Technical Design Document

Mystic Quest: The Forgotten Realms

Contents

Introduction

Purpose

- The Technical Design Document (TDD) outlines the technical specifications, architecture, and implementation details for “Mystic Quest: The Forgotten Realms”.
- It serves as a reference for the development team, ensuring a consistent approach to building and maintaining the game.

Scope

- This document covers the game’s system architecture, core mechanics implementation, UI/UX design, data management, performance optimization, and tools and technologies used.

System Architecture

Overview

- The game is developed using Unity 2023, leveraging its capabilities for 3D rendering, physics, animation, and UI design.
- The architecture is modular, allowing for flexible development and easy maintenance.

Core Components

- **Game Engine:** Unity 2023.
- **Programming Language:** C#
- **Version Control:** Git (GitHub repository).
- **Platforms:** PC (Windows), Consoles (PS5), and potentially other platforms (Switch, Mobile).

Subsystems

- **Rendering Engine:** Handles all graphics rendering, including lighting, shading, and post-processing effects.
- **Physics Engine:** Manages physics simulations, collision detection, and Rigidbody dynamics.
- **Audio Engine:** Manages all audio components, including background music, sound effects, and voice acting.
- **UI System:** Manages user interface elements and interactions.

- **Networking (if applicable):** Handles multiplayer functionalities, including player synchronization and server-client communications.

Core Mechanics Implementation

Combat System

- **Melee Combat:** Implemented using Unity's Animator for smooth transitions and Rigidbody for physical interactions. Each attack triggers specific animations and hit detection.
- **Ranged Combat:** Uses raycasting for hit detection and Rigidbody for projectile physics. Animations are synced with attack inputs.
- **Abilities:** Scripted in C# with cooldown management and effects triggered through Unity's event system.

Exploration and Interaction

- **Movement:** CharacterController component for player movement, handling walking, running, jumping, and climbing.
- **Interaction:** Raycasting for detecting interactive objects, triggering context-sensitive actions like opening doors or picking up items.

Puzzle-Solving

- **Basic Puzzles:** Scripting logic for puzzles, using triggers and events to manage puzzle states and player feedback.

User Interface (UI and User Experience (UX)

UI Components

- **HUD:** Displays health, mana, experience, and quick-access abilities using Unity's Canvas system.
- **Menus:** Main menu, settings, inventory, and quest log, implemented with Unity UI elements and animations.
- **Tooltips:** Contextual tooltips for items, abilities, and interactive objects.

UX Considerations

- **Navigation:** Clean and intuitive navigation paths, both in menus and in-game.

- **Feedback:** Visual and audio feedback for actions, interactions, and status changes.

Data Management

Game Data

- **Saving and Loading:** JSON-based save system for player progress, including character stats, inventory, and quest completion.
- **Localization:** Support for multiple languages using external localization files.

Asset Management

- **Textures and Models:** Organized in a structured folder hierarchy within the Unity project.
- **Audio Assets:** Managed using Unity's Audio Source components, with appropriate compression settings for optimization.

Performance Optimization

Profiling Tools

- **Unity Profiler:** Used for identifying performance bottlenecks in CPU, GPU, memory, and rendering.
- **External Tools:** NVIDIA Nsight, Intel VTune for hardware-specific optimization.

Optimization Techniques

- **Level of Detail (LOD):** Implement LOD for 3D models to reduce complexity at a distance.
- **Occlusion Culling:** Enabled to avoid rendering objects not visible to the camera.
- **Texture Compression:** Use appropriate compression formats for textures to reduce memory usage.
- **Multithreading:** Utilize Unity's Job System and Burst Compiler for performance-critical tasks.

Tools and Technologies

Development Tools

- **IDE:** Visual Studio 2022 with ReSharper for C# development.
- **Source Control:** Git with GitHub for version control and collaboration.

- **Project Management:** Jira for task management, issue tracking, and sprint planning.

Design Tools

- **3D Modeling:** Blender for creating and optimizing 3D assets.
- **Texturing:** Substance Painter and Photoshop for creating textures and materials.
- **Audio:** Audacity and FMOD for sound editing and integration.

Security and Networking (if applicable)

Security Considerations

- **Data Protection:** Ensure saved game data is encrypted and securely stored.
- **Cheat Prevention:** Implement server-side validation for critical game data and actions.

Networking Architecture (if applicable)

- **Server Architecture:** Dedicated servers using Unity's Transport Layer for managing multiplayer sessions.
- **Synchronization:** Implement predictive and authoritative server-client communication to handle player actions and game state synchronization.

Testing and Quality Assurance

Testing Strategy

- **Unity Testing:** Write unit tests for critical game systems using Unity Test Framework.
- **Automated Testing:** Implement automated tests for common gameplay scenarios.
- **Manual Testing:** Regular playtesting sessions, with focus on core mechanics, UI/UX, and performance.

Bug Tracking

- **Bug Tracking Tool:** Jira for logging, tracking, and prioritizing bugs.
- **Feedback Loop:** Continuous integration and feedback from QA to development for iterative improvement.

Deployment and Distribution

Build Process

- **Continuous integration:** Use Jenkins or GitHub Actions for automated build and deployment processes.
- **Distribution Platforms:** Steam for PC distribution, PlayStation Network, and Xbox Live for console releases.
- **Beta Testing:** Closed beta testing to selected players before public release.

Conclusion

This Technical Design Document outlines the technical foundation for “Mystic Quest: The Forgotten Realms”, providing a comprehensive guide for the development team to ensure consistent and efficient development. By adhering to the specifications and processes detailed here, the team can work cohesively towards creating a polished and engaging game experience.

Note: This TDD serves as a living document, to be updated as the project evolves and new challenges arise. By maintaining clear documentation, the development team can ensure that “Mystic Quest: The Forgotten Realms” is built on a solid technical foundation and is well-prepared for successful development and release.