**Evaluation of the development process**

**For Island Adventure 9 A True Beginning.**

**Tim Allen, Ryan Dietrich, Wesley Lavassaur, Connor Lindsey.**

**Evaluation of the Development Process**

**i. Clarity of our own requirements for the game**

**What went well:**

The features that were planned for the game were clearly understood and achievable. The game design document gave us a clear roadmap to create the game and subsequent features.

**What didn’t go well:**

The scope of our feature set was much larger than we realized and it was difficult to implement in the time that we had. The size of the project was larger than we anticipated and difficulties with tools and bug impeded our progress heavily. Lack of knowledge about key tools such as Git in relation to Unity and unit test for Unity also dampened our progress.

**ii. Team collaboration**

**What went well:**

When we worked together in person things were much more efficient. We were able to work well as a team and communicate quickly and efficiently. Communication was clear and concise through platforms such as a group chat and HacknPlan.

Also, whenever anybody had an idea, comment, or otherwise, everybody listened attentively and we could integrate our ideas together into one cohesive objective to move toward. The game feels like everyone's game and not just one person's idea that everyone else just worked on. We were almost always in constant contact with each other which made this a more efficient process.

**What didn’t go well:**

At times, constant communication with the entire team wasn’t possible which hampered our ability to work efficiently.. Changing directions and expectations for the game with the team was hard to solidify throughout the term. A lot of things were up in the air for most of development which made working on things hard as we didn’t know if they would be needed down the road. This could have been remedied had we worked as more of a team rather than assigning one major part to one person or another.

Our expectations for the game mirrored our expectations for what we thought everybody was able to accomplish. Sadly, it is difficult to keep expectations realistic in the middle of game development. There were key systems that were missing for large portions of the term that should have been adapted for very early on.

**iii. Designing the software**

**What went well:**

All of our brainstorming sessions went really well and always ended up with a solid for the game to go and what would be best to work on next. Hands down the most efficient part of the process was the actual creation of the game. We were able to quickly identify bugs and future issues and adapt to them. Besides Git issues, implementing any feature anyone was working on into the main game was successful with rare exceptions.

**What didn’t go well:**

It was absolutely nightmarish when we ran into problems with Git and how it works with Unity. We learned a little bit about how to commit and push changes from a branch to master, but we were not taught how to handle hundreds of merge conflicts in a single commit, or how to prevent or stop git from tracking all of the meta files which are created when Unity opens a project, but which do not change anything in the game. As such, we have had many commits that are only meta file changes because we could not discard them without losing the actual changes we wished to implement.

It should be clearly stated that the way Unity generates files and stores data is a major fault in our VCS pipeline. It was the creation of untracked meta data from the beginning of the project that hampered our progress. Without a stable base with which to create from we were constantly playing catch up with github and had we had more experience with the system this would have been fine.

**iv. Project management**

**What went well:**

HacknPlan was used frequently during the first two weeks of the project and the last two weeks. Many detailed user stories and tasks were added to facilitate easy task distribution between team members. Each task added was thorough and told the person assigned everything needed to complete that task and provided detailed descriptions. Time frames to complete at least the last two weeks of the project were accurate. When we weren’t using HacknPlan, when we would meet in class, we would decide what each person would work on going forward. So everybody usually had a loose task they were working on.

**What didn’t go well:**

We failed to consistently use HacknPlan throughout the term. After the first two week sprint was over, we did not add any more information or tasks for another sprint. I think this may have aided our disorganization and decreased productivity during the next 7 or so weeks of the term. Not having clear cut and realistic sprints designed surely made the team rather directionless, or at least confused as to exactly what needed to be done by when. Although we had a basic idea, it would have aided us to sit down and decide what to cut earlier and have more frequent meetings to plan what was the most urgent feature to work on next.

**Evaluation of the Finished Product**

**i. Assessment of the completeness and correctness of your system**

**1. Is the software done (and how do you know what “done” means)?**

The truth with game development is that a project is never done. There is always another feature that needs to be added to give that extra shine to the project. For this project though, our minimum viable product is present and the majority of systems have been created and accounted for.

**2. Did it turn out the way you wanted it to?**

While the game has not turned out as we had originally intended, we have ended in a place kind of close to our original design. We moved from a third person to a first person perspective, and the pirates became cats, but for the most part our original vision is displayed here. Our disappointment is in the quality and polish of the project, as we were dealing with Git problems and individual time management for most of our development time.

**ii. Assessment of the quality of your design**

**1. Were there any features that were especially hard to implement because of the design?**

Our design was made with implementation in mind, so we did not plan to add many features that would have been too difficult to implement with the other features. The character controller in our design was more than we could complete in the time constraints so we switched gears to a first person controller, rather than third person with target locking and multiple attack directions and locations.

**2. Is this design scalable (will it be easy to add features)?**

With our current design, it would be very simple to add on more features, such as crafting, different weapons, new enemies, etc. For example, the enemy controller script could be applied to any melee fighting enemies. A couple small changes would allow it to be used for ranged combat, as well as npcs that are neutral. Adding new items would only require adding new tags and what the items are used for. As the game isn’t multiplayer, the scaling would not need to take that into account.

**3. Is the design easy to understand and communicate to others?**

The survival and overall game progression systems are fairly easy to understand. Gather enough of the right resources to build a ship to leave the island you’re on, and keep doing that until you reach the last Island. Survival is simply collecting mushrooms, coconuts, rum, and drinking water to maintain your hunger and thirst levels. The depletion of these depletes health. Enemies also deplete health.

**4. Does this design result in sufficient performance.**

We allowed for many different resolution sizes and scales, and kept the poly count low enough on our models that it would run smoothly on most computers that are less than ten years old. There are no events in the game that result in an operation that ever crashes the game. Gameplay is smooth and is optimized for lag free gaming. There are few polish elements that make transitions in the game visually look like performance is insufficient, but this would not result in an application crash. Were we to add simultaneous users, the game would be able to handle them to a degree. After about 8 or so additional players it would begin to become laborious for mid level hardware. If we were to implement all the features and enemies we had planned, the performance would have been tricky to keep smooth. But since our game is fairly simple, this was not an issue.

**5. In retrospect, would a different design have been better?**

I don't think a different design would have worked or played better than our current design. We put a lot of thought into what we did, and adjusted things as we went along to create the best design that we could in the time permitted with our collective skill sets. A moe simple design would have resulted in less stress and adjusting throughout the project. This would have allowed us more time to polish the game. So in this way, a simpler design would have benefited us but it would have been a trade off.

**iii. Assessment of the implementation of your system**

**1. Will the code be easy to maintain?**

In its current state the code is well documented and commented throughout with a few exceptions.

**a. Is it easy to fix bugs?**

The bugs that we have found have all been fixed with relative ease. We have not run into any bugs in the game that have been problematic to fix, and we did not leave any bugs in the code after realizing that they existed.

**b. Will it be easy to add new features to the system (how do you know)?**

It will be very simple to add more features. We built a base system, and have been adding features to that system since the start of the project. Each feature we have added has been added with minimal issues, and each of our features is independent of other features, and can reference other scripts as necessary, but not change them. Because of this, adding another feature, as long as it is not directly conflicting something in the existing code, would be a very simple matter. In other words what we have created is very modular.

**c. To what degree did you follow accepted coding practices (separation of concerns, DRY code, no “dead” code, minimal dependencies)? Give examples.**

To reduce DRY code, we used for loops when we could to access multiple instances of objects and children of object within a scene whenever we wanted to change properties of multiple objects at once. To further reduce dry code, we used variables in situations where we had previously hard coded situations and numbers so our code to be applicable to many situations. Plenty of times, I (Wesley) would show my code to my members and they would see ways I could massively simplify the code to achieve the same result.

**2. Is the code well documented internally?**

Yes. We have commented the code very thoroughly and explained what each section of code is supposed to do. If the section of code is more complex, a longer comment was used to break it down into understandable chunks.

a. Descriptive names for classes, methods, and variables

b. Comments where needed to explain what your code is supposed to do

c. Are there unit tests that make the purpose of methods explicit?

In order to test our variables and ensure that there were no bugs in the code, we made frequent use of the debug.log command, which gives an output to the console. While this is atypical for programming majors, this is the only method of unit testing that we had learned up until this course for game design. It does not make sense to create a method that tests if the player moves forward when you press the button you wish to move forward with, as you can play the game and discover that when you press ‘w’, you move forward. When we used arrays to keep track of information such as the current island that the player was on, we would use debug logs to ensure that the numbers were behaving how we wanted them to, and that we did not get any out of bounds errors or have any problems with the variables changing when we did not want them to.

**3. Will your code run fast (for example, will your database support a heavy load of users, or will large numbers of moving objects be rendered smoothly)?**

Our game runs smoothly on low quality computers without any issues. Our code runs efficiently, and we also did our best to keep poly count low and use lower resolution textures in order to prevent lagging and other issues due to unnecessary stress on the gpu.

**4. Is your code efficient (did you accomplish your goal without writing a lot of extra code)?**

Much of our code was inherited from other scripts and we copied code when we needed. In our PickUp script on our player controller, we did seemingly write a lot of extra code that could have possibly been simplified. But most of what we wrote was only meant for that specific script. When we saw code that was similar to other code in our project, we simply copied it and re used scripts when we could. Whenever we could, we would try and use the same script for as many things as possible.

**5. How bug-free is your code (are there known bugs, does it crash unexpectedly, or does it do the wrong thing at random times)?**

Any bugs we found were fixed before we committed each time, so it is bug free as far as we know. That said, there may be bugs that neither we or the compilers did not catch.