

STEP-BY-STEP GAME DEVELOPMENT MANUAL

GNOME ON A ROPE (GNOME'S WELL)

2D GAME MANUAL

Gnome on a Rope is a 2D physics-based platformer where the player controls a gnome swinging on a rope to avoid traps, collect treasure, and reach the exit.

Group:

Bisquera, Jess Andrei S.

Manangkil. Gerald

Toquero, Dale

PEBSIT004 – IT Elective

Subject

Mr. Jason Nacorda

Adviser

January 07, 2026

Table of Contents

1. Project Overview	3
2. Tools & Technology	3
3. Installation & Setup.....	3
4. Architecture & Code Walkthrough	4
Architecture Diagram Description	4
5. Game Design & UI.....	4
UI Flow Diagram Description.....	4
6. Art & Sound Pipeline	4
7. Deployment & Release Notes.....	4
8. Future Work.....	4
9. Appendices.....	5
Appendix A. Key Code Snippets (Logic Overview)	5
Appendix B. Development Milestones	5
Appendix C. Asset Attribution & Tools	6
Appendix D. External Links & Citations	6

1. Project Overview

1.1 Game Concept Summary

Gnome's Well is a 2D physics-based mobile game designed for short and engaging gameplay sessions. The player controls a gnome suspended by a rope inside a deep well. The main objective is to descend safely by managing rope physics, avoiding traps, and collecting treasures scattered throughout the level.

The game combines simple touch controls with physics-driven movement, allowing players to swing, control descent speed, and react quickly to hazards.

1.2 Target Platform & Device Requirements

- Platform: Android
- Minimum Android Version: Android 8.0 (API Level 26)
- Recommended RAM: 2 GB or higher
- Controls: Touch input and accelerometer

Tools & Technology

2.1 Game Engine

- Unity 2023.3 LTS

2.2 Programming Language

- C#

2.3 Development Tools

- Unity Editor
- Visual Studio
- Git and GitHub (version control)
- Canva (UI wireframes and mockups)

2.4 External Services

- None

(The game focuses on offline gameplay and core mechanics without third-party services.)

3. Installation & Setup

1. Install Unity Hub and Unity 2023.3 LTS.
2. Install Android SDK, NDK, and OpenJDK via Unity Hub.
3. Clone the repository using Git.
4. Open the project folder in Unity Hub.
5. Configure Build Settings for Android and run the project.

4. Architecture & Code Walkthrough

The system follows a modular architecture separating gameplay logic, UI management, and data persistence.

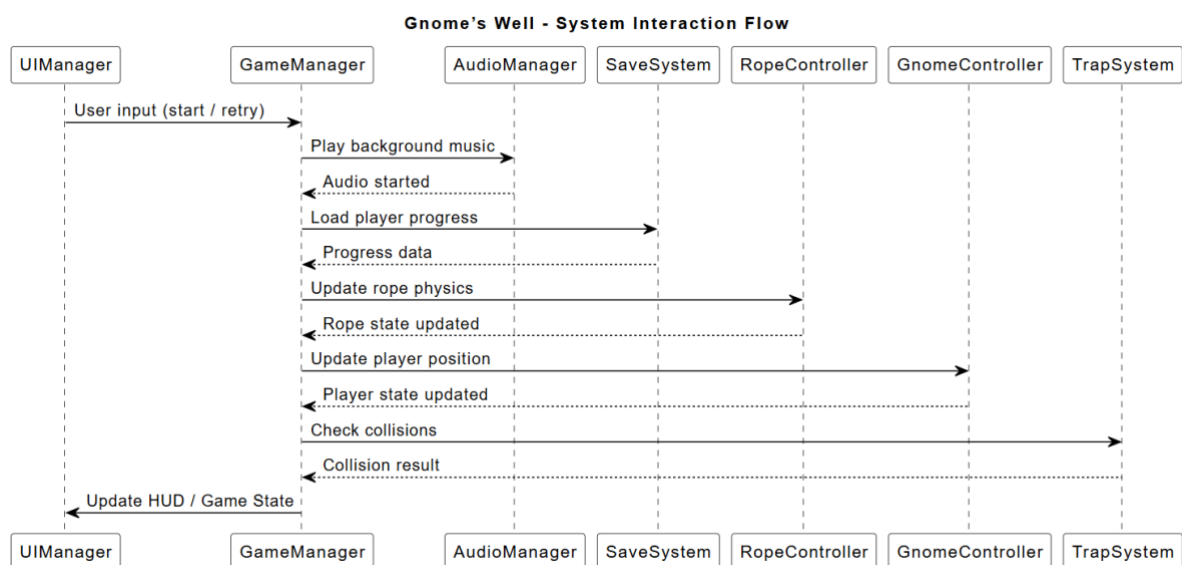


Figure 1.0 Architecture Diagram Description

Figure 1.0 shows the high-level architecture of Gnome's Well. The GameManager serves as the central controller responsible for managing game states and coordinating all subsystems. Input, physics, UI, audio, traps, and data persistence are implemented as independent modules that communicate with the GameManager through public methods and event-driven callbacks.

4.3 Main Modules

GameManager – Controls game state, score tracking, and scene transitions.

Input System – Processes touch and accelerometer input.

Physics & Rope Mechanics – Manages rope constraints, swinging, and gravity.

Trap System – Detects collisions with hazards.

UI Manager – Updates menus, HUD, win, and lose screens.

Audio Manager – Plays sound effects.

Data Persistence – Stores progress and scores locally.

4.4 Folder Structure

Scripts – All C# gameplay and system scripts

Scenes – Menu, Gameplay, Win, and Lose scenes

UI – UI prefabs and controllers

Audio – Sound effects

Sprites – 2D visual assets

4.5 Key Scripts

GameManager.cs – Manages overall game flow and states.

PlayerController.cs – Controls gnome movement and rope physics.

TrapController.cs – Handles collision logic with traps.

UIManager.cs – Updates score, menus, and game result screens.

4.6 Data Persistence

Uses PlayerPrefs to store player scores and basic progress locally on the device.

4.7 Adding a New Level (Step-by-Step)

1. Duplicate an existing gameplay scene.
2. Modify trap placement and layout.
3. Save the scene inside the ****Scenes/**** folder.
4. Add the new scene to ****Build Settings****.

5. Game Design & UI

Core mechanics include rope extension and retraction, accelerometer-based movement, collision-based trap detection, and treasure collection.

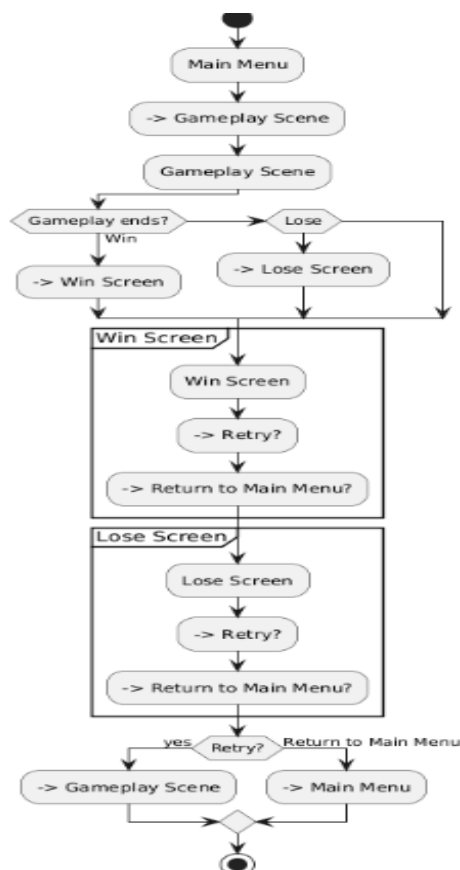


Figure 1.1 UI Flow Diagram Description

The UI flow diagram begins with the Main Menu, leading to the Gameplay Scene. From gameplay, the player transitions to either a Win Screen or Lose Screen, with options to retry or return to the Main Menu.

6. Art & Sound Pipeline

All visual assets are optimized 2D sprites imported at appropriate resolutions. Audio assets are compressed to reduce build size while maintaining clarity.

7. Deployment & Release Notes

The application is packaged as an Android App Bundle (AAB) for Google Play distribution. Version 1.0 includes core gameplay mechanics, traps, and basic UI.

8. Future Work

Planned enhancements include additional trap types, character skins, sound effects, leaderboards, and an iOS release.

9. Appendices

Appendix A. Key Code Snippets (Logic Overview)

1. Rope Control Logic

// Conceptual logic for extending/retracting the rope

```
void Update() {
```

```
    if (Input.GetKey(KeyCode.DownArrow)) {
```

```
        ExtendRope(ropeSpeed * Time.deltaTime); // [cite: 14]
```

```
    } else if (Input.GetKey(KeyCode.UpArrow)) {
```

```
        RetractRope(ropeSpeed * Time.deltaTime); // [cite: 14]
```

```
}
}
```

2. Tilt Movement Logic

// Conceptual logic for dodging traps using accelerometer

```
void FixedUpdate() {

    float tilt = Input.acceleration.x;

    transform.Translate(tilt * moveSpeed, 0, 0); // [cite: 15]

}
```

Appendix B. Development Milestones

PHASE	TASKS	STATUS
RESEARCH	Analysis of the market for physics-based games similar to Cut the Rope.	COMPLETED
DESIGN	Creating wireframes using Canva and establishing a "mischievous" art style.	COMPLETED
STEP 1	Movement mechanics involving the gnome and the rope.	COMPLETED

STEP 2	Adding traps and treasure objects into the game.	COMPLETED
TESTING	Ensuring device compatibility and conducting peer reviews with classmates.	COMPLETED

Appendix C: Asset Attribution & Tools

A summary of the resources used to build the game environment and mechanics.

- Platform: Android OS.
- Engine: Unity.
- UI/UX Design: Canva.com.
- Physics Style: Ragdoll physics with 2D animations.
- Visual Themes: Cartoonish, colorful, and playful.

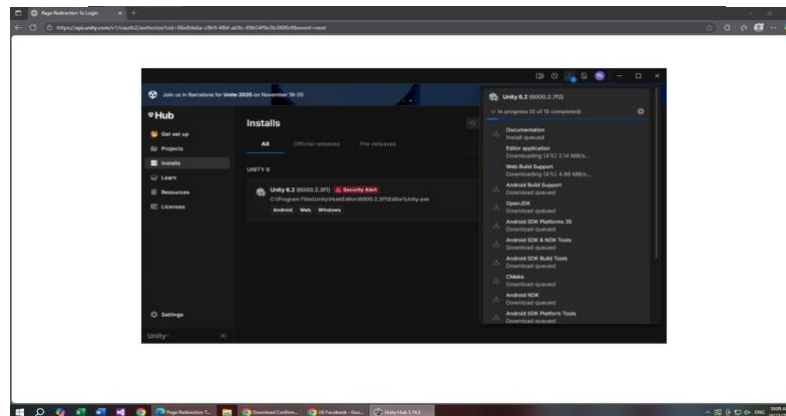
Appendix D: External Links & Citations

- **Unity Documentation:** [Unity Physics 2D](#)
- **Market Inspiration:** *Cut the Rope* and *Happy Wheels*.
- **Repository** [Github.com/Apollo155/PEBSIT004_GnomeOnARope_BisqueraJessAndreis](https://github.com/Apollo155/PEBSIT004_GnomeOnARope_BisqueraJessAndreis).

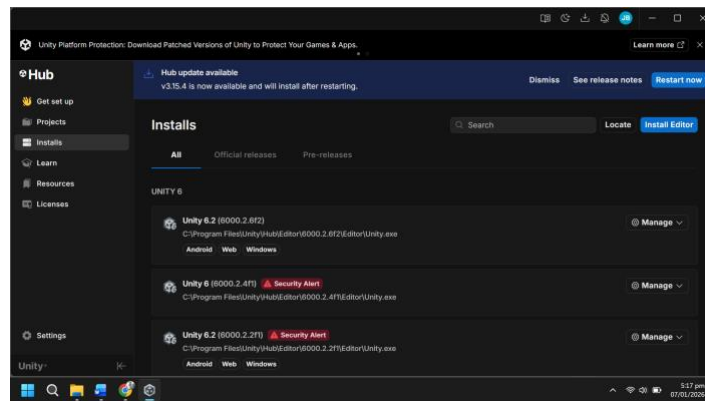
References

- Valente, F. S. *Mobile Game Development with Unity*, 2015

Development Tools Installation



Unity Hub was launched, and a new project was set up using the 2D template. Afterward, the project was renamed to “**Gnome’s Well1.**”



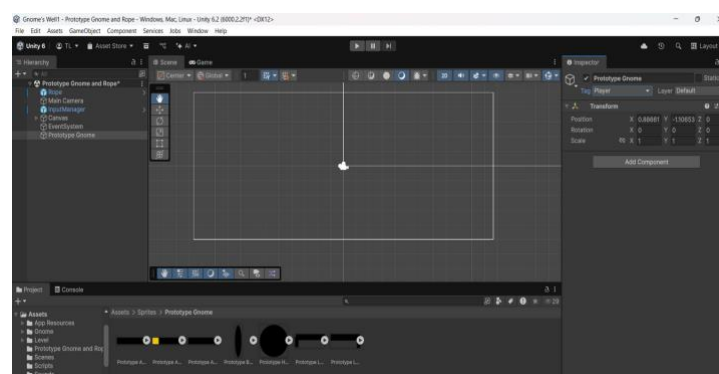
STEP 2 - Asset Preparation and Organization

A ZIP file was downloaded from Secretlab.com, and the 2D Game folder was chosen. After extracting the files, the Assets folder was identified, and the Sprites and Sounds folders were copied into the project's Assets directory. Additional folders were then created within the Assets folder to better organize resources, such as Scripts, Gnome, Level, and App Resources. This process was completed by accessing the project folder on the system and placing the extracted sprite and sound files into their respective directories.



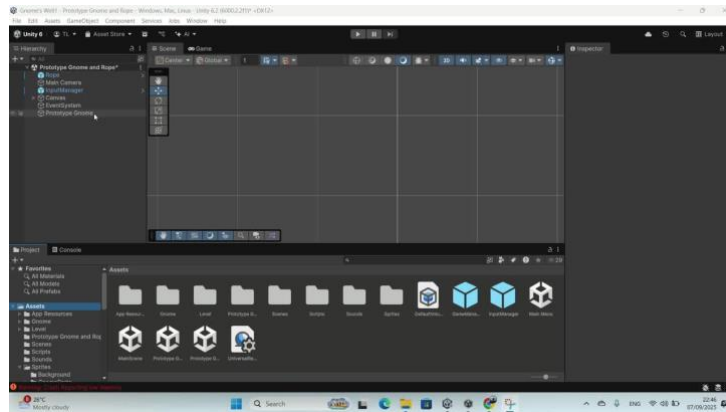
STEP – 3 Prototype Object Creation

A prototype gnome object was created by opening the GameObject menu and selecting Create Empty, after which the object was renamed “Prototype Gnome.”



STEP – 4 Sprite Setup and Hierarchy Arrangement

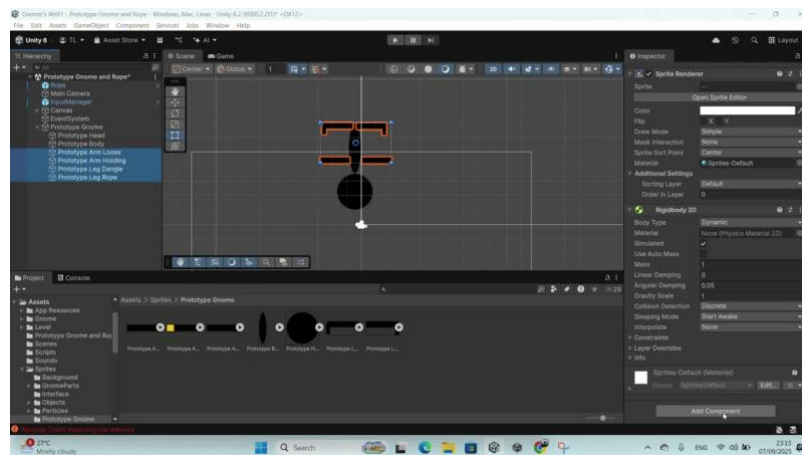
The necessary sprites were added to the scene, including the prototype arm holding, prototype arm loose, prototype body, prototype head, prototype leg dangle, and prototype leg rope. All of these sprites were assigned as child objects under the **Prototype Gnome** object. The sprites were then properly positioned so the character appears in an inverted orientation.



STEP - 5 Rigidbody Configuration

All body part sprites were selected, and a **Rigidbody2D** component was added to each object through the Inspector panel.

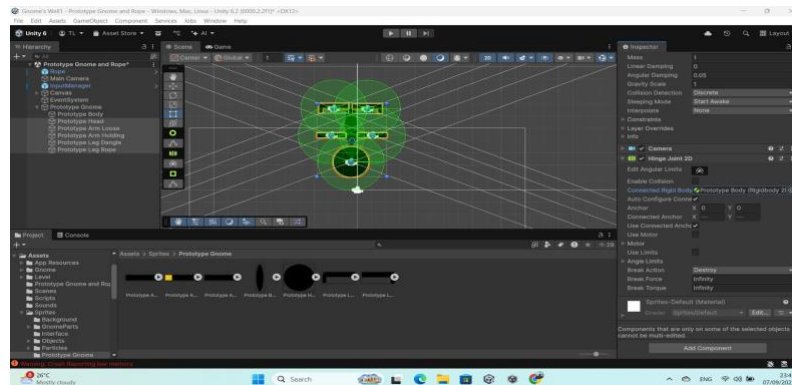
STEP - 6 Collider Assignment



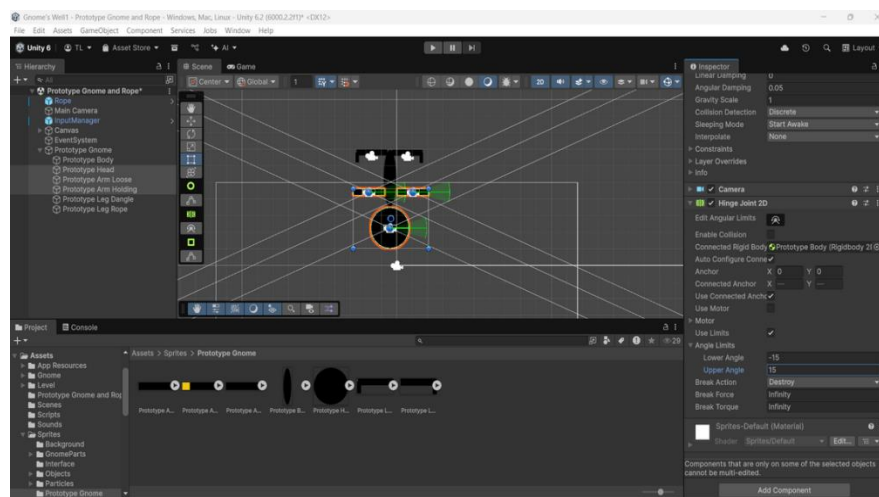
Colliders were added to the body part sprites using the Inspector panel. **BoxCollider2D** components were assigned to the arm and leg sprites. A **CircleCollider2D** component was added to the head sprite. Another **CircleCollider2D** component was added to the body sprite, with the collider radius adjusted to properly fit the body.

STEP - 7 JointConfiguration

All sprites except the body were selected, and a **HingeJoint2D** component was added to each selected sprite. The **Prototype Body** object was then dragged from the Hierarchy panel into the **Connected Rigidbody** field in the Inspector panel.



STEP – 8 Joint Limit Settings

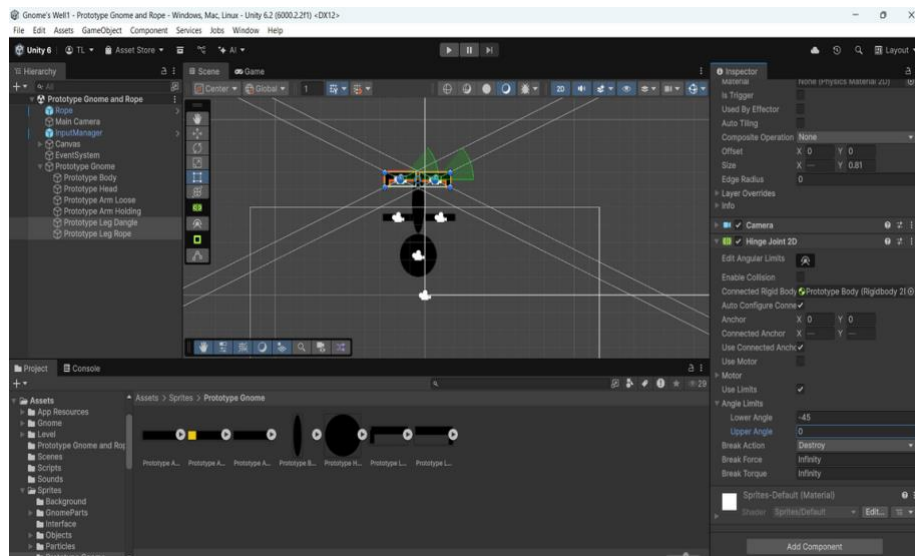


The arm and head sprites were selected, and the **Use Limits** option was enabled. The **Lower Angle** was set to **-15**, and the **Upper Angle** was set to **15**.

The leg sprites were then selected, the **Use Limits** option was enabled, and the **Lower Angle** was set to **-45**, while the **Upper Angle** was set to **0**.

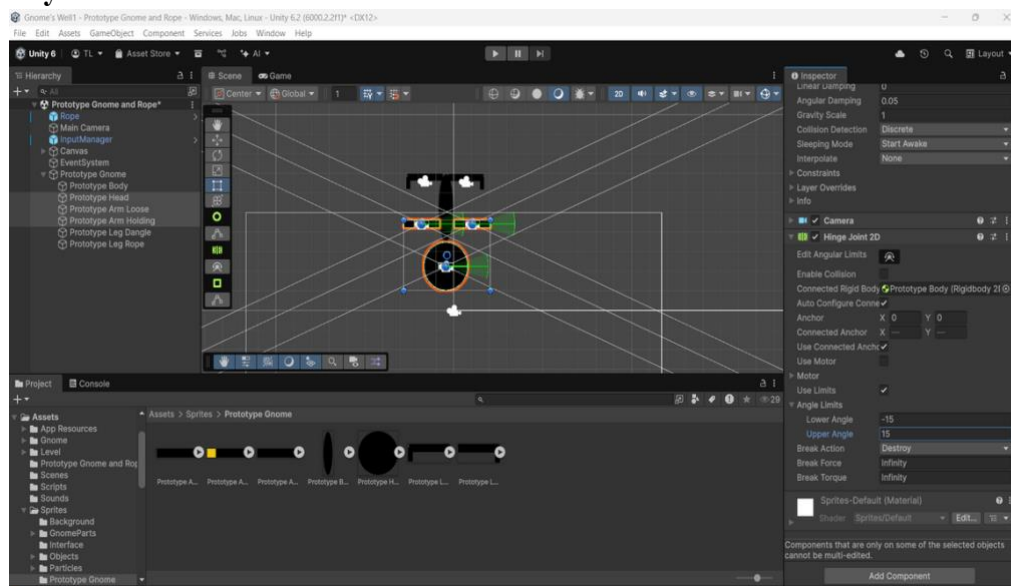
STEP – 9 Joint Position Adjustment

The joint anchor positions were modified to allow proper movement. The arms were attached at the shoulder joints, the legs were connected at the hips, and the head was linked at the base of the neck.



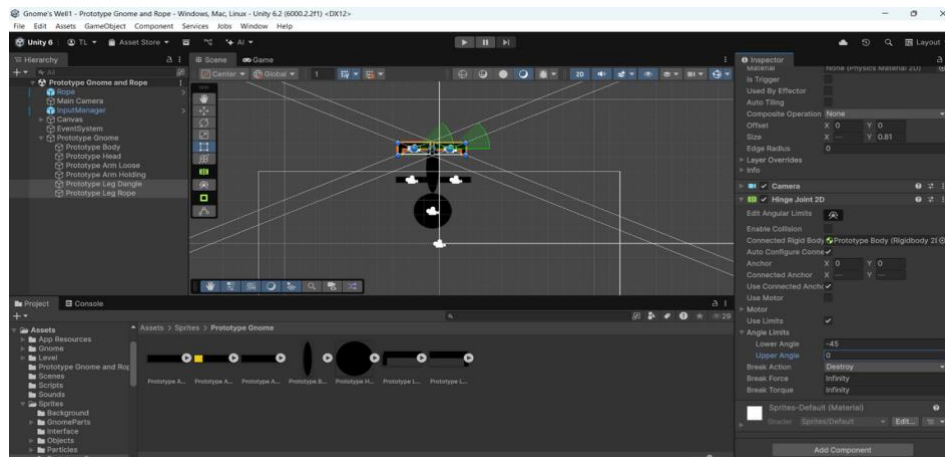
STEP - 10 Spring Joint Setup

The **Prototype Leg Rope** sprite was selected, and a **SpringJoint2D** component was added. The **Auto Configure Distance** option was disabled, the **Distance** was set to **0.01**, and the **Frequency** was set to **5**.



STEP – 11 Rope Segment and Rope Object Creation

A **Rope Segment** object was created, and a **Rigidbody2D** component was added with the mass set to **0.5**. A **SpringJoint2D** component was also added, with the **Damping Ratio** set to **1** and the **Frequency** set to **30**. The object was then converted into a prefab, after which the original rope segment object was deleted from the scene.



STEP – 11 Rope Object Creation

A new empty GameObject named “Rope” was created. Its icon was changed to a red rounded-rectangle shape to make it easily recognizable in the Hierarchy. Two components were added: Rigidbody2D and Line Renderer. The Rigidbody2D was set to Kinematic, and the Line Renderer width was adjusted to 0.075 to visually simulate a rope.

STEP – 12 Rope Script and Material Setup

A script component was attached to the Rope object, and its rope-related settings were configured through the Inspector panel. A new material named “Rope” was created and colored brown. After completing the setup, the project was run to test the rope’s behavior.

STEP – 13 Unity Remote Installation

The Unity Remote application was downloaded and installed from the Google Play Store on a mobile device to enable testing of the gnome game directly on the device.

STEP – 14 Singleton Script Creation

A new empty C# script was created by selecting Create > Scripting > C# Script and naming it “Singleton.” The code provided in the module was then pasted into the script.

STEP – 15 Input Manager Setup

An empty GameObject named “InputManager” was created. An InputManager.cs script was also created, and the provided module code was pasted into it. The script was then attached to the InputManager object.

STEP – 16 Swinging Script Assignment

The Prototype Body object was selected, and a new script named “Swinging.cs” was created and attached. The corresponding module code was pasted into the script and configured using the Inspector panel.

STEP – 17 UI Button Configuration

A UI button was added by selecting GameObject > UI > Button and named “Down.” The button was positioned at the bottom-right of the screen, and its text label was changed to “Down.”

An Event Trigger component was added, and a Pointer Down event was configured. The Rope object’s isIncreasing function was linked to this event. The same steps were repeated for the Up button.

STEP – 18 Camera Follow Script Setup

A camera-follow script named CameraFollow was created, and the provided module code was inserted. The script uses the LateUpdate method to ensure smooth camera movement.

STEP – 19 Gameplay Script Integration

Three scripts were added to the project:

BodyPart.cs for limb behavior and damage handling

Gnome.cs for managing overall gnome behavior

RemoveAfterDelay.cs for automatically removing objects after a set time

The BodyPart component was attached to each gnome limb.

STEP – 20 Bloody Fountain and Gnome Configuration

An empty GameObject named “Bloody Fountains” was created with child objects representing body parts. The Gnome.cs script was attached to the Prototype Gnome. The Prototype Body was assigned as the camera follow target, the Prototype Leg was assigned to the rope body slot, and all required references were set in the Inspector.

STEP – 21 Game Manager Setup and Final Integration

A GameManager object was created and assigned the GameManager.cs and Resettable.cs scripts. This object manages gnome spawning, death resets, and menu control.

A Start Point object was created, the gnome was converted into a prefab, and all references were connected. At this stage, swinging, rope climbing, and game flow were fully functional.

STEP – 22 Signal-on-Touch Script Creation

A new script named SignalOnTouch.cs was created to detect player interactions with traps, treasure, and exits. The default code was removed and replaced with the exact reference-book code. This script sends a signal whenever the player touches an object.

STEP – 23 Trap (Spikes) Setup

The first trap was added by dragging SpikesBrown from Sprites/Objects into the Scene. A PolygonCollider2D and SignalOnTouch script were added.

The On Touch event was linked to GameManager.TrapTouched, causing damage when the gnome touches the spikes. The object was then saved as a prefab in the Level folder.

STEP – 24 Exit Setup

The exit was placed at the top of the well using the Top sprite from Sprites/Background. A BoxCollider2D set as a trigger was added and resized.

The SignalOnTouch script was configured to call GameManager.ExitReached when the gnome reaches the exit.

STEP – 25 Treasure Setup

A SpriteSwapper.cs script was created using the reference-book code. The TreasurePresent sprite was placed near the bottom of the well and given a BoxCollider2D set as a trigger.

The SpriteSwapper component was configured to swap to TreasureAbsent. SignalOnTouch events were linked to GameManager.TreasureCollected and SpriteSwapper.SwapSprite. A Resettable component was added to reset the treasure after death.

STEP – 26 Treasure Testing

The treasure was tested to ensure it disappears upon contact and resets correctly when the gnome dies.

STEP – 27 Background Placeholder Setup

A placeholder background was created using GameObject > 3D Object > Quad, named Background, and positioned behind all objects. It was resized to span the full well depth, creating a clean gray backdrop.

STEP – 28 Gnome Art Update

Gnome artwork was updated using sprites from the GnomeParts folder. All Alive sprites were set to Sprite (2D and UI) texture type.

STEP – 29 Pivot Point Adjustment

Pivot points were adjusted using the Sprite Editor for all gnome parts except the body, placing each pivot at its joint rotation point.

STEP – 30 Gnome Prefab Update

The prototype gnome prefab was updated with Alive sprites and repositioned for better alignment. Sorting orders were set with the head and arms above the body.

STEP – 31 Physics Update

Existing colliders were removed and replaced with PolygonCollider2D components. The body's CircleCollider2D radius was increased, and joint anchors were realigned with pivot points.

STEP – 32 Final Gnome Configuration

The leg rope anchor was adjusted to the ankle, arm-holding sprites were assigned, and the gnome scale was set to 0.3. The prefab was saved and assigned to the GameManager for testing.

STEP – 33 Sorting Layer Setup

New sorting layers were created: Level Background, Level Objects, and Level Foreground.

STEP – 34 Level Background Construction

Background sections were organized, duplicated, rotated, and scaled to build a long vertical level with barriers separating sections.

STEP – 35 Well Bottom Setup

A Well Bottom object was created with bottom and sand wall sprites. The treasure was repositioned at the center of the sand area.

STEP – 36 Camera Adjustment

The Main Camera's orthographic size and movement limits were adjusted to fit the well's height.

STEP – 37 UI Setup

Interface sprites were imported and configured. The Up and Down buttons were updated with sprites, resized, repositioned, and grouped into a Gameplay Menu.

STEP – 38 Menu Screens Setup

Game Over, Pause, and Main Menu screens were created with buttons properly linked to GameManager functions.

STEP – 39 Invincibility Debug Mode

A Debug Menu toggle named Invincible was added and linked to GameManager.gnomeInvincible, preventing death when enabled.

STEP – 40 Themed Spikes Creation

Blue and Red spike variants were created by duplicating SpikesBrown, updating sprites, and resetting colliders.

STEP – 41 Spinning Blade Trap Setup

A spinning blade trap was built using Spinner sprites and layered correctly.

STEP – 42 Spinner Damage Configuration

A CircleCollider2D and SignalOnTouch script were added to the blades, triggering GameManager.TrapTouched.

STEP – 43 Spinner Animation Setup

An Animator component and spinning animation were added, controlled by a custom script.

STEP – 44 Block Prefabs Creation

Colored block sprites were assigned colliders and converted into reusable prefabs.

STEP – 45 Blood Material Setup

A blood texture was configured, and a Blood material using an Unlit/Transparent shader was created.

STEP – 46 Blood Fountain Effect

A Blood Fountain particle system was created, configured, and saved as a prefab.

STEP – 47 Blood Explosion Effect

A burst-style Blood Explosion particle system was created and assigned to the Gnome prefab.

STEP – 48 Main Menu Scene Creation

A new scene named Menu was created with a properly anchored and scaled background.

STEP – 49 New Game Button Setup

A New Game button was added, positioned, and labeled appropriately.

STEP – 50 Loading Overlay Setup

A loading overlay with a semi-transparent background and centered “Loading...” text was created.

STEP – 51 Scene Loading Configuration

A loading script was attached to the Main Camera, and scene order was configured in Build Settings for smooth transitions.

STEP – 52 Sound Effects Integration

Audio sources were added to traps, treasure, and the Game Manager, and appropriate sound effects were assigned to complete the audio system.