Software Engineering Group 27 Report  
Written by Wu Tong  
Reviewed by all members of Group 27  
30.04.2024  
This report serves as a brief summary of our group's accomplishments, challenges, and  
lessons learned over the past 12 weeks while developing “” for Watson Games.  
Key Achievements  
Meeting Project Objectives  
We take immense pride in the fact that we were able to develop and integrate so many  
game features within the given timeframe, with ~94% of the requirements outlined in our  
analysis (System, User & Domain) getting successfully implemented.  
Team Chemistry  
In our project team, each member is like a unique element, because our interactions resemble an intriguing chemical reaction. Firstly, our leaders act as catalysts, because they can promote teamwork and accelerate our progress. Their influence unlocks the team's potential, enabling us to achieve our goals more rapidly.

Next, we have creative members who function as our reactants, because they continually provide new ideas and innovations, injecting vitality and novelty into the project. Their contributions ensure that our work is always fresh and dynamic.

Communication serves as our solvent, because it binds our team together more closely. Through effective communication, we can better understand each other and collaborate more effectively, thereby achieving our common objectives.

Lastly, team spirit is the most crucial factor in our project team, because it is the heart and driving force of our team. It is this team spirit that enables us to overcome challenges, work together, and ultimately succeed.

What We Learned

Learning Unity

In this project development, our team is facing unprecedented challenges, especially because we have never been exposed to a development platform like Unity before. First of all, we need to spend a lot of time and energy to learn and master the basic knowledge and skills of Unity. This includes understanding the workflow of Unity, learning to use Unity Editor. Because our team has no previous experience, we may encounter many technical and design difficulties in the early stage of project development. We may encounter problems, such as how to correctly organize the project structure, how to achieve specific functions or effects, and how to solve common mistakes and problems in Unity. In addition, because members of our team may not have shared Unity development experience, communication and collaboration may also face challenges. Team members may have differences in understanding and implementing specific functions, resulting in poor communication and difficulty in collaboration. However, despite these difficulties and challenges, our team is full of confidence and determination. We realize that learning and growth is a gradual process, not an overnight thing. We are willing to work together, overcome difficulties, support and encourage each other, and finally achieve our project goals. Through team cooperation and efforts, we believe that we can overcome any difficulties and achieve success.

Learning C#

Considering that some members of the team have a weak foundation in programming, we decided to avoid using too many advanced concepts involving events and delegations in project development. On the contrary, we use the more basic callback function as the main programming means. Through the callback function, we can call a specific function or code block when necessary to realize the required function, without too much involvement in the subscription and trigger of events and the definition and transmission of delegates. There are several reasons for this choice. First of all, the programming method based on callback function is more intuitive and easy to understand, and even members with less programming experience can get started quickly. Secondly, the callback function can help us avoid some conceptual complexities, such as the usage and understanding of events and delegates, thus reducing the learning burden of team members. Although we choose a relatively simple programming method, we also realize that this is not the final solution. In the subsequent stage of the project, we plan to gradually introduce more advanced concepts, such as events and delegates, and other more complex programming techniques. We will make use of the actual needs of the project and the learning progress of team members to gradually explore these concepts and apply them in practice to improve the overall programming ability of the team and the quality of the project. In this way, we believe that the team will continue to grow and progress in the project development.

Documentation Guide  
The project's documentation is predominantly stored within a Notion workspace, which we  
used ourselves throughout the project for tracking and documenting progress.  
To access the workspace, you can use this link to join.  
Upon entry, all essential information can be found on the "Navigation & Marking Guide"  
page.  
Our Notion workspace is where the majority of our documentation is located, it’s essential  
that it is used during the marking process.  
We highly advise you to use this navigation & marking guide page while marking, as it  
shows (as best as possible) where we believe you can find evidence of our marks, with easy  
to follow links to different parts of our documentation, and any external sites.  
What Went Well

Meeting   
Our team holds a meeting at least once a week, and the meeting atmosphere is good. Everyone is free to express ideas, discuss project progress and solve problems. We pay attention to teamwork and joint decision-making, and this open atmosphere enhances team cohesion. The meeting not only pays attention to the project, but also cares about the working status and life challenges of team members. Through these meetings, we maintained good communication and cooperation, and improved team efficiency. We will continue to maintain a positive attitude and work together towards success..

UI Design

Our UI design focuses on simplicity and ease of use, maintaining consistency and aesthetics. The interface is intuitive and clear, and users can easily find the required functions. At the same time, we adopt responsive design to ensure good performance on different devices. User feedback mechanism has also been paid attention to, so that users can clearly understand the operation and status.

UI Design

Our team pays attention to the readability and maintainability of the code. We annotate each function, class and key variable in detail, so that team members can easily understand and use the code. Adopt a clear naming convention, embody that characteristics of object-oriented programming, and make the code more readable. At the same time, we design the code modularly, which reduces the complexity and is convenient for maintenance and expansion. Follow strict code specifications, unify code style, and strengthen teamwork. Finally, use version control tools to manage the code to ensure security and stability. These measures make our code readable and maintainable, which is helpful to develop and maintain the project efficiently.  
What Didn’t Go Well  
Uploading initial Data  
One of the main user requirements which we were unable to achieve was the ability for  
players to upload their own data to the game.  
Our interpretation of this requirement was to include logic which allows the player to upload  
their own textures for development cards, resource cards, and hex tiles.  
We considered various approaches to incorporate this, but ultimately found that the amount  
of time and resources it would require outweighed the potential benefits. Because of this, we  
chose to prioritise completing the game’s core functionality.  
A fully detailed list of items that haven’t been implemented can be found on our notion  
workspace’s missing features page.  
Testing  
One issue that we encountered during our project was the difficulty in testing our code,  
particularly with UE5 Blueprints. We initially considered using C++ to use Unreal’s built-in  
testing tools, but found that it would require at least some of us to learn C++ and how it ties  
into Unreal Engine and our existing blueprints. As C++ is known to be a difficult language to  
learn, we decided against it.  
The testing process was also challenging due to the presence of ‘impure functions’ in our  
code, which had various side effects, like changing values of other classes and changing  
game state values.  
To overcome this, we had to either create dummy objects and hook them together to unit test  
a function, or run through the basic gameplay systems to integration test the game. We have  
examples of both of these in our code, and it was a time-consuming process that slowed  
down our development progress.  
It’s worth noting that if our functions had been a simple input-output system (i.e., pure  
functions) with no side effects, unit testing would’ve been much more straightforward, even  
with the limitations that blueprints have with testing.  
Blueprints & Git Interaction  
One major issue that we had throughout the project had to do with our use of Unreal Engine  
5’s blueprint system, and how it interacts with source control and GitHub. We found that the  
way that Blueprint files are stored (as binary files, rather than plain text like other languages)  
made it impossible to easily track changes and merge git branches that our team was  
working on. This meant that we had to manually merge all of our code, which was Dan FH's  
responsibility for the entire project.  
To mitigate some confusion for Dan FH while merging, team members would go into extreme  
detail about the changes they made with each pull request.  
However, the task of merging was still much more difficult than letting GitHub manually  
merge code for us.  
As we created more blueprints, functions, and variables for our game, this job became more  
and more complex. For example, our game state refactor, which involved many changes to  
every class to fix UI visibility issues, took Dan FH around 5 hours to merge.  
We eventually found a solution in a UE5 plugin called "Graph Printer", which allowed us to  
relatively easily export and import code using high-resolution screenshots.  
Including these within our pull requests made merging code as 'easy' as downloading the  
image. This method still had its drawbacks though. Notably, variables and function names  
still needed to be manually created. Nevertheless, it still made the job of merging code  
easier than copying and pasting absolutely everything.  
What would we have done differently?  
In retrospect, using a framework or language that the team was all previously familiar with  
would have been a good decision. It would’ve allowed us to focus more on the project’s  
creative aspects rather than struggling with UE5’s technical limitations. A familiar language  
would’ve most likely given us a more seamless version control experience, and would’ve  
improved the project’s efficiency by allowing us to start planning and implementing more  
complex features earlier in the project’s timeframe, rather than learning how to use UE5.  
More thorough research to understand the limitations of the tools we used before starting the  
project would have helped us better anticipate and plan for potential issues, particularly  
concerning how version control interacted with blueprints, and how testing could’ve been  
implemented better. A different engine or language that works better with version control and

Software Engineering (G6046)  
testing could’ve been selected if we took a bit more time researching our development tools.  
Additionally, better distributing the workload of merging our code may have allowed Dan FH  
(who has professional experience with Unreal Engine) to focus on implementing more  
features, instead of spending time on the manual merging process.  
Finally, we think that, had we been able to solely focus on this project without having to  
juggle our other module assignments and exams, we would have been able to produce a  
polished product that meets all of Quentin’s requirements.