Tower Defense Builder Simple

Модульная система размещения зданий для Unity

🚀 Быстрый старт

1. Установка

1. Импортируйте пакет в ваш Unity проект
2. Убедитесь, что у вас установлен пакет Unity Input System
3. Добавьте необходимые UI элементы в сцену

2. Базовая настройка

Настройка сцены:

```csharp
// Создайте GameObject с компонентом BuildingPlacementView
var viewObject = new GameObject("BuildingPlacementView");
var view = viewObject.AddComponent<BuildingPlacementView>();

// Создайте базу данных зданий
var database = ScriptableObject.CreateInstance<BuildingDatabase>();

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

Конфигурация префабов:

1. Создайте UI Canvas с панелью выбора зданий
2. Назначьте префабы для зданий и UI элементов
3. Настройте материалы для визуальной обратной связи

3. Создание зданий

Шаг 1: Определение типов зданий

```csharp
public enum BuildingType
{
    BasicTower,      // Базовая башня
    AdvancedTower,   // Продвинутая башня
    SlowTower,       // Замедляющая башня
    SplashTower      // Взрывная башня
}
```

Шаг 2: Создание базы данных зданий

1. ПКМ в окне Project → Create → LG → Building Database
2. Назначьте префабы зданий и иконки
3. Настройте свойства зданий

Шаг 3: Конфигурация зданий

Здания автоматически настраиваются через класс `BuildingData` со следующими параметрами:

- Урон, дальность, скорость атаки
- Стоимость и стоимость улучшения
- Система прогрессии уровней

 Обзор архитектуры

Реализация паттерна MVC

Модель (`IBuildingPlacementModel`)

- Управляет данными зданий и состоянием игры
- Обрабатывает валидацию размещения
- Генерирует события при изменении состояния

Представление (`IBuildingPlacementView`)

- Обрабатывает отображение UI и пользовательский ввод
- Управляет визуальной обратной связью
- Создает и обновляет визуальные представления зданий

Контроллер (`IBuildingPlacementController`)

- Координирует работу между Моделью и Представлением
- Обрабатывает бизнес-логику

- Управляет пользовательскими взаимодействиями

Ключевые компоненты

BuildingPlacementModel

```csharp
public class BuildingPlacementModel : IBuildingPlacementModel
{
    // События для изменений состояния
    public event Action<BuildingData> OnBuildingPlaced;    // Здание размещено
    public event Action<BuildingData> OnBuildingUpgraded;  // Здание улучшено
    public event Action<BuildingData> OnBuildingDestroyed; // Здание уничтожено

    // Основной функционал
    bool CanPlaceBuilding(Vector2 position, BuildingType type);  // Можно ли разместить
    bool PlaceBuilding(Vector2 position, BuildingType type);     // Разместить здание
    bool UpgradeBuilding(Vector2 position);                      // Улучшить здание
    bool DestroyBuilding(Vector2 position);                      // Уничтожить здание
}
```

BuildingPlacementView

```csharp
public class BuildingPlacementView : MonoBehaviour, IBuildingPlacementView
{
    // События UI
    public event Action<Vector2> OnTileClicked;           // Клик по тайлу
    public event Action<BuildingType> OnBuildingTypeSelected; // Выбор типа здания
    public event Action OnUpgradeButtonClicked;          // Клик по кнопке улучшения
    public event Action OnDeleteButtonClicked;           // Клик по кнопке удаления

    // Управление визуалом
    void ShowBuildingSelection(Vector2 position, List<BuildingType> availableTypes); // Показат
    void CreateBuildingVisual(BuildingData building);     // Создать визуал здания
    void UpdateBuildingVisual(BuildingData building);     // Обновить визуал здания
}
```

⚙️ Опции конфигурации

Настройки сетки

- **Размер сетки**: Контролирует расстояние между слотами для зданий
- **Слой земли**: LayerMask для определения допустимых областей размещения

- **Материалы визуальной обратной связи**: Материалы для индикации допустимого/недопустимого размещения

Конфигурация ввода

- **Unity Input System**: Полная поддержка новой системы ввода
- **Поддержка тач-управления**: Мобильные тач-контролы
- **Поддержка мыши**: Традиционное управление мышью

Конфигурация UI

- **Панель выбора зданий**: Настраиваемый UI для выбора зданий
- **Динамическое создание кнопок**: Автоматическое создание кнопок из базы данных зданий
- **Кнопки улучшения/удаления**: Контекстные кнопки действий

🔧 Руководство по кастомизации

Добавление новых типов зданий

1. **Расширьте enum BuildingType:**

```csharp
public enum BuildingType
{
    BasicTower,
    AdvancedTower,
    SlowTower,
    SplashTower,
    MagicTower,  // Новый тип
    IceTower     // Новый тип
}
```

2. **Обновите конструктор BuildingData:**

```csharp
case BuildingType.MagicTower:
    damage = 30f;         // Урон
    range = 4.5f;         // Дальность
    attackSpeed = 0.7f;   // Скорость атаки
    cost = 150;           // Стоимость
    upgradeCost = 225;    // Стоимость улучшения
    break;
```

3. **Добавьте в BuildingDatabase и назначьте префаб/иконку**

Кастомная валидация размещения

Переопределите `CanPlaceBuilding` в вашей реализации модели:

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

    // Кастомная валидация (например, проверка ресурсов)
    if (!HasEnoughResources(type))
        return false;

    // Валидация местности
    if (!IsValidTerrain(position, type))
        return false;

    return true;
}
```

Кастомные визуальные эффекты

Расширьте `UpdateBuildingVisual` для кастомных эффектов улучшения:

csharp

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

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

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

📱 Мобильная оптимизация

Тач-ввод

- Встроенная поддержка тач-управления через Unity Input System
- Распознавание жестов для выбора зданий
- Оптимизация для различных размеров экранов

Производительность

- Эффективный пулинг объектов для UI элементов
- Минимальная сборка мусора
- Оптимизировано для мобильного рендеринга

Масштабирование UI

- Адаптивный дизайн UI

- Автоматическое масштабирование для разных разрешений
- Удобные для тач-управления размеры кнопок

🐛 Частые проблемы и решения

Проблема: Здания не появляются

Решение: Проверьте, что префабы зданий назначены в BuildingDatabase и у префабов есть правильные рендереры.

Проблема: Ввод не работает

Решение: Убедитесь, что пакет Unity Input System установлен и InputActionReferences правильно назначены.

Проблема: UI кнопки не реагируют

Решение: Проверьте наличие EventSystem в сцене и правильную конфигурацию UI элементов.

Проблема: Неправильное позиционирование на сетке

Решение: Проверьте настройки размера сетки и убедитесь, что коллайдеры земли находятся на правильном слое.

📚 Справочник по API

Основные классы

BuildingData

```csharp
public class BuildingData
{
    public BuildingType type;      // Тип здания
    public Vector2 position;       // Позиция
    public int level;              // Уровень
    public float damage;           // Урон
    public float range;            // Дальность
    public float attackSpeed;      // Скорость атаки
    public int cost;               // Стоимость
    public int upgradeCost;        // Стоимость улучшения
    public GameObject visual;      // Визуальный объект

    public void Upgrade();         // Повышает уровень и характеристики
}
```

BuildingDatabase

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

События

События модели

- `OnBuildingPlaced(BuildingData building)` - Срабатывает при размещении здания
- `OnBuildingUpgraded(BuildingData building)` - Срабатывает при улучшении здания
- `OnBuildingDestroyed(BuildingData building)` - Срабатывает при уничтожении здания

События представления

- `OnTileClicked(Vector2 position)` - Срабатывает при клике по тайлу
- `OnBuildingTypeSelected(BuildingType type)` - Срабатывает при выборе типа здания
- `OnUpgradeButtonClicked()` - Срабатывает при клике по кнопке улучшения
- `OnDeleteButtonClicked()` - Срабатывает при клике по кнопке удаления

🎯 Лучшие практики

Производительность

1. Используйте пулинг объектов для часто создаваемых/уничтожаемых объектов
2. Реализуйте LOD систему для визуалов зданий
3. Кэшируйте часто используемые компоненты
4. Используйте события вместо опроса состояния

Организация кода

1. Держите бизнес-логику в Контроллере
2. Используйте интерфейсы для слабой связанности
3. Реализуйте правильный паттерн освобождения ресурсов
4. Следуйте принципам SOLID

Пользовательский опыт

1. Обеспечьте четкую визуальную обратную связь
2. Реализуйте плавные анимации
3. Добавьте звуковые эффекты для действий
4. Включите обучение и подсказки

🛠️ Примеры интеграции

Интеграция с системой ресурсов

csharp

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

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

    // Проверяем ресурсы перед размещением
    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;
    }
}
```

Интеграция с системой сохранения/загрузки

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);
        // Восстанавливаем уровень и характеристики здания
        for (int i = 1; i < building.level; i++)
        {
            _model.UpgradeBuilding(building.position);
        }
    }
}
```

📞 Поддержка

По вопросам технической поддержки, сообщениям об ошибках или запросам новых функций обращайтесь:

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

📄 Лицензия

Данный ассет лицензирован согласно условиям Unity Asset Store.
Исходный код включен для модификации и расширения.
Коммерческое использование разрешено при наличии действующей лицензии Asset Store.

🔄 История версий

v1.0.0 (Текущая)

- Начальный релиз
- Полная MVC архитектура
- Система размещения на сетке
- Поддержка мобильного и десктопного ввода
- Система улучшения/уничтожения зданий
- Комплексная документация

Спасибо за выбор Tower Defense Builder Simple!