# **COAL PROJECT REPORT**



# SUFWAN MASOOD 2023-CS-710

# HAMNA NASIR 2023-CS-714

# MAHNOOR 2023-CS-732

# ABDUL REHMAN WAHLA 2023-CS-717

Safwan

# Problem Statement:

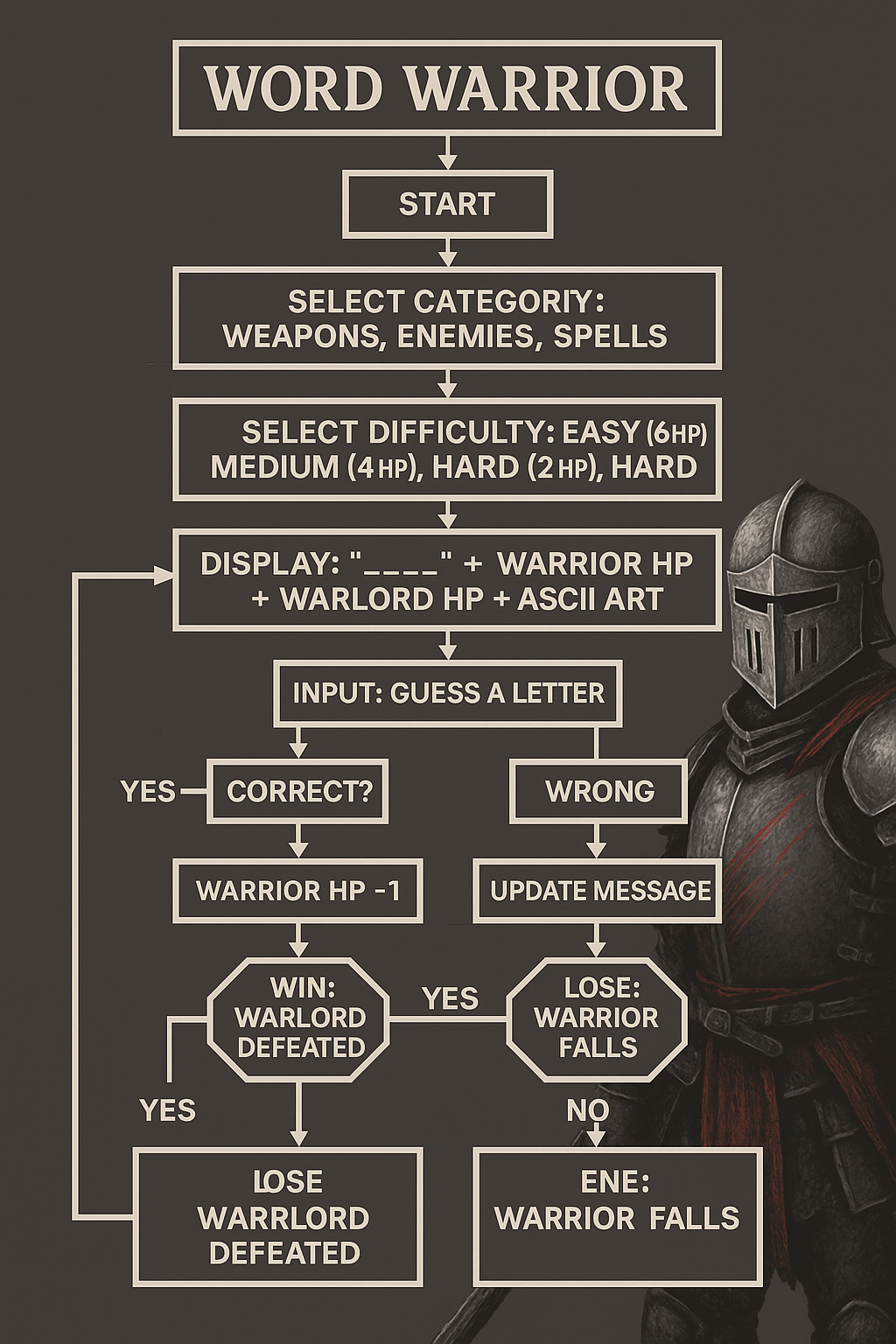
Playing Hangman can feel boring because it’s just guessing letters without a cool story or exciting visuals. People learning to code in a tough language like assembly want fun projects that let them be creative while practicing their skills. There’s no game out there where you fight an enemy by guessing words, with choices like difficulty levels or neat text art to make it lively. "Word Warrior" fixes this by turning Hangman into a battle where you, the warrior, take down a warlord by guessing his secret word. It’s fun, looks great with text pictures, and helps you learn coding in a way that feels like an adventure.

# Intro Objectives:

We want to create "Word Warrior," a game made with assembly coding that feels like a warrior facing off against a warlord. Here’s what it’ll do:

* **Pick a Theme**: Choose from lists like Weapons (e.g., "SWORD"), Animals (e.g., "HORSE"), or Places (e.g., "RIVER"). All words are five letters to keep it fair.
* **Choose How Hard**: Pick Easy (you start with 6 lives), Medium (4 lives), or Hard (2 lives) to match your skill.
* **Keep Score and Lives**: Guess a letter right, and you hit the warlord, lowering his lives (starts at 5, one per letter) and earning 10 points. Guess wrong, and you lose a life and 5 points. Win, and you get extra points for lives left.
* **Show Cool Text Art**: The warlord looks different as he loses lives similarly there are different ASCII animations for different game stages.
* **Learn While Playing**: Show off coding tricks like handling words, getting player guesses, and making decisions, all while making a game you want to play again.  
  The idea is to make coding feel like building a fun world, not just solving puzzles

# Flow Diagram:



[Start]

|

[Select Category: Weapons, Enemies, Spells]

|

[Select Difficulty: Easy (6 HP), Medium (4 HP), Hard (2 HP)]

|

[Set Warrior HP = Difficulty, Warlord HP = Word Length]

|

[Display: "\_ \_ \_ \_ \_" + Warrior HP + Warlord HP + ASCII Art]

|

[Input: Guess a Letter]

|-----------------|

Yes No

[Correct?] [Wrong]

| |

[Warlord HP -1] [Warrior HP -1]

[Update Word] [Update Message]

| |

[Word Guessed?] [Warrior HP = 0?]

Yes No Yes No

| | | |

[Win: Warlord [Lose: Warrior [Loop]

Defeated] Falls]

| |

[End]-------------[End]

# METHODOLOGY:

The Word Warrior game was developed using a structured software engineering approach tailored for Assembly language programming

**Requirements Analysis:**

* Identified core gameplay requirements: word selection, guessing mechanism, health/scoring system
* Defined visual elements: ASCII art for game states, UI layout for status information
* Outlined player interaction flow from game start to victory/defeat conditions

**Design Phase**

* Created memory structure for word categories and game state tracking
* Designed procedure architecture for modular code organization
* Planned user interface layout and information hierarchy
* Mapped game flow states and transitions between them

**Implementation Strategy**

* Built foundation procedures first: word selection, letter checking, game state verification
* Implemented core game loop with input validation
* Added ASCII art and visual elements for enhanced user experience
* Developed scaffolding for game reset and replay functionality

**Testing Methodology**

* Unit testing of individual procedures (wordSelect, checkLetter, etc.)
* Integration testing of procedure interactions
* End-to-end gameplay testing across different difficulty levels
* Edge case testing (invalid inputs, boundary conditions)

# Experimentation:

# We conducted extensive experimentation with the Word Warrior game to enhance player engagement and create a more immersive experience. The most significant experimental addition was the implementation of sound effects using the PC speaker's beep functionality, which provides audio feedback for different game events.

We also experimented with thematic enhancements, introducing a compelling battlefield aesthetic that permeates throughout the game. The visual design now incorporates battlefield elements in the ASCII art, with different stages of combat represented through evolving text-based illustrations.

Animation was another key area of experimentation. We developed a series of ASCII animations that trigger at crucial moments in gameplay. When players land a successful guess, a brief "strike" animation plays, showing the warrior attacking the warlord.

The user interface underwent significant refinement during our experimental phase. We implemented a more intuitive layout that clearly separates the gameplay area, status information, and word display.

We also tested difficulty balancing through multiple playtesting sessions, making adjustments to ensure the game presents an appropriate challenge across all difficulty levels.

# Future Enhancements:

Looking toward future enhancements, we've identified several promising avenues for expansion. A more comprehensive audio system could replace the basic beeps with fully realized music tracks that change dynamically based on game tension—peaceful melodies during word selection that intensify during wrong guesses and culminate in either triumphant or somber conclusions. We can also add 2 players mode etc. We're exploring ways to implement this within the constraints of assembly language programming.

# Understanding of Mechanisms:

"Word Warrior" turns guessing words into a battle. Here’s how it works:

* **Start Up**: You pick a theme (say, Weapons) and how hard you want it. The game grabs a random five-letter word, like "SWORD." You get 6, 4, or 2 lives based on difficulty; the warlord gets 5 lives (one per letter).
* **Play the Game**: The screen shows the word as blanks (\_ \_ \_ \_ \_), your lives, the warlord’s lives, your score, guessed letters, and a text picture of the warlord. You type a letter.
* **Right Guess**: If ‘S’ is in "SWORD," the screen updates to S \_ \_ \_ \_, the warlord loses a life (5 to 4), you get 10 points, and it says, "You strike!" His picture changes in ASCII format.
* **Wrong Guess**: If the letter’s not in the word, you lose a life (say, 6 to 5), lose 5 points, and see, "The warlord hits back!" The warlord’s picture might look smug.
* **Win or Lose**: You win if you guess the word (warlord’s lives hit 0), getting bonus points for lives left. You lose if your lives run out. Either way, you see the word and a big win or lose picture.
* **Play Again**: You can start over with a new word and theme, and everything resets so it feels fresh.

Platform:  
The game runs on a Windows computer and uses:

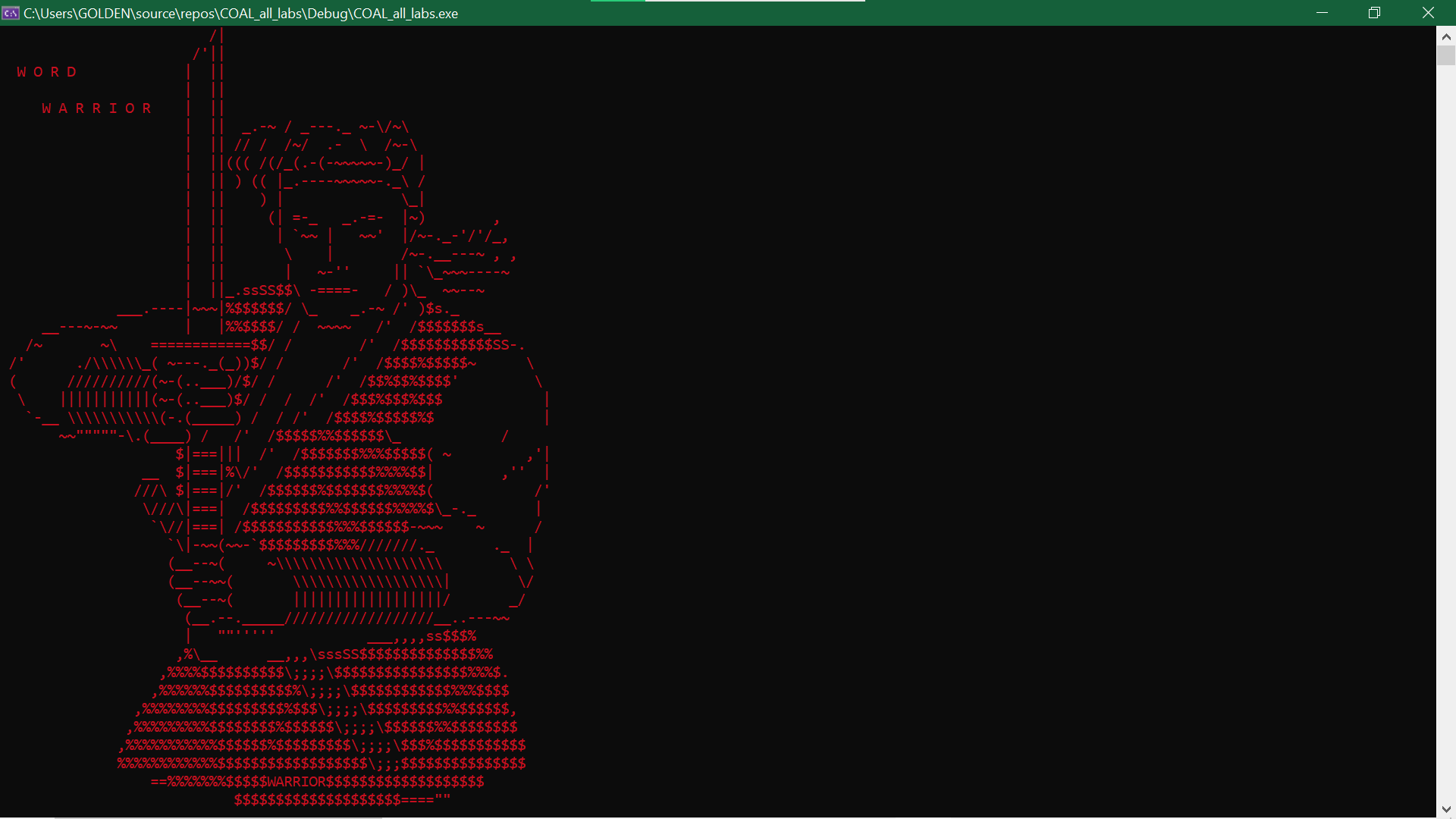
* **Coding Style**: Assembly language, a way to give the computer exact instructions.
* **Helper Tools**: A library called Irvine32 that makes it easy to show text, get guesses, move the cursor, pick random words, and add pauses for effect.
* **Setup**: Built in Visual Studio or another program that works with assembly, using Irvine32.

# Challenges:

Making "Word Warrior" won’t be easy. Here’s what might be tricky:

* **Text Art Updates**: Changing the warlord’s picture in ASCII as lives drop means carefully placing text so it doesn’t jump around or mess up the screen.
* **Tracking Lives and Points**: Keeping up with your lives, the warlord’s lives, and the score needs careful coding to make sure everything matches the guesses.
* **Guessing Right**: The game has to check if letters are valid (A-Z), turn small letters into capitals, and remember what’s been guessed without mistakes.
* **Random Words**: Picking a word fairly from a list takes some math to jump to the right spot in memory.
* **Starting Over**: Resetting the game to clear old guesses and lives is tough because you have to make sure nothing from the last game sticks around.  
  These are hard parts, but with the right tools and planning, I can make it work smoothly.

# Screenshots of Gameplay:



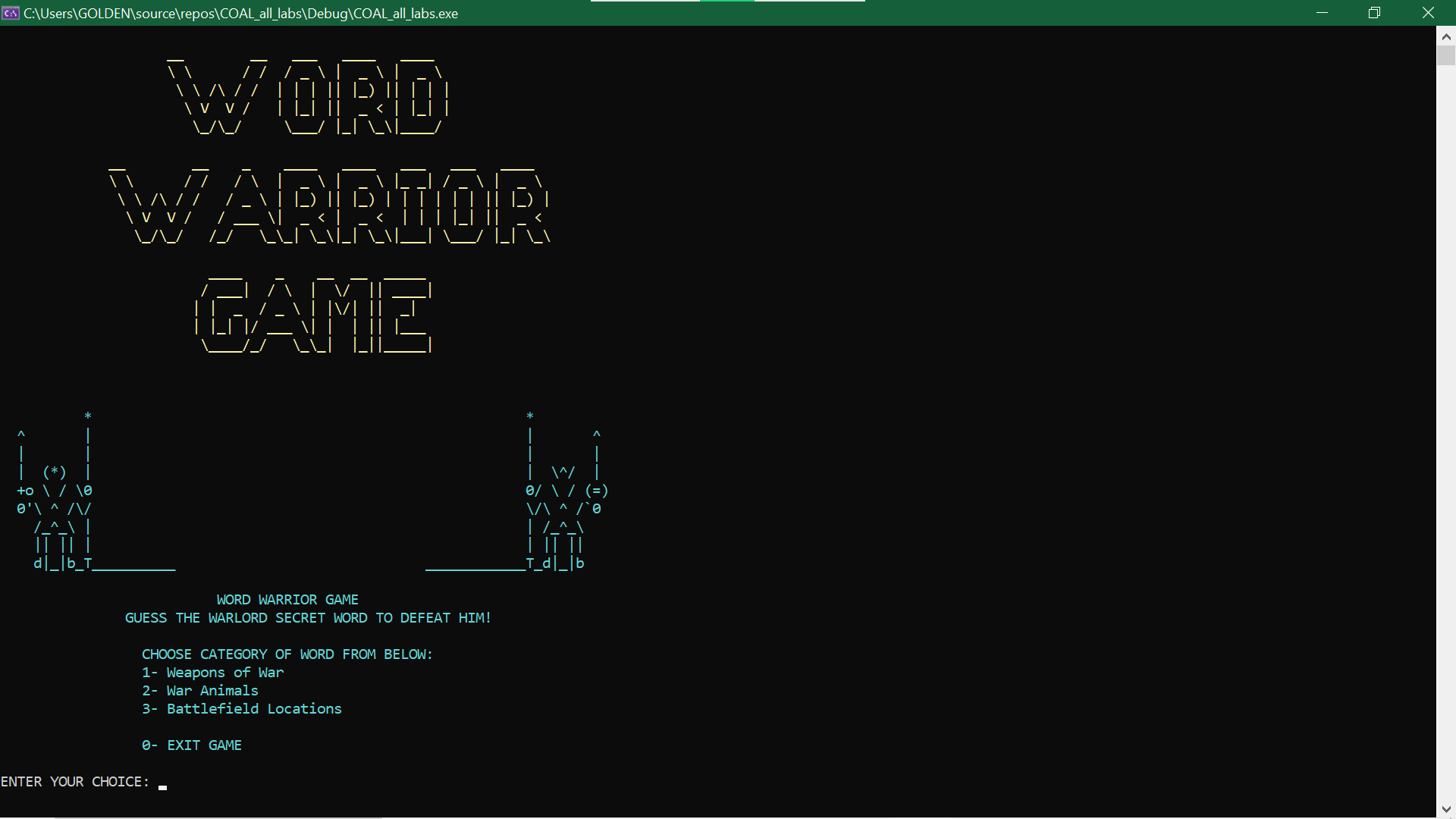
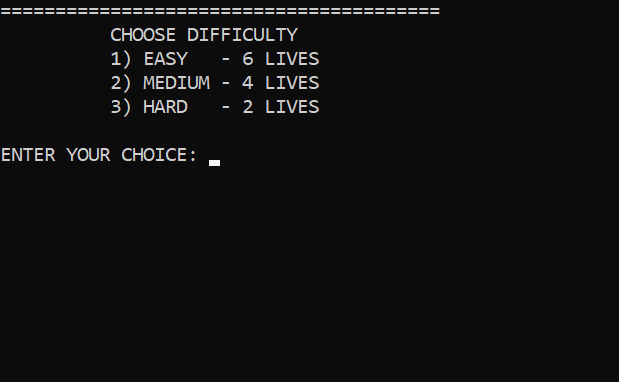
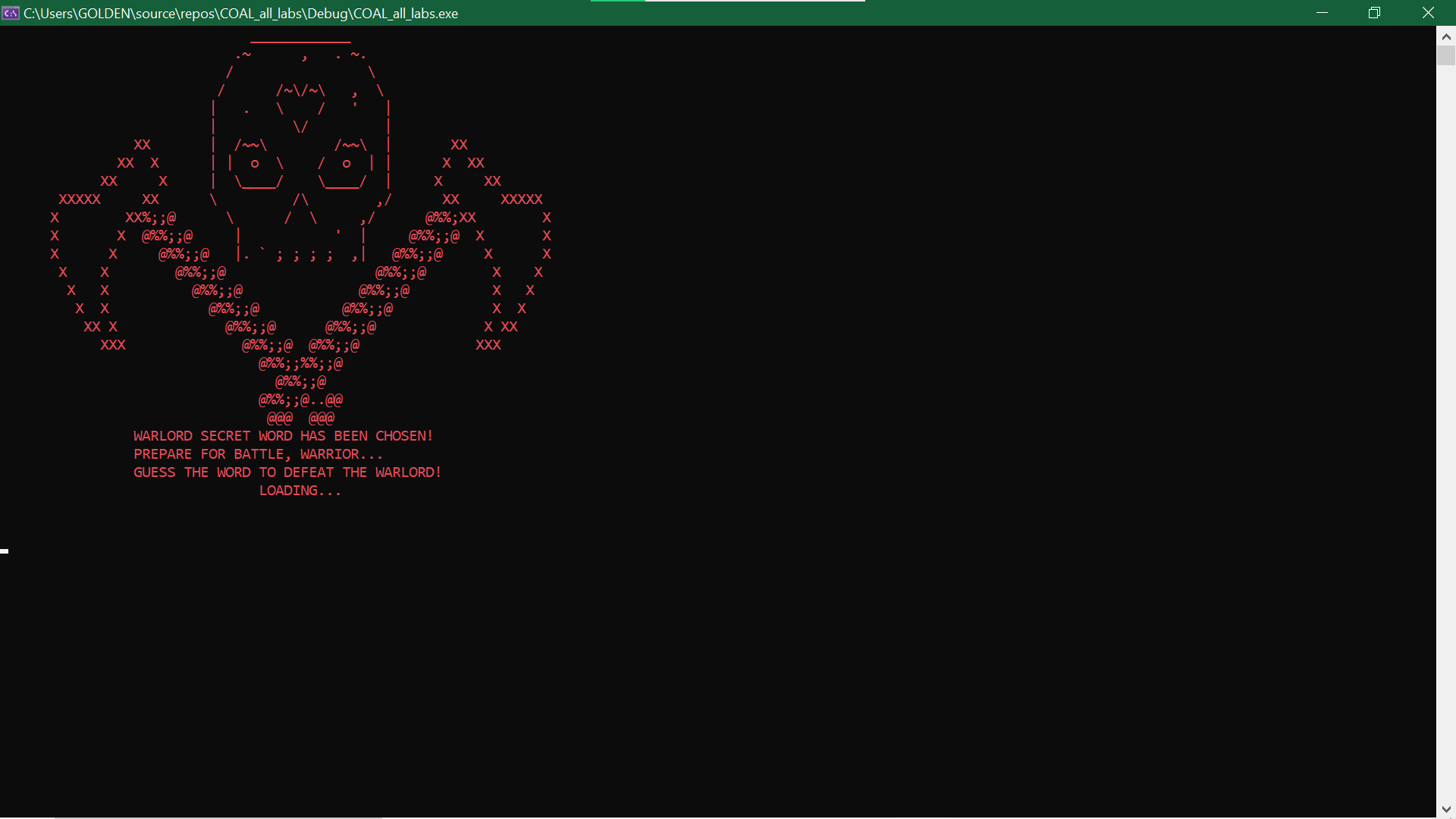
Figure 1: LOADING GAME

Figure 2: Main MENU





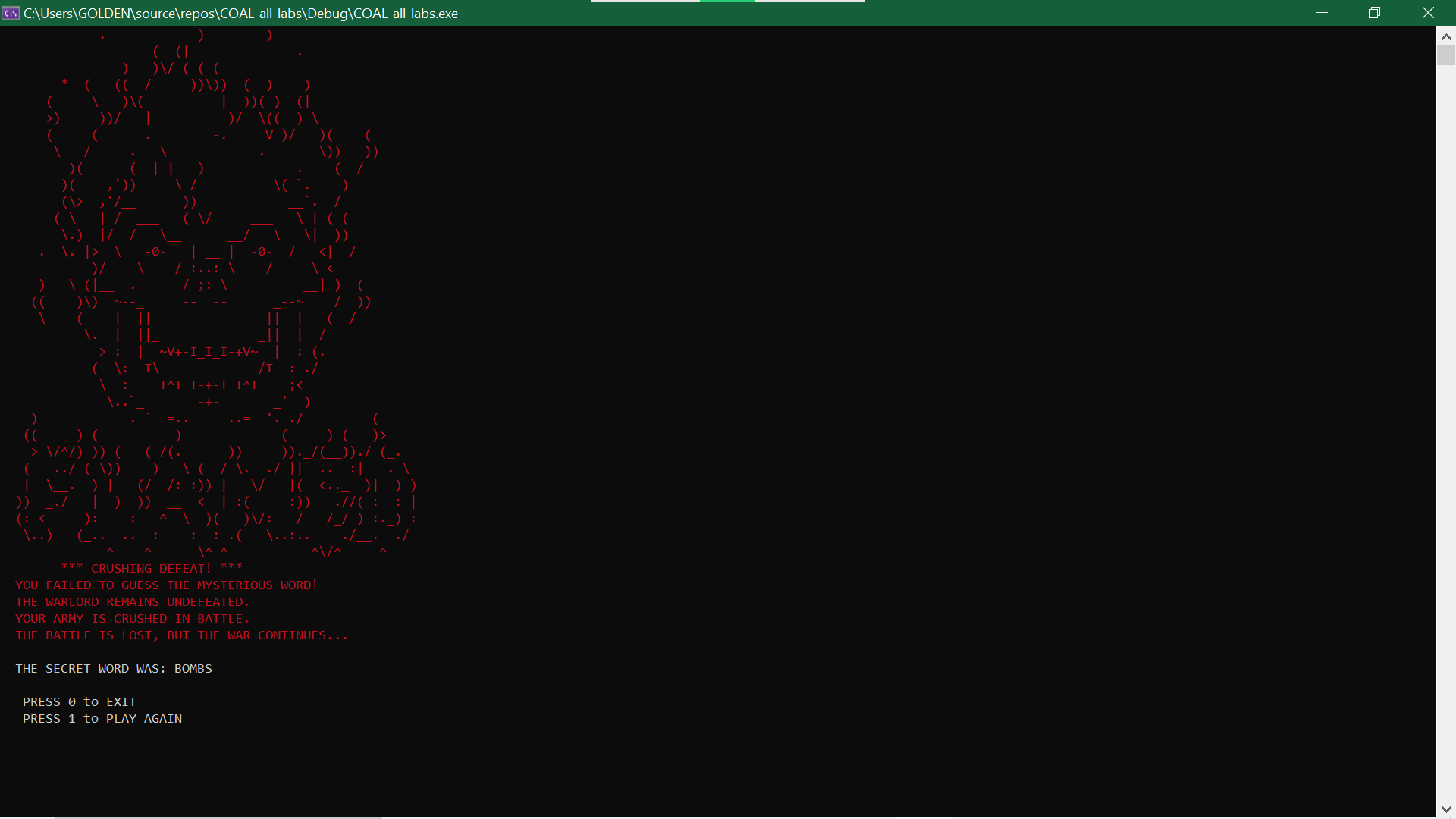


Figure 3: Loose Game

# Use Cases:

**Educational Tool for Assembly Programming**

Actors: Computer Science Students, Programming Instructors

***Scenario:***

Instructor demonstrates Word Warrior as an example of practical assembly programming

Students examine source code to understand key assembly concepts

Students identify and analyze specific programming techniques:

**Benefits:**

Provides concrete example of assembly programming principles

Demonstrates practical application of low-level programming concepts

Creates engaging context for learning otherwise abstract assembly techniques

**Programming Challenge Extension**

Actors: Student Programmers, Competition Participants

**Scenario:**

Participants receive base Word Warrior code as starting point

Challenge requires extending game with new features

**Benefits:**

Teaches code modification and extension skills

Encourages creative problem-solving within constraints

Provides scaffolding for learning assembly language

**Recreational Gaming**

Actors: Casual Players, Retro Gaming Enthusiasts

**Scenario:**

Player launches Word Warrior for entertainment

Player selects preferred category and difficulty level

Player engages with the battle narrative through multiple game sessions

Player experiences progression through increasingly difficult settings

**Benefits:**

Provides entertainment value beyond educational purpose

Creates nostalgic experience for text-based gaming enthusiasts

Demonstrates that assembly language can create engaging experiences

**Word Learning Tool**

Actors: Language Learners, Vocabulary Students

**Scenario:**

Instructor creates custom word lists for specific learning domains

Students play Word Warrior with vocabulary targeted to their learning needs

Game reinforces word recognition through repeated exposure

Battle narrative creates memorable context for word retention

**Benefits:**

Demonstrates that assembly language can create engaging experiences

Encourages creative problem-solving within constraints

# Conclusion:

"Word Warrior" takes the old Hangman idea and makes it a thrilling fight where you battle a warlord by guessing words. With choices like Weapons or Animals, easy or hard modes, lives to track, points to earn, and text art that changes as you play, it’s a game you’ll want to keep playing. Built with assembly coding and Irvine32 tools, it shows how even a tricky language can create something fun and lively. Sure, there are challenges like getting the art right or resetting the game, but they’re worth tackling. "Word Warrior" is a blast to play and a great way to learn coding, with room to grow—like adding sounds or more words later. It’s proof you can make awesome things with just text and a bit of imagination.