One Frog Band

A Game by FUMS?

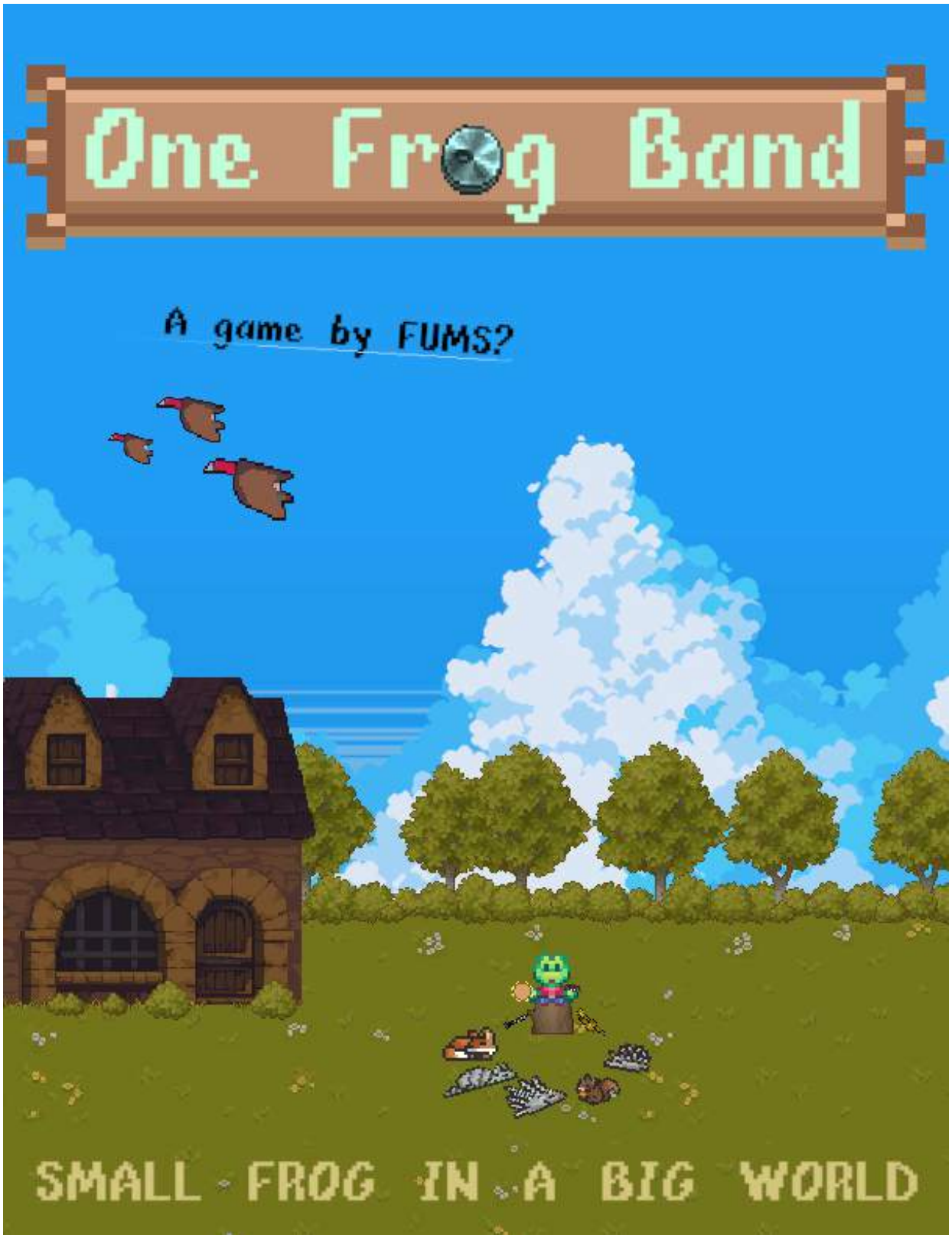


Table of Contents

0 Version Report

Version Number	**Date Reported**	**Person Reporting**	**Bugs Found**
0.8 (Demo Release)	05/09/23	Sammy	1. Transition screen occasionally gets stuck, requires game restart. a. Not consistently reproducible 2. Projectile sprites occasionally freeze in midair if player dies while grappled. a. Just a visual bug, doesn't affect gameplay b. Occasionally happens with enemies, but less often 3. Mushroom Forest has numerous invisible/misplaced colliders and death zones. 4. Trumpet Jumping on an enemy sometimes launches the player higher than other times. 5. Trumpet Jump launches the player higher if used in rapid succession.
Note: A full version history of bug fixes can be found in the GitLab repository, this list contains current known bugs in the release version!			

1 Designer Bios

1.1 FUMS?

We are FUMS?, a group of college students who made a game.

1.2 Sammy Levy

I tend to play a lot of roguelikes and JRPGs (some of my favorites being Hades, Enter the Gungeon, Persona 5, and the Xenoblade series). I mostly play on consoles, specifically Switch and Playstation. I want to design a game for someone who has a schedule similar to mine: most often able to play in short bursts, but enough variation/replayability that longer sessions are still fun. The game should have a low skill floor but a high skill ceiling. I think collaborating with team members who have different gaming experience / interests will help me come up with more ideas and develop a more unique game with a wider player base.

1.3 Max Cheng

I enjoy open-world exploration games like Breath of the Wild and Genshin Impact. I also am a fan of puzzle games and platformers including Celeste and Dead Cells. I play mainly on Nintendo Switch or mobile. I want to design a game for people who may not be able to sit down for long periods at a time to play a game, but are still looking for a fun, engaging, lighthearted experience that feels good to play at any level of skill or time commitment. I think having team members with different interests and approaches is vital to ensuring that our game is well-rounded and has appeal beyond what my own focus might be.

1.4 Bryanna Hernandez

I play mostly puzzle games, open-world adventure games, and narrative choice games. I play mostly on console, but sometimes on PC as well. I'll be designing for a player with a similar lifestyle to me: someone who is old enough to handle more mature or complicated mechanics like a young adult/college student and has a bit of free time to dedicate towards mastering the game. I think having multiple game interests as a team means that we'll be satisfied with focusing on different parts of the game, so the game is well-rounded and satisfies more people.

1.5 Nick Novak

I like Minecraft and its sandbox/software aspects. Other than that, I like action-adventure games (Tomb Raider, Uncharted), as well as older point-and-click adventure games (Myst, Dagger of Amon Ra). I play games exclusively on PC. I want the audience of the game to be wanting to find a good game, and otherwise want to find a good experience. I enjoy making specific parts of games, so collaboration would help with building out the other parts of the games.

2 Game Play Overview

2.1 Concept

Our game is a 2D side-scrolling momentum based platformer. The main character is a small frog named Claude, who lives in a small town in a fantasy world. He spends his days performing as a one-man band for the townspeople and anyone who will listen. One day, an evil rat wizard near the town gets fed up with the joy and the noise, and scatters Claude's collection of instruments across the far corners of their world. The mayor manages to save only his trumpet, and Claude

embarks on his mission of traveling the world to recover his instruments and his music. Accompanying him on this journey is his pet and best friend, a tarantula named Sir Jacques. Sir Jacques does not have significant individual motivations for coming along beyond not wanting his friend to be sad, and excitement at the prospect of adventuring around the world in service of their goal.

2.2 Player Objective and Obstacles

The superobjective of the player is to recover all of the frog's instruments and bring happiness back to his home in the Grenouille Village, and in the end to use all of his instruments to defeat the evil rat wizard once and for all. The main challenge lies in overcoming new environments and the inherent challenge of exploration, which will be communicated in the design of the levels themselves and the complexity of the platforming challenges. The win state of each level will be to recover the instrument at the end. Making contact with enemies, projectiles, or static obstacles (such as spikes) will kill the player, but in doing so will only respawn them at the most recent checkpoint, such that there is no true "lose" condition.

2.3 Player Activities

The core platforming mechanics will be a simple jump, as well as the ability to throw Sir Jacques the spider out in front of the player and swing from his spiderweb as a grappling hook. Enemies can be killed by jumping on their heads; there will not be complex combat beyond this. Our primary goal is to design mechanics and levels that make the flow of gameplay feel smooth, dynamic, and intuitive.

At the end of each level, Claude will recover a new instrument, unlocking a new movement mechanic that will become the main focus of the following level. Each instrument will have its own unique mechanic, building upon and synergizing with previous mechanics as the game progresses and more instruments are unlocked. The full game will have 7 main levels excluding the prologue, each focused on a different instrument: the trumpet, tambourine, clarinet, cymbals, violin, guitar, and keyboard. The visual theming of each level will be distinct from the others, and the level design will be closely tied to the use of its respective instrument mechanic.

The introduction and prologue will be a combination of tutorial for the basic mechanics (jump and grapple) and the basis of the plot, showing where Claude’s performance at the county fair stage is cruelly interrupted by the rat wizard's wrath. Claude then navigates to the mayor's podium, where he is presented with the saved trumpet. The following level, "The Brasslands," is themed around a grassland and will explore the trumpet's double jump functionality, unlocking the tambourine at the end. "Rolling Caverns" is themed around being stuck in a deep cave, and is focused around using the tambourine to pin enemies in place and grapple off of them. The clarinet level, "Reed Swamp," is themed around a marshland, with plenty of water pools for the

player to make use of the clarinet dash into water functionality. “Mushroom Forest” is set in a dark woodland area, where the player must use the newly introduced cymbals to bounce off of mushroom spore projectiles.

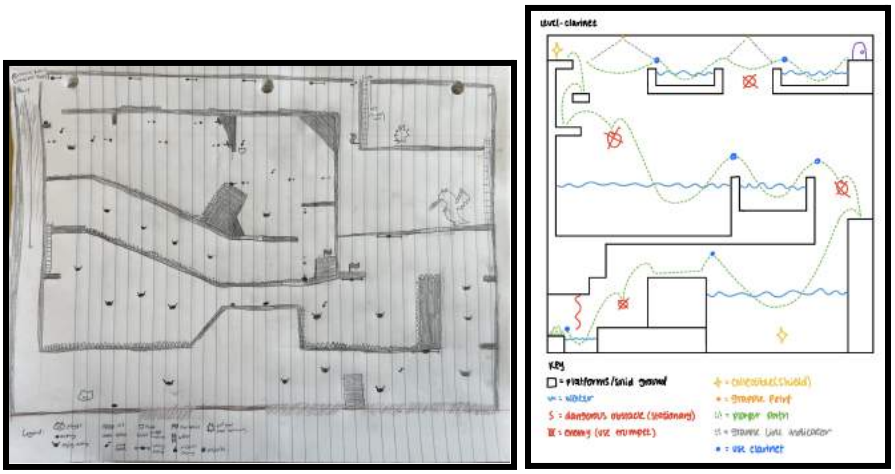


Figure 2.1: Early level designs for Rolling Caverns and Reed Swamp

Beyond the demo, there will be 7 distinct instruments across the same number of levels, including the four detailed above (the tutorial will not count as a full stage). At the end of the last level, there will be a final boss (the Rat Wizard himself). Once the wizard is defeated, Claude will finally return home and perform at the county fair, bringing joy back to the Grenouille Village.

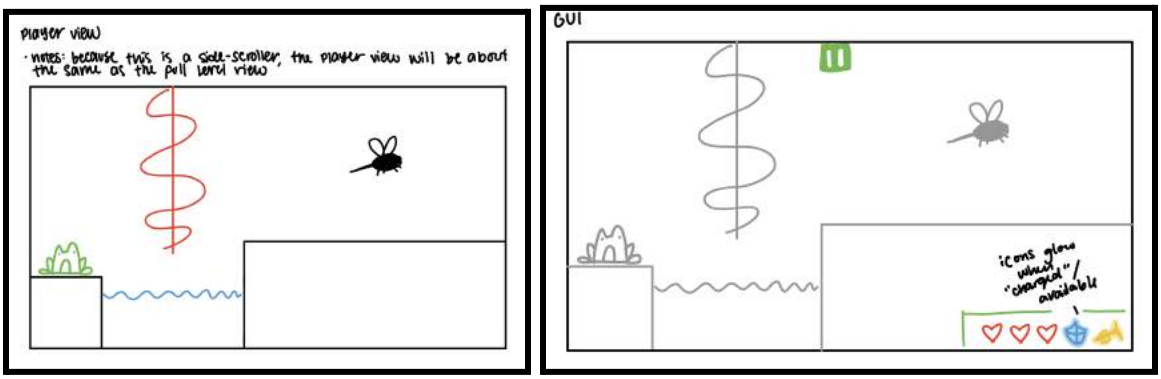


Figure 2.2: Early viewport and UI sketches

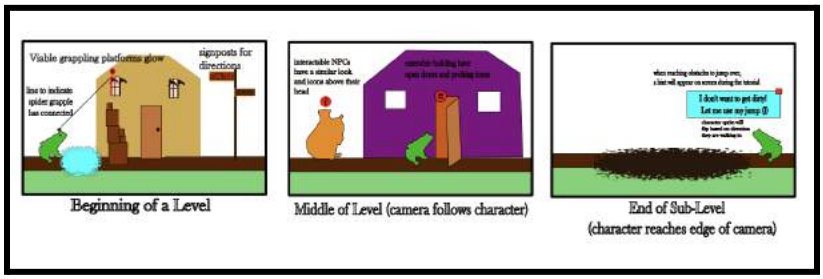


Figure 2.3: Early level progression sketches

3 Game Art Asset Samples

3.1 Environment: Architecture

Within our game, the main character is on a rescue mission to travel the world and recover his stolen instruments. These instruments are located in levels with highly different environments. The core mechanic of the game is fluid movement and platforming to recover the instruments from these diverse habitats. Therefore, the assets needed to distinguish each level appropriately vary widely. The narrative purpose of the tutorial level, located in Grenouille Village, is to emotionally orient the player to Claude the Frog and Sir Jacques, therefore the inclusion of furniture and architectural buildings is necessary. The art in this level should have an emphasis on a storybook style to convey an idyllic tone of peacefulness.

Figure 1: Grenouille houses and Town Hall

Figure 2: Furniture found in Claude's home.

Figure 3: Objects placed throughout the village

3.2 Environment: Terrain and Landscape

Outside of Grenouille Village, almost no man-made objects will appear. Instead, most obstacles should blend in with an otherwise natural habitat, intensifying the idea that Claude is an invader in a foreign environment that he must learn to navigate. The four environments highlighted are a grassy plain, a mushroom forest, caves, and a swamp. Additionally, our game has a forgiving loss condition; health cannot be regained, but a loss places the character at a checkpoint. Items within a similar level have a similar style and color palette, and even items that pose potential dangers, shouldn't have a particular ominous feeling.

Figure 3.4: Checkpoints, with both active and inactive sprites.
On the left is the checkpoint used in the Rolling Caverns.
On the right is the checkpoint used in the Reed Swamp.

Figure 3.5: Projectile spores found in the Mushroom Forest.

Figure 3.6: Dangerous plants that can cause death.
On the left is a flower found in the Brasslands.
On the right is a plant found in the Reed Swamp.

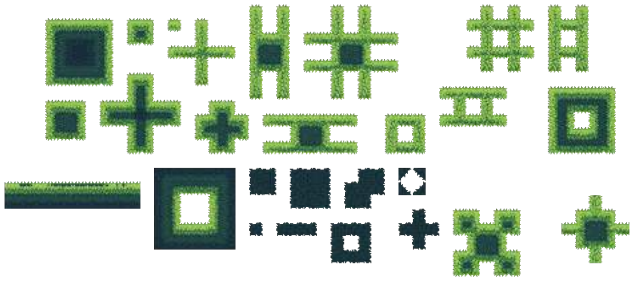


Figure 3.7: Tileset used to construct the Reed Swamp.

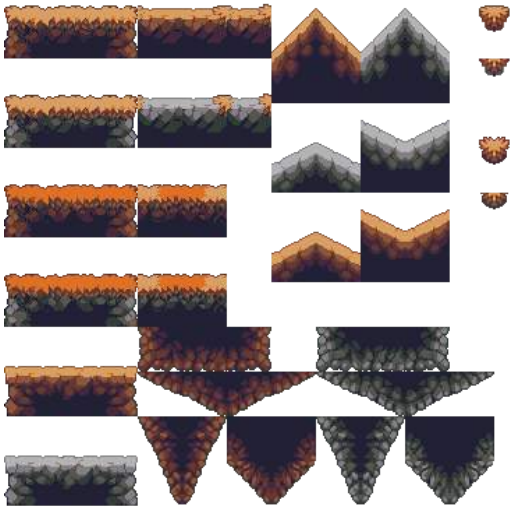


Figure 3.8: Tileset used to construct the Mushroom Forest.

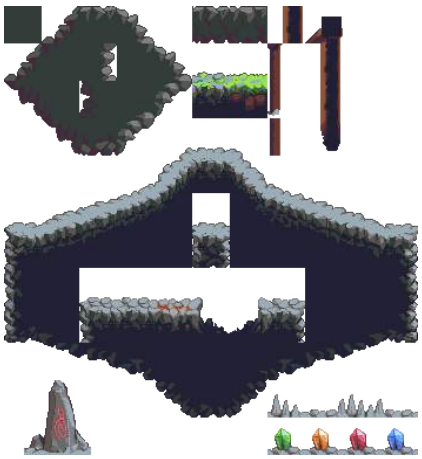


Figure 3.9: Tileset used to construct the Rolling Caverns.

3.3 Characters: Friends

All characters within our game are anthropomorphic to suggest a fantasy world without humans. The player character, Claude the Frog, seen in the third person, will always be accompanied by his tarantula, Sir Jacques. Claude's main actions are running, jumping, and using Sir Jacques's spider webs to grapple. Inside Grenouille Village, all other animals are friends. Sprites are

minimalist and 2D. These sprites convey a friendly, storybook feeling that would appeal to a child. Sprites are generic to represent adult and children villagers of all genders.

Figure 3.10: Squirrel sprite sheet.

Figure 3.11: Fox sprite sheet.

Figure 3.12: Sir Jacques

Figure 3.13: Claude sprite sheet.

3.4 Characters: Enemies

Antagonistic characters will not have weapons or projectiles. Instead, their main danger will be proximity to the character, therefore their unique shapes and methods of moving (flying, climbing, crawling) and speeds will be the primary difference between enemies. Enemies should resemble real animals without evoking feelings of “evil.”

Figure 3.14: Bat sprite sheet, found in the Rolling Caverns.

Figure 3.15: Slug sprite sheet, found in the Rolling Caverns.

Figure 3.16: Vulture sprite sheet, found in the Brasslands.

Figure 3.17: Dragonfly sprite sheet, found in the Reed Swamp.

3.5 Props

As part of the collection of Claude’s instruments, each level will expand gameplay. At the tutorial level, Claude will be limited to running, jumping, and grappling using his pet spider. In the first level, Claude will be able to double jump with his trumpet. In the second level, Claude will gain the ability to create additional grapple points that Sir Jacques can attach to using the tambourine. In the third level, Claude will gain the ability to use his clarinet to dive. In the final level, Claude will gain the ability to pogo-jump over projectiles using the cymbals. Prop sprites

are used both in-game and for the GUI to demonstrate availability to the player. The instruments look reasonably realistic.

Figure 3.18: Trumpet.

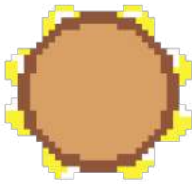


Figure 3.19: Tambourine.

Figure 3.20: Clarinet.

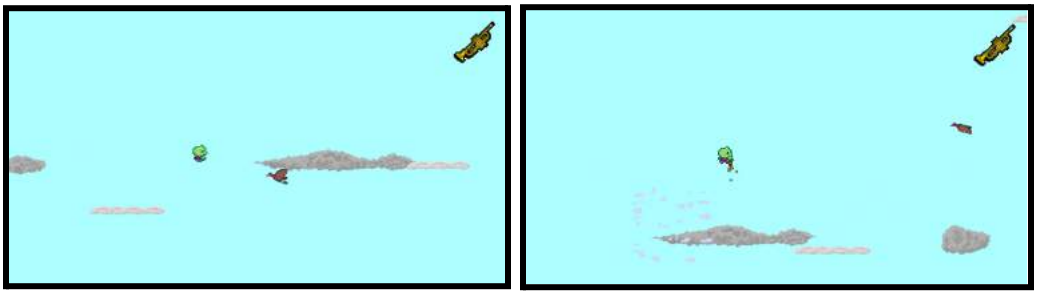
Figure 3.21: Cymbals.

4 Sample Gameplay Description

The player starts out in the game in Grenouille Village with 3 basic actions available to them: running, grappling, and jumping, demonstrated in the picture below. Tutorial text is provided to help the player learn these basic actions. The player starts out in Claude's house, and must collect Sir Jacques in order to leave his house. Then, the player runs through the village, jumping over obstacles and grappling to poles in order to make his way to Town Hall for his performance.

After Grenouille Village, the player moves onto the Brasslands and acquires the Trumpet Jump mechanic. Trumpet Jump has two functions. Firstly, it allows the player to double jump, as demonstrated below. With this function, the player can only jump once in the air, and when they can no longer jump, the trumpet on the top right corner of the screen will go black. This function is accompanied by a trumpet-blowing animation and sound. The player can use this function to jump onto platforms and avoid the poisonous flowers.

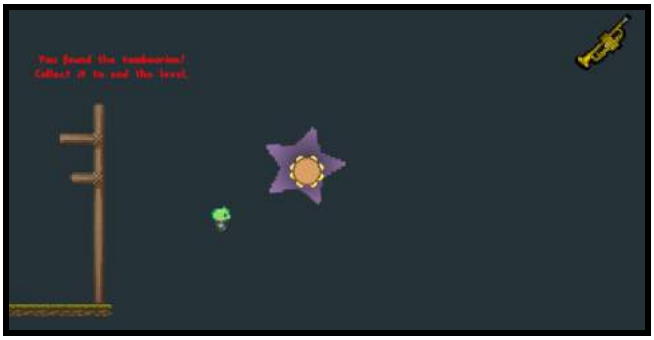
The other function of the Trumpet Jump occurs when the player jumps directly over an enemy. As seen in the two pictures below, the player performs the Trumpet Jump directly above an enemy, giving them an extra boost. This function allows the player to use an extra double jump in the air, as shown by the visible trumpet in the top right corner, and it can be chained as long as there are other enemies in the player's path. The player must use this jump to navigate floating cloud platforms over a dirty lake and reach the end of the level.



Both of these Trumpet Jump functions should be used to navigate difficult environmental stretches, and the player will retain these abilities for every level following the Brasslands.

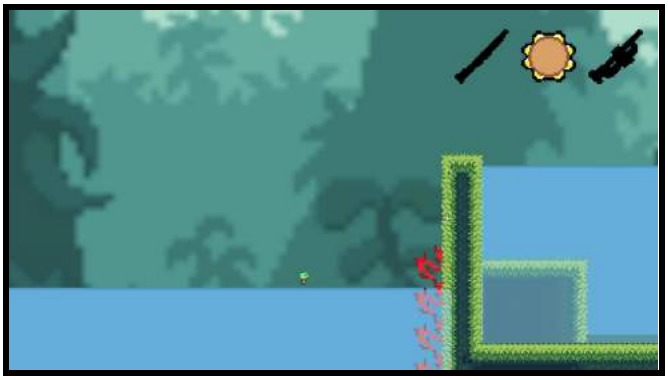
Additionally, as seen below, checkpoints are intermittently placed in each level, with an active and inactive sprite, to save the player's progress after defeating sections of the level.

As demonstrated in the next screenshot, when the player reaches the end of the level, they will be greeted with an instrument floating inside of a rotating star. In order to finish the level, the player must collect the instrument by jumping into the star. When this is successfully accomplished, the player will be able to progress.



When the player finishes the Brasslands, they will move on to the Rolling Caverns which gives the player the Tambourine. When the player throws the Tambourine, they will be able to pin enemies and projectiles in place, creating a new surface that the player can grapple to. This process is demonstrated in the screenshots below. The GUI will also update to reflect the tambourine's availability. In the Rolling Caverns, there are enemies on walls as well as enemies flying in the air that are suitable to be trapped by the Tambourine, and the player can chain these grappling points with preplaced grapple platforms. Additionally, crystal projectiles are being spawned from wall octopi that the player must strategically pin and grapple from. In order to succeed, the player must avoid both spikes on the ground and colliding with enemies or projectiles.

After the Rolling Caverns, the player will reach the Reed Swamp, gaining the ability to Clarinet Dive. When the player activates this ability, they will accelerate in a downward direction. If the player enters the water, they will be able to bounce back in an upward direction with extra speed, giving them the height necessary to bypass certain obstacles. Once again, the GUI updates to reflect the clarinet's usage. The Reed Swamp is filled with ponds of water that the player can Clarinet Dive through, and then use its height and momentum to grapple floating enemies or jump over them. Additionally, the player must avoid dangerous red plants that can cause death.



Finally, the player will reach the last level, the Mushroom Forest, which is accompanied by the introduction of the Cymbal Crash mechanic. If the player uses the cymbals to jump directly over a spore projectile, they will be able to cross vast gaps that cannot be traversed using any other instrument. Once again, the GUI updates to reflect the cymbal's availability. This is the only mechanic in which the player is encouraged to spam it. In the Mushroom Forest, there is an additional environmental mushroom that boosts the player upwards in a specified direction. This level has the least direction, and requires the player to carefully navigate the forest since falling can lead to falling off the level and dying or accidentally activating previous checkpoints.

5 Game Mechanics and Technical Design

5.0 Core Mechanic: Search and Rescue

The core mechanic of *One Frog Band* is Claude's search-and-rescue quest to recover his stolen instruments and bring joy back to his home in the Grenouille Village. By using the various movement mechanics, Claude will traverse a variety of levels through different environments.

5.1 Mobility Mechanic: Jump

The jump will be a primary mobility mechanic for the player; a necessary tool for navigating the platforming challenges of the various levels. The player can also jump on the heads of enemies to defeat them.

5.1.1 Input

By pressing the jump button (spacebar / bottom button of controller), the player will have vertical force added to their current momentum to send them into the air, roughly 1.5x their height. While in the air, the player can move left and right to adjust their trajectory. If the button is held longer, the player will jump higher.

5.1.2 Effects

At the start of the jump, Claude will squat and launch himself into the air. He will stay in this jump pose until they hit the ground, where he will return to the animation state needed to carry their momentum forwards. A sound effect will play at the start of the jump, and a different sound will play when Claude returns to the ground. If landing on an enemy, Claude will bounce off of it and be sent back into the air. Enemies will be defeated with a magical “poof” animation, and a sound effect will play.

5.1.3 Properties and Methods

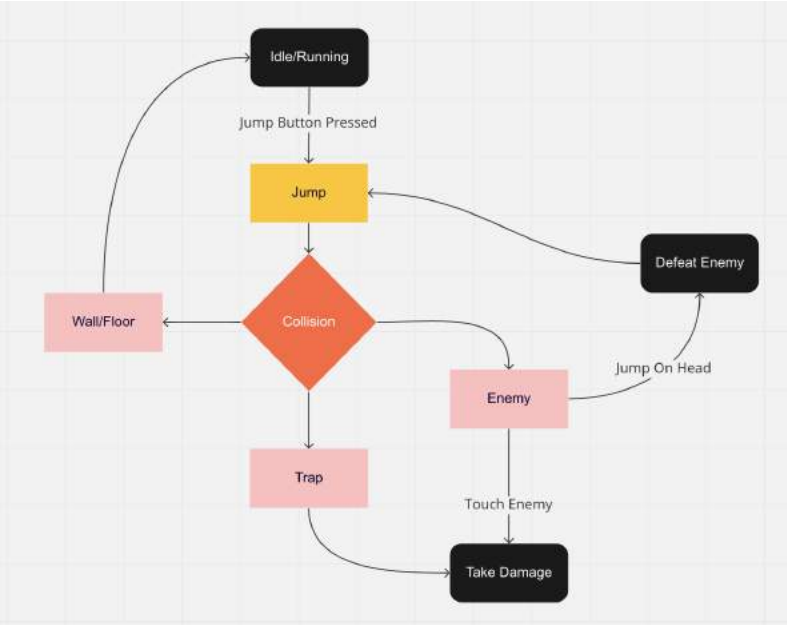
Properties

- *LastOnGroundTime* is a float that is used to check if the player is in Coyote Time, i.e. if they have fallen off a platform recently enough that it “feels” like the player should still be able to jump.
- *IsJumping* is a boolean that, if true, prevents the player from jumping a second time.
- As part of a player movement data object, there are properties for jump height, fall speed, and in-air acceleration that can all be easily edited by the designer in the Unity inspector.

Methods

- *IsGrounded()* determines whether or not the player is currently on the ground. If the player is not on the ground, they will not be able to jump, but may have other actions available to them.
- *OnCollisionEnter()* will check if the player has collided with an enemy, wall, or trap. When colliding with a wall or the floor, the player’s horizontal or vertical momentum respectively will be impacted. When colliding with an enemy, check to see who takes damage, and add vertical force to the player if they land on top of the enemy. If the player collides with a trap, take damage.
- *Jump()* will first check if the player is *grounded*, and if so will add the vertical force to the player for the jump.

5.1.4 Game States of Mechanic



5.2 Mobility Mechanic: Spider Grapple

Claude can use his pet spider, Sir Jacques, as a grappling hook to swing across chasms and other perilous sections of the world.

5.2.1 Input

Pressing and holding the grapple button (“L” key, controller triggers) will throw Sir Jacques to a grapple point, if there is one in range. When the button is pressed on the ground, the player will hop a small distance in the air and will be carried into the swing. If the player is in the air, their horizontal movement will be carried into the swing. The player can move left and right to affect their horizontal momentum while attached to the web as well. When the button is released, the grapple will be detached, and the player will be launched from the swing at their current speed and angle. Sir Jacques will return to the player when the button is released as well.

5.2.2 Effects

When the grapple button is pressed, Claude will throw Sir Jacques at the grapple surface, and a web between them will be created. Claude will have a swinging sprite that they will be in while in the air, but Sir Jacques will be idle on the grapple surface. A spider sound will play when Sir Jacques is thrown, a rope tension sound will play when swinging, and a snapping sound will play

when the player detaches from the web. When detaching, the web will snap, and Sir Jacques will jump off of the grapple surface to return to the player.

5.2.3 Properties and Methods

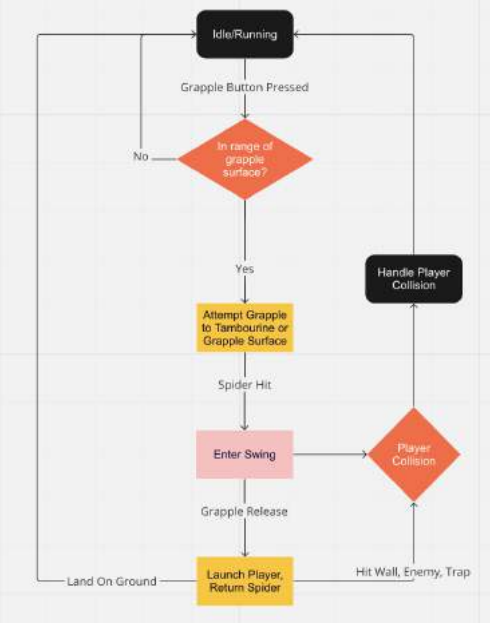
Properties

- *isGrappling* is a boolean that determines whether or not the player is currently grappling.
- *LineRenderer, SpringJoint,* and *DistanceJoint* are Unity components that will be enabled when the player grapples.
- *inDistanceRange* is a boolean that determines whether the player is in range of a grappleable object.
- *grapplePoint* is a Vector2 that describes the location of the point where the player will grapple to.

Methods

- *AttemptGrapple()* will be called when the grapple button is pressed. This will check if *isGrappling* is false and will enable the *LineRenderer, SpringJoint,* and *DistanceJoint* if so by calling the *GrappleToSurface()* or *GrappleToTambourine()* functions.
- *LetGoOfGrapple()* detaches the player from the web and releases Sir Jacques from the grapple surface, returning him to the player.
- *GrappleToSurface()* and *GrappleToTambourine()* attach the player to the nearest surface or tambourine, if they are in range.

5.2.4 Game States of Mechanic



5.3 Mobility/Weapons Mechanic: Trumpet Jump

The Trumpet allows the player to double jump. If used while directly above an enemy, the enemy will be defeated, and the jump charge will be restored, allowing the player to chain trumpet jumps across enemies.

5.3.1 Input

The jump button (spacebar, bottom button of controller) will also allow the player to perform the Trumpet Jump. If the player is already in the air and can no longer perform a regular jump, then pressing the jump button will cause the player to perform a Trumpet Jump instead. Similar to the normal jump, the player can jump higher if they hold the button down longer, and they can maneuver left and right in the air. If the player bounces on an enemy with the Trumpet Jump, the use will be restored, which will allow the player to perform another Trumpet Jump before landing. This can be repeated as long as there are enemies for the player to hit.

5.3.2 Effects

To perform the Trumpet Jump, Claude will reach behind themself to grab the trumpet, and then point it downwards to play the note. This part of the animation will be in time with a note sound effect. If there is no enemy below the player, then the animation continues as a normal jump. If there is an enemy below the player, then the enemy will be defeated with a magical “poof” and an accompanying sound effect. Claude will continue their forward movement with the trumpet in hand.

5.3.3 Properties and Methods

Properties

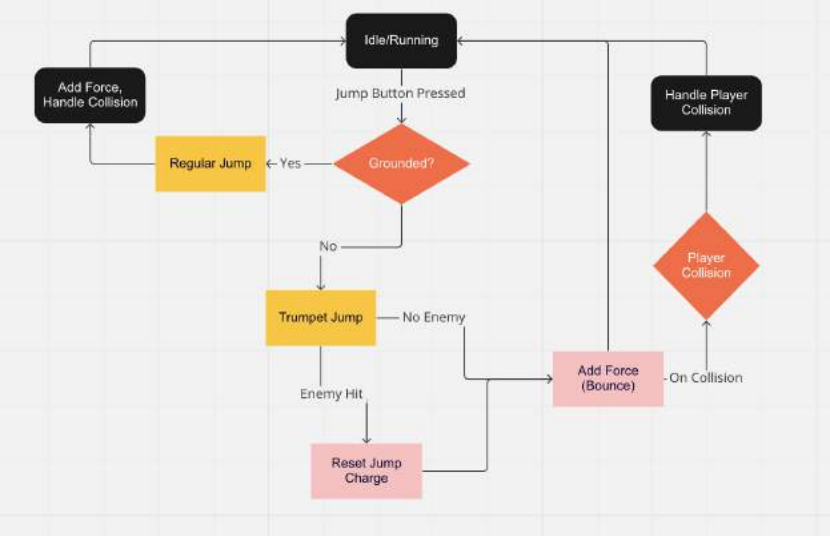
- *trumpets* is an integer that determines how many jumps the player has remaining. If the player has only one jump remaining when they press the jump button, it will be a Trumpet Jump. The number will be decreased whenever the player jumps, and will be increased if the player Trumpet Jumps on an enemy.
- *inRange* is a boolean that checks if the player is in range of an enemy to perform a Trumpet Jump.

Methods

- *OnCollisionEnter()* is used in a similar manner to the jump: the player may collide with walls, traps, and enemies, and all of those interactions will be handled as described in section 5.1.3.
- *OnTriggerEnter()* checks whether or not the trumpet blast collided with the enemy, and bounces the player back into the air if so.

- *Jump()* will first check if the player is currently not grounded. If that is the case, then the player will be able to perform a Trumpet Jump instead, which adds the same force as a jump and resets the jump charge if an enemy is hit.
- *ActivateTrumpetSprite()* plays the trumpet animation, which makes the trumpet sprite appear, plays the poof animation, and plays the trumpet sound.

5.3.4 Game States of Mechanic



5.4 Mobility/Weapons Mechanic: Tambourine Throw

The tambourine is a projectile that the player can throw in an arc. If it collides with an enemy, it will pin them in place, allowing the player to throw Sir Jacques to the enemy's location and grapple from them.

5.4.1 Input

Pressing the Tambourine Throw button ("K" key, controller west button) will launch a spinning tambourine from the player's current position forwards in an arc. If the tambourine gets close enough to an enemy, it will quickly snap to the enemy's location and pin them in place. For the next three seconds, the player If the tambourine collides with a wall, takes too long to hit a target, or the player waits too long to grapple to it, it will fly back to the player very quickly.

5.4.2 Effects

When the throw button is pressed, a "whoosh" sound effect will be played as the spinning tambourine begins to fly to the enemy. If the tambourine pins an enemy, it will play a short animation, where it stops spinning, and briefly shrinks and grows in size to indicate that it is

locked in place. This will play in time with a tambourine sound effect. Any enemy animation and movement will be paused while they are pinned.

5.4.3 Properties and Methods

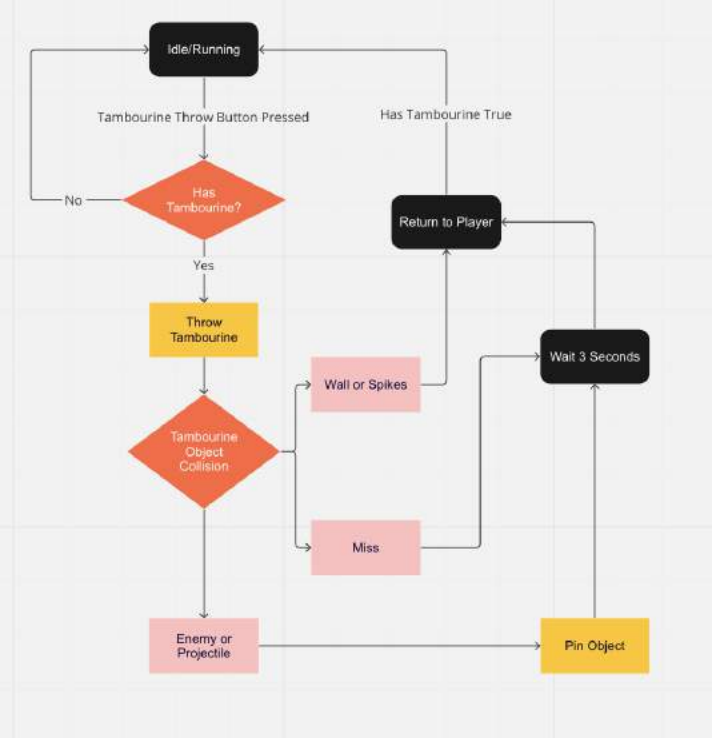
Properties

- *hasTambourine* is a boolean that checks whether or not the player current has a tambourine that they can throw
- *isPinned* is a boolean that checks if a tambourine is currently pinned to an enemy.
- *unlockedTambourine* is a boolean that prevents the Tambourine from being used unless the player has already unlocked it.
- *timeToLerp* is a float that determines how long the tambourine will take to reach the enemy once it is in range (very small amount of time).
- *returnToPlayer* is a boolean that defines whether or not the tambourine should return to the player instead of flying away.

Methods

- *ThrowTambourine()* launches a tambourine from the player's location away from them in an arc.
- *SetHasTambourine()* sets whether or not the player currently has a tambourine, and updates the UI as needed.
- *CheckToDestroy()* checks to see whether or not the tambourine has existed for more than 3 seconds and destroys it if so, and if the player is not currently grappling to it.
- *DestroySelf()* destroys the tambourine object, and unpins any enemy or projectile it may be pinned to, while also calling any necessary methods on those objects in the unpinning process.
- *OnTriggerEnter2D()* and *OnCollisionEnter2D()* check whether the tambourine has collided with something it can pin to or a wall.
- *Pin()* freezes the tambourine in place on an enemy, and calls necessary methods on the pinned object to freeze them in place.

5.4.4 Game States of Mechanic



5.5 Mobility/Weapons Mechanic: Clarinet Dive

The clarinet allows the player to perform an angled dive while in the air. If they collide with water while diving, they will dive under and be launched back up at a higher velocity.

5.5.1 Input

Pressing the Clarinet Dive button (“J” key, controller west button) sends the player into a 45° downwards dive through the air, adding a short burst of horizontal and downward force. If the player collides with water during their dash, their moment will be carried through the pond, and it will be reflected back upwards from the bottom, launching them back to reach locations further and higher than they normally can. Dashing through water also resets the trumpet jump, allowing the player to chain this with other movement mechanics.

5.5.2 Effects

When the button is pressed, a clarinet arpeggio plays, and a small clarinet appears for the duration of the dive. When the player hits the water, a splash sound plays, and a similar sound plays when they exit. Underwater, all background sound will become muffled, evoking the feeling of being underwater.

5.5.3 Properties and Methods

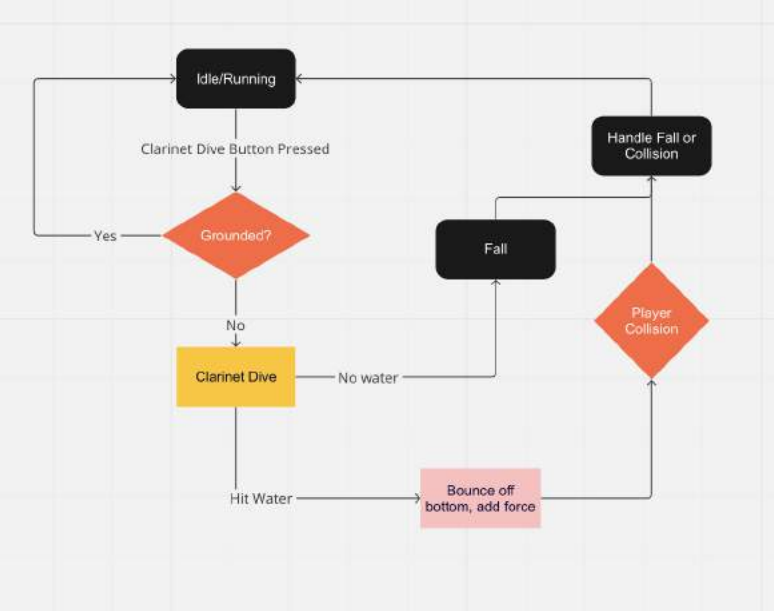
Properties

- *unlockedClarinet* is a boolean that prevents the Clarinet from being used unless the player has already unlocked it.
- *isDash* is a boolean that determines whether or not the player is currently dashing.
- *dashTime* is a float that defines how long a dash will be.
- *reflectForce* is a float that multiplies the player's entrance velocity to a body of water to bounce them out of it.

Methods

- *Dive()* applies the downwards force to the player and temporarily disables input, locking the player into the dash.
- *Water()* saves the player's entry velocity when they collide with water, which is then applied as a reflection to send the player hurtling out of the water with the *Bounce()* function.
- *Bounce()* applies the reflected entry force to the player when they reach the bottom of a body of water if they dashed into it.

5.5.4 Game States of Mechanic



5.6 Mobility/Weapons Mechanic: Cymbal Crash

By using the cymbals while in the air above an enemy projectile, the player can repeatedly bounce on it to travel greater distances.

5.6.1 Input

Pressing the Cymbal Crash button (“I” key, controller north button) while above an enemy projectile will add some vertical force to the player, bouncing them off the projectile slightly back into the air. Usage of the Cymbal is unlimited, so the player can repeatedly bounce off projectiles to travel great horizontal or vertical distances.

5.6.2 Effects

When the Cymbal Crash is triggered, Claude will crash two cymbals together below him, playing a crash sound at the same time as the animation. If this hits a projectile, the crash sound will be louder than if it misses (in which case it will be muffled), and Claude will be bounced back into the air.

5.6.3 Properties and Methods

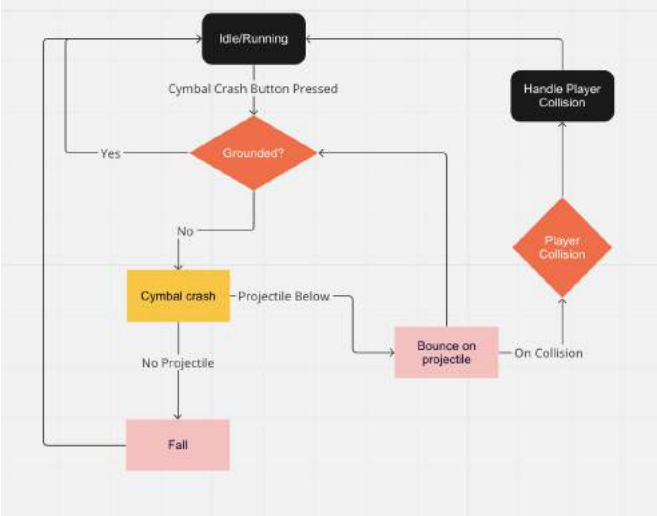
Properties

- *unlockedCymbal* is a boolean that prevents the Cymbal from being used unless the player has already unlocked it.
- *cymbalActiveTime* is a float that determines how long the cymbal is active for, and how long the player needs to wait before activating it again.
- *cymbalBounceForce* is a float that defines how much force is added (how high the player bounces) after using the Cymbal on a projectile.
- *cymbalHitboxRange* is a float that describes how far above a projectile the player is allowed to be while activating the Cymbal.

Methods

- *CymbalCrash()* is the function called when the player presses the Cymbal Crash button. It checks if the player is within *cymbalHitboxRange*, and if so adds *cymbalBounceForce* to the player’s upwards velocity.
- *HasProjectileInRange()* actually performs the check to see if the player has a projectile within the defined *cymbalHitboxRange* beneath them.

5.6.4 Game States of Mechanic



6 Player Profile

6.1 Bartle Types

Our game is made to appeal to players who are both explorer and achiever types.

Under Richard Bartle's definition, explorer types are players who want to experiment and master mechanics. This is a critical aspect of our game, as we built mechanics such as the grappling spider and the clarinet dive to start out simple on their own, but demand more mastery and creativity from the player as the levels progress and mechanics start to build off of each other. In order to successfully complete a level, it may be necessary that the player has the time to dedicate per session and really understand the controls. Additionally, our levels are based in an expansive environment, where the route to the end isn't always clear and requires some curiosity and thorough discovery.

As for the achiever type, Bartle defines the player as someone who is open to challenges and wants to brag. Due to the difficult nature of our game, which contrasts its cute fantasy setting and cottage-core aesthetic graphics, *One Frog Band* brings out the competitive nature of its players who want to be the first to master a section, or want to complete a level without any deaths, or want to finish a level under a certain time limit. While our game is not explicitly designed with these competitions in mind, as there is no timer or death counter, we find that there is a great sense of achievement that players feel when they manage to complete a level under certain conditions.

6.2 Vandenberghe Domains

Under the five Vandenberghe domains of play, *One Frog Band* is classified as a High Novelty, High Challenge, Low Stimulation, Low Harmony, and High Threat game.

While our game's levels are pre-built and do not change, the novelty comes from the addition of new mechanics at every level, and the new environments presented, keeping the game fresh even as the core search mechanic remains the same. Additionally, our game requires a methodological approach, as success cannot be found through random button-mashing, creating a high challenge. *One Frog Band* has a low-stimulation domain because its solo play encourages the player to sit down and carefully work out a solution. For a similar reason, *One Frog Band* is also considered to have low harmony, since sharing and cooperation are not part of the game. Lastly, our game is classified as high threat, not because it includes graphic violence, but rather because its difficulty often requires fast-based gameplay where a single mistake can result in death and the erasure of past work.

7 Similar Games

Our game draws inspiration from a wide variety of games from different genres and generations. The core gameplay loop of platforming through difficult, bite-sized sections of each level is inspired by *Celeste*, where the player is encouraged to persevere through a very quick respawn after death and frequent checkpoints. Similar to *Celeste*, each level of our game introduces a new movement mechanic that allows the player to interact with the world in a new way. However, where *Celeste* introduces these mechanics as world objects that the player can interact with using their existing movement techniques, *One Frog Band* implements these movement mechanics as tools for the player to use anytime with the press of a button. For example, *Celeste*'s dash mechanic can be used at any time, but level objects like the space jelly, bubbles, and moving platforms provide new interactions with it. In our game, once the clarinet is unlocked, it's up to the player to experiment and seek out opportunities to use this new action wherever they see fit to progress through the game.

Level design was largely inspired by *Celeste*, *Hollow Knight*, and the *Super Mario Bros.* series. The notoriously hard-but-fair difficulty of games like *Celeste*, *Hollow Knight* inspired the difficulty curve of our game, where responsive controls for interesting mechanics encourage the player to keep trying. This also allows us as developers to experiment with different level layouts that may be tough, knowing that the player will be able to get through it with a reasonable amount of perseverance. The design of all modern platformers can be traced back to *Mario* in

some way, and its 4-step level design philosophies served as a strong guiding principle in *One Frog Band*. The process is as follows: 1. Introduce the mechanic in a safe/simple environment, 2. Explore the mechanic further to give the player an opportunity to master it, 3. Introduce a new twist or wrinkle to the mechanic for the player to learn, 4. Give the player an opportunity to showcase their skills and look good while doing it. A further study of this level design process can be found in designer Mark Brown's videos here and here.

The inspiration for our instrument mechanics individually can be found in *Shovel Knight, Hollow Knight, Spider-Man (2018), Celeste*. The feel of *One Frog Band*'s grappling hook draws inspiration from the recent *Spider-Man* games, although notably grappling hooks are appearing noticeably more often in many modern games. By not having to aim the grappling hook, the player is able to keep their flow and momentum moving forwards while still feeling like they have control. The Trumpet Jump and Cymbal Crash abilities draw from pogo mechanics like *Shovel Knight*'s shovel drop and *Hollow Knight*'s nail bounce, allowing the player to bounce on projectiles and enemies to travel greater distances. The Clarinet Dive ability is inspired by horizontal dashes found in *Celeste* and *Hollow Knight*, which are used for traversal and sudden directional changes, providing the player with even more movement options for speedy traversal.

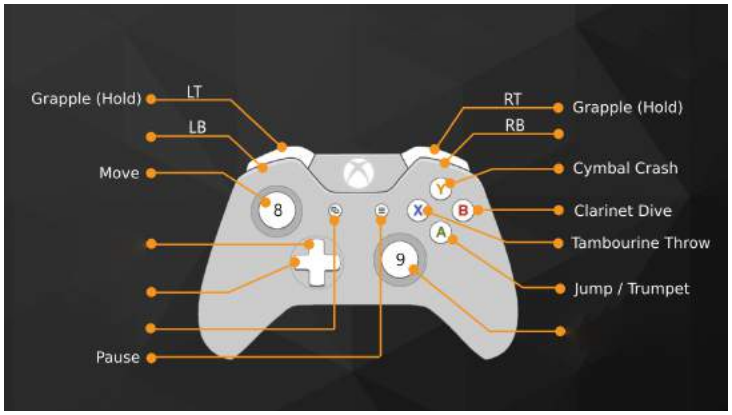
While not included in the demo, planned boss fights would have taken inspiration from *Celeste* and both the *Kirby* and *Mario* series. Rather than engaging in active combat with the boss, the player must dodge enemy attacks and navigate a much smaller arena using their movement skills in order to place themselves in a position to jump on an enemy's head, step on a switch, or counterattack in some other indirect way.

8 Controls

8.1 Keyboard Controls

This keyboard layout is fairly standard for movement, jumping, and pausing, which will be familiar to those who have played PC games before. The instrument controls are mapped in a similar pattern on the other side of the keyboard, where the right hand would sit naturally.

8.2 Gamepad Controls



Similarly, gamepad controls are fairly standard for moving, jumping, and pausing. The instrument controls naturally take up the rest of the face buttons, and assigning the grapple to the triggers evokes the feeling of holding onto the rope.

9 Technical Specifications

9.1 Platform

The primary platform that we are targeting is PC. As a 2D side-scrolling platformer game, it is critical that we are able to make use of a wide, high-resolution field of view and a system that provides access to precise, responsive inputs. Because our art style is pixel-based, it will scale well at any screen size or resolution, but our target resolution is 1920 x 1080 to accommodate most casual players' standard dimensions. Although we are not anticipating needing a high processor capacity, having access to these capabilities gives us, as developers, more flexibility and working room during the development process. PCs also have highly adaptable input systems. The keyboard allows for a wide range of possible inputs for any mechanics (present and future) we have or may want to add, while remaining comfortable and responsive to use. It is also possible to connect traditional game controllers to a PC and bind inputs there, allowing a player to choose whichever input system suits them best.

For a larger release, the game could also be ported to a console system such as the Nintendo Switch or PlayStation. This would open up wider, higher-resolution screens, and focus on the controller as an input system with the goal of increased player immersion. As stated, our game does not require heavy processing or particularly high-end graphics rendering, so consoles like these would be more than capable of handling it. Additionally, being available for console allows us to reach a wider range of players who otherwise might not have been interested or even exposed to a PC-only game.

9.2 Input System

The primary input system for our purposes will be a keyboard, but would be a controller for a larger release. On the keyboard, the standardized WASD movement and spacebar jump will cover primary movement mechanics, and other keys on the right side of the keyboard will cover the grapple and other instrument mechanics, such as “K” to throw a tambourine and “L” to grapple. The keyboard controls could also be re-bound if a player has a custom input that they would prefer. This could be extended to a controller, with a standardized movement scheme of the left stick to move and “A” button to jump, with other buttons and triggers handling secondary movement mechanics. On a controller, the grapple could feel more immersive, as a trigger would need to be held down, much like the grappling rope. Changing between these control schemes can be seamlessly handled by Unity’s input system.

9.3 Minimum Computer Specifications

Our game has fairly simple graphics, and does not require processor-heavy computations (such as shaders for lighting or real-time 3D rendering, or intense physics). However, a base system requirement for the game to run is a minimum of the OpenGL graphics specification 4.5. This can be met with the following specifications:

1. For integrated graphics:
    a. Any 6th generation or later Intel CPU
    b. Any AMD Ryzen CPU with Radeon Graphics
2. With a dedicated graphics card paired with a CPU of at least 1.6 GHz
    a. Any GPU at least or later in the Nvidia GeForce 400 series
    b. Any AMD GPU at the 5000 series or later

The game will only require a minimum 512 Mb of RAM with two cores, in large part due to the simplistic animation and sound that will be explored in the next two subsections. Because of the light-weight sprite art/animation and two simple audio channels, the game will easily adapt to the lower threshold of a computer that just barely meets the minimum requirements.

9.4 Art and Animation

Our game uses 2D pixel art, and is animated primarily through sprite sheets. Pixel art evokes a retro gaming aesthetic while additionally making visuals cleaner and easier to read, an especially valuable characteristic in the context of the precision required in platformers. Many other popular 2D platformers, such as Mario, Celeste, and Dead Cells, also use the pixel art style, so this choice was also made in our game to contribute to this tradition of the genre. Overall, the style and aesthetic we have chosen is not at all intended to be photorealistic, but scenes are well-populated and visually rich in a way that also does not lend itself to minimalism. We will not be relying heavily on visual special effects, especially used as cues to indicate events or actions, but certain visual cues, such as a checkpoint lighting up when the player reaches it, will be present environmentally.

This sprite-based style will be very light on the GPU, as it will not involve real-time 3D rendering, and all the animations will already be baked. There is very little that will have to adapt to real-time scenarios, which will require less processing power

9.5 Sound

The sound design of our game will use two audio channels, one for sound effects and one for background music (there is no dialogue sound). The sound effects will be short clips that indicate state changes to the player or the world, or to add ambiance. There will be independent sound

effects for all of the different choices of instruments that the player can use, and reaction sound effects that indicate whether or not an action has an impact. For example, successfully reflecting an enemy's attacks will make a sound to indicate a success. Another example of ambiance effects are the player's footsteps, which indicate the type of floor they are walking on, and that they are walking in the first place.

The background music will be a retro-sounding relaxed track that changes to indicate the stage that the player is currently in, such as the menu screen, in a transition, or in a main level (each of which will have its own song). In a finalized version of our game, the sound effects will be more customized, as well as mixed and filtered to fit the style of the character's current location. For example, there will be an added echo effect when in an enclosed space, such as the cave level, or a dampening to footstep sounds when running on grass.

10 Production Schedule

Preproduction Week 1: Curating Assets and Testing Mechanics									
Member Name	**Actual Date**	**Task**	**3/27**	**3/28**	**3/29**	**3/30**	**3/31**	**4/1**	**4/2**
Max	3/29	Curate 2d/3d: Lighting looks					X		
Sammy	3/31	Program and test Programmer HUD, curate avater					X		
Bryanna	3/31	Curate Level 0 environment, npcs, instrument assets					X		
Bryanna	3/31	Curate Level 1 environment, npcs, instrument assets					X		
Sammy	3/31	Curate Level 2 assets (environment, npcs, instrument)					X		
Max	3/29	Level 3 environment, npcs, instrument					X		
Nick		Level 4 environment, npcs, instrument					X		
Preproduction Week 2: Placing Assets and Setting Up Animations									
Member Name	**Actual Date**	**Task**	**4/3**	**4/4**	**4/5**	**4/6**	**4/7**	**4/8**	**4/9**
Nick	4/27	Program and test player avatar movement			X				

Bryanna	4/4	Program and test NPC movement and animation			X				
Sammy	4/5	Program and test grapple system			X				
Sammy	4/7	Program and test tambourine throw					X		
Sammy	4/7	Begin setting up level 2 (importing art assets, setting up tileset)					X		
Max	4/26	Setting up level 3 (importing art and tileset							
Production Week 1: Programming Animation									
Member Name	**Actual Date**	**Task**	**4/10**	**4/11**	**4/12**	**4/13**	**4/14**	**4/15**	**4/16**
Nick	4/10	Setup Gitlab	X						
Sammy		Create game levels as scenes and test access and updates on Plastic			X				
Max	4/26	Position 2d/3d: Position terrain, flora					X		
Bryanna	4/14	Position 2d/3d: Position architecture, props, terrain, avatar, NPCs for Level 0					X		
Bryanna	4/15	Position 2d/3d: Position architecture, props, terrain, avatar for Level 1						X	
Bryanna	4/16	Set up background music for levels 0, 1, and transition slides							X
Sammy	4/13	Program death collision / respawn system				X			
Sammy	4/16	Fix bugs in player movement and collision							X
Production Week 2: Programming Animation									
Member Name	**Actual Date**	**Task**	**4/17**	**4/18**	**4/19**	**4/20**	**4/21**	**4/22**	**4/23**
Bryanna	4/21	2d/3d: Position and program NPC animations					X		
Sammy		2d/3d: Program asset and health management and GUI status					X		
Bryanna	4/19	Program trumpet/double jump		X					

Sammy	4/20	Add level 2 music/SFX				X			
Sammy	4/20	Program and test camera movement				X			
Sammy	4/20	Animate player character				X			
Production Week 3: Programming Props, Weapons									
Member Name	**Actual Date**	**Task**	**4/24**	**4/25**	**4/26**	**4/27**	**4/28**	**4/29**	**4/30**
Max	5/3	2d/3d: Program instrument behavior and interactions			X				
Sammy	4/24	Program and test scene manager	X						
Sammy	4/24	Create start menu	X						
Sammy	4/25	Create transition scenes		X					
Sammy	4/26	Setup persistent data and debug scene switcher			X				
Sammy	4/25	Create Pause Menu		X					
Sammy	4/29	Weekly Testing and Bugfixing						X	
Sammy	4/30	Finish Level 2 Layout							X
Bryanna	4/30	Finish Level 0 & 1 Layout							X
Max	4/28	Finish Level 3 Layout							X
Nick	5/5	Finish Level 4 Layout							X
Production Week 4									
Member Name	**Actual Date**	**Task**	**5/1**	**5/2**	**5/3**	**5/4**	**5/5**	**5/6**	**5/7**
Sammy	5/2	Create player UI		X					
Sammy	5/3	Add movement sound effects			X				
Sammy	5/6	Final Debugging							X
Sammy	5/5	Add controller support					X		
Bryanna	5/6	Add transition backgrounds + poems						X	
Max	5/3	Scripting for clarinet				X			
Max	5/4	Scripting for water and bounce				X			

Max	5/4	Having people playtest, communicating feedback from playtest/QA experience				X			
Max	5/5	Finishing touches on level - adjusting parameters, reskinning enemies, making sure level feels good					X		
Sammy	5/6	Submission of prototype to Moodle							X
DEMONSTRATION DAY: 5/8 @ 4 PM									