**Experiment 02- Identifying Design Requirements for an Architecture of any specific domain**

**Learning Objective:** To implement UML and design patterns for the project.

**Tools:**  MS Word, draw.io or lucid chart.

**Theory:**

**Unified Modeling Language** (UML) is a standardized modeling language used in software engineering for visualizing, specifying, constructing, and documenting the artifacts of a software system.

**1. Use Case Diagram–** Use case diagrams in UML depict the functionality of a system from a user's perspective. Use Case for our project is –

****

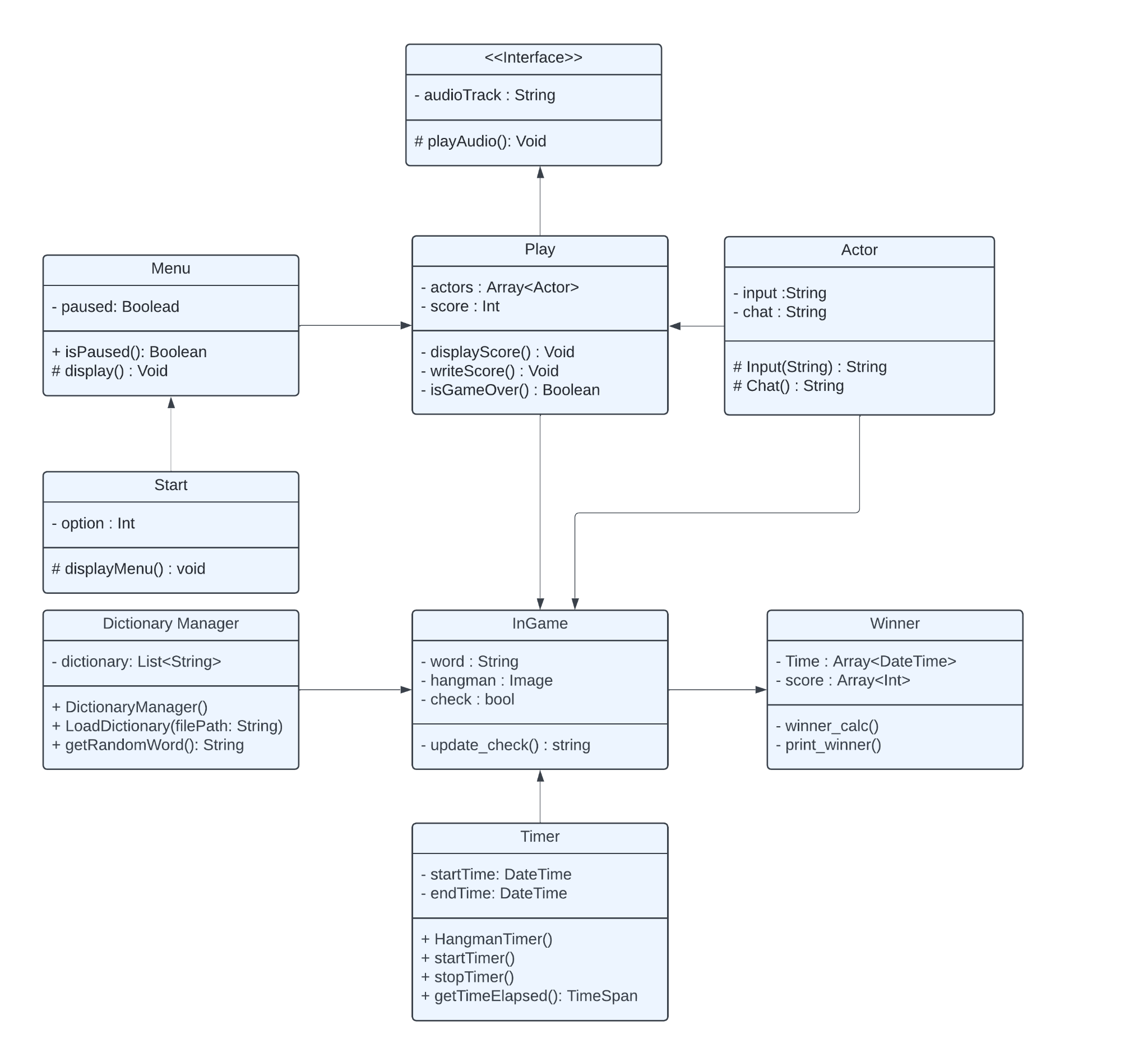
**Actors–**

1. Player- The player plays the game in either single player or multiplayer mode

**Use cases–**

1. Play game- When the player wants to play the game, this is the use case they will use.
2. Exit game- Use case for when the player wants to exit the game
3. Start game- Use case for when the player opens the application, this is a necessary action the user must take before they can perform the Play game or Exit game use cases
4. Guess the word- During the game, the player guesses letters to figure out the word.
5. Player Wins- The outcome when the player correctly guesses the word.
6. Hangman is Hanged- The outcome when the player fails to guess the word in the 6 attempts.
7. Play Again?- After either winning or losing, the player decides whether to play another game.

**2. Class Diagram–** Class diagrams in UML describe the structure of a system by showing the system's classes, their attributes, methods, and the relationships between objects.



Classes used –

1. **Menu Class**

isPaused(): Boolean: Returns whether the game is currently paused.

display(): Void: Displays the menu. (This method is protected, indicated by #, so it's accessible within this class and its subclasses.)

1. **Start Class (inherits from Menu)**

displayMenu(): void: Displays the start menu options. (Protected method.)

1. **Play Class**

displayScore(): Void: Displays the current score.

writeScore(): Void: Writes the score to a persistent storage or display.

isGameOver(): Boolean: Checks if the game is over.

1. **Actor Class**

Input(String): String: Processes input from the actor. (Protected method.)

Chat(): String: Handles chat messages from the actor. (Protected method.)

1. **InGame Class**

update\_check(): string: Updates the check status and returns a status message.

1. **Timer Class**

HangmanTimer(): Constructor to initialize the timer.

startTimer(): Starts the timer.

stopTimer(): Stops the timer.

getTimeElapsed(): TimeSpan: Gets the elapsed time between the start and end times.

1. **Winner Class**

winner\_calc(): Calculates the winner based on scores and times.

print\_winner(): Displays the winner.

### Interactions:

* Menu interacts with Play to manage game states (paused or active).
* Start inherits from Menu to provide initial game options and manage game startup.
* Play manages the core game logic, including the score and game actors.
* Actor represents the players, handling their inputs and chat functionalities.
* Dictionary Manager provides the words used in the game, loaded from an external source.
* InGame handles the current game session, including the word to guess and the hangman image.
* Timer is used to track the game time, important for determining the winner.
* Winner calculates and displays the winner based on the game scores and times.

**3. State Diagram–** State diagrams in UML depict the various states that an object or system can be in and how it transitions between those states in response to events.

**States–**

Initial State: The game starts here.

Transition: Correct guess or incorrect guess.

Hangman Stage 1 to Hangman Stage 5 (Red Rectangles):

These represent the stages of the hangman figure being drawn.

Transition:

Correct guess: Moves to the respective correct guess state.

Incorrect guess: Moves to the next hangman stage or the lose state if at Hangman Stage 5.

Correct Guesses 1 to 4 (Green Rectangles):

These states represent correct guesses where the word is incomplete.

Transition:

Correct guess: Moves to the next correct guess state or to the win state if the word is complete.

Incorrect guess: Moves to the respective hangman stage.

Initial and Consecutive Correct Guesses (Green Rectangle):

The state where initial and consecutive correct guesses occur, but the word is still incomplete.

Transition:

Correct guess and word incomplete: Stays in the same state.

Correct guess and word complete: Moves to the win state.

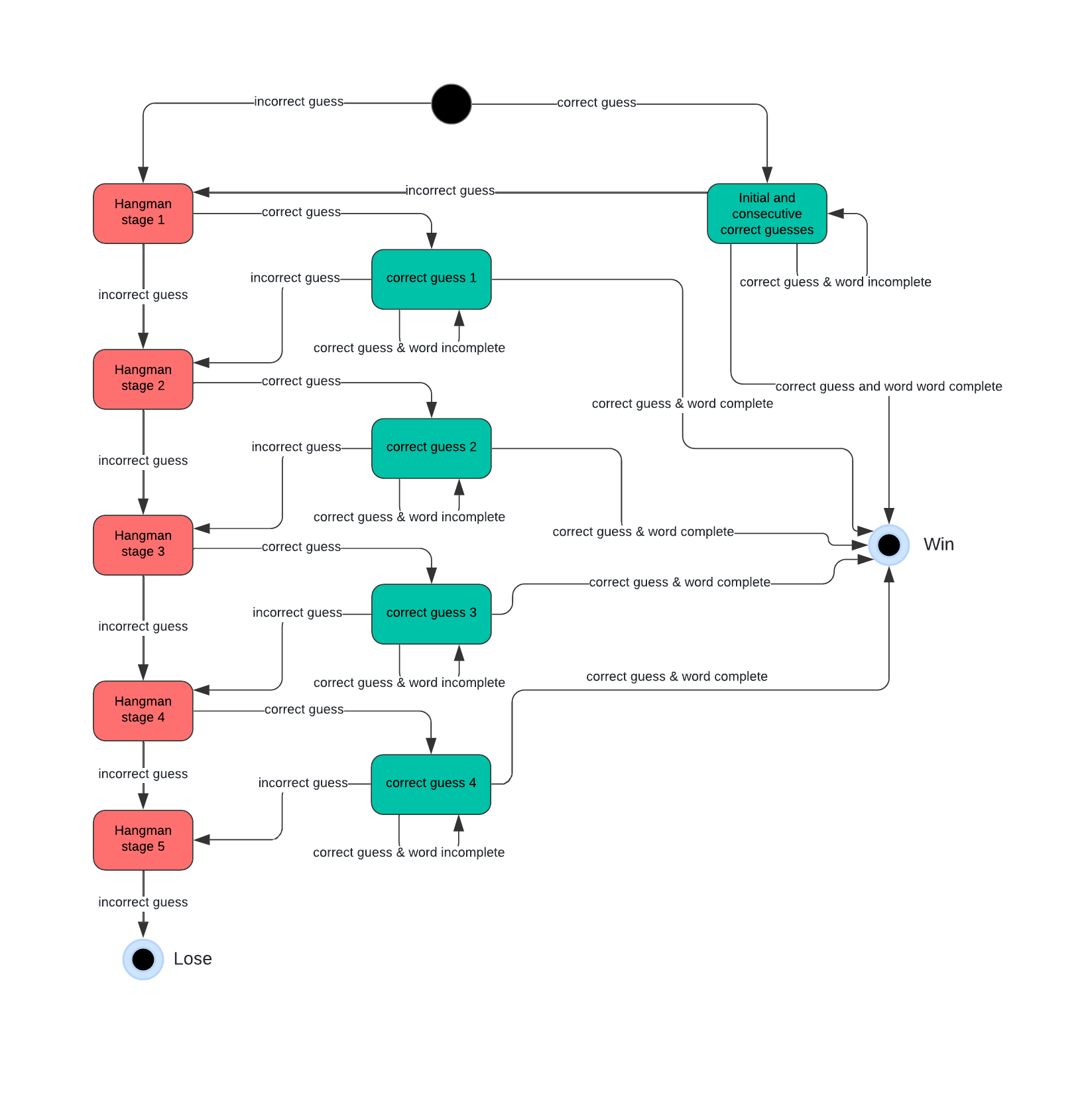
Incorrect guess: Moves to Hangman Stage 1.

Win (Black Circle with Blue Outline):

The state reached when the player correctly guesses the entire word.

Lose (Black Circle with Blue Outline):

The state reached when the player makes five incorrect guesses.



**Design patterns** are general, reusable solutions to common problems in software design. They provide templates on how to solve problems in various contexts and improve code readability, maintainability, and scalability.

The Design Patterns used in our project are –

1. **Singleton** **–**

We are using Singleton to manage Game Lobby. The lobby where players join by entering a password should be a single, globally accessible instance. This ensures that all players are interacting with the same lobby instance and that the game state is consistent across all players.

1. **Prototype –**

The prototype pattern is a creational design pattern in software development. It is used when the types of objects to create are determined by a prototypical instance, which is cloned to produce new objects. We are using Prototype pattern to prototype the game instances and clone the players which will allow players to have individual gameplay.

1. **State –**

The state pattern is a behavioral software design pattern that allows an object to alter its behavior when its internal state changes. This pattern is close to the concept of finite-state machines. We are using State design pattern to represent the parts of the hangman as different states. If the player reaches the final state, it means that the player wasn’t able to guess the word and lost the game. If he decides to play the game again then the state is set again to initial state.

**Learning Outcomes:** The student should have the ability to:

LO 1: Identify the importance of class diagrams.

LO 2: Draw class diagrams for a given scenario.

LO 3: Identify design patterns applicable to your project

**Course Outcomes:** Upon completion of the course students will be able to understand and demonstrate class diagrams.

**Conclusion:**

From this experiment we were able to understand what a class diagram is, its importance and how to draw a class diagram. Also we were able to draw it for our project Homicidal Hunch. Furthermore we were able to learn more about Design Patterns and were able to identify the design patterns applicable to our project.

**For Faculty Use**

| **Correction Parameters** | **Formative Assessment [40%]** | **Timely completion of Practical [ 40%]** | **Attendance / Learning Attitude [20%]** | **Total** |
| --- | --- | --- | --- | --- |
| **Marks Obtained** |  |  |  |  |