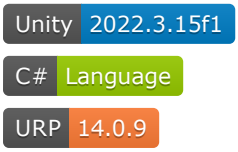
# Slime Rush - 2D Unity Platformer




## Description

Slime Rush is a 2D platformer game built in Unity, featuring dynamic movement, dash mechanics, and engaging platforming challenges. The player navigates through levels filled with cannons, portals, collectibles, and various hazards.

## Features

### Core Gameplay

- **Advanced Movement System**: Ground and air movement with physics-based controls
- **Dash Mechanic**: Multi-directional dashing with limited charges that reset on landing
- **Jump System**: Charge-based jumping with directional control
- **Cannon Launching**: Travel through cannons that propel the player at different angles
- **Portal System**: Scene transitions with smooth animations

### Game Systems

- **Coin Collection**: Collectible coins with score tracking and persistence
- **Health System**: Player health with damage, healing, and heart upgrades
- **Shop System**: Purchase and equip cosmetic items (hats)
- **Enemy Interactions**: Various enemy types with different behaviors (spikes, sticky traps, etc.)
- **Breakable Platforms**: Time-based and trigger-based destructible platforms
- **Visual Effects**: Ghost trails, particle effects, and camera shake

## Project Structure

```
Assets/
├── Scripts/
│   ├── Core/                    # Core systems (EventSystem,
GameEventDefinitions)
│   ├── Player/                  # Player scripts (PlayerScript,
PlayerAnimationHandler, PlayerAudioSystem)
│   │   └── SO/                  # Player ScriptableObject configurations
│   ├── Enemy/                   # Enemy types (Spikes, StickySpikes, etc.)
│   ├── Cannon/                  # Cannon system for launching player
│   ├── Coins/                   # Coin collection system
│   ├── UI/                      # UI components (menus, HUD, game over)
│   ├── Utils/                   # Utility scripts and interfaces
│   ├── InputManager/            # Input handling (touch, keyboard, controller)
```

```
│   ├── Camera/                 # Camera follow and effects
│   ├── Portal/                 # Portal and scene transition system
│   ├── Wall/                   # Wall types (bounce walls, breakable floors)
│   ├── items/                  # Item system (heal items, item boxes)
│   └── DangerZone/             # Level management and platform spawning
├── ShopSystem/                 # Shop functionality for purchasing items
├── Prefabs/                    # Game object prefabs
├── Scenes/                     # Game scenes
└── Animation/                  # Animation controllers and clips
```

# Architecture

## Event System

The project uses a centralized `EventSystem` for decoupled communication between systems:

- **Typed Events**: Strongly-typed event definitions in `GameEventDefinitions.cs`
- **Subscribe/Unsubscribe**: Components subscribe to events in `OnEnable`/`OnDisable`
- **Event Publishing**: Systems raise events without needing direct references

Example:

```
// Subscribe to player damage event
EventSystem.Subscribe<PlayerDamaged>(OnPlayerTakeDamage);

// Raise an event
EventSystem.Raise(new PlayerDamaged { Damage = 1, RemainingHealth = 2 });
```

## ScriptableObject Configuration

Player configuration is managed through `PlayerSO`:

- Movement settings (ground speed, air speed, max velocities)
- Combat stats (damage, health)
- Physics parameters (gravity, drag, mass)
- Dash settings (force, time, max counter)
- Jump settings (force, thresholds)

## Singleton Managers

- **GameManager**: Scene management, player lifecycle, runtime data persistence
- **InputManager**: Centralized input handling for all control schemes
- **ShopSystem**: Shop state and item management

# Unity Version & Dependencies

- **Unity Version**: 2022.3.15f1 (LTS)
- **Render Pipeline**: Universal Render Pipeline (URP) 14.0.9

- **Input System**: Unity Input System 1.7.0
- **TextMeshPro**: 3.0.6
- **Timeline**: 1.7.6

# Setup Instructions

## Prerequisites

1. Install Unity 2022.3.15f1 or later (LTS recommended)
2. Install Unity modules:
    - Universal Windows Platform Build Support (if building for Windows)
    - Android Build Support (if building for mobile)

## Installation

1. Clone the repository:

    ```
    git clone https://github.com/Reistoge/Slime-Rush-a-2D-Unity-Learning-Project.git
    ```

2. Open the project in Unity Hub:

    - Click "Open" in Unity Hub
    - Navigate to the cloned folder and select it
    - Wait for Unity to import all assets

3. Open the main scene:

    - Navigate to `Assets/Scenes/Menu.unity`
    - Press Play to start the game

## Build Instructions

### Windows

1. File → Build Settings
2. Select "Windows, Mac, Linux"
3. Click "Build" and choose output folder

### Android

1. File → Build Settings
2. Select "Android"
3. Click "Player Settings"
4. Configure package name and version
5. Click "Build" or "Build and Run"

# Controls

### Keyboard (PC)

- **Arrow Keys / WASD**: Move left/right
- **Space (Hold & Release)**: Charge and jump
- **Space (in air)**: Dash

### Touch (Mobile)

- **Swipe Up**: Jump
- **Swipe (in air)**: Dash in swipe direction
- **Touch Left/Right**: Move left/right

## Code Style Guidelines

This project follows the coding standards defined in `.editorconfig`:

- **Naming**: PascalCase for public members, camelCase with underscore prefix for private fields
- **Indentation**: 4 spaces for C# files
- **Documentation**: XML documentation comments for all public APIs
- **Braces**: Always use braces for control statements

## Recent Improvements

### Refactoring Updates

- ☑ Reorganized folder structure for better maintainability
- ☑ Added centralized EventSystem for decoupled communication
- ☑ Comprehensive XML documentation for public APIs
- ☑ Standardized code formatting with `.editorconfig`
- ☑ Enhanced interfaces with detailed documentation

## Contributing

When contributing to this project:

1. Follow the code style guidelines in `.editorconfig`
2. Add XML documentation for all public classes and methods
3. Use the EventSystem for inter-system communication
4. Test changes thoroughly before submitting
5. Keep pull requests focused on a single feature or fix

## Future Enhancements

- Refactor large monolithic classes (PlayerScript, GameManager) into smaller components
- Expand ScriptableObject usage for enemy and level configurations
- Implement pooling for frequently spawned objects
- Add more levels and platforming challenges
- Balance game difficulty and player abilities
- Develop narrative and lore

- Enhance game feel with improved juice and polish