UNIVERSITY OF PETROLEUM AND ENERGY STUDEIES

PROJECT REPORT

FOR THE

MAJOR PROJECT

(TOPIC : VEHICLE TOLL CONTROL SYSTEM)

USING

PROGRAMMING IN C (COURSE CODE : CSEG1032)

SESSION : 2025-2026 / 1ST SEMESTER

SUBMITTED TO :
SUBMITTED BY :

DR. TANU SINGH

NAME : Shreyansh Deshwal

SAP ID : 590024396

BATCH NO .: B37

ABSTRACT :

The Mines Betting Game is a simple console-based application developed using the C programming language. In this game, the player begins with an initial balance, places a bet, and selects a tile between 1 and 9. The program randomly hides two mines, and the player must avoid selecting a tile that contains a mine. If the chosen tile is safe, the player wins an amount equal to their bet; otherwise, the bet is lost.

The project demonstrates core C programming concepts such as random number generation, loops, conditional statements, input validation, and the use of standard libraries like stdlib.h and time.h. It provides a basic understanding of how chance-based games work and how user interaction can be handled in a console environment. Overall, the project helps improve logical thinking and strengthens the fundamentals of programming in C.

PROBLEM DEFINATION :

Betting-style games require randomness, decision-making, and continuous interaction with the user. In traditional offline games, managing randomness and balance manually is difficult and unreliable.

The problem is to create a system that:

- Randomly places mines in different positions each round
- Allows the user to enter a bet amount
- Checks whether the selected tile contains a mine

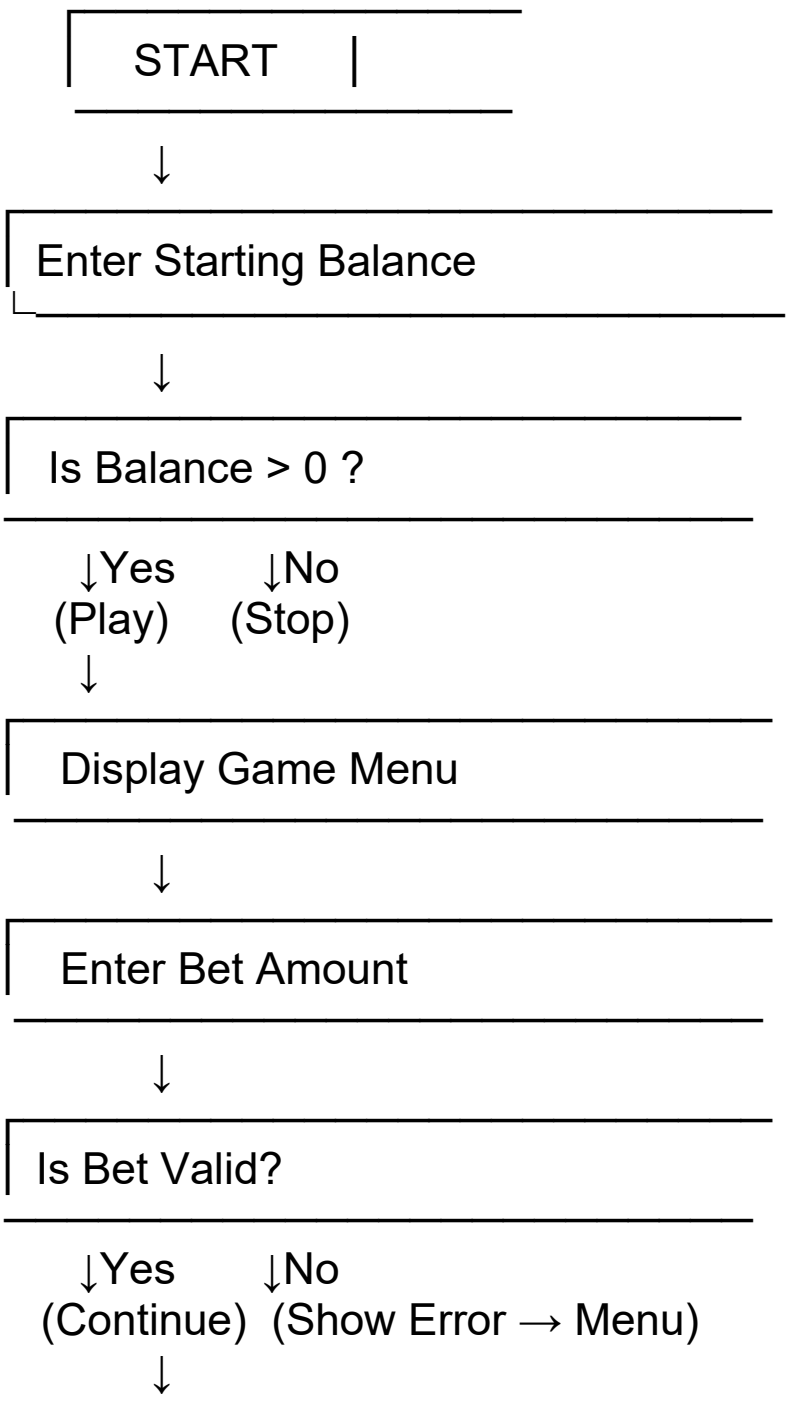
The goal is to implement all these features using only basic C programming concepts, without graphics or external frameworks.

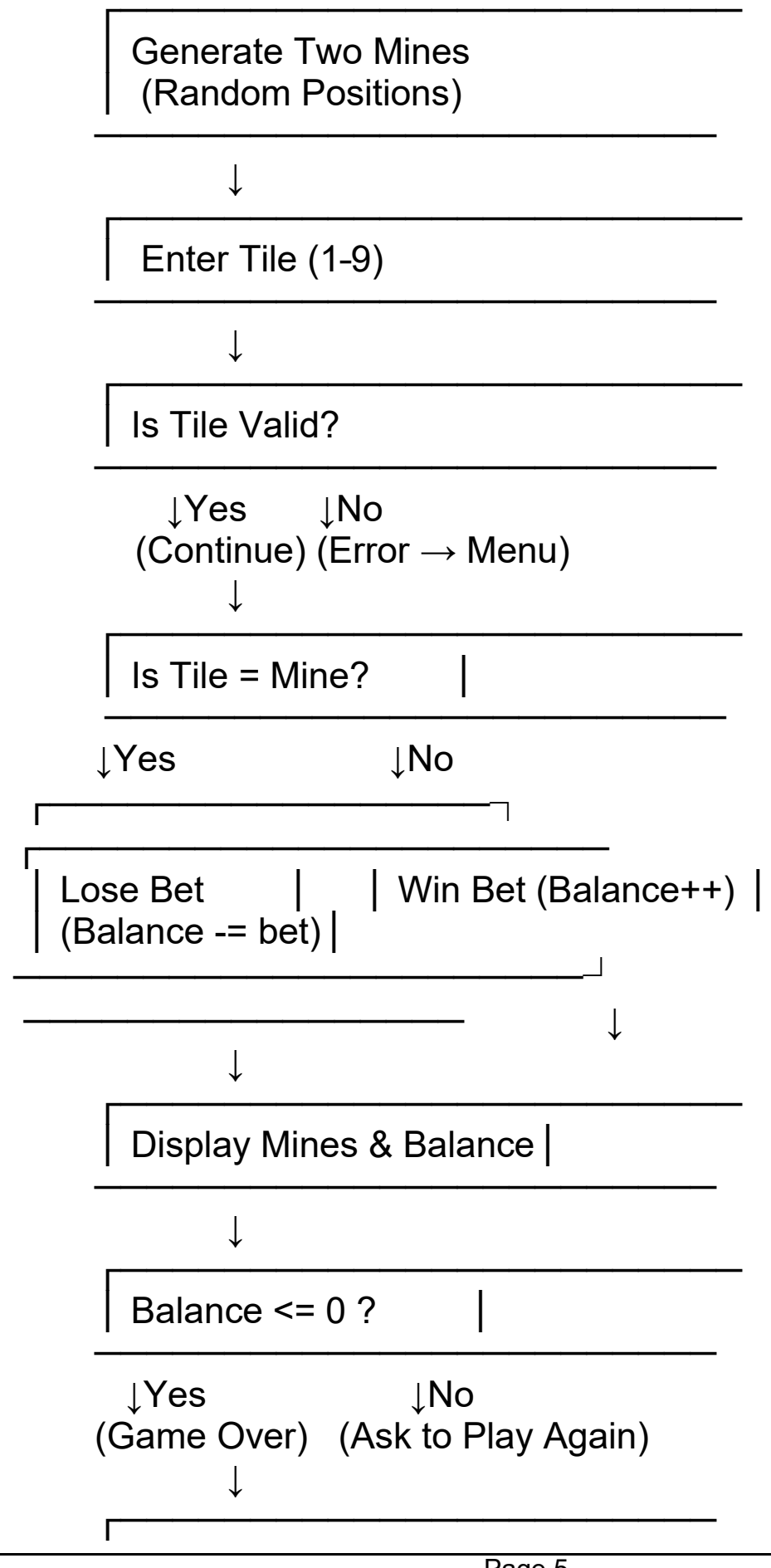
Objective:

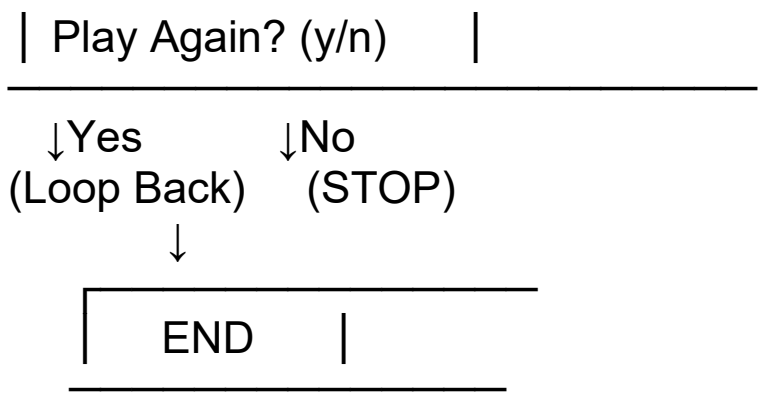
1. **To develop a simple console-based betting game using C programming.** The system should allow the user to interact with the game through a menu-driven interface.
2. **To generate random mine positions using rand() and srand() functions.** This ensures that the mine locations change every round, adding unpredictability to the game.
3. **To allow the user to place bets and select tiles between 1 and 9.** The program must validate the bet amount and the tile choice.
4. **To determine whether the chosen tile is safe or contains a mine.** If the user hits a mine, the bet is lost; otherwise, the player earns a reward.
5. **To update the user's balance after every round.**
   The system should accurately adjust the balance based on winning or losing.

SYSTEM DESIGN :

FLOWCHART :




Generate Two Mines
(Random Positions)
Enter Tile (1-9)
Is Tile Valid?
↓Yes
(Continue)
↓No
(Error → Menu)
Is Tile = Mine?
↓Yes
↓No
Lose Bet
(Balance -= bet)
Win Bet (Balance++)
Display Mines & Balance
Balance <= 0 ?
↓Yes
(Game Over)
↓No
(Ask to Play Again)


Play Again? (y/n)
Yes
(Loop Back)
No
(STOP)
END

ALGORITHM :

1. **Start the program.**
2. Ask the user to **enter the starting balance**.
3. If the balance is **less than or equal to 0**, display an error and **stop** the program.
4. Begin a **do-while loop** to continue gameplay until the user chooses to stop.
5. Display the current balance to the user.
6. Ask the user to **enter a bet amount**.
7. Check if the bet is valid:
    - It must be **greater than 0**.
    - It must be **less than or equal to the current balance**.
      If invalid, show an error and go back to the start of the loop.
8. Generate two different **random mine positions** between 1 and 9 using rand() % 9 + 1.
9. Ask the user to **choose a tile** (1 to 9).
10. Check if the tile number is valid:
    - If invalid, show "Invalid tile" and restart the loop.

    Compare the chosen tile with both mine positions:
    - If the tile equals mine1 or mine2 → **User hits a mine**
        - Deduct the bet from balance.
    - Else → **Tile is safe**
        - Add the bet to balance (player wins).
11. Display the mine positions and the updated balance.
12. If the balance becomes **0 or less**, display "Game Over" and exit the loop.
13. Ask the user, "Do you want to play again? (y/n)".
14. If the user enters **'y'**, repeat the loop.
    If the user enters **'n'**, exit the loop.
15. Display the **final balance**.
16. **End the program.**

IMPLEMENTAQTION DETAILS:

The Mines Betting Game is implemented using basic C programming concepts and standard libraries. The program consists of a simple loop-based game structure where the user places bets and selects tiles, while the program randomly generates mine positions and determines the outcome.

1. Use of Standard Libraries

- **stdio.h** – Used for input and output operations (printf, scanf).
- **stdlib.h** – Used for the random number generator (rand, srand).
- **time.h** – Used to seed the random number generator with the current time using time(0).
-

2. Random Mine Generation

The game randomly places **two mines** between tiles 1 and 9.

The expressions used:

mine1 = rand() % 9 + 1;

mine2 = rand() % 9 + 1;

A do-while loop ensures the two mine positions are never the same.

3. User Input Handling

The user is asked to:

- Enter starting balance
- Enter bet amount
- Select a tile
- Decide whether to play again

All inputs are validated to prevent invalid or out-of-range values.

4. Game Logic

- If the selected tile matches either mine:
  → The user **loses** the bet and balance decreases.
- If the tile is safe:
  → The user **wins**, and the bet amount is added to their balance.
-

5. Loop Structure

A **do-while loop** allows the player to keep playing until they choose to stop or their balance becomes zero.
Inside the loop:

- The menu is displayed
- Bet is processed
- Random mines are generated
- Outcome is shown
- User is asked if they want another round
-

6. Balance Management

Balance is updated using simple arithmetic:

- balance -= bet; → when losing
- balance += bet; → when winning

The game checks if the balance becomes **0**, and ends automatically.

7. Program Termination

After the user quits or loses all balance, the program prints the final balance and terminates normally using:
return 0;

TESTING AND RESULT :

1. Testing Approach

The Mines Betting Game was tested using basic input-based testing. Each feature of the program was executed with different values to ensure correct behavior. The main areas tested were:

- Valid and invalid starting balance
- Valid and invalid bet amounts
- Repeated gameplay using different choices
- Correct generation of random mine positions
- Correct detection of mine hits and safe tiles
- Proper balance updates after each round
- Handling of invalid tile numbers (less than 1 or greater than 9)
- Correct termination of game when balance becomes zero

The program was executed multiple times to verify that randomness works correctly and mine positions change every round.

---

2. Test Cases & Expected Behavior

Test Case	Input	Expected Result
Starting balance = 0 or negative	0 / -10	Program shows error and exits
Bet greater than balance	Balance = 100, Bet = 200	“Invalid bet amount” and restart round
Valid bet and safe tile	Tile not equal to mine1 or mine2	User wins; balance increases
Valid bet but mine hit	Tile equals mine1 or mine2	User loses; balance decreases
Tile < 1 or > 9	Tile = 0 or 15	Invalid tile message

Test Case	Input	Expected Result
User chooses to play again	'y'	Next round starts
User chooses not to play	'n'	Game ends and final balance shown

---

3. Sample Output

Sample gameplay output (example):

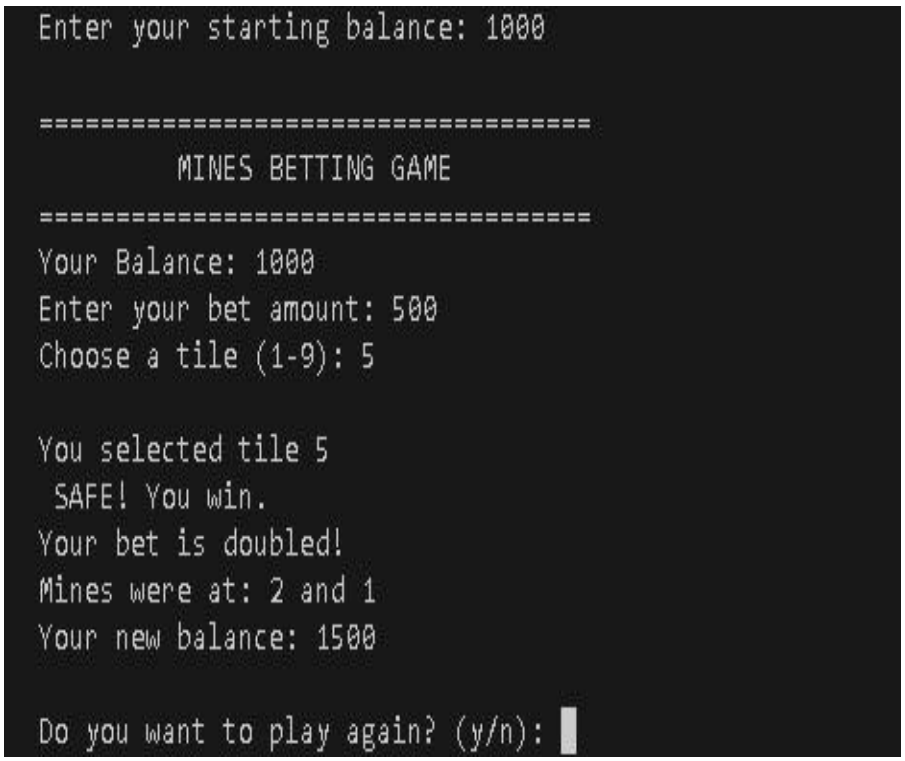
```
==================================
      MINES BETTING GAME
==================================
Enter your starting balance: 200

Your Balance: 200
Enter your bet amount: 50
Choose a tile (1-9): 4
You selected tile 4
SAFE! You win.
Mines were at: 2 and 7
Your new balance: 250
Another example when user hits a mine:
Choose a tile (1-9): 3
* BOOM! You hit a mine.
You lost your bet.
Mines were at: 3 and 8
Your new balance: 150
```

---

4. Result

The program performed successfully in all tested scenarios.
The game logic worked correctly, random mines were generated in each round, and the balance was updated accurately. The input validation prevented incorrect user entries, and the repeated gameplay loop functioned smoothly until the user exited the program.
Overall, the Mines Betting Game met all functional requirements and produced correct results under various test conditions.



CONCLUSION :

The Mines Betting Game was successfully developed using basic concepts of the C programming language. The project demonstrated how random number generation, user input handling, conditional statements, and looping structures can be combined to create an interactive console-based game. The game logic worked effectively by placing random mines, validating user bets, updating the balance, and determining win/loss outcomes.

Through this project, important programming skills were strengthened, including problem-solving, input validation, control flow design, and testing. The project also showed how a simple game can be implemented without advanced tools or graphics, relying only on fundamental C libraries. Overall, the Mines Betting Game met its objectives and provided a practical understanding of how chance-based systems and user-driven applications can be built in C.

FUTURE WORK :

The Mines Betting Game can be enhanced with several additional features to make it more engaging and user-friendly. Some possible future improvements include:

- **Adding a graphical interface (GUI)** instead of a text-based console to improve the user experience.
- **Increasing the number of tiles and mines**, allowing for more complex difficulty levels.
- **Implementing a bonus or multiplier system** to reward successful streaks.
- **Adding sound effects or animations** for better game feedback.
- **Allowing the user to choose the difficulty level**, such as Easy, Medium, or Hard.
- **Storing game history using file handling**, including winnings, losses, and mine positions for each round.
- **Creating a leaderboard** that tracks highest balances or scores.
- **Developing a mobile or web-based version** of the game for wider accessibility.

These enhancements can transform the simple console game into a more interactive and feature-rich application.

REFERENCES :

Let Us C – Yashavant Kanetkar

Class notes by DR. TANU SINGH