Final Project Deliverable I

Executive Summary

The following document outlines the Group 4 Final Project, *Lunar Kingdom*, by Haley Kloss, Hunter Beach, and Mark Johnson. Haley Kloss will be acting as the level designer and GD Extension developer, Hunter Beach will take the positions of UI/UX designer and GD Extension developer, and Mark Johnson will be the tester for the game, as well as writing scripts and doing audio design. The game will be a 2D platformer set in a moonlit forest, with a pixelated art style. The conceit of the game is that the player collects a key and jumps to the end to enter the castle, all while avoiding shooting stars and enemies. By getting to the castle with the key, the player wins the game, and by touching an enemy, hazard, or falling off the map, the player gets a game over. There will be many challenges in the making of this game, as are detailed below.

The key technical challenges that we will face will be in making the GD Extension features dynamic, designing a full level, and enforcing standardized coding practices. In terms of making the GD Extension, multiple aspects may contain obstacles. Among other things, the GD Extension will create an object that bounces naturally when colliding with the 'ground', as well as handling so that these impacts cause events that trigger sounds and particle systems. On these aspects, it may be difficult to calculate the velocity and ground texture to dynamically create a realistic bounce collision. It may also prove difficult to dynamically calibrate the impact events to be appropriate to the energy properties of the objects involved. In terms of the level design, in previous assignments, the length of the game has been smaller, so ensuring that there is enough content to fill at least 3-5 minutes of gameplay could prove to be tricky. Finally, in terms of standardized coding practices, all code must be consistent in its naming conventions, format, and documentation. With three team members who have not worked together before on this project, keeping all conventions standardized will require significant communication and mindful collaboration. These challenges will add difficulty to the programming process, but will all contribute to the value of the game by enhancing the realism of physics and hazard interaction in 2D platformers. Unlike Godot’s built-in rigid body system, our GD Extension dynamically computes bounce responses based on surface material and object energy. This project aims to demonstrate advanced physics integration using Godot’s GDExtension framework.

GD Extension Specifications (3-4 pages)

Functionality

The GD Extension plays a significant role in the overall functionality of *Lunar Kingdom*. The extension has three primary functions: to create a custom physics body that overrides the default collision response that allows objects to bounce dynamically, a class that emits custom signals for impact events, and a falling hazard object. For the physics body that overrides a collision response, the extension will provide a node that does not just collide with a ground object and stop moving immediately. Instead, it will bounce dynamically, using the velocity, object size, ground material, and energy loss involved to calculate the bounce height. The node will continue to bounce less and less until it runs out of energy, and inputs for the bounce calculation, such as the surface texture of a ground object, will be made available for adjustment in the editor.

In terms of the custom signals, this feature will be integrated closely with the bounce node. The system will be a class that emits certain signals when a collision occurs or when an object dissipates after losing all of its bounce energy. This will entirely remove the dissipated object from the game if the developer inputs ‘true’ for a bool called fade_on_stop in the editor. The signals themselves will trigger corresponding particles and sounds to be served to the player. Regarding the falling hazard, this object will have gravity applied so that it descends automatically unless there is ‘ground’ beneath it. It will also be integrated with bounce and impact events and send out a kill signal when touched by the player. This makes it an ideal choice for meteors, raining fireballs, and other such dangers.

The error handling for these functionalities will be important due to the significant use of the GD Extension that the game will require. Some of the errors that will have to be accounted for and handled by the extension are collisions without a defined object size, velocity, or ground texture. These will require that there are default values for each of these variables that the calculations can fall back on, and if these are needed, a warning will be printed in the console. With these mechanisms in place, gameplay will be able to continue without any issues, but there will be a note to the developer in the console that points them in the direction of the issue to resolve.

The purpose of this GD Extension is to produce more realistic and interesting physics and handling for falling objects in platform-style games. Instead of falling and becoming immediately static, this extension will allow objects to land more dynamically and with interesting effects, as well as providing an object type for falling hazards that is automatically integrated with the bounce and event system. This has many benefits for the game that it is incorporated into, such as aesthetic interest and additional challenges for the player. The dynamic bouncing in and of itself creates visual interest through movement, but combined with particle and sound effects through the event system, the extension can contribute extensively to creating an aesthetically pleasing atmosphere for a game. The extension also adds another layer of difficulty due to the movement of the hazards. Since the hazard objects will no longer stay in one place when they fall, players will have time their actions and evasions to account for this motion. By providing these additional features to the game, the GD Extension contributes to creating a more well-rounded and fun game for players.

Technical Architecture and Design Patterns

This GD Extension fits into the observer, factory, and singleton design patterns. In terms of the observer pattern, the impact or dissipation of an object triggers external particle and sound effects. The observer pattern is also invoked when a hazard object emits a kill signal on impact with the player. The factory method is used in this module to generate instances of SurfaceMaterial objects with different hardness values. The singleton design pattern is present in global physics constants such as gravity and default values for the bounce equation.

API Documentation

Falling Hazard Process Overview:

1. Detect a collision between the ground and a BounceNode2D
2. The signal class triggers particles on impact
3. Calculate the velocity of the impact
4. Get the size from the object and the surface factor of the ground material, and determine whether or not to dissipate for the signal class
5. Calculate the bounce and apply to the object
6. Repeat the process until the energy has dissipated

BounceNode2D: a custom node that extends RigidBody2D, creating an object that bounces dynamically when it collides with the ‘ground.’ This node has custom parameters for float decay_rate and float size.

SurfaceMaterial: A new resource that stores and applies surface material properties “soft”, “medium”, and “hard” to allow for custom bounce behavior on it. This node has a custom parameter for float hardness.

BounceResources: A class that retrieves factors needed for dynamic bounce calculations. This class has a String surface_name and a float surface_hardness as parameters.

Impact Events: A class that emits custom signals when a collision occurs or when the object dissipates after losing all of its bounce energy. It will trigger particles and sounds. If bool fade_on_stop is true, the object in question will be removed from the game once the dissipation particles and sound are completed.

HazardObject: A custom node that extends a RigidBody2D, this object has gravity applied, is integrated with bounce and impact events, and sends out a kill signal when the player touches it.

Properties Accessible in Editor:

Float decay_rate

Float size

String surface_name

Float surface_hardness

Bool fade_on_stop

Integration approach with Godot Engine:

This will be implemented as a GDExtension, not a built-in module, for greater flexibility. Once the GD Extension is written, it will be compiled as a library (.gdextension). The classes will be registered with GDCLASS(). The methods and signals will be exposed through ClassDB::bind_method() and ADD_SIGNAL(). The extension builds upon Godot's existing physics engine by creating new objects that have their own custom physics and collision callbacks.

Comparison with existing solutions

The built-in RigidBody2D does provide a static coefficient for bounce, but it does not dynamically calculate bounce strength or emit impact signals. Our GD Extension expands the physics engine of Godot by adding more dynamic, realistic physics combined with visual and auditory appeal.

Game Design Document (3-4 pages)

Game Concept and Genre:

The game will be a 2D platformer with a pixelated fantasy art style. In it, the player is a satyr (https://lucky-loops.itch.io/character-satyr) making their way through the woods of the Lunar Kingdom by hopping across platforms and avoiding hazards. The ultimate goal it to get to and unlock the castle at the end of the level. The path to the castle will include difficult jumps, enemies, and items that the player must collect in order to succeed, such as the key to the castle. There will also be other obstacles, such as deadly stars that rain down from the sky and bounce on the ground until they dissipate. Touching any of these stars as they are falling or bouncing on the ground will kill the player on contact. To win the game, the player must collect the key and make it to the castle doors at the end of the level without falling off the map, being killed by an enemy, or being killed by a falling star.

Core Gameplay Mechanics:

The core movement mechanics of the player character are to idle, run, jump, crouch, and pick up items. Idle will occur when the user is not causing movement or falling, and run will be linked to both the left and right arrow keys as well as the A and D keys. Jump will be linked to both W and the space bar. Crouch will be linked to the down arrow and the S key. These mechanics must enable the player to move from platform to platform to make their way to items and the end of the level; falling off will cause game over.

Outside of the player, key mechanics include the enemy, the falling stars, and the win condition. There will be an enemy patrolling on one of the platforms, and it will lock in on and follow the player when in range. Touching the enemy will kill the player, causing a game over. Falling stars and enemies must be avoided, as touching either will kill the player, causing a game over. To win the game, the player must collect the key by touching it, then bring the key to the castle door at the end of the level. Touching the door with the key in the player’s inventory causes a game win.

How the custom module enhances gameplay:

The custom module will enhance the game in terms of gameplay by causing more realistic movement of obstacles. This extra movement makes it more difficult for the player to evade these obstacles than if they were static, once reaching the ground. This additional challenge adds a level of interest and skill that the game would not otherwise exhibit.
Not only that, but the module also increases the visual appeal of the game. Having moving features instead of just a fixed landscape makes the game more interesting to look at.

Level Layout:

Start Area

- Player Spawn Point (forest clearing)
- Tutorial Sign: “Collect the key and reach the castle and reach the castle gate!”

Platform Sequence

- Platforms 1-3: Basic jumps introducing movements mechanics
- Midpoint Platform: Enemy patrol area
- Floating Platforms: Falling star hazard zone

Item Area

- Key placed on elevated platform guarded by an enemy
- Hint Sign: “Only the worthy with the key may enter the castle”

Final Stretch (potential goals)

- Narrow platforms over a pit; increased falling star frequency
- Visual cue: castle silhouette in background

Castle Gate (Goal)

- Locked door checks player inventory for key
- Win condition triggered if key present

UI Elements Overview:

The user interface (UI) of *Lunar Kingdom* is designed to be minimalistic yet informative, ensuring that the player's focus remains on the platforming challenge while still conveying essential gameplay information. The interface will include the following major elements:

- Start Menu: Displays the title logo and provides three interactive buttons—*Start Game*, *Instructions*, and *Quit*. The Instructions screen outlines basic controls, objectives, and a brief narrative summary.
- In-Game HUD: A small overlay positioned in the top-left corner showing the player's current *Key Status* (empty or acquired icon) and remaining *Lives/Attempts*. In the top-right corner, a subtle timer displays elapsed playtime to encourage replayability.
- Pop-Up Notifications: When the player reaches the castle gate without the key, a message window appears stating "You need the key to enter!" When the key is collected, a short pop-up confirms "Key Acquired!" accompanied by a sound cue.
- Pause Menu: Accessible via the *Esc* key, allowing the player to resume, restart, or exit to the main menu.
- Game Over Screen: Displays a darkened overlay with "Game Over" text, followed by a two-second delay before returning automatically to the main menu.
- Victory Screen: Activated when the player reaches the castle door while holding the key. A celebratory "You Win!" message appears with particle and sound effects before returning to the main menu.

Target audience and platform:

The target audience for this game is people 8 years and older. There is no graphic content or adult themes, and as such is only age-restricted due to difficulty level. The simplicity of a platformer also appeals to a broad range of age groups and gaming skills, as they are extremely common and easy to grasp. The intended platform is PC, as the game is wired for keyboard input.

Visual style and aesthetic direction:

The visual style of the game will be 2D pixel art. The style will incorporate nature and fantasy elements, and be set at night. The feeling evoked by the game should be cozy, nostalgic, and a tiny bit creepy, with cute but odd creature sprites and a player character.

Technical Implementation Plan (2-3 pages)

Development timeline with milestones:

The first deliverable for this project, the Project Design and Specification Document, is due on November 18, 2025, while the second deliverable, the game itself, is due on December 4th, 2025. As a result, our team must plan implementation accordingly, so that it can be finished to a high standard on time. The major milestones and their intended completion dates are as follows:

1. Nov. 16: Finish base game - background, platforms, tilemap
2. Nov. 19: Complete GUI - start menu, game over, game start, key in inventory bool, need key popup
3. Nov. 23: Complete Enemy and Player movement
4. Nov. 25: Module Completed
5. Nov. 30: All implementations of module - falling stars, bouncing, and event systems incorporated seamlessly into the game
6. Dec. 3: Finish testing, submit

Technology stack and dependencies:

Godot 4.5 with Mono for C# integration

C# for scripting

Testing strategy and success metrics:

Strategy: Play through the game and ensure all benchmarks are met:

1. Touching an enemy causes game over
2. Touching a fallen star causes game over
3. Falling off the map causes game over

4. Getting to the end without the key causes a GUI pop-up, not a game win
5. Getting to the end with the key causes the game to be won
6. Play time is at least 3 minutes
7. Player responds accurately to keyboard input
8. All platforms can be walked on without issues
9. No jumps that are meant to be performed are impossible or too difficult to complete in 5 tries or less
10. When the player dies, a game-over screen pops up, then the main menu pops up after 2 seconds
11. When the player wins, a game won screen pops up, then the main menu pops up after 2 seconds
12. Each time the player dies or wins, the game is restarted, with no falling stars on the ground and the key out of the player's inventory and back in the map

Risk assessment and mitigation strategies:

The development of a game comes with several risks for features that might not be possible within the timeframe, or even at all. Some of the risks that we face in our project are integrating the falling hazard objects with the dynamic bounce system and underestimating the time it will take to build the base game for our GD Extension and other elements to build on. To mitigate the potential issue of compatibility between the falling hazard objects and the dynamic bounce system, we will modularize the extension so that the hazard object directly inherits from the bounce class and is designed with that in mind to reduce potential integration conflicts. To mitigate the risk of spending too much time on the base game, we will reuse some of the more basic code and scenes from Haley's Assignment 4, so that more time can be spent on the requirements of the Final Project, rather than the basics that the class has already mastered.

Team Collaboration Plan (1-2 pages)

Role distribution and responsibilities:

Haley Kloss - Create the base game, GUI, and module

Hunter Beach - Code enemy and enemy movement, manage and review GitHub commits

Mark Johnson - Test game, Scripter, Audio design

Communication protocols and meeting schedule:

The primary communication forum that the team will use is a Discord server, set up on the day that teams were announced and shared through the Announcement tab of the Final Project 4 Group on Canvas.

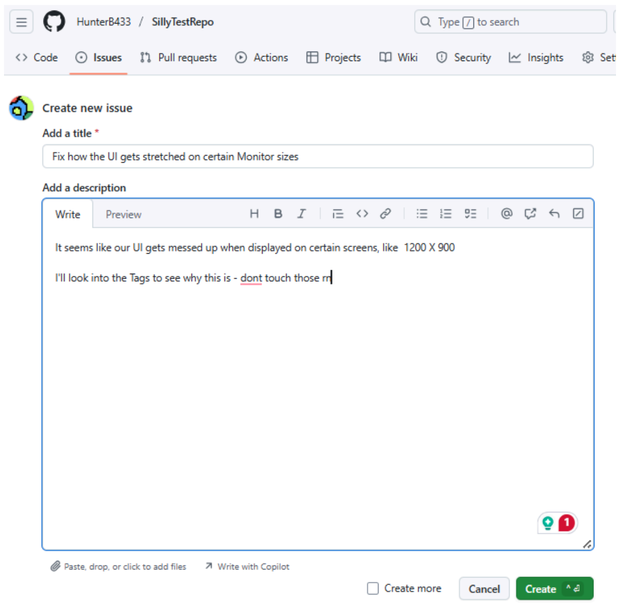
The meeting schedule for the team is to call on Discord voice chat every Tuesday at 6:15 PM.

Check that accepting meetings in Outlook works

GitHub workflow (branching strategy, code review process):

When contributing to our GitHub repository, our team will follow an industry-standard commit scheme for documentation and safe merging. To make a change to the main branch, one must follow this workflow strategy:

1) Make an issue - Give it a clear title on what needs to be done (i.e., "Fix how the UI gets stretched on certain Monitor sizes"). Also, offer a brief 1-2 sentence summary on what is being done in the description; this documentation takes very little time and allows each team member to understand what the others are doing.



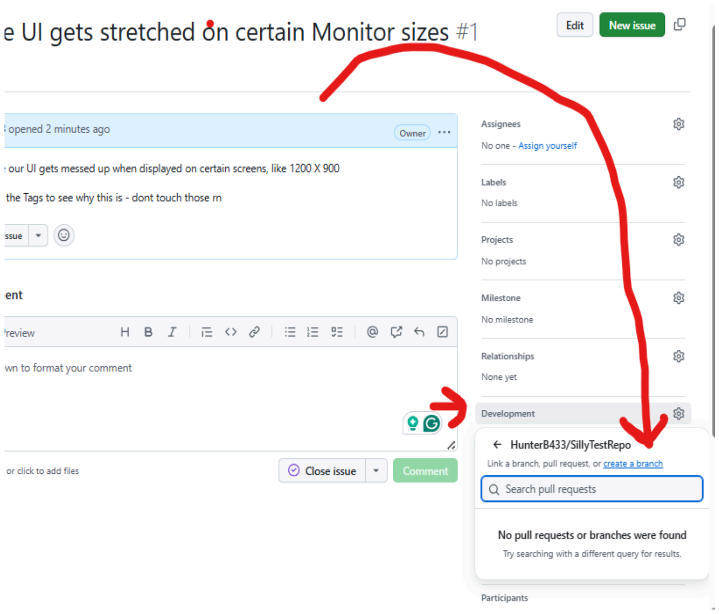
Add a title *

Fix how the UI gets stretched on certain Monitor sizes

Add a description

It seems like our UI gets messed up when displayed on certain screens, like 1200 X 900

I'll look into the Tags to see why this is - dont touch those rn


e UI gets stretched on certain Monitor sizes #1

Edit | New issue

opened 2 minutes ago | Owner

our UI gets messed up when displayed on certain screens, like 1200 X 900

the Tags to see why this is - dont touch those rn

Assignees: No one - Assign yourself

Labels: No labels

Projects: No projects

Milestone: No milestone

Relationships: None yet

Development

HunterB433/SillyTestRepo

Link a branch, pull request, or create a branch

Search pull requests

No pull requests or branches were found

Try searching with a different query for results.

Participants

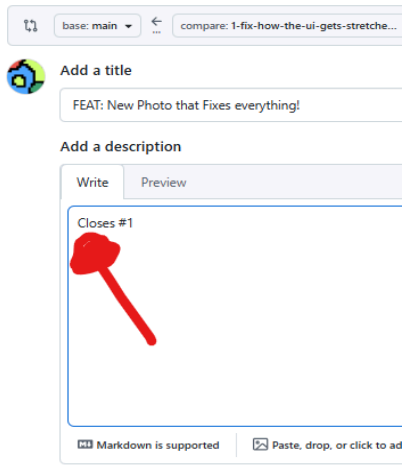
wn to format your comment

or click to add files

Close issue | Comment

2) Make a branch tied to that issue- This is done by clicking on the cog next to ‘development’ in the bottom left and clicking the blue link to create a branch. This helps keep the repo clean, as each branch will have a clear issue it is trying to solve. It also makes developers branch off the main before they begin coding, which forces them to use the most recent code and thus avoids confusion with outdated code.

3) Commit clearly - When finished with a section of code, submit it with a CLEAR commit message, either labeled with “FIX: or FEAT:”. The clear message makes it easier to track what is done in case we need to roll back the repo, with the added clarity of including the “FIX:” or “FEAT:.”


base: main ← compare: 1-fix-how-the-ui-gets-stretche...

Add a title

FEAT: New Photo that Fixes everything!

Add a description

Write | Preview

Closes #1

Markdown is supported | Paste, drop, or click to ad

4) Merge safely - When merging back into main, make sure there are no merge conflicts; if there are, get another person’s help to merge. It may also be beneficial to have another team member review a merge before committing if there are any concerns. Adding “Closes #<issue number>” to the merge description will auto-close the issue.

When this schema is followed correctly, it produces a repo with clear issues, commits, and merges that explain what has been worked on. Furthermore, it eliminates stale branches and issues.

References and Resources (1 page)

Tutorials and documentation consulted:

1. https://docs.godotengine.org/en/4.4/getting_started/first_2d_game/index.html
2. https://www.youtube.com/watch?v=93QBVvCzUGI&list=PLhXFaKLHQJdXpwaNt6gGwpHLTWL0m-TSL&index=9
3. 【ENG】SVWB PV4 CM 18 web 251017 (intro to 2D Platforming in Godot, tutorial)

Inspiration and prior art:

Player Character: https://lucky-loops.itch.io/character-satyr

Background/Platform Texture: https://trixelized.itch.io/starstring-fields

Pixel Platformer · Kenney (Art/Platforms/TileSet)

Free - Pixel Art Asset Pack - Sidescroller Fantasy - 16x16 Forest Sprites by Anokolisa (Fantasy Style)

Sky:

https://free-game-assets.itch.io/free-sky-with-clouds-background-pixel-art-set

Seamless Sky Backgrounds by Screaming Brain Studios (realistic style sky)

Enemy: https://phunguy9.itch.io/random-hallownest-npcs-free-assets, https://zneeke.itch.io/goblin-scout-silhouette, https://samuellee.itch.io/reaper-animated-pixel-art, https://au-pixel.itch.io/free-monster, https://pixramen.itch.io/2d-dino-character-velociraptor, https://caz-creates-games.itch.io/bat, https://jeevo.itch.io/insect-enemies, https://szadiart.itch.io/animated-character-pack,

Star: https://xyezawr.itch.io/free

Castle: https://ruskom.itch.io/moon-castle-asset,

https://malser-gamedev.itch.io/dark-gothic-castle-tiled-background,

https://stext25.itch.io/castle-in-the-fog, https://captainskolot.itch.io/pixel-art-parallax-backgrounds-dark-castle-dungeon-pixel-art-sprite-p

ack

Free Sprite/Texture Assets:

OpenGameArt.org | (Background, Effects)

Search results for 'castle' - itch.io (Tilesets for castle art and sky art)

Kenney's free assets (30,000+ assets) - Game Making Tools (Large library for game tools)