Name: Gayathri Poluri

Class Name: CPSC 5270 Graphics/Game Project (25 SQ)

Git Hub Link: https://github.com/gayathripoluri/Game-project.git

# Prince's Pursuit

**Summary:**

At its heart, Prince's Pursuit isn't just about jumping platforms or beating a timer. It's a story. A journey of a prince named Bhairava, who dives into a cursed forest not to fight monsters, but to prove something deeper: his love for Princess Mitravinda.

The game begins with a simple yet powerful setup. Mitravinda is under a curse. His kingdom, his love, and even his soul feel distant. But he's not the kind to give up. There's only one way to break her curse—by proving his love through action, not just words. So, he runs. Not randomly, but with purpose. To reach Mitravinda, he must collect five hidden gems scattered deep within a cursed forest. Each gem isn't just an item—it represents a piece of his effort. A proof that he's willing to go through chaos to win back what he lost.

But the forest doesn't make it easy.

- The timer ticks 60 seconds—a constant reminder that love doesn't wait forever.
- The ground is uneven, forcing players to adapt quickly and not just run blindly.
- Wolves, cursed by the forest, randomly appear and chase the prince. Every time one hits him, he loses 20 health. It feels personal, like every doubt or pain we face in love.

And those gems? They're not just lying around. They're placed in tricky spots, so players have to explore and take risks, just like Bhairava is doing emotionally.

**What makes the game feel real isn't just its design—it's the choices.**

- If the player collects all 5 gems, it's a happy ending. Mitravinda sees his effort. The curse breaks. Love wins.
- If the player finds only 4 gems, there's still a second chance. A special moment opens—a love letter panel where the player has to write to Mitravinda, from Bhairava's heart. There's no copy-pasting—because love letters should be original, just like real feelings. One chance. That's it.
- But if fewer than 4 gems are collected, it ends in heartbreak. Mitravinda turns away, and the curse lingers. The screen fades into silence—not as a punishment, but as a quiet ending full of "what-ifs".

This game, in many ways, mirrors how love works in real life. We run with purpose. We take hits. We mess up. But what counts is the effort we put in. Prince's Pursuit makes players feel the weight of that effort.

Behind the gameplay is a carefully crafted experience, with custom shaders, sound, randomized enemies, and smart level design. But more than anything, it's the emotional arc that makes it stand out.

It’s not just about finishing the game. It’s about finishing the story—with your choices, your effort, and your heart.

**Primary Aesthetics**

**1)Sensation:**

Prince's Pursuit transforms the aesthetic experience by transporting players to a living, breathing woodland that resembles a living piece of art. The visually stunning ParallaxBackground, which scrolls at 100 pixels/sec *(scroll_offset.x -= scrolling_speed * delta)*, and the Prince's smooth, fluid movements, which are powered by CharacterBody2D with *move_and_slide()* and a *SPEED of 400.0 pixels/sec*, are all carefully designed to appeal to the senses. Layered trees and fireflies *(motion_scale of 0.2 and 0.5)* create a captivating depth. Run, hop, fall, and idle animations made with AnimationPlayer react to player input instantaneously, and sprite flipping *($AnimatedSprite2D.flip_h)* provides helpful feedback. With its dynamic gem count *(gem_label.text = "Gems: %d/%d")* and timer *(timer_label.text = "%02d:%02d")*, the HUD is a masterwork in the parchment style. It turns red below 30 seconds to heighten the tension.Custom shaders such as *gem_pulse* and *player-outline.gdshader*.With the addition of pulsating effects and colorful outlines, gdshader transforms any gem collection into a visual extravaganza. Players will always feel in danger because the health meter changes colors (*green_style* ordinarily, *red_style* when health ≤ 30).

**2: Narrative**

Prince's Pursuit's narrative style is a storytelling success that turns a straightforward platformer into an intensely poignant fairy-tale journey. Every gem that is gathered (*gem_count* monitored in World) reveals a bit of the Princess's enchantment, making the Prince's mission to lift her curse more than just a gaming mechanic. It is a meaningful journey of love and sacrifice. Five gems unlock a joyful, happy ending, fewer than four lead to a heartbreaking, sad ending, and precisely four activate the *LoveLetterPanel*, a narrative innovation where players must write a heartfelt message (>20 characters, including "love" and "princess"). The World script expertly arranges branching endings. The heart-themed submit button (styled via *create_heart_style*) adds a romantic touch, while this panel's AI-driven plagiarism detection (*is_plagiarism_detected*) and disabled copy-paste *(_input event handling*) guarantee authenticity. With the help of AnimationPlayer's Cinematic Endings animations (*happy_end, sad_end*), which are enhanced by bg-tint. gdshader, the forest's cursed atmosphere provides closure with emotional weight. Prince's Pursuit is a storytelling masterpiece that lingers long after the game is over, thanks to its intricate plot, which is expertly combined with time constraints and adversary obstacles.

## Primary Mechanics:

### 1)Movement:

Prince's Pursuit's movement is a masterwork of platforming excellence, offering a physics-driven, responsive, intuitive, and thrilling experience. The Prince, which is implemented in the Player script (CharacterBody2D), moves at a *SPEED of 400.0 pixels/sec*. Jumps are triggered by *ui_up (JUMP_VELOCITY = -400.0),* and direction is controlled by horizontal input (*Input.get_axis("ui_left", "ui_right")*). The frame-independent calculation of gravity *(velocity.y += gravity * delta)* guarantees natural jump arcs for consistency across devices, while move_and_slide() manages collisions with unmatched accuracy, avoiding problems like wall-sticking. By scrolling the forest at 100 pixels per second, the *ParallaxBackground* intensifies immersion by evoking a sensation of unrelenting forward movement**.** Animations *(anim.play("run"), anim.play("jump")*, and sprite flipping *($AnimatedSprite2D.flip_h)* offer visual gloss while syncing smoothly with state changes. This mechanism, which is supported by the *player-outline*, is the foundation of the game and allows players to skillfully traverse roots, gaps, and opponents. gdshader, which raises the bar for 2D platformers in terms of visibility and physics realism.

### 2: Item Collection (Gem Collecting)

Prince's Pursuit revolves around a mechanic called gem collection, which masterfully blends gameplay and story. Using *get_random_position_in_zone* for variable placement and RayCast2D to guarantee gems land at exact heights on the *ground (position.y = collision_point.y - ($CollisionShape2D.shape.radius * 0.15))*, the GemSpawner script instantiates five gems from a *PackedScene* within predetermined *gem_zones*. Gems are collected within a *50-pixel radius (COLLECT_DISTANCE)* and fall with a *GRAVITY of 300.0*, which adds realism. This causes the *gem_collected* signal to increase the *gem_count* in World. This is immediately reflected in the HUD *(gem_label.text = "Gems: %d/%d")*, and each gem is transformed into a luminous ray of hope by the gem_pulse.gdshader. This gameplay loop is both addictive and meaningful since it propels the story—gems are the Princess's heart—and promotes strategic exploration under time constraints. It is a key component of the game's genius because it incorporates visual feedback and ending conditions.

### 3: Time Management

Prince's Pursuit's masterful use of time management gives each moment a sense of suspense and strategic nuance. A 60-second countdown is implemented using the *TimerSetup* script *(total_time = 60.0)*, which further amplifies urgency by updating the HUD *(timer_label.text = "%02d:%02d")* with a color shift to red below 30 seconds *(add_theme_color_override("font_color", Color(1, 0, 0)))*. The *game_over* scene, which is controlled by the world's *check_end_condition*, initiates a disastrous ending if time runs out with fewer than four gems; the joyful ending is unlocked when 11,500 pixels are reached with five gems. Players must balance speed and exploration in this skillfully balanced risk-reward dynamic, which is created by the timer's interaction with gem collection and player location. Prince's Pursuit is elevated into a heart-pounding race against time by this feature, which is further reinforced by the *ParallaxBackground's* unrelenting scroll and the narrative stakes of proving loyalty. This makes Prince's Pursuit a genre-defining experience.

**4: Enemy Avoidance and Combat:**

Prince's Pursuit's enemy avoidance and battle display a level of complexity uncommon in independent games, fusing strategic depth with AI-driven obstacles. To create a dynamic battlefield, the EnemySpawner script spawns up to 12 wolves *(max_enemies = 12)* with randomized places *(find_valid_spawn_position),* making sure that the *min_player_distance is 400 pixels*, and using PhysicsPointQueryParameters2D to avoid overlaps. With chase = true on player detection *(_on_player_detection_body_entered)*, the Enemy script applies AI. It moves at 200 pixels per second (SPEED = 200) and has gravity for realism. By stomping enemies *(is_above and is_falling)*, players can use the *enemy_flash* to trigger *death()*. Visual feedback is provided by gdshader. Every encounter is a skill test because damage is reflected in the health meter (World script) *(take_damage(20))*. Prince's Pursuit is a gripping masterwork of action and strategy thanks to this mechanic, which is enhanced by movement accuracy and shader effects, and turns the forest into a living gauntlet.

## Creational Programming Patterns:

**Pattern 1: Factory Method (Gem Spawner):**

The GemSpawner script instantiates gems from a PackedScene (gem_scene) inside *gem_zones*, utilizing the Factory Method pattern with unmatched beauty. While RayCast2D guarantees exact ground alignment, the *spawn_random_gems* function places gems dynamically using *get_random_position_in_zone*. This is fundamental—gems power the story and gameplay loop—and taking it away would destroy the essence of the game. The design is a scalable method that future-proofs Prince's Pursuit for expansions because it encapsulates creation logic and allows for smooth changes to spawn zones or amounts.

**Pattern 2: Builder (Love Letter Panel Styling)**

The LoveLetterPanel script carefully sets *bg_color, corner_radius,* and *border_color* to suggest romance, using a Builder-like pattern to design the submit button's style *(create_heart_style, create_heart_style_hover)*. This adds depth to the story and is essential to the four-gem ending; without it, the emotional impact would be lessened. The pattern's modular approach to UI styling, which guarantees flexibility and reusability, is its technical strength. This design decision raises Prince's Pursuit's polish to AAA levels.

**Pattern 3: Singleton (World Node)**

The World node functions as a singleton, a central hub that controls signal connections *(gem_collected, game_over)* and game state *(gem_count, has_triggered_end)*. It is the backbone of the game, orchestrating crucial logic such as HUD updates and ending triggers (check_end_condition); its removal would result in systemic collapse. Prince's Pursuit's architectural genius is demonstrated by the pattern's centralized control, which reduces node coupling and permits smooth coordination throughout the game's intricate components.

**Pattern 4: Factory Method (Enemy Spawner)**

To spawn wolves from a PackedScene (enemy_scene), the EnemySpawner script uses the Factory Method pattern. AI-driven randomization (find_valid_spawn_position) guarantees a variety of encounters. The forest would feel lifeless without this essential mechanic, which balances difficulty with constraints like *min_player_distance = 400.0* and *max_enemies = 12*. The pattern's strong handling of spawn logic, which permits dynamic adversary distribution that adjusts to player progress—a defining feature of Prince's Pursuit's inventive design—is It's technical strength.

## Structural Programming Patterns

**Pattern 1: Facade (World Script)**

The World script streamlines interactions between the player, timer, gems, and endings, making it the epitome of a Facade. It ensures a fluid gameplay flow by managing intricate logic such as *check_end_condition* and signal connections (connect_to_gems). Without it, subsystems would disintegrate and the game would be broken. A key component of Prince's Pursuit's sturdy design, scalability is made possible by the pattern's unified interface, which abstracts complexity.

**Pattern 2: Composite (ParallaxBackground)**

The ParallaxBackground scrolls at 100 pixels per second using the Composite pattern to control ParallaxLayer and Sprite2D (Forest.png) as a single, integrated entity *(scroll_offset.x -= scrolling_speed * delta)*. The world would feel flat without this amazing sense of depth, which is essential for immersion. Prince's Pursuit is visually stunning because of the pattern's hierarchical structure, which makes it simple to add layers (like clouds) with different *motion_scale*.

## Behavioral Programming Patterns

**Pattern 1: Observer (Gem Collection)**
Gem's gem_collected signal, which alerts World to update *gem_count* and HUD *(_on_gem_collected)*, is an example of the Observer pattern. Removing this would disrupt the collection mechanism, which is the foundation of progression tracking. Decoupling gem objects from game state allows for scalable signal processing, which enhances Prince's Pursuit's dynamic gameplay. This is the pattern's technical strength.

**Pattern 2: Strategy (Enemy Behavior)**

The Strategy pattern is used by the Enemy script, which alternates between idle and chasing behaviors *(chase = true on detection)*. The AI-driven pursuit of wolves, who travel at 200 pixels per second, adds dynamic and difficult confrontations; eliminating this would make the game boring. The pattern's technical strength is in its adaptability to player actions, which sets Prince's Pursuit apart from other adversary designs.

## Shaders

**Shader 1: player-outline.gdshader:**

The outline of the player. Using edge detection, gdshader highlights the Prince against the forest by adding a glowing outline. This improves vision during fighting and keeps players on course; removing it would make games more difficult. Prince's Pursuit is a visual wonder thanks to the shader's real-time processing, which resolves visibility issues in a crowded setting.

**Shader 2: gem_pulse.gdshader**

Using time-based modulation, the gem_pulse.gdshader gives gems a pulsating glow to attract attention. Without it, diamonds would disappear into the backdrop, making this essential for directing players to collectibles. Prince's Pursuit's attention to detail is seen in the shader's effective effect, which improves usability without sacrificing efficiency

**Shader 3: enemy_flash.gdshader**

In order to confirm stomps, the enemy_flash.gdshader uses a brief color overlay to trigger a flash upon enemy hits. The removal of this feedback would decrease reactivity, which is crucial for combat clarity. Prince's Pursuit seems sleek and responsive thanks to the shader's event-driven execution, which blends in perfectly with gameplay.

**Shader 4: bg-tint.gdshader**

Using color modulation, the bg-tint.gdshader adds a faint, cursed tint to the background, adding to the spooky feel of the woodland. By bringing the images into line with the story, this enhances immersion; otherwise, the tone would be weakened. Prince's Pursuit is elevated to dramatic heights by the shader's atmospheric augmentation, a small yet powerful touch.

## Tools/Technologies

**Tool/Technology 1: CharacterBody2D**

The foundation of movement in Player is the CharacterBody2D node, which manages physics (*velocity.y += gravity * delta)* and *move_and_slide()*. This guarantees responsive platforming; if it were removed, controls would be broken. A key component of Prince's Pursuit's outstanding gameplay is its strong physics engine, which saves development time while producing AAA-caliber mobility.

**Tool/Technology 2: AnimationPlayer**

The AnimationIn Player and Endings, the player coordinates *animations (anim.play("run"), animation_player.play("happy_end")* to make sure actions correspond with visual feedback. Eliminating this would make the immersion less engaging. Prince's Pursuit is a visual beauty thanks to its timeline-based control, which is a technological merit.

## Alpha Reflection

From the very beginning, Prince's Pursuit was envisioned as more than just a 2D platformer—it was designed to be a story of love, courage, and legacy. During the Alpha stage, the groundwork was carefully laid. Replacing KinematicBody2D with CharacterBody2D was a transformative step that made player movement feel more natural and responsive, aligning better with the emotional urgency of Bhairava's quest. Movement speed was fine-tuned (from 300.0 to 400.0), and falling animations were improved by tying them to vertical velocity, making jumps and landings more believable.

The game's polish was further elevated by introducing a sleek main menu that allowed players to easily start or exit the game, implemented using *get_tree().change_scene_to_file() and get_tree().quit()*. The ParallaxBackground was added to introduce depth and motion, teaching valuable lessons about frame-rate independence and improving visual immersion.

Early on, some technical hurdles emerged, particularly with node pathways and signaling, but these were resolved, paving the way for smoother scene transitions and object interactions. Lastly, combat mechanics were reimagined by removing the "magic bolts" and focusing on a stomp-based attack that felt more grounded in the fairy-tale world, giving Bhairava a move set that was both symbolic and satisfying.

**Beta Reflection:**

The Beta phase marked a dramatic shift from potential to presence, transforming Prince's Pursuit into a more emotionally immersive and gameplay-rich experience. One of the most impactful changes was the introduction of AI-driven enemy spawning. Thanks to the new EnemySpawner system with find_valid_spawn_position() logic. This made encounters feel unpredictable and alive.

Gems—The gems are created, which is a crucial part of the game, and also using RayCast and gems spawning gems are placed at difficult, high spots, which makes it hard to reach

To raise the stakes, a 120-second timer was implemented, adding pressure and purpose to every step the prince takes. Paired with a new health bar system—where each wolf attack deducts 20 points—survival became a more urgent concern, creating a real sense of risk.

Narratively, multiple endings were introduced to reflect the player's performance. Achieving all five gems leads to a happy reunion with Princess Mitravinda. Collecting fewer than four results in heartbreak. But reaching four gems unlocks the standout innovation of the Beta: the LoveLetterPanel. Here, players must compose a heartfelt letter to the princess, without the ability to copy-paste, to ensure authenticity. It's a bold integration of storytelling and player intent, forcing the player to channel Bhairava's sincerity.

Game Over scenes were also added to provide closure if the player runs out of time or health. These transitions now feel cinematic, seamlessly moving between action and narrative with emotional weight.

## Final Reflection

By the final phase, Prince's Pursuit felt more complete, both technically and thematically. A key focus was fixing issues with the enemy spawner, which had previously caused wolves to embed into the terrain. Through improved collision detection and spawn-point validation, enemies now appear cleanly on level ground, enhancing immersion and playability.

Gem placement was also refined. Instead of predictable or easy-to-reach areas, gems were repositioned across more scattered and challenging zones. This restructured the pacing and encouraged players to explore further, adding replay value and narrative depth.

Beyond mechanics, the emotional core of the story was brought into sharper focus. The journey was no longer just about collecting items—it became about proving love through perseverance. A short story sequence was introduced to reinforce this theme, reminding players that Bhairava's path is driven by purpose, not just platforming.

The final scene transitions—from victory to heartbreak—were also stabilized. Earlier bugs with the ground and camera were resolved, ensuring that players experienced smooth conclusions based on their choices. Finally, background music was added to the plot sequence, aligning the emotional rhythm of the game with its visual and narrative beats.

Together, these additions created a more emotionally resonant and mechanically robust experience—one that truly captured the essence of a cursed forest, a relentless prince, and a love worth fighting for.

## Future Scope

Looking ahead, Prince's Pursuit still has room to grow into something even more magical. In the happy ending scene, I plan to add celebration effects using CPUParticles2D to make the victory feel more rewarding and visually immersive. The aim is not just to mark success but to celebrate love's triumph over adversity.

To deepen the cursed atmosphere, I want to integrate ambient whispers that subtly play in the background—soft, eerie sounds that hint at the darker truths of the forest. It will reinforce the idea that beauty can be deceptive, and not everything that looks peaceful is safe.

One of the most ambitious additions planned is the introduction of a rare orchid tree. This mystical tree will appear only once and offer the player a powerful choice—either instantly grant three gems or slow down the timer. It's meant to symbolize an unexpected blessing, a reminder that the universe often conspires to help those who are truly determined.

This final touch is personal. It's my way of expressing that no matter how cursed the path may seem, sincere intent and perseverance always find a way to bloom—even in the darkest forest.