Roll With It

The Entire Equestrian

Description:

An infinite runner starring an unnamed small woodland creature monster girl who is running on a snowball in a race against time (and the yeti chasing her trying to ~~eat~~ cuddle her). You must avoid obstacles, collect defensive powerups, and keep your snowball going fast enough to outrun the yeti. Try to make it as far as you can without being eaten!

Mechanics overview:

- Transitioning b/w Player States (Size of Snowball, movement): **Justin**
- Powerups: **Justin**
    - Acorn: Increased Turning Speed
    - Pinecone: Points Multiplier
    - Reasons to jump off snowball - - - - - - - - - - - - - - - - not included
- Obstacles/Enemies: **Jared/Sam**
    - Yeti Girl
        - Jump Attack w/ Prediction
        - Snowball Bowl
    - Trees, Rocks, Pits
    - Yellow Snow: Confusion Effect - - - - - - - - - - - - - - - - not included
    - Other Woodland Animals - not included

- Infinite Running w/ Modular Level Building: **Sam/Danin**
- HUD: **Lincoln**
    - High Score/Leaderboard - not included
    - Powerup Durations - not included
    - Snow meter - not in build currently
    - Particle Effects - done by Sam
- Scene Switching: **Lincoln**
- Art Assets/Animation: **Jordan**

Jordan Bischer - Art and Animation

Jordan created the art assets for the game. First, the model for the squirrel girl was created. After applying textures, some still pictures were taken to possibly be used in menu screens (and to show progress). Soon after, the backwards running animation was created.

The model for the yeti is considerably larger than the squirrel girl. As with the squirrel girl, some stills were also taken, as well as new stills involving both the yeti and squirrel girl. Multiple running animations were made to see which would best fit the game, and in the end, the running animations with arms forward in a chasing motion was chosen.

The acorn and pinecone power ups were made last. Luckily, these items did not need an animation, since there is a script to rotate them already created.

Intermittently, Jordan oversaw any art-related things (such as the environment and obstacles) and gave helpful advice for the coders. Her help in these aspects were greatly appreciated and kept up The Entire Equestrian's morale. "Basically, she is a goddess who has blessed us with her presence".

- Jordan

Jared Ebels – Yeti AI and Project Management

I was tasked with writing the code for the main enemy of Roll With It, the yeti. The yeti had two attacks that I'd have to program, and when not attacking would follow the player at varying distances. The first iteration of the yeti was just a red ball that followed the white ball of the player, following behind them as the player would move left and right. This wasn't too difficult to program, although the yeti would generally overshoot and just end up swerving back and forth forever, but it was functional.

The next step was to make the yeti jump in front of the player, which was significantly more difficult than I thought it was going to be. The yeti would never land right on the target, either undershooting or wildly overshooting, and trying to figure out the math that would actually work was a challenge. I kept attempting to launch the yeti in the right direction with the built-in AddForce() function, but it wasn't very effective and didn't get me the results I wanted. Eventually, I turned to the internet for answers, which gave me the perfect formula for the problem. The other attack, a snowball that would launch forward to kill the player, was much easier, since I'd already figured out how to spawn things around a certain point with the jump attack & the reticle that went along with it.

Once the yeti attacks had been coded, balanced, and automatically triggered (for the most part), my next big task was combining the code from everybody's

features into a coherent, functioning game. Using GitHub and GitKraken repositories and some general importing of assets, I was able to bring all the assets together into one Unity file, and only a few things broke along the way. Once everything was in, Justin and I went through and figured out what broke along the way, removing weird additions to the player movement code in particular that messed up sideways movement. The UI was the last main thing to be added to the game, and with that we officially had a fully functional, albeit barebones, infinite runner.

Overall, I think that the learning curve for this project was steep but manageable. The first task of making the yeti jump attack was definitely the most challenging and time-consuming aspect of the project for me, but once I'd figured that out, everything else went by a lot smoother. I think I have a much more solid handle on how Unity works and how to make things do what I want them to within the game engine, even in aspects I didn't do much work on, like UI and animation. As project manager, I think the main thing I took away from this was that we needed to keep better track of our progress and keep on top of documenting things in a process book.

Lincoln Grinenko - HUD and Scenes

I was tasked with the creation of the HUD and Menus, as well as powerups. I did not make any powerups though as the code required was integrated with Justin's player code and the prefab code created by Danin and Sam. The code needed for switching scenes was done by me instead of Justin as scene switching is done with a few lines of code that were integrated into my menus. Most of what I did was learned or adapted from youtube tutorials.

The menus consist of the starting menu, the options menu, the pause menu, and the game over screen. Every menu is created on a 2d canvas with all of the interactable assets placed on it. The start screen and game over screen are the simplest in construction consisting of sprite switch buttons and code that either loads a new scene or quits the application if the game has been built. The pause menu is next and besides just setting a canvas active, the menu has code to manipulate the timescale to stop the game. Finally, the options menu has the most code attached with a system of arrays and lists to populate a dropdown menu with all possible resolution options. Using Unity’s settings presets there is a dropdown menu that sets the visual quality of the game to either low, medium, or high. The game defaults to high but I had difficulty getting the menu to properly display that it was set to high, but I was able to use some code from the resolution dropdown to force the quality dropdown to update to the correct index upon launch.

When I was integrating in the sprites, I was stuck for two days trying to get the images to apply to the buttons. I eventually came to the realization that I had to convert my PNGs to UI sprites and once that was done, I could properly drag and drop the assets in.

The leaderboard was cut for time and the cooldown bars for the powerups were cut due to causing visual clutter. The game moves fast and having anything on the screen that isn't totally necessary, is a detriment to the player experience.

Justin Lai - Player States, Movement, and Power Ups

Originally, I was tasked with transitioning between player states, which included the size of snowball and player movement, and scene switching. However, as time went on, we realized that the power ups were more aligned with what I was already doing, and Lincoln was already doing things like scene switching.

The player states are the small, medium, and large sizes of the ball, which change dynamically as the player moves forward. Larger sizes increase the player's max velocity, but at the cost of turning ability/acceleration and having a larger hitbox. The power ups included are an acorn, which powers up the squirrel girl to have twice the turning ability/acceleration for ten seconds, and a pinecone, which doubles points earned for ten seconds.

The first thing I did was create a base for the team by following a Roll-a-Ball tutorial from Unity's website: https://learn.unity.com/project/roll-a-ball-tutorial. I needed to do this because most other members had to at some point interact with the player, so some base movement had to be available (in the end, I would be the only one still using this base playground to test code). This allowed me to make basic movement using AddForce() and power up interaction with OnTriggerEnter().

The next step in my process was adding code to relate to the snowball (player) size, mass, and speed at which it moves. To make it easier to view speed changes, I added a canvas that shows X-velocity (and later the Z-velocity). This was a

very fickle process, as using AddForce() deals with a lot of physics that makes fine-tuning numbers difficult. For weeks of class and out-of-class time, I would try to figure out the perfect formula for increases of mass, size, and angular drag to desired maximum velocity, acceleration, and negative acceleration (returning back to zero horizontal velocity), but could never get a set of values/multipliers that could make acceleration and negative acceleration equal, and many times, max speed would decrease with size due to mass increasing while force remained constant (when trying to isolate equal accelerations). These problems were because of using AddForce() instead of directly manipulating velocity. While a tutorial I followed on youtube mentioned to use AddForce() when dealing with physics, Varun told me that I can directly manipulate velocity if I want to because that tutorial is for absolute beginners who may not know how physics works. After switching to velocity manipulation, making the snowball reach its max speed with appropriate velocities became an easy task that was mostly fine-tuned within two weeks.

While I was still using AddForce(), I added a point display on the canvas, tied to Z distance traveled. Along with that, mass and size/scale were also tied to Z distance traveled, stored in a separate variable called distanceTraveled that ranges from 0 to 100, which maps to a mass between 1 and 2, and scale from 2 to 5.

For the final part of individual work, I took the squirrel girl model that Jordan created and added a script to keep it running permanently on top of the snowball

based on its size. The script also uses the snowball's current X and Z velocities to calculate what direction the squirrel girl should be rotated to appear to be correctly running on the ball. This script was modified and used for Jared's yeti to solve an issue of the yeti model rotating as if it were a sphere.

Since base power ups were already made from following the Roll-a-Ball tutorial, they were modified to now increase the multiplier for points and velocity. Similar modifications were done with the group to make sure collisions with obstacles causes the obstacles to disappear and collisions with the yeti jump/landing and large snowball attack causes the player to die and go to the game over screen. While these interactions are great, we unfortunately did not get to jumping mechanics we originally envisioned, which were intended to be a side/mini game anyway.

While a majority of my part in this project was individually done, I understand and left placeholders for the parts of the project that was done by the rest of The Entire Equestrian. I also gained a greater understanding of making a game through helping to make the merged code function correctly.

Sam Robey

//

Danin Tenerife – Infinite Level Building with Random Obstacle Spawning

I helped make the code and prefabs that would endlessly create the world the player runs through. Jared, Jordan, Justin, and Sam were instrumental in helping complete this task. The first task to complete was to figure out how to infinitely spawn prefabs, and the nature of how the prefabs would spawn. Much of the first couple of weeks was spent finding ways to implement the infinitely spawning prefabs.

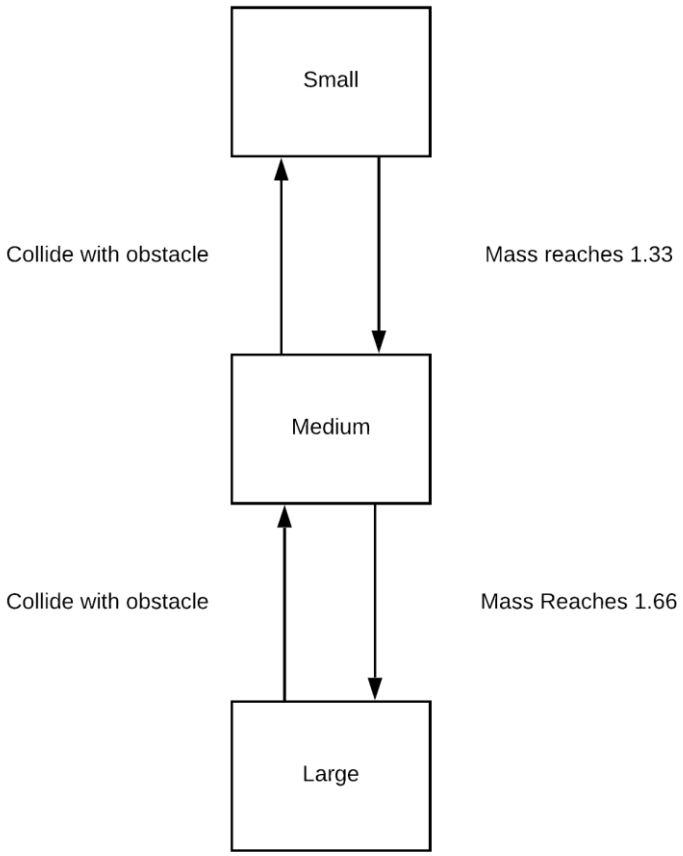
I started out creating a plane with a sphere and a box and worked on getting more planes spawning. Justin helped implement the movement code he was working on to aid in the testing process. I tried looking at different tutorials on Youtube for creating an algorithm that would infinitely spawn platforms until Varun ultimately said to create this without looking it up on Youtube. He then created a template for me and Sam to work with. This template consisted of three planes split into three distinct lanes. A connector was placed a certain distance away from the planes that would spawn more planes as the player passed over it. This worked as a good starting point from which more ideas could be derived.

From here, it was decided that a simpler approach to creating platforms would be to create one large platform instead of several small ones. Problems

immediately began to manifest, due to several factors. Firstly, the movement code provided by Justin was still being worked on, so our movement was clunky and had no cap on speed. Given enough speed, it was possible for players to phase through the floor. The ball would also inexplicably jump and rolled like a box down the hill. This would not be remedied until much later. However, other tweaks such as platform size and spawn distance were quickly worked into the game.

From here, trees, rocks, and logs were provided by Jordan and promptly implemented into the level prefab. Jared helped to properly spawn obstacles in random places along the prefab, but we ran into issues when items would not collide or despawn properly. This was fixed by adjusting hitbox sizes and changing the point the player had to pass for obstacles to despawn. One problem that was plaguing the project for weeks was the strange collision on the sphere. This was eventually discovered to be a problem with the sphere's hitbox, as it was a box collider instead of a sphere collider. Finally, once everything was in order, the assets were uploaded to GitKraken and shared with the team to be merged with the rest of the code.

State Machine Graph


Small
Collide with obstacle
Mass reaches 1.33
Medium
Collide with obstacle
Mass Reaches 1.66
Large

UML Diagram

355 UML Diagram

Jared Ebels | March 5, 2020