University of Glasgow | School of Computing Science

Level 3 Project Case Study Dissertation

SH36 - Project Gundam

Sujay Patil
Cameron Fraser
Fergus Young
Luke Ormiston
Nik Harith Sharifuddin Mohd Suhaimi
Jan Dulmant

3rd April 2024

Abstract

This paper documents the process of creating *Project Gundam*. A mobile app utilising Augmented Reality (AR) technology to visualise and comprehend the massive scale of Gundams - Japanese anime robots - in the real world. This case study explores the challenges we faced as a team, such as software and operating system incompatibilities; Unity and Git integration; and navigating unfamiliar technologies. We document how these challenges were resolved and the consequent lessons learned.

In sharing our experiences and resultant insights, we hope to inspire those wishing to develop similar augmented reality projects. Derived from our experiences, we aim to provide advice on resolving common Unity issues, adapting to adverse challenges, and successfully delivering an alternate reality project to a given customer.

1 Introduction

Project Gundam is an Android app that allows users to view Gundam models superimposed in the real world through their phones. Gundams are fictional giant robots manned or AI controlled, pictured in Japanese animations under the animation series *"Mobile Suit Gundam"*.

We developed this project for the University of Glasgow Games and Gaming Lab to deploy during WorldCon 2024, a science fiction convention. It will encourage attendees of the convention to explore Glasgow while guided by these Gundams with gamified elements including a Pokémon GO inspired collection system and informative quizzes relating to each Gundam model. In its current state, the app will take the users on a tour of the Scottish Event Campus, Kelvingrove Art Galleries, and around the University of Glasgow's main campus.

Project Gundam introduced the use of new technologies for the whole team such as Unity, Google's Geospatial API, and C#. Throughout this dissertation, we explore the challenges we faced from slow internet to software incompatibilities. We explain workarounds we found and how we would face these challenges in future.

We begin by discussing the project in more detail, before exploring how we started the project. We talk at length about the technical challenges we faced, leading us into using Git with Unity and how we handled customer communication. Next, we discuss our approaches to issues, branching strategy, and quality assurance. Finally, we explore the challenges faced with setting up the pipeline, ending with how we transferred the project to the customer.

2 Project Background

The World Science Fiction Convention (WorldCon) [1] is a science fiction convention hosted at a different venue globally each year. WorldCon 2024 is hosted in Glasgow and marks the 82nd WorldCon Event. The event provides a large range of stalls, panels and, famous guests - attracting thousands of fans and creators each year.

Our customer, Timothy Peacock [2], aimed to capitalise on the influx of Gundam fans brought into Glasgow by the convention. Tim wished to create an augmented reality app providing a Gundam themed sci-fi experience for attendees of WorldCon. He envisioned the app would encourage attendees to venture away from *Scottish Event Campus* and visit the University of Glasgow. The app provides rendered life-sized Gundams for attendees to find and Gundam trivia quizzes to answer. During WorldCon, the app will attract more visitors towards the university *Advanced Research Centre* where the UofGGamesLab [3] will be hosting various sci-fi based events.

On first launch of the app, the title screen is displayed (Fig. 1). Upon clicking "Start" the user is met with a tutorial for the game (Fig. 2). Afterwards the user is prompted to enter their chosen username (Fig. 3) and acknowledge a safety prompt (Fig. 4). Following this, the user reaches the main gameplay loop. Here, their objective is to explore Glasgow, discovering all Gundam models, whilst utilising the compass for guidance. Once in a Gundam location, the model will render in life-size and equipped with animations. The user will be able to view the Gundam's description and complete its associated quiz (Fig. 5).

Upon completion of the quiz, the Gundam will be *found* and the compass will guide the player to the closest un-found Gundam. The *Gundam Tracker* (Fig. 6) displays the *found* status of each Gundam with respect to the user - all un-found Gundams are blacked out. Subsequent launches of the app skips over the user guide but the safety prompt is displayed every time - reminding the user to stay safe.

Fig. 1: Title Screen

Fig. 2: Guide

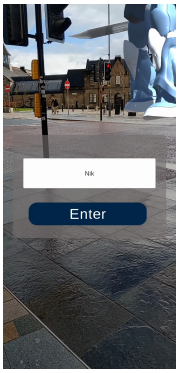

Fig. 3: Name Entry

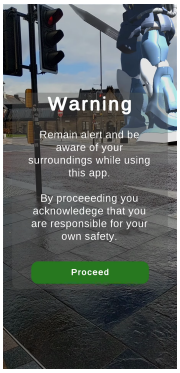

Fig. 4: Warning Popup

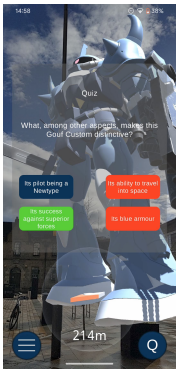

Fig. 5: Quiz Screen

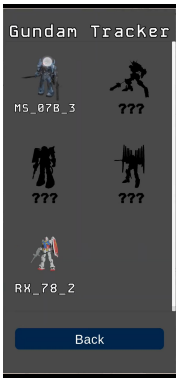

Fig. 6: Gundam Tracker

3 Development Overview

In the interest of creating a sophisticated app using established technologies, the team was required to upskill in unfamiliar tools. Therefore, it was essential that our initial customer meeting defined a clear set of requirements for the app. During this, we conducted a MoSCoW analysis which outlined the *Must, Should, Could and Won't have* requirements for the project[4]. This was beneficial as we could definitively prioritise project tasks without needing to understand the implementation details - most of which we were unfamiliar with.

The minimum viable product, outlined in our MoSCoW's "Must haves", was to display an AR Model at the University's *Advanced Research Centre* using a mobile device. Fortunately, the minimum viable product was accomplished early during development which meant the team could shift their focus to the "Should have" requirements.

These "Should have" requirements included gamification elements, basic accessibility features and tangible player progress. The majority of development time was spent on these. Following this, the "Could have" requirements or stretch goals of the project included a summary of previously visited Gundams; the ability to add additional models after deploy-

ment; and the possibility of adding whole new experiences to the app outwith the Gundam theme. It was also paramount to the team that we defined an upper limit to what the app would achieve. Through the "Wont have" requirements, our customer outlined that the app won't cost users anything; models will not be reused and it would not contain a model creation tool - instead popular third-party tools would be used.

Alongside the MoSCoW requirements, three User Stories were established with the Customer to contextualise the app requirements. We used the "As a... I want to... So that I can..." story structure to clearly identify different types of users, their use cases and their motivations when using the app. These stories were beneficial, throughout development, as they held the team accountable for creating an app different users can all benefit from.

As a team, we identified that selecting the correct platform for development was vital and that the wrong choice could have cascading effects. Each team member spent a morning independently researching possible platforms then the team convened in the afternoon to discuss their findings. Contenders included Unreal Engine thanks to its tools for mobile development or Snapchat Filters because of its beginner-friendly support design and support for AR and location-based events. Ultimately, Unity triumphed as our platform of choice. Each member independently recognised its capabilities and its prominence within the industry - highlighting its position as the standard for AR mobile app development.

Unity has many software development kits (SDKs) available with some catered towards specific platforms. Through our research, we opted for the *AR Foundation*[5] package which interfaces intermediate SDKs enabling development and deployment for Android, iOS and many other platforms. Further, Google's Geospatial API is compatible with *AR Foundation*, enabling 3D models to be placed at real-world GPS coordinates which is vital for our project.

We considered completing an additional course for AR within Unity, but from our experience with Codecademy, we decided to instead direct our efforts to a bottom-up development approach [6]. This allowed us to learn through practice instead of theory and solve smaller problems to eventually build a larger application. In an ever-evolving and unfamiliar development environment, this approach was invaluable and greatly accelerated progress.

Focusing on upskilling meant that tangible progress was slow within the first two sprints. Combating this, a portion of each sprint was spent working towards realising our minimum viable product. Within the first sprint, an AR model was rendered on a mobile device and by the end of the second sprint the same model was rendered at a real-world location - achieving our minimum viable product. These achievements helped motivate the team to finish the upskilling process.

Upon completion of the Codecademy course we discussed what we had learned throughout the team and proceeded into full time development. Work was completed individually or in pairs. Work was broken down into issues and feature branching was utilised. The benefits of selecting a bottom-up approach were most evident here. Whenever a more efficient way to realise a requirement was identified, lowly-coupled and highly cohesive components could be refactored and "swapped" without cascading problems.

During our final presentation of the project, our client expressed that we had exceeded their expectations. Our final product realised all of our *Must* and *Should have* requirements from

our MoSCoW and the app could run on any Android device. Unsurprisingly, two of our stretch goals were not reached within the development cycle. This was expected as the customer had outlined they were very much only achievable if no blockers were met during development.

The capabilities of the team by the end of the project compared to the start is night and day. Many team members had little experience using Git collaboratively at the beginning and are now proficient enough to utilise feature branching and conduct code reviews. Alongside this each team member can now confidently add C# and Unity to their developer toolbox. The team also learned how valuable it is to design systems with extensibility and maintainability in mind, as a result of this initial refactors were costly and slowed progress whilst later ones were forgiving. The project also helped to develop skills beyond specific technologies. Often we were faced with problems resulting from little or outdated documentation, forcing us to learn to use alternative sources like source code or by working with teammates to synthesise a solution. The skills we have developed from this experience are invaluable and will be utilised throughout our careers.

If we were to work on a similar project in future our approach would differ. To reiterate previously lessons learned, less time would be spent on general training courses and more on courses that build upon existing understanding. Alongside this we would aim to employ bottom up approaches sooner and get all team members writing scripts within the first sprint.

4 Project Challenges

Throughout the project, we encountered a variety of technical hurdles, some more significant than the others, which collectively delayed our progress.

Initially we encountered issues with the university WiFi being slow. This caused many delays as Eduroam's limited bandwidth significantly limited download speeds which hindered progress as the Unity editor is massive - around 9GB before dependencies - leading to a lack of productivity as the team couldn't start work on the project for several hours. Ongoing downloads also harmed performance when trying to complete other tasks on the device. This challenge persisted for the duration of the project and as the project grew in size, pulling changes from GitLab became increasingly slower.

Unity installation also proved challenging as selecting a standardised setup was critical to limit incompatibilities. This proved difficult as we had to trial a number of editor and package versions to find ones which were compatible leading to inconsistent versions across the team and due to the issues with Eduroam detailed above, it was difficult to get everyone on the same version during our collaborative work sessions in the lab. Eventually we arrived at a setup which worked and created a setup guide for the team detailing all required editor and package versions then published that on the GitLab [7] for the team to follow. This step should have been more of a priority early on in the process to ensure every team member is running a validated version of the software before development is started.

Using the Geospatial API also proved challenging as a lack of official documentation and tutorials led to considerable delays integrating the API with our project and required lots of debugging. Our trouble continued as the Geospatial API utilises a package called 'Cesium'

to display a Google Earth style satellite view in the editor and this package has no support for Linux, forcing some team members to switch to Windows for the project.

As with all new technologies, using Unity came at a cost. C# is the primary language for the Unity framework and with none of the team having used the language before, upskilling was required. Fortunately, C# syntactically resembles Java, a language the team was very familiar with from previous courses. Therefore to standardise our understanding across the team, we each completed the Codecademy C# course [8]. Retrospectively, this was a good idea because regardless of varying programming abilities, we could be certain that by the end of the course, each team member had the same understanding of C#. However, if we did this again, we would alter the chosen course to one more suited towards experienced programmers. Codecademy is great for beginners, but as experienced programmers we found ourselves covering foundational concepts we were already familiar with.

The final challenge we faced during development was the availability of pipeline runners. As mentioned in later section 6, our codebase was configured such that merge requests could only be considered for approval after its pipeline passed successfully. The majority of our development was completed each Wednesday between 10am and 6pm. Many other teams were also developing during this time therefore the growing number of jobs overwhelmed the runners available. This led to lengthy delays as merge requests could be left pending for hours before acquiring a runner. We quickly identified this as a problem and setup dedicated runners for our project only and we learned to work on the project outside of peak hours when runners were more readily available. We suggested this to other teams to reduce the load on the shared runners considerably. This highlights a flaw in the course setup as the runners dedicated to the course don't cater to the wide range of projects on offer making the estimation of a 10 minute runtime unattainable for more resource intensive projects.

4.1 Source Code Management and Unity

Source Code Management is used to document changes made to source code over time. Typically each change is documented with a unique identifier. This enables changes to be reverted or referenced. Source Control Management can also be used collaboratively, enabling multiple developers to make changes to the same source code asynchronously. If two developers are to change the same file, conflicts can occur and require manual resolutions. Common SCM tools are Git, Subversion, Mercurial and Fossil.

Git [9] was our chosen source code management due to the requirements of the Team Project course. The source code for our project was centralised on the School of Computing Science's GitLab [10] instance [11].

The biggest source of blockers within the development cycle of the project originated from the incompatibilities between Unity and Git. Typically Unity projects utilise Unity's bespoke *Unity Version Control* [12] which is powered by Plastic SCM [13]. Due to this, there is less support for using Git with Unity and most guides are written and maintained by the community.
An average unity project is large with a few key files - the scene and script files. A scene represents a game world instance where objects can be placed and interact with each other - like a level in a modern video game. Each game scene is stored in a plain text file with Unity formatting for the editor to parse - as seen in Fig. 7. Unity formatting makes scene

Fig. 7: Scene File Contents Example

files difficult to read because all objects and files are given unique identifiers, randomly assigned by the Unity Editor. Because these IDs are only used internally by the Unity Editor, it is not possible to look up an object by its ID. Within our project, there were three scenes: main menu, tutorial and in-game.

Merge conflicts occur within scene files when multiple developers edit the same game scene in different commits. Git is unable to resolve conflicts with IDs automatically hence, manual review is required each time. Because objects are only identifiable by their child attributes within the scene file, each merge conflict must be resolved with utmost care making conflict resolution very time consuming. Occasionally, the same ID would be generated for two objects in different branches, this was hard to catch in merges and prevented Unity from opening the project. To resolve it we had to comb through the scene file to find where the duplicate ID was and delete it.

Unfortunately, this did not come naturally to the team. Initially, we were unfamiliar with the structure of scene files which led to many changes being reverted within merge requests. For example, two features being developed in parallel both wouldn't be implemented, because the feature which had its merge request last would overwrite some if not all, changes made from the prior merge request. After spotting features missing within the latest merge request, the team sat down together and discussed how to handle conflicts within scene files. From that point, cascading bugs were minimised within merge conflicts.

Merge conflicts within merge requests were the largest of these blockers. Any merge request that altered the main in-game scene would throw merge conflicts. This did not work well with our feature branching approach either. At points within the development cycle, we could have had six merge requests being worked simultaneously. If any merge requests were merged, each of the remaining ones would have to resolve merge requests with the main scene. This process repeated each time for any merge request that made changes to the main scene - which was unfortunately most of them.

A solution one could propose would be to have multiple scenes and have only one developer work on a scene at once. Unfortunately, this is not possible for our project. Cesium, the 3D satellite map in the editor, exists within a single scene, requiring all models to be placed in one scene.

In a real world project, it would be unlikely for Git to be a requirement. This would give us more freedom to pick our Version Control System where we would have used the Unity's system as it appears to be more targeted for Unity development and handles scene conflicts more intuitively.

4.2 Customer Communication

To ensure our customer was satisfied with the product we produced we recognised the importance of their involvement regarding key decisions. We held ourselves accountable to this by including our customer in all key decisions - from establishing requirements to final handover - the customer was included throughout the development cycle.

Meeting date	**Meeting Purpose**
27/09/23	Introduction and Project requirement discussion
1/11/23	Customer Meeting 1 - Framework Research
29/11/23	Customer Meeting 2 - Familiarisation of Mobile AR within Unity
17/01/24	Customer Meeting 3 - Geospatial and Mobile UI Development
14/02/24	Customer Meeting 4 - Integration and Polishing

Table 1: Customer Meetings

The customer meetings happened at the end of each sprint as seen in table 1. We used this time to update the customer on the progress of key issues for the previous sprint, demonstrated any new features and outlined the goals we had for the upcoming sprint. Throughout the meetings we allocated time for our customer to share any feedback and ask questions which ensured development continued to meet his expectations.

Outside of meetings, we frequently had questions that required input from our customer. After we started each sprint, we established a team member to be the communication link for the sprint. Any subsequent questions the team had for the customer were relayed through this one team member. This was effective as it prevented multiple team members from asking the same, or similar question. Additionally, it made it easier for the customer as he did not have to keep track of multiple conversations throughout each sprint, making our communication with him more organised.

However, it still was not ideal. The customer still had to keep track of multiple conversations and our use of both Teams and email for communication led to some confusion for both the customer and the team. In future, we would assign a single person as the communication link at the start of the project and would stick to a single communication channel to mitigate any chance of miscommunication and confusion for both the customer and the team.

5 Development Workflow

5.1 Issues and Branching Strategy

Without realising, we settled into the Trunk branching strategy, we would each pull from main, develop our feature and push back to main. This caused many merge conflicts, every time we committed, there would be a minimum of one conflict, resulting in us spending more

time resolving conflicts than developing. This slowed our development down considerably; we had not developed much of the app before the winter break. Additionally, we had focused much of our time on learning C# and AR development in Unity as mentioned in section 3.

In November, we looked at our branching strategy in more detail. We decided trunking was not good for our application, so we researched which branching strategy would be best for us. Our project was set out such that we had to develop different features and integrate them into the final product. This allowed us to rule out strategies such as release or integration as they typically provide a rolling release [14].

We decided feature branching without releases would be the most effective for us. It allows us to develop our features in separate branches merging them to main when the feature is complete; this helped with merge conflicts on every commit favouring one larger conflict on merge. If we were to attempt another project like this we would look into branching strategies in more depth earlier in the process to avoid these issues in the future.

At the start of the project, we created a standardised issue template that all issues followed. It consisted of four sections as seen in Fig. 8.

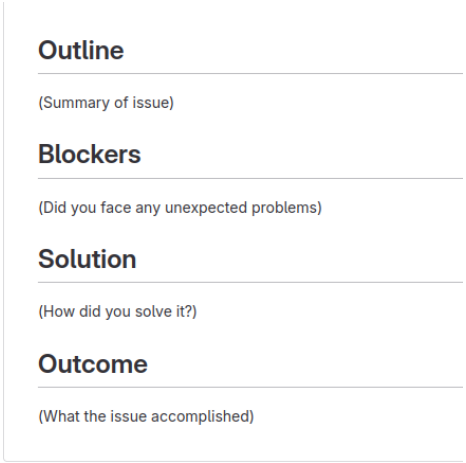

Fig. 8: Issue Template

We created issues for every feature, bug, and piece of documentation giving them relevant tags. When we switched to feature branching, we branched main for each issue, making the branch name the issue number with the title of the issue. Once the issue and its related branch was created we would begin working on the issue. On completion of the issue, we would merge the branch into main. This created its own problems as mentioned in section 4.1.

5.2 Quality Assurance

We took many steps to ensure the project was of good quality: merge requests always had a reviewer, we used pair and mob programming to get different ideas for the best way to implement features, and we included unit tests to ensure the codebase was functional after each commit.
Merge requests consist of a four sections as seen in Fig. 9.

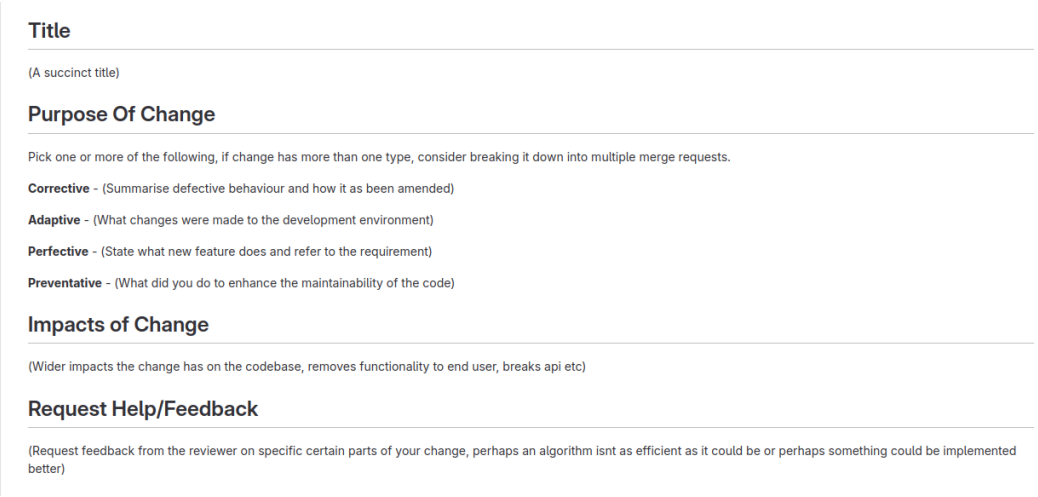

Fig. 9: Merge Template

We would assign another team member as the reviewer and wait for them to complete the review. Reviews mainly consisted of testing the branch to ensure it did not break other features, was not buggy, and implemented what the merge request detailed. Once the reviewer was satisfied with the state of the branch, they would approve the request.

We found this approach to merge requests effective, the standardised merge request format made it fast and efficient for reviewers to know what they were checking. Assigning reviewers to merge requests proved exceptionally effective as many bugs were found before branches were merged.

We made use of mob and pair programming often throughout development. Pair programming was mainly used for debugging, we would frequently ask each other to check our code if there were bugs we were struggling to find, or if we wanted advice on how to implement a feature. Pair programming proved extremely useful as bugs were found much quicker, and features were implemented consistently and smoothly.

Mob programming was used less than pair programming but it still proved useful. We often had different ideas for how we should implement a feature, so we would gather round one laptop to discuss how we think the feature should be implemented. We found this was effective as it allowed the entire team to understand how certain features were implemented, however, as it took more team members away from the issues they were working on, it was less efficient than pair programming.

In the future, we would plan to make more use of pair programming due to its efficacy, favouring it over mob programming. We found mob programming consumed too much man power for the results it gave us.

Unit tests were added late in the development, currently our unit tests ensure the quiz functionality is functional after each commit. After discussion, we regret not making more use of tests as it was relatively simple to setup in the pipeline, however they were time consuming to setup. Unity claims it is able to load a scene easily for a unit test [15], however, the method to load the scene is not blocking. This means while the scene is being loaded the rest of the unit test is trying to test with a partially loaded scene. It was not

possible to wait for the scene to be fully loaded, any attempt to do so resulted with the unit test entering an infinite loop.

The solution was to build up a new scene in code for each test. This was time consuming as we had to create an instance of each object in the scene, attach objects to each other, and add components to the objects. To reduce how long it takes to create the test scenes, we decided to make tests for each script instead of the entire AR scene. Unfortunately, we only had time to implement tests for the quiz scripts due to time restrictions from starting unit tests too close to the code freeze.

We recommend a team developing any program to implement unit tests early in the development cycle. There were multiple occasions throughout development where the quiz functionality was broken due to a merge request. Had we implemented unit tests earlier, many of these issues would not have occurred and we could have spent more time developing rather than bug fixing.

6 CI/CD Pipeline

From a DevOps perspective, our project and processes needed to be robust. Due to our customer's non-technical nature, we desired the ability to access our app file without needing to install the Unity Editor. Alongside this, we wanted to reduce cascading bugs where possible.

Therefore, from these requirements, we derived the need for a Continuous Integration/Continuous Deployment GitLab pipeline. The pipeline consisted of two stages, build and test, where each stage realised one of the requirements mentioned above.

A CI/CD pipeline acts like a conveyor belt within the codebase which produces artifacts as the end product. When developers make changes, tests get automatically run on said changes to detect defective behaviour and then the project gets built/deployed producing artifacts like binaries and code coverage reports.

However, achieving this pipeline was not easy. As highlighted in 4.1, Unity is not designed to be used with Git and less so with specifically GitLab pipelines. Fortunately, a community of developers, GameCI [16], have worked together to make Continuous Integration achievable when using Unity with GitLab. Despite this, the guides available were outdated and generalised, requiring a significant investment of time and effort to get the pipeline functioning.

Another blocker arose with the incompatibility of Cesium [17] on Linux-based machines. The *Cesium for Unity* plugin is packaged with *.dll* files which are only compatible with Windows machines. The SoCS GitLab runners are all Linux-based and hence threw errors when trying to parse the *.dll* files during compilation. To resolve this, we tried two different approaches sequentially:

1. Initially we built the Cesium for Unity package from source following some vague guides online. This approach worked so Cesium could be used on Linux systems within the Unity Editor but when the project was built into an *.apk* the geospatial

systems did not work correctly. This is most likely due to the followed guide containing scripts that needed manual fixes to work - where these changes probably didn't cascade correctly.

2. After the failure of the previous approach, we discovered that Cesium wasn't required for the final build and was just needed within the Unity Editor to visualise the real world. Motivated by this, a bash script was written, utilising jq [18], to remove cesium from the project within the pipeline before the build and test stages start. This enabled the pipeline to produce *.apk* app files as artefacts.

We came across another blocker, credentials were required to build the project using Unity Editor CLI [19]. To resolve this while keeping sensitive personal information safe, we made use of GitLab's CI variable infrastructure. This enabled us to store said information within the GitLab website rather than in plain text within our CI scripts. Therefore protecting sensitive information whilst simultaneously being able to utilise it within our pipeline.

Another prominent blocker was outdated GameCI [16] documentation. The guides provided were beneficial but details were obfuscated and instructions were not up to date. Countless times throughout the set-up of the pipeline we were left with unanswered questions due to our Unity Editor version being released after the guide's latest update in 2022. Resolution of this blocker was not as straightforward either. Some solutions required the pipeline scripts to be run manually in a docker container to test logic. Other solutions required the cross-referencing of multiple Unity GitLab pipelines to deduce the correct approach.

Despite the aforementioned blockers, once they were resolved, the pipeline was fully realised and provided both the building of the *.apk* app files and testing of the codebase upon each commit. Each pipeline built for base and il2cpp android for phones that support either. The structure of the pipeline can be seen in Fig. 10. Finally, this allowed us to protect our main branch and only allow merge requests to be considered for merging if the pipeline passed.

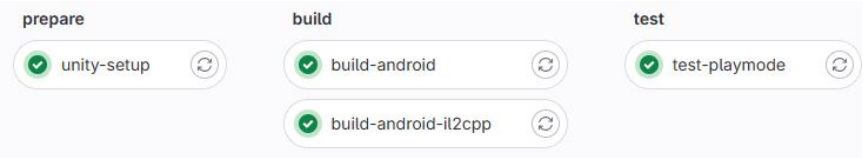

Fig. 10: Pipeline Structure

Despite the success of our pipeline, if we were to develop another Unity project, we would use Unity Version Control [12]. This is due to the native support within the Unity Editor for Unity Version Control. This native support has many benefits such as handling of large binary files without any extra configuration - unlike Git which requires GitLFS. Alongside this files can be locked - preventing merge conflicts - and graphical versioning allows developers, who are less experienced with version control, to easily manage changes to the project.

In criticism of the Team Project course, the requirement to use GitLab is understandable but equally tedious. The decision forces teams allocated Unity projects to circumvent common industry practices and workflows in exchange for convenient marking.

7 Handover

The handover process began on the 21st of February 2024. Initially outlined our following vision of what the customer will be able to do once the handover is complete:

1. The customer should have their own copy of all of the source code
2. The customer should be able to edit the project on your local machines
3. The customer should be able to retrieve the project .apk file from the pipeline
4. The customer should be able to install the .apk on multiple devices
5. The customer should be able to run the app to showcase it works on multiple devices - especially geospatially.

Following our customer's approval of the goals, we derived deliverables which would be necessary to realise the handover goals. They were as follows:

- **GitLab**: Code Published onto a Private GitLab Repository on GitLab.com
  - The customer will be made the owner of the repository and can add any further GamesLab members
- **PDF & ReadME File Comprising of**:
  - **Specifications**: Device specifications and requirements necessary to work on the project and run the app on a mobile device
  - **Guide**: Unity Editor Installation and Setup
  - **Guide**: Project Installation and Setup
  - **Guide**: Using the pipeline to get .apk files
  - **Guide**: Uploading an .apk onto an android phone
  - **Video Guide**: How to place new Gundams around the world and add them into the game system
- **Testing Outline**: Features we would like tested and feedback given on to ensure they work for the customers set up
- **Dates**: Key dates such as when the code will be on GitLab and how long we'll be able to provide support for fixing any unforeseen problems.

All handover goals were realised by the 19th of March 2024. A repository and group were created on *GitLab.com* for the customer which comprised of the source code and aforementioned deliverables.

8 Conclusion

This project served as invaluable software engineering experience. Given our prior unfamiliarity with Unity and AR technology, we can now confidently recognise our proficiency with the aforementioned tools. Being assigned a real world customer had many advantages, especially allowing us to exercise our communication and requirement gathering skills.

By the end of development the team produced an augmented reality mobile app which will enhance the experience of WorldCon 2024 attendees. To ensure customer expectations were met and requirements were clear, we held monthly meetings with our customer where we provided updates on progress. We also utilised these meetings to involve our customer when making critical decisions regarding the project.

Regular sprint meetings and the prioritization of issues highlighted the benefits and strengthened our understanding of the agile methodology. Alongside this, the use of techniques, such as planning poker, made solving problems and setting goals more efficient - enabling us to frequently hit our targets.

This project provided several blockers not commonly present within a university curriculum. Each of these experiences taught us valuable lessons and strengthened skills we can use for the future. We employed methods like pair/mob programming to effectively tackle and address challenges, such as integrating the Geospatial API and setting up the CI/CD pipeline.

Overall, this project has shown us that effective software development and delivery requires not only well written code but also the utilisation of project management, agile methodologies and customer relations. As a team we overcame many issues and in doing so learnt the importance of task prioritisation, branching strategies and frequent standup meetings. The lessons learned have been invaluable and we can now confidently employ these best practices in future endeavours.

References

[1] W. Con. [Online]. Available: https://glasgow2024.org/ (visited on 04/01/2024).

[2] University-of-Glasgow. [Online]. Available: https://www.gla.ac.uk/schools/humanities/staff/timothypeacock/ (visited on 04/01/2024).

[3] U. G. Lab. [Online]. Available: https://www.gla.ac.uk/colleges/arts/research/artslab/labsandthemes/ourlabs/gamesandgaming/ (visited on 04/01/2024).

[4] A. B. Consortium, *Chapter 10: Moscow prioritisation*. [Online]. Available: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html (visited on 03/26/2024).

[5] Unity, *Ar foundation*. [Online]. Available: https://unity.com/unity/features/arfoundation (visited on 04/03/2024).

[6] GeeksForGeeks, *Difference between bottom-up model and top-down model*. [Online]. Available: https://www.geeksforgeeks.org/difference-between-bottom-up-model-and-top-down-model/ (visited on 04/02/2024).

[7] SH-36, *Unity project setup*. [Online]. Available: https://stgit.dcs.gla.ac.uk/team-project-h/2023/sh36/sh36-project/-/wikis/Unity-Project-Setup (visited on 04/03/2024).

[8] CodeCademy, *C# courses & tutorials — codecadamy*. [Online]. Available: https://www.codecademy.com/catalog/language/c-sharp (visited on 03/26/2024).

[9] Git, *Git scm*. [Online]. Available: https://git-scm.com/ (visited on 03/29/2024).

[10] GitLab, *Gitlab*. [Online]. Available: https://gitlab.com/ (visited on 03/29/2024).

[11] S. of Computing Science, *Socs gitlab instance*. [Online]. Available: https://stgit.dcs.gla.ac.uk/ (visited on 03/29/2024).

[12] Unity, *Unity version control*. [Online]. Available: https://unity.com/solutions/version-control (visited on 03/28/2024).

[13] P. SCM, *Plastic scm*. [Online]. Available: https://www.plasticscm.com/ (visited on 03/29/2024).

[14] J. Store, *Qualities and issues of branching: A method proposal for formulating a branching strategy*, Nov. 23, 2020. [Online]. Available: https://helda.helsinki.fi/server/api/core/bitstreams/893b2467-0bf3-4dc9-8ea8-2757d2fd7a5b/content (visited on 03/26/2024).

[15] Github, *Unity scripting api: Scenemanager*. [Online]. Available: https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html (visited on 03/26/2024).

[16] GameCI, *Gameci*. [Online]. Available: https://game.ci/ (visited on 03/29/2024).

[17] Cesium, *Cesium for unity*. [Online]. Available: https://cesium.com/learn/unity/ (visited on 03/29/2024).

[18] JQLang, *Jq*. [Online]. Available: https://jqlang.github.io/jq/ (visited on 03/29/2024).

[19] Unity, *Unity editor cli*. [Online]. Available: https://docs.unity3d.com/Manual/EditorCommandLineArguments.html (visited on 03/29/2024).