GUIDA DI SVILUPPO PER LIVELLI RANDOMIZZATI IN RUNAWAY HEROES

1. PANORAMICA DEL SISTEMA DI RANDOMIZZAZIONE

1.1 Obiettivi

- Mantenere la struttura topologica dei livelli invariata
- Randomizzare posizionamento di ostacoli, nemici e collezionabili
- Preservare il bilanciamento e la difficoltà progressiva
- Garantire compatibilità con le abilità di tutti i personaggi
- Creare variabilità ad ogni sessione di gioco

1.2 Metodologia

- Sistema a celle modulari con regole di posizionamento
- Pool predefiniti di elementi per ogni livello
- Seed di randomizzazione per garantire riproducibilità
- Validazione automatica per assicurare completabilità

2. CATALOGO COMPLETO DEGLI ELEMENTI

2.1 Tipologie di Ostacoli

Gli ostacoli sono categorizzati secondo quattro criteri principali:

- **Altezza**: Basso, Medio, Alto
- **Interazione**: Fisso, Mobile, Distruttibile, Attraversabile
- **Dimensione**: Piccolo, Medio, Grande
- **Effetto**: Standard, Danno, Stordimento, Rallentamento, Effetto Speciale

2.1.1 Ostacoli Universali (Presenti in tutti i mondi)

ID	Nome	Altezza	Interazione	Dimensione	Effetto	Note
U01	Barriera Bassa	Basso	Fisso	Piccolo	Standard	Superabile con salto
U02	Barriera Alta	Alto	Fisso	Medio	Standard	Superabile con scivolata
U03	Gap Stretto	-	Fisso	Piccolo	Standard	Superabile con salto
U04	Gap Largo	-	Fisso	Medio	Standard	Superabile con doppio salto o abilità
U05	Barriera Mobile	Medio	Mobile	Medio	Standard	Pattern prevedibile
U06	Trappola	Basso	Fisso	Piccolo	Danno	Causa danno al contatto
U07	Piattaforma Instabile	Basso	Distruttibile	Medio	Standard	Crolla dopo un tempo
U08	Piattaforma Mobile	Basso	Mobile	Medio	Standard	Trasporta il giocatore

2.1.2 Ostacoli del Mondo Urbano

ID	Nome	Altezza	Interazione	Dimensione	Effetto	Note
C01	Auto	Alto	Fisso	Grande	Danno	Attraversabile da Alex con Scatto Urbano
C02	Cassonetto	Medio	Distruttibile	Piccolo	Standard	Sfondabile con forza sufficiente
C03	Barriera Stradale	Medio	Distruttibile	Medio	Standard	Sfondabile con Scatto Urbano
C04	Cancello Elettronico	Alto	Attraversabile	Grande	Standard	Attraversabile da Neo con Glitch
C05	Drone Sorveglianza	Alto	Mobile	Piccolo	Danno	Si sposta su pattern
C06	Pozza Tossica	Basso	Fisso	Medio	Danno	Attraversabile da Ember con immunità
C07	Ventola Industriale	Medio	Fisso	Medio	Rallentamento	Crea corrente contraria
C08	Cantiere	Alto	Fisso	Grande	Standard	Richiede deviazione

2.1.3 Ostacoli della Foresta Primordiale

ID	Nome	Altezza	Interazione	Dimensione	Effetto	Note
F01	Tronco	Medio	Distruttibile	Medio	Standard	Sfondabile con forza sufficiente
F02	Radici Sporgenti	Basso	Fisso	Piccolo	Rallentamento	Rallenta il movimento
F03	Pianta Carnivora	Medio	Distruttibile	Medio	Danno	Attacca quando vicino
F04	Liane	Alto	Attraversabile	Medio	Standard	Utilizzabile per oscillazione
F05	Pozza Melmosa	Basso	Fisso	Medio	Rallentamento	Attraversabile da Marina con Bolla
F06	Albero Caduto	Alto	Fisso	Grande	Standard	Richiede aggiramento
F07	Fungo Velenoso	Basso	Fisso	Piccolo	Danno	Emette gas tossico
F08	Palude	Basso	Fisso	Grande	Rallentamento	Attraversabile da Maya con bonus

2.1.4 Ostacoli della Tundra Eterna

ID	Nome	Altezza	Interazione	Dimensione	Effetto	Note
T01	Muro di Ghiaccio	Alto	Distruttibile	Grande	Standard	Scioglibile da Kai con Aura
T02	Spuntoni di Ghiaccio	Medio	Distruttibile	Piccolo	Danno	Scioglibili da Kai
T03	Crepaccio	-	Fisso	Medio	Standard	Richiede salto preciso
T04	Superficie Ghiacciata	Basso	Fisso	Grande	Rallentamento	Riduce controllo, Kai immune
T05	Valanga	Alto	Mobile	Grande	Danno	Si muove verso il giocatore
T06	Vento Gelido	-	Fisso	Grande	Rallentamento	Spinge indietro, Kai immune
T07	Stalattite	Alto	Mobile	Piccolo	Danno	Cade quando il giocatore passa sotto
T08	Igloo	Medio	Distruttibile	Medio	Standard	Riparo temporaneo, sfondabile

2.1.5 Ostacoli dell'Inferno di Lava

ID	Nome	Altezza	Interazione	Dimensione	Effetto	Note
V01	Pozza di Lava	Basso	Fisso	Medio	Danno	Attraversabile da Ember con Corpo Ignifugo
V02	Geyser	Alto	Mobile	Piccolo	Danno	Eruzione periodica
V03	Roccia Vulcanica	Medio	Distruttibile	Medio	Standard	Sfondabile con forza
V04	Parete di Fuoco	Alto	Fisso	Grande	Danno	Attraversabile da Ember
V05	Vapore Bollente	Medio	Mobile	Medio	Danno	Nuvola che si sposta
V06	Fessura nella Terra	-	Fisso	Medio	Standard	Richiede salto
V07	Roccia Incandescente	Medio	Mobile	Piccolo	Danno	Rotola verso il giocatore
V08	Terra Instabile	Basso	Distruttibile	Medio	Standard	Crolla nella lava

2.1.6 Ostacoli degli Abissi Inesplorati

ID	Nome	Altezza	Interazione	Dimensione	Effetto	Note
A01	Corrente Marina	-	Fisso	Grande	Rallentamento	Spinge in una direzione
A02	Corallo Urticante	Medio	Fisso	Piccolo	Danno	Causa danno al contatto
A03	Zona Senza Ossigeno	-	Fisso	Grande	Danno	Marina immune con Bolla
A04	Anemone Gigante	Alto	Distruttibile	Medio	Danno	Tentacoli urticanti
A05	Rete da Pesca	Alto	Distruttibile	Grande	Rallentamento	Intrappola temporaneamente
A06	Vortice Sottomarino	-	Mobile	Grande	Rallentamento	Trascina il giocatore
A07	Miniera Sottomarina	Medio	Distruttibile	Piccolo	Danno	Esplode al contatto
A08	Grotta Stretta	Alto	Fisso	Grande	Standard	Richiede nuoto preciso

2.1.7 Ostacoli della Realtà Virtuale

ID	Nome	Altezza	Interazione	Dimensione	Effetto	Note
D01	Firewall	Alto	Attraversabile	Grande	Danno	Attraversabile da Neo con Glitch
D02	Blocco di Dati	Medio	Distruttibile	Medio	Standard	Sfondabile con forza
D03	Glitch Spaziale	-	Mobile	Medio	Teletrasporto	Sposta il giocatore
D04	Virus	Basso	Mobile	Piccolo	Danno	Insegue il giocatore
D05	Buffer Overflow	-	Fisso	Grande	Stordimento	Confonde i controlli
D06	Barriera Crittografata	Alto	Attraversabile	Grande	Standard	Solo Neo può attraversarla
D07	Frammentazione	Basso	Distruttibile	Piccolo	Rallentamento	Terreno instabile
D08	Loop Temporale	-	Fisso	Medio	Rallentamento	Rallenta il tempo localmente

2.2 Tipologie di Nemici

I nemici sono categorizzati secondo cinque criteri:

- **Comportamento**: Statico, Pattuglia, Inseguitore, Volante, Speciale
- **Attacco**: Corpo a Corpo, A Distanza, Area, Speciale
- **Resistenza**: Bassa, Media, Alta
- **Velocità**: Lenta, Media, Veloce
- **Vulnerabilità**: Abilità specifiche che sono efficaci contro di loro

2.2.1 Nemici Universali

ID	Nome	Comportamento	Attacco	Resistenza	Velocità	Vulnerabilità
EN01	Drone Base	Pattuglia	Distanza	Bassa	Media	Scatto (Alex), Glitch (Neo)
EN02	Sentinella	Statico	Area	Media	-	Tutte le abilità
EN03	Cacciatore	Inseguitore	Corpo a Corpo	Media	Veloce	Natura (Maya), Bolla (Marina)

2.2.2 Nemici del Mondo Urbano

ID	Nome	Comportamento	Attacco	Resistenza	Velocità	Vulnerabilità
CN01	Drone di Sorveglianza	Volante	Distanza	Bassa	Media	Scatto (Alex)
CN02	Drone Pubblicitario	Volante	Area	Bassa	Lenta	Scatto (Alex), Glitch (Neo)
CN03	Robot Operaio	Pattuglia	Corpo a Corpo	Media	Lenta	Tutte le abilità
CN04	Drone di Sicurezza	Volante	Distanza	Media	Media	Scatto (Alex)
CN05	Drone Danneggiato	Volante	Speciale	Bassa	Variabile	Scatto (Alex), Glitch (Neo)
CN06	Drone Contaminato	Inseguitore	Area	Media	Lenta	Tutti tranne Ember

2.2.3 Nemici della Foresta Primordiale

ID	Nome	Comportamento	Attacco	Resistenza	Velocità	Vulnerabilità
FN01	Serpente Corrotto	Pattuglia	Corpo a Corpo	Bassa	Media	Natura (Maya)
FN02	Ragno Gigante	Inseguitore	Corpo a Corpo	Bassa	Veloce	Natura (Maya), Aura (Kai)
FN03	Pesce Predatore	Pattuglia	Corpo a Corpo	Bassa	Veloce	Bolla (Marina)
FN04	Rana Velenosa	Statico	Area	Bassa	Lenta	Natura (Maya)
FN05	Pipistrello di Ghiaccio	Volante	Distanza	Bassa	Veloce	Aura (Kai)
FN06	Golem di Neve	Inseguitore	Corpo a Corpo	Alta	Lenta	Aura (Kai), Corpo (Ember)

2.2.4 Nemici della Tundra Eterna

ID	Nome	Comportamento	Attacco	Resistenza	Velocità	Vulnerabilità
TN01	Lupo Artico	Inseguitore	Corpo a Corpo	Media	Veloce	Natura (Maya)
TN02	Corvo delle Nevi	Volante	Distanza	Bassa	Veloce	Natura (Maya)
TN03	Foca Predatrice	Pattuglia	Corpo a Corpo	Media	Media	Bolla (Marina)
TN04	Piccolo Orso	Inseguitore	Corpo a Corpo	Media	Media	Natura (Maya), Aura (Kai)
TN05	Volpe Artica	Inseguitore	Corpo a Corpo	Bassa	Veloce	Natura (Maya)
TN06	Spettro della Tormenta	Speciale	Area	Media	Variabile	Aura (Kai)
TN07	Guardiano di Neve	Statico	Distanza	Alta	Lenta	Aura (Kai), Corpo (Ember)
TN08	Orso Polare	Inseguitore	Corpo a Corpo	Alta	Media	Aura (Kai)

2.2.5 Nemici dell'Inferno di Lava

ID	Nome	Comportamento	Attacco	Resistenza	Velocità	Vulnerabilità
VN01	Salamandra di Fuoco	Pattuglia	Corpo a Corpo	Media	Media	Corpo (Ember)
VN02	Vespa Lavica	Volante	Corpo a Corpo	Bassa	Veloce	Corpo (Ember)
VN03	Pipistrello Infuocato	Volante	Area	Bassa	Veloce	Corpo (Ember)
VN04	Operaio Spettrale	Pattuglia	Distanza	Media	Lenta	Bolla (Marina), Corpo (Ember)
VN05	Scorpione di Lava	Inseguitore	Corpo a Corpo	Media	Media	Corpo (Ember)
VN06	Spettro del Calore	Speciale	Area	Media	Variabile	Corpo (Ember), Aura (Kai)
VN07	Guardiano di Ossidiana	Statico	Distanza	Alta	Lenta	Corpo (Ember), Scatto (Alex)
VN08	Cultista di Fuoco	Pattuglia	Distanza	Media	Media	Corpo (Ember)

2.2.6 Nemici degli Abissi Inesplorati

ID	Nome	Comportamento	Attacco	Resistenza	Velocità	Vulnerabilità
AN01	Pesce Palla	Pattuglia	Area	Media	Media	Bolla (Marina)
AN02	Murena	Statico	Corpo a Corpo	Media	Veloce	Bolla (Marina)
AN03	Guardiano di Pietra	Statico	Corpo a Corpo	Alta	Lenta	Scatto (Alex), Glitch (Neo)
AN04	Pesce Piranha	Inseguitore	Corpo a Corpo	Bassa	Veloce	Bolla (Marina)
AN05	Calamaro Gigante	Speciale	Distanza	Media	Media	Bolla (Marina)
AN06	Medusa Velenosa	Pattuglia	Area	Bassa	Lenta	Bolla (Marina)
AN07	Anguilla Elettrica	Pattuglia	Area	Media	Media	Bolla (Marina), Glitch (Neo)
AN08	Granchio Gigante	Inseguitore	Corpo a Corpo	Alta	Lenta	Bolla (Marina), Scatto (Alex)

2.2.7 Nemici della Realtà Virtuale

ID	Nome	Comportamento	Attacco	Resistenza	Velocità	Vulnerabilità
DN01	Antivirus Base	Pattuglia	Distanza	Bassa	Media	Glitch (Neo)
DN02	Pacchetto Corrotto	Speciale	Area	Bassa	Variabile	Glitch (Neo)
DN03	Spyware	Inseguitore	Speciale	Bassa	Veloce	Glitch (Neo)
DN04	Bot Maligno	Pattuglia	Corpo a Corpo	Media	Media	Glitch (Neo), Scatto (Alex)
DN05	File Corrotto	Statico	Area	Media	-	Glitch (Neo)
DN06	Trojan	Speciale	Speciale	Media	Variabile	Glitch (Neo)
DN07	Bug Logico	Speciale	Area	Bassa	Variabile	Glitch (Neo)
DN08	Programma Guardiano	Pattuglia	Distanza	Alta	Media	Glitch (Neo)

2.3 Tipologie di Collezionabili

ID	Nome	Effetto	Nota
COL01	Moneta Standard	+1 moneta	Base del sistema monetario
COL02	Moneta d'Argento	+5 monete	Meno comune
COL03	Moneta d'Oro	+10 monete	Rara
COL04	Gemma	+1 gemma	Valuta premium rara
COL05	Frammento	Componente collezionabile	Per potenziamenti permanenti
POW01	Kit Medico	Ripristina salute	Varia per quantità
POW02	Boost di Velocità	Aumenta la velocità	Temporaneo
POW03	Invincibilità	Rende invulnerabile	Breve durata
POW04	Magnetismo	Attrae monete	Raggio medio
POW05	Doppio Valore	Raddoppia le monete	Durata media
POW06	Riduzione Cooldown	Riduce i cooldown delle abilità	Effetto immediato

3. INTERAZIONE DELLE ABILITÀ DEI PERSONAGGI CON ELEMENTI DEL MONDO

3.1 Alex (Città in Caos)

- **Scatto Urbano**:
  - Permette di sfondare ostacoli distruttibili di dimensione piccola e media
  - Attiva l'invulnerabilità per 2 secondi, evitando danni da nemici e ostacoli
  - Efficace contro: Droni, Barriere Stradali, Cassonetti, Blocchi di Dati, Tronchi
  - Non efficace contro: Muri di Ghiaccio (T01), Pozze di Lava (V01), Firewall (D01)

3.2 Maya (Foresta Primordiale)

- **Richiamo della Natura**:
  - Evoca animali che distraggono nemici per 5 secondi
  - Gli animali evocati attraggono nemici con comportamento Inseguitore e Pattuglia
  - Efficace contro: Tutti i nemici animali, particolarmente quelli della Foresta e Tundra
  - Non efficace contro: Droni meccanici, Guardiani di pietra, Entità digitali

3.3 Kai (Tundra Eterna)

- **Aura di Calore**:
  - Crea un campo che scioglie gli ostacoli di ghiaccio nel raggio
  - Protezione dal rallentamento causato dal freddo
  - Efficace contro: Muri di Ghiaccio (T01), Spuntoni (T02), Superfici Ghiacciate (T04)
  - Parzialmente efficace contro: Nemici di ghiaccio (riduzione danni), Spettri della Tormenta

3.4 Ember (Inferno di Lava)

- **Corpo Ignifugo**:
  - Permette di attraversare superfici di lava e fuoco senza subire danni
  - Resistenza agli attacchi basati sul fuoco e calore
  - Efficace contro: Pozze di Lava (V01), Pareti di Fuoco (V04), Vapore Bollente (V05)
  - Efficace contro nemici: Salamandre di Fuoco, Vespe Laviche, Cultisti di Fuoco

3.5 Marina (Abissi Inesplorati)

- **Bolla d'Aria**:
  - Crea una sfera protettiva che fornisce ossigeno extra
  - Respinge nemici acquatici nel raggio
  - Efficace contro: Zone Senza Ossigeno (A03), Vortici (A06), Correnti (A01)
  - Efficace contro nemici: Pesci, Murene, Meduse, Calamari, e altri nemici acquatici

3.6 Neo (Realtà Virtuale)

- **Glitch Controllato**:
  - Permette di attraversare barriere digitali
  - Breve teletrasporto attraverso ostacoli
  - Efficace contro: Firewall (D01), Barriere Crittografate (D06), Cancelli Elettronici (C04)
  - Efficace contro nemici: Antivirus, Pacchetti Corrotti, File Corrotti, Bug Logici

3.7 Risonanza dei Frammenti (Tutti i personaggi)

- Permette il cambio istantaneo tra personaggi sbloccati
- Genera un'onda di energia che danneggia i nemici in un raggio di 5 metri
- Garantisce invulnerabilità per 1.5 secondi dopo il cambio
- Consuma il 10% della barra Focus Time
- Utilizzo strategico: cambiare personaggio per superare ostacoli specifici

4. SISTEMA DI RANDOMIZZAZIONE PER LIVELLI

4.1 Struttura di Celle e Segmenti

Ogni livello è diviso in segmenti, ciascuno composto da celle modulari:

- **Lane**: 3 corsie standard (sinistra, centro, destra)
- **Segment Length**: 10-20 celle per segmento di percorso
- **Cell Types**:

- Base (terreno neutro)
- Obstacle (posizionamento ostacoli)
- Enemy (spawn nemici)
- Collectible (monete/power-up)
- Junction (biforcazione)
- Special (eventi particolari)

4.2 Regole di Posizionamento

- **Ostacoli**:
  - Distanza minima tra ostacoli consecutivi: 3 celle
  - Garantire sempre almeno una corsia libera per gli ostacoli non saltabili
  - Limitare sequenze di ostacoli consecutivi a max 3 senza pausa
  - Mai posizionare ostacoli direttamente dopo una biforcazione
- **Nemici**:
  - Distanza minima tra gruppi di nemici: 5 celle
  - Nemici volanti possono sovrapporsi ad altri tipi di entità
  - Max 2 nemici aggressivi (Inseguitori) in sequenza
  - Distanza minima di 8 celle tra nemici di resistenza alta
- **Collezionabili**:
  - Distribuire monete formando pattern (linee, curve, zigzag)
  - Power-up solo in celle libere da ostacoli e nemici
  - Gemme e frammenti in posizioni che richiedono manovre rischiose
  - Kit medici posizionati dopo sezioni difficili

4.3 Configurazione di Randomizzazione

csharp

```csharp
[Serializable]
public struct LevelRandomizationConfig
{
    // Seed per la generazione pseudo-casuale
    public int Seed;

    // Pool di elementi per il livello
    public Dictionary<string, int> ObstacleCounts;
    public Dictionary<string, int> EnemyCounts;
    public Dictionary<string, int> CollectibleCounts;

    // Curva di difficoltà (0.0-1.0 per ogni quarto di livello)
    public float[] DifficultyCurve;

    // Densità di elementi (elementi per segmento)
    public float ObstacleDensity;
    public float EnemyDensity;
    public float CollectibleDensity;

    // Probabilità di biforcazioni
    public float BifurcationProbability;

    // Flag di ambienti speciali
    public bool HasWaterSections;
    public bool HasLavaSections;
    public bool HasIceSections;
    public bool HasDigitalSections;
}
```

4.4 Esempi di Configurazione per Livelli

Livello Tutorial: "Primi Passi"

```csharp
var tutorialConfig = new LevelRandomizationConfig
{
    Seed = 12345,
    ObstacleCounts = new Dictionary<string, int>
    {
        {"U01", 3}, // Barriere Basse
        {"U02", 1}, // Barriera Alta
        {"U03", 2}, // Gap Stretto
    },
    EnemyCounts = new Dictionary<string, int>
    {
        {"EN01", 1}, // Drone Base
    },
    CollectibleCounts = new Dictionary<string, int>
    {
        {"COL01", 30}, // Monete Standard
        {"POW01", 1},  // Kit Medico
    },
    DifficultyCurve = new float[] {0.1f, 0.2f, 0.2f, 0.1f},
    ObstacleDensity = 0.2f,
    EnemyDensity = 0.05f,
    CollectibleDensity = 0.4f,
    BifurcationProbability = 0.0f, // No biforcazioni nel tutorial
};
```

Livello 3 - Mondo Urbano: "Quartiere Residenziale" (Mid-Boss)

csharp

```csharp
var urbanLevel3Config = new LevelRandomizationConfig
{
    Seed = System.DateTime.Now.Millisecond, // Seed casuale
    ObstacleCounts = new Dictionary<string, int>
    {
        {"U01", 8}, {"U02", 6}, {"U03", 5}, {"U05", 3},
        {"C01", 4}, {"C02", 5}, {"C03", 3}, {"C05", 4},
    },
    EnemyCounts = new Dictionary<string, int>
    {
        {"EN01", 3}, {"EN03", 2},
        {"CN01", 3}, {"CN03", 2}, {"CN04", 2}, {"CN05", 1},
    },
    CollectibleCounts = new Dictionary<string, int>
    {
        {"COL01", 80}, {"COL02", 12}, {"COL03", 3},
        {"COL04", 1}, {"POW01", 2}, {"POW02", 1},
    },
    DifficultyCurve = new float[] {0.3f, 0.5f, 0.7f, 0.9f},
    ObstacleDensity = 0.4f,
    EnemyDensity = 0.25f,
    CollectibleDensity = 0.6f,
    BifurcationProbability = 0.5f, // 50% di probabilità di biforcazione
};
```

5. IMPLEMENTAZIONE TECNICA

5.1 Componenti ECS per la Randomizzazione

csharp

```csharp
// Componente per il seed e configurazione di livello
public struct LevelRandomizationComponent : IComponentData
{
    public int Seed;
    public BlobAssetReference<LevelDefinitionBlob> LevelDefinition;
    public float DifficultyMultiplier;
    public bool GenerationComplete;
}

// Componente per la definizione di una cella
public struct CellComponent : IComponentData
{
    public int CellIndex;
    public int LaneIndex;
    public int SegmentIndex;
    public CellType Type;
    public Entity OccupyingEntity;
    public bool IsOccupied;
}

// Componente per gli elementi posizionabili
public struct PlaceableObjectComponent : IComponentData
{
    public PlaceableType Type;
    public string ObjectID;
    public float Difficulty;
    public BlobAssetReference<PlaceableAttributesBlob> Attributes;
}
```

5.2 Sistemi ECS per la Randomizzazione

csharp

```csharp
// Sistema per generare la struttura del livello
public class LevelGenerationSystem : SystemBase
{
    // Crea la struttura base del livello con biforcazioni
    protected override void OnCreate() {...}
    protected override void OnUpdate() {...}
    private void GenerateBaseLevelStructure(ref LevelRandomizationComponent randomizat
    private void CreateSegments(int segmentCount, int lanesCount) {...}
    private void CreateBifurcations(float probability) {...}
}

// Sistema per il posizionamento degli elementi
public class ElementPlacementSystem : SystemBase
{
    // Posiziona ostacoli, nemici e collezionabili secondo la configurazione
    protected override void OnCreate() {...}
    protected override void OnUpdate() {...}
    private void PlaceObstacles(Dictionary<string, int> obstacleCounts) {...}
    private void PlaceEnemies(Dictionary<string, int> enemyCounts) {...}
    private void PlaceCollectibles(Dictionary<string, int> collectibleCounts) {...}
    private Entity CreateEntityFromID(string id, float3 position) {...}
}

// Sistema per la validazione del livello
public class LevelValidationSystem : SystemBase
{
    // Verifica che il livello sia completabile
    protected override void OnCreate() {...}
    protected override void OnUpdate() {...}
    private bool ValidatePath(int laneIndex, int startSegment, int endSegment) {...}
    private bool CheckForBlockingCombinations() {...}
    private void ApplyCorrections() {...}
}
```

5.3 Esempio di Flusso di Generazione

csharp

```csharp
// Pseudocodice per il flusso di generazione di un livello
public void GenerateRandomizedLevel(LevelRandomizationConfig config)
{
    // 1. Inizializzazione del seed e configurazione
    var randomEntity = entityManager.CreateEntity();
    entityManager.AddComponentData(randomEntity, new LevelRandomizationComponent
    {
        Seed = config.Seed,
        DifficultyMultiplier = 1.0f,
        GenerationComplete = false
    });

    // 2. Generazione della struttura base con biforcazioni
    levelGenerationSystem.GenerateBaseLevelStructure(randomEntity, config);

    // 3. Posizionamento degli elementi
    elementPlacementSystem.PlaceObstacles(randomEntity, config.ObstacleCounts);
    elementPlacementSystem.PlaceEnemies(randomEntity, config.EnemyCounts);
    elementPlacementSystem.PlaceCollectibles(randomEntity, config.CollectibleCounts);

    // 4. Validazione
    bool isValid = levelValidationSystem.ValidateLevel(randomEntity);
    if (!isValid)
    {
        levelValidationSystem.ApplyCorrections(randomEntity);
    }

    // 5. Finalizzazione
    entityManager.SetComponentData(randomEntity, new LevelRandomizationComponent
    {
        // Componenti aggiornati...
        GenerationComplete = true
    });
}
```

6. CONSIDERAZIONI PER L'IMPLEMENTAZIONE

6.1 Prestazioni

- Eseguire la generazione asincrona all'inizio del livello
- Pre-caricare e istanziare gli elementi più comuni
- Utilizzare object pooling per elementi ricorrenti

- Implementare il level of detail per elementi distanti

6.2 Testing e Debugging

- Sviluppare una modalità editor che mostri la struttura delle celle
- Implementare un registro visivo dei posizionamenti e decisioni
- Creare strumenti per salvare e riprodurre seed interessanti
- Visualizzatori di densità e difficoltà per sezioni del livello

6.3 Estensibilità

- Sistema di data-driven design per aggiungere facilmente nuovi elementi
- Parser per importare configurazioni da file JSON o ScriptableObjects
- Hooks per eventi speciali durante la generazione

6.4 Casi Speciali

- Boss Arena: Non randomizzata, design fisso con regole specifiche
- Tutorial: Randomizzazione limitata, focus su insegnare meccaniche
- Eventi Narrativi: Trigger e checkpoint in posizioni fisse

7. LINEE GUIDA PER L'IMPLEMENTAZIONE IN UNITY

7.1 Architettura dei Prefab

7.1.1 Struttura delle Celle

Per minimizzare il lavoro lato Unity, organizzeremo i prefab in modo gerarchico:

```
Prefabs/
  Cells/
    Base/
      BaseCell.prefab        // Cella base generica
      JunctionCell.prefab    // Cella di biforcazione
    Environments/
      Urban/
        UrbanBaseCell.prefab
        UrbanJunctionCell.prefab
      Forest/
        ForestBaseCell.prefab
        ForestJunctionCell.prefab
       [Altri ambienti...]
```

Ogni cella è un prefab con:

- Mesh di base (terreno/pavimento)
- Collider semplice (box collider)
- Punti di ancoraggio per elementi (spawn points)
- Trigger di transizione

7.1.2 Pool di Elementi Posizionabili

Tutti gli elementi che possono essere posizionati nelle celle vengono organizzati in pool:

```
Prefabs/
  Placeables/
    Obstacles/
      Universal/
        U01_LowBarrier.prefab
        U02_HighBarrier.prefab
        [...]
      Urban/
        C01_Car.prefab
        [...]
      [Altri ambienti...]
    Enemies/
      Universal/
        EN01_BaseDrone.prefab
        [...]
      Urban/
        CN01_SurveillanceDrone.prefab
        [...]
      [Altri ambienti...]
    Collectibles/
      COL01_StandardCoin.prefab
      [...]
```

Ogni prefab deve includere:

- Modello 3D
- Collider appropriati
- Componenti Unity di base
- Script MonoBehaviour di collegamento al sistema ECS
- VFX e audio opzionali

7.2 Integrazione ECS-Unity

7.2.1 Sistema di Conversione dei Prefab

Per minimizzare il codice lato Unity, usiamo un sistema automatizzato di conversione:

```csharp
// Script da allegare ai prefab per il collegamento automatico al sistema ECS
public class PlaceableAuthoring : MonoBehaviour, IConvertGameObjectToEntity
{
    // Identificatore univoco mappato al catalogo degli elementi
    public string elementID;

    // Attributi specifici dell'elemento (opzionali)
    public float difficulty = 1.0f;
    public bool isDestructible = false;

    public void Convert(Entity entity, EntityManager dstManager, GameObjectConversionS
    {
        // Aggiunge i componenti ECS necessari in base all'ID
        var attributes = ElementCatalog.GetAttributesForID(elementID);

        dstManager.AddComponentData(entity, new PlaceableObjectComponent
        {
            ObjectID = elementID,
            Type = attributes.Type,
            Difficulty = difficulty
        });

        // Aggiunge altri componenti in base al tipo di elemento
        switch (attributes.Type)
        {
            case PlaceableType.Obstacle:
                AddObstacleComponents(entity, dstManager, attributes);
                break;
            case PlaceableType.Enemy:
                AddEnemyComponents(entity, dstManager, attributes);
                break;
            // Altri casi...
        }
    }

    private void AddObstacleComponents(Entity entity, EntityManager dstManager, Elemen
    private void AddEnemyComponents(Entity entity, EntityManager dstManager, ElementAt
}
```

7.2.2 Sistema di Registro Centralizzato

Un catalogo centralizzato mantiene la corrispondenza tra ID e prefab:

csharp

```csharp
// ScriptableObject che mappa gli ID agli asset prefab
[CreateAssetMenu(fileName = "ElementCatalog", menuName = "Runaway Heroes/Element Catal
public class ElementCatalogSO : ScriptableObject
{
    [Serializable]
    public class ElementEntry
    {
        public string elementID;
        public GameObject prefab;
        public PlaceableType type;
        public ElementAttributes attributes;
    }

    public List<ElementEntry> entries = new List<ElementEntry>();

    // Dictionary per accesso rapido in runtime
    private Dictionary<string, ElementEntry> _lookupTable;

    public void Initialize()
    {
        _lookupTable = entries.ToDictionary(e => e.elementID, e => e);
    }

    public GameObject GetPrefabForID(string id)
    {
        if (_lookupTable.TryGetValue(id, out var entry))
            return entry.prefab;

        Debug.LogWarning($"No prefab found for ID: {id}");
        return null;
    }

    public ElementAttributes GetAttributesForID(string id)
    {
        if (_lookupTable.TryGetValue(id, out var entry))
            return entry.attributes;

        Debug.LogWarning($"No attributes found for ID: {id}");
        return default;
    }
}
```

7.2.3 Factory per la Creazione di Entità

Una factory centralizzata gestisce la creazione delle entità in Unity:

csharp

```
public class EntitySpawnSystem : SystemBase
{
    private ElementCatalogSO _catalog;
    private BlobAssetStore _blobAssetStore;
    private GameObjectConversionSettings _settings;

    protected override void OnCreate()
    {
        _catalog = Resources.Load<ElementCatalogSO>("ElementCatalog");
        _catalog.Initialize();

        _blobAssetStore = new BlobAssetStore();
        _settings = GameObjectConversionSettings.FromWorld(World, _blobAssetStore);
    }

    protected override void OnUpdate()
    {
        // Cerchiamo richieste di spawn
        Entities.WithAll<SpawnRequestComponent>().ForEach((Entity requestEntity, ref S
        {
            // Otteniamo il prefab dall'ID
            var prefab = _catalog.GetPrefabForID(request.ElementID);

            if (prefab != null)
            {
                // Convertiamo il prefab in un'entità
                var prefabEntity = GameObjectConversionUtility.ConvertGameObjectHierar

                // Creiamo l'istanza finale
                var instance = EntityManager.Instantiate(prefabEntity);

                // Posizioniamo l'entità
                EntityManager.SetComponentData(instance, new Translation
                {
                    Value = request.Position
                });

                // Notifichiamo che l'elemento è stato creato
                if (request.CallbackEntity != Entity.Null)
                {
                    EntityManager.AddComponentData(request.CallbackEntity, new SpawnCo
                    {
                        RequestEntity = requestEntity,
                        SpawnedEntity = instance
                    });
                }
```

```
            }

            // Rimuoviamo la richiesta di spawn
            EntityManager.DestroyEntity(requestEntity);
        }).WithoutBurst().Run();
    }

    protected override void OnDestroy()
    {
        _blobAssetStore.Dispose();
    }
}
```

7.3 Implementazione del Level Generator in Unity

7.3.1 ScriptableObject per la Configurazione del Livello

Creiamo un asset per ogni configurazione di livello:

csharp

```
[CreateAssetMenu(fileName = "LevelConfig", menuName = "Runaway Heroes/Level Configurat
public class LevelConfigurationSO : ScriptableObject
{
    // Parametri generali
    public WorldType worldType;
    public int levelIndex;
    public string levelName;
    public Difficulty difficulty;

    // Parametri di randomizzazione
    public int defaultSeed;
    public bool useRandomSeed = true;
    public float[] difficultyCurve = new float[4] {0.3f, 0.5f, 0.7f, 0.8f};

    // Biforcazioni
    public int minBifurcations = 1;
    public int maxBifurcations = 3;
    public float bifurcationProbability = 0.5f;

    // Pool di elementi (serializzati come array per l'editor Unity)
    [Serializable]
    public class ElementCount
    {
        public string elementID;
        public int count;
    }

    public ElementCount[] obstacles;
    public ElementCount[] enemies;
    public ElementCount[] collectibles;

    // Converti in Dictionary per il sistema di generazione
    public Dictionary<string, int> GetObstacleCounts()
    {
        return obstacles.ToDictionary(o => o.elementID, o => o.count);
    }

    public Dictionary<string, int> GetEnemyCounts()
    {
        return enemies.ToDictionary(e => e.elementID, e => e.count);
    }

    public Dictionary<string, int> GetCollectibleCounts()
    {
        return collectibles.ToDictionary(c => c.elementID, c => c.count);
    }
```

```
    // Genera una configurazione di randomizzazione
    public LevelRandomizationConfig CreateRandomizationConfig()
    {
        return new LevelRandomizationConfig
        {
            Seed = useRandomSeed ? UnityEngine.Random.Range(1, 999999) : defaultSeed,
            ObstacleCounts = GetObstacleCounts(),
            EnemyCounts = GetEnemyCounts(),
            CollectibleCounts = GetCollectibleCounts(),
            DifficultyCurve = difficultyCurve,
            BifurcationProbability = bifurcationProbability,
            // Altri parametri...
        };
    }
}
```

7.3.2 Bootstrap del Livello

Componente Unity che avvia la generazione del livello:

csharp

```
public class LevelBootstrap : MonoBehaviour
{
    // Riferimento alla configurazione del livello
    public LevelConfigurationSO levelConfig;

    // Riferimento al world ECS
    private World _ecsWorld;

    void Start()
    {
        _ecsWorld = World.DefaultGameObjectInjectionWorld;

        // Otteniamo la configurazione dal livello
        var config = levelConfig.CreateRandomizationConfig();

        // Prepariamo i chunk di memoria per i dati del livello
        var levelEntity = _ecsWorld.EntityManager.CreateEntity();

        _ecsWorld.EntityManager.AddComponentData(levelEntity, new LevelRandomizationCo
        {
            Seed = config.Seed,
            GenerationComplete = false
        });

        // Passiamo tutti i dati della configurazione
        _ecsWorld.EntityManager.AddComponentData(levelEntity, new LevelConfigurationCo
        {
            WorldType = (int)levelConfig.worldType,
            LevelIndex = levelConfig.levelIndex,
            DifficultyLevel = (int)levelConfig.difficulty,
            // Altri parametri...
        });

        // Aggiungiamo le pool di elementi come buffer
        var obstacleBuffer = _ecsWorld.EntityManager.AddBuffer<ObstaclePoolElement>(le
        foreach (var entry in config.ObstacleCounts)
        {
            obstacleBuffer.Add(new ObstaclePoolElement
            {
                ElementID = new FixedString64Bytes(entry.Key),
                Count = entry.Value
            });
        }

        // Buffer per nemici e collezionabili (simile a ostacoli)...
```

```
        // Aggiungiamo un tag per indicare che il livello è pronto per la generazione
        _ecsWorld.EntityManager.AddComponent<LevelGenerationTag>(levelEntity);

        Debug.Log($"Level generation started with seed: {config.Seed}");
    }
}
```

7.3.3 Utilizzo dei Chunks per Ottimizzare le Performance

Per massimizzare le performance con ECS:

csharp

```csharp
// JobSystem per la generazione parallela
[BurstCompile]
public struct GenerateSegmentJob : IJobParallelFor
{
    [ReadOnly] public NativeArray<SegmentDefinition> SegmentDefinitions;
    [ReadOnly] public int CellsPerSegment;
    [ReadOnly] public int LanesCount;

    [WriteOnly] public NativeArray<CellData> CellsOutput;

    public void Execute(int segmentIndex)
    {
        var segment = SegmentDefinitions[segmentIndex];
        var baseIndex = segmentIndex * CellsPerSegment * LanesCount;

        for (int cellIndex = 0; cellIndex < CellsPerSegment; cellIndex++)
        {
            for (int laneIndex = 0; laneIndex < LanesCount; laneIndex++)
            {
                var outputIndex = baseIndex + (cellIndex * LanesCount) + laneIndex;

                // Impostiamo i dati della cella
                CellsOutput[outputIndex] = new CellData
                {
                    CellIndex = cellIndex,
                    LaneIndex = laneIndex,
                    SegmentIndex = segmentIndex,
                    Type = segment.IsBifurcation && cellIndex == 0 ? CellType.Junction
                    IsOccupied = false,
                    // Altri dati...
                };
            }
        }
    }
}
```

7.4 Best Practices per l'Ottimizzazione Unity-ECS

1. **Hybrid ECS**:
   - Utilizzare componenti MonoBehaviour solo per l'authoring e la visualizzazione
   - Mantenere tutta la logica di gameplay nel sistema ECS
   - Utilizzare GameObjectConversionSystem per convertire automaticamente prefab in entità

2. **Object Pooling**:
   - Implementare un sistema di pooling per gli elementi più utilizzati (monete, ostacoli comuni)
   - Gestire le entità disattivate nel pool tramite tag ECS
   - Riutilizzare gli oggetti Unity quando possibile invece di distruggerli/ricrearli
3. **Streaming di Contenuti**:
   - Generare il livello in "chunk" che vengono attivati/disattivati dinamicamente
   - Mantenere in memoria solo gli elementi visibili e quelli imminenti
   - Utilizza la funzionalità di LOD (Level of Detail) di Unity per gli elementi distanti
4. **Serializzazione Efficiente**:
   - Utilizzare BlobAssets per dati immutabili di grandi dimensioni
   - Memorizzare le configurazioni come ScriptableObjects per facile modifica nell'editor
   - Usare strutture compatte per i dati di runtime (FixedString, NativeArray, etc.)

7.5 Strumenti Editor di Supporto

1. **LevelDesigner Window**:
   - Finestra Unity custom per la configurazione di livelli
   - Visualizzazione 2D semplificata della topologia del livello
   - Editor visuale per la distribuzione degli elementi
2. **SeedTester Utility**:
   - Strumento per testare rapidamente diversi seed di generazione
   - Visualizzazione di statistiche sulla distribuzione degli elementi
   - Possibilità di salvare seed interessanti per riutilizzo
3. **AutoValidator**:
   - Sistema che verifica automaticamente la validità di una configurazione
   - Simulazione di percorsi del giocatore attraverso il livello generato
   - Rapporti sulla difficoltà e bilanciamento

8. CONCLUSIONI E RACCOMANDAZIONI

8.1 Priorità di Implementazione

1. Sistema di celle e segmenti base
2. Posizionamento di ostacoli semplici
3. Distribuzione di collezionabili
4. Aggiunta di nemici
5. Implementazione delle biforcazioni

6. Sistema di validazione
7. Integrazione completa con abilità dei personaggi

8.2 Potenziale per il Futuro

- Sistema di difficoltà adattiva basata sulle performance del giocatore
- Modalità challenge con seed condivisi tra la community
- Editor di livelli user-generated content
- Integrazione con sistema di missioni e achievement

8.3 Raccomandazioni Finali

- Mantenere un equilibrio tra randomizzazione e design controllato
- Prioritizzare l'esperienza di gioco piuttosto che la variabilità pura
- Assicurarsi che la progressione di difficoltà rimanga coerente
- Testare ampiamente con diversi stili di gioco
- Raccogliere feedback dai giocatori sui layout generati