International University of Sarajevo

Faculty of Engineering and Natural Sciences

CS305 Programming Languages

Text Based Escape Room Project Report

Prepared by:

Fatih Bahadır Karakuş
Student ID: 220302370

Course Instructor:

Mirza Selimovic

Sarajevo, 2025

Contents

1 Introduction

The purpose of this project is to design and implement a text-based escape game using the C# programming language. The game places the player inside a mysterious device and challenges them to escape by navigating through multiple levels, solving puzzles, interacting with enemies, and managing limited resources such as health and oxygen.

The main problem this project aims to solve is the difficulty of applying abstract programming language concepts in a meaningful and practical way. While concepts such as loops, conditional statements, functions, and regular expressions are often taught theoretically, students may struggle to understand how these constructs work together in a complete software system. This project addresses that problem by embedding these concepts into an interactive, real-time application where every player action directly triggers program logic.

The project is particularly relevant to the CS305 Programming Languages course because it demonstrates how core language features are used in a structured and realistic scenario. The game relies heavily on control flow to manage player movement, collision detection, puzzle validation, and game progression. Loops are used to continuously update the game state, while functions are employed to ensure modularity and code readability. Additionally, regular expressions are used in puzzle challenges to perform pattern matching and input validation, reflecting real-world programming use cases.

Another important aspect of the project is state management. The game maintains and updates multiple state variables such as player position, health, oxygen level, inventory, score, and checkpoint data. Managing these states correctly is essential to ensure consistent gameplay, especially when handling events such as player death and respawning.

The expected learning outcomes of this project include:

- Understanding and applying control flow structures in an interactive environment
- Using loops to manage continuous execution and real-time updates
- Designing and organizing programs using functions and modular logic
- Applying regular expressions for pattern matching and input validation
- Managing application state across multiple game levels
- Improving problem-solving and debugging skills through iterative development

Through this project, students gain hands-on experience in transforming theoretical programming knowledge into a functional software system.

2 Project Overview

The project is a Windows Forms-based game titled **Text Based Escape Device - Ultimate Edition**. Although the visual style resembles a traditional console game, all rendering and user interaction are handled within a Windows Forms interface.

The player wakes up inside a strange mechanical device and must find a way to escape by progressing through multiple levels. Each level introduces new challenges, puzzles, and enemies, increasing the overall difficulty and complexity of gameplay.

The game is designed to be interactive and reactive, meaning that every player input immediately affects the game state. Movement, interactions, puzzle solving, and combat are all handled dynamically within a continuous game loop.

3 Summary of Features

The main features of the project include:

- Multi-level gameplay with progressive difficulty (4 challenging levels)
- Real-time player movement using keyboard input (WASD controls)
- Puzzle-solving mechanics based on logic and regular expressions
- Multiple enemy types: Melee (M), Range (R), and Boss (B)
- Health and oxygen resource management
- Weapon system with shooting mechanics
- Power-ups: Shield, Speed Boost, Invisibility, Health Packs, Oxygen Tanks
- Inventory and achievement tracking system
- Checkpoint-based respawn mechanism
- Customizable visual themes (6 different color themes)
- Three difficulty levels: Easy, Normal, Hard

4 Scope of the Project

The scope of this project is limited to a console-style application focusing on programming logic rather than graphical complexity. While the game provides a visually structured ASCII-based interface, advanced graphical rendering, networking features, and persistent save files are beyond the scope of this implementation.

The project focuses primarily on demonstrating programming language concepts rather than game engine development.

5 Inputs and Outputs

User inputs are provided through keyboard commands. Movement is controlled using the W, A, S, and D keys, while additional actions such as interaction (E), inventory access (I), shooting (F), pausing (P), theme change (T), and quitting (Q) are mapped to specific keys.

Program outputs are displayed directly in a Windows Forms window using ASCII graphics, colored text, status bars, and system messages. Visual feedback is provided through color-coded elements to enhance user awareness of game events.

6 User Interaction

User interaction plays a central role in the game. The player must constantly make decisions based on available resources, enemy positions, and puzzle requirements. Clear messages and visual indicators guide the user through the gameplay experience and provide feedback for both correct and incorrect actions.

The game features:

- Real-time movement and collision detection
- Interactive elements: doors, keys, puzzles, items
- Combat system with melee and ranged enemies
- Boss battle with 10 HP tracking
- Achievement system tracking player accomplishments

7 Limitations or Constraints

Although the project successfully implements the intended gameplay mechanics, it has several limitations and constraints due to the scope of the course and the technologies used.

First, the game is implemented as a Windows Forms application with a grid-based, console-style visual design. While this approach effectively demonstrates programming logic and game mechanics, the graphical rendering is intentionally kept simple and does not include advanced animations or graphical effects.

Second, the project is designed primarily for the Windows operating system. Certain functionalities, such as keyboard input handling and window-based rendering, rely on Windows-specific behavior, which may limit portability to other platforms.

Another limitation is the checkpoint system. Checkpoints are stored in memory during runtime and are not written to external files. As a result, if the program is closed, all progress is lost and the game must be restarted from the beginning.

Additionally, enemy behavior and artificial intelligence are intentionally kept simple. Enemies follow basic movement rules and collision-based interactions without advanced pathfinding algorithms. This design choice was made to keep the focus on core programming language concepts rather than complex AI implementations.

Finally, the size and complexity of levels are constrained by the grid-based layout and window dimensions. Large maps or excessive visual elements may affect alignment and readability depending on the application window size and display resolution.

8 System Requirements

This section describes the software environment required to build, run, and test the project successfully.

8.1 Programming Language

The project is implemented using the C# programming language. C# was chosen because it provides strong support for structured programming, object-oriented design, and rich standard libraries suitable for developing interactive applications.

8.2 Frameworks and Libraries

The project uses the .NET 9.0 framework. The main libraries used include:

- **System:** Core classes and base data types
- **System.Threading:** Managing delays and timing effects during gameplay
- **System.Text.RegularExpressions:** Implementing regex-based puzzle challenges
- **System.Runtime.InteropServices:** Console window management
- **System.Drawing:** Graphics operations
- **System.Windows.Forms:** GUI framework

8.3 IDE and Tools Used

The project was developed using Microsoft Visual Studio with integrated debugging tools, IntelliSense, and project management features.

8.4 Operating System

The project is designed to run on Windows operating system.

8.5 Technologies Used

8.5.1 Programming Language Features

Functions: Used extensively to modularize code. Each major functionality is implemented as a separate method.

Loops: A continuous while loop serves as the main game loop. Additional for and foreach loops iterate over maps, enemies, and collections.

Conditional Statements: if-else and switch statements control game logic, movement validation, collision outcomes, and puzzle success/failure.

Regular Expressions: Used in puzzle challenges for pattern matching and input validation.

8.5.2 Libraries or Modules

All required libraries are part of the standard .NET framework. No external packages were used.

8.5.3 External Packages

No external or third-party packages were used in this project.

8.6 Program Structure and Logic

8.6.1 Major Components

The program is divided into logical components:

- **Main Program and Game Loop**
- **Game State Management** (GameState.cs)
- **Level and Map Handling** (LevelData.cs)
- **Player Interaction and Movement**
- **Enemy and Boss Logic** (Enemy.cs, Boss.cs)
- **Puzzle System**
- **Checkpoint System**
- **Theme System** (ThemeColors.cs)

8.6.2 Flowchart and Pseudocode

Main program flow:

```
START
    Show Intro
    Select Difficulty (Easy/Normal/Hard)
    Select Theme
    Load Level 1
    Save Initial Checkpoint

    WHILE game is running
        Draw Game Screen
        Read User Input
        Process Movement or Interaction
        Move Enemies and Boss
        Check Collisions
        Check Win/Loss Conditions
    END WHILE

    Show Ending Screen
END
```

Listing 1: Main Game Flow Pseudocode

8.6.3 Data Structures Used

- **Arrays:** Two-dimensional character arrays (char[,]) for game maps
- **Lists:** Dynamic collections for enemies and achievements
- **Objects:** Enemy, Boss, GameState classes
- **Primitive Variables:** Integers and booleans for game state tracking

8.6.4 Explanation of Key Algorithms

Enemy Movement Algorithm:

Melee enemies use random movement:

```
public void Move(char[,] map)
{
    MoveCounter++;
    if (MoveCounter < 3) return;
    MoveCounter = 0;

    int[] dirs = { -1, 1, 0, 0 };
    int[] dcols = { 0, 0, -1, 1 };

    for (int i = 0; i < 10; i++)
    {
        int idx = rand.Next(4);
        int newR = Row + dirs[idx];
        int newC = Col + dcols[idx];

        if (newR > 0 && newR < map.GetLength(0) - 1 &&
            newC > 0 && newC < map.GetLength(1) - 1)
        {
            if (map[newR, newC] == ' ')
            {
                Row = newR;
                Col = newC;
                break;
            }
        }
    }
}
```

Listing 2: Enemy.cs - Random Movement Algorithm

Boss Tracking Algorithm:

The boss tracks the player's position:

```
public void MoveTowardsPlayer(char[,] map, int pRow, int pCol)
{
    MoveCounter++;
    if (MoveCounter < 2) return;
    MoveCounter = 0;

    int dr = 0, dc = 0;
    if (pRow < Row) dr = -1;
    else if (pRow > Row) dr = 1;
    if (pCol < Col) dc = -1;
    else if (pCol > Col) dc = 1;

    int newR = Row + dr;
    int newC = Col + dc;

    if (newR == pRow && newC == pCol) return;

    if (newR > 0 && newR < map.GetLength(0) - 1 &&
        newC > 0 && newC < map.GetLength(1) - 1)
    {
        if (map[newR, newC] != '#')
        {
            Row = newR;
            Col = newC;
        }
    }
}
```

Listing 3: Boss.cs - Player Tracking Algorithm

8.6.5 Important Code Decisions and Reasoning

- Console-style interface was chosen to focus on programming logic
- Centralized GameState class simplifies data sharing
- Regex puzzles directly reflect course content
- Checkpoint system improves user experience
- Enemy behavior kept simple to maintain focus on core concepts

8.7 User Interface – Console Interaction

8.7.1 Menu Structure

The game includes:

- Introduction screen
- Difficulty selection menu
- Theme selection menu
- Pause menu
- Inventory screen

8.7.2 Command Inputs

- **W/A/S/D:** Movement
- **E:** Interact
- **I:** Inventory
- **P:** Pause
- **T:** Change theme
- **F:** Shoot (requires weapon)
- **Q:** Quit

8.7.3 Error Handling

- Invalid movements are blocked
- Warning messages for incomplete exit conditions
- Health penalties for incorrect puzzle inputs
- Confirmation dialogs for critical actions

8.7.4 Example User Session

1. Launch game, view introduction
2. Select difficulty and theme
3. Navigate Level 1, collect items
4. Solve puzzle, open door
5. Progress through levels 2-4
6. Defeat boss in Level 4
7. View victory screen

9 Team Member Responsibilities

The project was developed individually by Fatih Bahadır Karakuş.

9.1 Fatih Bahadır Karakuş

9.1.1 Assigned Tasks

- Designing overall architecture and game flow
- Implementing all game mechanics and systems
- Creating both Console and GUI modes
- Developing all algorithms and logic
- Testing and debugging
- Writing comprehensive documentation

9.1.2 Specific Features Implemented

- Grid-based player movement system
- Health and oxygen resource management
- Multi-level progression (4 levels)
- Enemy AI systems (Melee, Range, Boss)
- Puzzle mechanics with regex validation
- Weapon and shooting system
- Power-up collection and effects
- Checkpoint save/restore mechanism
- Achievement tracking system

- Theme customization (6 themes)
- Difficulty level variations
- Inventory and status display

9.1.3 Testing Responsibilities

- Testing movement and collision detection
- Verifying exit conditions and level progression
- Testing checkpoint restoration
- Debugging oxygen and health systems
- Testing enemy behavior and boss AI
- Verifying puzzle validation logic
- Playtesting all difficulty levels
- Edge case testing

9.1.4 Documentation Work

- Writing project report in LaTeX
- Creating detailed code explanations
- Documenting algorithms and data structures
- Preparing system architecture documentation
- Creating flowcharts and pseudocode
- Organizing GitHub repository
- Writing README.md

10 Challenges and Solutions

10.1 Technical Issues

One main technical issue involved managing real-time updates within a Windows Forms environment while keeping gameplay responsive. The game loop was structured to separate rendering, input processing, and game logic updates.

Another challenge was handling keyboard input reliably. Input validation was implemented at multiple levels with state checks.

10.2 Logical Errors

Several logical errors emerged regarding game progression and state consistency. For example, the exit could initially be reached without completing required objectives.

Solution: Explicit condition checks were introduced that enforce level completion rules in the correct order.

10.3 Implementation Challenges

The checkpoint system was one of the most significant implementation challenges. It required saving multiple aspects of game state simultaneously.

Solution: All relevant state variables were stored in a centralized GameState class, allowing atomic save and restore operations.

Enemy reinitialization after checkpoint reload also presented challenges.

Solution: Enemy lists were cleared and repopulated from level data during checkpoint restoration.

10.4 How the Challenges Were Solved

Challenges were addressed through:

- Modular design separating concerns
- Incremental testing after each feature
- Careful state management using GameState class
- Clear validation rules enforcing execution order
- Atomic checkpoint save/restore operations
- Iterative debugging and refinement

11 Conclusion

This project successfully demonstrates the design and implementation of **Escape Device - Ultimate Edition** using C# and Windows Forms. The main objective was to apply core programming language concepts in a practical manner, achieved through a fully playable, multi-level game.

Key gameplay mechanics such as player movement, enemy interaction, puzzle solving, level progression, and checkpoint-based recovery were successfully implemented. The game reacts dynamically to user input and enforces logical conditions for progression.

From an educational perspective, the project contributed significantly to programming skills development. Core concepts such as control statements, loops, functions, and conditional logic were used extensively. Regular expressions were applied in puzzle challenges for pattern matching and validation. Problem-solving skills were strengthened through debugging, state management, and handling complex interactions.

The original objectives were successfully met. The final implementation integrates programming language concepts into a functional system. Additional features such

as achievements, visual themes, multiple difficulty levels, and checkpoints enhanced overall quality.

Future improvements could include:

- Persistent checkpoints using file-based storage
- Advanced enemy AI with pathfinding algorithms
- More complex level designs
- Sound effects and background music
- Multiplayer functionality
- Level editor for user-generated content
- Enhanced graphics with animations
- More puzzle types and challenges

This project provided valuable opportunity to apply theoretical programming knowledge in a practical context, reinforcing technical and analytical skills essential for software development.

12 References

Tutorials and Learning Materials

- Microsoft Learn. C# Programming Guide. Available at: https://learn.microsoft.com/en-us/dotnet/csharp/
- Windows Forms Programming Guide. Available at: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/

Documentation Pages

- Microsoft Docs. System.Text.RegularExpressions Namespace. Available at: https://learn.microsoft.com/en-us/dotnet/api/system.text.regularexpressions
- Microsoft Docs. Console and Keyboard Input Handling. Available at: https://learn.microsoft.com/en-us/dotnet/api/system.console

Books

- Albahari, J. and Albahari, B. *C# in a Nutshell*. O'Reilly Media.
- Freeman, E. and Robson, E. *Head First C#*. O'Reilly Media.

Code Usage and Attribution

All source code used in this project was implemented by the project team. No external code snippets were copied directly from online sources. Online documentation and tutorials were used solely for reference and understanding of programming concepts.

A Project Repository

The complete source code for the project **Escape Device - Ultimate Edition** is hosted on GitHub:

https://github.com/bahadirkarakus/Text-Based-Escape-Room

The repository includes:

- Full C# source code
- Windows Forms project structure
- Game logic implementation
- Documentation and README files

B Development Process and Academic Integrity Statement

B.1 Independent Development Process

The project was developed through an independent and iterative development process. All core gameplay mechanics, system logic, and design decisions were implemented manually without relying on automated code generation tools.

B.2 Use of External Resources

External resources such as official documentation and programming tutorials were consulted solely for reference and learning purposes. These resources were used to understand language syntax, framework behavior, and best practices, rather than to copy or reuse complete solutions.

B.3 Academic Integrity Assurance

The student affirms that:

- All source code was written by the student
- No third-party code was copied directly
- All implementation decisions were made based on student's understanding
- AI tools were used only for improving documentation clarity and LaTeX formatting
- No AI-generated code was used in implementation

B.4 Final Compliance Statement

All content presented in this report reflects the student's original work and understanding. The project fully complies with the academic integrity guidelines and AI usage policies defined by the CS305 Programming Languages course.