Tower Defense Builder Simple

Modular Building Placement System for Unity

🚀 Quick Start Guide

1. Installation

1. Import the package into your Unity project
2. Ensure you have Unity's Input System package installed
3. Add the required UI elements to your scene

2. Basic Setup

Scene Setup:

```csharp
// Create a GameObject with BuildingPlacementView component
var viewObject = new GameObject("BuildingPlacementView");
var view = viewObject.AddComponent<BuildingPlacementView>();

// Create building database
var database = ScriptableObject.CreateInstance<BuildingDatabase>();

// Initialize the system
var model = new BuildingPlacementModel();
var controller = new BuildingPlacementController(model, view, database);
controller.Initialize();
```

Prefab Configuration:

1. Create UI Canvas with building selection panel
2. Assign prefabs for buildings and UI elements
3. Configure materials for placement feedback

3. Creating Buildings

Step 1: Define Building Types

```csharp
public enum BuildingType
{
    BasicTower,
    AdvancedTower,
    SlowTower,
    SplashTower
}
```

Step 2: Create Building Database

1. Right-click in Project → Create → LG → Building Database
2. Assign building prefabs and icons
3. Configure building properties

Step 3: Building Configuration

Buildings are automatically configured through the `BuildingData` class with:

- Damage, Range, Attack Speed
- Cost and Upgrade Cost
- Level progression system

 Architecture Overview

MVC Pattern Implementation

Model (`IBuildingPlacementModel`)

- Manages building data and game state
- Handles placement validation
- Fires events for state changes

View (`IBuildingPlacementView`)

- Handles UI display and user input
- Manages visual feedback
- Creates and updates building visuals

Controller (`IBuildingPlacementController`)

- Coordinates between Model and View
- Handles business logic

- Manages user interactions

Key Components

BuildingPlacementModel

```csharp
public class BuildingPlacementModel : IBuildingPlacementModel
{
    // Events for state changes
    public event Action<BuildingData> OnBuildingPlaced;
    public event Action<BuildingData> OnBuildingUpgraded;
    public event Action<BuildingData> OnBuildingDestroyed;

    // Core functionality
    bool CanPlaceBuilding(Vector2 position, BuildingType type);
    bool PlaceBuilding(Vector2 position, BuildingType type);
    bool UpgradeBuilding(Vector2 position);
    bool DestroyBuilding(Vector2 position);
}
```

BuildingPlacementView

```csharp
public class BuildingPlacementView : MonoBehaviour, IBuildingPlacementView
{
    // UI Events
    public event Action<Vector2> OnTileClicked;
    public event Action<BuildingType> OnBuildingTypeSelected;
    public event Action OnUpgradeButtonClicked;
    public event Action OnDeleteButtonClicked;

    // Visual Management
    void ShowBuildingSelection(Vector2 position, List<BuildingType> availableTypes);
    void CreateBuildingVisual(BuildingData building);
    void UpdateBuildingVisual(BuildingData building);
}
```

⚙️ Configuration Options

Grid Settings

- **Grid Size**: Controls spacing between building slots
- **Ground Layer**: LayerMask for valid placement areas

- **Visual Feedback Materials**: Materials for valid/invalid placement

Input Configuration

- **Unity Input System**: Full support for new input system
- **Touch Support**: Mobile-friendly touch controls
- **Mouse Support**: Traditional mouse input

UI Configuration

- **Building Selection Panel**: Customizable UI for building selection
- **Dynamic Button Generation**: Automatic button creation from building database
- **Upgrade/Delete Buttons**: Contextual action buttons

🔧 Customization Guide

Adding New Building Types

1. **Extend BuildingType enum:**

```csharp
public enum BuildingType
{
    BasicTower,
    AdvancedTower,
    SlowTower,
    SplashTower,
    MagicTower, // New type
    IceTower    // New type
}
```

2. **Update BuildingData constructor:**

```csharp
case BuildingType.MagicTower:
    damage = 30f;
    range = 4.5f;
    attackSpeed = 0.7f;
    cost = 150;
    upgradeCost = 225;
    break;
```

3. **Add to BuildingDatabase and assign prefab/icon**

Custom Placement Validation

Override `CanPlaceBuilding` in your model implementation:

```csharp
public override bool CanPlaceBuilding(Vector2 position, BuildingType type)
{
    // Base validation
    if (_buildings.ContainsKey(position))
        return false;

    // Custom validation (e.g., resource checks)
    if (!HasEnoughResources(type))
        return false;

    // Terrain validation
    if (!IsValidTerrain(position, type))
        return false;

    return true;
}
```

Custom Visual Effects

Extend `UpdateBuildingVisual` for custom upgrade effects:

csharp

```csharp
public override void UpdateBuildingVisual(BuildingData building)
{
    if (building.visual != null)
    {
        // Scale effect
        var targetScale = Vector3.one * (1f + building.level * 0.1f);
        building.visual.transform.localScale = targetScale;

        // Color effect
        var renderer = building.visual.GetComponent<Renderer>();
        if (renderer != null)
        {
            Color levelColor = Color.white * (1f + building.level * 0.2f);
            renderer.material.color = levelColor;
        }

        // Particle effects
        var particles = building.visual.GetComponent<ParticleSystem>();
        if (particles != null)
        {
            var emission = particles.emission;
            emission.rateOverTime = building.level * 10f;
        }
    }
}
```

📱 Mobile Optimization

Touch Input

- Built-in touch support through Unity Input System
- Gesture recognition for building selection
- Optimized for various screen sizes

Performance

- Efficient object pooling for UI elements
- Minimal garbage collection
- Optimized for mobile rendering

UI Scaling

- Responsive UI design

- Automatic scaling for different resolutions
- Touch-friendly button sizes

🐛 Common Issues & Solutions

Issue: Buildings not appearing

Solution: Check that building prefabs are assigned in BuildingDatabase and prefabs have proper renderers.

Issue: Input not working

Solution: Ensure Unity Input System package is installed and InputActionReferences are properly assigned.

Issue: UI buttons not responding

Solution: Verify EventSystem is present in scene and UI elements are properly configured.

Issue: Grid positioning incorrect

Solution: Check grid size settings and ensure ground colliders are on correct layer.

📚 API Reference

Core Classes

BuildingData

```csharp
public class BuildingData
{
    public BuildingType type;
    public Vector2 position;
    public int level;
    public float damage;
    public float range;
    public float attackSpeed;
    public int cost;
    public int upgradeCost;
    public GameObject visual;

    public void Upgrade(); // Increases level and stats
}
```

BuildingDatabase

```csharp
[CreateAssetMenu(fileName = "BuildingDatabase", menuName = "LG/Building Database")]
public class BuildingDatabase : ScriptableObject
{
    public BuildingConfig GetConfig(BuildingType type);
    public BuildingConfig[] GetAllConfigs();
}
```

Events

Model Events

- `OnBuildingPlaced(BuildingData building)` - Fired when a building is placed
- `OnBuildingUpgraded(BuildingData building)` - Fired when a building is upgraded
- `OnBuildingDestroyed(BuildingData building)` - Fired when a building is destroyed

View Events

- `OnTileClicked(Vector2 position)` - Fired when a tile is clicked
- `OnBuildingTypeSelected(BuildingType type)` - Fired when a building type is selected
- `OnUpgradeButtonClicked()` - Fired when upgrade button is clicked
- `OnDeleteButtonClicked()` - Fired when delete button is clicked

🎯 Best Practices

Performance

1. Use object pooling for frequently created/destroyed objects
2. Implement LOD system for building visuals
3. Cache frequently accessed components
4. Use events instead of polling

Code Organization

1. Keep business logic in the Controller
2. Use interfaces for loose coupling
3. Implement proper disposal pattern
4. Follow SOLID principles

User Experience

1. Provide clear visual feedback
2. Implement smooth animations
3. Add sound effects for actions
4. Include tutorials and tooltips

🛠️ Integration Examples

Resource Management Integration

csharp

```csharp
private void HandleBuildingTypeSelected(BuildingType type)
{
    if (!_currentSelectedTile.HasValue || !_isSelectionMode)
        return;

    var position = _currentSelectedTile.Value;
    var buildingData = new BuildingData(type, position);

    // Check resources before placement
    if (!ResourceManager.Instance.CanAfford(buildingData.cost))
    {
        ShowInsufficientResourcesMessage();
        return;
    }

    if (_model.CanPlaceBuilding(position, type))
    {
        ResourceManager.Instance.SpendResources(buildingData.cost);
        _model.PlaceBuilding(position, type);
        _view.HideBuildingSelection();
        _currentSelectedTile = null;
        _isSelectionMode = false;
    }
}
```

Save/Load System Integration

csharp

```csharp
[System.Serializable]
public class BuildingPlacementSaveData
{
    public List<BuildingData> buildings;
}

public BuildingPlacementSaveData GetSaveData()
{
    return new BuildingPlacementSaveData
    {
        buildings = _model.GetAllBuildings()
    };
}

public void LoadSaveData(BuildingPlacementSaveData saveData)
{
    foreach (var building in saveData.buildings)
    {
        _model.PlaceBuilding(building.position, building.type);
        // Restore building level and stats
        for (int i = 1; i < building.level; i++)
        {
            _model.UpgradeBuilding(building.position);
        }
    }
}
```

📞 Support

For technical support, bug reports, or feature requests, please contact us at:

- Email: lordgames.contact@gmail.com
- Unity Asset Store reviews

📄 License

This asset is licensed under Unity Asset Store terms.
Source code is included for modification and extension.
Commercial use is permitted with valid Asset Store license.

🔄 Version History

v1.0.0 (Current)

- Initial release
- Complete MVC architecture
- Grid-based placement system
- Mobile and desktop input support
- Building upgrade/destruction system
- Comprehensive documentation

Thank you for choosing Tower Defense Builder Simple!