INTERACTIVE CINEMA

A Major Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In Partial Fulfillment

of the Requirements for the Degree of

BACHELOR OF SCIENCE

Ben Korza

PRELIMINARY REPORT

December 17, 2015

Professor Brian J. Moriarty, Advisor

---

2 BACKGROUND

2.1 The Pied Piper

(Begin this section with a part on the Piper's art direction)

2.1.2 Piper Movement

The Piper travels along a spline. In order to allow the Piper to stop and conduct other actions, special volumes exist that delay the Piper when he enters them. These delays end when either a set amount of time expires or a Boolean is flipped. The delay volumes are used at several set piece locations, such as when the Piper raises the bridge. The Piper also needs to teleport to another location at one point. This issue is circumvented by creating a second instance of the Piper at the target location. The combination of these techniques results in the Piper's movement system.

2.2 Constraining the Player to a Linear Path

Since the experience outlined in this project is almost strictly linear, it was necessary to have a way to limit the player's movement to a specified path. To accomplish this, the project employs several tricks that not only constrain the player, but also give them an illusion of freedom in order to enhance the project's experience goal. One trick was to use walls that would spring up out of the ground whenever the player went down an alleyway straying from the Piper's path. These would naturally block the player's movement and force them to turn around. To prevent this mechanic from becoming stale, each pop-up wall was outfitted with a different texture and special effect. These pop-up walls were placed at most of the locations where the player could get off track in the city. (Include a paragraph on Will's spatial trick and any other relevant tricks after this).

2.3 Flocking

One of the biggest requirements for this project was to have a crowd of children travel along with the player as they follow the Piper. The effect would require children to keep away from other nearby children, and do so without having a significant impact on the application's frame rate. In order to accomplish this an AI technique called flocking was introduced.

At its basic level, flocking works through the usage three simple steering behaviors: separation, alignment, and cohesion. Cohesion is the behavior that causes flockers to move toward the average position of their local flockmates. Alignment is the behavior that causes flockers to steer towards their local flockmates' heading. Separation is the behavior that causes flockers to avoid each other (Reynolds, n.d.). Combined, these behaviors form the basis of a general flocking technique.

In contrast to most flocking simulations, the children in this project move along a spline. This has some implications on the way their flocking can be conducted. Since the children's cohesion to the spline is the most crucial attribute of their movement, cohesive forces between children are out prioritized and therefore unnecessary. There is also no need for an alignment parameter since a child's alignment should match the direction of its velocity vector. The only behavior needed is separation. Therefore, the children are programmed to flock away from each other by first calculating vectors of separating forces. These vectors are then applied in two different ways. The first pushes children that are too close away from one another. The second involves offsetting their spline path using the forces calculated in flocking. The final behavior is a result of these two sub-behaviors combined. This resultant behavior is demonstrated in Figure 1, where all of the children maintain a distance of roughly several yards from each other.

Fig 1: Flocking in Action

In terms of performance, flocking can be a computationally expensive task. The technique requires each child to do checks against all their nearby children to determine how to apply forces. As such, the algorithm employs a number of adjustable parameters in order to improve performance. For instance, the maximum number of children that the flocking algorithm will consider and the rate at which the flocking algorithm updates can be modified. By changing these parameters, the algorithm's performance can be improved, but usually at the expense of accuracy. The correct balance between performance and accuracy is necessary to generate a convincing crowd of children.

2.4 Shaders

In order to achieve the project's experience goal, the player has to be enticed to follow the Piper. One way of doing this is by dissuading the player from straying too far from the Piper by desaturating the world. Implementing such an effect requires a shader, or a method of producing special effects via post-processing. Two different shader techniques were investigated

in order to determine how to best accomplish this effect. The first involves calculating the player's angle to the Piper and applying a screen space desaturation shader whenever the player is looking away from the Piper. This has the effect shown in Figure 2. It has the limitation that it can only be cast over the entire screen, and cannot be applied on a per-texture basis. However, the technique is easy to implement in Unreal and is not very expensive computationally.

Fig 2: Screen Space Shader Technique

The second technique uses node-based shaders attached to the material components of objects and is outlined in Reference 1. As seen in Figure 3, this technique has the advantage of being able to apply the desaturation effect both to objects individually and at different locations on their texture; however, it has the disadvantage of being computationally more expensive than the former solution. It also requires more work to integrate it into all of the materials using the effect.

Fig 3: Node Based Material Shader Technique

In the end the first solution was used because it was easier to implement, the effect was more desirable, and it was less computationally expensive than the second solution.

OR

In the end the second solution was used because it provided more control and flexibility over the desaturation effect than the first solution. This second solution was also used to implement several other effects in the application, like a rainbow bridge that materializes from thin air. (Complete once we know all circumstances where we use node based shaders).

2.5 Stat Tracking

Player metrics are both a valuable tool in the debugging process and a good way to determine faults that undermine the project's experience goal. In order to gather data on a player's run, the application records various statistics and outputs them to a log file. These statistics can then be viewed by opening the log file in a text editor. The statistics recorded in this log file are listed below:

- Playtime
- Time matching Piper direction
- Time looking at Piper
- Time standing still
- Number of popups triggered
- Number of hallway teleports
- Average frames per second

The log file also records a history of data collected in real-time. This real-time data includes the time and positions of the player when certain events occur, like the player stopping or triggering pop-ups. This real-time data in combination with the recorded statistics provides a fine granularity of data to identify bugs and assess how closely the player's experience matches the projects experience goal.

2.6 Wwise Integration

The project needed a flexible and optimized audio pipeline. To resolve this concern, Wwise was integrated into Unreal. Wwise is an interactive sound engine for games ("https://www.audiokinetic.com/products/wwise/," n.d.). Although Unreal Engine also has functions that handle audio, these do not allow for the same level of control as software dedicated to integrating audio like Wwise. The expense of using Wwise is the time it takes to integrate it into the engine. The software requires building a custom version of Unreal Engine from the source code found on their GitHub page

("https://github.com/audiokinetic/WwiseUE4Integration," n.d.). This is a considerable time sink, but it only lasts as long as the custom engine's build time. Once the build is complete, all the functionality of Wwise is easily accessible from within Unreal.

(Jake should pad out this section with more details on Wwise)

REFERENCES

1. H, O. (n.d.). 3D Modeling & Texturing. Retrieved December 6, 2015, from http://oliverm-h.blogspot.com/2014/08/ue4-localized-post-process-effects.html

2. (n.d.). Retrieved December 9, 2015, from https://github.com/audiokinetic/WwiseUE4Integration

3. Reynolds, C. (n.d.). Boids (Flocks, Herds, and Schools: A Distributed Behavioral Model). Retrieved December 9, 2015, from http://www.red3d.com/cwr/boids/

4. Wwise. (n.d.). Retrieved December 9, 2015, from https://www.audiokinetic.com/products/wwise/