# Global Frog Games

# Sir Stanley's Well Rounded Adventure System Architecture

**Version 1.1**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 15/4/20 | 1.0 | Initial writeup | Brooke Smith |
| 3/5/20 | 1.1 | Final Edits | Brooke Smith, Nick Bonavia |
| | | | |
| | | | |

# Table of Contents

# Global Frog Games

# Iteration Plan

## 1. Introduction

### 1.1 Purpose

The purpose of this document is to explain our system architecture and explain the choices we made for our development environment and plans.

### 1.2 Definitions, Acronyms, and Abbreviations

See the glossary.

### 1.3 References

Team website:

http://riogrande.cs.tcu.edu/1920GlobalGameApp/index.html

Glossary

Vision Document:

Developers Guide

Software Development Plan

Installation and User Guide

Software Requirements Specification

Testing Plan

Github Repository:

https://github.com/tcuseniordesigncourse/globalgameapp

## 2. Architecture Choices

### 2.1 Game Engine Choice

At the start of this project, our team discussed several game engines as well some different non-game frameworks that we could use in development. Our team felt that Godot would be a great game engine for *Sir Stanley's Well Rounded Adventure* for several reasons:

1. It's completely free.
2. Great UI that is easy to use without any previous experience using a game engine
3. Can easily upload creations to Android devices (iPhones are not that much harder either)
4. GDScript, Godots python-like scripting language, was easy for us all to learn

5. Editor runs on most platforms

We decided that using a game engine for development would be extremely helpful for collaboration and efficiency. We decided against using non-game frameworks because we needed an easy way to decouple different parts of the game so that team members could work on multiple parts simultaneously without affecting the rest of the game. This, along with the fact that game engines are extremely helpful while organizing game files, was our reason for wanting to use a game engine. While Unity and UnrealEngine are both very powerful and widely-used game engines, we decided that they both would take a lot of time to learn and that Godot would be perfect for a smaller project, like ours.

### 2.2 2D vs 3D

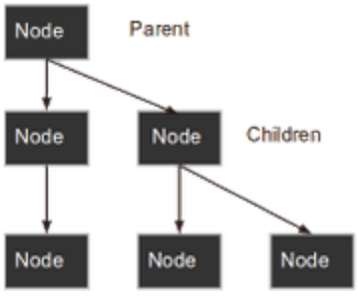
We decided to make *Sir Stanley's Well Rounded Adventure* a 2D game because of several reasons:

1. Most of our team is new to game development and working in a 2D plane was less intimidating than working in a 3D space.
2. We lacked art assets and 2D assets were easier for our team to create.

## 3. Our Game's Organization

### 3.1 Godot's Organization

**3.1.1 Scenes and Nodes:** Godot structures games into *Scenes*, which are made from *Nodes* in a tree structure. Each *scene* is a reusable component which can be instanced each time you need to use it somewhere else in your game. Each child node's position is relative to its parent node, and certain properties of children nodes can be inherited. For example *themes* are passed down to children and, by default, a child nodes *pause_mode* is inherited. The child's *theme* and *pause_mode* inherited properties can be overridden easily from the *Inspector* panel within the Godot editor or within the nodes attached script. Below is an example *scene* structure provided from the Godot website. It's important to note that the *scene tree* is similar in concept and use as the DOM in Javascript. Accessing *nodes* directly based on their path in the *scene tree* is not always the preferred method, since the structure of the *scene tree* might change throughout development or during runtime of the game. The preferred method should be the use of *signals*.




**3.1.2 Signals:** In Godot the *Signals* system is essentially an event system. There are many built-in *signals* in Godot that are specific to the various built-in *node* types. In addition to those built-in *signals* you can create your own *signals* that are unique to the classes you write.

### *3.2 Sir Stanley's Well Rounded Adventure*

**3.2.1 Scene Layout - Minigames:** Typical *minigame* layouts go as follows; *Node2d* with two *CanvasLayer* children and *AudioStreamPlayer* children for sound effects. One *CanvasLayer* child is used for game *nodes* and the other *CanvasLayer* is used for the *User Interface*. This separation allows us to properly sort the draw order of *scene* elements, i.e. draw *UI* on top of game graphics.