*(working title)*

# MODULAR

GitHub Repository Link: IntroGameProject – GitHub [https://github.com/JaydenS115/IntroGameProject]

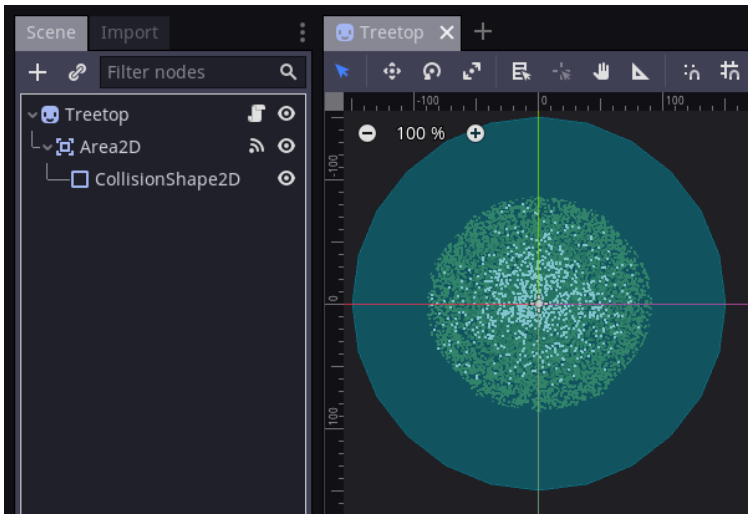
Project By: Jayden Stearns (810989967)

**[Jump to Video Presentation]**

## GODOT ENGINE INTRODUCTION

My Introduction to Game Programming Final Project was implemented using the Godot Engine (specifically, Godot Mono version 3.3.4). Godot is very similar to Unity in many ways, excluding the obvious differences in terminology at times. Some of the primary similarities and differences are highlighted below.

First and foremost, Godot heavily leans upon the idea of a tree-based scene structure. Where Unity features GameObjects to which Components can be added, Godot instead uses “Nodes” to which the programmer can add other nodes as children to expand their functionality. Consider the following example of the scene tree:



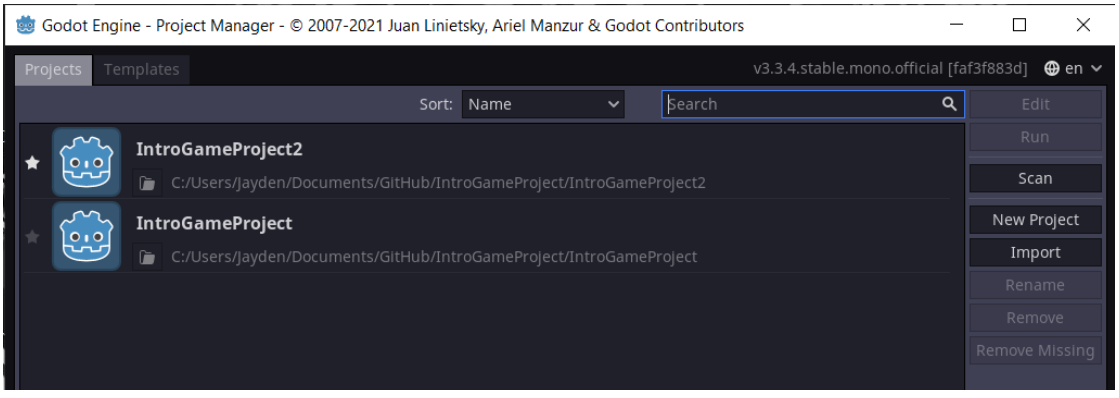
Pictured above in the scene tree hierarchy is a sprite node, storing the sprite of the top-down tree pictured to the right of the image. For additional functionality, it was useful to include a trigger area as part of this sprite, shown by the light blue circle sxtending beyond the sprite itself. As opposed to a CircleCollider component being added to the sprite object itself, instead an Area2D node is added as a child of the sprite. Then, a CollisionShape2D node is added as a child of the Area2D to define its actual bounds (in this case, a simple circle). Finally, through code, the Area2D is connected with the sprite to allow for the trigger entering and exiting functionality very similarly to how it would exist in Unity.

One very notable similarity between Unity and Godot is the ability to save a GameObject and later create multiple instances of the object in another scene. In Unity, this is accomplished through Prefabs. In Godot, you can save a scene to the project directory, then add it to another scene as an object of the other scene. In the above image, the Treetop prop is saves as its own individual scene. Then, in the level scene, an instance of this Treetop scene is added to the scene as its own node, allowing for a much more orderly scene view and quicker replication, as shown below:

As children of the “UpperProps” Node (used here as a container for the many upper-visibility props of the scene), the Treetop scene is instantiated many times (as shown through the movie “clacker”). Each is an instance of the Treetop scene itself. Each of these Treetops has its own Area2D and collision shape, but they are not shown here because they are simply instances of the master Treetop scene.

A final fundamental similarity between Unity and Godot is their use of programming languages. Godot supports a Python-like language known as “GDScript”. Additionally, if using a “Mono” version of Godot, it offers full C# support, a fact it shares with Unity. Personally, I used exclusively C# exclusively, due to both my level of experience in C-based languages and the fact that C# supported my heavily-modular coding style well, as explained in later sections below.

To load a Godot project, there are many possible approaches. First and foremost, download the appropriate version of the Godot Engine. Because my project is implemented exclusively in C#, I believe that the Mono version of the engine will be required to build my source code properly. After executing the Godot Engine application, you will likely be greeted with the following screen:



From here, a directory can be scanned for Godot Project files (signified by their extension “.godot”).

Alternatively, simply opening the Godot Project file should result in a similar outcome of opening the project.

## A FOREWORD ABOUT A CHANGE IN PLANS

The original working title for my Final Project was “Archivist”. The change to a completely different title is much more than simply a symbolic one. I have encountered many development issues throughout the semester that caused me to make the unfortunate decision to drastically change the direction of my Final Project.

Specifically, Archivist aimed to implement a hexagonal-grid turn-based system in which the player had infinite time to think of their next action. Only once decided would the actions of all entities be carried out simultaneously. This came with *many* implementation difficulties. First and foremost, due to the obscure format of such a game, the Engine did not have many tools built-in to support this development at all. For example, as noted below, I needed to implement an entirely-custom action management system just to have this original idea function. One of the complicated actions was even the not-so-simple movement of entities around the grid. Following my previous Iteration Report, a nightmare of mine became reality. When further testing the system, I realized that the entire fundamental concept *simply was not fun*. As I alluded to in a previous discussion post, game controls “feeling good” is important to me, but the concept I was working towards had fundamental flaws that made it feel clunky.

Rather than try to cobble together a system that I had fallen out-of-love with, I decided to largely abandon my original idea.

As such, I changed my plans. Instead, I would include the *spirit* behind the original system in a much more workable format. Instead of being a simultaneous turn-based system based in a grid, I opted to move toward a real-time system. However, because of ym own predispositions towards real-time games, I include an at-will paused functionality to stop and absorb the current situation, as well as freely-usable time slowing related to high-action moments.

# CORE MECHANICS & GAMEPLAY

## VERSION 1

These mechanics encompass the functionality in the first attempt at a game (my original concept) that I was able to accomplish.

- Top-down perspective, controlling the player character from essentially directly above.

- Tile grid movement based upon a hexagonal grid, allowing movement in 6 adjacent directions. This is highly preferable (in my opinion) over square tile movement, in which diagonals should be taken into account differently than adjacent-tile movement.

- Time-based gameplay, somewhat mimicing real-time gameplay by having the player select actions to take, after which time proceeds for all entities as normal.

- Camera control over the game field as desired by the player.

## VERSION 2

These mechanics encompass the functionality of my second (much more complete) attempt at an enjoyable game.

- Top-down perspective, controlling the player character from a directly-overhead perspective.
- More engaging real-time movement using physics simulation and collision.
- Time-control mechanics, where the player can freely pause the real-time actions as desired to think. Additionally, at-will time slowing effects are readily accessible to the player.
- Camera control over the game scene and player, allowing the (somewhat more limited) ability to view the player’s surroundings.
- More refined heads-up display and visual effects for player engagement.
- Easily-extensible node structure to freely and easily create new weapons and items.

- A more dynamic play area, such as through visual effects that can change in response to the player, making the environment feel more alive and dynamic.

# SUBTASKS & GENERAL TIMELINE

## VERSION 1

As explained further below, Version 1 of my game project was in development through and including the 4th iteration report. I then changed directions for this, the Final Iteration Report.

### ITERATION 1

| **Assigned Tasks** | **Completed Tasks** |
|---|---|
| • Action Manager [*REMOVED*]<br>◦ Handler class designed to handle all actions of entities on the scene<br>◦ *Removed from iteration* due to brainstorming about a better way to implement using the Entity base class.<br>• Implement Action Class<br>◦ Base class that generalizes into different action types, including movement, attacks, etc. | • Location Structure<br>◦ Completed and implemented as member of Entity class, storing location, as well as converting TileGrid location to global coordinates for the purposes of drawing and locating entities.<br>• Entity Class<br>◦ Completed (for current functionality), and used to generalize the functionality of the player. Can be used for |

| | |
|---|---|
| • Implement Location Structure<br>  ◦ Store the location of the entity using TileMap coordinates<br>• Implement Entity Class<br>  ◦ Generalize all action-taking nodes (Entities) as one class that they can inherit from. | any entity in the future.<br>**Incomplete Tasks**<br>• Action Class<br>  ◦ Insufficient time to complete action task. The main action of movement is currently implemented in a barebones manner around the tile grid. |

**ITERATION 2**

| **Assigned Tasks** | **Completed Tasks** |
|---|---|
| • Colliding Movement<br>  ◦ Update Gridlines TileMap to add collision with a TileSet of Hexagons with various walls.<br>• Camera Movement & Control<br>  ◦ Move Camera using Mouse controls and the camera script & node.<br>• Action Base Class<br>  ◦ Abstract base class from which all other “Action” types will inherit.<br>• Basic Time System<br>  ◦ Adjust real-time elapsed by a factor that can change as needed during play | • Colliding Movement<br>  ◦ Completed using TileMap collision and a forced RayCast2D that detects if movement is possible.<br>• Camera Movement & Control<br>  ◦ Implemented Camera panning using mouse movement, Zoom in/out, and center onto Player.<br>• Action Base Class<br>  ◦ Implemented in conjunction with an Action Handler node that all Entities will have as a child Node.<br>• Basic Time System<br>  ◦ Complete with a script in the base level Node.<br>**Incomplete Tasks** |

| | |
|---|---|
| | • Re-implement TilePosition Class<br>◦ Originally combined with the Entity base class, I aimed to re-separate (more modularly) into a discrete class. |

**ITERATION 3**

| **Assigned Tasks** | **Completed Tasks** |
|---|---|
| • Complete Time System<br>◦ Fully implement tick time functionality, through speed increase/decrease and pausing<br>• Re-introduce Tile Position Class<br>◦ Properly implement a utility class to manage Tile Positions<br>• Implement Proper Entity Movement<br>◦ Allow entities to move properly and uninterruptably<br>• Add basic Mouse-Over Information<br>◦ Implement tooltips on mouseover of entity near mouse cursor | • Tick Time System Completed<br>◦ Tick Time system functionality was completed, scripted in the root node of the scene<br>• Tile Position Class<br>◦ Tile Position class was re-implemented as a member of the Entity base class<br>• Proper Entity Movement<br>◦ Movement implemented using Action class inheritance and action Handler |
| | **Incomplete Tasks** |
| | • Add basic Mouse-Over Information<br>◦ Other functionality was much more involved than expected. As such, I intended to approach this concept when |

| | |
|---|---|
| | looking at other UI elements in later iterations. |

**ITERATION 4**

| **Assigned Tasks** | **Completed Tasks** |
|---|---|
| • Design Simple Ground Tileset<br>◦ Design a simple Tileset to have a visible ground surface.<br>• Implement Background Layer<br>◦ Design a background layer behind the ground for 3D-like perspective.<br>• Begin work on HUD<br>◦ Start learning HUD and draw layers to move with camera node.<br>• Add basic Mouse-Over Information<br>◦ Add information that pops-up on mouseover of entities. | • Ground Tiles<br>◦ Created Tileset and TileMap for ground layer, which includes some basic autotiling.<br>• Background Layer<br>◦ Created using ParallaxLayer functionality to follow camera smoothly.<br>• HUD Introduction<br>◦ Much more complicated than it needed to be. Now that I had grasped the basics, I had hoped it would move more smoothly. |
| | **Incomplete Tasks** |
| | • Add basic Mouse-Over Information<br>◦ Other UI elements were more complex than anticipated this iteration. |

# VERSION 2

It was shortly following the completion of the 4th iteration that I realized the shortcomings of my approached that are detailed in previous sections. For the final iteration, I massively changed directions. However, not all was lost, as I was able to adapt code into the new, more interesting, format.

## ITERATION 5

| Assigned Tasks | Completed Tasks |
|---|---|
| • Implement player movement<br>◦ Introduce more engaging player movement to support enjoyment.<br><br>• Create a general TileSet<br>◦ Design a simple TileSet for the purposes of testing the new system.<br>◦ The new system is not contingent upon a Hexagonal structure. As such, square textures are used for simplicity and speed of development.<br><br>• Dynamic-visibility Props<br>◦ To support the top-down game style, create a prop handler that can control visibility based upon player proximity.<br><br>• Adapt Camera control<br>◦ Modify camera control as needed to support the new system.<br><br>• Adapt Tick handler<br>◦ Modify the tick handler | • Implement player movement<br>◦ Designed using engine-implemented KinematicBody2D movement.<br><br>• Create a general TileSet<br>◦ Created many layers, including props and ground tiles, to create a simple but aesthetically-pleasing environment<br><br>• Dynamic-visibility Props<br>◦ Created special signal-based nodes that control their transparency based upon player adjacency.<br><br>• Adapt Camera control<br>◦ Viewport control adapted to new format.<br><br>• Adapt Tick handler<br>◦ Tick handler changed nature to support real-time elements. Now attached to a non-root node.<br>◦ Utilizes Engine-level |

<table>
<tr>
<td>
as needed to support the new time-based system.<br><br>
• Implement a simple Heads-up-display<br>
◦ Display general information and some visual effects.<br><br>
• Create an item handler system<br>
◦ Used for entities to control their inventory and equipped items.<br>
◦ Highly extensible by design and heavily modular.<br><br>
• Implement weaponry<br>
◦ Create an item type for weapons, used to propel non-hitscan projectiles through the environment.<br><br>
• Implement Projectile class<br>
◦ To support the intentional modularity of weapons, implement a generalized projectile that can be easily adapted.<br><br>
• Implement Enemy AI<br>
◦ Create an enemy entity type and include intelligent movement and hostility.
</td>
<td>
time scaling to allow for easy time control.<br><br>
• Implement a simple Heads-up-display<br>
◦ Displays general information and even some visual effects.<br><br>
• Create an item handler system<br>
◦ Implemented using a state machine-like design pattern.<br>
◦ Highly adaptable and generalizable for any entity (by design).<br><br>
• Implement weaponry<br>
◦ Create a general ranged weapon class that is highly extensible for many different weapon types as desired.<br><br>
• Implement Projectile class<br>
◦ Successfully implemented and tested on multiple projectile types.<br><br>
<u>Incomplete Tasks</u><br><br>
• Implement Enemy AI<br>
◦ Unfortunately, with the time allotted, I was unable to implement the level of navigation control necessary for an Enemy AI, for example.
</td>
</tr>
</table>

# PROJECT OVERVIEW & DEMONSTRATION

The following is a link to a video file containing my video presentation for my developed system(s). In the video, I identify the core mechanics and gameplay elements of both versions of my projects.

Jayden Stearns – Final Project Video Presentation
[https://drive.google.com/file/d/1hW--UgsyQUvuDUxHGf9Zac-MjIiiUbRH/view?usp=sharing]

This video goes into further depth on the topics mentioned in this document, especially including the differences and similarities between versions of my project.

# REFLECTION

I have learned an incredible amount through this course, especially including the opportunity to work up-close on a semester-long project such as this one. As mentioned throughout this report, I truly recognize the importance of having a solid foundation upon which to build an idea. For years I have been enamored with the idea of a simultaneous turn-based system such as that implemented in the first version of my project. However, it required me to actual begin testing such a system to recognizing its glaring issues. Namely, the system simply was not enjoyable to use. Because games are rated by enjoyment, that functionally makes it a bad system, from my perspective. This experience is priceless, and improved my focus on what is truly important in the early life cycle of game development.

# CONCLUSION & FUTURE PLANS

This experience, while admittedly disappointing in some ways, was unquestionably important for me to progress as a game developer. Instead of being frustrated with my faulty foundation, I decided to change my direction. While this was undeniably difficult, I do not regret it, as I believe my end result is much better because of it. Additionally, I hope to continue working on this project, such as over Winter Break, and develop it even well beyond the scope of this course. This is something I could have only dreamed to have the knowledge and motivation for before completing this class.

Special Thanks to Professor Shaker and Kevin Jarrells for their support and encouragement throughout the entirety of this semester.