North South University

Project Title - A 2D Horror Game Using Java
(Escape From The Ghost)

Course Name: Object Oriented Programming (CSE215 Lab)

Section: 07

Instructor's Name: Arfana Rahman

Group Members (Full Names and IDs):

Mazba Uddin Saif	2522638642
Saad Un Maznun	2522674642
Ariyan Siddique Mahi	2523146642
Afrin Zaman	2523289642

Submission Date: 15/12/2025

1. Abstract

This project is a 2D horror survival game developed using **Java and Object-Oriented Programming (OOP) principles**. The main objective of the game is to allow the player to navigate through a maze, collect coins and a key, avoid a hostile ghost, and successfully escape through the exit door. The game features real-time movement, sound effects, score calculation, game history storage, and a graphical user interface built with Java Swing.

The project demonstrates the practical application of core OOP concepts such as **encapsulation, inheritance, abstraction, polymorphism, and composition**, along with file handling, event-driven programming, and basic game loop design. The result is a fully functional interactive game that showcases both programming skills and software design principles.

2. Introduction

2.1 Objective of the Project

The primary objective of this project is to design and implement a **fully functional Javabased game** that demonstrates strong usage of **Object-Oriented Programming concepts**. The project aims to:

- Apply OOP concepts in a real-world scenario
- Develop a graphical game using Java Swing
- Implement file handling for persistent data storage
- Create an interactive user experience with sound and visuals

2.2 Background and Significance

Object-Oriented Programming is one of the most important paradigms in modern software development. Understanding how OOP concepts work together in a large project is essential for building scalable and maintainable applications.

This project is significant because it:

- Moves beyond simple console programs
- Demonstrates real-time interaction and game logic
- Simulates real-world software architecture
- Provides hands-on experience with Java GUI and event hand

3. Features and Implementation

This section describes the major features of **The 2D Horror Game**, along with how each feature is implemented using Java and object-oriented programming.

3.1 Game State Management System

Feature Description

The game supports multiple states:

- Main Menu
- Playing
- Paused
- Game Over
- Escaped
- History View

This allows smooth transitions between gameplay, menus, and result screens.

Implementation

An enum named GameState is used to represent the current state of the game.

```
public enum GameState {

MENU,  PLAYING,  PAUSED,  GAME_OVER,  ESCAPED,

HISTORY    }
```

The GamePanel class stores and controls the current state:

```
private GameState gameState = GameState.MENU;

Rendering and logic execution depend on the

state:    if (gameState== GameState.PLAYING) {

updateGame();

}
```

Different screens are drawn using a switch statement:

```
switch (gameState) {

case MENU: drawMenuScreen(g2, overlayColor);

break;

case GAME_OVER: drawGameOverScreen(g2);

 break;

case ESCAPED: drawEscapedScreen(g2);

break;

}
```

This ensures **clean separation of logic** and prevents unnecessary updates.

3.2 Tile-Based Map System

Feature Description

The game world is represented as a grid-based maze containing:

- Walls
- Floor tiles
- Key
-

spawn

Exit door

Implementation

A 2D integer array defines the map layout:

```
final int[][] map = {

{1, 1, 1, 1, 1, ...},

{1, 0, 0, 0, 0,

...},     {1, 0, 1,

1, 1, ...}

};
```

Tile values represent:

- 0 → Floor
- 1 → Wall
- 4 → Key spawn
- 5 → Exit door**Rendering is done tile-by-tile:**

```
if (map[y][x] .= 1) {

g2.fillRect(xPos, yPos, tileSize,

tileSize);

}
```

This approach allows easy modification and scalability.

3.3 Player Movement and Controls

Feature Description

The player can move using:

- WASD keys
- Arrow keys
- Collision with walls

Implementation

Input handling is managed inside the Player

class: `public void keyPressed(int`

```
key) {     if (key ==

KeyEvent.VK_W) up = true;

}
```

Movement calculation is done in the update() method:

```
if (up) dy -

=speed; if

(down) dy +=

speed;
```

Diagonal movement normalization:

```
if (dx == 0 =& dy == 0)
{    dx=(int) (dx /
Math.sqrt(2));

dy = (int) (dy /
Math.sqrt(2)); }
```

Collision detection is handled by the parent Entity class. Ensures realistic movement and prevents wall clipping.

3.4 Ghost (Enemy Behavior)

Feature Description

The Ghost:

- Continuously chases the player
- Passes through walls
- Ends the game upon collision

Implementation

The monster calculates direction based on the player’s position:

```
int deltaX =targetX - monsterCenterX;

int deltaY = targetY -
monsterCenterY; Movement is
normalized and scaled by speed:    dx
= (int) (speed * (deltaX /
distance));
```

The monster overrides collision logic:

```
@Override protected boolean

isColliding(==.) {

return false;

}
```

This creates a **ghost-like horror effect**.

3.5 Item Collection System (Coins & Key)

Feature Description

- Coins increase score
- Key unlocks the exit door

Implementation

Items are represented using the Item class:

public enum Type { KEY, COIN }

Collision detection:

```
if (player.getBounds().intersects(item.getBounds())) {

player.collectItem(item);

}
```

Score handling:

```
score += Item.COIN_POINTS;
```

Simple and efficient item management.

3.6 Exit Door and Escape Mechanism

Feature Description

- Door remains locked until key is collected
- Player escapes using ENTER key

Implementation

Door interaction check:

```
exitDoor.canInteract =
player.getBounds().intersects(exitDoor.getBounds());
```

Escape logic:

```
if (player.hasKey) {

finalizeGame("Escaped");

}
```

3.7 Scoring and Game History System

Feature Description

- Score based on survival time + coins
- Game history saved to file
- Paginated history view

Implementation

Score calculation:

```
finalScore = survivalTime +

player.score; History is saved

using GameHistory:

gameHistory.saveHistory(newRecord);
```

Records are sorted by score:

```
historyList.sort((r1, r2) -> Long.compare(r2.finalScore,
r1.finalScore));
```

Persistent storage improves replay value.

3.8 Audio System

Feature Description

- Background music
- Sound effects for actions

Implementation

Sounds are managed by AudioPlayer:

audioPlayer.playSound(SoundType.COIN); audioPlayer.startBGM();

4. Implementation of OOP Concepts

4.1 Encapsulation

Encapsulation is achieved by:

- Grouping data and methods inside classes Controlling
- access using access modifiers

Example:

```
private boolean isLocked;

public boolean

canInteract;
```

Protects internal state and improves maintainability.

4.2 Inheritance

Inheritance reduces code duplication and enables hierarchy.

public abstract class Entity extends GameObject

public class Player extends Entity public class

Ghost extends Entity

Shared movement and collision logic reused efficiently.

4.3 Abstraction

Abstract classes define common behavior without implementation details.

```
public abstract class GameObject {

public abstract void draw(Graphics2D

g2);

} public abstract class Entity {     public abstract

void update(int[][] map, int tileSize);
}
```

4.4 Polymorphism

Polymorphism allows different behaviors using the same method signature.

Example:

```
entity.update(map, tileSize);
```

- Player updates
- Ghost updates

Method overriding: `@Override`

```
protected boolean

isColliding(...) {     return

false;

}
```

Enables flexible and extensible design.

4.5 Composition / Aggregation

Objects are composed inside GamePanel:

```
private Player

player; private Ghost

ghost; private Door

exitDoor;
```

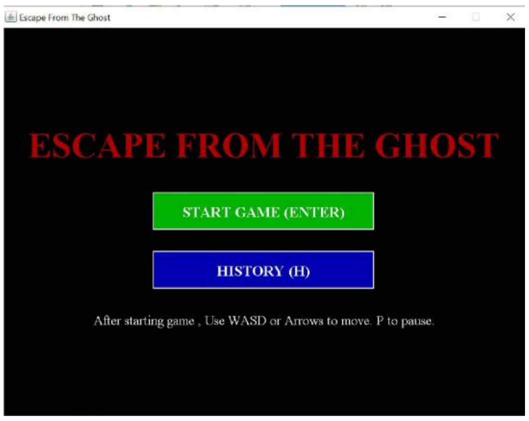
5. Member Contribution

Feature	Contribution (Name & ID)
Entity & Collision Logic	Saad Un Maznun – 2522674642
GUI (Screens, Buttons , Objects)	Mazba Uddin Saif 2522638642 Saad Un Maznun – 2522674642
Action event Management	Mazba Uddin Saif 2522638642
File and data process	Ariyan Siddique Mahi - 2523146642
Entity Package	Afrin Zaman 2523289642

6. Testing and Results

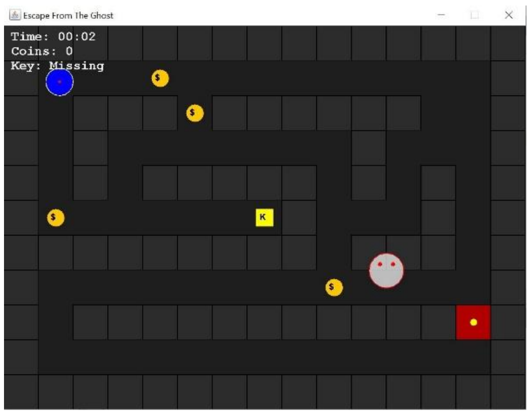
6.1 Test Cases

Test Case	Input	Expected Output
Player hits wall	Move into wall	Player stops
Collect coin	Player overlaps coin	Coin disappears, score increases
Collect key	Player overlaps key	Door unlocks
Monster collision	Monster touches player	Game Over
Escape	Player presses ENTER with key	Escaped screen

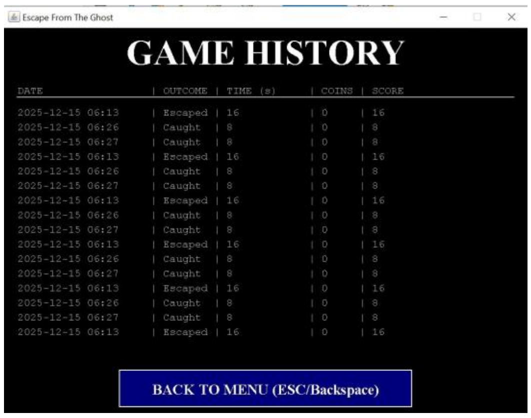


ESCAPE FROM THE GHOST

After starting game , Use WASD or Arrows to move. P to pause.

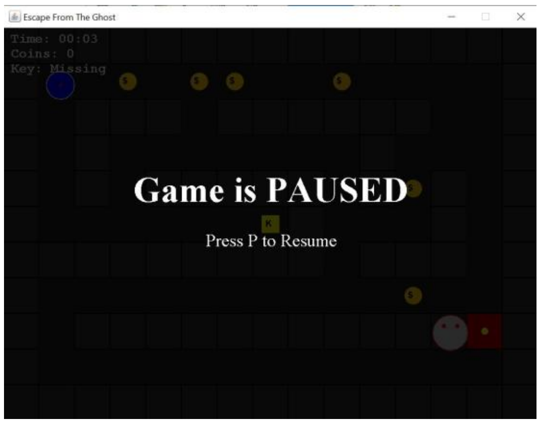


```
Time: 00:02
Coins: 0
Key: Missing
```



GAME HISTORY

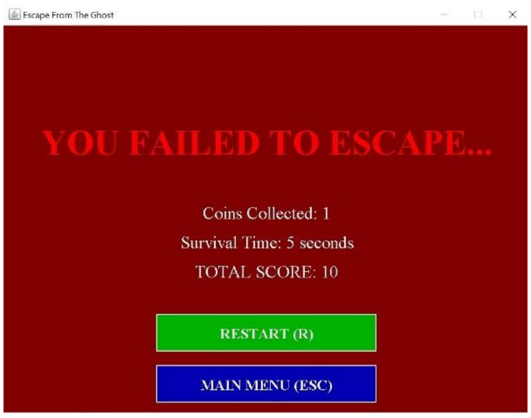
```
DATE                 | OUTCOME | TIME (s)       | COINS | SCORE
2025-12-15 06:13     | Escaped | 16             | 0     | 16
2025-12-15 06:26     | Caught  | 8              | 0     | 8
2025-12-15 06:27     | Caught  | 8              | 0     | 8
2025-12-15 06:13     | Escaped | 16             | 0     | 16
2025-12-15 06:26     | Caught  | 8              | 0     | 8
2025-12-15 06:27     | Caught  | 8              | 0     | 8
2025-12-15 06:13     | Escaped | 16             | 0     | 16
2025-12-15 06:26     | Caught  | 8              | 0     | 8
2025-12-15 06:27     | Caught  | 8              | 0     | 8
2025-12-15 06:13     | Escaped | 16             | 0     | 16
2025-12-15 06:26     | Caught  | 8              | 0     | 8
2025-12-15 06:27     | Caught  | 8              | 0     | 8
2025-12-15 06:13     | Escaped | 16             | 0     | 16
2025-12-15 06:26     | Caught  | 8              | 0     | 8
2025-12-15 06:27     | Caught  | 8              | 0     | 8
2025-12-15 06:13     | Escaped | 16             | 0     | 16
```



```
Time: 00:03
Coins: 0
Key: Missing
```

Game is PAUSED

Press P to Resume

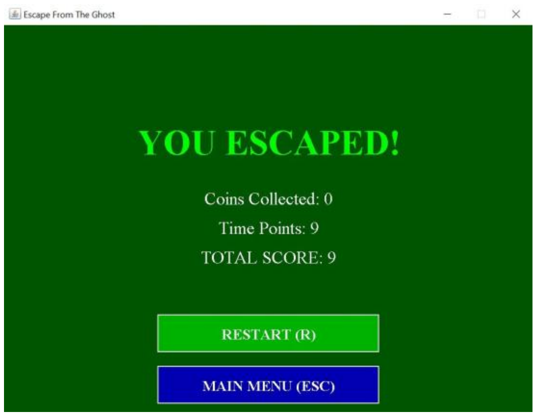


YOU FAILED TO ESCAPE...

Coins Collected: 1

Survival Time: 5 seconds

TOTAL SCORE: 10



YOU ESCAPED!

Coins Collected: 0

Time Points: 9

TOTAL SCORE: 9

6.2 Summary of Results

All game features worked as expected. Player movement, collision detection, sound playback, scoring, and file-based history storage functioned correctly without crashes or logical errors

6.3 Challenges and Solutions

Major Challenges

- Managing game states correctly
- logic behind moving objects

Solutions

- Introduced GameState enum
- Used bounding-box collision & math based logic

Conclusion

Summary of the Project

The Escape From The Ghost game successfully demonstrates a complete Java-based game using Object-Oriented Programming. The project integrates GUI design, game logic, audio, file handling, and OOP concepts into a single cohesive application.

What Was Learned

Through this project, we gained:

- Strong understanding of OOP principles
- Experience with Java Swing and event handling
- Practical knowledge of game loops and logic
- Confidence in building large-scale Java applications