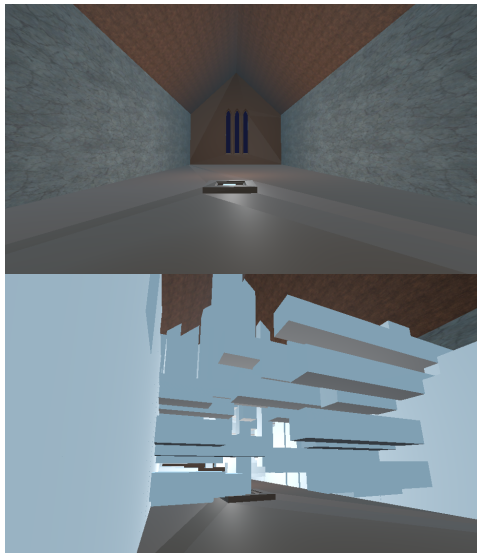
Game Engine Architecture: Environmental Reconfiguration

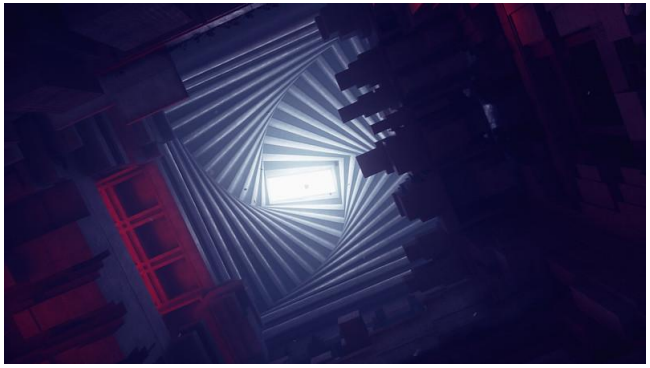


Charles Hollingworth
Department of Computer Sience and Creative Technologies
University of the West of England
Coldharbour Lane
Bristol, UK
charles2.hollingworth@live.uwe.ac.uk

Abstract
Reconfiguration of room using the Hiss style of the videogame Control focusing on simple usage and high degrees of freedom to level designers.

Author Keywords
Authors' choice; of terms; separated; by semi-colons
Optional section to be included in your final version, but strongly encouraged.

Introduction
The effect created is that of a room that appears normal, or empty. Which is then reconfigured by a large series of intersecting blocks. The blocks can be configured individually or as a set, Allowing for groups to move at different speeds, adjust to different lengths or change positions with the end goal of changing the entire space or to create an obstacle for the player. This effect was implemented within the Godot Engine and the Unity Engine.



Background & Research

The effect being replicated is from the video game Control we first take a look at their implementation of the effect, as seen in **Figure 1a** We can see the variability of the cubes and their random nature, to create an obstacle for the player. Comparatively in **Figure 1b** We can see the same effect being used to create a corridor for the player.

The effect is consistently made large blocks and changes the environment in a noticeable way, the blocks tend to move in groups in a sequential fashion.

Implementations

With both implementations in both the Godot Engine and the Unity engine the main features were,

- Hiss Blocks are grouped together, and each group activates at the same time.
- Each Block has a normalized vector which acts as the direction the block should move in global space.
- Each Block has a distance variable it will move in the normalized direction.
- Each block has a speed variable which determines how fast it will reach its target position.

While both implementations used a gameobject with the hiss cubes as children the code applied to each varied.

Within the Godot engine the code was shortest and used their inbuilt Tween function that interpolates

between two values on a gameobject. Whereas in the unity engine this code had to be created for the effect.

Evaluation

Both implementations were short and functional however there are a few aspects that could have been adjusted to allow designers more ease of use, for example automatic placement, or the blocks not being kept outside of map bounds.

When it comes to the preferred engine to create this effect Godot wins over slightly due to its feature rich scripting language allowing for simplifications such as the use of Tween unlike unity.

Another feature that could have been added would be texture stamping to allow designers faster creation of hiss groups. This feature would take the texture from the surface it passed through, unfortunately within both engines this process seems too difficult to achieve within the required time frame.

References

505 Games

Control

Juan Linietsky, Ariel Manzur and the Godot community https://docs.godotengine.org/en/stable/classes/class_tween.html