Spin-a-Yarn: Unity Yarn Spinner Dialogue Project

A Unity sample project demonstrating dialogue systems using **Yarn Spinner 3.1.0** with interactive NPCs, quest mechanics, and player interaction patterns.

Project Overview

This project contains three progressive scenes that teach Yarn Spinner dialogue integration:

1. **Hello World Scene** - Basic dialogue introduction
2. **Get a Coffee Scene** - Interactive choice-based dialogue with variables
3. **Quest Scene** - Multi-NPC quest system with state management

Project Structure

```
Assets/_game_dkcharles/
├── Dialogue/           # Yarn Spinner dialogue files
|   ├── MainDialogueProject.yarnproject
|   ├── Start.yarn
|   ├── GetCoffee.yarn
|   ├── Person1.yarn
|   ├── Person2.yarn
|   └── QuestVariables.yarn
├── Scripts/            # Three main C# controller scripts
|   ├── DialoguePlayerController.cs
|   ├── NPCInteraction.cs
|   └── CoffeeTrigger.cs
├── Scenes/             # Three playable scenes
|   ├── 1_HelloWorld.unity
|   ├── 2_GetaCoffee.unity
|   └── 3_SimpleTask.unity
└── Materials/          # Visual materials for NPCs and objects
```

The Three Main Scripts

1. DialoguePlayerController.cs

Purpose: Manages player movement state during dialogue sequences

Functionality:

- Disables player movement when dialogue starts
- Re-enables controls when dialogue completes
- Integrates with Unity's StarterAssets ThirdPersonController
- Optionally freezes player position and camera look

Key Methods:

- `DisablePlayerMovement()` - Called on dialogue start
- `EnablePlayerMovement()` - Called on dialogue complete

Configuration:

- `disableCameraLookDuringDialogue` - Lock camera during dialogue
- `freezePlayerPosition` - Prevent all movement during dialogue

Integration:
Connected to DialogueRunner's `onDialogueStart` and `onDialogueComplete` events.

---

2. NPCInteraction.cs

Purpose: Proximity-based NPC interaction with manual trigger

Functionality:

- Detects when player enters/exits NPC interaction range
- Shows/hides interaction indicator UI
- Requires player to press Space bar to start dialogue
- Prevents jump action conflicts during interaction

Key Methods:

- `OnTriggerEnter()` - Player enters NPC range, shows prompt
- `OnTriggerExit()` - Player leaves range, hides prompt
- `StartDialogue()` - Initiates dialogue with specified Yarn node

Configuration:

- `dialogueNode` - Name of Yarn node to start (e.g., "Person1_Talk")
- `interactionKey` - Key to press for interaction (default: Space)
- `interactionIndicator` - GameObject to show when in range

Usage:
Attached to NPC gameobjects (Person_1, Person_2) in the quest scene.

---

3. CoffeeTrigger.cs (DialogueTrigger)

Purpose: Automatic dialogue triggering on zone entry

Functionality:

- Automatically starts dialogue when player enters trigger zone
- No button press required
- Prevents re-triggering until player exits and re-enters

Key Methods:

- `OnTriggerEnter()` - Auto-starts dialogue
- `OnTriggerExit()` - Resets trigger state

Configuration:

- `nodeName` - Yarn node to start (default: "GetCoffee")
- `dialogueRunner` - Reference to DialogueRunner component

Usage:
Attached to coffee shop entry zone for automatic barista interaction.

The Three Scenes

Scene 1: 1_HelloWorld.unity

Purpose: Introduction to basic Yarn Spinner dialogue

Content:

- Uses Start.yarn
- Simple welcome message: "Welcome to this demo! Press space to continue"
- Demonstrates basic dialogue flow

Learning Goals:

- Understand Yarn node structure
- See basic DialogueRunner integration
- Experience simple continue-style dialogue

Scene 2: 2_GetaCoffee.unity

Purpose: Choice-based dialogue with variable storage

Content:

- Uses GetCoffee.yarn
- Coffee shop with barista NPC
- Automatic dialogue trigger on entry
- Three drink choices: Latte, Tea, or Special Caramel Macchiato

Dialogue Flow:

```
Barista_Greeting
├─ Choose Latte → $drink_ordered = "latte"
├─ Choose Tea → $drink_ordered = "tea"
└─ Choose Caramel Macchiato → $drink_ordered = "special caramel
macchiato"
    ↓
Barista_ServeDrink (displays: "Here is your {$drink_ordered}")
```

Learning Goals:

- Dialogue choices with options
- Variable assignment and storage
- Variable interpolation in dialogue
- Conditional node jumping

Scene 3: 3_SimpleTask.unity

Purpose: Multi-NPC quest system with state management

Content:

- Two NPCs: Person_1 (quest giver) and Person_2 (quest receiver)
- Message delivery quest

- Persistent state tracking across multiple conversations
- Uses Person1.yarn, Person2.yarn, and QuestVariables.yarn

Quest Flow:

```
1. Player approaches Person_1, presses Space
2. Person_1 offers to help
3. Accept quest → $quest_active = true
4. Person_1 asks player to deliver message to Person_2
5. Player approaches Person_2, presses Space
6. Person_2 receives message
7. Quest completes → $quest_completed = true, $quest_active = false
```

Quest Variables:

- `$quest_active` (bool) - Quest currently in progress
- `$quest_completed` (bool) - Quest successfully finished

Dialogue Adaptation:

- Person_1 has different dialogue based on quest state
- Person_2 responds differently if player has the message
- NPCs remember previous interactions

Learning Goals:

- Multi-NPC interaction systems
- Quest state management
- Conditional dialogue based on variables
- Using `<<if>>` statements for branching
- Persistent variables across conversations

Yarn Dialogue Files Explained

Start.yarn

Single-node introduction dialogue.

```
title: Start
---
Welcome to this demo! Press space to continue
===
```

GetCoffee.yarn

Two-node dialogue with choices and variables.

Node: Barista_Greeting

- Barista greets player
- Offers three drink choices
- Sets `$drink_ordered` variable based on choice

Node: Barista_ServeDrink

- Confirms order using variable interpolation
- Displays: "Here is your {$drink_ordered}"

Person1.yarn

Quest giver with conditional branching.

Node: Person1_Talk

- Entry point that checks quest state
- Routes to appropriate dialogue based on:
    - First meeting
    - Quest active
    - Quest completed

Node: Person1_FirstMeeting

- Initial greeting
- Offers to help

Node: Person1_GiveQuest

- Explains the quest task
- Player accepts → sets `$quest_active = true`

Person2.yarn

Quest receiver with message delivery logic.

Node: Person2_Talk

- Checks if `$quest_active` is true
- Different responses based on quest state
- Completes quest when message delivered
- Sets `$quest_completed = true` and `$quest_active = false`

QuestVariables.yarn

Variable declarations file.

```
title: QuestVariables
---
<<declare $quest_active = false>>
<<declare $quest_completed = false>>
===
```

Key Concepts Demonstrated

1. Dialogue Interaction Patterns

- **Automatic Triggers:** CoffeeTrigger for zone-based activation
- **Manual Triggers:** NPCInteraction for proximity + key press

2. Player Control Integration

- Movement disabled during dialogue
- Camera lock options
- Input system integration with StarterAssets

3. Variable Types Used

- **Boolean:** Quest state tracking
- **String:** Storing player choices

4. Yarn Spinner Features

- Node jumping with `<<jump>>`
- Conditional logic with `<<if>>`
- Variable declaration with `<<declare>>`
- Variable assignment with `<<set>>`
- Variable interpolation with `{$variable}`
- Dialogue options with `->` syntax
- Stopping dialogue with `<<stop>>`

5. State Management

- Quest states: Not Started → Active → Completed
- Persistent variables across scenes
- Conditional NPC responses based on state

Dependencies

- **Unity Version:** Compatible with Unity 2021.3+ (URP)
- **Yarn Spinner:** 3.1.0
- **StarterAssets:** Third Person Controller package
- **Input System:** New Unity Input System

How to Use This Project

Running the Scenes

1. Open Unity and load the project
2. Navigate to Assets/_game_dkcharles/Scenes/
3. Play scenes in order: HelloWorld → GetaCoffee → SimpleTask

Understanding the Code

1. Start with DialoguePlayerController.cs to see movement control
2. Compare NPCInteraction.cs vs CoffeeTrigger.cs for different interaction patterns
3. Read the Yarn files in Dialogue/ to understand dialogue structure

Modifying Dialogue

1. Open `.yarn` files in Yarn Spinner editor or text editor
2. Edit node content, choices, and variables
3. Test changes by playing the scene

Adding New NPCs

1. Create new NPC gameobject
2. Add collider with "Is Trigger" enabled
3. Attach NPCInteraction script
4. Set `dialogueNode` to your Yarn node name
5. Assign `interactionIndicator` UI element
6. Create corresponding `.yarn` file with matching node

Learning Path Recommendation

Beginner:

1. Play through all three scenes to understand gameplay flow
2. Read Start.yarn to see basic node structure
3. Examine DialoguePlayerController.cs for Unity-Yarn integration

Intermediate:

1. Study GetCoffee.yarn to understand choices and variables
2. Analyze CoffeeTrigger.cs for automatic dialogue triggering
3. Modify drink choices and add new options

Advanced:

1. Dissect Person1.yarn and Person2.yarn for quest logic
2. Study NPCInteraction.cs for proximity detection
3. Extend quest system with multiple objectives
4. Add more NPCs and branching quest paths

Common Patterns

Starting Dialogue from Script

```
dialogueRunner.StartDialogue("NodeName");
```

Checking if Dialogue is Running

```
if (dialogueRunner.IsDialogueRunning) {
    // Dialogue active
}
```

Listening to Dialogue Events

```
dialogueRunner.onDialogueStart.AddListener(OnDialogueStart);
dialogueRunner.onDialogueComplete.AddListener(OnDialogueComplete);
```

Declaring Variables in Yarn

```
<<declare $variableName = defaultValue>>
```

Conditional Branching in Yarn

```
<<if $quest_active is true>>
    You're on a quest!
<<else>>
    No active quests.
<<endif>>
```

Extending This Project

Ideas for Expansion

- Add inventory system to quest
- Create branching quest outcomes
- Implement reputation system with NPCs
- Add more complex dialogue trees
- Create timed dialogue choices
- Implement dialogue history log
- Add voice-over system
- Create dialogue-based puzzles

Additional Features to Practice

- Multiple quest tracking
- NPC schedules and availability
- Dialogue skill checks
- Reputation-based dialogue options
- Item trading through dialogue
- Companion systems

Educational Purpose

This project is designed for teaching:

- **Game Development Students:** Unity + Yarn Spinner integration
- **Narrative Designers:** Interactive dialogue scripting
- **Programming Students:** Event-driven systems and state management
- **Game Design Students:** Quest system architecture

Credits

Designed and created by Prof Darryl Charles

Built using:

- Unity Engine
- Yarn Spinner by Secret Lab
- StarterAssets by Unity Technologies

License

Further Resources

- Yarn Spinner Documentation
- Yarn Spinner for Unity
- Yarn Language Syntax

Project Type: Educational Demo
Complexity: Beginner to Intermediate
Estimated Learning Time: 2-4 hours
Last Updated: January 2026