📹 Camera System - Immersize Framework

Le système de caméra Immersize offre une solution complète et modulaire pour la gestion des caméras dans Unity, avec support multi-plateforme et types de caméras variés.

🎯 Caractéristiques Principales

☑ Types de Caméras Supportés

- **TopDown** : Vue de dessus classique (Skylanders style)
- **TopDownHigh/Mid/Low** : Variations d'altitude pour vue de dessus
- **ThirdPerson** : Caméra à la troisième personne avec collision
- **FreeLook** : Caméra libre avec contrôles souris/tactile
- **FirstPerson** : Vue à la première personne
- **BrawlStarsTopDown** : Style Brawl Stars avec contraintes de niveau
- **TunicTopDown** : Style Tunic avec contrôles tactiles avancés
- **OvercookedFixed** : Caméra orthographique fixe style Overcooked

⚡ Fonctionnalités Avancées

- **Async/Await** : Système entièrement asynchrone pour les performances
- **Collision Detection** : Détection automatique des collisions avec environnement
- **Smooth Interpolation** : Lissage séparé horizontal/vertical
- **Builder Pattern** : Configuration fluide et intuitive
- **Presets** : Configurations prédéfinies pour différents genres
- **Multi-Platform** : Support PC, Mobile, AR
- **Performance Optimized** : Frame budget et optimisations mémoire

🚀 Utilisation Rapide

Configuration Basic

```
// Obtenir le service caméra
var cameraService = FrameworkCore.Instance.GetService<CameraService>();

// Créer une caméra avec preset
var camera = cameraService.CreateCameraController("GameCamera",
CameraPreset.PlayBook);

// Changer de type de caméra
cameraService.SwitchCameraType(CameraType.ThirdPerson);
```

Configuration Avancée avec Builder

```
// Configuration personnalisée
CameraControllerBuilder.Create(cameraController)
    .WithPreset(CameraPreset.ActionGame)
    .WithTarget(playerTransform)
    .WithFollowSpeed(8f)
    .WithSmoothTimes(0.02f, 0.04f, 0.1f)
    .WithThirdPersonOffset(new Vector3(2, 3, -8))
    .WithCollisionSettings(LayerMask.GetMask("Environment"))
    .WithTopDownZoomSettings(5f, 20f, 1.5f, 12f)
    .Build();
```

🎮 Types de Caméras Détaillés

Top-Down Cameras

```
// Configuration pour différents styles top-down
builder.WithAllTopDownSettings(
    standard: new Vector3(0, 6, -4),
    high: new Vector3(0, 12, -8),
    mid: new Vector3(0, 8, -6),
    low: new Vector3(0, 4, -2)
);

// Angles de tilt
builder.WithAllTopDownTilts(
    standard: 30f,
    high: 45f,
    mid: 35f,
    low: 25f
);
```

Third Person Camera

```
// Caméra troisième personne avec collision
builder.WithThirdPersonOffset(new Vector3(0, 2, -10))
       .WithThirdPersonTilt(15f)
       .WithCollisionSettings(LayerMask.GetMask("Walls", "Environment"), 0.5f);
```

Free Look Camera

```
// Caméra libre avec contrôles
builder.WithFreeLookOffset(new Vector3(0, 2, -12))
       .WithFreeLookTilt(0f)
       .WithFreeLookZoom(-20f, -2f); // Distance min/max
```

🎨 Presets Disponibles

Action Game

- Follow Speed: 8f
- Smooth Times: (0.02f, 0.04f, 0.1f)
- Third Person optimisé pour l'action

RPG

- Follow Speed: 3f
- Top-Down avec zoom étendu
- Smooth Times plus lents pour immersion

Racing

- Follow Speed: 12f
- Très réactif pour courses
- Third Person dynamique

Strategy

- Follow Speed: 2f
- Top-Down haute altitude
- Zoom étendu pour vue d'ensemble

PlayBook / Dialogue

- Configurations optimisées pour jeux narratifs

🔧 API du CameraService

Méthodes Principales

```
// Gestion des caméras
cameraService.SetActiveCameraController(controller);
cameraService.CreateCameraController(name, preset);
cameraService.SwitchCameraType(CameraType.TopDown);

// Configuration
cameraService.SetCameraTarget(transform);
cameraService.ApplyPresetToActiveController(preset);
cameraService.ConfigureActiveController(); // Retourne un Builder

// Effets
cameraService.ShakeCamera(duration: 0.5f, magnitude: 0.2f);

// Debug
```

```
string info = cameraService.GetCameraInfo();
bool isValid = cameraService.ValidateSetup();
```

Événements

```
cameraService.OnCameraTypeChanged += (type) => Debug.Log($"Switched to
{type}");
cameraService.OnCameraControllerChanged += (controller) => Debug.Log($"New
controller: {controller.name}");
```

📱 Support Multi-Plateforme

Mobile (Tunic Style)

```
// Configuration pour mobile avec contrôles tactiles
builder.WithTunicSettings(
    levelMin: new Vector2(-50, -50),
    levelMax: new Vector2(50, 50),
    angle: 45f,
    height: 15f,
    zoomMin: 6f,
    zoomMax: 20f
).WithZoomSettings(
    tunicZoomMin: 6f,
    tunicZoomMax: 20f,
    tunicZoomSensitivity: 0.2f,
    tunicPanSensitivity: 0.025f
);
```

Brawl Stars Style

```
// Configuration style Brawl Stars
builder.WithBrawlStarsSettings(
    height: 15f,
    distance: 12f,
    angle: 55f,
    levelMin: new Vector2(-20, -20),
    levelMax: new Vector2(20, 20),
    smoothTime: 0.15f
);
```

⚙️ Configuration Avancée

Collision Detection

```
// Paramètres de collision précis
builder.WithCollisionSettings(
    mask: LayerMask.GetMask("Environment", "Walls"),
    minDistance: 0.4f
);

// Smooth times séparés
builder.WithSmoothTimes(
    horizontal: 0.02f,
    vertical: 0.04f,
    collision: 0.2f
);
```

Zoom et Sensibilité

```
// Zoom pour différents types
builder.WithTopDownZoomSettings(3f, 15f, 1f, 8f)
       .WithThirdPersonZoomSettings(5f, 20f, 1f, 10f)
       .WithBrawlStarsZoomSettings(8f, 25f, 1f, 15f);
```

🏛️ Exemples d'Intégration

Dans un Script de Jeu

```
public class GameManager : MonoBehaviour {
    private CameraService cameraService;

    void Start() {
        cameraService = FrameworkCore.Instance.GetService<CameraService>();

        // Configuration initiale
        cameraService.CreateCameraController("MainCamera",
CameraPreset.ActionGame);
        cameraService.SetCameraTarget(player.transform);
    }

    public void OnPlayerDamaged() {
        // Effet de secousse
        cameraService.ShakeCamera(0.3f, 0.15f);
    }

    public void SwitchToDialogueMode() {
        cameraService.ApplyPresetToActiveController(CameraPreset.Dialogue);
        cameraService.SwitchCameraType(CameraType.TopDownMid);
    }
}
```

Interface Utilisateur

```
public class CameraUI : MonoBehaviour {
    [SerializeField] private Dropdown cameraTypeDropdown;
    [SerializeField] private Dropdown presetDropdown;

    private CameraService cameraService;

    void Start() {
        cameraService = FrameworkCore.Instance.GetService<CameraService>();

        // Setup UI
        cameraTypeDropdown.onValueChanged.AddListener(OnCameraTypeChanged);
        presetDropdown.onValueChanged.AddListener(OnPresetChanged);
    }

    void OnCameraTypeChanged(int index) {
        if (System.Enum.IsDefined(typeof(CameraType), index)) {
            cameraService.SwitchCameraType((CameraType)index);
        }
    }

    void OnPresetChanged(int index) {
        if (System.Enum.IsDefined(typeof(CameraPreset), index)) {
            cameraService.ApplyPresetToActiveController((CameraPreset)index);
        }
    }
}
```

🔍 Debug et Diagnostic

Informations de Debug

```
// Informations détaillées
string info = cameraService.GetCameraInfo();
// Output: "[CameraService] Type: ThirdPerson, Camera: MainCamera, Target:
Player, Position: (5,3,-8)"

// Validation de configuration
bool isValid = cameraService.ValidateSetup();
if (!isValid) {
    Debug.LogError("Camera setup has issues!");
}
```

Tests avec le CameraExample

Le `CameraExample` fournit des contrôles clavier pour tester :

- **1-4** : Changer types de caméra
- **Space** : Secousse caméra
- **F1** : Informations debug

Bonnes Pratiques

1. **Utilisez les Presets** pour démarrer rapidement
2. **Configurez les LayerMasks** pour la collision detection
3. **Testez sur mobile** avec les contrôles tactiles
4. **Utilisez les événements** pour synchroniser avec d'autres systèmes
5. **Validez la configuration** en debug
6. **Optimisez les smooth times** selon le genre de jeu

Intégration Framework

Le système s'intègre automatiquement dans le framework Immersize :

- **Priority 3** : Après Input (2), avant autres services
- **Service Locator** : Accessible via `FrameworkCore.Instance.GetService<CameraService>()`
- **Tokenization** : Toutes les méthodes sont documentées et traçables
- **Performance** : Monitoring intégré via PerformanceMonitor

---

Le système de caméra Immersize offre une solution complète et flexible pour tous vos besoins de caméra dans Unity, des jeux mobiles aux expériences PC complexes.