

Game Design Document & Technical Report

Project Name: Roll-a-Ball Expanded: Time Attack Survival

Developer: Manav Kheni

Date: 02-17-2026

Engine Version: Unity 6 (LTS)

1. Feature Implementation & Deviation from Tutorial

This project expands significantly upon the standard Unity "Roll a Ball" tutorial. While the tutorial provides a basic physics controller and static collectibles, this submission implements a complete "Time Attack Survival" game loop with the following 5 custom enhancements:

A. Advanced Mobility (Jump & Dash)

- **Tutorial:** Only offered basic rolling using `Input.GetAxis`.
- **My Implementation:** I implemented a Jump mechanic using `Physics.Raycast` for ground detection to prevent infinite mid-air jumping. I also added a Dash ability using `Rigidbody.AddForce` with `ForceMode.VelocityChange` (or `Impulse`) to provide an instantaneous burst of speed that ignores the object's mass for snappier movement.

B. Enemy AI (NavMesh Pursuer)

- **Tutorial:** No enemies; single-player only.
- **My Implementation:** Created a new script `EnemyAI.cs` utilizing Unity 6's `NavMeshSurface` component. The enemy uses a `NavMeshAgent` to calculate the shortest path to the player in real-time, navigating around walls and obstacles. I implemented state checks so the enemy stops moving immediately upon Game Over or Victory.

C. Dynamic Hazards (Kinematic Sweeper)

- **Tutorial:** Static walls only.
- **My Implementation:** Designed a "Sweeper Arm" hazard using **Kinematic Physics**. The script `SweeperHazard.cs` uses `rb.MoveRotation` to spin the object with infinite mass, ensuring it pushes the player physically without stopping itself. I also implemented a

Waypoint System allowing the hazard to patrol between defined coordinates while spinning.

D. Data Persistence (High Score System)

- **Tutorial:** No data saving; scores reset on close.
- **My Implementation:** Integrated Unity's PlayerPrefs system to save the user's Best Time to the computer's registry. The PlayerController script compares the current run's time against the saved float (GetFloat("BestTime")) and overwrites it if the new time is faster. This persists data across game sessions.

E. Game Feel & Polish (Audio & Particles)

- **Tutorial:** Silent gameplay with objects simply disappearing.
- **My Implementation:** Implemented a **Singleton** AudioManager to handle sound effects globally. I added visual "Juice" by instantiating a Particle System prefab (Instantiate()) whenever a pickup is collected, and attached a Trail Renderer to the player to visually communicate speed and momentum.

2. Code Documentation: Original vs. Borrowed

To adhere to academic integrity and standards, the code is categorized as follows:

Component	Source Status	Description
Basic Movement	Borrowed / Adapted	The core rb.AddForce logic is derived from the Unity "Roll a Ball" tutorial.
Camera Controller	Borrowed	Standard camera follow script from the Unity tutorial.
EnemyAI.cs	Original	Script written entirely by me to handle NavMeshAgent destinations, player tracking, and rotation logic.
Jump & Dash Logic	Original	Custom logic added to PlayerController.cs utilizing Raycasts for ground checks and Coroutines for dash cooldowns.

SweeperHazard.cs	Original	Custom script implementing Kinematic rotation and Waypoint array logic for patrolling.
Data Persistence	Original	Custom logic using PlayerPrefs to save/load high scores and update the UI.
AudioManager.cs	Original	Custom Singleton implementation to manage AudioSource and prevent duplicate audio listeners.

3. Citations & References

The following resources were used to understand the API and syntax for this project:

1. **Unity Technologies.** (n.d.). *Roll a Ball Tutorial*. Unity Learn.
 - a. *Used for:* Basic setup of the PlayerController and Camera.
2. **Unity Technologies.** (n.d.). *Scripting API: NavMeshAgent*.
 - a. *Used for:* Implementing the Enemy AI pathfinding and NavMeshSurface baking.
 - b. *Source:* docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html
3. **Unity Technologies.** (n.d.). *Scripting API: PlayerPrefs*.
 - a. *Used for:* Implementing the High Score save system.
 - b. *Source:* docs.unity3d.com/ScriptReference/PlayerPrefs.html
4. **Unity Technologies.** (n.d.). *Scripting API: Rigidbody.MoveRotation*.
 - a. *Used for:* Implementing the spinning physics of the Sweeper hazard.
 - b. *Source:* docs.unity3d.com/ScriptReference/Rigidbody.MoveRotation.html