2D Game Design Workshop - Part 3

Unity Fundamentals: Prefabs, Tags, Boosts, Particle Systems, and More

Lucas P. Cordova, Ph.D.

In this workshop, you'll practice the fundamentals of 2D game development in Unity. By the end of Part 3, you'll have seen the necessary building blocks to create a simple 2D game.

This workshop is divided into three parts:

- Part 1: Unity Fundamentals (this part)
- Part 2: Enhancing Your Game with Physics and Collisions
- Part 3: Adding more interaction, Polishing and Expanding Your Game

Table of contents

0.1 Workshop Overview

In this three-part series, you'll learn the fundamentals of 2D game development in Unity. By the end of Part 3, you'll have seen the necessary building blocks to create a simple 2D game.

Today we'll further explore essential Unity features that transform simple sprites into interactive game elements:

- Creating reusable Prefabs
- Organizing objects with Tags
- Implementing Boosts and power-ups

- Destroying objects through interaction
- Adding visual flair with Particle Systems
- Creating collision-based gameplay mechanics
- Building dynamic user interfaces
- Developing compelling game stories
- Publishing your finished game

1 Prefabs

1.1 Understanding the Game Development Challenge

Before diving into Unity's solutions, let's understand the problems we face:

- Managing multiple similar objects becomes tedious
- Tracking different types of objects gets complex
- Creating satisfying player feedback requires multiple systems
- Building polished games needs professional workflows

Unity's component-based architecture provides elegant solutions to these challenges.

1.2 Creating Prefabs - The Foundation of Scalable Games

What Problem Do Prefabs Solve?

Without prefabs, every game object must be created and configured individually:

- Placing 50 enemy sprites means configuring 50 separate objects
- Changing enemy behavior requires updating each instance manually
- Consistency across objects becomes nearly impossible
- Development time increases exponentially

Prefabs as Templates

Prefabs act as blueprints or templates that solve these issues:

- Create once, instantiate many times
- Update the prefab to change all instances
- Maintain consistency across your game
- Enable efficient spawning during gameplay

1.3 Demo

Prefab Creation and Collision Blocks

Switching to Unity to demonstrate creating prefabs from sprites and setting up collision blocks for walls and barriers

2 Game Object Tags

2.1 Game Object Tags - Smart Organization

Why Tags Matter

Tags provide a lightweight way to categorize and identify game objects:

- Group objects by function rather than appearance
- Enable efficient object detection in scripts
- Simplify collision detection logic
- Organize complex scenes with hundreds of objects

Common Tag Strategies

- **Player**: The main character
- **Enemy**: Hostile entities
- **Boost**: Power-ups and collectibles
- **Wall**: Static collision boundaries
- **Finish**: Goal or endpoint markers

3 Demo

Creating and Assigning Tags

Switching to Unity to demonstrate creating custom tags and assigning them to game objects

3.1 Implementing Boosts - Player Rewards

What Makes Boosts Engaging?

Effective boost systems combine multiple elements: - **Visual Design**: Attractive, noticeable sprites - **Behavior**: Movement patterns that create challenge - **Feedback**: Clear indication when collected - **Game Impact**: Meaningful effect on player experience

Boost Categories

- **Speed Boosts**: Temporarily increase player velocity
- **Score Multipliers**: Enhance point values
- **Power-ups**: Grant temporary abilities
- **Health Items**: Restore player vitality
- **Collection Items**: Progress toward objectives

3.2 Demo

Boost Implementation

Switching to Unity to demonstrate creating boost objects with movement and collection behavior

4 Destroying Objects

4.1 Destroying Objects - Creating Interaction

When and Why Objects Disappear

Strategic object destruction creates engaging gameplay:

- **Collection**: Boosts vanish when picked up
- **Completion**: Targets disappear when objectives are met
- **Cleanup**: Remove off-screen objects to optimize performance
- **Progression**: Clear obstacles to advance levels

Destruction Patterns

- **Immediate**: Object vanishes instantly upon trigger
- **Delayed**: Brief animation before removal
- **Conditional**: Only destroy under specific circumstances
- **Spawning**: Destruction triggers new object creation

4.2 Demo

Object Destruction on Collection

Switching to Unity to demonstrate boost pickup with immediate object destruction

5 Particle Systems

5.1 Particle Systems - Visual Polish

The Power of Particle Effects

Particle systems transform static interactions into dynamic experiences:

- **Feedback**: Confirm player actions visually
- **Atmosphere**: Create environmental mood
- **Impact**: Emphasize important game moments
- **Polish**: Elevate overall production value

Particle System Applications

- **Collection Effects**: Sparkles when picking up boosts
- **Impact Effects**: Explosions from collisions
- **Environmental**: Falling leaves, rain, snow
- **Magic Effects**: Spell casting, power activation
- **Trail Effects**: Following moving objects

5.2 Demo

Particle System Activation

Switching to Unity to demonstrate particle effects triggering when boosts are collected

5.3 Collision-Based Bumps - Dynamic Gameplay

Creating Responsive Game Feel

Collision detection enables rich interaction between game objects:

- **Speed Modification**: Walls slow down player movement
- **Direction Changes**: Bumpers redirect object paths

- **State Changes**: Collisions trigger new behaviors

- **Damage Systems**: Contact reduces player health

Bump Implementation Strategies

- **Physics-Based**: Use Rigidbody components for realistic responses
- **Script-Controlled**: Manually adjust object properties
- **Temporary Effects**: Changes that revert after time
- **Permanent Changes**: Lasting modifications to game state

5.4 Demo

Collision Detection and Speed Modification

Switching to Unity to demonstrate collision detection changing player speed when hitting walls

6 Dynamic UI Text

6.1 Dynamic UI Text - Player Communication

Information Design Principles

Effective UI text keeps players informed and engaged:

- **Clarity**: Information is easy to understand
- **Timing**: Updates happen at appropriate moments

- **Placement**: Text appears where players expect it
- **Consistency**: Styling matches game aesthetic

Dynamic Text Applications

- **Score Display**: Real-time point tracking
- **Status Updates**: Health, energy, lives remaining
- **Progress Indicators**: Completion percentages
- **Achievement Notifications**: Success confirmations
- **Timer Information**: Remaining time or elapsed duration

6.2 Demo

Updating Boost Counter Text

Switching to Unity to demonstrate dynamic text updates when collecting boosts

7 Story Development

7.1 Story Development Framework

Creating Compelling Game Narratives

Great games combine mechanics with meaning through structured storytelling.

7.2 Step 1: Create a Story

Essential Story Elements

Every engaging game needs:

- **Protagonist**: Who is the main character?
- **Goal**: What does the character want to achieve?

- **Obstacles**: What stands in their way?
- **Stakes**: What happens if they fail?
- **Resolution**: How does the journey conclude?

Example Story Framework

"A brave explorer must collect ancient crystals to power their escape ship before the planet explodes, avoiding dangerous creatures and navigating treacherous terrain."

7.3 Step 2: Build a Course/Level

Level Design Principles

Effective levels support your story through:

- **Pacing**: Moments of challenge balanced with relief
- **Progression**: Difficulty increases appropriately

- **Exploration**: Multiple paths encourage discovery
- **Visual Storytelling**: Environment communicates narrative
- **Player Agency**: Meaningful choices affect outcomes

Course Elements Integration

- Place boost crystals at strategic locations
- Position wall barriers to create interesting paths
- Design spaces that encourage player experimentation
- Balance accessibility with appropriate challenge

7.4 Step 3: Identify Main Character

Character Development Considerations

Your protagonist should have:

- **Visual Identity**: Distinctive appearance players remember
- **Mechanical Identity**: Unique movement or abilities
- **Narrative Purpose**: Clear role in the story
- **Player Connection**: Relatable motivations or personality

Character Implementation

- Choose sprites that match your story theme
- Assign appropriate tags for game logic
- Configure movement speed and collision properties
- Plan character progression or ability unlocks

7.5 Step 4: Define Boosts and Bumps Behavior

Boost System Design

Align boost mechanics with narrative purpose:

- **Story Integration**: Why do these items exist in your world?
- **Player Motivation**: What drives collection behavior?
- **Risk/Reward**: What challenges must players overcome?
- **Progression**: How do boosts advance the story?

Bump System Design

Create meaningful obstacles:

- **Narrative Justification**: Why do these barriers exist?
- **Gameplay Impact**: How do they change player behavior?
- **Learning Opportunities**: What skills do players develop?
- **Thematic Consistency**: Do they match your story world?

7.6 Step 5: Integrate Particle Systems

Particle Effects as Storytelling Tools

Particle systems should reinforce narrative elements:

- **Magical Worlds**: Sparkles and glowing effects
- **Sci-Fi Settings**: Energy beams and electrical discharges

- **Natural Environments**: Organic elements like leaves or water
- **Industrial Themes**: Sparks, smoke, and mechanical effects

Effect Timing and Placement

- **Collection Moments**: Celebrate player success
- **Achievement Points**: Mark significant progress
- **Environmental Details**: Enhance world believability
- **Emotional Beats**: Support story climax moments

7.7 Step 6: Implement Dynamic UI Text

UI as Narrative Support

Text elements should enhance rather than distract from your story:

- **Diegetic Integration**: Text that exists within the game world
- **Contextual Information**: Details that support player understanding
- **Progress Communication**: Clear indication of advancement
- **Emotional Reinforcement**: Language that matches story tone

Information Architecture

- **Primary Information**: Essential gameplay data
- **Secondary Information**: Supporting context and flavor
- **Feedback Systems**: Confirmation of player actions
- **Guidance Elements**: Subtle direction without heavy-handedness

8 Releasing Your Game

8.1 Building and Releasing Your Game

Preparation for Release

Before publishing, ensure your game meets quality standards:

Technical Checklist - All prefabs function correctly across different scenarios - Collision detection works reliably - Particle effects perform well on target platforms
- UI text displays properly at various resolutions - Game runs smoothly without crashes or bugs

Content Review - Story elements are clear and engaging - Level design provides appropriate challenge - Visual and audio elements support the narrative - Player feedback systems function effectively

8.2 GitHub Release Process

Version Control Best Practices

GitHub provides essential tools for game development:

- **Repository Setup**: Initialize with Unity .gitignore
- **Commit Strategy**: Regular saves with descriptive messages
- **Branching**: Separate features from main development
- **Documentation**: README with game description and instructions
- **Release Tags**: Mark stable versions for distribution

Building for Distribution

Unity's build process creates platform-specific executables:

- **Target Platform**: Choose appropriate build settings
- **Optimization**: Configure for best performance
- **Testing**: Verify builds work on target devices
- **Packaging**: Create distribution-ready files
- **Upload**: Use GitHub releases for public distribution

8.3 Publishing Workflow

Step-by-Step Release Process

1. **Final Testing**: Complete gameplay verification
2. **Build Creation**: Generate platform-specific executables

3. **Quality Assurance**: Test builds on clean systems
4. **Documentation**: Update README and release notes
5. **GitHub Release**: Create tagged release with assets
6. **Community Engagement**: Share with players and gather feedback

Post-Release Considerations

- **Player Feedback**: Monitor and respond to user reports
- **Bug Fixes**: Address issues through patches
- **Content Updates**: Add new levels or features
- **Community Building**: Engage with your player base
- **Iteration**: Plan improvements for future versions

9 Workshop 3 Conclusion

What We've Covered

You've seen how to integrate multiple Unity systems:

- Create efficient, reusable prefabs
- Organize complex scenes with tags
- Implement satisfying boost collection mechanics
- Add professional polish with particle effects
- Build responsive collision systems
- Create dynamic user interfaces
- Develop compelling game narratives
- Publish finished games through GitHub