RUNAWAY HEROES

Standard di Sviluppo e Naming Convention

Versione 1.0 - Approvata il 27 Aprile 2025

Questo documento definisce lo standard ufficiale da seguire per lo sviluppo del gioco mobile "Runaway Heroes: Escape The Elements". Tutte le parti coinvolte nello sviluppo sono tenute a rispettare queste linee guida per garantire coerenza e qualità.

1. CONVENZIONI DI NAMING

1.1 File e Directory

Tipo	Convenzione	Esempio
Classi C#	PascalCase	`PlayerController.cs`, `FocusTimeManager.cs`
Prefab	PascalCase	`AlexCharacter.prefab`, `DroneTurret.prefab`
Scene	PascalCase con underscore	`Level1_FirstSteps.unity`, `World2_Forest.unity`
Asset file	snake_case	`alex_run_animation.fbx`, `main_menu_background.png`
File di configurazione	snake_case	`input_mappings.json`, `game_settings.xml`

Separatori nei nomi di file:

- Underscore (`_`) per separare concetti diversi
- Trattino (`-`) solo per versioni o varianti specifiche

1.2 Assets

Modelli 3D (.fbx)

```
[categoria]_[nome]_[variante].fbx
```

Esempi:

- `character_alex_run.fbx`
- `prop_focus_bracelet.fbx`
- `environment_city_building_tall.fbx`

Texture (.png)

```
[target]_[tipo]_[variante].png
```

Esempi:

- `alex_diffuse_default.png`
- `alex_normal_default.png`
- `city_background_night.png`

Audio

```
[categoria]_[nome]_[variante].[estensione]
```

Esempi:

- `sfx_jump_alex.wav`
- `music_city_chase.mp3`
- `voice_jenkins_tutorial_01.ogg`

Animazioni

```
[personaggio]_[azione]_[variante].anim
```

Esempi:

- `alex_jump_high.anim`
- `alex_run_fast.anim`
- `ember_ability_activate.anim`

1.3 Scripting C#

Elemento	Convenzione	Esempio
Namespace	PascalCase	`RunawayHeroes.Characters`
Classi/Struct	PascalCase	`PlayerController`, `FocusTimeManager`
Interfacce	I + PascalCase	`ICollectible`, `IUsableItem`
Metodi	PascalCase	`ActivateAbility()`, `HandleCollision()`
Variabili pubbliche	camelCase	`playerHealth`, `movementSpeed`
Variabili private	_camelCase	`_health`, `_isJumping`
Parametri	camelCase	`void Damage(float damageAmount)`
Proprietà	PascalCase	`Health`, `CurrentSpeed`
Eventi	PascalCase	`OnDamageReceived`, `OnLevelCompleted`
Costanti	UPPER_SNAKE_CASE	`MAX_HEALTH`, `DEFAULT_SPEED`
Enum tipo	PascalCase	`CharacterType`, `GameState`
Enum valori	PascalCase	`CharacterType.Alex`, `GameState.MainMenu`

2. STRUTTURA DEL PROGETTO

2.1 Organizzazione delle Cartelle

> **Nota**: Questa struttura riflette l'organizzazione attuale del progetto come documentato nel file `unity_folder_structure.txt`. Per qualsiasi aggiornamento o modifica alla struttura, fare riferimento sempre a questo file che rappresenta lo stato reale del progetto.

```
Assets/
  _Project/                    # Directory principale del progetto
    Art/                       # Asset artistici
      Animations/              # Animazioni
        Characters/            # Organizzate per personaggio
          Alex/
          Ember/
          Kai/
          Marina/
          Maya/
          Neo/
        Enemies/
          Bosses/
          Common/
          MidBosses/
        Environment/
          Tutorial/
          World1_City/
          World2_Forest/
          World3_Tundra/
          World4_Volcano/
          World5_Abyss/
          World6_Virtual/
        UI/
      Materials/               # Materiali
        Characters/
          Alex/
        Enemies/
        Environment/
          Tutorial/
          World1_City/
          World2_Forest/
          World3_Tundra/
          World4_Volcano/
          World5_Abyss/
          World6_Virtual/
        FX/
      Models/                  # Modelli 3D
        Characters/
          Alex/
          Ember/
          Marina/
        Enemies/
          Tutorial/
          World1_City/
          World2_Forest/
```

```
      World3_Tundra/
      World4_Volcano/
      World5_Abyss/
      World6_Virtual/
    Environment/
      Tutorial/
      World1_City/
      World2_Forest/
      World3_Tundra/
      World4_Volcano/
      World5_Abyss/
      World6_Virtual/
    Props/
  Particles/                  # Sistemi di particelle
    Abilities/
    Enemies/
    Environment/
    FX/
  Shaders/                    # Shader custom
  Textures/                   # Texture
    Characters/
      Alex/
        Materials/
      Ember/
      Marina/
    Enemies/
    Environment/
      Tutorial/
      World1_City/
      World2_Forest/
      World3_Tundra/
      World4_Volcano/
      World5_Abyss/
      World6_Virtual/
    FX/
    UI/
  UI/                         # Asset UI
    Collectibles/
    FocusTime/
    HUD/
    Menus/
Audio/                        # File audio
  Music/                      # Tracce musicali
    Boss/
    Menu/
    World1_City/
    World2_Forest/
```

```
    World3_Tundra/
    World4_Volcano/
    World5_Abyss/
    World6_Virtual/
  SFX/                        # Effetti sonori
    Abilities/
    Characters/
    Enemies/
    Environment/
    UI/
  Voice/                      # Dialoghi e voci
Code/                         # Script C#
  Abilities/                  # Abilità dei personaggi
    AirBubble/
    ControlledGlitch/
    FireproofBody/
    FocusTime/
    HeatAura/
    NatureCall/
    UrbanDash/
  Characters/                 # Logica dei personaggi
    Alex/
    Base/                     # Classi base condivise
    Ember/
    Kai/
    Marina/
    Maya/
    Neo/
  Core/                       # Sistemi fondamentali
    AudioManager/
    GameManager/
    InputSystem/
    LevelManager/
    ObjectPooling/
    SaveSystem/
    UI/
      Menu/
  Enemies/                    # Logica dei nemici
    Architect/
    Base/
    Bosses/
    Drones/
      Projectiles/
    MidBosses/
  Environment/                # Elementi ambientali
    Base/
    Tutorial/
```

```
    World1_City/
      Terminals/
    World2_Forest/
    World3_Tundra/
    World4_Volcano/
    World5_Abyss/
    World6_Virtual/
  Fragments/                  # Frammenti del Nucleo dell'Equilibrio
    AbyssalFragment/
    DigitalFragment/
    FragmentBase/
    GlacialFragment/
    IgneousFragment/
    NatureFragment/
    UrbanFragment/
  Items/                      # Oggetti collezionabili e usabili
    Base/
    Collectibles/
    Consumables/
    Equipment/
    Relics/
  Mechanics/                  # Meccaniche di gioco
    Combat/
    Movement/
    Purification/
    Resonance/
    WorldSpecific/
  UI/                         # Interfaccia utente
    FocusTimeUI/
    HUD/
    Menus/
    Shop/
  Utilities/                  # Utility varie
Data/                         # Dati e configurazioni
  GameConstants/
  Localization/
  ScriptableObjects/
    Abilities/
    Characters/
    Enemies/
    Items/
    Levels/
Prefabs/                      # Prefab organizzati per categoria
  Characters/
  Enemies/
    Tutorial/
    World1_City/
```

```
    World2_Forest/
    World3_Tundra/
    World4_Volcano/
    World5_Abyss/
    World6_Virtual/
  Environment/
    Tutorial/
    World1_City/
    World2_Forest/
    World3_Tundra/
    World4_Volcano/
    World5_Abyss/
    World6_Virtual/
  FX/
  Items/
  UI/
Scenes/                       # Scene di gioco
  Cutscenes/
  PremiumWorlds/
  Tutorial/
    Level1_FirstSteps/
    Level2_PerfectSlide/
    Level3_ReadyReflexes/
    Level4_ItemPower/
    Level5_EscapeTrainer/
  UI/
    Credits/
    MainMenu/
    Shop/
    WorldSelection/
  World1_City/
    Level1_CentralPark/
    Level2_CommercialAvenues/
    Level3_ResidentialDistrict/
    Level4_ConstructionArea/
    Level5_IndustrialZone/
    Level6_AbandonedSite/
    Level7_RundownPeriphery/
    Level8_PollutedDistrict/
    Level9_TechCenter/
  World2_Forest/
  World3_Tundra/
  World4_Volcano/
  World5_Abyss/
  World6_Virtual/
Settings/                     # Impostazioni di progetto
  InputActions/
```

```
    PostProcessing/
    ProjectSettings/
    RenderPipeline/
  ThirdParty/                  # Asset di terze parti
```

2.2 Standard per Prefab

- Ogni prefab deve avere un nome descrittivo che indica la sua funzione
- I prefab complessi devono essere composti da prefab più piccoli (nesting)
- Tutti i prefab devono essere correttamente taggati e organizzati in layer
- Utilizzare varianti per diverse configurazioni dello stesso prefab
- Ogni prefab deve avere collider appropriati se interagibili

3. STANDARD PER ASSETS

3.1 Modelli 3D

- **Unità di misura**: 1 unità = 1 metro
- **Orientamento**: Forward = Z+, Up = Y+
- **Origin**: Al centro della base dell'oggetto
- **Scale**: Tutti i modelli importati devono avere scale uniformi (1,1,1)
- **Poligoni**: Mobile-friendly (low-poly quando possibile)
  - Personaggi principali: max 7.500 triangoli
  - Nemici comuni: max 3.000 triangoli
  - Elementi ambientali: max 1.500 triangoli
- **UV Mapping**: Ottimizzato per texture atlas
- **Rigging**: Max 35 bones per character mesh
- **Weights**: Max 4 bone influences per vertex

3.2 Texture

- **Risoluzione**: Potenze di 2 (256x256, 512x512, 1024x1024, 2048x2048)
- **Formati**:
  - Diffuse/Albedo: PNG a 24-bit
  - Normal Maps: PNG a 24-bit
  - Mask/Metallic/Roughness: PNG a 8-bit
- **Dimensioni massime**:
  - Personaggi principali: 2048x2048
  - Nemici comuni: 1024x1024

    - Props: 512x512
    - UI: 256x256 (quando possibile)
- **Texture Atlas**: Utilizzare per elementi simili o correlati
- **Compressione**: ETC2 (Android) / PVRTC (iOS)

3.3 Audio

- **Musica**: MP3 a 192 kbps, 44.1kHz
- **SFX**: WAV non compressi o OGG a 128 kbps
- **Durata massima SFX**: 3 secondi (quando possibile)
- **Normalizzazione**: -3 dB per tutti i file audio
- **Sample Rate**: 44.1kHz per musica, 22kHz per SFX

4. PRATICHE DI CODICE

> **Nota**: Per informazioni dettagliate sulle classi esistenti, fare riferimento ai seguenti file:
> - `class_index.txt`: Indice completo di tutte le classi nel progetto
> - `class_skeleton.txt`: Struttura scheletrica delle classi con metodi e proprietà
> - `class_dependencies.dot`: Diagramma delle dipendenze tra le classi (in formato DOT/Graphviz)
>
> Questi file rappresentano la fonte di verità per la struttura del codice esistente e devono essere consultati prima di creare nuove classi o modificare quelle esistenti.

4.1 Pattern di Design

- **Singleton**: Per manager globali (GameManager, AudioManager)
- **Component-based**: Per comportamenti modulari e riutilizzabili
- **State Pattern**: Per la gestione degli stati (nemici, UI)
- **Observer Pattern**: Per eventi e comunicazione tra sistemi
- **Factory Pattern**: Per la creazione di oggetti complessi

4.2 Documentazione del Codice

Ogni classe deve iniziare con un header che descrive lo scopo:

csharp

```csharp
/// <summary>
/// Gestisce il comportamento del personaggio Alex, incluse le sue abilità specifiche
/// e le interazioni con l'ambiente urbano.
/// </summary>
public class AlexCharacter : CharacterBase
{
    // Implementazione
}
```

Metodi pubblici devono avere commenti che descrivono:

- Funzionalità
- Parametri
- Valori di ritorno
- Eccezioni (se applicabile)

csharp

```csharp
/// <summary>
/// Applica danno al personaggio e attiva feedback visivo
/// </summary>
/// <param name="damageAmount">Quantità di danno da applicare</param>
/// <param name="knockbackDirection">Direzione del knockback (opzionale)</param>
/// <param name="knockbackForce">Forza del knockback (opzionale)</param>
public virtual void TakeDamage(float damageAmount, Vector3 knockbackDirection = defaul
{
    // Implementazione
}
```

4.3 Organizzazione del Codice

- **Ordinamento all'interno delle classi**:
    1. Campi (fields)
    2. Proprietà (properties)
    3. Eventi (events)
    4. Costruttori
    5. Metodi Unity (Awake, Start, Update, etc.)
    6. Metodi pubblici
    7. Metodi protetti
    8. Metodi privati

- **Regioni**: Utilizzare regioni per organizzare sezioni logiche di codice

```csharp
#region Movement
// Metodi relativi al movimento
#endregion

#region Combat
// Metodi relativi al combattimento
#endregion
```

5. CONTROLLO QUALITÀ

5.1 Performance Targets

- **FPS**: 60 fps su dispositivi target
- **Memoria**: Max 500 MB di RAM utilizzata
- **Dimensione build**: <100 MB per installazione iniziale
- **Tempo di caricamento**: <3 secondi per livello

5.2 Checklist per Review

- Il codice segue le convenzioni di naming?
- Gli assets rispettano i formati e le risoluzioni definiti?
- Le scene sono organizzate gerarchicamente in modo chiaro?
- I prefab sono riutilizzabili e modulari?
- I commenti sono chiari e informativi?
- La funzionalità è stata testata su dispositivi target?

6. WORKFLOW DI SVILUPPO

1. **Pianificazione**: Definire chiaramente requisiti e acceptance criteria
2. **Implementazione**: Seguire gli standard definiti
3. **Test**: Verificare funzionalità e performance
4. **Review**: Controllo di qualità tramite checklist
5. **Integrazione**: Merge nel progetto principale
6. **Documentazione**: Aggiornare documentazione tecnica se necessario

---

CONFORMITÀ E ADOZIONE

Questo documento rappresenta lo standard ufficiale di Runaway Heroes. Tutti i membri del team di sviluppo sono tenuti ad aderire a queste linee guida. Per proposte di modifica a questo standard, contattare il responsabile tecnico del progetto.

Fine del documento