Ember’s Edge

Technical Design Document

SACHDEVA, Sarthak (Pixx)

B00007I@student.staffs.ac.uk

Contents

Project Introduction

This project is a fast-paced, first-person parkour platformer where players must traverse a series of challenging obstacles while avoiding the deadly lava floor. The game emphasizes fluid movement, quick reflexes, and creative pathfinding, rewarding players for skilful navigation and speed.

Objective

The goal is to create an engaging and dynamic platformer that provides:

- Satisfying and responsive movement mechanics inspired by games like *Mirror's Edge* and *Clustertruck*.
- Challenging platforming sections with increasing difficulty and various obstacles.
- A sense of urgency and speed, requiring players to think fast and react quickly.

Challenges and Risks

1. Movement System Complexity

- Designing a fluid and responsive parkour system with mechanics like wall-running, vaulting, and dashing can be challenging.
- Ensuring the movement feels intuitive and satisfying while maintaining precise control is a key focus.
- Balancing momentum and physics interactions to avoid exploits or frustrating inconsistencies.

2. Level Design & Flow

- Creating levels that encourage creative movement solutions while maintaining a natural difficulty curve.
- Avoiding overly linear or frustrating designs that could limit player expression.
- Ensuring readability of platforms and obstacles so players can react quickly without confusion.

3. Performance Optimization

- The game requires fast-paced gameplay, so maintaining a high and stable frame rate is crucial.
- Optimizing lava rendering and dynamic physics interactions to prevent performance bottlenecks.
- Ensuring the game runs smoothly on a range of hardware, especially if targeting lower-end PCs.

4. Player Guidance & Accessibility

- Ensuring players can quickly read and react to the environment without excessive UI clutter.
- Creating visual and audio cues that effectively communicate hazards and parkour opportunities.
- Balancing difficulty and accessibility, making the game challenging but not overwhelming.

5. Replayability & Engagement

- Designing a time-based challenge system that encourages players to improve their runs.
- Adding multiple route options for different skill levels without making paths feel unbalanced.
- Implementing leaderboards or ghost replays to keep players engaged beyond initial completion.

6. Testing & Bug Fixing

- Ensuring collision detection works correctly for wall-running and vaulting to avoid game-breaking issues.
- Preventing lava-related bugs, such as players getting stuck or bypassing hazards unintentionally.
- Extensive playtesting to fine-tune movement physics and level layouts.

Risk Mitigation

- Early prototyping of movement mechanics to refine controls before building complex levels.
- Iterative level design with playtesting feedback to ensure balanced difficulty.
- Performance profiling and optimization techniques to maintain smooth gameplay.
- Implementing debugging tools to quickly identify and resolve movement or physics issues.

Hardware Requirements

Minimum System Requirements (Estimated for Smooth Gameplay at Low Settings)

- **OS:** Windows 10 (64-bit)
- **Processor:** Intel Core i5-4590 / AMD Ryzen 3 1200
- **Memory:** 8GB RAM
- **Graphics:** NVIDIA GTX 970 / AMD Radeon RX 570
- **Storage:** 10GB available space
- **DirectX:** Version 11
- **Additional Notes:** SSD recommended for faster load times

Recommended System Requirements ***(For High Settings & Stable Performance)***

- **OS:** Windows 10/11 (64-bit)
- **Processor:** Intel Core i7-9700K / AMD Ryzen 5 5600X
- **Memory:** 16GB RAM
- **Graphics:** NVIDIA RTX 2060 / AMD Radeon RX 6600 XT
- **Storage:** 10GB available space (SSD strongly recommended)
- **DirectX:** Version 12

Development Environment Requirements

- **Game Engine:** Unreal Engine 5 (Latest Stable Version)
- **Software:** Visual Studio 2022, Blender (for assets), Substance Painter (for texturing)
- **Development PC Specs:**
    - **CPU:** Intel Core i9 / AMD Ryzen 9
    - **RAM:** 32GB+
    - **GPU:** NVIDIA RTX 3060 / AMD Radeon RX 6800 XT
    - **Storage:** NVMe SSD for fast asset loading and compilation

Platforms

Target Platform

The game is being developed primarily for **PC**.

Primary Platform:

- **PC (Windows, possibly Linux & Mac)**
- Supports **keyboard & mouse** as well as **controller input**

Engine Specific Specifications and Limitations

Game Engine: Unreal Engine 5

Specifications:

- Rendering: Utilizing Nanite for optimized geometry and Lumen for dynamic lighting (may be scaled down for performance).
- Physics: Using Chaos Physics for accurate movement and collision detection.
- Animation: Animation Blueprint system for fluid movement mechanics (wall-running, vaulting, dashing).
- AI & Navigation: NavMesh for dynamic pathfinding if AI elements are included.

Limitations & Constraints:

1. Performance Optimization

- High-speed movement and physics interactions require constant performance profiling.
- Dynamic lighting and shadows may need optimizations for smooth frame rates.
- Level streaming must be efficient to prevent stuttering in large environments.

2. Disk Space & Memory Usage

- Targeting a game size of ≤3GB to ensure manageable storage needs.
- Reducing texture and asset sizes while maintaining visual quality.
- LOD (Level of Detail) optimization for distant objects to save memory.

3. Platform-Specific Constraints

- PC version: Can support higher visual fidelity, but needs scalable settings for lower-end machines.
- Potential console ports: May require lower texture resolutions, reduced physics complexity, and locked frame rates for stability.

4. Physics & Collision Handling

- High-speed parkour movement may cause unintended clipping or collision issues, requiring precise hitbox adjustments.
- Lava interaction needs optimization to prevent unnecessary physics calculations.

Engine Summary

Engine Version:

- Unreal Engine 5.X (Latest stable version)

Key Features Utilized:

- Nanite (if applicable): For efficient rendering of complex geometry.
- Lumen: For real-time dynamic lighting and shadows.
- Chaos Physics: For handling player movement, parkour mechanics, and object interactions.
- Niagara VFX: For environmental effects like dust, sparks, and lava interactions.

Plugins Used:

- Enhanced Input System – For improved player input handling and rebindable controls.

Systems and Diagrams

This section outlines the core systems driving the gameplay, including movement mechanics, level progression, and game logic. Each system includes a brief explanation and relevant diagrams or tables for clarity.

System 1

System 1: Parkour Movement System

Overview

The parkour system allows players to traverse the environment dynamically through sprinting, jumping, vaulting, wall-running, and dashing.

Core Features

- **Momentum-based movement:** Player speed affects jump distance and vaulting height.
- **Wall-running mechanics:** Detects vertical surfaces and applies movement logic.
- **Vaulting system:** Checks for obstacles within reach and triggers an animation.
- **Dash mechanic:** Short burst of speed with a cooldown.

Flowchart of Movement Logic:

Action	Trigger Condition	Additional Notes
Sprinting	Hold "Sprint" Key	Increases movement speed
Jumping	Press "Jump" Key	Height depends on momentum
Wall-Run	Jump while near a wall	Limited by duration
Vaulting	Detects obstacle < waist height	Triggers vault animation
Dashing	Press "Dash" Key	Short burst, cooldown required

Figure 1 - Movement Flowchart

System 2

Lava Hazard System

Overview

The lava acts as a **persistent environmental hazard** that dynamically interacts with the level and player movement.

Core Features

- **Lava rising mechanic:** Gradually increases difficulty.
- **Instant death on contact:** Forces players to keep moving.
- **Visual & audio cues:** To warn players when lava is near.

Diagram: Lava System State Machine

Parameter	Description	Adjustable?
Speed	Rate at which lava rises	✅
Damage	Instant death on contact	❌
Visual FX	Glow, heat waves, sparks	✅

Figure 2 - Lava system

System 3

Level Progression & Checkpoints

Overview

The game progresses through levels with checkpoints that save progress and respawn players.

Core Features

- **Checkpoints save progress:** Players respawn at the last checkpoint.
- **Timer-based level completion:** Encourages speedrunning.

- **Dynamic difficulty adjustments:** Potential for scaling difficulty based on player performance.

Flowchart: Checkpoint System Logic

Feature	Functionality	Notes
Checkpoints	Saves player position	Placed strategically in levels
Timer System	Tracks time for completion	Can be disabled for casual mode
Difficulty Scaling	Adjusts obstacles dynamically	Based on player performance

Figure 3 - Checkpoint System

Coding Standards

Programming Standards

Since all development is done using **Blueprints**, the focus is on maintaining clean, organized, and efficient visual scripting. Best practices include:

- **Minimizing spaghetti nodes** by using reroute nodes for better readability.
- **Avoiding deep nesting** by breaking complex logic into separate functions or Blueprint components.
- **Using event-driven logic** where possible instead of relying on frequent tick updates.
- **Optimizing Blueprint execution** by reducing unnecessary loops and avoiding excessive casting.

Naming Conventions

Consistent naming conventions help keep Blueprints easy to navigate. Standard rules include:

- **BP_ for Blueprints** (e.g., BP_PlayerCharacter, BP_EnemyAI).
- **BPC_ for Blueprint Components** (e.g., BPC_HealthSystem).
- **WBP_ for UI Widgets** (e.g., WBP_MainMenu).
- **MAC_ for Macros** (e.g., MAC_ApplyDamage).
- **ENUM_ for Enumerations** (e.g., ENUM_PlayerStates).
- **Use PascalCase** for Blueprint names and camelCase for variables (e.g., JumpHeight, playerSpeed).

Style Guide

To maintain readability and consistency:

- **Nodes are aligned properly** with clear flow from left to right.
- **Comment boxes** are used to label different logic sections within a Blueprint.
- **Color coding** is used for wires (e.g., execution flow on top, data flow below).
- **Functions and macros** are used instead of duplicating logic across multiple Blueprints.
- **Collapsed Graphs** are used to clean up large Blueprint scripts.

Commenting Rules

All Blueprints should be **fully commented** to explain their functionality, especially in complex logic sections.

- Every Blueprint event, function, and macro should have a **clear description**.
- Variables should have **tooltips** explaining their purpose.
- Unused or deprecated code should be clearly marked and **not deleted immediately** in case it needs to be restored.

Code Review Procedures

Blueprints will be reviewed regularly to ensure efficiency and maintainability.

- **Self-reviews** before pushing changes to ensure clarity and efficiency.
- **Peer reviews** (if applicable) to check for redundant logic, potential optimizations, and maintain consistency.
- **Performance testing** after significant updates to verify that Blueprint logic doesn't cause frame drops or unnecessary processing overhead.

Production Overview

Moscow

1. **Must-Have (Essential Features)**
   These are the **core mechanics and systems** necessary for the game to function as intended. The game is incomplete without them.
   - Core Movement Mechanics (Jumping, Sprinting, Sliding, Climbing, Wall Running)
   - Hazards & Obstacles (Lava, Swinging Axes, Moving Platforms, Spinning Bus)
   - Punchable Interactions (Doors, Skulls)
   - Grappling Hook Mechanic
   - HUD Elements (Timer, Crosshair, Health System)
   - Win & Lose Conditions
   - Pause Menu
   - Level Progression System (Start, Obstacles, End Portal)
2. **Should-Have (Important but Not Critical)**
   These features improve the experience but are not game-breaking if absent. They enhance gameplay, visuals, or user interaction.
   - Dynamic Main Menu with Animated Background
   - Settings Menu (Adjust Graphics, Controls, Audio)
   - Checkpoints for Longer Levels
   - Post-Processing & Visual Enhancements (Bloom, Color Grading)
   - Additional Level Variations for Replayability
   - Basic Audio & Sound Effects (Jumping, Punching, Lava Sizzle, etc.)

3. **Could-Have (Nice-to-Have, but Non-Essential)**
   These are bonus features that would be great but are not a priority unless time permits.
   - Speedrun Mode (Best times, leaderboards)
   - Advanced Parkour Moves (Wall Kicks, Vaulting)
   - Adaptive Soundtrack (Music intensifies as level progresses)
   - Unlockable Cosmetics for the Player Character
   - Advanced Physics Interactions (Ragdoll on Death, More Destructible Objects)
4. **Won't-Have (Out of Scope for This Project)**
   These features **will not be included** due to time constraints or scope limitations.
   - Multiplayer Mode
   - Procedural Level Generation
   - AI Enemies or NPCs
   - Extensive Story or Narrative Elements
   - Full Character Customization

Timeline

Week	Tasks
Week 1	Project Planning & Documentation (TDD, MoSCoW List)
Week 1-2	Core Movement System (Jumping, Sprinting, Sliding, Wall Running, Climbing)
Week 2	Level Blockout in Photoshop & Unreal Engine
Week 3	Implementing Hazards (Lava, Swinging Axes, Spinning Bus, Moving Platforms)
Week 3-4	Interactable Objects (Punchable Doors, Skulls, End Portal)
Week 4	Grappling Hook Mechanic Implementation
Week 4-5	UI & HUD Development (Timer, Crosshair, Health System)
Week 5	Pause Menu, Win/Lose Screens, Main Menu Layout
Week 6	Audio & Sound Effects Integration
Week 6-7	Level Polish (Lighting, Post-Processing, Visual Effects)
Week 7	Playtesting, Bug Fixing & Optimizations
Week 8	Final Refinements & Submission Preparation

Figure 4 - Gantt Chart