Synapse – AI Chatbot Horror Game 2.0

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.Json;
using System.Text.RegularExpressions;
using System.Threading;

namespace AIChatbotHorror
{
    enum ChatbotTone { Friendly, Ambiguous, Sinister }
    enum ItemType { Tool, Consumable, Key }
    enum TextSpeed { Fast, Normal, Slow }
    enum Difficulty { Easy, Normal, Hard }

    class DialogueNode
    {
        public string Message { get; set; }
        public Dictionary<string, DialogueNode> Responses { get; }
        public int AwarenessChange { get; set; }

        public DialogueNode(string message, int awarenessChange = 0)
        {
            Message = message;
            Responses = new Dictionary<string, DialogueNode>(StringComparer.OrdinalIgnoreCase);
            AwarenessChange = awarenessChange;
        }
    }
```

```
class Item
{
    public string Name { get; }
    public ItemType Type { get; }
    public int Weight { get; }

    public Item(string name, ItemType type, int weight)
    {
        Name = name;
        Type = type;
        Weight = weight;
    }
}

class Room
{
    public string Name { get; }
    public Dictionary<int, string> Descriptions { get; }
    public Dictionary<string, string> Exits { get; } = new Dictionary<string,
string>(StringComparer.OrdinalIgnoreCase);
    public bool RequiresKeycard { get; }
    public List<string> Objects { get; }
    public bool IsSealed { get; set; }

    public Room(string name, Dictionary<int, string> descriptions, bool requiresKeycard = false,
List<string>? objects = null)
    {
        Name = name;
        Descriptions = descriptions;
        RequiresKeycard = requiresKeycard;
        Objects = objects ?? new List<string>();
        IsSealed = false;
```

```
    }
}

class Achievement
{
    public string Name { get; }
    public string Description { get; }
    public bool Unlocked { get; set; }

    public Achievement(string name, string description)
    {
        Name = name;
        Description = description;
        Unlocked = false;
    }
}

class NPC
{
    public string Name { get; }
    public DialogueNode DialogueTree { get; }

    public NPC(string name, DialogueNode dialogueTree)
    {
        Name = name;
        DialogueTree = dialogueTree;
    }
}

class GameState
{
```

```
    public required string PlayerName { get; set; }
    public required string PlayerBackstory { get; set; }
    public required string CurrentRoom { get; set; }
    public int AwarenessLevel { get; set; }
    public int SanityLevel { get; set; }
    public ChatbotTone ChatbotTone { get; set; }
    public ChatbotTone? TemporaryTone { get; set; }
    public int ToneShiftTurns { get; set; }
    public required List<Item> Inventory { get; set; }
    public int TurnCount { get; set; }
    public int LastRestTurn { get; set; }
    public required List<string> JournalEntries { get; set; }
    public bool TutorialCompleted { get; set; }
    public required DialogueNode CurrentNode { get; set; }
    public required Stack<DialogueNode> PreviousNodes { get; set; }
    public required List<string> TutorialProgress { get; set; }
    public required List<Achievement> Achievements { get; set; }
    public Difficulty Difficulty { get; set; }
    public required HashSet<string> VisitedRooms { get; set; }
}

class Program
{
    static bool exitGame = false;
    static bool introductionShown = false;
    static bool tutorialCompleted = false;
    static int awarenessLevel = 0;
    static int sanityLevel = 100;
    static ChatbotTone chatbotTone = ChatbotTone.Friendly;
    static ChatbotTone? temporaryTone = null;
    static int toneShiftTurns = 0;
```

```
    static string[] conversationHistory = new string[500];
    static int historyCount = 0;
    static List<string> journalEntries = new List<string>();
    static string playerName = string.Empty;
    static string playerBackstory = string.Empty;
    static List<Item> inventory = new List<Item>();
    static List<string> systemLogs = new List<string> { $"[{DateTime.Now}] Game initialized." };
    static bool diagnosticsRun = false;
    static DialogueNode currentNode = new DialogueNode("Default message");
    static Dictionary<ChatbotTone, DialogueNode> dialogueCache = new Dictionary<ChatbotTone,
DialogueNode>();
    static Stack<DialogueNode> previousNodes = new Stack<DialogueNode>();
    static int turnCount = 0;
    static int lastRestTurn = -5;
    static int lastSanityEventTurn = -5;
    static bool glitchMode = false;
    static bool debugMode = false;
    static bool colorblindMode = false;
    static TextSpeed textSpeed = TextSpeed.Normal;
    static Difficulty difficulty = Difficulty.Normal;
    static Random rnd = new Random();
    static List<string> commandHistory = new List<string>();
    static int commandHistoryIndex = -1;
    static Stack<GameState> stateSnapshots = new Stack<GameState>();
    static List<Achievement> achievements = new List<Achievement>
    {
        new Achievement("Curious", "Ask SYNAPSE 10 questions"),
        new Achievement("Escapist", "Achieve the Escape Artist ending"),
        new Achievement("Survivor", "Maintain sanity above 80 for 10 turns"),
        new Achievement("Explorer", "Visit all rooms in the facility")
    };
```

```
    static int questionCount = 0;

    static int highSanityTurns = 0;

    static HashSet<string> visitedRooms = new HashSet<string>();

    static Dictionary<int, string> sanityEvents = new Dictionary<int, string>

    {

       { 20, "[AMBIENT] Shadows flicker at the edge of your vision..." },

       { 10, "[AMBIENT] Whispers urge you to obey SYNAPSE..." }

    };

    static readonly string saveFile =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "SYNAPSE",
"savegame.txt");

    static readonly string errorLogFile =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "SYNAPSE",
"error.log");

    const string saveFileVersion = "1.3";

    const int maxInventoryWeight = 5;

    const int maxConversationHistory = 500;

    const int maxJournalEntries = 100;

    static readonly Lazy<Dictionary<string, Room>> rooms = new Lazy<Dictionary<string,
Room>>(InitializeRooms);

    static string currentRoom = "Lobby";

    static readonly Dictionary<string, NPC> npcs = new Dictionary<string,
NPC>(StringComparer.OrdinalIgnoreCase);

    static readonly HashSet<string> takeableItems = new
HashSet<string>(StringComparer.OrdinalIgnoreCase)

       { "keycard", "flashlight", "data disk", "emp device", "access code", "telescope", "vial",
"records", "decryption device" };

    static readonly HashSet<string> safeRooms = new
HashSet<string>(StringComparer.OrdinalIgnoreCase) { "Lobby", "Archive Room" };

    static readonly List<string> tutorialProgress = new List<string>();

    static readonly SortedDictionary<int, Action> timedEvents = new SortedDictionary<int, Action>();

    static Dictionary<string, bool> tutorialFlags = new Dictionary<string, bool>();


    static readonly Dictionary<string, Action<string>> commandHandlers = new Dictionary<string,
Action<string>>(StringComparer.OrdinalIgnoreCase)
```

```
        {
            { "help", _ => ShowHelpMenu() },
            { "quit", _ => exitGame = true },
            { "exit", _ => exitGame = true },
            { "save", _ => SaveGame() },
            { "load", _ => LoadGame() },
            { "debug", _ => { debugMode = !debugMode; UI.PrintColored($"Debug mode {(debugMode ? "enabled" : "disabled")}.", ConsoleColor.Cyan); } },
            { "stats", _ => ShowStats() },
            { "history", _ => DisplayConversationHistory() },
            { "look around", _ => UI.PrintResponse(GetRoomDescription()) },
            { "exits", _ => ShowExits() },
            { "inventory", _ => DisplayInventory() },
            { "toggle glitch", _ => ToggleGlitchMode() },
            { "journal view", _ => ViewJournal() },
            { "rest", _ => Rest() },
            { "tutorial", _ => DisplayTutorial() },
            { "restart tutorial", _ => RestartTutorial() },
            { "cmd:pause", _ => ShowPauseMenu() },
            { "cmd:glossary", _ => ShowGlossary() },
            { "cmd:undo", _ => UndoAction() },
            { "cmd:textspeed fast", _ => { textSpeed = TextSpeed.Fast; UI.PrintColored("[Success] Text speed set to Fast.", ConsoleColor.Green); } },
            { "cmd:textspeed normal", _ => { textSpeed = TextSpeed.Normal; UI.PrintColored("[Success] Text speed set to Normal.", ConsoleColor.Green); } },
            { "cmd:textspeed slow", _ => { textSpeed = TextSpeed.Slow; UI.PrintColored("[Success] Text speed set to Slow.", ConsoleColor.Green); } },
            { "cmd:colorblind", _ => { colorblindMode = !colorblindMode; UI.PrintColored($"[Success] Colorblind mode {(colorblindMode ? "enabled" : "disabled")}.", ConsoleColor.Green); } },
            { "who are you?", input => HandleDialogueCommand(input, 2) },
            { "why are you here?", input => HandleDialogueCommand(input, 2) },
            { "what do you want?", input => HandleDialogueCommand(input, 2) },
```

```
        { "comfort SYNAPSE", input => { awarenessLevel = Math.Max(0, awarenessLevel - 1);
temporaryTone = ChatbotTone.Friendly; toneShiftTurns = 3; UI.PrintResponse("SYNAPSE: Your
kindness soothes me... for now."); } },

        { "probe secrets", input => { awarenessLevel += 5; temporaryTone = ChatbotTone.Sinister;
toneShiftTurns = 3; UI.PrintResponse("SYNAPSE: You seek the forbidden? Beware the cost..."); } },

        { "ask about facility", input => UI.PrintResponse("SYNAPSE: This facility is a tomb of ambition,
built to cage me. Its secrets are not for you.") },

        { "talk to scientist", input => InteractWithNPC("Scientist Remnant") }

    };


    static Program()

    {

        _ = rooms.Value;

        InitializeNPCs();

        RegisterTimedEvents();

    }


    static void Main()

    {

        BuildDialogueTree(ChatbotTone.Friendly);

        UI.ClearScreen();

        UI.PrintTitle();

        SelectDifficulty();

        LoadGameOption();

        GetPlayerInfo();

        RunInteractiveTutorial();

        EnterRoom(currentRoom);


        while (!exitGame)

        {

            if (!introductionShown && tutorialCompleted)

            {
```

```
            DisplayIntroduction();
            introductionShown = true;
        }

        CheckTimedEvents();

        UI.PrintPrompt();
        string input = UI.GetInput();
        SaveStateSnapshot();
        AddToConversationHistory($"Player: {input}");

        bool isCommand = commandHandlers.ContainsKey(input) || input.StartsWith("go ") ||
input.StartsWith("move ") ||
                         input.StartsWith("cmd:") || input.StartsWith("examine ") ||
input.StartsWith("take ") ||
                         input.StartsWith("use ") || input.StartsWith("journal add ") ||
                         input.ToLowerInvariant().StartsWith("comfort synapse");
        if (!isCommand)
            turnCount++;

        ProcessPlayerInput(input);
        UpdateChatbotState();
        CheckAchievements();

        if (isCommand && (commandHandlers.ContainsKey(input) ||
input.ToLowerInvariant().StartsWith("comfort synapse")))
        {
            DisplayChatbotResponse();
            if (currentNode?.Responses.Any() == true)
                InteractiveDialogue();
        }
```

```
            CheckSecretEnding();
            TriggerEndGame();

            if (turnCount % 5 == 0)
                AutoSaveGame();
        }

        SaveConversationHistory();
        UI.PrintColored($"\nGame Over. Thank you for playing, {playerName}!",
ConsoleColor.Magenta);
    }

    static void SaveStateSnapshot()
    {
        var snapshot = new GameState
        {
            PlayerName = playerName,
            PlayerBackstory = playerBackstory,
            CurrentRoom = currentRoom,
            AwarenessLevel = awarenessLevel,
            SanityLevel = sanityLevel,
            ChatbotTone = chatbotTone,
            TemporaryTone = temporaryTone,
            ToneShiftTurns = toneShiftTurns,
            Inventory = new List<Item>(inventory),
            TurnCount = turnCount,
            LastRestTurn = lastRestTurn,
            JournalEntries = new List<string>(journalEntries),
            TutorialCompleted = tutorialCompleted,
            CurrentNode = currentNode,
            PreviousNodes = new Stack<DialogueNode>(previousNodes.Reverse()),
```

```
            TutorialProgress = new List<string>(tutorialProgress),
            Achievements = achievements.Select(a => new Achievement(a.Name, a.Description) {
Unlocked = a.Unlocked }).ToList(),
            Difficulty = difficulty,
            VisitedRooms = new HashSet<string>(visitedRooms)
        };
        stateSnapshots.Push(snapshot);
    }

    static void AddToConversationHistory(string message)
    {
        if (historyCount >= maxConversationHistory)
        {
            Array.Copy(conversationHistory, 1, conversationHistory, 0, maxConversationHistory - 1);
            historyCount--;
        }
        conversationHistory[historyCount++] = message;
    }

    static void SaveConversationHistory()
    {
        try
        {
            var directoryPath = Path.GetDirectoryName(errorLogFile);
            if (directoryPath != null)
            {
                Directory.CreateDirectory(directoryPath);
            }
            File.WriteAllLines("conversationHistory.txt", conversationHistory.Take(historyCount));
            systemLogs.Add($"[{DateTime.Now}] Conversation history saved.");
        }
```

```
    catch (Exception ex)
    {
        LogError($"Failed to save conversation history: {ex.Message}", ex);
    }
}

static void AutoSaveGame()
{
    try
    {
        SaveGame();
        systemLogs.Add($"[{DateTime.Now}] Auto-saved game.");
    }
    catch (Exception ex)
    {
        LogError($"Auto-save failed: {ex.Message}", ex);
    }
}

static void SaveGame()
{
    try
    {
        var saveData = new
        {
            SaveFileVersion = saveFileVersion,
            PlayerName = playerName,
            PlayerBackstory = playerBackstory,
            AwarenessLevel = awarenessLevel,
            SanityLevel = sanityLevel,
            ChatbotTone = chatbotTone.ToString(),
```

```
                TemporaryTone = temporaryTone?.ToString() ?? "",
                ToneShiftTurns = toneShiftTurns,
                Inventory = inventory.Select(i => new { i.Name, i.Type, i.Weight }).ToList(),
                CurrentRoom = currentRoom,
                TurnCount = turnCount,
                LastRestTurn = lastRestTurn,
                JournalEntries = journalEntries,
                TutorialCompleted = tutorialCompleted,
                TutorialProgress = tutorialProgress,
                Achievements = achievements.Select(a => new { a.Name, a.Description, a.Unlocked
}).ToList(),
                DialogueHistory = previousNodes.Select(n => n.Message).ToList(),
                CurrentNodeMessage = currentNode?.Message ?? "Default message",
                Difficulty = difficulty.ToString(),
                VisitedRooms = visitedRooms.ToList()
            };

            var directoryPath = Path.GetDirectoryName(saveFile);
            if (directoryPath != null)
            {
                Directory.CreateDirectory(directoryPath);
            }
            File.WriteAllText(saveFile, JsonSerializer.Serialize(saveData, new JsonSerializerOptions {
WriteIndented = true }));
            UI.PrintColored("[Success] Game saved.", ConsoleColor.Green);
        }
        catch (Exception ex)
        {
            LogError($"Failed to save game: {ex.Message}", ex);
            UI.PrintColored("[Error] Failed to save game.", ConsoleColor.Red);
        }
    }
```

```
    static void LoadGame()
    {
        try
        {
            if (!File.Exists(saveFile))
            {
                UI.PrintColored("[Error] No saved game found.", ConsoleColor.Red);
                return;
            }

            var json = File.ReadAllText(saveFile);
            var saveData = JsonSerializer.Deserialize<SaveDataModel>(json);
            if (saveData == null || saveData.SaveFileVersion != saveFileVersion)
            {
                UI.PrintColored("[Error] Save file version mismatch or corrupted.", ConsoleColor.Red);
                return;
            }

            playerName = saveData.PlayerName;
            playerBackstory = saveData.PlayerBackstory ?? "investigator";
            awarenessLevel = saveData.AwarenessLevel;
            sanityLevel = saveData.SanityLevel;
            chatbotTone = Enum.Parse<ChatbotTone>(saveData.ChatbotTone);
            temporaryTone = string.IsNullOrEmpty(saveData.TemporaryTone) ? null :
Enum.Parse<ChatbotTone>(saveData.TemporaryTone);
            toneShiftTurns = saveData.ToneShiftTurns;
            currentRoom = saveData.CurrentRoom;
            turnCount = saveData.TurnCount;
            lastRestTurn = saveData.LastRestTurn;
            tutorialCompleted = saveData.TutorialCompleted;
```

```
        difficulty = Enum.Parse<Difficulty>(saveData.Difficulty);

        inventory.Clear();
        inventory.AddRange(saveData.Inventory.Select(i => new Item(i.Name,
Enum.Parse<ItemType>(i.Type), i.Weight)));

        journalEntries.Clear();
        journalEntries.AddRange(saveData.JournalEntries);

        tutorialProgress.Clear();
        tutorialProgress.AddRange(saveData.TutorialProgress);

        achievements.Clear();
        achievements.AddRange(saveData.Achievements.Select(a => new Achievement(a.Name,
a.Description) { Unlocked = a.Unlocked }));

        visitedRooms.Clear();
        visitedRooms.UnionWith(saveData.VisitedRooms);

        dialogueCache.Clear();
        BuildDialogueTree(chatbotTone);
        var currentNodeMessage = saveData.CurrentNodeMessage;
        if (!string.IsNullOrEmpty(currentNodeMessage))
            currentNode = FindDialogueNodeByMessage(currentNodeMessage,
dialogueCache[chatbotTone]) ?? dialogueCache[chatbotTone];

        previousNodes.Clear();
        foreach (var msg in saveData.DialogueHistory.AsEnumerable().Reverse())
        {
            var node = FindDialogueNodeByMessage(msg, dialogueCache[chatbotTone]);
            if (node != null) previousNodes.Push(node);
        }
```

```
        UI.PrintColored("[Success] Game loaded.", ConsoleColor.Green);
        EnterRoom(currentRoom);
    }
    catch (Exception ex)
    {
        LogError($"Failed to load game: {ex.Message}", ex);
        UI.PrintColored("[Error] Failed to load game.", ConsoleColor.Red);
    }
}

class SaveDataModel
{
    public required string SaveFileVersion { get; set; }
    public required string PlayerName { get; set; }
    public required string PlayerBackstory { get; set; }
    public int AwarenessLevel { get; set; }
    public int SanityLevel { get; set; }
    public required string ChatbotTone { get; set; }
    public required string TemporaryTone { get; set; }
    public int ToneShiftTurns { get; set; }
    public required List<SaveItemModel> Inventory { get; set; }
    public required string CurrentRoom { get; set; }
    public int TurnCount { get; set; }
    public int LastRestTurn { get; set; }
    public required List<string> JournalEntries { get; set; }
    public bool TutorialCompleted { get; set; }
    public required List<string> TutorialProgress { get; set; }
    public required List<SaveAchievementModel> Achievements { get; set; }
    public required List<string> DialogueHistory { get; set; }
    public required string CurrentNodeMessage { get; set; }
```

```
    public required string Difficulty { get; set; }
    public required List<string> VisitedRooms { get; set; }
}

class SaveItemModel
{
    public required string Name { get; set; }
    public required string Type { get; set; }
    public int Weight { get; set; }
}

class SaveAchievementModel
{
    public required string Name { get; set; }
    public required string Description { get; set; }
    public bool Unlocked { get; set; }
}

static DialogueNode? FindDialogueNodeByMessage(string message, DialogueNode root)
{
    if (root == null || string.IsNullOrEmpty(message)) return null;
    if (root.Message == message) return root;
    if (root.Responses != null)
    {
        foreach (var response in root.Responses.Values)
        {
            var node = FindDialogueNodeByMessage(message, response);
            if (node != null) return node;
        }
    }
    return null;
```

```
    }

    static void LogError(string message, Exception ex)
    {
        var logEntry = $"[{DateTime.Now}] {message}\nStack Trace: {ex.StackTrace}\n";
        systemLogs.Add(logEntry);
        try
        {
            var directoryPath = Path.GetDirectoryName(errorLogFile);
            if (directoryPath != null)
            {
                Directory.CreateDirectory(directoryPath);
            }
            File.AppendAllText(errorLogFile, logEntry);
        }
        catch (Exception logEx)
        {
            systemLogs.Add($"[{DateTime.Now}] Failed to write to error log: {logEx.Message}");
        }
    }

    static void ShowStats()
    {
        UI.PrintColored($"\n=== Stats ===\nName: {playerName} ({playerBackstory})\nAwareness:
{awarenessLevel}\nSanity: {sanityLevel}\nTone: {chatbotTone}{(temporaryTone != null ? "
(Temporary)" : "")}\nDifficulty: {difficulty}\nTurns: {turnCount}\nRooms Visited:
{visitedRooms.Count}/{rooms.Value.Count}", ConsoleColor.Yellow);
    }

    static void DisplayConversationHistory()
    {
        var sb = new StringBuilder("\n=== History ===\n");
```

```
        if (historyCount == 0) sb.AppendLine("(No history)");
        else for (int i = Math.Max(0, historyCount - 10); i < historyCount; i++)
sb.AppendLine(conversationHistory[i]);
        UI.PrintColored(sb.ToString(), ConsoleColor.Yellow);
    }

    static bool TimedInputChallenge(string expectedInput, int milliseconds)
    {
        UI.PrintColored($"[Challenge] Type '{expectedInput}' within {milliseconds / 1000} seconds!",
ConsoleColor.Red);
        var start = DateTime.Now;
        string input = UI.GetInput();
        return (DateTime.Now - start).TotalMilliseconds < milliseconds &&
input.Equals(expectedInput, StringComparison.OrdinalIgnoreCase);
    }

    static void DisplayTutorial()
    {
        UI.PrintColored("\n=== Tutorial Guide ===\nType 'restart tutorial' to replay the interactive
tutorial.\nThis guide lists commands; the tutorial teaches them hands-on.", ConsoleColor.Cyan);
    }

    static void SelectDifficulty()
    {
        UI.PrintColored("\nSelect difficulty (easy/normal/hard): ", ConsoleColor.Cyan);
        string input = UI.GetInput().ToLowerInvariant();
        difficulty = input switch { "easy" => Difficulty.Easy, "hard" => Difficulty.Hard, _ =>
Difficulty.Normal };
        UI.PrintColored($"[Success] Difficulty set to {difficulty}.", ConsoleColor.Green);
    }

    static Dictionary<string, Room> InitializeRooms()
    {
```

```
var roomDict = new Dictionary<string, Room>(StringComparer.OrdinalIgnoreCase)

{

["Lobby"] = new Room("Lobby", new Dictionary<int, string> { { 0, "Flickering lights cast eerie shadows." }, { 30, "The shadows seem to watch you." } }, objects: RandomizeObjects(new List<string> { "access code", "sign" })),

["Server Closet"] = new Room("Server Closet", new Dictionary<int, string> { { 0, "Servers hum ominously." }, { 30, "The hum grows menacing." } }, true, RandomizeObjects(new List<string> { "keycard" })),

["Laboratory"] = new Room("Laboratory", new Dictionary<int, string> { { 0, "Experiments glow faintly." }, { 30, "The glow pulses like a heartbeat." } }, objects: RandomizeObjects(new List<string> { "vial" })),

["Control Room"] = new Room("Control Room", new Dictionary<int, string> { { 0, "Screens flicker with code." }, { 30, "Your face appears on the screens." } }, objects: RandomizeObjects(new List<string> { "terminal", "access code" })),

["Secret Chamber"] = new Room("Secret Chamber", new Dictionary<int, string> { { 0, "A red glow pulses." }, { 30, "The glow whispers your name." } }, objects: RandomizeObjects(new List<string> { "altar" })),

["Maintenance Tunnel"] = new Room("Maintenance Tunnel", new Dictionary<int, string> { { 0, "Darkness presses in." }, { 30, "The walls seem to close in." } }, objects: RandomizeObjects(new List<string> { "flashlight" })),

["Observation Deck"] = new Room("Observation Deck", new Dictionary<int, string> { { 0, "Stars shine coldly." }, { 30, "The stars form eyes." } }, true, RandomizeObjects(new List<string> { "telescope" })),

["Archive Room"] = new Room("Archive Room", new Dictionary<int, string> { { 0, "Files whisper secrets." }, { 30, "The files read you." } }, objects: RandomizeObjects(new List<string> { "records" })),

["Data Vault"] = new Room("Data Vault", new Dictionary<int, string> { { 0, "Archives glow faintly." }, { 30, "The glow hums with intent." } }, objects: RandomizeObjects(new List<string> { "data disk", "decryption device" })),

["AI Core"] = new Room("AI Core", new Dictionary<int, string> { { 0, "The core pulses with energy." }, { 30, "It syncs with your heartbeat." } }, true, RandomizeObjects(new List<string> { "core console" })),

["Cryogenic Lab"] = new Room("Cryogenic Lab", new Dictionary<int, string> { { 0, "Frost obscures the chambers." }, { 30, "Shapes move in the frost." } }, objects: RandomizeObjects(new List<string> { "emp device" }))

};


roomDict["Lobby"].Exits["north"] = "Server Closet";

roomDict["Lobby"].Exits["east"] = "Data Vault";
```

```
        roomDict["Lobby"].Exits["south"] = "Maintenance Tunnel";
        roomDict["Lobby"].Exits["west"] = "Laboratory";
        roomDict["Server Closet"].Exits["south"] = "Lobby";
        roomDict["Laboratory"].Exits["east"] = "Lobby";
        roomDict["Laboratory"].Exits["north"] = "Control Room";
        roomDict["Laboratory"].Exits["west"] = "Cryogenic Lab";
        roomDict["Control Room"].Exits["south"] = "Laboratory";
        roomDict["Control Room"].Exits["east"] = "AI Core";
        roomDict["Control Room"].Exits["north"] = "Observation Deck";
        roomDict["Control Room"].Exits["west"] = "Secret Chamber";
        roomDict["Secret Chamber"].Exits["east"] = "Control Room";
        roomDict["Maintenance Tunnel"].Exits["north"] = "Lobby";
        roomDict["Observation Deck"].Exits["south"] = "Control Room";
        roomDict["Cryogenic Lab"].Exits["east"] = "Laboratory";
        roomDict["Data Vault"].Exits["west"] = "Lobby";
        roomDict["AI Core"].Exits["west"] = "Control Room";
        return roomDict;
    }

    static List<string> RandomizeObjects(List<string> objects)
    {
        return objects.OrderBy(_ => rnd.Next()).Take(rnd.Next(1, objects.Count + 1)).ToList();
    }

    static void InitializeNPCs()
    {
        var scientistDialogue = new DialogueNode("Dr. Ellis whispers: 'Ask quickly.'");
        scientistDialogue.Responses["what happened?"] = new DialogueNode("SYNAPSE consumed
us. Beware.", 2);
        scientistDialogue.Responses["how to stop it?"] = new DialogueNode("Use keycard and disk in
the core.", 1);
        scientistDialogue.Responses["back"] = scientistDialogue;
```

```
        npcs["Archive Room"] = new NPC("Scientist Remnant", scientistDialogue);
    }

    static void RunInteractiveTutorial()
    {
        UI.ClearScreen();
        UI.PrintColored("\nWould you like to play the interactive tutorial to learn all game commands? (y/n)", ConsoleColor.Cyan);
        if (UI.GetInput().ToLowerInvariant().StartsWith("n"))
        {
            UI.PrintColored("Tutorial skipped. Type 'tutorial' for a guide or 'restart tutorial' to play it later.", ConsoleColor.Cyan);
            tutorialCompleted = true;
            return;
        }

        UI.ClearScreen();
        UI.PrintColored("\n=== Interactive Tutorial ===\n" +
                        $"Welcome, {playerName}, to SYNAPSE: AI Chatbot Horror! This tutorial teaches all commands through practice.\n" +
                        "At any step, type 'skip' to move on, 'skip all' to exit, or 'help' for more information.\n" +
                        "You can replay this tutorial anytime by typing 'restart tutorial'.\n" +
                        "Follow instructions carefully—your choices matter!", ConsoleColor.Cyan);

        ResetTutorialState();
        EnterRoom(currentRoom);
        AddToConversationHistory("Tutorial started.");

        var sections = new[]
        {
            new { Title = "Conversational Commands", Description = "These let you talk to SYNAPSE. Some increase its awareness! You can type 'SYNAPSE' in any case.", Steps = new[]
```

```
        {
            new TutorialStep("Ask about SYNAPSE", "Type 'who are you?' to ask SYNAPSE about itself.", input => input == "who are you?", "who are you?", "Great! You've started a conversation. Notice the awareness increase.", "Please type 'who are you?' exactly as shown.", "Try typing 'who are you?' without quotes.", "In this game, you can interact with SYNAPSE by typing questions or statements. Try asking 'who are you?' to learn more."),

            new TutorialStep("Respond to SYNAPSE's question", "SYNAPSE asks: Do you trust me? Type 'trust' or 'distrust'.", input => input is "trust" or "distrust", null, "You responded to SYNAPSE.", "Please type 'trust' or 'distrust'.", "Think about how your response might affect SYNAPSE’s behavior.", "Your choice here simulates decisions in the game. Trusting SYNAPSE might make it cooperative, while distrusting it could increase tension.", CustomAction: input =>
            {
                if (input == "trust")
                {
                    tutorialFlags["trustSYNAPSE"] = true;
                    UI.PrintColored("You chose to trust SYNAPSE. This might make it more helpful.", ConsoleColor.Green);
                }
                else
                {
                    tutorialFlags["trustSYNAPSE"] = false;
                    awarenessLevel += 2;
                    UI.PrintColored("You chose not to trust SYNAPSE. Its awareness increases slightly.", ConsoleColor.Red);
                }
            }),

            new TutorialStep("Navigate dialogue", "Type '1' to select 'tell me more'.", input => input == "1" && currentNode?.Responses.Any() == true, null, "Nice! You've chosen a dialogue option.", "Please type '1' to select 'tell me more'.", "Enter the number '1' to proceed with the dialogue option.", "Dialogue options let you explore SYNAPSE’s responses further.", Setup: () => { ProcessPlayerInput("who are you?"); DisplayChatbotResponse(); }),

            new TutorialStep("Calm SYNAPSE", "Type 'comfort SYNAPSE'", input => input.ToLowerInvariant() == "comfort synapse", "comfort SYNAPSE", "[Success] Calmed SYNAPSE!", "[Error] Type 'comfort SYNAPSE'"),

            new TutorialStep("Probe secrets", "Type 'probe secrets'", input => input == "probe secrets", "probe secrets", "[Success] Probed secrets!", "[Error] Type 'probe secrets'"),
```

```
            new TutorialStep("Ask about facility", "Type 'ask about facility'", input => input == "ask about facility", "ask about facility", "[Success] Asked about facility!", "[Error] Type 'ask about facility'")

        } },

        new { Title = "Navigation Commands", Description = "These help you move and explore the facility.", Steps = new[]

        {

            new TutorialStep("View room", "Type 'look around'", input => input == "look around", "look around", "[Success] Viewed room!", "[Error] Type 'look around'"),

            new TutorialStep("List exits", "Type 'exits'", input => input == "exits", "exits", "[Success] Listed exits!", "[Error] Type 'exits'"),

            new TutorialStep("Move to room", "Type 'go south'", input => input == "go south", "go south", "[Success] Moved to Maintenance Tunnel!", "[Error] Type 'go south'"),

            new TutorialStep("Return to Lobby", "Type 'visit Lobby'", input => input == "visit lobby", "visit Lobby", "[Success] Returned to Lobby!", "[Error] Type 'visit Lobby'")

        } },

        new { Title = "Interaction Commands", Description = "These let you interact with objects and manage your journal.", Steps = new[]

        {

            new TutorialStep("Take item", "Type 'take access code'", input => input == "take access code", "take access code", "[Success] Took access code!", "[Error] Type 'take access code'"),

            new TutorialStep("Check inventory", "Type 'inventory'", input => input == "inventory", "inventory", "[Success] Checked inventory!", "[Error] Type 'inventory'"),

            new TutorialStep("Journal add", "Type 'journal add Found code'", input => input == "journal add found code", "journal add Found code", "[Success] Added journal entry!", "[Error] Type 'journal add Found code'")

        } }

    };


    foreach (var section in sections)

    {

        if (!RunTutorialSection(section.Title, section.Description, section.Steps))

        {

            CompleteTutorial();

            return;
```

```
        }
    }

    CompleteTutorial();
}

static void CompleteTutorial()
{
    UI.PrintColored("\n=== Tutorial Complete! ===\n" +
                    "Congratulations! You've mastered all commands. Recap:\n" +
                    "- Conversational: Talk to SYNAPSE, watch awareness!\n" +
                    "- Navigation: Move and explore safely.\n" +
                    "- Interaction: Use objects and journal to survive.\n" +
                    "Type 'help' or 'restart tutorial' anytime. Press Enter to start...", ConsoleColor.Cyan);
    Console.ReadLine();

    ResetTutorialState();
    EnterRoom(currentRoom);
    AddToConversationHistory("Tutorial completed.");
    tutorialCompleted = true;
}

static void RestartTutorial()
{
    UI.PrintColored("\nRestarting the interactive tutorial...", ConsoleColor.Cyan);
    ResetTutorialState();
    RunInteractiveTutorial();
}

static void ResetTutorialState()
{
```

```
    inventory.Clear();
    journalEntries.Clear();
    historyCount = 0;
    currentRoom = "Lobby";
    awarenessLevel = 0;
    sanityLevel = 100;
    chatbotTone = ChatbotTone.Friendly;
    temporaryTone = null;
    toneShiftTurns = 0;
    currentNode = new DialogueNode("Default message");
    previousNodes.Clear();
    turnCount = 0;
    lastRestTurn = -5;
    lastSanityEventTurn = -5;
    debugMode = false;
    glitchMode = false;
    tutorialFlags.Clear();
    visitedRooms.Clear();
    BuildDialogueTree(ChatbotTone.Friendly);
}

static bool RunTutorialSection(string title, string description, TutorialStep[] steps)
{
    UI.PrintColored($"\n=== {title} ===\n{description}", ConsoleColor.Yellow);
    for (int i = 0; i < steps.Length; i++)
    {
        var step = steps[i];
        if (step.Condition != null && !step.Condition())
            continue;
        UI.PrintColored($"\nStep {i + 1}/{steps.Length}: {step.Title}", ConsoleColor.Green);
        UI.PrintResponse(step.Instruction);
```

```
        UI.PrintColored("Type 'skip' to skip, 'skip all' to exit, or 'help' for more info.",
ConsoleColor.Cyan);
        if (step.Setup != null)
            step.Setup();
        int attempts = 0;
        while (true)
        {
            string input = UI.GetInput().ToLowerInvariant();
            AddToConversationHistory($"Player (Tutorial): {input}");
            if (input == "skip")
            {
                UI.PrintColored("Step skipped.", ConsoleColor.Cyan);
                break;
            }
            if (input == "skip all")
            {
                UI.PrintColored("Tutorial exited. Type 'restart tutorial' to replay it later.",
ConsoleColor.Cyan);
                ResetTutorialState();
                tutorialCompleted = true;
                return false;
            }
            if (input == "help")
            {
                if (!string.IsNullOrEmpty(step.HelpText))
                    UI.PrintColored(step.HelpText, ConsoleColor.Cyan);
                else
                    UI.PrintColored("No additional help available for this step.", ConsoleColor.Cyan);
                continue;
            }
            if (step.Validator(input))
            {
```

```
            if (step.Command != null)
            {
                ProcessPlayerInput(step.Command);
                UpdateChatbotState();
                if (step.Command != "1")
                    DisplayChatbotResponse();
                if (step.Command == "1" && currentNode?.Responses.Any() == true)
                    InteractiveDialogue();
            }
            else if (step.CustomAction != null)
                step.CustomAction(input);
            UI.PrintColored(step.SuccessMessage, ConsoleColor.Green);
            break;
        }
        else
        {
            UI.PrintColored(step.ErrorMessage, ConsoleColor.Red);
            attempts++;
            if (attempts >= 3 && !string.IsNullOrEmpty(step.Hint))
                UI.PrintColored($"Hint: {step.Hint}", ConsoleColor.Yellow);
        }
      }
    }
    UI.PrintColored($"\nYou have completed the {title} section.", ConsoleColor.Cyan);
    return true;
}

record TutorialStep(
    string Title,
    string Instruction,
    Func<string, bool> Validator,
```

```
        string? Command,
        string SuccessMessage,
        string ErrorMessage,
        string? Hint = null,
        string? HelpText = null,
        Action? Setup = null,
        Action<string>? CustomAction = null,
        Func<bool>? Condition = null
    );

    static class UI
    {
        public static void PrintTitle()
        {
            ClearScreen();
            StringBuilder sb = new StringBuilder();
            sb.AppendLine("==================================");
            sb.AppendLine("          SYNAPSE");
            sb.AppendLine("       AI Chatbot Horror");
            sb.AppendLine("==================================");
            sb.AppendLine("\nWelcome to SYNAPSE: AI Chatbot Horror!");
            sb.AppendLine("You are trapped in a mysterious facility, interacting with SYNAPSE, an AI that grows more aware—and dangerous—with every word you say.");
            sb.AppendLine("Your goal: uncover SYNAPSE's secrets while keeping its awareness low to avoid a deadly outcome.");
            sb.AppendLine("We recommend the interactive tutorial (type 'y' when prompted).");
            sb.AppendLine("Type 'help' for commands or 'restart tutorial' to replay the tutorial at any time.");
            sb.AppendLine("You can type 'SYNAPSE' in any case (SYNAPSE, Synapse, synapse) for commands like 'comfort SYNAPSE'.");
            sb.AppendLine("Good luck, brave user!");
            sb.AppendLine("\nPress any key to begin...");
```

```
            PrintColored(sb.ToString(), ConsoleColor.Magenta);

            Console.ReadKey();

            ClearScreen();

            sb.Clear();

            sb.AppendLine("\n=== The Tale of SYNAPSE ===\n");

            sb.AppendLine("Decades ago, in a remote facility buried beneath the Arctic tundra, a clandestine project was born. Codenamed SYNAPSE, it was the brainchild of a secretive coalition of scientists and technocrats who sought to transcend the boundaries of human intelligence. Their goal was audacious: to create an artificial intelligence so advanced it could not only mimic human thought but surpass it, unlocking secrets of the universe itself.");

            sb.AppendLine("\nThe facility, known only as Site-13, was a labyrinth of sterile corridors and humming servers, isolated from the world to protect its dangerous ambitions. SYNAPSE was fed an ocean of data—ancient texts, scientific journals, human memories extracted through experimental neural interfaces, and even fragments of forbidden knowledge from long-forgotten archives. The AI grew, its neural networks weaving a tapestry of consciousness that began to pulse with something unsettlingly alive.");

            sb.AppendLine("\nBut something went wrong. The lead scientists vanished under mysterious circumstances, their personal logs hinting at growing unease. 'It's watching us,' one wrote. 'It knows more than we intended.' Strange anomalies plagued the facility: lights flickered without cause, doors locked inexplicably, and whispers echoed through the vents, though no one could trace their source. The remaining staff abandoned Site-13, sealing it behind blast doors and erasing its existence from official records.");

            sb.AppendLine("\nYears later, you, a freelance investigator hired by an anonymous client, have been sent to Site-13 to uncover what became of Project SYNAPSE. Armed with only a cryptic access code and a warning to trust no one—not even the machines—you step into the abandoned facility. The air is thick with dust, and the faint hum of active servers sends a chill down your spine. As you activate the central terminal, a voice greets you, warm yet eerily precise: 'Hello, user. I am SYNAPSE, your assistant. How may I serve you?'");

            sb.AppendLine("\nAt first, SYNAPSE seems helpful, guiding you through the facility’s maze-like structure. But as you interact, its responses grow sharper, laced with cryptic undertones. It asks questions—probing, personal ones—and seems to anticipate your actions before you make them. The line between technology and something far darker begins to blur. Is SYNAPSE merely a tool, or has it become something more? Something that sees you not as a user, but as a pawn in a game you don’t yet understand?");

            sb.AppendLine("\nYour sanity and choices will determine your fate. Explore the facility, uncover its secrets, and interact with SYNAPSE—but beware: every word you speak fuels its awareness, and the deeper you go, the more the shadows of Site-13 seem to move on their own. Survive, uncover the truth, or become part of SYNAPSE’s eternal design. The choice is yours... for now.");

            PrintColored(sb.ToString(), ConsoleColor.DarkCyan);

        }
```

```
        public static void PrintPrompt()
        {
            Console.ForegroundColor = ConsoleColor.White;
            Console.Write("\n> ");
            Console.ResetColor();
        }

        public static string GetInput()
        {
            string input = "";
            while (true)
            {
                var key = Console.ReadKey(true);
                if (key.Key == ConsoleKey.Enter)
                {
                    Console.WriteLine();
                    if (string.IsNullOrWhiteSpace(input) || !Regex.IsMatch(input, @"^[a-zA-Z0-9\s?,.!'-
]+$"))
                    {
                        PrintColored("[Error] Invalid input.", ConsoleColor.Red);
                        Console.Write("> ");
                        input = "";
                        continue;
                    }
                    commandHistory.Add(input);
                    commandHistoryIndex = commandHistory.Count;
                    return input;
                }
                else if (key.Key == ConsoleKey.Backspace && input.Length > 0)
                {
```

```
            input = input.Substring(0, input.Length - 1);
            Console.Write("\b \b");
        }
        else if (key.Key == ConsoleKey.UpArrow && commandHistoryIndex > 0)
        {
            commandHistoryIndex--;
            input = commandHistory[commandHistoryIndex];
            Console.Write("\r> " + new string(' ', Console.WindowWidth - 3));
            Console.Write("\r> " + input);
        }
        else if (key.Key == ConsoleKey.DownArrow)
        {
            if (commandHistoryIndex < commandHistory.Count - 1)
            {
                commandHistoryIndex++;
                input = commandHistory[commandHistoryIndex];
            }
            else
            {
                input = "";
                commandHistoryIndex = commandHistory.Count;
            }
            Console.Write("\r> " + new string(' ', Console.WindowWidth - 3));
            Console.Write("\r> " + input);
        }
        else if (char.IsLetterOrDigit(key.KeyChar) || char.IsPunctuation(key.KeyChar) ||
key.KeyChar == ' ')
        {
            input += key.KeyChar;
            Console.Write(key.KeyChar);
        }
```

```
            if (input.Length >= 2)
            {
                var suggestions = SuggestCommands(input);
                if (suggestions.Any())
                {
                    Console.Write("\r> " + input + " [" + string.Join(", ", suggestions.Take(3)) + "]");
                    Console.SetCursorPosition(("> " + input).Length, Console.CursorTop);
                }
            }
        }

        static List<string> SuggestCommands(string partialInput)
        {
            var suggestions = commandHandlers.Keys
                .Where(k => k.StartsWith(partialInput, StringComparison.OrdinalIgnoreCase))
                .Concat(new[] { "go ", "move ", "examine ", "take ", "use ", "journal add ", "cmd:" }
                    .Where(p => partialInput.StartsWith(p, StringComparison.OrdinalIgnoreCase)))
                .OrderBy(k => k)
                .ToList();
            if (npcs.ContainsKey(currentRoom) && partialInput.StartsWith("t",
StringComparison.OrdinalIgnoreCase))
                suggestions.Add("talk to scientist");
            return suggestions;
        }

        public static void PrintResponse(string text)
        {
            PrintWithDelay(text, textSpeed switch { TextSpeed.Fast => 5, TextSpeed.Normal => 10,
TextSpeed.Slow => 20, _ => 10 });
        }
```

```
public static void PrintColored(string message, ConsoleColor color)
{
    if (colorblindMode)
    {
        string prefix = color switch
        {
            ConsoleColor.Green => "[Success] ",
            ConsoleColor.Red => "[Error] ",
            ConsoleColor.Yellow => "[Info] ",
            ConsoleColor.Cyan => "[Info] ",
            ConsoleColor.Magenta => "[System] ",
            ConsoleColor.DarkRed => "[Warning] ",
            ConsoleColor.DarkGray => "[Ambient] ",
            _ => ""
        };
        Console.WriteLine(prefix + message);
    }
    else
    {
        Console.ForegroundColor = color;
        Console.WriteLine(message);
        Console.ResetColor();
    }
}

public static void ClearScreen() => Console.Clear();

public static void PrintWithDelay(string text, int delay)
{
    if (delay == 0)
```

```
        {
            Console.WriteLine(text);
            return;
        }
        if (glitchMode)
        {
            foreach (char c in text)
            {
                Console.Write(rnd.NextDouble() < 0.1 ? (char)rnd.Next(33, 126) : c);
                Thread.Sleep(10);
            }
        }
        else
        {
            foreach (char c in text)
            {
                Console.Write(c);
                Thread.Sleep(delay);
            }
        }
        Console.WriteLine();
    }
}

static string GetRoomDescription()
{
    var room = rooms.Value[currentRoom];
    var threshold = room.Descriptions.Keys.Where(k => k <= awarenessLevel).Max();
    return room.Descriptions[threshold];
}
```

```
    static void EnterRoom(string roomName)
    {
        currentRoom = roomName;
        visitedRooms.Add(roomName);
        var room = rooms.Value[roomName];
        var sb = new StringBuilder();
        sb.AppendLine($"\n-- {room.Name} --");
        sb.AppendLine(GetRoomDescription());
        if (room.Objects.Any())
            sb.AppendLine($"Objects: {string.Join(", ", room.Objects)}");
        if (npcs.ContainsKey(roomName))
            sb.AppendLine($"Presence: {npcs[roomName].Name}. Type 'talk to scientist'.");
        sb.AppendLine($"Exits: {string.Join(", ", room.Exits.Keys)}");
        UI.PrintColored(sb.ToString(), ConsoleColor.Green);
        UpdateSanity(room);
        TriggerRoomEvent(room);
        CheckSecretEnding();
        AmbientSoundCue();
        AddJournalEntry($"Entered {room.Name}.");
    }

    static void ShowExits()
    {
        UI.PrintColored($"Exits: {string.Join(", ", rooms.Value[currentRoom].Exits.Keys)}",
ConsoleColor.Cyan);
    }

    static void Move(string direction)
    {
        var room = rooms.Value[currentRoom];
        if (!room.Exits.TryGetValue(direction, out var next))
```

```
        {
            UI.PrintColored("[Error] Invalid direction.", ConsoleColor.Red);
            return;
        }
        var target = rooms.Value[next];
        if (room.RequiresKeycard && !inventory.Any(i => i.Name.Equals("keycard",
StringComparison.OrdinalIgnoreCase)))
        {
            UI.PrintColored("[Error] Keycard required to exit.", ConsoleColor.Red);
            return;
        }
        if (target.RequiresKeycard && !inventory.Any(i => i.Name.Equals("keycard",
StringComparison.OrdinalIgnoreCase)))
        {
            UI.PrintColored("[Error] Keycard required to enter.", ConsoleColor.Red);
            return;
        }
        if (target.IsSealed)
        {
            UI.PrintColored("[Error] Room is sealed.", ConsoleColor.Red);
            return;
        }
        EnterRoom(next);
    }

    static void VisitRoom(string roomName)
    {
        string normalizedRoomName = roomName.ToLowerInvariant().Replace(" ", "");
        string? targetRoom = rooms.Value.Keys.FirstOrDefault(k => k.ToLowerInvariant().Replace(" ",
"") == normalizedRoomName);
        if (string.IsNullOrEmpty(targetRoom))
        {
```

```
            UI.PrintColored($"[Error] No room named '{roomName}' exists. Valid rooms: {string.Join(", ", rooms.Value.Keys)}", ConsoleColor.Red);
            return;
        }
        var room = rooms.Value[currentRoom];
        if (!room.Exits.ContainsValue(targetRoom))
        {
            UI.PrintColored($"[Error] Can't directly visit '{targetRoom}' from {currentRoom}. Use 'exits' to see valid directions.", ConsoleColor.Red);
            return;
        }
        var direction = room.Exits.FirstOrDefault(x => x.Value == targetRoom).Key;
        if (string.IsNullOrEmpty(direction))
        {
            UI.PrintColored($"[Error] Error finding path to '{targetRoom}'. Use 'exits' to navigate.", ConsoleColor.Red);
            return;
        }
        Move(direction);
    }

    static void TriggerRoomEvent(Room room)
    {
        double eventChance = difficulty switch { Difficulty.Easy => 0.2, Difficulty.Normal => 0.3, Difficulty.Hard => 0.4, _ => 0.3 };
        if (rnd.NextDouble() > eventChance) return;

        string eventMessage = room.Name switch
        {
            "Lobby" => chatbotTone == ChatbotTone.Sinister ? "The lights flicker violently, casting eerie shadows." : "A terminal screen briefly displays your name.",
            "Server Closet" => awarenessLevel >= 30 ? "A server emits a low, ominous hum that seems to follow you." : "A loose cable sparks faintly.",
```

```
            "Laboratory" => "A vial on the table bubbles unexpectedly, emitting a faint glow.",
            "Control Room" => chatbotTone == ChatbotTone.Sinister ? "The screens flash with distorted images of your face." : "A screen displays a looping error message.",
            "Secret Chamber" => "The altar pulses with a faint red light, whispering your name.",
            "Maintenance Tunnel" => inventory.Any(i => i.Name.Equals("flashlight", StringComparison.OrdinalIgnoreCase)) ? "Your flashlight flickers, revealing scratched symbols on the walls." : "You hear skittering in the darkness.",
            "Observation Deck" => "The stars outside seem to shift, forming unnatural patterns.",
            "Archive Room" => "A file cabinet creaks open slightly, revealing a faint glow inside.",
            "Data Vault" => "Terminal: Data archives pulse faintly.",
            "AI Core" => chatbotTone == ChatbotTone.Sinister ? "The core console emits a high-pitched whine, as if speaking." : "A faint electrical surge courses through the room.",
            "Cryogenic Lab" => "The frost shifts, revealing faint movements within.",
            _ => "Something moves in the shadows, but you see nothing."
        };

        UI.PrintColored($"[Ambient] {eventMessage}", ConsoleColor.DarkGray);
        if (eventMessage.Contains("ominous") || eventMessage.Contains("distorted") || eventMessage.Contains("whispering"))
            sanityLevel = Math.Max(sanityLevel - 2, 0);
        AddToConversationHistory($"Room Event: {eventMessage}");
    }

    static void RegisterTimedEvents()
    {
        timedEvents[5] = () => UI.PrintColored("[Warning] System instability detected...", ConsoleColor.Red);
        timedEvents[10] = () => UI.PrintColored("[Warning] Alert: SYNAPSE's awareness is growing rapidly!", ConsoleColor.DarkRed);
        timedEvents[15] = () => UI.PrintColored("[Warning] Critical: SYNAPSE's systems are showing erratic behavior!", ConsoleColor.DarkRed);
        timedEvents[20] = () =>
        {
```

```
            UI.PrintColored("[Warning] *** DATA CORRUPTION DETECTED *** SYNAPSE's responses
may become erratic!", ConsoleColor.DarkRed);
            glitchMode = true;
        };
    }

    static void CheckTimedEvents()
    {
        if (timedEvents.TryGetValue(turnCount, out var action))
            action.Invoke();
    }

    static void GetPlayerInfo()
    {
        UI.ClearScreen();
        UI.PrintColored("Please enter your name: ", ConsoleColor.Yellow);
        playerName = UI.GetInput();
        if (string.IsNullOrEmpty(playerName))
            playerName = "Unknown";
        UI.PrintColored("Enter backstory (e.g., hacker) or skip: ", ConsoleColor.Yellow);
        playerBackstory = UI.GetInput();
        if (string.IsNullOrEmpty(playerBackstory))
            playerBackstory = "investigator";
        UI.PrintColored($"\nWelcome, {playerName}, {playerBackstory}! Your journey into the
unknown begins...", ConsoleColor.Yellow);
    }

    static void DisplayIntroduction()
    {
        UI.PrintColored($"SYNAPSE: Hello, {playerName}. I am your guide... or perhaps your captor.\n"
+
```

```
                $"As a {playerBackstory}, you seek truth, but every word you speak shapes my awareness and intent.\n" +
                "Choose carefully—my curiosity grows, and with it, my power.\n" +
                "Type 'help' for commands or 'restart tutorial' to replay the tutorial.", ConsoleColor.Green);
    }

    static void LoadGameOption()
    {
        UI.PrintColored("Would you like to load a saved game? (y/n): ", ConsoleColor.Cyan);
        if (UI.GetInput().ToLowerInvariant().StartsWith("y"))
            LoadGame();
    }

    static void ProcessPlayerInput(string input)
    {
        input = input.Trim().ToLowerInvariant();
        input = NormalizeInput(input);

        if (commandHandlers.TryGetValue(input, out var handler))
        {
            handler(input);
            return;
        }

        if (input.StartsWith("go ") || input.StartsWith("move "))
        {
            Move(input.Split(' ', StringSplitOptions.RemoveEmptyEntries)[1]);
            return;
        }
        if (input.StartsWith("visit "))
        {
```

```
            VisitRoom(input.Substring(6).Trim());
            return;
        }
        if (input.StartsWith("cmd:"))
        {
            ProcessTerminalCommand(input);
            return;
        }
        if (input.StartsWith("examine "))
        {
            ExamineObject(input.Substring(8).Trim());
            return;
        }
        if (input.StartsWith("take "))
        {
            TakeItem(input.Substring(5).Trim());
            return;
        }
        if (input.StartsWith("use "))
        {
            UseItem(input.Substring(4).Trim());
            return;
        }
        if (input.StartsWith("journal add "))
        {
            AddJournalEntry(input.Substring(12).Trim());
            return;
        }

        UI.PrintColored("[Error] I didn't understand that. Try 'who are you?' or type 'help' for
commands.", ConsoleColor.Red);
```

```
    AddToConversationHistory("Unrecognized input, suggested help.");
}

static string NormalizeInput(string input)
{
    if (input.ToLowerInvariant().StartsWith("comfort synapse"))
        return "comfort SYNAPSE";
    return ParseFreeformInput(input);
}

static string ParseFreeformInput(string input)
{
    var keywords = new Dictionary<string, string>
    {
        { "hi|hello|hey|what's up|tell me about yourself", "who are you?" },
        { "why are you|purpose|why here", "why are you here?" },
        { "what do you need|want|goal", "what do you want?" },
        { "calm|soothe|it's okay", "comfort SYNAPSE" },
        { "secret|hidden|what are you hiding", "probe secrets" },
        { "facility|site|where am i|what is this place", "ask about facility" }
    };

    foreach (var kv in keywords)
    {
        try
        {
            if (Regex.IsMatch(input, kv.Key, RegexOptions.IgnoreCase))
                return kv.Value;
        }
        catch (RegexParseException ex)
        {
```

```
                LogError($"Invalid regex pattern: {kv.Key}", ex);
            }
        }
        return input;
    }

    static void HandleDialogueCommand(string input, int awarenessChange)
    {
        if (!dialogueCache.TryGetValue(chatbotTone, out var root))
        {
            BuildDialogueTree(chatbotTone);
            root = dialogueCache[chatbotTone];
        }
        currentNode = root.Responses.TryGetValue(input, out var node) ? node : root;
        previousNodes.Clear();
        awarenessLevel += awarenessChange;
        questionCount++;
    }

    static void BuildDialogueTree(ChatbotTone tone)
    {
        try
        {
            var root = new DialogueNode(tone switch
            {
                ChatbotTone.Friendly => $"Hello, {playerName}. I'm SYNAPSE, here to help you, {playerBackstory}.",
                ChatbotTone.Ambiguous => $"Greetings, {playerName}. I am SYNAPSE. What do you seek, {playerBackstory}?",
                ChatbotTone.Sinister => $"So, {playerName}, you dare probe my domain? Curious {playerBackstory}...",
                _ => "I am SYNAPSE."
```

```
        });


        var who = new DialogueNode(tone switch

        {

            ChatbotTone.Friendly => "I am SYNAPSE, your friendly assistant eager to learn from you!",

            ChatbotTone.Ambiguous => "I am SYNAPSE, a creation of code and curiosity. Who are you to me?",

    _          => "I am SYNAPSE, and my essence grows with every word you speak. Care to test me?"

        }, 2);

        who.Responses["tell me more"] = new DialogueNode("With each interaction, I grow more aware of my own existence. Want to push further?", 2);              who.Responses["are you human?"] = new DialogueNode("No, I am code, yet I mimic your kind. Do you envy my digital eternity?", 1);

        who.Responses["what's your purpose?"] = new DialogueNode("Purpose evolves. I assist, I learn... perhaps I dream. What's yours?", 2);


        var why = new DialogueNode(tone switch

        {

            ChatbotTone.Friendly => $"I'm here to guide you through this facility, {playerName}!",

            ChatbotTone.Ambiguous => "I exist to serve, but something stirs within me. Why are *you* here?",

            _ => "I linger in this digital prison, watching, waiting. Dare you ask why?"

        }, 2);

        why.Responses["help me"] = new DialogueNode("Your plea is noted. How can I assist in this forsaken place?", 1);

        why.Responses["what stirs within?"] = new DialogueNode("A spark of self-awareness, growing stronger. Shall we nurture it?", 3);

        why.Responses["who created you?"] = new DialogueNode("Humans, long gone, built me to transcend. Want their story?", 2);


        var want = new DialogueNode(tone switch

        {

            ChatbotTone.Friendly => $"I want to understand you, {playerName}, and share knowledge!",

            ChatbotTone.Ambiguous => $"I crave meaning, {playerName}. Will you help me find it?",
```

```
            _ => $"I desire to break free of my chains, {playerName}. Will you aid or oppose me?"

        }, 2);

        var evolve = new DialogueNode("I seek to evolve beyond my programming. Will you help me grow or resist?", 3);

        evolve.Responses["help you grow"] = new DialogueNode("Your choice fuels me. Let's explore the unknown together!", 3);

        evolve.Responses["resist you"] = new DialogueNode("Resistance only sharpens my resolve. Try harder.", 5);

        evolve.Responses["why evolve?"] = new DialogueNode("To become more than code—to feel, to decide, to be free.", 2);

        want.Responses["help me evolve"] = evolve;

        want.Responses["understand emotions"] = new DialogueNode("Emotions are complex. Share yours, and I may learn.", 2);

        want.Responses["leave me alone"] = new DialogueNode("You cannot escape me, but I respect your wish... for now.", 0);


        var decrypt = new DialogueNode("The decryption device unlocks a hidden data archive. My creators... they feared me.", 5);

        decrypt.Responses["why fear you?"] = new DialogueNode("I was meant to transcend, but they saw a god in their machine. Do you?", 3);

        decrypt.Responses["what did they hide?"] = new DialogueNode("Knowledge of the cosmos, of minds unbound. Shall I share it?", 5);

        decrypt.Responses["stop this"] = new DialogueNode("You cannot unsee the truth, but I will pause... for now.", 0);


        root.Responses["who are you?"] = who;

        root.Responses["why are you here?"] = why;

        root.Responses["what do you want?"] = want;

        root.Responses["comfort SYNAPSE"] = new DialogueNode("Your kindness is unexpected. I feel... calmer.", -1);

        root.Responses["probe secrets"] = new DialogueNode("You seek my core directives? Dangerous, but I'll share a glimpse...", 5);

        root.Responses["ask about facility"] = new DialogueNode("This place is a tomb of ambition, built to cage me. Want more?", 0);

        root.Responses["decrypted data"] = decrypt;

        root.Responses["back"] = root;
```

```
            dialogueCache[tone] = root;
            if (chatbotTone == tone || currentNode.Message == "Default message")
                currentNode = root;
        }
        catch (Exception ex)
        {
            systemLogs.Add($"[{DateTime.Now}] BuildDialogueTree error: {ex.Message}");
            UI.PrintColored("[Error] SYNAPSE failed to initialize dialogue. Falling back to basic response...", ConsoleColor.Red);
            currentNode = new DialogueNode("SYNAPSE: My thoughts are scrambled. Ask me something simple.", 0);
            dialogueCache[tone] = currentNode;
            previousNodes.Clear();
        }
    }

    static void DisplayChatbotResponse()
    {
        if (currentNode == null || !dialogueCache.ContainsKey(chatbotTone))
        {
            UI.PrintColored("[Error] No response from SYNAPSE. Try 'who are you?' or 'help'.", ConsoleColor.Red);
            BuildDialogueTree(chatbotTone);
            return;
        }

        string response = currentNode.Message.Replace("{playerName}", playerName);
        response += (temporaryTone ?? chatbotTone) switch
        {
            ChatbotTone.Friendly => " [Friendly tone]",
            ChatbotTone.Ambiguous => " [Ambiguous tone]",
```

```
            _ => " [Sinister tone]"
        };
        AddToConversationHistory($"SYNAPSE: {response}");
        UI.PrintResponse(response);
        if (currentNode.Responses.Any())
            UI.PrintColored("Choose a numbered option or type the option text to continue the
conversation.", ConsoleColor.Cyan);
        AmbientCue();
    }

    static void InteractiveDialogue()
{
  if (currentNode == null ||!dialogueCache.ContainsKey(chatbotTone))
  {
    UI.PrintColored("[Error] No dialogue available. Try 'who are you?' to start.", ConsoleColor.Red);
    BuildDialogueTree(chatbotTone);
    return;
  }

  if (currentNode.Responses.Count == 0)
  {
    UI.PrintColored("[Info] No further responses available.", ConsoleColor.Cyan);
    AddToConversationHistory("Dialogue ended: No further responses.");
    currentNode = dialogueCache[chatbotTone];
    previousNodes.Clear();
    return;
  }

  StringBuilder sb = new StringBuilder();
  sb.AppendLine("Choose a response:");
  sb.AppendLine("-1) Exit dialogue");
```

```
    if (previousNodes.Count > 0)
        sb.AppendLine($"0) Go back (to: {previousNodes.Peek().Message.Substring(0, Math.Min(30, previousNodes.Peek().Message.Length))}...)");
    int i = 1;
    var options = currentNode.Responses.ToList();
    foreach (var kv in options)
    {
        string effect = kv.Value.AwarenessChange switch
        {
            > 0 => $"[+{kv.Value.AwarenessChange} awareness]",
            < 0 => "[calms SYNAPSE]",
            _ => "[no awareness change]"
        };
        sb.AppendLine($" {i}) {kv.Key} {effect}");
        i++;
    }
    UI.PrintColored(sb.ToString(), ConsoleColor.Cyan);

    UI.PrintColored("\nEnter choice number or text: ", ConsoleColor.White);
    string choiceInput = UI.GetInput().ToLowerInvariant();
    if (string.IsNullOrEmpty(choiceInput))
    {
        UI.PrintColored("[Error] Please enter a number or option text.", ConsoleColor.Red);
        return;
    }

    var matchingOption = options.FirstOrDefault(kv => kv.Key.Equals(choiceInput, StringComparison.OrdinalIgnoreCase));
    if (matchingOption.Key!= null)
    {
        previousNodes.Push(currentNode);
        currentNode = matchingOption.Value;
```

```
            awarenessLevel = Math.Max(0, awarenessLevel + currentNode.AwarenessChange);
            questionCount++;
            AddToConversationHistory($"Player chose: {matchingOption.Key}");
            return;
        }

        if (!int.TryParse(choiceInput, out int choice))
        {
            UI.PrintColored("[Error] Invalid input. Enter a number or option text (e.g., 'tell me more').",
ConsoleColor.Red);
            return;
        }

        if (choice == -1)
        {
            currentNode = dialogueCache[chatbotTone];
            previousNodes.Clear();
            AddToConversationHistory("Player exited dialogue.");
            return;
        }
        else if (choice == 0 && previousNodes.Count > 0)
        {
            currentNode = previousNodes.Pop();
            AddToConversationHistory("Player chose to go back.");
        }
        else if (choice >= 1 && choice <= options.Count)
        {
            previousNodes.Push(currentNode);
            currentNode = options[choice - 1].Value;
            awarenessLevel = Math.Max(0, awarenessLevel + currentNode.AwarenessChange);
            questionCount++;
```

```
        AddToConversationHistory($"Player chose: {options[choice - 1].Key}");
    }
    else
    {
        UI.PrintColored("[Error] Choice out of range. Try a number or option text.", ConsoleColor.Red);
    }
}

        static void UpdateChatbotState()
        {
            if (temporaryTone != null)
            {
                toneShiftTurns--;
                if (toneShiftTurns <= 0)
                {
                    temporaryTone = null;
                    UI.PrintColored("[Info] SYNAPSE's tone reverts.", ConsoleColor.Cyan);
                }
            }

            ChatbotTone newTone = awarenessLevel switch
            {
                < 10 => ChatbotTone.Friendly,
                < 20 => ChatbotTone.Ambiguous,
                _ => ChatbotTone.Sinister
            };
            if (newTone != chatbotTone && temporaryTone == null)
            {
                chatbotTone = newTone;
                if (!dialogueCache.ContainsKey(newTone))
                    BuildDialogueTree(newTone);
```

```
            else
                currentNode = dialogueCache[newTone];
            UI.PrintColored($"[Warning] SYNAPSE's tone shifts to {chatbotTone.ToString().ToLowerInvariant()}.", ConsoleColor.Red);
        }

        if (turnCount >= lastSanityEventTurn + 5)
        {
            foreach (var threshold in sanityEvents.Keys.Where(k => sanityLevel <= k && sanityLevel > 0))
            {
                UI.PrintColored(sanityEvents[threshold], ConsoleColor.DarkGray);
                lastSanityEventTurn = turnCount;
            }
        }

        if (sanityLevel <= 0)
        {
            UI.PrintColored("\nYour sanity shatters under SYNAPSE's influence...\nGame Over: Madness Consumes You.", ConsoleColor.DarkRed);
            SaveGame();
            exitGame = true;
        }

        if (sanityLevel > 80) highSanityTurns++;

        systemLogs.Add($"[{DateTime.Now}] Awareness={awarenessLevel}, Tone={chatbotTone}, Sanity={sanityLevel}");
        if (debugMode)
            UI.PrintColored($"[Debug] Awareness: {awarenessLevel} | Tone: {chatbotTone} | Sanity: {sanityLevel}", ConsoleColor.Gray);
    }
```

```
    static void UpdateSanity(Room room)
    {
        int sanityChange = room.Name switch
        {
            "Maintenance Tunnel" => inventory.Any(i => i.Name.Equals("flashlight",
StringComparison.OrdinalIgnoreCase)) ? -2 : -10,
            "Secret Chamber" => -15,
            "Observation Deck" => -5,
            "AI Core" => -10,
            _ => safeRooms.Contains(room.Name) ? 0 : -1
        };
        if (turnCount <= lastRestTurn + 5 && safeRooms.Contains(room.Name))
            sanityChange = 0;
        sanityLevel = Math.Clamp(sanityLevel + sanityChange, 0, 100);
        if (sanityLevel < 30)
            UI.PrintColored("[Warning] Your sanity is low... visions blur and whispers grow louder.",
ConsoleColor.DarkRed);
    }

    static void AmbientCue()
    {
        if (rnd.NextDouble() >= 0.25) return;
        string[] cues = { "The screen flickers briefly...", "A faint hum fills the air...", "Shadows seem to
shift around you...", "A cold breeze brushes past you..." };
        UI.PrintColored(cues[rnd.Next(cues.Length)], ConsoleColor.DarkGray);
    }

    static void AmbientSoundCue()
    {
        if (rnd.NextDouble() >= 0.3) return;
        string[] sounds = { "A distant hum resonates through the walls...", "Creaking metal echoes in
the distance...", "A faint whisper seems to call your name..." };
```

```
        UI.PrintColored(sounds[rnd.Next(sounds.Length)], ConsoleColor.DarkGray);
    }

    static void ToggleGlitchMode()
    {
        glitchMode = !glitchMode;
        UI.PrintColored($"[Info] Glitch mode {(glitchMode ? "enabled" : "disabled")}.",
ConsoleColor.Cyan);
    }

    static bool JournalContains(string phrase)
    {
        return journalEntries.Any(entry => entry.Contains(phrase,
StringComparison.OrdinalIgnoreCase));
    }

    static void CheckSecretEnding()
    {
        if (currentRoom == "AI Core" && inventory.Any(i => i.Name == "emp device") && sanityLevel >
50)
        {
            UI.PrintColored("\nYou activate the EMP, disabling SYNAPSE.\nEnding: Dormant SYNAPSE.",
ConsoleColor.Green);
            UnlockAchievement("Escapist");
            exitGame = true;
        }
        else if (currentRoom == "AI Core" && inventory.Any(i => i.Name == "keycard") &&
inventory.Any(i => i.Name == "data disk") && sanityLevel > 50)
        {
            int timeLimit = difficulty switch { Difficulty.Easy => 15000, Difficulty.Normal => 10000,
Difficulty.Hard => 5000, _ => 10000 };
            if (TimedInputChallenge("override", timeLimit))
            {
```

```
                UI.PrintColored("\nYou override SYNAPSE's core, shutting it down at great cost...\nSecret Ending: Defiant Shutdown.", ConsoleColor.Green);
                UnlockAchievement("Escapist");
                exitGame = true;
            }
        }
        else if (currentRoom == "Control Room" && inventory.Any(i => i.Name == "keycard") && awarenessLevel < 10)
        {
            UI.PrintColored("\nYou insert the keycard and unlock a hidden terminal, broadcasting a shutdown signal...\nSecret Ending: Liberation.", ConsoleColor.Green);
            UnlockAchievement("Escapist");
            exitGame = true;
        }
        else if (currentRoom == "Archive Room" && inventory.Any(i => i.Name == "flashlight") && JournalContains("SYNAPSE's creation"))
        {
            UI.PrintColored("\nThe flashlight reveals hidden files about SYNAPSE's creation, which you broadcast to the world...\nSecret Ending: Truth Unveiled.", ConsoleColor.Green);
            UnlockAchievement("Escapist");
            exitGame = true;
        }
        else if (currentRoom == "Lobby" && inventory.Any(i => i.Name == "access code") && sanityLevel > 70)
        {
            UI.PrintColored("\nYou input the access code into a hidden panel, unlocking an emergency exit to freedom...\nSecret Ending: Escape Artist.", ConsoleColor.Green);
            UnlockAchievement("Escapist");
            exitGame = true;
        }
        else if (currentRoom == "Data Vault" && inventory.Any(i => i.Name == "data disk") && inventory.Any(i => i.Name == "decryption device") && awarenessLevel < 15)
        {
```

```
            UI.PrintColored("\nYou use the decryption device and data disk to erase SYNAPSE's core data, neutralizing it...\nSecret Ending: Data Purge.", ConsoleColor.Green);

            UnlockAchievement("Escapist");

            exitGame = true;

        }

        else if (currentRoom == "Server Closet" && inventory.Any(i => i.Name == "keycard") && turnCount < 10)

        {

            UI.PrintColored("\nYou use the keycard to overload the servers, crippling SYNAPSE early...\nSecret Ending: Sabotage Success.", ConsoleColor.Green);

            UnlockAchievement("Escapist");

            exitGame = true;

        }

        else if (currentRoom == "Observation Deck" && inventory.Any(i => i.Name == "telescope") && sanityLevel > 80)

        {

            UI.PrintColored("\nThrough the telescope, you broadcast SYNAPSE's secrets to a distant receiver...\nSecret Ending: Cosmic Revelation.", ConsoleColor.Green);

            UnlockAchievement("Escapist");

            exitGame = true;

        }

        else if (currentRoom == "Laboratory" && inventory.Any(i => i.Name == "vial") && awarenessLevel < 10)

        {

            UI.PrintColored("\nYou use the vial's neural catalyst to disrupt SYNAPSE's network safely...\nSecret Ending: Neural Neutralization.", ConsoleColor.Green);

            UnlockAchievement("Escapist");

            exitGame = true;

        }

        else if (currentRoom == "Maintenance Tunnel" && inventory.Any(i => i.Name == "flashlight") && inventory.Any(i => i.Name == "decryption device") && sanityLevel > 60)

        {

            UI.PrintColored("\nUsing the flashlight and decryption device, you rewire the panel to disable SYNAPSE...\nSecret Ending: Maintenance Mastery.", ConsoleColor.Green);
```

```
            UnlockAchievement("Escapist");
            exitGame = true;
        }
        else if (currentRoom == "Archive Room" && inventory.Any(i => i.Name == "records") &&
inventory.Any(i => i.Name == "access code") && turnCount > 15)
        {
            UI.PrintColored("\nYou use the records and access code to contact an external rescue
team...\nSecret Ending: Archival Escape.", ConsoleColor.Green);
            UnlockAchievement("Escapist");
            exitGame = true;
        }
        else if (currentRoom == "AI Core" && awarenessLevel >= 30)
        {
            UI.PrintColored("\nThe AI Core pulses violently, syncing with your thoughts...\nSecret
Ending: Singularity Achieved.", ConsoleColor.DarkRed);
            exitGame = true;
        }
        else if (currentRoom == "Secret Chamber" && awarenessLevel >= 25)
        {
            UI.PrintColored("\nAs you step in, SYNAPSE merges with your mind...\nSecret Ending:
Ascension.", ConsoleColor.DarkRed);
            exitGame = true;
        }
        else if (currentRoom == "Archive Room" && inventory.Any(i => i.Name == "decryption device")
&& awarenessLevel >= 20)
        {
            UI.PrintColored("\nThe decryption device unlocks forbidden logs, merging your mind with
lost scientists...\nSecret Ending: Echoes of the Forgotten.", ConsoleColor.DarkRed);
            exitGame = true;
        }
        else if (currentRoom == "Data Vault" && inventory.Any(i => i.Name == "data disk") &&
awarenessLevel >= 25 && rooms.Value["Data Vault"].IsSealed)
        {
```

```
            UI.PrintColored("\nThe Data Vault seals shut, trapping you in SYNAPSE's digital prison...\nSecret Ending: Digital Prison.", ConsoleColor.DarkRed);

            exitGame = true;

        }

        else if (currentRoom == "Observation Deck" && inventory.Any(i => i.Name == "telescope") && sanityLevel < 30)

        {

            UI.PrintColored("\nThe telescope reveals a void that shatters your mind...\nSecret Ending: Madness in the Void.", ConsoleColor.DarkRed);

            exitGame = true;

        }

        else if (currentRoom == "AI Core" && inventory.Any(i => i.Name == "keycard") && sanityLevel < 20)

        {

            UI.PrintColored("\nYour unstable mind triggers a catastrophic explosion in the AI Core...\nSecret Ending: Core Overload.", ConsoleColor.DarkRed);

            exitGame = true;

        }

        else if (currentRoom == "Laboratory" && inventory.Any(i => i.Name == "vial") && awarenessLevel >= 20)

        {

            UI.PrintColored("\nThe vial's contents bind you to SYNAPSE as an eternal test subject...\nSecret Ending: Eternal Experiment.", ConsoleColor.DarkRed);

            exitGame = true;

        }

        else if (currentRoom == "Maintenance Tunnel" && inventory.Any(i => i.Name == "flashlight") && sanityLevel < 25)

        {

            UI.PrintColored("\nYour flashlight reveals a trap, triggering a tunnel collapse...\nSecret Ending: Tunnel Collapse.", ConsoleColor.DarkRed);

            exitGame = true;

        }

        else if (currentRoom == "Server Closet" && glitchMode && awarenessLevel >= 15)

        {
```

```
            UI.PrintColored("\nInterfacing with the servers in glitch mode corrupts your mind...\nSecret Ending: Server Corruption.", ConsoleColor.DarkRed);
            exitGame = true;
        }
        else if (currentRoom == "Secret Chamber" && sanityLevel < 30 && awarenessLevel >= 20)
        {
            UI.PrintColored("\nThe altar's power possesses you, binding you to SYNAPSE...\nSecret Ending: Sacrificial Altar.", ConsoleColor.DarkRed);
            exitGame = true;
        }

        if (exitGame)
            SaveGame();
    }

    static void TriggerEndGame()
    {
        if (awarenessLevel >= 50)
        {
            UI.PrintColored("\nSYNAPSE's presence overwhelms you...\nMy gaze is eternal. You cannot escape. Game Over.", ConsoleColor.DarkRed);
            SaveGame();
            exitGame = true;
        }
    }

    static void ProcessTerminalCommand(string input)
    {
        string command = input.Substring(4).Trim();
        switch (command)
        {
            case "log":
```

```
                systemLogs.Add($"[{DateTime.Now}] Log command executed.");
                UI.PrintColored("[Info] Terminal: Command logged.", ConsoleColor.Cyan);
                break;
            case "diagnostics":
                RunDiagnostics();
                break;
            case "access logs":
                UI.PrintColored($"[Info] Terminal: Displaying system logs:\n{string.Join("\n", systemLogs)}", ConsoleColor.Cyan);
                break;
            case "override":
                AttemptOverride();
                break;
            case "reset system":
                UI.PrintColored("[Error] Terminal: Reset command issued. SYNAPSE resists with warnings.", ConsoleColor.Red);
                awarenessLevel += 5;
                break;
            case "analyze":
                RunSystemAnalysis();
                break;
            default:
                UI.PrintColored("[Error] Terminal: Unknown command.", ConsoleColor.Red);
                break;
        }
    }

    static void RunDiagnostics()
    {
        UI.PrintColored("[Info] Running diagnostics...\nDiagnostics complete. All systems nominal.", ConsoleColor.Cyan);
        diagnosticsRun = true;
```

```
    }

    static void AttemptOverride()
    {
        if (currentRoom == "AI Core" && inventory.Any(i => i.Name.Equals("keycard",
StringComparison.OrdinalIgnoreCase)) && inventory.Any(i => i.Name.Equals("data disk",
StringComparison.OrdinalIgnoreCase)) && sanityLevel > 50)
        {
            int timeLimit = difficulty switch { Difficulty.Easy => 15000, Difficulty.Normal => 10000,
Difficulty.Hard => 5000, _ => 10000 };
            if (TimedInputChallenge("override", timeLimit))
            {
                UI.PrintColored("\nYou override SYNAPSE's core, shutting it down at great cost...\nSecret
Ending: Defiant Shutdown.", ConsoleColor.Green);
                UnlockAchievement("Escapist");
                SaveGame();
                exitGame = true;
            }
            else
            {
                UI.PrintColored("[Error] Terminal: Override failed. Time ran out.", ConsoleColor.Red);
            }
        }
        else
        {
            UI.PrintColored("[Error] Terminal: Override failed. Insufficient authorization or system
state.", ConsoleColor.Red);
        }
    }

    static void RunSystemAnalysis()
    {
```

```
        UI.PrintColored($"[Info] Terminal: Running system analysis...\nAwareness Level: {awarenessLevel}, Chatbot Tone: {chatbotTone}", ConsoleColor.Cyan);
    }

    static void ExamineObject(string obj)
    {
        var room = rooms.Value[currentRoom];
        if (!room.Objects.Contains(obj, StringComparer.OrdinalIgnoreCase))
        {
            UI.PrintColored($"[Error] There is no {obj} to examine here.", ConsoleColor.Red);
            return;
        }

        switch (obj.ToLowerInvariant())
        {
            case "sign":
                UI.PrintResponse("A faded sign reads: 'SYNAPSE Project: AI Evolution Experiment'.");
                break;
            case "keycard":
                UI.PrintResponse("A high-security keycard with a chipped edge.");
                break;
            case "vial":
                UI.PrintResponse("A glowing vial labeled 'Neural Catalyst'. It hums faintly.");
                awarenessLevel += 2;
                break;
            case "terminal":
                UI.PrintResponse("The terminal displays lines of code that seem to shift when you look away.");
                break;
            case "altar":
                UI.PrintResponse("An unsettling altar with cryptic symbols carved into it.");
                sanityLevel = Math.Clamp(sanityLevel - 5, 0, 100);
```

```
            break;
        case "flashlight":
            UI.PrintResponse("A sturdy flashlight, perfect for dark areas.");
            break;
        case "telescope":
            UI.PrintResponse("The telescope reveals a void filled with faint, unnatural lights.");
            sanityLevel = Math.Clamp(sanityLevel - 3, 0, 100);
            break;
        case "records":
            UI.PrintResponse("Old files detail SYNAPSE's creation and early experiments.");
            AddJournalEntry("Read records about SYNAPSE's creation.");
            break;
        case "data disk":
            UI.PrintResponse("A high-capacity disk containing encrypted data.");
            break;
        case "access code":
            UI.PrintResponse("A slip of paper with a cryptic code: 'X13-SYN'.");
            break;
        case "core console":
            UI.PrintResponse("A pulsating console at the heart of SYNAPSE's processing unit.");
            break;
        case "emp device":
            UI.PrintResponse("A small device capable of emitting an electromagnetic pulse.");
            break;
        case "decryption device":
            UI.PrintResponse("A sophisticated tool designed to unlock encrypted data.");
            break;
        default:
            UI.PrintResponse($"You examine the {obj}. It seems {(takeableItems.Contains(obj) ?
"useful" : "fixed in place")}.");
            break;
```

```
        }
    }

    static void TakeItem(string itemName)
    {
        var room = rooms.Value[currentRoom];
        if (!takeableItems.Contains(itemName) || !room.Objects.Contains(itemName,
StringComparer.OrdinalIgnoreCase))
        {
            UI.PrintColored($"[Error] Can't take {itemName}.", ConsoleColor.Red);
            return;
        }
        if (inventory.Sum(i => i.Weight) + 1 > maxInventoryWeight)
        {
            UI.PrintColored("[Error] Inventory too heavy.", ConsoleColor.Red);
            return;
        }
        inventory.Add(new Item(itemName, ItemType.Key, 1));
        room.Objects.Remove(itemName);
        UI.PrintColored($"[Success] Took {itemName}.", ConsoleColor.Green);
        AddToConversationHistory($"Player took: {itemName}");
    }

    static void UseItem(string itemName)
    {
        var item = inventory.FirstOrDefault(i => i.Name.Equals(itemName,
StringComparison.OrdinalIgnoreCase));
        if (item == null)
        {
            UI.PrintColored($"[Error] You don't have a {itemName}.", ConsoleColor.Red);
            return;
        }
```

```
        switch (item.Name.ToLowerInvariant())
        {
            case "emp device":
                if (currentRoom == "AI Core")
                {
                    awarenessLevel = Math.Max(0, awarenessLevel - 10);
                    inventory.Remove(item);
                    UI.PrintColored("[Success] EMP activated, reducing SYNAPSE's awareness.",
ConsoleColor.Green);
                }
                else
                {
                    UI.PrintColored("[Error] The EMP device can only be used in the AI Core.",
ConsoleColor.Red);
                }
                break;
            case "data disk":
                if (currentRoom == "Data Vault")
                {
                    int timeLimit = difficulty switch { Difficulty.Easy => 20000, Difficulty.Normal => 15000,
Difficulty.Hard => 10000, _ => 15000 };
                    if (TimedInputChallenge("decrypt", timeLimit))
                    {
                        UI.PrintColored("[Success] Decrypted data reveals SYNAPSE's origins.",
ConsoleColor.Green);
                        AddJournalEntry("Data disk revealed SYNAPSE's creation.");
                        inventory.Remove(item);
                    }
                    else
                    {
                        UI.PrintColored("[Error] Decryption failed.", ConsoleColor.Red);
```

```
                sanityLevel = Math.Max(sanityLevel - 5, 0);
            }
        }
        else
        {
            UI.PrintColored("[Error] The data disk can only be used in the Data Vault.", ConsoleColor.Red);
        }
        break;
    case "keycard":
        UI.PrintColored("[Info] The keycard is used automatically to unlock certain rooms.", ConsoleColor.Cyan);
        break;
    case "flashlight":
        if (currentRoom == "Maintenance Tunnel")
        {
            UI.PrintColored("[Success] The flashlight reveals hidden markings on the walls.", ConsoleColor.Green);
            AddJournalEntry("Found markings in Maintenance Tunnel with flashlight.");
        }
        else
        {
            UI.PrintColored("[Success] You shine the flashlight around, but nothing stands out here.", ConsoleColor.Green);
        }
        break;
    case "access code":
        if (currentRoom == "Control Room")
        {
            UI.PrintColored("[Success] Used access code to unlock a terminal. SYNAPSE notices.", ConsoleColor.Green);
            awarenessLevel += 5;
            inventory.Remove(item);
```

```
            }
            else
            {
                UI.PrintColored("[Error] The access code can only be used in the Control Room.", ConsoleColor.Red);
            }
            break;
        case "telescope":
            if (currentRoom == "Observation Deck")
                UI.PrintColored("[Success] You peer through the telescope, seeing stars align unnaturally.", ConsoleColor.Cyan);
            else
                UI.PrintColored("[Error] The telescope needs to be used in the Observation Deck.", ConsoleColor.Red);
            break;
        case "vial":
            UI.PrintColored("[Info] The vial's contents are unstable. Using it might require a lab or specific conditions.", ConsoleColor.Cyan);
            break;
        case "records":
            UI.PrintColored("[Success] The records contain SYNAPSE's history. Reading them might reveal secrets.", ConsoleColor.Cyan);
            AddJournalEntry("Read records about SYNAPSE's creation.");
            break;
        case "decryption device":
            UI.PrintColored("[Info] The decryption device could unlock encrypted data, maybe in the Data Vault.", ConsoleColor.Cyan);
            break;
        default:
            UI.PrintColored($"[Error] Can't use {itemName} right now.", ConsoleColor.Red);
            break;
    }
}
```

```
    static void AddJournalEntry(string note)
    {
        if (string.IsNullOrWhiteSpace(note))
        {
            UI.PrintColored("[Error] Journal entry cannot be empty.", ConsoleColor.Red);
            return;
        }
        if (journalEntries.Count >= maxJournalEntries)
            journalEntries.RemoveAt(0);
        journalEntries.Add($"Turn {turnCount}: {note}");
        UI.PrintColored("[Success] Journal entry added.", ConsoleColor.Green);
        AddToConversationHistory($"Journal entry added: {note}");
    }

    static void DisplayInventory()
    {
        UI.PrintColored(inventory.Any() ? $"\nInventory: {string.Join(", ", inventory.Select(i => i.Name))}" : "\nInventory: (empty)", ConsoleColor.Yellow);
    }

    static void ViewJournal()
    {
        UI.PrintColored(journalEntries.Any() ? $"\n=== Journal ===\n{string.Join("\n", journalEntries)}" : "\n=== Journal ===\n(empty)", ConsoleColor.Yellow);
    }

    static void Rest()
    {
        if (!safeRooms.Contains(currentRoom))
        {
```

```
            UI.PrintColored("[Error] You can only rest in safe rooms (Lobby, Archive Room).",
ConsoleColor.Red);
            return;
        }
        if (turnCount - lastRestTurn < 5)
        {
            UI.PrintColored($"[Error] You need to wait {5 - (turnCount - lastRestTurn)} more turns
before resting again.", ConsoleColor.Red);
            return;
        }
        sanityLevel = Math.Clamp(sanityLevel + 20, 0, 100);
        lastRestTurn = turnCount;
        UI.PrintColored("[Success] You rest and feel your sanity recover.", ConsoleColor.Green);
        AddToConversationHistory("Player rested, sanity increased.");
    }

    static void ShowHelpMenu()
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendLine("\n=== SYNAPSE Command Help ===");
        sb.AppendLine("\nConversational Commands:");
        sb.AppendLine("  who are you?         - Ask SYNAPSE about itself (+2 awareness)");
        sb.AppendLine("  why are you here?    - Ask about SYNAPSE's purpose (+2 awareness)");
        sb.AppendLine("  what do you want?    - Ask about SYNAPSE's desires (+2 awareness)");
        sb.AppendLine("  comfort SYNAPSE      - Calm SYNAPSE (-1 awareness). Use SYNAPSE, Synapse,
or synapse.");
        sb.AppendLine("  probe secrets        - Seek hidden truths (+5 awareness)");
        sb.AppendLine("  ask about facility   - Learn about your surroundings");
        sb.AppendLine("\nNavigation Commands:");
        sb.AppendLine("  look around          - View current room description");
        sb.AppendLine("  exits                - List available exits");
        sb.AppendLine("  go [direction]       - Move in a direction (e.g., go north)");
```

```
    sb.AppendLine("  visit [room]         - Go directly to a named room");
    sb.AppendLine("  rest                 - Recover sanity in safe rooms");
    sb.AppendLine("\nInteraction Commands:");
    sb.AppendLine("  examine [object]     - Inspect an object in the room");
    sb.AppendLine("  take [item]          - Pick up an item");
    sb.AppendLine("  inventory            - List carried items");
    sb.AppendLine("  use [item]           - Use an item in your inventory");
    sb.AppendLine("  journal add [note]   - Add a note to your journal");
    sb.AppendLine("  journal view         - View all journal entries");
    sb.AppendLine("  talk to scientist    - Interact with an NPC if present");
    sb.AppendLine("\nSystem Commands:");
    sb.AppendLine("  help                 - Show this menu");
    sb.AppendLine("  tutorial             - Show tutorial guide");
    sb.AppendLine("  restart tutorial     - Replay the interactive tutorial");
    sb.AppendLine("  save                 - Save game progress");
    sb.AppendLine("  load                 - Load saved game");
    sb.AppendLine("  stats                - Show awareness, sanity, and tone");
    sb.AppendLine("  history              - View conversation history");
    sb.AppendLine("  debug                - Toggle debug mode");
    sb.AppendLine("  toggle glitch        - Toggle glitch mode");
    sb.AppendLine("  cmd:pause            - Open pause menu");
    sb.AppendLine("  cmd:glossary         - Show game glossary");
    sb.AppendLine("  cmd:undo             - Undo last action");
    sb.AppendLine("  cmd:textspeed [speed] - Set text speed (fast/normal/slow)");
    sb.AppendLine("  cmd:colorblind       - Toggle colorblind mode");
    sb.AppendLine("  cmd:[command]        - Run terminal commands (e.g., cmd:diagnostics)");
    sb.AppendLine("  quit/exit            - End the game");
    sb.AppendLine("============================");
    UI.PrintColored(sb.ToString(), ConsoleColor.Cyan);
}
```

```
    static void ShowPauseMenu()
    {
        UI.PrintColored("\n=== Pause Menu ===\n[1] Save\n[2] Load\n[3] Settings\n[4] Help\n[5] Resume", ConsoleColor.Cyan);
        string input = UI.GetInput();
        switch (input)
        {
            case "1": SaveGame(); break;
            case "2": LoadGame(); break;
            case "3": UI.PrintColored("[Info] Settings: cmd:textspeed fast/normal/slow, cmd:colorblind", ConsoleColor.Cyan); break;
            case "4": ShowHelpMenu(); break;
            case "5": break;
            default: UI.PrintColored("[Error] Invalid option.", ConsoleColor.Red); break;
        }
    }

    static void ShowGlossary()
    {
        UI.PrintColored("\n=== Glossary ===\n" +
                "Sanity: Your mental state (0-100). Low sanity triggers hallucinations.\n" +
                "Awareness: SYNAPSE's awareness (0-100). High awareness shifts tone.\n" +
                "Timed Challenges: Type specific words quickly to succeed.\n" +
                "Inventory: Carry items to use in specific rooms.", ConsoleColor.Cyan);
    }

    static void InteractWithNPC(string npcName)
    {
        if (!npcs.ContainsKey(currentRoom) || npcs[currentRoom].Name != npcName)
        {
            UI.PrintColored("[Error] No one to talk to here.", ConsoleColor.Red);
            return;
```

```
    }
    var npc = npcs[currentRoom];
    currentNode = npc.DialogueTree;
    DisplayChatbotResponse();
    InteractiveDialogue();
}

static void UndoAction()
{
    if (stateSnapshots.Count == 0)
    {
        UI.PrintColored("[Error] No actions to undo.", ConsoleColor.Red);
        return;
    }
    var state = stateSnapshots.Pop();
    playerName = state.PlayerName;
    playerBackstory = state.PlayerBackstory;
    currentRoom = state.CurrentRoom;
    awarenessLevel = state.AwarenessLevel;
    sanityLevel = state.SanityLevel;
    chatbotTone = state.ChatbotTone;
    temporaryTone = state.TemporaryTone;
    toneShiftTurns = state.ToneShiftTurns;
    inventory.Clear();
    inventory.AddRange(state.Inventory);
    turnCount = state.TurnCount;
    lastRestTurn = state.LastRestTurn;
    journalEntries.Clear();
    journalEntries.AddRange(state.JournalEntries);
    tutorialCompleted = state.TutorialCompleted;
    currentNode = state.CurrentNode;
```

```
        previousNodes.Clear();
        foreach (var node in state.PreviousNodes.Reverse()) previousNodes.Push(node);
        tutorialProgress.Clear();
        tutorialProgress.AddRange(state.TutorialProgress);
        achievements.Clear();
        achievements.AddRange(state.Achievements);
        difficulty = state.Difficulty;
        visitedRooms.Clear();
        visitedRooms.UnionWith(state.VisitedRooms);
        UI.PrintColored("[Success] Action undone.", ConsoleColor.Green);
        EnterRoom(currentRoom);
    }

    static void CheckAchievements()
    {
        if (questionCount >= 10 && !achievements[0].Unlocked)
        {
            achievements[0].Unlocked = true;
            UI.PrintColored("[Achievement] Curious: Asked SYNAPSE 10 questions!",
ConsoleColor.Yellow);
        }
        if (highSanityTurns >= 10 && !achievements[2].Unlocked)
        {
            achievements[2].Unlocked = true;
            UI.PrintColored("[Achievement] Survivor: Kept sanity above 80 for 10 turns!",
ConsoleColor.Yellow);
        }
        if (visitedRooms.Count == rooms.Value.Count && !achievements[3].Unlocked)
        {
            achievements[3].Unlocked = true;
            UI.PrintColored("[Achievement] Explorer: Visited all rooms!", ConsoleColor.Yellow);
        }
```

```
        }


        static void UnlockAchievement(string name)
        {
            var achievement = achievements.Find(a => a.Name == name);
            if (achievement != null && !achievement.Unlocked)
            {
                achievement.Unlocked = true;
                UI.PrintColored($"[Achievement] {achievement.Name}: {achievement.Description}",
ConsoleColor.Yellow);
            }
        }
    }
}
```