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:**

Prince's Pursuit is an enthralling 2D platformer created in Godot, in which the Prince sets out on a passionate adventure through a cursed forest to show the Princess how much he loves her by gathering five gems in 120 seconds while avoiding obstacles and adversaries. A story-driven experience with smooth mechanics, branching endings (happy with five gems, sad with fewer than four, or a special love letter panel with four gems), and an eye-catching fairy-tale look combining pixel art and dynamic shaders are the main objectives. This project intends to push the limits of immersive gameplay by incorporating cutting-edge AI for plagiarism detection and adversary behavior, guaranteeing a well-executed standalone.Exe was tested for smooth operation.

The process starts with a solid project skeleton incorporating player movement, gem generation, enemy AI, and a timing system. It then uses Godot's iterative, modular design technique. Along with a health bar, concluding scenes, and a love letter panel with anti-plagiarism features, post-alpha AI-driven randomization was used for adversary and gem placement. Scalability and maintainability are guaranteed by creational, structural, and behavioral programming patterns (such as Factory Method, Observer, and Command). OpenGameArt.org assets improve the visual story, and shaders improve the visual experience. Frequent GitHub commits and thorough playtesting improve compatibility and balance, establishing this project as a masterwork of innovative game creation.

**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 120-second countdown is implemented using the *TimerSetup* script *(total_time = 120.0)*, 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 its 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: Decorator (Health Bar Styling)**

World's health bar skillfully uses the Decorator pattern, dynamically applying *green_style* or *red_style* (health ≤ 3) to the ProgressBar depending on *player.health*. The player's awareness depends on this visual feedback, and its removal would hide important information. Prince's Pursuit's user experience is improved by the pattern's non-invasive augmentation of the health bar, which permits style modifications without compromising essential functionality.

**Pattern 3: 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*.

**Pattern 4: Adapter (Timer Integration)**

The TimerSetup script serves as an adapter, connecting game logic *(game_over, update_timer_display)* with the Timer node's raw timing. It makes time management easy by formatting time *(%02d:%02d)* and triggering endings. This is essential because the countdown system wouldn't work without it. The technical strength of the pattern is in its ability to modify engine functionality to satisfy narrative-driven limitations, guaranteeing that Prince's Pursuit's time mechanic is both practical and captivating.

## 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: State (Player Animations)**

The Player script employs the State pattern to control physics-based animations *("run," "jump," "fall," and "dead") (velocity.y, is_on_floor())*. Without it, the Prince would seem static. This guarantees that movements and visuals mirror each other, which is essential for immersion. Clean state transitions, which improve animation logic and add to Prince's Pursuit's professional feel, are the pattern's technical strength.

**Pattern 3: 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.

**Pattern 4: Command (Love Letter Submission)**
The LoveLetterPanel uses AI-driven plagiarism detection (is_plagiarism_detected) to validate input and uses the Command pattern for the submit button (_on_submit_pressed). Because of this, the four-gem conclusion is a narrative highlight; if it were removed, a significant emotional beat would be lost. Encapsulated input handling, which ensures strong, cheat-proof interaction and is a smart touch in Prince's Pursuit's design, is the pattern's technical merit.

## 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

Prince's Pursuit has developed into a genre-defining platformer since the first checkpoint. CharacterBody2D replaced KinematicBody2D, simplifying physics and improving movement realism. Movement tuning (SPEED from 300.0 to 400.0) and animation fixes (velocity.y > 0 for fall) improved responsiveness, while the main menu (main.tscn) with Play and Quit buttons (get_tree().change_scene_to_file, get_tree().quit) provided expert polish. A strong foundation was laid by the ParallaxBackground implementation, which taught frame-rate independence. After early issues with node pathways were fixed, attention turned to AI-driven elements, doing away with magic bolts in favor of a more unified stomp mechanism that matched the game's fairy tale theme.

## Reflection in Beta

With revolutionary additions, Prince's Pursuit reached new heights after alpha. A dynamic battlefield was created via the Enemy scene's AI-driven spawning using EnemySpawner, which also introduced randomized wolf placement (find_valid_spawn_position). With the use of RayCast2D, gems now spawn at exact heights, increasing the depth of exploration. Endings and game_over scenes provided dramatic closure, while a health bar (World) and TimerSetup (cut to 120 seconds) raised the stakes. With its AI plagiarism detection (is_plagiarism_detected) and disabled copy-paste (_input), the LoveLetterPanel brought a secure and poignant storytelling innovation. Prince's Pursuit is now positioned as a perfect gem that is prepared for final polishing with shaders and weather effects thanks to lessons learned about signal efficiency (gem_collected) and node path fixes.

## Final Reflection

With the addition of four shaders (player-outline, gem_pulse, enemy_flash, and bg-tint), each of which contributes levels of visual brilliance and feedback, Prince's Pursuit has cemented its reputation as a masterpiece since the beta. Without deviating from its central idea, the project's objectives centered on providing a memorable story and sensory experience. The gameplay was improved by additions like shader effects and improved AI (such as opponent stomping mechanics), while balance was maintained via optimization and playtesting lessons. The plagiarism prevention feature of the love letter panel was strong, and cross-platform compatibility was verified by the.exe export.