E:\New folder\project_annex" showing a checkbook request. The terminal output is as follows:

```
== Checkbook Request ==
Enter number of leaves (25/50/100): 50
```

The terminal window is positioned over a code editor window for "project_annex_pay.c". The code editor shows C/C++ code with line numbers from 2640 to 2657. The code includes several `else if` statements for different menu options, including one for a checkbook request.

Figure 25: Messaging System

A screenshot of a computer screen displaying an integrated development environment (IDE) and a terminal window. The IDE window shows a C program named 'project_annex_pay.c' with code lines 2640 through 2657. The terminal window, titled 'E:\New folder\project_annex', displays a message from the application asking for a message to be sent to an administrator.

```

Start here × project_annex_pay.c ×
2640         adminPanel();
2641     } else if (ch == 4) {
2642         accountRecovery();
2643     } else if (ch == 5) {
2644         aboutAnnexPay();
2645     } else if (ch == 6) {
2646         clearScreen();
2647         printf("%sThank you for using Annex Pay. Goodbye.%s\n", COLOR_GREEN, COLOR_RESET);
2648         sleep_ms(1000);
2649         break;
2650     } else {
2651         printf("%sInvalid choice.%s\n", COLOR_RED, COLOR_RESET);
2652         sleep_ms(1000);
2653     }
2654 }
2655 return 0;
2656
2657

```

==> "E:\New folder\project_annex" + v
==> "E:\New folder\project_annex" + v
==== Send Message to Admin ====
Enter your message: I need help with my loan Application. Can you Check the status?

Gaming Module Screenshots:

Figure 26: Integrated Game Interface

A screenshot of a computer screen displaying an integrated development environment (IDE) and a terminal window. The IDE window shows a C program named 'project_annex_pay.c' with code lines 2640 through 2657. The terminal window, titled 'E:\New folder\project_annex', displays the welcome message for a Rock, Paper, Scissors game, current scores, and a prompt for the user's next move.

```

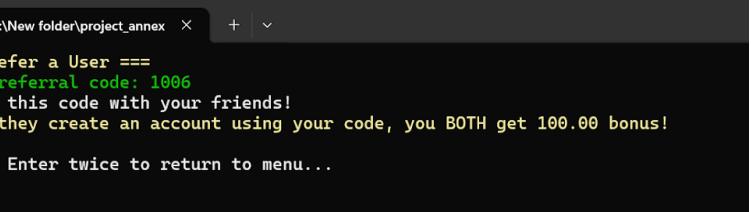
Start here × project_annex_pay.c ×
2640         adminPanel();
2641     } else if (ch == 4) {
2642         accountRecovery();
2643     } else if (ch == 5) {
2644         aboutAnnexPay();
2645     } else if (ch == 6) {
2646         clearScreen();
2647         printf("%sWelcome to Rock, Paper, Scissors, brought to you by Annex Pay!\n", COLOR_GREEN);
2648         sleep_ms(1000);
2649         break;
2650     } else {
2651         printf("%sTake a break and have some fun!\n", COLOR_RESET);
2652         sleep_ms(1000);
2653     }
2654 }
2655 return 0;
2656
2657

```

==> "E:\New folder\project_annex" + v
==== CURRENT SCORE
You: 0 Annex Pay: 0 Ties: 0
=====
--- New Round ---
Rock, Paper, Scissors
Enter your choice (r/p/s): r

Additional Features Screenshots:

Figure 27: Referral Program



A screenshot of a terminal window titled "E:\New folder\project_annex". The window displays a message about a referral program:

```

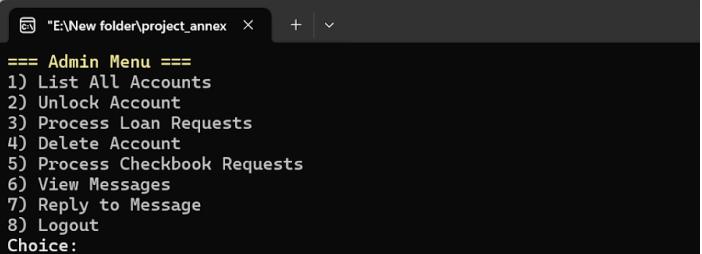
    === Refer a User ===
    Your referral code: 1006
    Share this code with your friends!
    When they create an account using your code, you BOTH get 100.00 bonus!

    Press Enter twice to return to menu...
  
```

The terminal window is positioned over a code editor window titled "Start here x project_annex_pay.c x". The code editor shows a portion of the C/C++ source code, specifically the main loop of the program.

Admin Panel Screenshots:

Figure 28: Admin Login



A screenshot of a terminal window titled "E:\New folder\project_annex". The window displays an admin menu:

```

    === Admin Menu ===
    1) List All Accounts
    2) Unlock Account
    3) Process Loan Requests
    4) Delete Account
    5) Process Checkbook Requests
    6) View Messages
    7) Reply to Message
    8) Logout
    Choice:
  
```

The terminal window is positioned over a code editor window titled "Start here x project_annex_pay.c x". The code editor shows the same portion of the C/C++ source code as Figure 27.

Figure 29: Admin Account Management

```

Start here ✘ project_annex_pay.c ✘
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using Annex Pay\n");
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice\n");
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

"E:\New folder\project_annex" ✘ + | ⏹
===== Admin Menu =====
1) List All Accounts
2) Unlock Account
3) Process Loan Requests
4) Delete Account
5) Process Checkbook Requests
6) View Messages
7) Reply to Message
8) Logout
Choice: 1

===== All Accounts ====
AccNo: 1001 | Name: Monirul Hasan | Balance: 1100.00 | Locked: 0
AccNo: 1002 | Name: Monirul Hasan Mithu | Balance: 1100.00 | Locked: 0
AccNo: 1003 | Name: Monirul Hasan Mithu | Balance: 0.00 | Locked: 0
AccNo: 1004 | Name: Monirul Hasan | Balance: 0.00 | Locked: 0
AccNo: 1005 | Name: Monirul Hasan Mithu | Balance: 0.00 | Locked: 0
AccNo: 1006 | Name: Monirul Hasan Mithu | Balance: 17960.00 | Locked: 0

```

Figure 30: Loan Approval Process

```

Start here ✘ project_annex_pay.c ✘
2640     adminPanel();
2641 } else if (ch == 4) {
2642     accountRecovery();
2643 } else if (ch == 5) {
2644     aboutAnnexPay();
2645 } else if (ch == 6) {
2646     clearScreen();
2647     printf("%sThank you for using Annex Pay\n");
2648     sleep_ms(1000);
2649     break;
2650 } else {
2651     printf("%sInvalid choice\n");
2652     sleep_ms(1000);
2653 }
2654
2655 return 0;
2656
2657

"E:\New folder\project_annex" ✘ + | ⏹
===== Admin Menu =====
1) List All Accounts
2) Unlock Account
3) Process Loan Requests
4) Delete Account
5) Process Checkbook Requests
6) View Messages
7) Reply to Message
8) Logout
Choice: 3

===== Pending Loan Requests ====
ID: 2 | Acc: 1006 | Amount: 50000.00 | Tenure: 12
Enter request ID to process: 2
1) Approve
2) Deny
Choice: 1

```

5.4 System Operation Procedures

Gaming Module Procedures:**Procedure for Accessing Game:**

1. Login to banking system using account credentials
2. From user dashboard, select option 19) Play Game
3. System displays game welcome screen with branding
4. Follow on-screen instructions for gameplay

Procedure for Gameplay:

1. View current score display (initialized to 0-0-0)
2. Enter choice: R (Rock), P (Paper), or S (Scissors)
3. Watch animation as computer makes choice
4. View results with color-coded outcome
5. Choose to play another round or return to banking

Procedure for Returning to Banking:

1. After final game round, select "N" or "No" when prompted to play again
2. System automatically returns to user dashboard
3. Banking session context is fully restored
4. Continue with banking operations from option menu

5.5 Installation Guide

```
bash
```

```
# Windows  
gcc annex_pay.c -o annex_pay.exe
```

```
# Linux/macOS
```

```
gcc annex_pay.c -o annex_pay
```

Execution:

```
bash
```

```
# Windows  
annex_pay.exe
```

```
# Linux/macOS
```

```
./annex_pay
```

First-Time Setup:

1. Compile source code
2. Execute binary
3. System creates data directory structure
4. Create initial accounts
5. Access admin features with default credentials

Default Credentials:

- Admin Password: bibt1234
- Account Numbers: Auto-generated from 1001

Chapter 6: Conclusion

6.1 Conclusion

ANNEX PAY has successfully achieved its objectives by creating an innovative banking system that integrates entertainment features without compromising core functionality. The project demonstrates that utility applications can be enhanced with engaging elements to improve user experience and session enjoyment.

The integrated Rock, Paper, Scissors game provides a refreshing break option during banking sessions, addressing user fatigue while maintaining session continuity. This approach represents a novel application of gaming principles in utility software, creating a more enjoyable banking experience.

Technical achievements include seamless module integration, consistent user experience across banking and gaming modes, robust session management, and cross-platform compatibility. The system successfully balances security requirements with accessibility needs, proving that entertainment features can be integrated into secure systems without compromising protection measures.

From an educational perspective, ANNEX PAY demonstrates advanced software engineering concepts including modular design, system

integration, user experience optimization, and practical application of gaming principles in non-gaming contexts.

6.2 Limitations

Security Considerations:

- Gaming module operates within authenticated sessions only
- No persistent storage of game statistics across sessions
- Limited game complexity due to console interface constraints

User Experience Constraints:

- Basic game graphics limited by console capabilities
- No sound effects or advanced animations
- Limited game variations available

Technical Limitations:

- Game state not preserved between banking sessions
- No multiplayer or competitive features
- Basic scoring system without achievement tracking

Integration Challenges:

- Separate modules with minimal data sharing
- Basic transition animations
- Limited customization options for gaming experience

6.3 Future Works

Enhanced Gaming Features:

- Multiple game options (Tic-Tac-Toe, Number Guessing, Memory Games)
- Achievement system linked to banking activities
- Tournament mode with leaderboards
- Sound effects and enhanced animations

Integration Improvements:

- Persistent game statistics across sessions
- Banking rewards for gaming achievements
- Social features (challenge friends, share scores)
- Game customization options

Technical Enhancements:

- Graphical user interface for both banking and gaming
- Network capabilities for multiplayer gaming
- Database integration for game statistics
- Mobile app version with touch controls

Educational Extensions:

- Financial literacy mini-games
- Budgeting and saving challenges
- Investment simulation games
- Financial quiz competitions

Enterprise Applications:

- Corporate banking with team challenges
 - Educational banking simulations
 - Financial training through gamification
 - Customer engagement analytics
-

References

1. Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (2nd ed.). Prentice Hall.
2. Kanetkar, Y. (2020). *Let Us C* (18th ed.). BPB Publications.
3. Schell, J. (2020). *The Art of Game Design: A Book of Lenses* (3rd ed.). CRC Press.
4. Rogers, S. (2014). *Level Up! The Guide to Great Video Game Design*. Wiley.
5. "Game Development Principles in Utility Applications." (2022). *Journal of Interactive Systems*.
6. "User Engagement Strategies in Financial Software." (2023). *International Conference on Human-Computer Interaction*.

Stevens, W. R. (1992). *Advanced Programming in the UNIX Environment*. Addison-Wesley.

Appendix A: Source Code

Source Code Overview:

Complete source code in `annex_pay.c` (~2,500 lines) featuring:

- Integrated banking and gaming modules
- Comprehensive documentation
- Consistent coding standards
- Cross-platform compatibility

Compilation Instructions:

bash

```
# Recommended compilation  
gcc -Wall -Wextra -std=c99 -O2 annex_pay.c -o annex_pay
```

Execution:

bash

```
./annex_pay    # Linux/macOS  
annex_pay.exe # Windows
```

Test Cases for Gaming Module:

1. Basic Gameplay Test:
 - Login to banking system
 - Select option 19 (Play Game)
 - Play multiple rounds

- Verify score tracking
 - Return to banking menu
2. Session Continuity Test:
- Perform banking transactions
 - Play game
 - Return to banking
 - Verify session context preserved
3. Input Validation Test:
- Test valid game inputs (R, P, S)
 - Test invalid inputs (other characters)
 - Verify error handling and re-prompting
4. Cross-Platform Test:
- Compile and run on Windows
 - Compile and run on Linux
 - Verify consistent gaming experience

Key Gaming Features Demonstrated:

- Real-time game processing
- Visual animations
- Score tracking
- Input validation
- Seamless banking integration
- Cross-platform compatibility
- User-friendly interface

File Structure After Gaming Sessions:

```
text
```

```
AnnexPay/
└── annex_pay (executable)
└── AnnexPay/ (Data Directory)
    ├── accounts.dat
    └── transaction files
```

| └— (Gaming statistics maintained in memory)
| └— (No separate game files - integrated storage)

END OF PROJECT REPORT