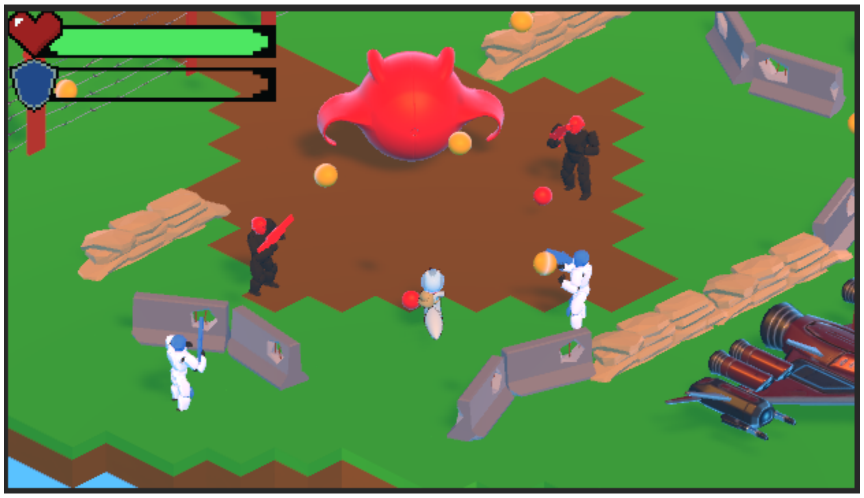
Overview



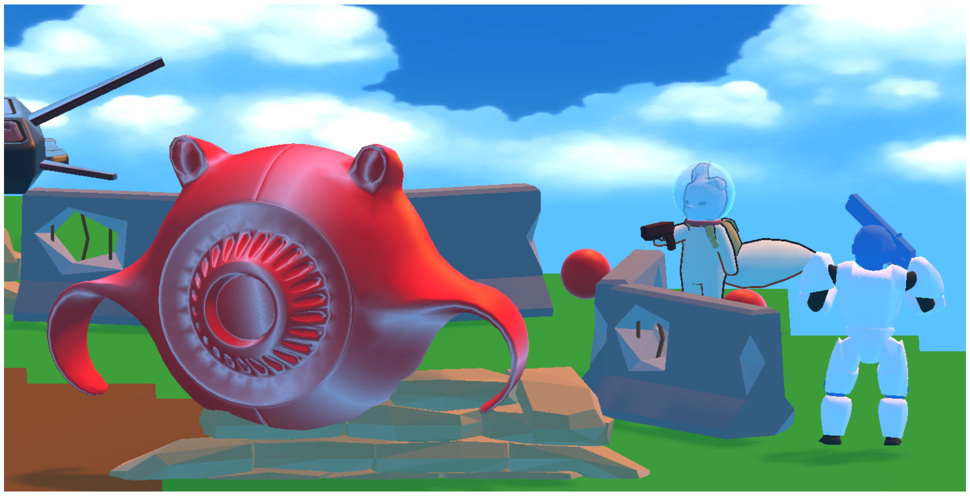
Star Trails is an isometric roguelike twin-stick shooter that features a variety of enemies, guns, and levels for the player to explore. The goal is to get through three levels full of enemies in order to reach the boss. In true roguelike fashion, if you die you have to go back to the beginning of the level.

Team

Brian Lim
Christian Caparroso
Francisco Quiroz-Vivar
Ellen Yim
Huiwen “CoCo” Chen

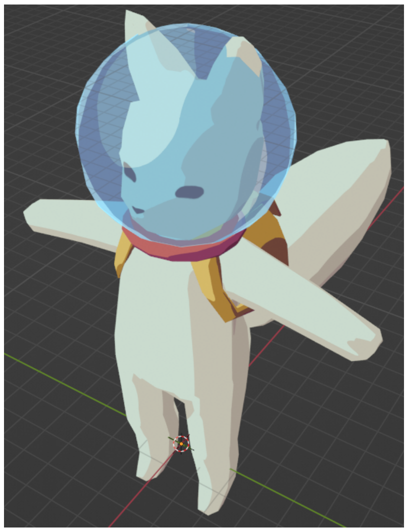
Game Description

Story



The Fox lives on a peaceful forest planet with their fox mother. All is well until The Fox’s mother falls ill and the only cure is a mystical space flower that is found in the deepest parts of the forest. To make matters worse, an alien force has invaded the planet and captured the space flower! It’s up to The Fox to drive them away from their home and save their mother.

Player Character: The Fox



The Fox is the character the player controls. Their model as well as their arsenal was hand-modeled by our very own Huiwen (CoCo) Chen.

Enemies

- Rifle guy
- Pistol guy
- Flamethrower guy
- Sniper guy
- Invader Drone Boss

Game Flow

The player starts from level 0 and follows the path to fight and defeat the enemies. Once all the enemies are defeated a teleporter will appear and it will transport the player to the next level; either forest trail, desert, and underwater, and then to Boss Battle. Per level, the player can pick up and use items to restore health/shield. If the player's health value goes to 0, they die and have to start back at level 1. After Level 0, the player must go through three levels before finally arriving at the boss stage and confronting the Invader Drone. Each level has a random chance of being one of three designs from a pool of pre-generated levels. Once the Invader Drone and its underlings are defeated, the game is won!

Implementation

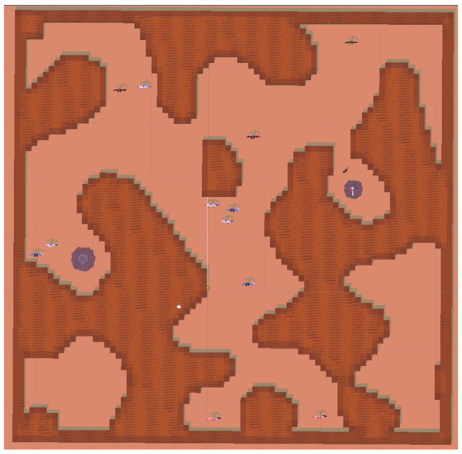
The game engine we used is Unity, using procedural generation to generate the levels and scenes for our game.

Inventory System

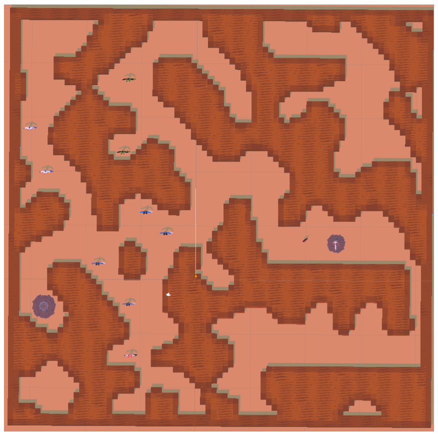
The inventory system is composed of Scriptable objects that makes saving data for an item object easy to manage. Scriptable objects are data containers used to store amounts of data, independent of class instances. In prefabs for items, can simply attach the pick up item script and drag the scriptable object to the field. BaseItem script creates the item object consisting of item name, item value, item count, and ui sprite. The player character interacts with items by picking them up and getting stored in inventory. It is managed by a pick up item, InventoryManager, item controller scripts.

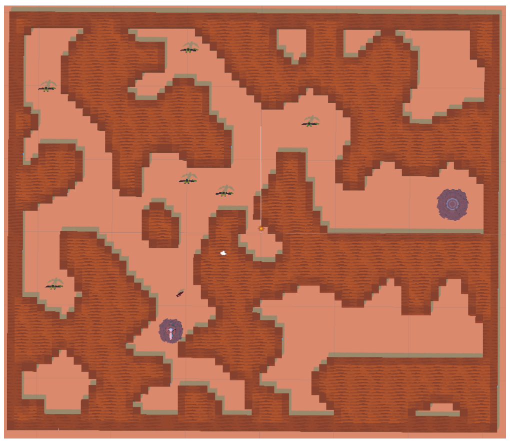
Procedural Level Progression

The "Procedural Progression" levels are scenes that have environments pre-generated in editor mode using a [2D procedural level generation algorithm written by Unity dev Ethan Bruins]. Bruin's algorithm uses the 2D RuleTile system and Perling Noise to create procedurally generated side-scrolling levels; we adapted his system for our 3D isometric tilemap system by switching the orientation of the tilemap grids and adding gameObjects to a RuleTile palette. In the demo, the player progresses through three levels: Trail, Desert, and Water. Each level has a pool of three scenes which have different maze designs/enemy variety but the same theme. When the player teleports to the next level, the game randomly chooses one of the three scenes for a certain progression level. The game keeps track of the player's progression level through a progression_level value attached to the scene's teleporter trigger script.



Desert 1


Desert 2


Desert 3

UI

The current UI covers the HPbar, ShieldBar, pause menu, death menu, camera, and camera zoom, and their functions and visuals. The HPbar is an indicator of how much more health a player has; this is attached to the player's stats and has two special cases, when it is 0 and the range from 0-100. The health at 100 is green, turns yellow when the players health drops below 66, red when they drop below 33, and it is empty when they reach 0, and when it reaches 0 the game is over and the death menu appears allowing the player to restart the game. The ShieldBar represents a sort of additional or "temporary", this value changes a lot as the player is able to regenerate shields. This is also attached to player stats and displays how much shield they have left. When the player is not taking damage this bar fills slowly, when they take damage there is a timer that will not allow the shield to fill until it is up. When the player sees an empty shield bar, the Hp bar will begin to deplete upon taking damage.

The menus have buttons that have their own functions, the pause menu can only be accessed when the player hits the 'esc' key and the death menu only appears when the player has 0 health. The pause menu has a button option to resume the game or open the inventory. The death menu only gives the player the option to restart the game

Combat/Movement

The combat system revolves around the concept of the player picking up and using the same weapons that the enemies drop. As such both the player and enemies use a very similar implementation of attacking. Any entity (i.e. player and enemies) in the game contains stats which are scripted objects which represent different values they may contain. Some examples of stats are max health, shields, base damage, and speed. These values are used to determine when an entity receives damage and how much damage they deal. Weapons also contain values that track different properties such as fire rate, bullet force, bullets per shot and delay per bullet. With this system weapons simply act as presets that are used to update the values for the entity who currently has them equipped. Bullets are game objects that are created and shot towards the direction the entity is facing. When a bullet is created it is assigned a tag which indicates what type of entity it is from. This is used when an entity detects a collision with a bullet and is able to determine whether to ignore it or be damaged by it depending on who shot it.

Boss

The boss enemy is a specialized version of both the combat system and the enemy entity framework. The boss uses the same process of locating and following the player; however instead of using the weapon it has, it instead randomly selects one of three attacks depending on the distance between the player and itself. If the player is too close to the boss, it will use a melee attack to swipe at them. If the player is out of this melee range the boss will randomly select between using a mine attack or cannon blast. The mine attack launches multiple static projectiles that sit around the map for about 10 seconds before disappearing. If the player comes close to one of these objects it damages them dealing high amounts of damage. The cannon attack is a slow and large bullet that after a certain distance splits into much smaller and faster. While the cannonball does a high amount of damage the smaller pellets deal less to account for the larger spread and speed.

Project Post Mortem

What tasks were accomplished?

- Player Movement (With both controller and Mouse+Keyboard)
- Isometric Camera
- Enemy Movement & Behavior
- Handmodeled & Animated Player Character
- Item collection & interaction

- Pause Feature
- Twin-Stick Shooting Combat System
- Custom Projectile System for Player and Enemies
- Pistol & Rifle weapons
- Player UI (Health & Shield)
- Inventory System
- Weapon Switching
- Boss
- Procedural Level Progression System
- Scene Teleportation
- Enemy Item Drops
- Enemy Variety & Assets
- Projectile Assets
- Starting Level, Three Procedural Progression Levels, Boss Level

What planned tasks were not done?

- Procedural Generation at runtime
  - Since the RuleTile system is available for use with GameObjects in 3D space, we were able to adapt it to 3D but had trouble figuring out how to effectively place the fox, teleporters, and enemies when generating at run time.
  - CoCo Chen wrote her own procedural generation algorithm that creates expansive and varied maps with random seeds; however, it is in progress and we're still trying to figure out how to place the teleporters and player/enemy entities effectively.
- Sound
  - We were unfortunately not able to have sound for our game but we plan to have it when working on it on our own.
- Dialogue/Menu UI
  - We wanted to have a start menu and menus for dialogue and more UI for story.
- Multiplayer
  - This was a stretch goal which involved local multiplayer; the plan was that someone would press start and their controller to start playing couch co-op style. We planned to not try developing this task until we got all the core gameplay features done first.
- Bounties
  - Another stretch goal. The plan was to have a system in place where the player would be rewarded for taking out a specific target in the next level.

How did scrum work for you?

Early during development we agreed to meet every Monday at 6PM and Wednesday at 10:30PM for 30 minutes to sync up what we worked on the previous day, what we plan on tackling for the current day, and any questions we had in terms of development or dependencies from other team members. We would meet on Discord voice chat since it is an easy platform for communication and has a screen share feature, making it easy to show other people development errors or demos. During our Friday lab is when we plan for the succeeding week's sprint and assign tasks. Brian Lim acted as the SCRUM master and facilitated the meetings, but generally in terms of picking and creating tasks the team would work together to brainstorm what needed to get done during sprint planning meetings. The entire team was consistent in joining these meetings and getting the appropriate amount of work done every week.

Midway through development we decided to turn our standup meetings into full on work meetings in order to strengthen collaboration and motivate progress in the project. We would have a brief phase in the beginning of the meeting where anyone can ask questions/comment about the project, then we work on the project while staying on the voice chat to keep each other accountable.

What would you do differently?

One area of the game we could have spent more development time on is the roguelike elements–ie. Variety of items/weapons, persistent progression, and resource management. In the 7 weeks that we had to develop we needed to get the core gameplay features such as movement, combat, and level progression finished so we had to pick and choose which stretch features we'd want in the game. Midway through after finishing player movement and combat we were interested in trying to figuring out how procedural generation works; this ended up taking a considerable amount of time, so if we were to do the project again, we'd probably try to get a good variety of simple stat-changing items in the game first before tackling a complex task like proc gen.

For development, we would have started doing our collaborative work meetings from the start. We felt that being together on the same call kept us all accountable and any blockers (ie. development questions, directly talking to team members working on a task dependency, etc.) can be resolved during these work sessions.

Conclusion

Thank you for proctoring the class! We all had a lot of fun coming up with ideas for the game and learning the development techniques we wanted to tackle this quarter. Here’s an alternative design of our game cover 🙂