**Metallic Detective Treasure Hunter Testing Plan**

Cameron Binda

Southern New Hampshire University

GAM 303: Design of Virtual Game Environments

Patrick Spencer

3/1/24

Runtime Errors

Runtime errors will either usually give a warning message in Unreal Engine, whether or not that crashes the game. Therefore, it will be simple to find and fix, as the warning message will direct you to exactly where the problem occurred. I will have that blueprint open on a different monitor, and watch the flow through the nodes. This will help trace the problem. If necessary, such as if the blueprint runs at the start of the game, I will use the built-in debugger. By adding breakpoints, I can go step-by-step to see where the problem occurs. Runtime errors may not always give an error message, so I will use steps outlined in Bugs to solve the issue. I constantly had problems with runtime errors that didn’t crash the game. One I remember is I tried to improve the metal detection system by making the light get brighter as you got closer. I tried to use tick events that calculate distance to decide the light’s brightness, but it gave infinite errors after one light was destroyed, due to accessing properties of a component that doesn’t exist. I could’ve used collision spheres, but I felt like that many wouldn’t have worked well, since there would be multiple per treasure, and it would’ve been a lot of gates for decision branching. When I upgrade the detection system so that it is auditory, I will need to test that the sounds stop when the ground is broken. I can foresee an issue where the sound will get stuck on when the actor is destroyed.

Program Logic

Based on my traceability matrix, I can split separate pieces of logic and build and test them one by one. Within each set of logic, I will code until I get to a point where there can be a result, test it, and either modify the code, or move on.. This stepwise technique is useful in all programming, as it prevents errors from piling up over the course of development. Each set of logic will be scrutinized against its summary of logic in the traceability matrix. A traceability matrix is a great way to break down development into easier steps. For example, I built the treasure actors to just be an item that I place and you can pick up. Then I made the breakable ground actor that breaks on interaction. Then I made it so when it breaks, it spawns the treasure actor. Finally, I made the treasure actor add a point to the player’s total on interaction. If I had done this all at once without testing all the individual pieces of logic, I would’ve had so many issues to fix at the end. I didn’t track all the problems I had, but there were a lot. One problem I had was that randomly spawned actors couldn’t be interacted with. I decided to try enabling interaction when the overlap event occurs, instead of on event begin play, and it fixed the problem. I got this idea because the enable interact node mentions that there is a queue. This is a great example that I should be more familiar with the nodes I use, and not just follow the tutorials.

Bugs

For bug testing, I will play through the game many times. Although this probably won’t be much of a problem, as it is limited what you can do, I will try to think out of the box and try to break the game. If I get a runtime error, it will be clear where to look, and some of these steps will be followed. Otherwise, when I notice a bug, I will pull up each relevant blueprint, and first just look it over. If I can’t find the problem, I will run the game with the blueprint open on another monitor, and try to trace as the code flows. If I still can’t figure it out, I will try bypassing certain nodes to see which ones cause the problem. This is really only possible for certain cases, so it may not always be used, and may not even help in those that it is possible. Another technique I will use is the built-in debugger to add breakpoints to step through the code. I feel like step-through only has specific use cases, usually in which many pieces of logic interact with each other, or when the code moves really fast in real time. This will be used when I see fit, and will always be a first consideration. One current bug is that you can be launched when the ground breaks under you, since it is a box that is bigger than its hole. For future testing, I will experiment with different hole sizes and box sizes on a separate landscape to see what shapes I need for it to not really explode. I have the explode amount on 0 already.