Edward Johnson

1. Problems Encountered:

a. **Discuss any problems encountered and how they were mitigated. Provide specific examples from your project.**

The problems that I encountered over the course of working on this prototype were primarily due to my inexperience with game development. This was both my first course in game development, as well as my first time using modern game development tools. I did run into several roadblocks, but I was able to get past them by rebuilding the implementations for the various mechanics that I implemented into this prototype. I had planned to also spend some time learning blender so that I could create the player character model and enemy guard model, but ended up running out of time and didn't want to leave the game in too much of a messy state. This was probably the first problem I ran into, and I left it for last so that I could work on features in the game that would have a greater impact on the player's ability to play the game. The environment design and structures of the levels were likely the most time consuming process I faced while developing this project, and this was primarily due to the need to add collision to everything. The majority of the static meshes that I created ended up needing to have collision created / mapped / tweaked by hand, and I found the process fairly tedious.

The next implementation related problem that I ran into was with my first attempt to implement the hidden mechanic into my game. I started off by creating the hidden state, as well as all of the accompanying variables and functions to trigger it. This included a ‘can hide’ boolean that is triggered once the player character moves into an overlap area where they should be able to hide. The process then would check if the player is hidden, and if they were not and they were pressing the correct key bind, then it would set the player state to hidden. My initial implementation I was trying to use an

external cue in the form of an arrow above the spot that would change material / color if the player triggered hide within that spot. I was unable to get this implementation to work properly so it took some extra time to think on it, and look for a better approach. What I ended up later settling on, then getting working, was to have the player's collision and visibility be changed along with the state change from ‘isHidden’ being false, to becoming true. I also implemented a crouching mechanic, and tied the crouch key bind of 'C' or 'Control', into my hide ability. I used a dark cylinder on the ground to give a visual indicator of the areas where a player could hide, and it ended up working out really well in my final version. I did add the ability to track which spot the player was hidden at, but ended up not implementing any other mechanics that relied on that location knowledge.

Another problematic area that I faced was in getting collision to function correctly and using line traces to do so. I was having a problem where the enemy guard patrols would catch the player even if the player was standing behind hard cover (one that included collision). I ended up having to dig into why we use line traces on overlaps to make sure that the conditions to trigger an event are actually in the position where the event should be triggered. This included adjusting the collision visibility as well within the enemy guard and player character blueprints. I also was initially having this problem with my security camera system, as hard cover / walls with collision were not stopping the camera from catching a player. I did test that both the enemy guard, and the security cameras do not catch the player while they are on a hiding spot with the crouch button pressed, but the updated versions included not being visible / behind a collision object as cover from triggering death. The hide ability avoids this due to the fact that the player has collision disabled for the duration of their key press, in addition to them being no longer visible.

The last major headache that I ran into while working on this project was implementing my valuable goods system. I started off by duplicating my base blueprints that I had created for my locked door and destructible key pickup system. I very soon realized that I was going to need to recreate everything from scratch though, as the two interfaces that I implemented for the key / door system would make a lot more sense if I created a separate version for my valuables and objective pickups. I used the same pickup / interaction system from them, but changed the interface to implement a grab loot function. I implemented a blueprint for the loot pickup, which is the actual item that the player would grab. I then tried to implement 2 different value objects which could be used to pair to the valuable pickup in the same way my key pickup could be to the door it went to. The problem I ran into though was that I would have had to implement a new valuable object for every different pairing. So instead, I created a generic parent class value blueprint, and then created child blueprints from that parent for my single valuable good item. Instead of needing to redefine the object that it would reference between different value objects, I had it reference the base parent blueprint. This allowed me to then pair the valuable good to different value amounts. I was able to then use these pickups to increase the score count for the player when they pick up one of the objects.

2. Successes in the Development Process:

a. **Discuss what went well and how future developers could replicate it. Provide specific examples from your project.**

There were a number of things that I think went well over the course of this project. I think most of my success in building this project comes down to my strategy and approach to doing work. I tend to lean toward using a divide and conquer based approach to doing work, and this approach works

especially well when it comes to programming. I think that if I had one bit of advice to other newer developers, it would just be to not get stuck on a single problem. Attention to detail is obviously important to the goal of producing a quality end-product, but not when that attention to detail causes you to stagnate and not get anything accomplished. I don't think I am alone in stating that working through other, closely connected problems can many times give you context or insights that allow you to loop back and readjust areas where you were previously stuck. The main examples of this for me at least were in figuring out how the various collision settings needed to be done so that the enemy mechanics were correctly respecting hard boundaries, or boundaries with collision enabled. It wasn't until I finished my security cameras and lasers that I realized how the line trace by channel function works in the game environment. I had already implemented my enemy guard models, and just finished the core of my security cameras, but both were having issues with catching the player through walls. When I was watching a tutorial video about how to create lasers, he briefly covered the reason for using a combination of collision, and a line trace to prevent the lasers from penetrating walls. I was then able to recreate this process to fix / finish the implementations of my guards and security cameras. A saying that comes to mind is, "to a hammer, everything is a nail". This saying is referring to the law of the instrument, or Maslow's hammer. This is a cognitive bias in which we over rely on familiar tools. I think that this is obviously true for more than developers, but if we hit a roadblock where the tools that we are familiar with aren't working, it may be time to branch out and learn new / more tools. If you are stuck, move on to a new but related problem / feature. Taking a break will both allow your mind to better process information regarding how to approach the prior problem, and so long as you are regularly exposing yourself to the methods / strategies of others, you will likely end up finding a new tool (or combination of tools) to help you fix your problem

3. Lessons Learned:

a. **Discuss overall lessons learned about rapid prototyping. Provide specific examples from your project.**

While working through this project, I learned a tremendous amount about game design through rapid prototyping. The majority of what I learned was related to the approach to work, and what is necessary to create a game within Unreal Engine 5. Having a deep level of exposure to Unreal Engine in the way that I did this semester, opened my understanding greatly of what it would be like to be a game developer. I really do believe that rapid prototyping is a strong tool for development. The way that I developed this game project via rapid prototype, also greatly reflects how I write code. What I mean by this is that generally when I am writing code, I start off by noting what the software will need to achieve in the way of value. Once I identify these values, I then outline and decompose them into segments that I feel would be small and modular enough to make up functions. And once I get to the point where I have a general idea of what the structure of my program will look like. I stop planning, and I start writing code. I don't focus on getting everything perfect, I focus on filling out the majority of the functions so that I can get a better idea of what will work, and what will need to be redesigned or implemented differently. Most if not all of my initial code will be rewritten, but having the application to look at gives me an opportunity to better see the areas that might need to be broken down into isolated functions, or that might need to use a different implementation approach. I was able to use the same approach that I use in coding for this prototyping project. I started off the semester by writing down all of the major features that I thought a game in the category of "stealthy" would include. Over the past 2-3 weeks, I then spent all of my free time working on implementing those features and mechanics. If I hit a roadblock implementing a feature, as I did with the hiding system and with

implementing enemies, I moved on and used my next block of free time to get as far into implementing another feature as possible. This helped keep me on track, making steady progress, and before I really realized how much I had done, most of the prototype was finished. By the time I looped back around to working on the features that had earlier been giving me trouble, I realized that a great deal of the work that I had done could be broken apart and used to implement features that I was stuck on. This was because during the process of implementing other features, I ended up learn the functionality of many of the tools that Unreal Engine provides its developers. In this early stage, seeing how the various tools within Unreal Engine work in action gave me enough experience to then use those tools in a better/different way.

I really enjoyed the opportunity I had to work on the rapid prototype for my game level this semester. I really feel even more strongly now that this is a great approach to development and problem solving. I think that there is definitely a certain amount of preplanning that should be done prior to any development work taking place. That being said, once the "vision of value" so to speak has be developed, I think the best way to move a project such as this forward toward reaching its value goals is through rapid prototyping.

4. Task Log:

a. **Explain how you refactored your project to maintain a feasible scope in the time allotted, using specific examples from your task log.**

In regards to refactoring my project, most of my refactorings were to change implementations to use approaches that would create a better, smoother implementation. I did not have much in the way of refactoring to maintain a feasible project scope, given the time we had to work on the project. I did not initially expect to get as far as I did into the project, and I did end up getting the opportunity to implement all of the major features into my rapid prototype game level. The primary things that I did not get a chance to implement, and something that I will have to end up teaching myself later, is how to create character models as well as how to animate them. I did include all of the character animations and models that I would technically need to implement for a full game, but I have no experience creating them, and the few short hours I spent learning blender did not yield me anything that was usable for my project. I did end up refactoring my game to remove the time limit that I had initially thought to implement, as the more I worked on the level, the more I felt that it did not fit into the play style that the level would impose on the player. The level is designed in a way to get the player to pay attention, and to take their time. There are plenty of safe spots, but the level does get more aggressive in the amount of obstacles thrown at you as the player progresses towards the end.

b. **Explain how you used a balanced approach to each project task to maintain a feasible scope, ensuring all tasks were relatively equally developed, using specific examples from your task log.**

In building this rapid game prototype, I used a balanced approach to build and develop all of my game features. I did this in the way that I mention in section 3, through creating the general structure of the functions, blueprints and classes, and not overly worrying about perfecting the implementations until I had the general systems that it fit into implemented. I did this for all of the related systems, and it ended up working out really well. The past few days I have been mostly finishing up and tweaking the various system implementations, such as with fixing the collision issues I was having with the guards when the player is not visible to the guard. This balanced approach provided me with the details that I needed to implement my heads up display (HUD) for the player that tracks objective statuses, as well as the score a player has achieved. If I had spent all of my time focusing on trying to get a single system implemented, I likely would not have had the time to implement as much as I did. By forcing myself to move on to a new problem / implementation when I hit a real block, I kept my development process moving forward. And as I already mentioned, this also provided me with better context and understanding of the tools, and allowed me to quickly correct issues that had previously been significant obstacles. The most time consuming portions of this project were in implementing the Enemy guards as well as their behaviors, with the loot value system, the hiding / crouch system, and then most significantly, in the modeling and building out of the level.