Willow Wood Refuge was made by a team of 12 people over 7 months

A 2D classic pixel art style platformer and exploration game where the player lives as a hermit in a dangerous forest, the player stumbles across a group of refugees escaping a mysterious magical attack on their home. Explore, forage, and cook to nurse them back to health.

For this project, we used the MonoGame framework. MonoGame is a basic C# cross-platform framework implementing the XNA API. We chose this framework rather than an engine like Unity in order to learn by essentially building our own engine to best fit our game and learn along the way. With this in mind we created many of the libraries in this project from scratch rather than using pre-existing libraries. We used OpenGL in order to create builds for both Windows and MacOS.

My Contributions

Custom 2D Physics Engine

CollisionBox – The basic physics component added to any physics object. Based on MonoGame.Extended.RectangleF class, contains all additional functionality necessary for platformer physics with custom collision and overlap events.

Collision/Overlap info – All information associated with a collision or overlap event, created and passed to event handlers when event called.

PhysicsHandler – The core class that handles all physics interactions with layers based on object labels and masks for collision and overlap with other objects.

CellGrid – Container used for PhysicsHandler layers in order to cull unnecessary comparisons in collision and overlap updates.

Idle NPC Dialogue System

NPCDialogueSystem – Core class for holding and managing all interactions, with functionality for weighted selection of possible interactions based on conditions. Reads and separates interactions from a tsv created by our writer, sending unparsed interactions to interaction class.

NPCDialogueInteraction – Class for a single interaction, with all relevant information and events. Parses information when loaded, sending unparsed individual line info to dialogue line class.

NPCDialogueLine – Class for a single line of dialogue. Contains the all relevant information for a single line of dialogue, parsed fully when loaded.

AI Movement/Behavior

NavMesh – Class containing all relevant information about a character’s navigation map, including edges. Generated on construction.

NavMap – Class containing all possible navigation points, generated dynamically when entity is created based on collision box size and wall tile information.

NavPoint – Information about a single point with relevant tile and platform type information.

BaseCharacter – A basic abstract class for movement and animation behavior shared by all characters.

AICharacter – An abstract class inheriting from BaseCharacter for all behavior control and navigation shared by all AI characters. Uses a simple state machine for behavior.

Enemy – Class used by all enemy objects inheriting from AICharacter, with all relevant information to define individual monster types and behavior states with wandering, attacking, and cool down.

NPC – Refugee class inheriting from AICharacter with behavior states for wandering and conversing.

Tile Map System

TileMap – Extends the MonoGame.Extended.Tiled tile map loading and rendering classes to generate collision objects, populate spawn/forage points, world lights, and manage map entities.

SpawnPoint – Abstract class for all information common for game entity spawn locations, including type or range of types that can be spawned at given location.

EnemySpawn – Inherits from SpawnPoint, defining Spawn() to create a new enemy of given type at spawn point.

ItemSpawn – Inherits from SpawnPoint, defining Spawn() for a dropped item.

ForageSpot – Class containing all relevant texture and growth information for a foraging location of type given by the TileMap.

ForageInfo – Static class defining growth and name information for all possible foraging types.

PickupItem – Simple class for items that have been dropped, with collision and despawn timer.

Stylized Shadow Shader

LightManager – Class managing static and dynamic light objects, as well as formatting arrays of data for the shader.

AreaLight/DirectionalLight – Classes for easily defining relevant information for a given light object.

CastShadows – A custom HLSL shader that casts rays from light sources given light info and occlusion mask to calculate light value of each given pixel, assigning the value to the alpha of the pixel. Smooth shadow render target is then drawn with a custom HLSL shader created by a teammate to map opacity to a dither spritesheet.

Stylized Weather Shaders (Not in current build)

WeatherElement – Abstract class for general information about a given type of weather common among all weather, including a generated noise texture.

Rain –

C# class – Generates a random noise texture and manages scroll of rain falling as well as direction.

Shader – An HLSL shader that draws only the pixels above a given density value as the specified color.

Fog –

C# class – Generates an OpenSimplex noise texture with horizontal wrap and manages horizontal scroll of fog.

Shader – An HLSL shader that uses a vertical falloff and density to determine the opacity based on the noise texture of a given pixel, finally clamping opacity values based on a user defined step value and drawing pixel as specified color.

TextureWrap – An HLSL shader that draws a texture in a given space with a given offset, tiling to fill in the space. Used by both weather classes for continuous textures.

Custom Camera Controller

Extends the MonoGame.Extended.OrthographicCamera class to manage pixel dimensions of game world in both windowed and full screen views, as well as controlling camera movement with the player’s location, world bounds, and player bounds.

Custom Debugging Display

A series of debug views that can be cycled between for both the game world camera overlay and a smaller mini-display of the entire current scene. States include camera debug, physics debug, AI debug, and player debug.

Sprite Font Manager

Simple class managing the loaded in sprite fonts and adding functionality for formatted printing of text with different alignments and custom parsing for line breaks.

Texture Atlas Manager

An extension of the XNA Texture2D class, adding functionality for reading in a json file associated with a texture to generate a texture atlas. Contains functionality for different methods of drawing textures from atlas, as well as a default error texture.

Content Loading Scene

A scene which uses the main graphics thread to display the amount of time spent loading and the current batch being loaded. ThreadPool is then used to load, initialize, and generate all game content, tracked by the main thread to give user feedback.

Something Fishy is a solo project originally done over 5 weeks, with ongoing work

A 3D low poly casual game where you run around on islands floating in the sky catching fish, trading with a turtle for bait, and catching more fish.

This project was developed in Unreal Engine 4 using C++ for all elements except for UI, materials, and particles, which used Blueprints. The term game is used loosely for this project, as it’s mostly an environment for me to experiment with AI and procedural generation techniques that interest me with a vague game loop.

My Work

Flocking Behavior

Flock – An actor that works as a management container for the boids in the world, creating boids and managing shared information including behavior weights and movement limits.

Boid – An actor inside the flock, with the three basic steering behaviors described by Craig Reynolds as well as five additional steering behaviors.

Separation – Steer to avoid crowding local flock mates.

Alignment – Steer toward the average heading of local flock mates.

Cohesion – Steer to move toward the average position of local flock mates.

Target – Steer toward closest target, if in perception range.

AvoidObstacle – Steer away from local obstacles.

AvoidPlayer – Steer away from player, if in observable range.

Bounds – Steer away from area bounds.

Centralize – Steer toward center of area.

CellGrid – Simple container to optimize neighbor calculation by storing boids in a grid and only comparing to calculate neighbors with boids in adjacent cells.

Bait Targets

BaitManager – An actor managing information and spawning pertaining to bait in the game world.

Bait – A simple actor spawned by BaitManager which acts as a target for boids. Emits simple GPU particles to represent scent and bite particles. Shrinks after each bite until value reaches zero, destroying actor.

Custom Player Pawn

PlayerPawn – A custom pawn actor controlling the player’s camera, capsule collision mesh, HUD, and line trace for interaction.

PawnMovementComponent – Custom movement component, processing player input for pawn movement.

World Layout

Designed the base layout of the test island with prop positioning and lighting.

Art Assets

I created the bait model, as well as all of the UI assets and sign art.

UI

All UI with value tracking and animation was created by me.

Procedural Island Mesh Prototype (not currently in build)

Prototype for procedural algorithm to generate island meshes to be used in place of the current flat box for variation and platforming. Created with THREE.js library for WebGL.

Additional Sources (Models)

Acorn Bringer ‐ https://www.acornbringer.com/

Pulsar Bytes ‐ https://www.pulsarbytes.com/

Just Create ‐ https://www.artstation.com/olehlila

Project Nature ‐ https://www.artstation.com/nilsarenz

Paltergeist was made by a team of 3 people over 5 weeks

A 2D web-based puzzle game in which you play as a ghost who must use their scare energy to possess, move, and scale objects in an old mansion in order to help a child ascend through each floor and retrieve their lost balloon from the attic.

This project was developed in Phaser3 using JavaScript and GitHub for version control. This project also used the Matter.js physics library. As a shorter project with a team where we were all fairly new to game development, we really wanted to push what we were capable of but still keep a reasonable scope with backup plans for if we were unable to execute all of our desired features.

My Contributions

Custom Level System

A system for loading in levels our designer created in JSON format as well as managing resources. Level sprite assets are loaded and displayed as they become visible with the ascent. When level is changed the physics calculation is disabled for the previous level and physics bodies are created and enabled for the new level.

AI Finite State Machine

A simple state machine for the child character to wander the available space taking into account the obstructions the player moves around, run to hide and glance back frantically when scared by witnessing ghost activity, and run to the level exit when the path is cleared.

Physics Wrangling

Setting up all of the physics objects that were generated, refactoring when switching from Phaser physics engine to Matter.js, and fiddling with variables for the best movement feeling with different scales.

Character Movement

The interpolation and direction changes used for the kid and the ghost’s movement to give the kid a solid, tangible feeling in the level and the ghost a light floaty feeling.

Team Size: 8 Duration: 3 Weeks Role: Lead Programmer

A pair of humans compete against a raccoon to gather the most trash from the neighborhood by using their unique abilities.

This was a digital adaptation of a board game designed by the same team of people. It was developed in Unity, doing our best to keep the scope small enough that we could make all of our assets ourselves and keep all of the functionality needed for the board game during the limited timeframe that the project was allowed.

My Contributions

Map Generation

HexMap – Manages generating the hex map from a txt file, formatting the tile placement, and functions for interacting with the tile map.

TileInfo – Stores all information about the specific tile including its type, coordinate, and token occupant.

Possible Path Finding

Calculated all tiles that the player could move to and highlighted given tiles, not allowing the user to select impossible moves.

Audio Management

Simple audio driver that plays sound effects on given events.

Camera Controller

User input controller to manipulate camera at fixed angle with rotation, panning, and zoom.

Player Movement

Paths movement between start and destination with interpolated movement between tiles and piece rotation.

Team Size: Solo Duration: 1 Week

A 2 person co-op remake of Rocket Patrol where a rock and scissors must work together to take out paper airplanes while avoiding cloud obstacles.

This was a very short independent project based off of provided code replicating the game Rocket Patrol. Developed in Phaser3, my main goal was to step back from the coding I normally focus on and instead focus on asset creation and reworking the game mechanics.

My Work

Art and Sound Assets

All art and sound assets were created by me.

Rock and Scissor Mechanic

Reworked the logic for the previous rocket launch to instead be dependent on a second actor, the scissors, overlapping and cutting in order to drop.

Cloud Obstacles

Set up a system for randomly choosing and spacing a series of clouds to act as obstacles scrolling across the landscape with frequency based on difficulty. Also set up collision event to stop the rock and play an animation if the falling rock hits a cloud.