COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE LAB

National University
of computer and emerging sciences

PROJECT REPORT

"2D PLATFORMER"

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCS KARACHI CAMPUS

Group Members:

Muhammad Ali 23K-0730

Ayan Hussain 23K-0608

Umer Khan 23K-0798

Submitted to: Engr Nadeem Ghouri

Objective:

This project aimed to use the Irvine32 library to create and construct a basic 2D platformer game in Assembly language. The user controls a character in the game that moves around the screen, avoids enemies, and gathers coins. Basic gravity mechanisms, score tracking, and a game-over condition upon player-enemy collision are all included in the game.

Tools and Technologies:

- **Programming Language:** Assembly Language (Irvine32)
- **Library:** Irvine32 (for input/output functions and screen handling)
- **Platform:** Windows (assuming compatibility with Irvine32 library)

Project Overview:

This game simulates a 2D platformer environment where the player's character ("X") can move left, right, and jump. The game includes the following key features:

1. **Player Movement:** The player can move left, right, and jump.
2. **Coin Collection:** The player can collect coins, which are randomly generated at the bottom of the screen.
3. **Enemy:** A simple enemy ("E") moves around and can cause the game to end if the player collides with it.
4. **Score Tracking:** The score increases each time the player collects a coin.
5. **Game Over:** The game ends when the player collides with the enemy, and the final score is displayed.

The game uses assembly macros and procedures for better code organization, with specific routines for drawing characters on the screen, handling user input, and updating the game state.

Functionality Breakdown:

1. **Macros:**
    - **Positioning Macros:** Used to move the cursor on the screen for placing game elements (e.g., Position2, IndexPosition).

- **Print String Macro:** Oversees printing messages to the screen with the PrintString macro.
- **Welcome Note Macro:** Displays the welcome screen with game instructions and credits.

2. **Game Mechanics:**
   - **Player Movement:** The player's character is moved using keyboard inputs. The available controls are:
     - W: Move up (jump)
     - A: Move left
     - D: Move right
     - X: Exit the game
   - **Gravity:** A simple gravity system is implemented where the player falls on the ground.
   - **Collision Detection:** If the player's position matches the enemy's position, the game ends.
   - **Coin Generation:** Coins are randomly generated on the screen and are collected when the player overlaps with them.
3. **Game Flow:**
   - **Start Screen:** The welcome screen displays instructions and credits before starting the game.
   - **Main Game Loop:** The game runs in a loop, where the player can interact with the game world by moving, jumping, and collecting coins while avoiding the enemy.
   - **Game Over:** If the player collides with the enemy, the game ends, and the final score is displayed.
4. **Graphics and User Interface:**
   - **Player:** Represented by the character "X" in light blue.
   - **Enemy:** Represented by the character "E" in red.
   - **Coin:** Represented by "X" in Yellow.
   - **Ground:** The ground is represented by a line of hyphens ("-").
5. **Game Over Screen:**
   - The game-over screen displays a message and a simple ASCII art representation of the game-over state, followed by the final score.

Project Design:

Basic gameplay features like movement, gravity, and collision detection were all included in the project's simple yet captivating design. For repetitive activities like moving the mouse and printing strings, the game's design mostly relied on macros. Readability was improved and the quantity of code was decreased because of this structure. The game is divided into three sections:

1. **Macros for Screen Handling:** To manage screen output, positioning, and text printing.
2. **Game Logic:** Handling player input, updating positions, and checking for collisions.
3. **Procedures for Drawing Objects:** These include procedures for drawing the player, coins, and enemies on the screen.
4. **Game Over Handling:** When the player collides with the enemy, the game transitions to a game-over state, displaying ASCII art and the final score.

Challenges:

1. **Implementing Gravity:** Simulating gravity and player movement was challenging, especially ensuring smooth player jumps and falls. The jump mechanism needed to be timed carefully with gravity, ensuring the player would return to the ground after jumping.
2. **Collision Detection:** The basic collision detection system checks if the player’s position matches the enemy's position. A more sophisticated method could be developed to check for proximity or overlap, which would improve the gameplay experience.
3. **Random Coin Generation:** Ensuring that the coin does not spawn on top of the player or enemy requires additional logic to avoid overlaps.
4. **Performance Issues:** Although this was a simple project, assembly language does not allow the same level of abstraction as high-level programming languages. Optimizing performance and ensuring that the screen is updated efficiently was an important consideration.

Conclusion:

The Irvine32 library was used to successfully develop this 2D platformer game project in assembly code. Player movement, coin collection, opponent avoidance, and score monitoring are all standard platformer mechanics in the game. To keep things organized and guarantee effective screen management and user input processing, the code makes use of macros and procedures.

Notwithstanding the game's simplicity, the project gave participants useful experience with low-level programming by showing how basic game logic may be implemented in assembly language. More intricate enemy behavior, improved collision detection, and other game features like power-ups or multiple levels are probable future additions.

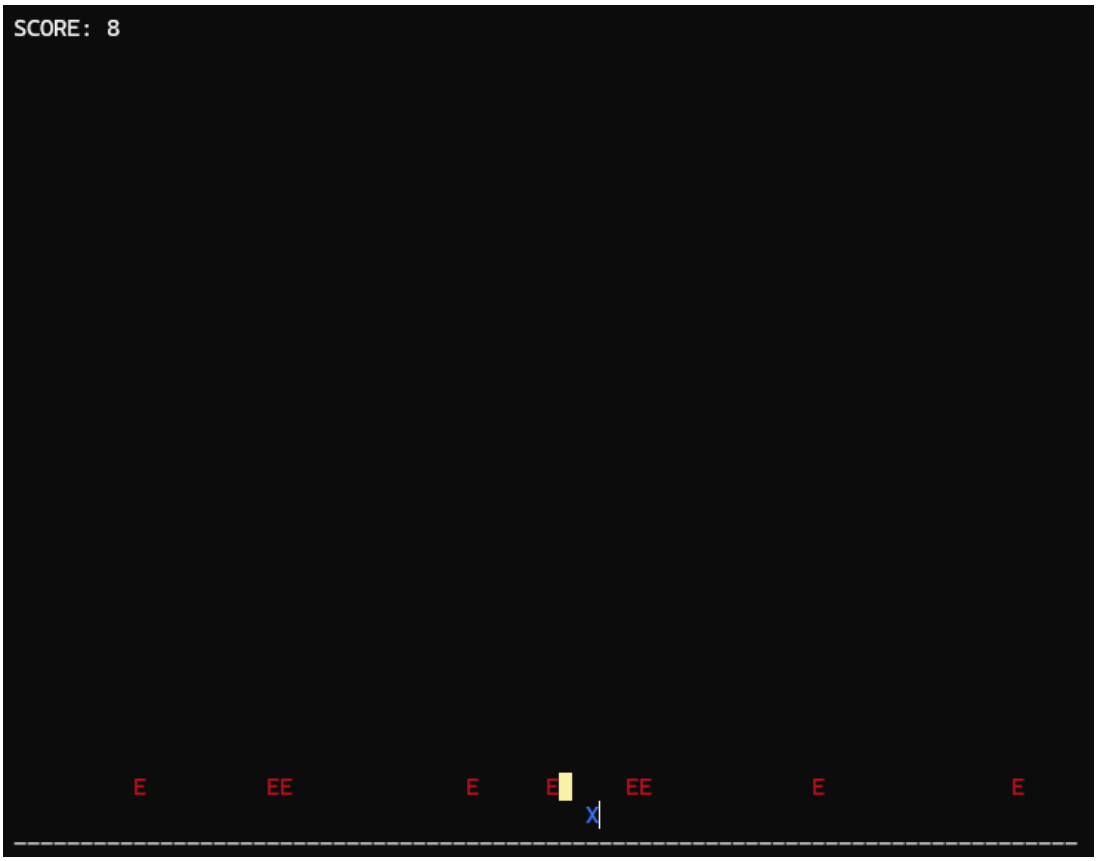
Future Improvements:

1. **Enhanced AI for the Enemy:** The enemy could have more complex behavior, such as chasing the player or moving in a specific pattern.
2. **Better Collision Detection:** Implementing a more accurate and flexible collision detection system could improve gameplay.
3. **Player Animations:** Adding animations for the player’s movement (e.g., jumping or running) would enhance the game’s visual appeal.
4. **Sound Effects and Music:** Introducing sound effects for actions like jumping, collecting coins, or enemy collisions would create a more immersive experience.
5. **Multiple Levels:** Adding various levels with increasing difficulty or more obstacles could extend the gameplay and challenge the player further.

Screenshots

1. Initial Game Screen

2. Gameplay in Action



3. Game Over Screen