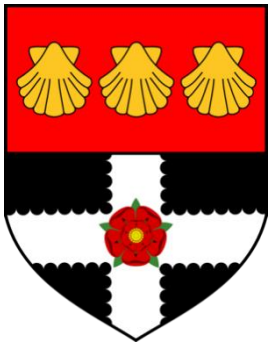
University of Reading

Department of Computer Science



Multiplayer Networked Application Utilising Photon and Unity

Callum Apps

Supervisor: Muhammed Shahzad

A report submitted in partial fulfilment of the requirements of the University of Reading for the degree of
Bachelor of Science in Computer Science

Declaration

I, Callum Apps, of the Department of Computer Science, University of Reading, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

I give consent for my work to be made available more widely to members of UoR and public with interest in teaching, learning and research.

Callum APPS

LIST OF ABBREVIATIONS
API Application Programming Interface
CCU Concurrent Users
CPU Central Processing Unit
EA Electronic Arts
ECS Entity Component System
FPS First Person Shooter
GEQ Game Experience Questionnaire
ID Identifier
IP Internet Protocol
LAN Local Area Network
LLAPI Low Level Application Programming Interface
LTS Long Term Support
MMORPG Massively Multiplayer Online Role-Playing Game
OS Operating System
PC Personal Computer
PUN Photon Unity Networking, might refer to the asset or the service in general
PVP Player Versus Player
RPC Remote Procedure Call

Supporting Documents

Webpage Promoting Project –

https://bitjump.tech

Questionnaire –
https://docs.google.com/forms/d/e/1FAIpQLSczPEVDgY8YVQmT5GZxyTd9mcLJTc5ReOUfcGb6LKW0i0pxPA/viewform?usp=sf_link

E-Logbook –
https://docs.google.com/document/d/1zybB1WQ4yHoQSuIEAJk_qL9WACubvg6b-W7UxQbW17c/edit?usp=sharing

GitHub link –

https://github.com/callumeppz/BitJump.git

Table of Contents

Question:

What factors should be considered when selecting a networking framework for integrating multiplayer features into game engines, and how can developers effectively implement their chosen framework?

Abstract

This document will provide a comprehensive overview of the developmental stages and the inspirational elements behind the creation of my Multiplayer Networked application, utilising Unity, and the Photon networking libraries. It aims to offer insights into the intricate process of bringing this networking project to life, shedding light on the creative journey and technical milestones achieved throughout the development journey.

Chapter 1 Introduction

1.1 Background

The inspiration of my game stemmed from the ideas of other captivating titles that had captured my interest before and during the development process. One of these influential games was the recently launched Super Mario Wonder, along with other notable platformers like Celeste. These titles inspired me to embrace their design choices and embark on creating a platformer that blended the realms of 2D and 3D elements.

I chose to begin development of this application as I have always had a real enjoyment when it comes to platformer games and playing online with friends and felt I would enjoy contributing towards the online multiplayer gaming environment. The genre of choice being platformer and adventure type was a result of primary and secondary data sources that i had collected and curated some figures on which genre people would prefer, in hopes that I could release my project for the enjoyment of others.

I considered various popular online multiplayer genres, including Action-Adventure, First-Person Shooter (FPS), Role-Playing Games (RPGs), Sports Simulation, Platformers, Real-Time Strategy (RTS), and Survival Horror. After conducting an online survey, it became apparent that platformers resonated most with the audience, as reflected in the figure below.

The survey not only highlighted the popularity of platformers but also underscored the significance of multiplayer features in gaming experiences. This insight further fuelled my commitment to delivering a project that aligns with the preferences and desires of potential players.

To create the most captivating applications possible I also ran a survey based on the type of multiplayer game that people would enjoy the most, 2D, 3D or a mixture of aspects. This resulted in the mixture genre being picked, which gave me the opportunity to expand on my skills and work on both 2 and 3 dimensional environments with the same characters, movement physics and networking scripts.

By having access to this primary data, it has allowed me to cater my networked game towards an audience that would thoroughly enjoy the play through, as well as having a range of different testers

to provide feedback on the networking and multiplayer aspects within the title on how they can be improved.

1.2 Aims and Objectives

1.3 Summary of contributions and achievements

Throughout this project I conducted an extensive analysis on the research and documentation behind the most optimum networking framework to utilise within a unity environment. This was achieved through the comparison of various comparative features being reliability, latency, and server architecture across frameworks such as Photon PUN, Mirror, UNET, and Fishnet, as well as providing a description on how each of these features have their contribution towards game development. My aim was to provide upcoming or new developers into the field with valuable, up-to-date insights on the most optimal networking solution for an array of different Unity projects.

In addition to this, I have outlined provided arguments for and against the use of Unique Selling Points (USP) and their requirements in applications for success. By weighing up the strengths and weaknesses of utilising USPs, an in-depth analysis will portrait the requirement for developers to find the 'in-between' value of familiarity and unique features.

Moreover, alongside these contributions I discovered various real-world uses, and the best networking framework overall, showcasing examples of when different frameworks were utilised, therefore reducing the little-to-no information surrounding real-world relevance to this topic.

Furthermore, I achieved the successful development of a fully-fledged application named BitJump making use of the Photon Unity Network (PUN) solution to instantiate a reliable, seamless connection between clients on a server. This was achieved because of my previous research and analysis into the best networking framework, the insights gained from this research served as a guideline on creating this application. Throughout this development process, I encountered various challenges, which further enhanced my understanding of networking frameworks and how the array of issues can be overcome.

To ensure that my application included an emerging technology as its unique selling point, I did some research into the implementation of the ChatGPT API for my NPC's dialogue, this required iterative prototyping and editing to allow for each of the prompts to be serializable and editable within the unity environment. This then allowed for me to tailor the prompt for each NPC dependant on what the NPC is utilised for. In doing so I was able to achieve a unique response for each client that joins the game.

Overall, my contributions and achievements in this project were outlined through my research, analysis, practical application of this research and overall understanding of the successful integration of networking functionality into Unity, to provide an enjoyable user experience.

1.4 Organisation of the Report

Chapter 1: Abstract and Introduction to Report

Chapter 2:

- Section 1: Literature Review based on USP in game development .
- Section 2: Integration of most optimal framework into a Unity environment

Chapter 3: Methodology

- Planning and Design
- Implementation

- Testing
- Issues

Chapter 4: Results

Chapter 2 Literature Review: Networking in Game Development: A Comprehensive Review

2.05 Abstract and Question

This short literature review would be used to answer the question of how advancements in networking within game development align with feature selection to drive success, and how does this influence network integration as a whole? The findings of this literature review could provide valuable insights for game developers, industry professionals, and aspiring developers on whether to research and implement specific networking tailored towards multiplayer in their applications.

2.1 Introduction and Overview: The Significance of Multiplayer Features in Game Development

2.1.1 The Evolution of Gaming: Embracing Multiplayer Experiences

Over the past 40 years, gaming in the community has become much more well-known and welcomed, with more and more people becoming a part of the experience. Gaming can be seen to have replaced more traditional activities and has a transformational impact on how we spend our leisure time'. In line with this, 56% of gamers choose to play online with friends rather than alone [statisca 2022]. This statistic entails an exaggeration of the fact that networking within a game is imperative for its success. Commercial games are almost expected to have a form of multiplayer game mode, allowing for their player base to wirelessly communicate over a game server. Leaders in this industry being companies such as Activision/Blizzards Battle.net, Electronics Arts (EA) Play, or RIOT's Vanguard each have their own popular titles being hosted on these launchers with large playerbases reaching the millions each day. Without the addition of Multiplayer to large titles such as Overwatch or FIFA, players would begin to become bored, and the ever-expanding player base of these massive titles would gradually fall off. This information is further reinforced due a survey that was taken in 2018 found that 53% **(Krassen Cindy, 01 Jan 2018)** of gamers suggested that multiplayer features were crucial in adding to the experience and enjoyment of the game.

2.1.2 Impact of Multiplayer Modes on Player Engagement and Community Building

Multiplayer game modes have become an integral part of the gaming experience, bringing players together from all around the world. These modes not only provide a platform for players to compete against each other but also foster a sense of community among gamers. The inclusion of multiplayer features keeps the gameplay fresh and exciting, as players can constantly challenge themselves against real opponents. Moreover, the social aspect of multiplayer gaming, such as teaming up with friends or making new connections, adds an extra layer of enjoyment and engagement to these popular titles.

2.1.3 Accessibility and Advancements in Multiplayer Networking Libraries

Networking libraries and their surrounding documentation are becoming simpler and more on-depth within game engines, making it much easier for new, upcoming developers to gain an understanding and develop their own indie or AAA titles within the field. Photon, Unity Netcode, and Bolt are all examples of libraries and APIs that aspiring developers can use to implement the dynamics of multiplayer gameplay and networking into their titles. Each of these libraries provides its users with a set of different functions and features that make it easier to handle the complexities of multiplayer

networking. These libraries can greatly simplify the process of implementing multiplayer gameplay and networking in their titles. In this respect, networking libraries such as the ones listed will continue to be improved upon and provide developers with a streamlined way to create robust and seamless multiplayer experiences, with examples of this advancement already taking place being the fact that the multiplayer experience adopting cloud-based servers and further accessible dedicated networking libraries.

Within this literature review, I will be exploring the development of applications as well as the role of Networking within online games and the impact that it continues to have on the development of new titles. This will be conducted using existing research, as well as my own primary research on the matter.

2.2 Literature Review Section 1: Exploration of Optimal Features and Balancing Innovation with Familiarity

2.2.1 Design as a Search Algorithm, is it useful in game development?

Previous research mentioned in a YouTube video (Tyroller, 2024) reaching an audience of 232,700 at this time, portraying the significance and importance of treating game development as a search algorithm, finding the most optimum features to guarantee success. This research would delve into the idea of offering a transformative perspective, allowing for game developers to search for optimal game ideas/inspirations just as that of a software designer searching for an optimal solution of a product. This method would require consistent exploration, refinement and prototyping like that of attempting an exploration to space. Which would be further reinforced by my own analysis within this section, and in which, how it can be applied to Unity Game development.

2.2.1.1 Evaluating the Efficiency of Design Search Methodologies

Through the implementation of this 'search algorithm' process, various stages will be required to find the most optimal features and ensure that the application receives a positive welcome once released. This concept would be used to find a USP (unique selling point) used to distinguish an upcoming game from its competitors present within the market, whilst also effectively targeting the correct audience. While the literature discussed the importance of finding the right balance between speed and accuracy, as well as setting objectives, it would be crucial to delve deeper into how these aspects contribute towards finding a USP and whether all games require this for succession.

- **The Searching Process in software design:** This iterative procedure investigates various design methodologies and evaluating their efficiency [2]. Relating back again to space exploration, being the decision of thrusters, in the realm of game development, the selection of best networking library stands as a pivotal point of success.
- **Continuous Prototyping and Adjustment:** Once an effective design practise has been discovered, akin to game software design, game development would require regular assessments and adjustments to be made. This can be executed through continuous prototyping and measurement of game versions and refinement based off the provided feedback. Therefore, further ensuring the alignment of the application with goals and user needs.
- **Effective searching strategies:** Crucial within software design and, therefore, game development, for the launch of a successful application. Focusing on the right features/parameters is crucial to enhance an in-use methods quality and relevance.

2.2.1.2 Exploring Problem solving dynamics in Game Development

The research found from the video mentioned delves into a range of different problems and their solutions during the process of game development.

Speed vs. Accuracy Trade-off: This would be an important aspect of game development and would alter based on the type of game being created. Beginning with an exploration phase, relating back to speed and accuracy in which the rate that potential features are found. This would refer to the amount of time that developers spend prototyping various design features or game mechanics to meet user goals most accurately. This phase would refer to the brainstorming of ideas that will be used within the application. However, this would come with the drawback of developers having to find a

‘balance’ between speed and accuracy. This would refer to the idea of rapid prototyping, to ensure that deadlines are met, whilst also confirming that the features being suggested, aligns with user needs. Achieving this balance would allow for the successful testing and development of user-required features within an application.

Infinite Search Space: Setting clear objectives in game design is another crucial feature, due to the boundless possibilities and ideas, the utilisation of previous data trends, reviews or ratings as well primary collected data can guide in the direction of the creation of a unique, relevant title.
To find the sweet spot for game development, finding the optimum space within the innovation scale is essential. Creating an application that takes inspiration from popular, past releases, without reaching the point of pointless or excessive innovation. Relating back to the search process, this may involve a wide spectrum of ideas, narrowed down later to ensure an optimum amount of innovation is found.

Meeting Objectives: Furthermore**,** the importance of objectives is further emphasised through ensuring that financial success or unique innovation are met within the design process.
This would require the consideration of feedback, managed resources and time, clear communication between stakeholders, developers, and avid users when it comes to development and maintaining the game after release. All these features would ensure a smooth paved way to meeting the projects goals.

2.2.2 Critical Analysis and Synthesis

2.2.2.1 Redefining the Role of Unique Selling Points (USP’s)

This research video would go into detail about finding a middle between a title with ‘over-the-top’ innovation and no innovation, creating a recognisable title without the requirement of a solid USP whilst also meeting objectives and goals set by their target audience. This video would present an intriguing perspective on treating game development as a search algorithm, advocating for accurate identification of unique features to resonate with players on an emotional level, this would suggest on finding a balance between innovation and similarity, creating distinct titles without relying on USP. Whilst this approach would challenge conventional notions about the necessity of a USP in game development for success, it would prompt a critical examination of practises within industry.

2.2.2.2 Reconsidering the Necessity of Unique Selling Points (USPs)

However, this contradicted through various articles stating that USP’s are a requirement for a game’s success (LinkedIn, 2024). These articles underscore the importance of finding a USP that would set games aside from the current market. Unique selling points would be a common sales technique used within game development to captivate an audience and assist titles to stand out from its competitors. Companies take this into consideration to boost their game and ensure success.
The emphasis on USP’s reflect on the competitive mature of the gaming industry, with numerous titles battling for players attention. In this crowded marketplace, a strong USP would serve as a crucial differentiator, whether that be gameplay mechanics, compelling storytelling, or improved graphics, this could be a factor impacting a games success.

2.2.2.3 Debating the Role of USPs in Game Success

On the other hand, further reinforcing the idea of finding a middle and putting less of a focus on finding a USP is reinforced through arguments suggesting against USP [WhatGamesAre 2011], this

article would go into detail about how USP's are often superficial and should not be the focus of an application being unable 'to resonate with players on an emotional level'. The concept of searching for optimal features and distinguishing a game through its USP built from other applications is valuable, however it would be crucial to examine whether this approach is universally applicable. This is where Jonas's idea of treating a game as a search algorithm operates effectively, being able to accurately explore various features that would provide familiarity to players whilst also suggesting 'some' new changes in a timely manner that will ensure objectives are met, would irradicate the ideology that mass amounts of USP and innovation is crucial to a game's success.

2.2.3 Conclusion: Revisiting USP's are they crucial elements depicting game success?

It would be essential to navigate the 'infinite search space' stated in the video to an accurate degree to ensure that applications meet that of user and stakeholder goals, being similar to previous successful games, but unique and robust enough to be popular itself without too much focus on innovation. This aspect would be crucial in the success of a release, and can be evaluated through iterative prototyping, testing, and working off the feedback provided on whether extra effort is required to meet goals.

2.2.3.1 Striking the Balance between Innovation and Familiarity

Overall, while the idea of game development requiring a unique USP does provide correlation to success, it would warrant further examination which can be achieved through the concept of treating it as a 'searching algorithm'. 'Jonas Troyller's' idea of finding a balance between innovation and similarity, would address both the merits and shortcomings of game development, providing users with nostalgia and novelty at the same time. This approach would ensure that players experience the comfort of familiarity whilst also being enticed by these new elements, consequently enhancing the overall gaming experience catering to a diverse range of player preferences, also leading to longevity of a title. In conclusion finding a clear balance between speed and accuracy, innovation and familiarity and consistent refinement off feedback would be some of the crucial considerations that will need to be explored.

2.3 Literature Review Section 2: Determining Unity's most suitable Networking framework solution

The article at hand (Jokiniemi, 2014) would surround the idea that networking is an essential feature in game development requiring research and prototyping to find the correct library. Accessibility to networking has increased significantly due to the extensive documentation available, stating its functionalities, and the array of features provided by networking libraries. In the realm of multiplayer game development, leveraging client-server architecture requires the utilisation of networking APIs. This is important as the central game server facilitates numerous game functions and client interactions, such as server switching, username storage, and movement synchronisation. While client-side tasks encompass audio, graphics, and input management.

Additionally, Jokiniemi's article focuses on the unity game engine, being a cross platform game development platform with access simplistic networking frameworks. The article goes into detail describing the different features that Unity has to offer, being able to effectively 'build and modify your game' allowing for easy 'dragging and dropping of game assets' into your virtual world.

Unity has access to a wide range of networking frameworks as well as having its own Unet framework utilising the RakNet library. Some of these frameworks that would allow for 'more possibilities' would be Photon, Ulink, SmartFox and Tnet, with Photon being the most popular of those due to its robust and extensive feature set.

2.3.1.1 Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)

User Datagram Protocol (UDP) and its implementation
The field of Game development and Networking API's (Application Program Interface) employs a variety of protocols with TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) being the most prominent. UDP provides the client with an unreliable connection utilising a low-overhead network transfer, therefore stating that whilst the connection will be responsive, data is not always guaranteed to be delivered to all clients.
Jokiniemi does not expand on UDP's application for real-time applications. UDP thrives in situations in which swift transmission is required, allowing for a smooth user experience. Fighters, shooters, battle royales and other games utilising player vs player, voice chat techniques or other latency-sensitive features, benefit most from UDP's application and ability to provide a seamless experience, without the worry of delays or interruptions.

Transmission Control Protocol (TCP) and its implementation
TCP (Transmission Control Protocol) is another networking protocol mentioned throughout Jokiniemi's article, it differs from UDP due to it being described to provide a reliable connection between clients ensuring reliable and ordered delivery of data packets. Although this protocol would come with a higher overhead, leading to a potentially higher latency and slower transmission speeds than that of UDP. TCP in game development would be utilised for features such as matchmaking, authentication or transactional operations which are not latency sensitive.

The implementation of either one of these networking protocols will be required to implement multiplayer-features. It would be crucial that developers consider the benefits of both protocols and ensure that the correct choice is made to provide their users with a seamless, enjoyable experience.

2.3.1.2 Peer-to-Peer vs. Client-Server Architecture: A Comparative Analysis for Online Games

This article uses these two quotes directly from Microsoft to describe the two forms of networking that can be utilised within game development, peer to peer and server based. Understanding server architecture is paramount in designing robust online games. According to Microsoft (Microsoft, 2009) peer-peer networking would utilise a decentralised environment with a group of computers being connected, therefore allowing for direct interaction between connected clients. Users can share resources and communicate without the requirement of a middleman authenticating users. Moreover, server-based-networking, as spoke about by the same source (Microsoft, 2009), would more be the use of a dedicated server that would govern resource access and data storage, clients would connect to this server and all communication will go through it.
This literature would present alternate perspectives on the suitability of these structures for seamless online gaming.

Peer-to-Peer Architecture:
When utilising a peer-to-peer architecture, latency will be minimised due to the direct connection between peers, which could be potentially useful for the enhancement of real-time gameplay experiences (Pandey, 2022). Some AAA titles being GTA Online by Rockstar games or Super Smash Bros:Ultimate from Nintendo have inherently used this structure to ensure seamless connectivity between players with little loss, which is crucial in titles involving player vs player scenarios (Servers, 2024, Pandey, 2022). However, challenges will arise, such as stability and scalability, if a host was to disconnect due to network issues or their being to many clients for the personal device to handle, this would cause issues within gameplay and potential cause a loss in data. (Pandey, 2022).

Client-Server Architecture:
Applications utilising the client-server architecture may experience a higher latency due to all connections being made through a central server, however at a reduced risk of data loss due to its higher capability. Client-server approaches benefit from having dedicated servers to scale more players in a room (Pandey, 2022) but there would be a limit on how many concurrent users can interact in Realtime. In addition to this, if a server was to go down, all clients would lose access rather than just individuals if their personal device fails.
In the dynamic landscape of online gaming, the choice in which server architecture should be utilised is crucial. In popular online games, the use of Client-server may be beneficial over that of peer-to-peer due to the security benefits offered by its configuration. The hiding of player IP addresses would assist to mitigate the risk of detrimental denial-of-service attacks targeting individual players.

Furthermore, the choice should also be based on a multifaceted analysis considering factors including game genre, player count, desired gameplay experience and pace of game. Each of these factors are crucial aspects determining the enjoyability of the application and directly contribute to the facilitation of seamless multiplayer experiences. Future work in this area should focus on the refinement and optimisation of both peer-to-peer and client-server architectures. These endeavours would pave way for innovative solutions developed to enhance performance, security, and scalability of gaming environments, ultimately enriching the experience these applications provide to millions of players worldwide.

2.3.1.3 Latency and Bandwidth and their contribution to a seamless experience

Although latency and bandwidth were two crucial elements that were not reinforced to much within Jokiniemis article (Jokiniemi, 2014), these will need to be taken into consideration when developing multiplayer applications using Unity. These would ensure a seamless synchronisation between players and add to the sense of immersion portrait by the game.

Managing Latency for Fluid Gameplay

Latency is known as the amount of time it takes for data to travel from one point to another in a network. This will need to remain low for games due to the real-time nature of gameplay. Latency will always experience certain times in which the user will 'Jitter' or experience interruptions in the gameplay; this can never be eliminated, but it can be minimised through network optimisations [2]. Latency is usually dependent on the client's internet connection, referring to the time it takes for their data to travel to a server; however, it can also be affected by server performance and network congestion, which would need to be handled by game developers. This can be solved through the implementation of various techniques and optimisations, such as client-side prediction, server-side interpolation, and network smoothing algorithms. Fluid control is possible when latency is below 200ms; after this, the resulting gameplay becomes inconsistent and unresponsive. In real-time strategy games, latency can reach up to 500 Ms and still be acceptable if the game remains ‘static (jitter is low)' [2] and does not require quick reaction times. However, for fast-paced action games or competitive multiplayer games, a latency of 500 Ms would severely hinder the player's experience and make it almost unplayable, with first-person shooting games needing to be under 100 Ms. While the lowest latency is ideal, between 5 and 60 Ms is usual for real-time gaming [5], depending on the specific game and network conditions.

Optimising Bandwidth for Seamless Connectivity

Bandwidth is another important factor that affects the player's experience and can impact gameplay. A higher bandwidth would allow for less lag due to the higher data transfer speeds, whereas with a lower bandwidth, the user may experience lag or slower reactions within gameplay. This can be helped by game developers through multiple network optimisation techniques such as latency simulation, bandwidth profiling, and prioritising network traffic. Latency simulation would be the process in which developers test their applications under various network conditions to simulate real-life latency experiences. This would give them insights on how they can tailor their application to respond to these different experiences to provide users with a lower bandwidth and a pleasant experience [5]. Furthermore, bandwidth profiling would be used to determine the optimal bandwidth requirements for the application to deliver a seamless video and audio experience to the user. This would involve analysing the user's network conditions and adjusting the quality of the video and audio in real-time [6]. Finally, network traffic analysis would provide developers with an idea of which of their networking algorithms will need to be optimised based on player latency in various areas of the title. This information can be used to improve the overall performance of the game.

2.3.1.4 Further Research and Comparison of models, a gap in knowledge (link this back to articles)

A critical aspect that this literature review looks to envelop is the identification of gaps within existing research articles. Within the realm of networking frameworks and the gathering of research to determine the most optimal for game development, there is an array of studies portraying the benefits of individual frameworks or comparisons of a couple (Aaltonen, 2022). My study looks to bridge this gap, providing in-depths examinations of each of the networking frameworks and comparing them via the key features determining their application.

Despite the vast number of literatures discussing the benefits of various networking protocols, there is a notable scarcity of articles depicting the comparison of multiple frameworks. Because of these actions, developers usually struggle with the decision of what framework to utilise based off their requirements. This section of the literature review looks to fill this gap, providing developers with a 'central point' for all information regarding this sector.

Unity Networking has a range of different networking protocols, with Mirror, PUN, FishNet and UNet all using a client-server model [Reddit Multiplayer Networking solutions 2023] meaning that all scripts will need to be run on a client's side, before being forwarded to a server that would then transmit this information to all clients. Based on the current research in this article on various networking frameworks (Chapter 3 Section 3.1) as well as current articles making comparisons between a select number of frameworks, the figure below has been curated, containing crucial information regarding the most popular frameworks being: Mirror, PUN, FishNet, Unet and LiteNetLib.

Feature	Photon PUN	Mirror	UNET	FishNet	LiteNetLib
Supported Platforms	Android, iOS, Windows, macOS, Linux, WebGL, others	Android, iOS, Windows, macOS, Linux, WebGL, others	Android, iOS, Windows, macOS, Linux, WebGL, others	Android, iOS, Windows, macOS, Linux, WebGL, others	Android, iOS, Windows, macOS, Linux, WebGL, others
Documentation	Extensive	Extensive	Extensive	Limited	Limited
Ease of Use	Beginner-friendly	Beginner-friendly	Beginner-friendly	Intermediate	Intermediate
Network Architecture	Client-server	Client-server	Client-server	Client-server	Peer-to-peer
Protocol	UDP	UDP	UDP	UDP	UDP
Latency Handling	Good	Good	Good	Good	Good
Reliability	High	High	High	Medium	Medium
Scalability	High	High	Medium	Medium	Medium
Community Support	Strong	Strong	Strong	Limited	Moderate
Price	Freemium model with paid plans	Open-source with paid support plans	Server hosting $5 a day dependant on CCU.	Free andss open-source	Free and open-source

This table portraits the image that both Photon and Mirror are the most popular options due to achieving the strongest results in all areas, whilst having the most community support and documentation, it is the most reliable when it comes to higher scalability and low latency, which as discussed in previous sections, is crucial in achieving seamless interaction between players.

2.1.3.5 Most Optimal Solution for Unity Networking

The information above states about the most popular networking framework Photon PUN being utilised for best connection/communication between clients, as well as other corresponding articles relating to using this. This therefore makes it an optimum choice for Unity networking, in which the next section would go into greater depth. The PUN framework offers extensive documentation as well as cost-effective implementation compared to that of Unity's solution for networking (Unet). Although it requires installing an SDK, the benefits seem to outweigh challenges.

The accompanying figure illustrates the reduced number of hops required for communication between clients, highlighting the efficiency of the frameworks. On the left-hand side of the image, Unets framework is depicted, showing a host (Host A) connected to two clients (Client B and C) via a relay server. Whereas on the right is Photons architecture following a similar approach by using Host A to connect to clients through the means of a relay server also. This figure portrays that both networking frameworks utilise relay servers to provide connectivity between clients. However, a benefit of Photon as discussed in Unet vs Photon Engine 2018 – a Comparison is its ability to handle host migration when a client leaves seamlessly. Contrastingly, if a client left when the application is using Unet, the complete game stops [] due to all messages having to go through the host. This ability is displayed by Photon due to the fewer number of hops required (2 vs 4) to communicate from client to client, resulting in much faster performance.

Photon, being the focus of this report, has made this decision and utilises the UDP protocol when connecting its users across a network, due to it being the most reliable while maintaining low latency and high performance.

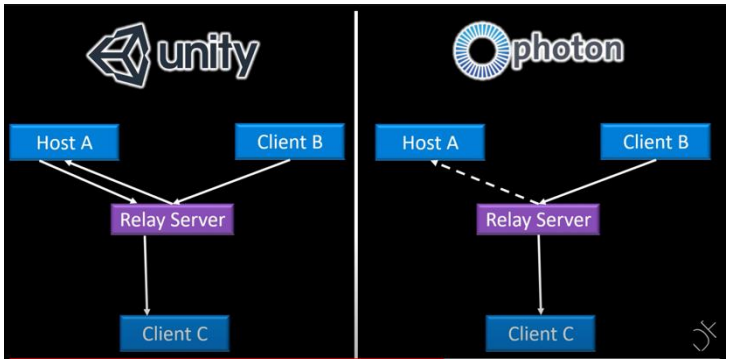


Figure 1 Comparison of Networking 'Hops'

From my own research and analysis, using both frameworks, I found that Photons version of hosting an application is free for the non pro version and would be suitable for smaller projects, compared to that of Unity incorporating a $5 charge for each day of hosting. This is another drastic reason for smaller, indie developers to choose Photon as their means of networking.

2.4: Literature Review Section 3: Integration of Photons and its Key features, a relation back to game development and how networking can be applied effectively

2.4.1 Integrating Networking into Unity, A review of existing literature

This on-going section would explore the various networking resources available to developers with focus being on the Photon API, as discussed previously, due to its low-cost. It will go into detail on the rationale of choosing to integrate Photon based on Jokiniemi's, Mika's (Anttonen, 2019) and Dao Quans (Quan, 2021) articles revolving around networking and expanding on this using articles directly about Photon. Additionally, it would outline the process of developing a networked title using Photon and Unity, highlighting the ease-of-use and effectiveness for implementing multiplayer functionality.

When developing a networked game in unity the developer will need to decide on the most optimum networking solution based on factors such as game genre, expected player count, and desired gameplay experience. Photon will provide a comprehensive set of tools and features to facilitate seamless and efficient multiplayer functionality within Unity, whilst also remaining cost effective for developers.

2.4.1.1 Setup of Networking (PUN)

This subsection would provide an insight into existing literature correlating to the setup of networking within a Unity environment using Photon Unity Networking (PUN). Expanding on previous research and findings, it will explore the rationale of integrating PUN as a networking solution and highlights the key features that it has to offer.

Rationale for Choosing PUN

The decision to integrate Photon Unity Networking would stem from the previous section of this literature review as well as online sources that focus on the Photon API. Dao (Quan, 2021) and Mika (Anttonen, 2019) articles emphasised the significance of networking in game development describing it as 'an essential part of gaming' (Anttonen, 2019). After the depreciation of Unet, clients decided to switch to PUN, leveraging it to gain access to a cost-effective, powerful networking solution, aligning with the needs of unity game development.

Implementation Process

Anttonen provides practical insights into the implementation process of PUN within the context of their project. Within their article they emphasise the importance of downloading PUNs plugin from the Unity Asset Store, offering both a free Photon Unity Networking Classic and a paid Pro Version. (PhotonEngineFusion). Within the article, the accessibility of PUN for aspiring developers is mentioned due to the free version containing an array of Photons advanced features.

Integration Process and Configuration:

Anttonen discusses the initial integration of PUN. Developers will be required to download its plugin from the Unity Asset Store and input their app ID assigned from the Photon website for authentication and to establish a connection to the server handling client connections. After setup, developers can customize parameters to further tailor towards their vision, enabling for the configuration of region, maximum number of concurrent users to a room, NickNames, and other essential elements. In addition to this, further information regarding the regions is covered by Anttonen, this would be the process of optimising hosting methods through the alteration of regional settings within PhotonServerSettings, allowing for clients to connect from various regions, which defaults to 'best region' (PhotonEngine, 2024b) selecting the optimal region for the current user.

Syncing Game States

Syncing game states is a crucial milestone in the development of multiplayer applications, ensuring that all players have a consistent experience provided by a effectively synchronised environment. Anttonens article provides insights into synchronisation using Photons Network. The article goes on to talk about the PhotonView component stating that 'Every component that has network features having one' (Anttonen, 2019) enabling 'gameobjects states to synchronise between clients' (Quan, 2021) whilst then comparing this to Unity's vanilla Unet Network identity. This subsection aims to further explore the concept of synchronisation in PUN utilising previous literature and discussing the functionalities of various Photon Components.

Anttonen states that the PhotonView component to contain an observed Components section which is idealised as an array that includes all networked components including scripts and components included within PUN on a gameobject, acting as a bridge connecting local game instances to the Photon Cloud. Furthermore, alongside the PhotonView component discussed, the

In addition to this Anttonen also mentions Photon Transform view. This component is compared to the Unets network transform as a way to 'change settings considering a gameobjects transform In network' (Anttonen, 2019). This component is essential for the synchronizing of position, rotation, and scale of a networked game object, therefore ensuring consistency among clients. Similarly, the Photon Animator View is also mentioned, handling animation synchronisation, enabling smooth and synchronised animations.

Comparative Analysis:

Being compared to Unity's Vanilla Unet Network identity multiple times, the Photon components offer many advantages in terms of performance, scalability, and simplicity. An example mentioned within Anttonen's analysis would be the fact that the Photon Trasform View would allow for synchronisation of an objects rigidbody where Unet does not. This is advantageous as it would increase the sense of realism into the game, incorporating gravity, object movement and collisions into the synchronisation, which in Unet will need to be programmed separately.

The PhotonView component provides a more robust solution for the syncing of game states within the Unity environment over that of Network Identity, this would be due of remote procedure calls (RPCS) providing 'high-level communications. However, it would be crucial for game developers to ensure that Photon meets the requirements of their game's genre before implementation. As interpreted from Photons website they provide 'the ease of use of Unity's Networking with the performance and reliability of Photons Realtime' making them a prominent choice for game developers. (PhotonRealtime, 2024)

RPC Calls, PUN in programming

Furthermore, Anttonens article delves deeper into the realm of synchronisation, highlighting the significance of the previously mentioned Remote procedure calls (RPC's) and Photons pre-made methods. With the OnJoinedRoom() method mentioned as an example illustrating a function being called when a player joins. Antonnens research describes how Photon uses similar methodology to Unity with methods named intuitively reflecting the functionality, contributing to the idea of simplicity within programming.

RPC calls are explicitly featured throughout Antonnens PUN in programming aswell as within the Photon Documentation. Described as 'method calls to all clients in the same room' (PhotonEngineFusion), RPC's serve as crucial tools that are used to carry 'data between communication programmes'. Anttonen's analysis further emphasies the significance of RPC alongside the parameters used when it is called and the PUNRpc and PhotonView attribute that is required prior to its execution. Seen in the figure below is the command exclaimed throughout the analysis as means to call and RPC and will be followed by either All, Others, or MasterClient parameters to determine who will receive the call.

$$PhotonView.RPC(\text{FunctionName}, RPCTarget, parameters)$$

RPCTargets [RPC Explained Video] –

- **All – Executes function instantly on master, then sends function to everyone else to call this function their side.**
- **AllViaServer – Sends function call to the server then server calls all clients.**
- **Others – Calls functions for everyone else but you.**
- **MasterClient – Calls function for the MasterClient only.**
- **AllBuffered – Cache function call, calls function for all incoming users.**
- **AllBuffferedViaServer**
- **OthersBuffered**

For instance, RPC targets with the ALL parameter will assume that all clients currently within the server requires the function to be run, others are the same except it will exclude the client who calls it and Masterclient will only run for the host of the server. Additionally, Anttonen exclaimed that AllBuffed and OtherBuffered can be utilised as a means of caching function calls for all incoming users, enhancing efficiency and ensuring consistency among all current and new users.

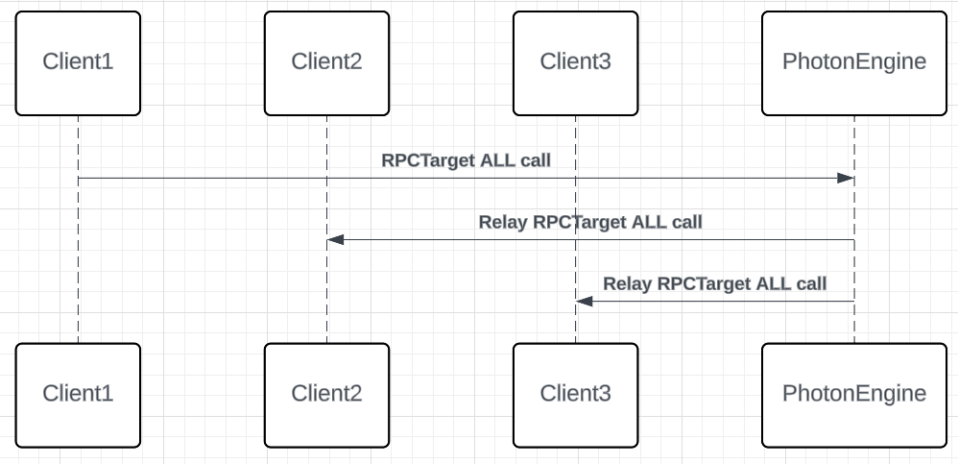


Figure 2 RPC Calls from client to server

Summary of Setup Section:

Overall, the insights provided by Anttonens article would offer valuable considerations and steps for developers seeking to implement synchronised multiplayer into their Unity application utilising awthe PUN framework sharing their own personal experience of the configuration process. Alongside this, insights on more advanced synchronisation such as animation or movement can be examined from this analysis, ensuring that clients receive a consistent experience. Through the shared experience, considerations and steps providing insights, Anttonen has successfully contributed towards a deeper understanding of PUN and its usage.

Future studies could take place to focus on the evaluation of the performance and scalability of photon components in large scale multiplayer environments, and how Photon reacts when more than 20 Concurrent users are connected, being the limit of Photons abilities. In summary, Photon's robust networking framework, coupled with PUN's intuitive features will make a large contribution towards to development of seamless multiplay between clients within Unity applications, as evident through the vast amount of projects Photon displays as well as being the leading networking framework for game development within unity.

Photons Key Features

One of these key reasons for Photons implementation would be its extensive features reinforced by solid documentation (PhotonEngineFusion). After making previous decisions on which networking framework to implement considering the strengths and weaknesses of each, it would be time to plan the game itself, this would be the decision of how to split matchmaking and gameplay, determining how they can work with each other effectively (Anttonen, 2019).

The articles references throughout this literature review emphasize a range of Photons key features, which can encompass a wide range of different functionalities including the incorporation of matchmaking and chat features which are essential in the creation of an engaging multiplayer experience. In addition to this, Photon enablement of synchronisation of player movement, enemy movement and interactable are crucial for ensuring a consistent gameplay where users can work together towards one goal.

Creation of Rooms (matchmaking cont.)

Anttonen's article goes into depth explaining the process of matchmaking, exclaiming the use of both private and public lobbies. Prior to the creation of rooms, PhotonNetwork.player.Nickname is utilused to assign users unique ID's, diffrentiating between the clients, thereby laying the groundwork for connection of clients to rooms. The next stage would involve the creation of either public of private rooms both utilising the Photon method PhotonNetwork.CreateRoom() with the first parameter being the name of the room and second being the properties. This would just create a public room, however it is to be mentioned that using IsVisble will prevent other clients from seeing this room, thus making it private.

Anttonens article follows a iterative approach with setting the username prior ro the creation of lobby, therefore ensuring a seamless transition between user authentication to room creation.

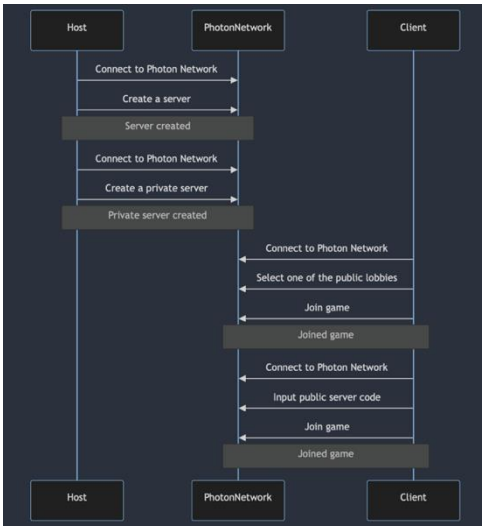


Figure 3 Creation of rooms depicted

Gaps in knowledge, real world examples of Photon

Most research articles do not bring to life some of the real-world examples of photons implementation. Popular games such as Hasbro's Beyblade have successfully implemented cross-platform multiplayer features, seamlessly connecting PC and mobile players using Photon's technology(PhotonEngine). Although multiplayer games seem to be undoubtedly popular and in high demand, there are still challenges to overcome in terms of network stability and player synchronisation, which single-player games do not have to contend with. These real-world examples would highlight the practically of photon in the creation of engaging multiplayer experiences with 'beyblade' receiving a rating of 4.3 on the IOS app store (Hasbro.Inc, 2017), portraying the enjoyable experience that is granted when playing this online experience.

Additionally, single-player games such as Starfield and Nintendo's new Zelda: Breath of the Wild 2 provide immersive experiences without the need for an internet connection or coordination with other players; however, if they were to include multiplayer, popularity may be positively impacted by attracting a larger audience and fostering a sense of community among players.

Through the exploration of more real-world examples this would widen the discussion of the opportunities and challenges that would arise when implementing Photon and how different games can benefit off its use. Additionally, the examination of multiplayer integration within games and how it can affect popularity, may convince further developers to add their opinions on the matter of best networking solutions, enriching our knowledge on these frameworks and therefore their application to games.

2.4.4 Analysis

This section was used as a summary regarding choice and implementation of required networking framework being suitable for Unity applications, emphasising the difference between more 'modern' networking means compared to that of Unity's legacy networking. This would be due to Photon being a more simplistic and user-friendly way of introducing networking to an application, whilst also coming at a lower cost to that of Unet, *being 10x cheaper per gigabyte*. To leverage the power of Photon within applications, developers must understand the key features required to seamlessly connect players with thorough synchronisation techniques. This would entail delving into Photons extensive documentation, exploring its API and the many features in which the library has to offer and staying up to date with its Photons regular updates.

2.5 Conclusion of Literature Review

In conclusion this literature review would envelop three essential sections, each crucial in the development of multiplayer applications. Beginning with section one, having a particular focus on the searching of

specific features and whether incorporating a unique selling point (USP) within applications and their correlation to a multiplayer game's success. Central to this examination is the exploration of treating game development as a search algorithm, an exploration of whether USP's are a requirement, and if so how to go around finding one. Drawing from a variety of external articles and forums, this section provides an analysis, weighing both for and against arguments, determining its significance.

The second section would then swiftly move onto the utilisation of networking and a comparison of the most optimal networking methods. This involves the assessment of various factors impacting the effectiveness of connections, methodologies used to counteract these factors alongside the various technological frameworks that can be applied to games. It navigates the intricate networking architecture of each framework, evaluating their efficiency and suitability determined by genre or type of game. Furthermore, mostly revolved around Jokiniemi's article (Jokiniemi, 2014), my own analysis of various other articles comparing frameworks was required to fill a gap in knowledge currently present. This would involve the comparison of various networking frameworks, involving their reliability, documentation/community support and cost, giving aspiring developers a 'helping hand' determining which to implement in their upcoming multiplayer projects.

The final section, drawing from Mika's article (Anttonen, 2019), will then go into further detail explaining the key features delved from the previous section determining the most popular networking framework to be Photon. This section is essential due to it providing developers with insights on the specific features that they can implement to make full use of this framework and enable them to develop fully fledged multiplayer applications. Throughout this section, gaps in knowledge are also filled, the implementation of real-world examples and practical strategies for optimisation of performance and user experience are mentioned. Through the synthesizing of findings from Anttonens article and other relevant sources, suitable recommendations are provided to developers for leveraging Photons capabilities effectively.

In conclusion, this study envelops the use of networking within titles and its ability to undoubtedly revolutionise the gaming industry and enhanced the gaming experience for players worldwide. This study has also been a comprehensive overview of beginning game development through exploration of ideas, to successful implementation of Networking Libraries. Through the further developments in a range of different networking libraries and the more in-depth documentation providing new and aspiring developers with instructions on how to implement multiplayer into their applications with ease, multiplayer gaming has become more accessible and enjoyable for users of all skill levels.

Chapter 3 Methodology – Development of BitJump

3.1 Requirements and Analysis

To gain a retrospective of what people would like to see within my application as well external testing to gain insights on any bugs or issues seen within the game, I employed the use of a questionnaire and a website to show off the features the game has to offer, as well as a review box for people to leave ideas on improvement. This would help dramatically within the development as it would more greatly improve the seamless integration by reducing number of bugs or issues seen in play mode. In addition to this secondary research needed to be done to identify the requirements of multiplayer game development as discussed throughout this methodology.

The feedback gained from these means of communication laid as a groundwork for the creation of BitJump, providing insight on the necessary pre-requisites needed for the completion of a fully operational multiplayer title and can be seen in section 3.1.3. Ensuring synchronised movement, enjoyable progression and team collaboration were crucial objectives during the development of BitJump. In addition to this, 'quality of life' contingency plans was also set in place, for example, in the event of a server or client disconnection and how the application reacts to this change. Furthermore, attention will also need to be given to frequent error messages being implemented, this will be a requirement as it would prevent the 'breaking' of the title through means of invalid or null lobby names or unsuccessful connection to the server.

On the other hand, in terms of lobby creation and displaying, I plan to implement several different ways in which clients can connect to eachother. Ideally being through either private or public lobbies. Private lobbies, being a way to separate clients into friends only, is a way that clients can give the unique room code to people only they want to join, thus creating a more exclusive gaming environment. Where on the other hand, public lobbies, once a name has been input for them will be displayed on the public lobby scroll view for all players to see and join.

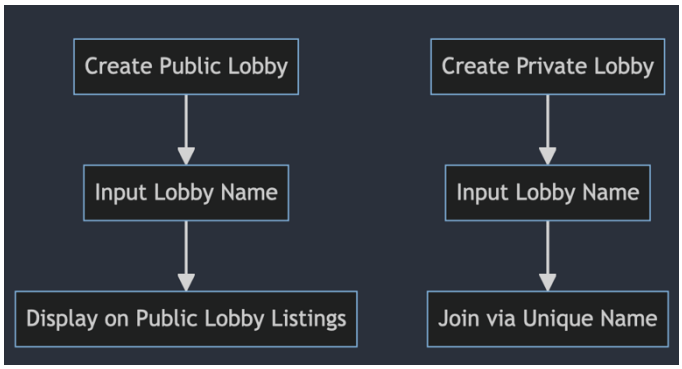


Figure 4 Flow of room creation in BitJump

Subsequently, development would then continue and move onto more game-related mechanics, switching the focus from UI elements to the way in which the user plays throughout the title, alongside the synchronisation elements that would work collaboratively with these features. Considering the idea synchronisation, due to the game being online and collaborative, this would act as one of the most important elements regarding the ideology of realism and immersion players seek to find in-games.

Once initial development has been completed, rigorous testing will be required, this would be another pre-requisite to release of the application, ensuring that the game meets requirements and is at an optimal stage for release. This would involve running unit testing as well as regular prototyping and working off the feedback, ensuring that the game is at a stable state for submission and release.

Finally, the ending stage of the software lifecycle, in game development, would be the maintenance and bug fixing stage. This would be a stage in which, after release, hotfixes and smaller updates will be announced, adjusting some settings and fixing some new-found issues.

3.1.2 Choice of Game development Engine

My chosen development engine was Unity, this was not only based off of my previous knowledge existing in this tool, but due to the effectiveness and number of available networking frameworks I could choose from compared to its competitors.

Comparison of Unity vs Unreal

This comparative table comparing the two giants of the game development industry was curated using the information from Evelyn from Evercast (Trainor-Fogleman, 2024). It would go into detail on the pros and cons of each of the engines, portraying the strengths of each.

Criteria	Unity	Unreal Engine
Ease of Use	Unity is slightly easier to use due to its native C# coding language and workspace layout.	Unreal Engine has a steeper learning curve.
Visual Effects (VFX) Quality	Unreal Engine has a slight edge over Unity in producing high-quality VFX.	Unreal Engine is favoured for its ability to create photorealistic visuals.
Rendering	Unreal Engine has slightly faster and higher-quality rendering capabilities.	Unreal Engine leads in rendering speed and graphic quality.
Animation	Unreal Engine is preferred for its superior animation quality and tools.	Unreal Engine stands out for its animation capabilities.
Scripting	excellent scripting tools.	robust scripting capabilities.
Quality of Support	24/7 support services.	24/7 support services.
Pricing	Unity offers a free version and a Pro version for $75/month.	Unreal Engine is free to use but charges a 5% royalty fee on game sales.

After careful consideration and based on the initial ideas that I had prior to beginning development of BitJump, Unity seemed like the most optimal choice. Due to its shallower learning curve, and superiority when it comes to pricing, it seemed like the best choice for development of an indie-like game.

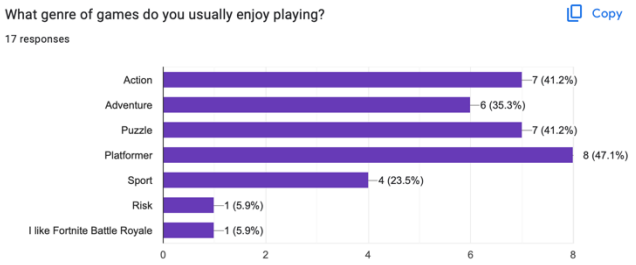
3.1.3 Choice of Game Mechanics

Game mechanics are essentially what makes up a game and the features behind it. To gain an understanding on the genre and type of game I would be looking to develop, various questionaries were utilised and answered by diverse individuals that were present during the testing stages of the developed application. In addition to this, by doing my own research I gained insight into essential features that users seem to find enjoyable. Taking inspiration from retro-like games such as vintage Mario's, celeste or super bomber man, my ideal vision was to create a platformer game with features inspirated from these titles.

Primary and Secondary Research

In order to gain further insight on feature preferences when it comes to. Games, a google forms survey was created and sent out to an audience with 17 responses received. From these responses I was able to conjure an initial idea for BitJump and the features that I was looking to implement.

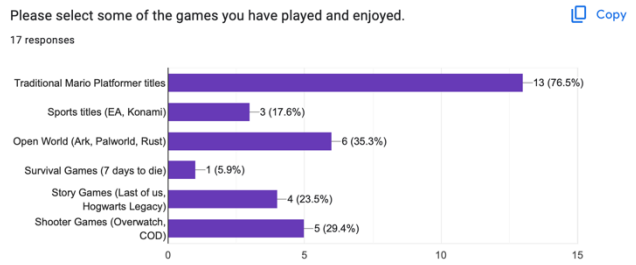
Question 1 and 2: What genre of games do you usually enjoy playing? Please select some of the game you have enjoyed.



The repsonses gained from this question offered a taste spectrum of insightful data, revealing the diverse amongst the gamers that partook in the survey. Whilst a large proportion (41.2%) of the responses expressed interest in action-related games, a larger percentage (47.8%) opted for the platformer response, initially inspiring the development of BitJump.

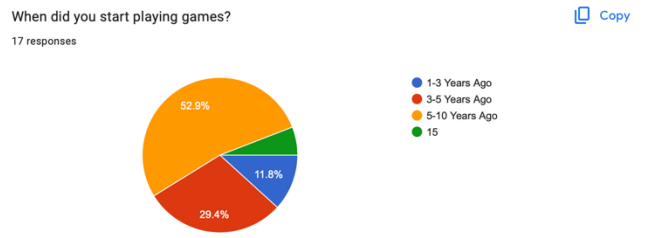
After conducting further research on the realm of platformer applications, it is evident that platformer games will always 'keep fans coming back every update' (BROOKS, 2021) due to the nostalgic feel that they bring to gamers, alongside their challenging gameplay. The platformer genre would be a mix of precise controls, creative design and rewarding progression which is what I look to strive for when developing BitJump. Drawing inspiration from this feedback, my visions for development of BitJump solidified, as well as the initial mock-up designs of the artwork style I was attempting to achieve to provide players with this 'nostalgic feel' Elements such as rigid controls, creative level design, visions of nostalgia key objectives for the success of this game development project.

Like that of previous popular platformer or natively 2D games released such as Stardew valley were researched and provided inspiration on the intriguing 32-bit artstyle, alongside traditional Mario platformer titles being the inspiration for game mechanics and level building. As mentioned in the second figure, 76.5% of responses were inclined towards Mario platformer titles portraying the common, shared enjoyment of users towards these applications, further encouraging the idea of BitJump being a 2D 32-Bit platformer title.



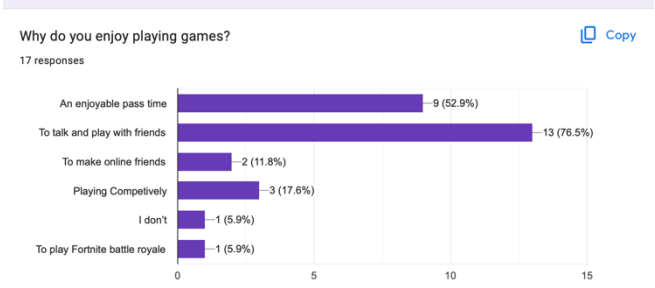
Question 3: When did you start playing games?

This question was designed as means to see whether most of the answers were from avid gamers, or more causal recent gamers, therefore allowing me to get greater insights into my target demographic, providing me with more information on the features I should investigate researching.



Question 4: Why do you enjoy playing games?

The last question allowing me to gain insight into as to why gaming is important to this wider audience. From this verdict it was evident that multiplayer features were of largest interest to gamers, therefore most of my focus was integrating online play effectively. By prioritising multiplayer features and ensuring seamless connectivity, I aim to enhance the overall gaming experience and cater to the preferences of my target audience.



Webpage – Promotion of application

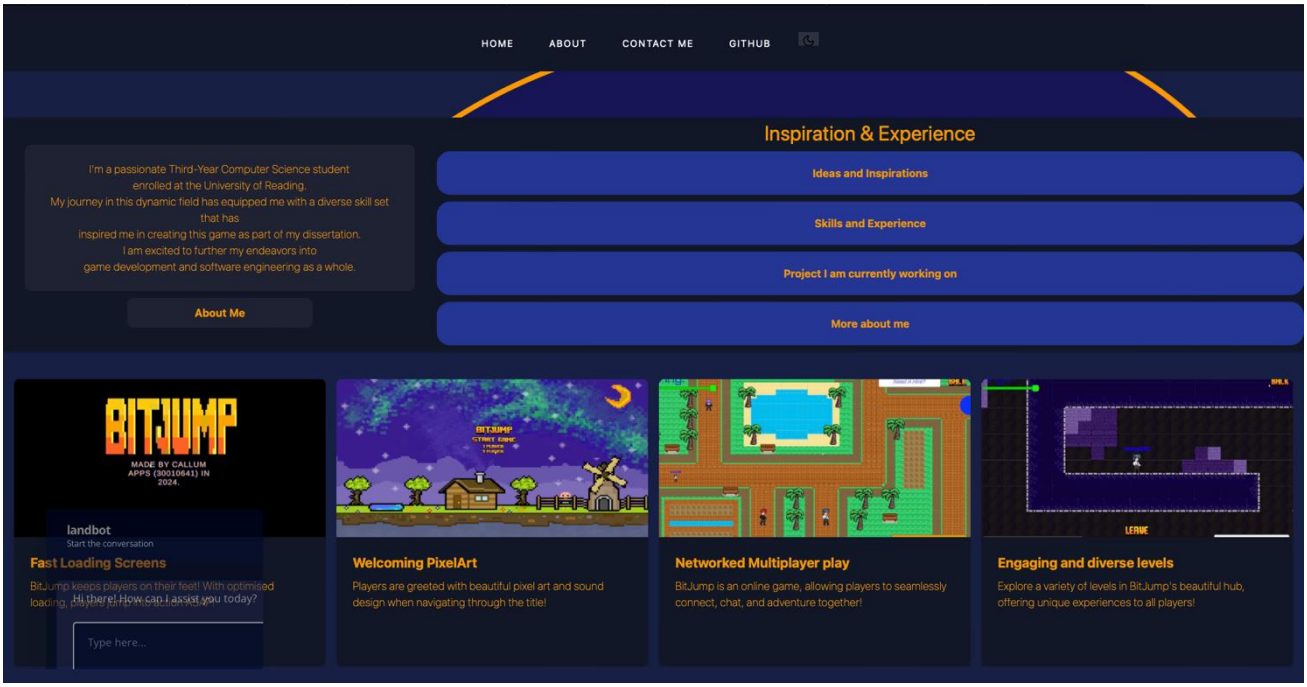


Figure 5 Webpage created to promote features and gain feedback

Choice of Networking Framework

In order to implement multiplayer features into a unity application it is required to gain an understanding on which networking framework to utilise, further research had to take place on the strengths and weaknesses and how the genre in which each genre thrives within. Building upon the previous chapter (literature review) as well as further research into the field, I found that Unity Netcode, Fishnet and Photon were relevant to the application that I aim to develop. Both networking frameworks integrate seamlessly with Unity, synchronising players across a network with ease.

From my previous literature review, it is to my understanding that all provide a client-server framework, with Photon and Unet providing extensive documentation and great community support. As a result of FishNet containing a smaller community and fewer supporting documentation, therefore was not in the running for my development. Moreover, there are benefits belonging to both parties (UNet and Photon), beginning with Unity Netcode (UNet) providing further flexibility when it comes to customisation, allowing for developers to tailor their networking process towards their aims. Netcode provides developers with the ability to determine their own hosting means, compared to that of Photon defaulting to their PUN servers but with the added benefit of matchmaking, (TheNullReference, 2018).

Matchmaking alongside as cost of Photon is advantageous compared to Netcode. Photon provides a ‘freemium’ (Chapter 2) version allowing for users to develop applications and host them on PUN servers with up to 20 CCU, whereas Unity includes a charge of $5 a day when using their hosting platform ‘Game Server Hosting’, which when I tested, resulted in a charge of $185 for the 2 months. Matchmaking included is greatly beneficial, allowing for clients to find eachother with ease once connected to the PUN network and is called simply using one method JoinRandomOrCreateRoom().

Summing up the benefits and challenges of each framework, it is evident that the Photon framework, provides further advantages over the previous frameworks mentioned. Due to extensive documentation, matchmaking features, alongside readable methods being packed into one framework, this was the Networking option of choice for my application.

Ethical considerations of game development

Within Game Development millions of people are captivated, therefore the requirement for ethical design methods are fundamental. It is imperative to design each aspect of an application with ethical means in mind, ‘fostering inclusivity and promotion player well-being’ (Bindumon, 2023). Integrity, transparency, and respect for players rights are essential, incorporating informed consent if the application were to involve collection of user data or infringe user privacy. Transparent communication between the developers and players are essential ethical foundations. Alongside this, the promotion of inclusivity and diversity and two more basic fundamentals of ethical game design, embracing the requirements for a diverse community via means of character customisation and themes

Accessibility

Accessibility within games are a necessity, having the option to cater the gameplay towards every audience regardless of their health or social statuses. This is something I aim to implement within BitJump with useful settings menus allowing those who may struggle with louder noises or a colourblind filter who may have issues related to their sight. These features would vastly improve the ethical considerations and help a more diverse array of players to join the BitJump community.
Overall, the inclusion of ethical considerations are not just a checkbox, they lay the groundwork for responsible game development, embracing inclusivity, transparency and player well being as a core value.

Chapter 4: Design and Implementation

To achieve the goal of developing a fully functional multiplayer title, I opted to integrate the Photon API and library. This choice enabled my game to establish connections among players across a network, utilising the Photon servers. I selected this library based on my prior experience, its affordability, and the inclusion of a matchmaking API. Additionally, its advantages in handling real-time applications proved crucial for a platformer game, where players may need to collaborate to navigate through diverse levels.

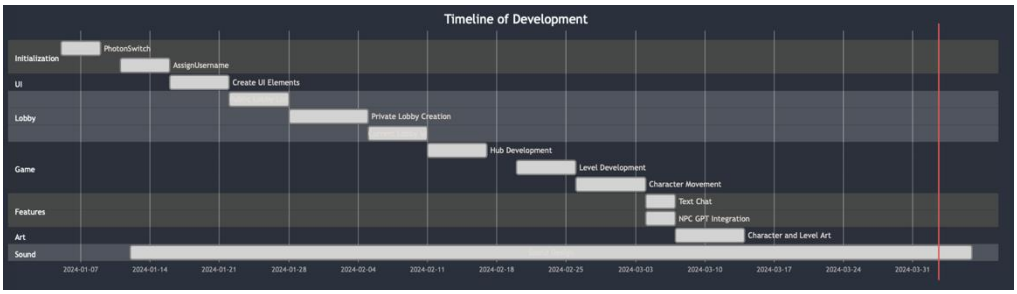


Figure 6 Gannt Chart displaying implementation schedules

The above figure dictates the development journey that my application underwent prior to rigorous testing, this involved the networking integration, game development and sound design.

4.1 Networking Implementation –

Initial setup of PUN –

To begin the setup of PUN and it integration into a Unity environment I had to follow the introductory steps provided throughout their extensive documentation (PhotonEngine, 2024a), in order to reinforce this knowledge provided by their website, useful video tutorials (Games, 2020, Diving_Squid, 2021) and various community forums were present in the development stages.
The initial setup stages for the integration of Photon Unity Networking into Unity is to download the plugin from the Asset Store, then importing it into my newly created Unity scene. Once this step was complete, I needed to configure some PUN settings, such as AppID, which is specifically generated for each new application created on Photons Cloud dashboard.

After the configuration of the settings, I then proceeded to setup my network manager known as 'TestConnect' within my application, utilised to create a connection between clients and the Photon network. This script utilised Photons built in method, PhotonNetwork.ConnectUsingSettings(), this would use Photons pre-built server settings to connect to allow clients to connect to the network. Once thos method has been called, the Photon Wizard then added a PhotonServerSettings to my project, thus saving all previous configurations after ConnectUsingSettings is called.

Within PhotonServerSettings is the option to configure the specific region that clients would connect to, due to my application not requiring a specific region to be used, the default 'Best Region' is utilised which would be used to connect incoming clients to the server providing them the lowest Ping.

After successful configuration of these settings, the next stage was the configure network manager utilising singleton and mastermanager scripltable objects to handle Photon NickNames and Game Version Settings. Scriptable objects were chosen to be used as it would allow me to have a central control over the networking functionalities. This singular access point mean that these crucial settings will not suffer conflicts from other scripts present within the build and effective in version control.

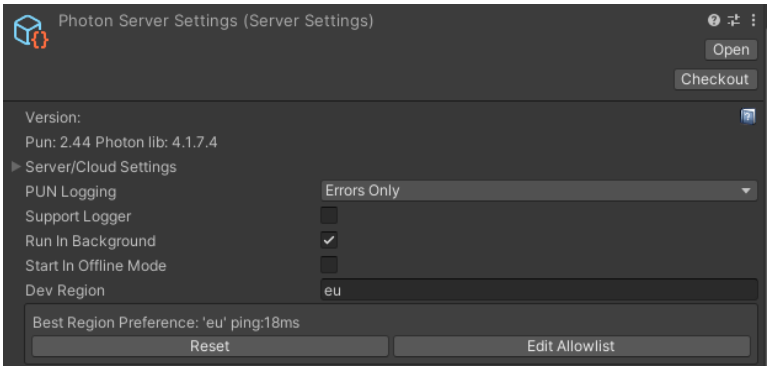


Figure 8 Photon Server settings, holding information such as Region and Version

```
public class TestConnect : MonoBehaviourPunCallbacks
{
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        // Print a message indicating that the connection process is starting
        print("Connecting. . .");
        // Enable automatic sync of the loaded scene across the network
        PhotonNetwork.AutomaticallySyncScene = true;
        // Set the player's nickname using the nickname retrieved from the GameSettings
        PhotonNetwork.NickName = MasterManager.GameSettings.NickName;
        // Set the game version for the application (used for matchmaking)
        PhotonNetwork.GameVersion = MasterManager.GameSettings.GameVersion;
        // Connect to the Photon Cloud or Master Server using gamesettings applied earlier
        PhotonNetwork.ConnectUsingSettings();

    }
```

Figure 7 TestConnect Script, used to allow clients to initally connect to Photon Severs

In order to allow for users to connect to the same game session via matchmaking features, a specific GameVersion will need to be implemented for each build. This ‘GameVersion’ would serve as an identifier allowing for Photon to distinguish between different build versions of the same game. This will be applied when players go to join sessions, Photon will check the ‘GameVersion’ to ensure compatibilit between clients. If the version between the masterclient and the client attempting to join matches, the connection will be accepted, otherwise Photon may prevent this client from joining the session to prevent unsolicited behaviour from occurring. Therefore, throughout development of BitJump, ensuring that the GameVersion was consistent between clients was crucial to ensure a seamless matchmaking process.

Nicknames were also utilised as a means of differentiating between connected clients. This is a useful feature of Photon that was implemented within my application that provides all connected users with unique ID’s which in my case was BitJumper followed by an array of different numbers to tell two players apart. This is achieved via the game settings scriptable, using a serialized field to edit the beginning of the nickname in the inspector as seen in the corresponding figure.

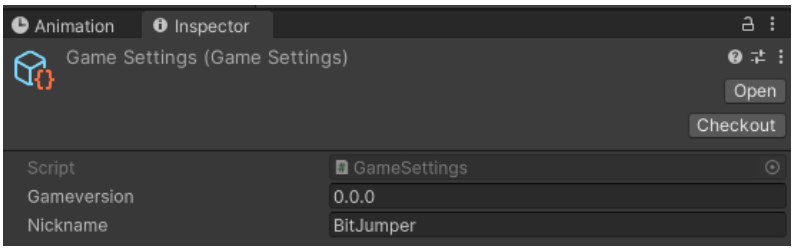


Figure 9 Game Settings within inspector

The player prefab is provided with a Text object just above the selected character, displaying their username preventing any confusion between mismatched players. After completion or quitting of the application, this username is then voided, with a new username being provided on next launch.

```
private string _gameversion = "0.0.0";
2 references
public string GameVersion { get { return _gameversion; } }

[SerializeField]
private string _nickname = "BitJumper";
2 references
public string NickName
{
    get
    {
        int value = Random.Range(0, 9999);
        return _nickname + value.ToString();
    }
}
```

Figure 10 Implementation of Game Verison and Nicknames

Upon connection to the Photon network, a player will be assigned a nickname, utilising the PhotonNetwork.Nickname command. This step occurs in my application through the nickname being assigned to the PhotonNetwork from the gamesettings discussed in the previous figure. These methods are called during the initialisation phase of the game, with the loading screen delayed until a user is connected to the network, to prevent any issues occurring whilst navigating menus.

```
// Set the player's nickname using the nickname retrieved from the GameSettings
PhotonNetwork.NickName = MasterManager.GameSettings.NickName;
// Set the game version for the application (used for matchmaking)
PhotonNetwork.GameVersion = MasterManager.GameSettings.GameVersion;
// Connect to the Photon Cloud or Master Server using gamesettings applied earlier
PhotonNetwork.ConnectUsingSettings();
```

Figure 11 Retrieval local settings and assignment to Photon

Implementation of usernames was a crucial aspect of BitJump, providing distinct nicknames to tell character models apart in scenarios where they look the same it would’ve been harder to identify the current user. In addition to this, nicknames were utilised during the main menu when joining rooms. This would allow users to see when new players join based off a new unique ID being added to the scroll view.

Creation/Joining of Multiplayer Lobbies –

Due to the nature of the task at hand, this was an important feature that is a crucial feature of networking titles. Without this feature, clients would not be connected successfully to the same lobbies, potentially disrupting, or halting all multiplayer experience.

To achieve this milestone, research was required on the aspects of Photon Revolving around the joining and creation of Photon lobbies. This involved the comprehensive view of going through multiple documentation pages provided by Photon in a means to find the best practise of efficiently integrating lobby management into my BitJump application. In gaining an insight of Photon's functionality, I was able to develop a robust system that would streamline the process of connecting players and maintaining synchronisation.

Prior to players joining or creating lobbies, a connection to the Photon Network would be required as well as a NickName. If clients were to create their own lobbies, rather than using the inbuilt matchmaking feature, a lobby name or ID would be required thus, displaying a dismissible error message if null. This would then use the OnJoinedRoom() Photon function to determine on what is displayed on the UI, whether the client is the Master Client (host) or the client.

If the matchmaking feature is utilised, and the player chooses quick play, they would be assigned a random lobby which, if there are not any active, a lobby would be created and allocated a random ID allowing for the joining of other players. In addition to this, private lobbies would have been implemented into my application, allowing for users to create lobbies with their unique ID's that would not be joinable from the public server list, rather through the joining of lobbies via typing in the unique ID.

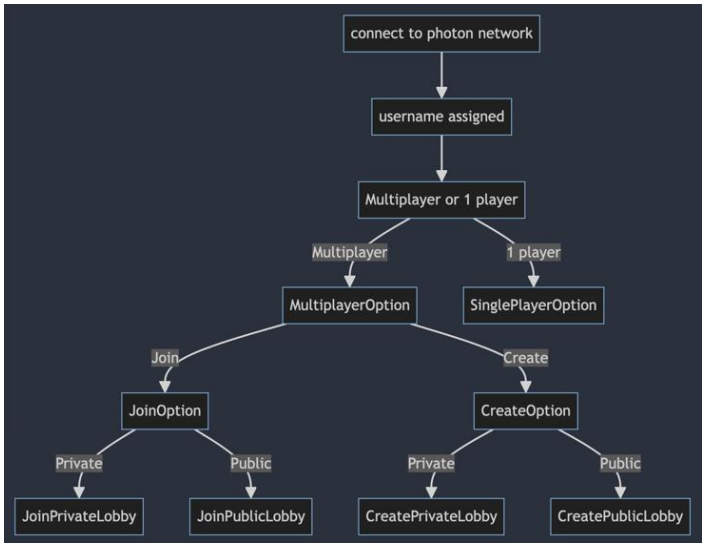


Figure 12 BitJumps flow of game

Servers were one of the initial features I chose to implement that would take use of Photon and its ability to allow users to create servers and give each server its own unique username. This was done using Photons inbuilt CreateRoom() method. After the script handling server creations had been made, I then had to move onto implementation, this required the creation of a button prefab with the script attached and the TMP Text of the button to be assigned to the Serialised fiel d within the script component. Meaning that the text will change to the created servers name when prefab is called. The text is also altered to the number of slots left in the lobby, using Photons.Maxplayer() command to do so. To display the list of multiple servers I used a scroll view object, allowing for players to scroll down the list and find an open server.

```
RoomOptions roomOptions = new RoomOptions();
roomOptions.MaxPlayers = 6;
roomOptions.IsVisible = true;
roomOptions.IsOpen = true;
roomOptions.PublishUserId = true;
HostOnlyButton.SetActive(true);
PhotonNetwork.CreateRoom(roomNameInputField.text, roomOptions, TypedLobby.Default);
```

Figure 13 Creation of Room Method

Matchmaking

Photon simplifies the matchmaking process by providing easy-to-use call-backs, components for synchronising GameObjects, remote procedure calls (RPCs), and chat features, allowing seamless communication between players. Matchmaking is achieved through PUN's **'JoinOrCreateRandomRoom'** function where if a room is found it will be joined, otherwise one is created. [5]

Matchmaking would require a range of different variables to operate correctly, this would involve all clients being on the same AppID, App Version and region, typically set to 'Best Region', alongside unique NickNames being assigned to all users via Game Settings. Handling Matchmaking failure will utilise **'OnJoinedRoomFailed'** command from the Photon Library, this would return the error code via its parameters **'short returncode, string message'** and would usually create a room rather as an alternative to joining [5].

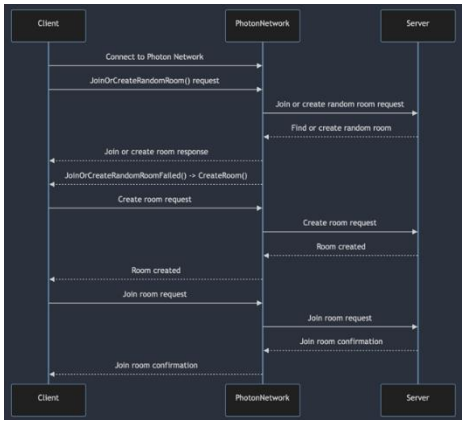


Figure 14 Visual depiction of Matchmaking

Private matchmaking (playing with friends) is another feature that can be achieved via Photons library.
This can be achieved through creation of a unique room code either pre-set by photon or allowing for text box input and making it invisible to all users using **'roomOptions.IsVisible = false',** ensuring that the lobby is only available to those who know the password.

Slot reservation as guided by Photon's matchmaking guide, this would be the process of blocking access to lobbies, unless a certain Player ID attempts to join. This can be achieved via using the expectedUsers parameter after the room has been created using:

$$PhotonNetwork.Createroom(roomName, roomOptions, typedLobby, expectedUsers)$$

This would then allow for users to join that exist in the allowed array of userIDs. This feature is particularly useful for team matchmaking scenarios. [5]

Multiplayer implementation and Synchronisation –

PhotonView Implementation

To ensure seamless synchronisation over a network, each component that would require syncing needed to have a PhotonView component attached. This component would be utilised as a way to ensure the seamless synchronisation of players and objects. This component would ensure that the 'gameobject is the same across all clients. Meaning that if a given barrel has photonViewID = 1, anything we do to that barrel in my instance of the game can by synced with the same barrel in your instance of the game' (Martinez, 2021).

The PhotonView component within my application would act as the 'bridge between a local and a networked game state', thus handling the serialisation and deserialisation of data. This would ensure that all changes made to synchronise objects are distributed accurately to all clients currently connected to the sevrer. PhotonView would also facilitate the use of Remote Procedure Calls (RPCs), utilised to allow players to interact with networked gameobjects in real time, being a fundamental aspect of multiplayer gameplay.

The PhotonView component within BitJump utilises the *Unreliable on change* observe option to ensure that all data is sent regardless on checking if received, therefore being acceptable for applications where frequent updates are more important than missed packets. (joebain, 2014) This option aligns well for BitJump due to the constant movement of characters necessitates the rapid updates to maintain synchronisation between local

and networked game states. Through the employment of this observe option, Bitjump optimises its network bandwidth, minimises latency and enhances the overall multiplayer experience. Through the prioritisation of frequent updates over ensuring every packet is received, BitJump ensures gameplay remains smooth and responsive even in fast paced movement. As a result, clients can enjoy seamless, immersive gameplay characterised by fluid gameplay and dynamic interactions.

PhotonTransferView, PhotonAnimatorView, RPCs and NetworkRigidBody2D

In addition to this component, PhotonTransferView and PhotonAnimatorView components were utilised, this would ensure that objects transform, rotation, and animations are also transmitted across a network. These components are recognised by the PhotonView component and placed within the observed components section. These components were essential within BitJump, contributing towards overall immersion. Due to my application being a platformer, players will need to interact with one another to ensure timely completion of each of the levels therefore both vertical and horizontal movements are crucial features requiring synchronisation. To reinforce this method, the NetworkRigidBody2D was also utilised, this would be the process of replicating a Unity RigidBody2D state from the NetworkObject.StateAuthority to all other peers. (Exit-Games, 2024)

The AnimatorView component is utilised to ensure that all animations are synced across the servers. This was utilised in the BitJump application to synchronise animations between clients. Alongside this, remote procedure calls was utilised to ensure all clients receive the method called. For example, when utilisting the RPC ‘All’ when a player initiates an attack or dies, an RPC call is utilised to trigger the corresponding attack and death animations across all connected devices as seen in the corresponding figure.

These methods of synchronisation were implemented within BitJump to ensure visual consistency of the gameplay experience therefore contributing towards the overall immersion that all players receive. Transform and animation real-time synchronisation are crucial features in multiplayer applications, and further reinforced by the use of RPC calls, further complimenting the efficiency of the chosen Networking Framework.

Text Chat, Player Feed and Ping Counter -

This was another essential part of a multiplayer title that users seem to utilise heavily, allowing them to communicate within the title, boosting the effects of teamwork when completing the array of levels that my game has to offer. Within Photon, text chat was a relatively simple feature to implement within my application. Various audio-visual materials streamlined the development process for this feature, allowing for me to gain further insights into its development.

Figure 16 Method to initialise chat features

A ping counter was also a crucial feature of Photon, this would allow for clients to understand how fast their ‘computer is communicating with a gaming sever and receiving a response’(Grind, 2023), which enabled me to gain an idea of the most optimal regions for clients. This was included in the build of BitJump within the top left of the UI.

Figure 15 Ping Counter

In addition to this, the player feed allowed for the masterclient to see when new clients join the currently open room, this would be a useful feature as it would mean that people can greet each other and there would be no surprises for when another client is active within the next

4.2 Game mechanics features implementation –

*U*ser Interface:

The user interface (UI) within BitJump was a crucial factor due its ability to 'set the scene' for the application, which in my case, was mostly 32-bit artwork. A variety of different artwork was created in preparation for development to begin as seen in figure 16. Each of these buttons created in pixel art, would hold their own unique features, with a large proportion involving the use of Photon.

Figure 17 BitJump UI Art

Splash Screen:

The splash screen, seen in figure 17, served the purpose of connecting the user to the PhotonNetwork and assignment of username prior to allowing any user interaction. These loading screens would effectively manage the delay, and prevent any errors being thrown from an invalid connection, therefore reducing the chance of application crashes or unresponsiveness.

Figure 18 BitJump splash Screen

Start Screen:

The initial UI that would greet users would be the starting screen, swiftly followed by the mode selector UI seen within figure 18. This would be the interface that allows for users to see their usernames, currently open public servers, matchmake or create their own private/public lobbies. This is the scene that would handle all creation , joining and matchmaking into lobbies as discussed previously within the networking integration section. Public lobbies will be displayed through the use of a prefab being called into the content of the scroll view once after the CreateLobby on click even is executed, allowing users to choose a lobby to join.

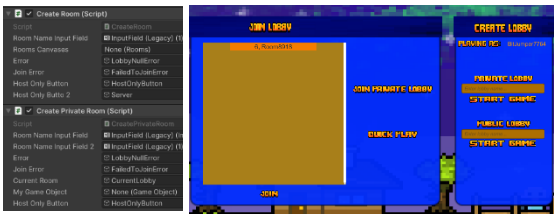


Figure 18 BitJump Mode Selector

Player List Screen:

Similar to the server list, the player list screen seen within figure 19 will be used to display all current players within a lobby. This was achieved utilising the OnPlayerEntered room method that will run once a new player has entered the room. This method would then instantiate the player prefab to be placed onto the scroll view and alter it towards the players information that recently joined. Summarily, once the player leaves the opposite method is run, OnPlayerLeftRoom, which finds and destroys the index of which the player was utilising.

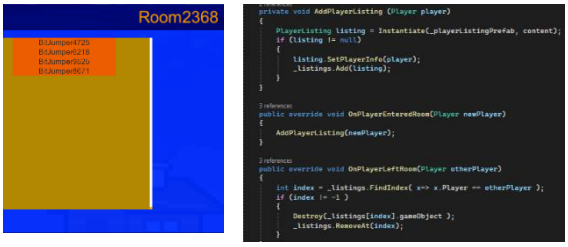


Figure 19 Player Listings

Pause Menus and Loading Screens:

The pause and start menu included the settings tab to allow users to adjust their volume, resolution, save the game or graphics quality. This was required as it would allow for players to tailor the application specifically for their device, for it to run most optimally. This was achieved through a singular, modular script that was embedded within the buttons displayed on the Start menus, including volume, graphics, and resolution. This script was then further utilised within pause menus for the same features. Example use of colour-blind filter setting:

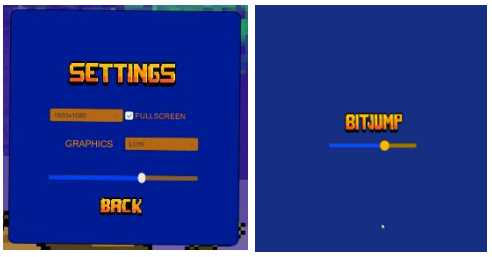


Figure 20 BitJump Loading Screen

Figure 19 Colour Blind Filter, Achromatopsia – Protanopia - Protanomaly

Level Selection and Scene Management –

Level Selection within Unity would be another pivotal feature that is integrated into my application, facilitating the choice of players, and allowing for further engagement. This functionality would seamlessly operate on both the Starting Screens and the Hub as seen in the figure, displaying each of the separate levels through the NPC characters circled in figure 21.

Figure 21 Level Selection Hub

Photon serves as a fix to this issue, providing a robust platform for achieving synchronisation whilst also ensuring peak performance and stability across the varied levels. By incorporating Photon's features such as `PhotonNetwork.AutomaticallySyncScenes(true)` and `Photon.LoadLevel()`, we were able to ensure that clients would be synced automatically surpassing Unity's built in Scenemanager capabilities.

This was achieved within BitJump through the use of OnClickEvents, therefore once the masterclient interacts with the Start Level button, Photon.LoadLevel() seen within figure 22 will be called and run, transitioning all clients in a synchronised manner. This process includes the display of a loading screen until the scene is successfully loaded, enhancing the ideology of seamless transitions.

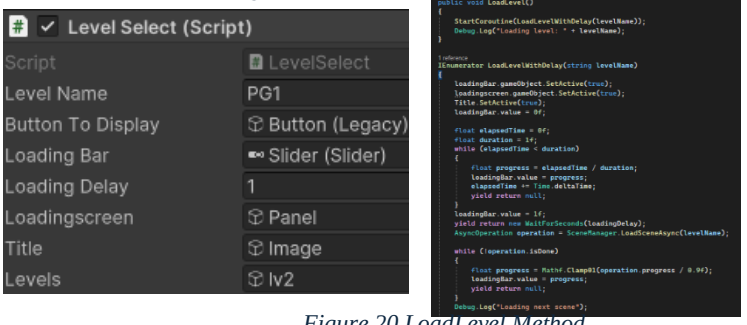


Figure 20 LoadLevel Method

NPC Dialogue –

NPC Dialogue was another essential feature within my application, this would enhance the sense of immersion provided to users, allowing them to feel more engrossed within the title. To achieve a unique feel for each playthrough, I decided to implement another script that would use the ChatGPT-3.5 Turbo API to develop random NPC dialogue.

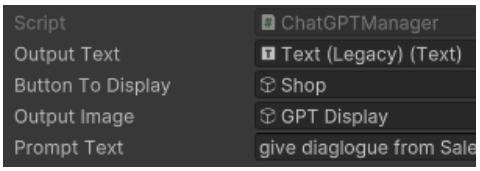


Figure 21 ChatGPT dialogue prompt in inspector

Integration of ChatGPT-3.5 for NPC dialogue in game development

I had chosen to implement this feature as it would provide the game with a unique feel throughout the playthroughs, allowing the player to gain tips throughout the levels with the alternating NPC's seen throughout, as well as the Hub as discussed previously.

Within the realm of game development, the utilisation of the natural language processing model OpenAI developed would be a revolutionary technique for the enhancement of non-playable characters interactions and dialogue systems. Using the advanced capabilities provided by this model, NPC dialogue can transcend scripted application and offer players more immersive, and unique options whilst gaming.

Figure 22 Shop Dialogue Example

Technical Implementation:

To seamlessly integrate the ChatGPT API provided by OpenAI to my application. My game would be provided with the ability to communicate and send prompts to the model to receive relevant quotes onto the specific tasks at hand. Each tailored towards the specific NPC via Serialized fields within Unity with an example being the shop seen in figure 24 with the prompt of 'give dialogue from Salesperson: welcome to me shop purchase goods to help you along the way here! (limit to 15 words)'.

For this feature to be successful, an API key was required from OpenAI's dashboard and stored within a JSON file [ ] which would connect to my account with OpenAI, this would then charge per request based on the number of tokens utilised for each prompt, totalling to around 0.001 pence per request. This is an effective feature to include in my application due to it ensuring optimal performance and responsiveness, due to it being an asynchronous technique, therefore occurring within the background of my application.

Networked Character Setup –

Within platformer games, character movement is an integral feature, contributing massively towards the overall enjoyability of the application. This had to be carefully considered to find the most suitable movement mechanics for my application.
To do this, I investigated previous platformer-esc games and borrowed ideas from games such as celeste.

To implement such movement, multiple scripts were required, thus needing to be observed by the PhotonView component to enable seamless synchronisation. As depicted in figure 19, significant changes were made to player movement throughout development, initially player movement was basic, relying on solely left and right user input. However as development progressed more complex features were added, to enhance the realims and responsiveness of playermovement.

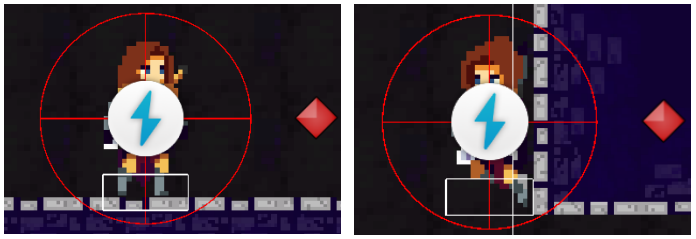


Figure 23 Depiction of Character Movement

Raycasting:

Figure 19 and the collision script in figure 22 portrays the implementation of recasting to determine whether the player was grounded or on a wall. Raycasting in game development is the ideology of simulating the behaviour of rays or beams in s specified direction, therefore implementing this into my character movement allowed me to create rays from the characters position. Downard rays were utilised for ground detection whilst horizontal rays for walls, therefore ensuring that the players relationship with the environment is consistent. This was required within BitJump for a couple reasons, the first for animations, for instance when a player isWalled() = true, the Update method will check if the players Photon Component is specific to them, then play the animation for all clients to see, alongside allowing wall jumping dependant on what wall they are touching. Followed by the isGrounded() method, which would check if the player is touching the ground, determining whether jumping is possible.

Figure 25 Representation of IFrames

Figure 26 Player Attack

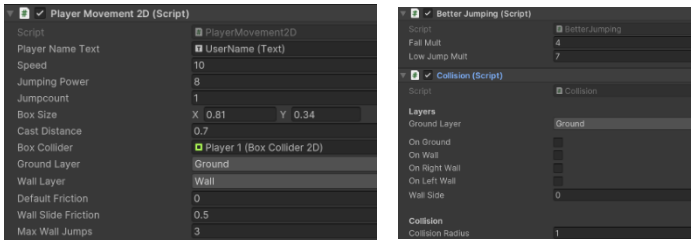


Figure 24 Player Scripts

Player Attacking and Damage:

Depicted within figure 20 and 21, the player attack and damage methods are displayed. Attacking was an integral feature to implement within BitJump and utilised a ‘Slash’ Component prefab. The playerAttack script would find this prefab once loaded into the game using the FindObjectOfType method built into Unity. If the player were to press the mouse button to attack, the application would check if the player prefab is grounded and is not walled prior to this function being executed. If so, the players photonView component will then be checked to ensure it is theirs before a slash is sent flying at enemies and the animation is played.
Player Health is a component within BitJump that utilises IFrames and RPC calls for all animations. IFrames (Invincibility Frames) were implemented to ensure fairness

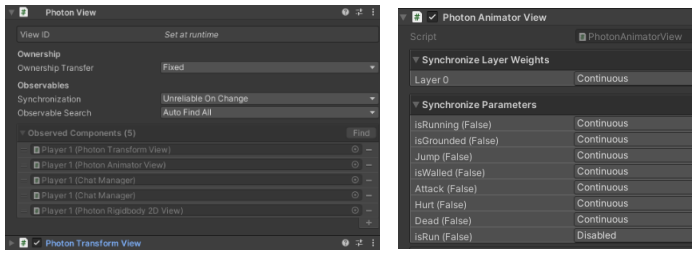


Figure 27 Player Photon Components

Figure 28 Player death RPC Call

within the game, providing the player with a short period of invincibility as seen in figure 20, to ensure that players do not die too quickly. In addition to this, when a player were to take damage, the playerHealth script will check the players health variable and determine which RPC to call. On the other hand, if player is not dead, health with be decreased by one and the player hurt animation will be called via RPC calls, else the player is dead and the death RPC will be called portrayed in figure 24, halting all player movement, and playing the death animation for all clients to see.

Player Photon Components

Photon Components were a crucial aspect of the PlayerMovement2D script, as briefly mentioned in previous sub sections, this is utilised within the player script to synchronise all animations and alterations to a players transform to all clients currently in the room. Figure 23 displays the components utilised on the player prefab with all animations being marked as 'continuous' for most accuracy (Games, 2020), whilst also displaying that animations and transforms are within the observed component section of PhotonView. In essence, the use of these Photon Components within the player scripts ensures seamless synchronisation of animations and transformations across all clients.

Enemies –

Within BitJump Enemy movement is visualised through the use of Gizmos, and raycasting similar to that of the player character. The enemies provide a sense of danger throughout BitJump, being able to damage players and kill them if they damage the player enough.

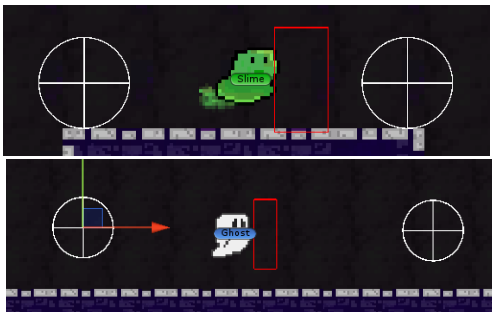


Figure 29 Enemy Movement Example

Enemy Player Detection (RayCast and Gizmos) –

Within BitJump enemies move in a way that they 'patrol' within a certain range. This was achieved through the use of two game objects used as waypoints for the enemies to patrol between, The utilisation of raycasting is also involved in enemies, Physics2D.Boxcast, is implemented to detect if there is an obstruction between the enemy and a gameobject containing the player tag. This raycast is in the direction that the enemy is facing, and if a player was to enter that box, the PlayerHealth variable is retrieved and reduced by the amount of damage that the enemy can do, else the enemy will continue patrolling as normal.

The range that the enemy can attack from is determined via the raycast method using boxcollider bounds and range, which can be altered in size via the collider distance and range serialized fields as seen in figure 28. The range and points are visualised through the utilisation of Gizmos and was useful for debugging during development stages.

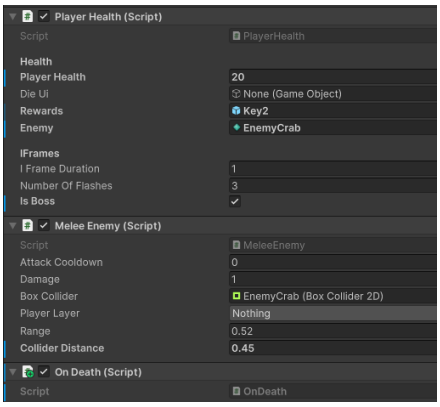


Figure 31 Enemy Scripts

Photon Components

Enemies were also required to use the MonoBehaviourPun component as well as a PhotonView component to sync its transform. In order to sync the enemies death and animations, another RPC illustrated in figure 28, was utilised within the playerhealth script, starting that once the enemies health reaches 0, set its rewards gameobject active and the current enemy gameobject inactive, therefore synchronising an enemies death across all clients.

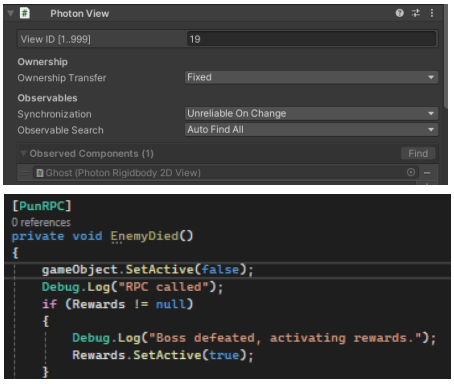


Figure 30 Enemy Photon Components

Procedural Level Creation –

Procedural level creation is a feature in which my application would support. Within the final stage, the level has been procedurally generated utilising a specific script and a Rule Tile to place tile maps onto the scene in the correct format. When developing the procedural generation script, I utilised serializable fields for each of the parameters determining the look of these maps, such as Width, Height, Smoothness and Seed seen within figure 30. These parameters can be altered until finding the perfect map.

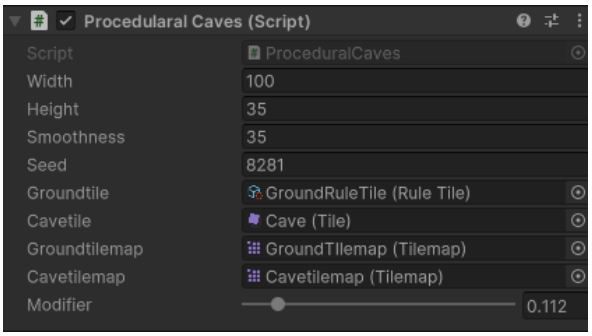


Figure 32 Procedural Generation Parameters

This was implemented within BitJump to ensure that each level is different, due to 'algorithms generating more diverse content'. (Ikeda, 2023)
A visual representation can be viewed in the corresponding figures below, portraying the variables in different states.

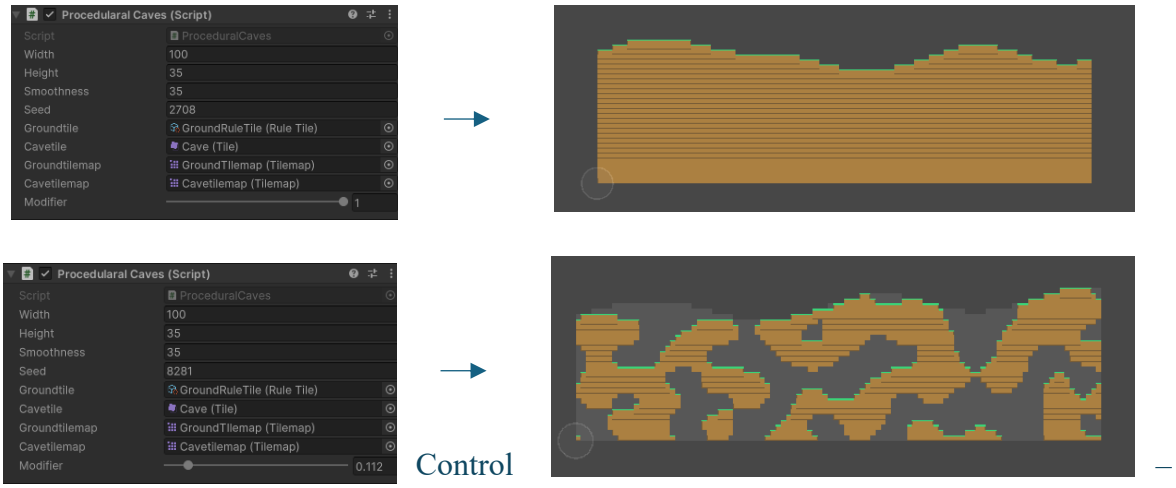


Figure 33 Different configurations of Procedural levels

Version Control –

Throughout development, Unity Version Control was used as a way to save my changeset on the web, with access to the full game version from anywhere. This was useful in situations where I was developing on an external Personal Computer (PC) that was not my central PC. In addition to this, the practicality of Version Control Software (VCS) what its enhanced ability to 'journey back to earlier versions' which was utilised when making errors during development. (GitLab)

In order to submit, I began to commit working builds to GitHub, this was due to Unitys system not having the option to publicly share the Git URL for other developers to view. This involved the packaging of Unity into a new repository and pushing from the command line. A useful readme.md file was also included for simplicity of downloading the entire game file or just the zip required to run.

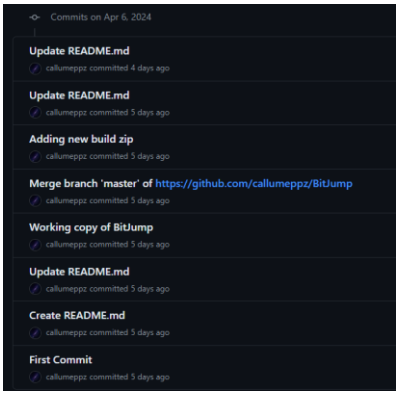


Figure 34 Example of GIT VCS integration

4.3 Testing and Validation

For the testing phase of my application, I opted for the Unit Testing method to systematically evaluate each feature independently. This approach ensures that individual components are robust before conducting comprehensive tests involving all features simultaneously.

Continuous Prototyping and User feedback

User feedback is an integral component of game development, the insights retrieved from beta-players that test the development version of applications provide valuable feedback that can assist in enriching the overall experience that the game provides.
Throughout development of BitJump, I ensured to focus on iterative prototyping, this allowed me to get feedback on individual features that I added and work off this to ensure that they are resilient to any user error.
Testing and quality assurance efforts were further progressed past the initial development, this would be through consistent prototyping with corresponding unit testing. Through this monitoring, player feedback and playthrough sessions, developers such as myself can refine Rigid Body and Collider parameters, until implemented most optimally, elevating gameplay experience dramatically.
Through the incorporation of extensive testing and quality assurance techniques, technical and networking robustness will be displayed by the application. Prioritising immersion would be a crucial feature within the virtual world, which multiplayer synchronisation and overall physics would largely contribute towards.

Unit Testing

This testing method would involve the evaluation of individual techniques displayed throughout the application as well as validating multiple components connection from the client to the server. The goal is to guarantee seamless functionality and connectivity between clients, ensuring synchronisation of animations, movements, and features such as chat and gamertags.

Connection To Photon Network

Initial Loading Screen:
This required its own unit testing due to it being one of the fundamental ways in which players would connect to the photons network, this loading screen would delay the user based on the amount of time it would take for their device to connect to the servers. Therefore, preventing the game from crashing when the user attempts to create a lobby. This was tested for a number of users, each connecting to the network before gaining access to the main UI.

Player NickName:
This check was required to ensure that each player was assigned their own Nickname using Photons inbuilt function and after OnPlayerConnected() is set to true. This Nickname will be unique to each user and would start with BitJumper followed by their player specific ID. This should be individual to each user and should not be the same between players to avoid confusion or connection errors.

Player Synchronisation:
This test was also essential to ensure that Players, as well as external factors such as enemies, chat features, NPC Dialogue, sound effects and scene transition. This test is a crucial requirement as it would determine how seamless the game can be, affecting the enjoyability and usability. This was tested using a variety of concurrent users all connected to the same server to see how the application would react to this. The results of this test were positive, with all users connected successfully, with unique usernames, ability to use the text feature and scenes were synced. The only issue found was players spawning on top of each other causing some instability, however this was fixed by making spawn a random value between Min X and Max X, to ensure that spawns were randomly generated within a range

Room Creation and Joining UI:

After a successful connection to the PhotonNetwork had been extensively tested and established correctly, I then needed to move onto individual testing of each of the Room Creation and Joining functionalities, ensuring their operating consistently. In order for this testing to become viable, two or more instances of the Unity application needed to be run concurrently. Users should then be granted the ability to create lobbies either utilising the Private, Public or Matchmaking features built into BitJump.

Public Lobby Testing:

The initial room creation test was the implementation of public lobbies and their display onto the scroll view UI for lobby listings. This required meticulous testing to ensure that lobbies were consistently displayed, provided the correct error handling messages, and illustrated the correct names/playercount regardless of the second clients application state during lobby creation.

The first of many issues that I faced was an issue regarding room creation and its visibility. When testing the creation of rooms, I found that they were not being displayed on the scroll view consistently, especially when the second clients game was not yet launched. The fix for this issue required a separate script to fetch currently open rooms on launch of the application, ensuring that all available lobbies were consistently displayed to users, regardless of their application state at time of creation. The provided script validates the connection to Photon, within the start method then utilises the OnJoinedLobby() callback method to ensure that all room listings are active, providing users with access to multiple public lobbies.

Private Lobby Testing:

Furthermore, in order to enable private lobbies to operate correctly, an alternate createRoom method was required. This would be to prevent the lobby from being displayed alongside the public lobbies on the scrollview. In order to stop this from happening, Photons built in method isVisible was utilised, hiding the lobby name and description from those that do not have the join code. This then required a separate UI element seen in figure 35, that will only be activated using an OnClickEvent, setting the UI to active when true.

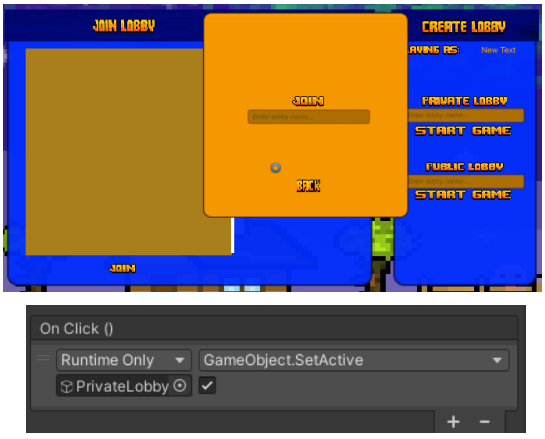


Figure 35 OnClickEvent for private lobby UI

Matchmaking Testing:

Similar to that of public lobbies, matchmaking needed to be meticoulsly tested to ensure that the implemented Photon method of PhotonNetwork.JoinRandomRoom, is operating correctly and the OnJoinRandomFailed callback method is effectively catching any errors, and instead creating a public room with a random name. In addition to this testing, testing that only public lobbies can be joined was also necessary which would involve the altering of code to ensure that matchmaking filters out rooms that are not visible (private).

Creating lobby: Moreover, each lobby creation method required error handling to be tested in a variety of ways from leaving the text box blank to filling out the text box and ensuring that the player count is correct. If the user was to leave the create lobby input blank, an error message UI would be thrown stating that 'Lobby Name is required' to be created.

Joining Lobby: This is another element of my application that would require unit testing individually. This was tested meticulously, from where the lobby names should be placed to what happens if a lobby wasn't selected prior to utilising the 'Join Lobby' button. If a lobby is not selected, another error message would be thrown, stating that a Lobby needs to be selected, and if in any case it allows the user to join a just closing, closed, or invalid lobby a leave button will be displayed at the bottom of the screen, meaning players can simply return to the Join/Create lobby screen.

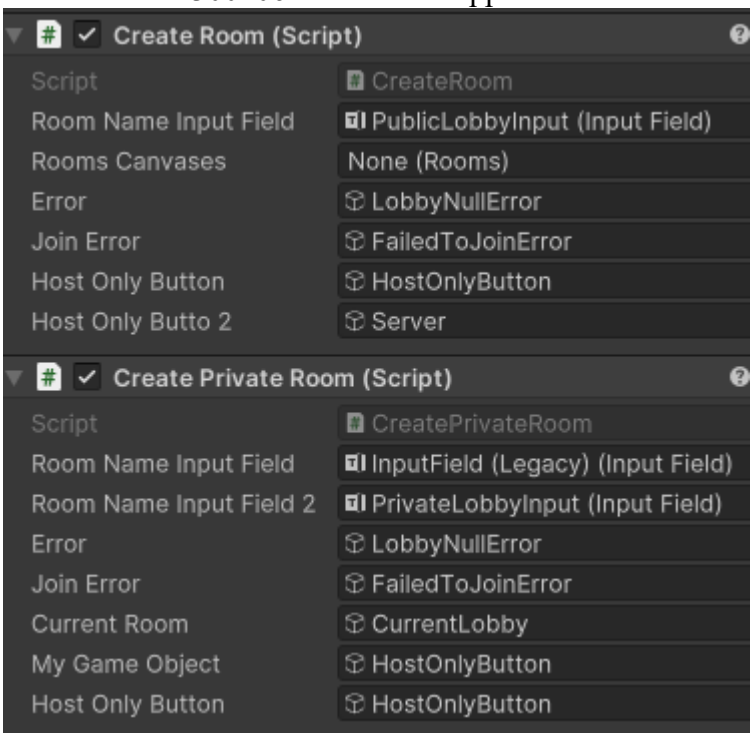


Figure 36 Inspector View of creation of room setup with Error Handling

Starting Lobby: After and if the creation of these lobbies are successful, the clients will be greeted with a player list and start game button. The player list was tested with several players ranging from 1-6, each displaying the individuals gamertags (Photon NickNames).

Overall, meticulous testing was required to ensure that this crucial feature operates correctly within BitJump due to it being the groundwork of client connectivity. By rigorously testing room creation functionality and its associated error handling mechanisms, BitJump provides users with a robust and seamless multiplayer experience without the risk of game crashes.

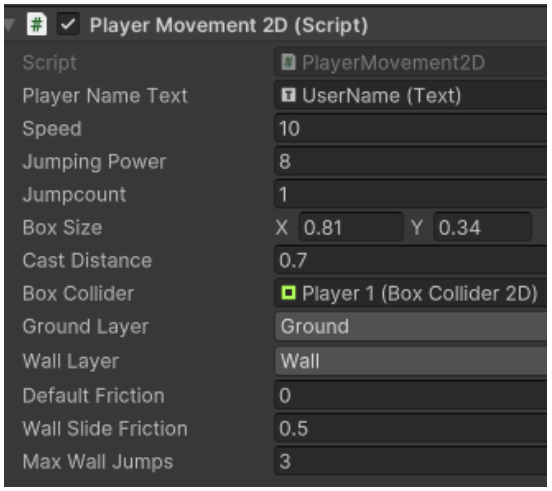
Spot Check Design:

- **Loading Screens:** These needed to be spot checked to ensure that they display whilst transitioning between screens, preventing from game crashes from scenes being loaded before Photon is connected.
- **Main Menu:** Main menu canvases also needed to be spot checked to ensure that all items on screen scale correctly with screen size.
- **Hub:** Ensuring that all levels are paired to the corresponding NPC, along with NPC dialogue and 4 directional movement working efficiently.
- **Audio Features:** Ensuring audio is matched up correctly to each scene/character to further bolster the idea of immersion.

Movement Unit Testing:

Walking:

In multiplayer scenarios, walking mechanics between players is a crucial requirement needing to be synchronised. Unit testing can be utilised in this scenario to verify that the PhotonTransformView component is behaving correctly, ensuring that movement updates are correctly transmitted over the network to remote clients. In addition to this, animations also required testing, utilising the PhotonAnimatorView and trailing each different parameter as well as testing whether the use of RPC's are necessary for this were carried out. Furthermore, movement speed was a factor determining the enjoyability of the application, multiple tests were involved in the discovery of the optimal movement speed before settling at a speed of 10 as seen in figure 37 illustrated by the serialized speed variable. Thus providing players with a balanced, synchronised multiplayer experience.



Jumping and Wall Jumping:

The jumping physics were another feature in my game that required attention. An example of this was to provide the best user experience, it would be to use progressive jumping, with the height based off the amount of time the user has the space or up button pressed. With a jumping height of 14, the progressive jumping would allow for it to build up to this value, dependant on whether the user is touching the ground and how long it is held for. Specific testing was required to ensure that all levels within the title include the 'ground' tag, otherwise jumping would not operate correctly and the IsGrounded function would not operate.

Like jumping, the wall jumping functions would need all walls to include the 'wall' tag, allowing for the function to recognise the player is wall jumping rather than the normal jump, changing jumpingForce and the current animation. This would be a lower JumpForce, adding a sense of reality due to wall jumping being a harder obstacle than normal jumping. Unit testing was a requirement of both of these features to ensure that the Jumping power, friction, and Max Wall Jumps were sufficient to complete all levels without getting stuck. In addition to this, after completion of the first test, testers found that vertical movement was not being synchronised correctly, therefore further research had to be conducted in this area finding that the players update function needed to utilise the PhotonView.isMine method to ensure that the player is using its own PhotonView Component, alongside this the PhotonRigidBody2D was used therefore syncing vertical movement perfectly. (jeanfabre, 2015)

Player Attacking and Death:

Attacking needed to be tested to ensure that the player was able to successfully reduce the health of other enemies, this can be done via weapons and power-ups picked up throughout the maps and the damage needed to be tweaked until finding the best value which wouldn't make killing too hard, or too easy. This required unit testing to ensure that the attacking and death animations synced correctly. This is stage is still undergoing testing, with the animations syncing at a 90% rate, with some exceptions of the idle animation still displaying upon death. Whilst further research is being conducted within this area, RPC calls are currently implemented in attempts to synchronise these animations effectively.

```
[PunRPC]
0 references
private void PlayDeadAnimation()
{
    // play death animation
    anim.SetTrigger("Dead");
    /** stop all player movement and invoke respawn function
    for all players **/
    body.velocity = Vector2.zero;
    body.freezeRotation = true;
    dead = true;
    Invoke("Respawn", 10f);
}
```

Figure 37 RPC Call for death animation and stoping movement

Rigidbody and Collision Check

Rigidbody physics would play a crucial role in features of my application regarding collision, player movement and enemy movement/attacking. This RigidBody would seamlessly enhance player experience, through the prevention of players falling through maps or incorrectly colliding with objects.

Fine-tuning Unity Components for Optimal Performance

Unity includes a pre-determined set of Rigidbody and Collider components that only require parameters being fine-tuned, this powerful toolkit provides developers with opportunity to create immersive experiences. This toolkit would allow for iterative prototyping and testing.

Specific checks for the RigidBody physics are implemented throughout the title. This step aims to prevent scenarios where the character might 'fall through' the floor, requiring users to restart levels due to such issues. Identifying and rectifying these issues contributes to a smoother and more enjoyable gaming experience.

Synchronising Physics between Photon Clients

Once local Rigidbody and Collider physics have been meticulously tested and implemented correctly, seamless synchronisation of these features will need to be implemented across Photon clients in a multiplayer environment. This process would check the consistency of physics between connected clients and ensure there is no issues.

Whilst testing, Collider issues were explored, when the players were too close to each other or the wall, clipping was taking place, reducing the immersion provided by the game. For this to be fixed, an alternate collider was required, swapping from a box collider to a polygon collider more suited towards the shape of the player. Therefore, fostering an engaging gameplay environment, providing all participants with the relevant immersion.

ChatGPT API Check:

Unit Testing would be a requirement here to evaluate individual dialogue snippets and verify grammatical correctness whilst also adhering to the characters specific qualities embedded in the provided prompt. This would therefore allow for fine-tuning of prompts to find the best that most suits the character. In addition to this integration testing would be a requirement, this would be used to examine the integration of ChatGPT-3.5 Turbo ensuring that it is compatible with the application, whilst fine-tuning the script to minimise latency and ensure the dialogue is synced between players.

Synchronisation would be another feature that would be handled on Photon’s side via the PhotonView component. The dialogue will need to be added to the players PhotonView to ensure that both players can see the hint/dialogue rather than just the one, ensuring both players gain the same advantage or immersion. Furthermore, extensive playtesting would involve various testers that would simulate player interactions to refine and fine-tune prompt parameters enhancing the overall player experience. Through the continuous monitoring of interactions with the API, this would provide me with useful feedback that would address inconsistencies found within dialogue generation.

Chapter 5 Results and Discussion

Results are an important aspect of game development; they would be the determining factor deciding whether a feature within the game is ready for deployment and would consist of various prototyping and testing. Attaining results for applications require numerous amounts of unit testing and prototype releases, which would then provide feedback to be worked off until optimum results are demonstrated, and feedback returned becomes positive.

Within the development of BitJump, results played a crucial role in addition of features as well as shaping the future of the games development. Through the extensive testing mentioned throughout Chapter 3, it is evident on the features that underwent ever-changing development overtime based on user feedback. Feedback was gained from a variety of external testers unit testing specific parts of my application and providing valuable feedback, therefore initiating iterative improvements.

These results would serve as a foundation of readiness for the deployment of new features, ensuring that they meet a certain quality level before being integrated into the games main build or committed to version control. As BitJump continues to progress, the pursuit of gaining positive feedback on each of the features implemented is instrumental, further prototyping and testing endeavours are required to ensure the game reaches its full potential. Results are crucial elements in dictating whether a game is ready for release, whether it will captivate the audience and provide an enjoyable, immersive experience.

Overall, the results gained from my testing portrait that many of my planned features were implemented to an optimum level, allowing for users to join and leave lobbies at their own will, communicating with new players or simply utilising the private lobby feature with their friends. In addition to this, the continuous testing for new movement physics, level quality, communication and other features adding to the immersion of the application, ensured that the game is finished to a good level, with room for further updates to be implemented in the future.

Photon Client to Server Connection

Consistant prototyping was required to ensure the correct and seamless integration of the Photon framework. The initial results were getting clients to connect to the network and was tested using a range of debug methods portraying when a function was carried out correctly or when it had failed. For this to be setup, gamesettings had to be implemented, which would be used to facilitate connection between clients using the ConnectUsingSettingsMethod().

The successful establishment of client-server connections were crucial in beginning of development of my application. This would not onlt lay a foundation for future developments but also served as a testament to the robustness of my system architecture.

Connection between clients

Client connection was dependant on the AppID, region and version of the game to be the same. As a prerequisite for utilising Photon the AppID needed to be registered on their website for it to be instantiated correctly. This registration process would enable for seamless integration using the Photon network ensuring that clients experience smooth communication and are synced correctly.

The achievement of a stable connection between clients would mark as a milestone within my game development process, this demonstrated to simplicity of implementing multiplayer within applications and underscored the projects commitment to meeting the specific requirements of delivering a reliable multiplayer experience.

Communication and Synchronisation between clients

ChatGPT-3.5 Turbo for NPC dialoguew

Overall, after various integration and unit testing, the use of ChatGPT-3.5 Turbo for NPC dialogue within my application would represent further complexity within immersion and player engagement. By harnessing this power of advanced NLP techniques, games can feel more as if they are 'breaking the fourth barrier', enriching the overall gaming experience and blurring the line between reality and virtual worlds.
Once the dialogue had been synchronized between the connected clients, they would both have the ability to work off of the fine-tuned clues provided, in order to complete the various sections of my application in a co-operative manner.

Issues in development

Network Issues during development I faced multiple issues during the development of my project, the main one being the idea that using photon 1 for network implementation confronted me with multiple issues. The inability to use the PUN framework caused a range of different problems, such as not being able to use certain functions that they provide making it much more complicated to implement features such as lobby selection or player names. In order to find a solution for this, I decided to rework my implementation of Photon by using their 2nd framework, this was a lengthy yet rewarding process as the user is now presented by a much more welcoming interface when first loading into the title, as well as having the ability to leave and join lobbies when games have completed or whilst it is running. Photon 2 is a better, more improved version of the previously released photon 1, providing better Realtime integration, further integrated functionality and allows for…

Discussion and Analysis

After completion of this project, I have found that Photon is an effective framework for the implementation of networking into a Unity environment. The extensive documentation provided by Photon, alongside the vast number of videos and web pages analysing its integration and configuration allowed for seamless development. BitJump provided users with an array of menus and UI elements enabling the tailor of the application to suit their requirements. Through the successful development of private, public and matchmaking lobbies, it gave users options to either play alone, meet new users or play with their friends. In addition to this, the implemented settings menu provided users with the ability to either use different resolutions, volume settings or colourblind filters, a sense of diversity was also witnessed, expanding my target audience.

Technical Proficiency (Integration of Photon Network)

Within development the technical proficiency demonstrated was through the integration of the Photon Networking framework into the BitJump application. I utilised a varied number of different methods that is provided by PUN. Firstly the utilisation of Photons features enabled the creation of private, public and matchmade lobbies therefore offering a versatile range of gameplay options. Alongside this, the synchronisation between clients as well as chat features offered a more interactive experience for all players, seamless interaction is a crucial aspect of multiplayer games and their completion.

Versatile Lobby Creation

One of the primary strengths of utilising the Photon Network would be its versatile lobby creation capabilities. Utilising Photons API means that BitJump gains the capabilities to provide private, public and matchmade lobbies with a variable number of players, thus allowing for a diverse number of gameplay options. This flexibility is seen as a bonus as it allows clients to fine tune their multiplayer experience, whether it be online with random clients via matchmaking or public lobbies or privately with friends.

Synchronisation and Real-time communication

Furthermore within multiplayer applications, effective synchronisation between clients is paramount. Without the guarantee of a solid, uninterrupted connection players will lose out on the immersive and cohesive experience this genre of games are meant to provide. Leveraging a UDP approach, Photon would

ensure low-latency communication between connected clients therefore minimising delays in synchronisation facilitating smooth gameplay interactions. This real-time communication infrastructure that Photon provides would enable accurate message exchange, animation synchronisation and coordinated movement between players which are all crucial features for the enjoyability of my application.

Error Handling

Moreover, the error handling as well as the incorporation of previously mentioned ‘quality of life’ features underscore the enhancement of user satisfaction through prevention of the game crashing/experiencing data loss. Throughout development, I encountered various issues, for example when the user did not enter a name into the server creation field, however this was quickly resolved through the utilisation of further UI elements, and responsive debug logs, the game minimises disruption, therefore maximising user enjoyment.

Customisable and Accessibility Options

Furthermore, the options provided within BitJump allows for a totally customisable and accessible experience. These options would include variable resolution, volume settings, graphical settings and colourblind filters, thus reflecting and overall commitment to diverse user requirements. This commitment not only improves user experience as a whole, but also broadens the games appeal to a larger audience.

State-of-the-art features and innovation

Within BitJump, some emerging features were employed, this was utilised as a unique selling point for my application, as well as experimentation for future analysis. This would enable BitJump to be distinguished from other titles.
One notable example is the idea of my application being in communication with the Chat GPT 3.5 API, this was utilised for my NPC dialogue, thus meaning that each user will receive a different unique experience different to any other playthrough. Furhtemrore the utilisation of cutting-edge technology like this, underscores BitJump’s commitment to pushing the boundaries of gaming innovation. These features would not only embrace the games appeal but also be a contributing factor towards the games success.

Summary of development

In summary, the technical proficiency demonstrated throughout the development of BitJump, particularly within the seamless integration of Photons networking capabilities, ensuring that players are consistently in perfect synchronisation being key features of multiplayer applications. Through effective use of Photons multiple methods, via server versatility, user communication/synchronisation, room joining or overall connectivity, BitJump provides users with an interactive experience characterised by user-friendliness and inclusivity.

Chapter 7 Conclusions and Future Work

Conclusions

In conclusion throughout development I gained insights into the development process of a multiplayer application utilising Networking frameworks. The insights gained from the research behind Chapter 2's literature review alongside the pre-implementation planning of my application, allowed me to envision the most optimal game development tools and the crucial stages of networking integration. After investigation into multiplayer features, frameworks, game engines and development as a whole, I found that Unity and Photon went hand in hand for creating indie-like applications which was my aim with this project. After, once development began, this allowed me to gain practical experience of integrating a Network framework into a Unity environment alongside its configuration for best results.

BitJump portrays effective utilisation of the Photon framework, integrated effectively through the following of extensive documentation, YouTube tutorials as referenced below, and online forums allowing for the implementation of a variety of robust network capabilities and leverage its functionalities to enhance BitJumps multiplayer experience. Throughout the development process, BitJump served as an experimental playground, allowing for the exploration and implementation of various techniques/technologies in the realm of multiplayer game development.

Overall, the question regarding choice of networking framework and the choice of game engine for implementation has been successfully answered through research and practical means. The development of BitJump portrayed successful integration of the Photon network which was decided to be the most optimal networking framework due to its ability to provide a robust networking service for a low cost and extensive documentation making integration simple.

Future Work

Within my future work and development further features can be implemented into my application. Features such as a saving option, voice chat, further levels, character customisation and variation of enemies. Each of these unique features will be integral for the continued success of the application and will be implemented within the maintenance cycle through updates.

Save Option –

This was an essential feature as players looking to speed run or complete the full game would need to have the progress and times of each of the levels they have completed saved. Saving was enabled via the pause menu feature and would allow for both clients current progress to be saved, I have implemented it in this way as it would mean that clients can continue to progress either alone or together, without either client losing any data. Further research is required in order to implement and optimise this feature, one potential way of integrating this feature would be the utilisation of Remote Procedure Calls and the AllBuffered parameter to save carious games features to the cache, therefore ensuring smooth and efficient saving processes.

Voice Chat –

Although text chat was implemented within the game during the development stage, voice chat would add another level of communication for players and it achievable using Photon. ‘Photon voice 2 is an SDK that makes it easy to add high quality low latency voice chat to a Unity application’ https://doc.photonengine.com/voice/current/getting-started/voice-intro, therefore this can be further researched and implemented within further updates.

Character Customisation -

Further level implementation -

s

Chapter 8 Reflection

References

YouTube Tutorials Used in development:

Using ChatGPT within unity for NPC -https://www.youtube.com/watch?v=lYckk570Tqw [7]

Syncing levels - https://subscription.packtpub.com/book/game-development/9781849692328/2/ch02lvl1sec32/syncing-a-level-between-players#:~:text=You%20simply%20set%20PhotonNetwork.,between%20players%20in%20a%20room.

Enemy attack range - https://www.youtube.com/watch?v=d002CljR-KU&ab_channel=Pandemonium

AALTONEN, P. 2022. Available: https://www.theseus.fi/bitstream/handle/10024/755310/Aaltonen_Pasi.pdf?sequence=2&isAllowed=y [Accessed].
ANTTONEN, M. 2019. *ONLINE MULTIPLAYER ON MOBILE GAME* [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/221353/Anttonen_Mika.pdf?sequence=2&isAllowed=y [Accessed 20/02 2024].
BINDUMON, K. 2023. *Ethical Considerations in Game Development* [Online]. Available: https://medium.com/@karthikbindumon/ethical-considerations-in-game-design-dfb216f21249 [Accessed 14/02 2024].
BROOKS, N. 2021. *Why Platformers Will Never Go Out Of Style* [Online]. Available: https://www.cbr.com/platformers-important-gaming/ [Accessed].
DIVING_SQUID 2021. HOW TO MAKE AN ONLINE MULTIPLAYER GAME - UNITY EASY TUTORIAL.
EXIT-GAMES. 2024. Available: https://doc-api.photonengine.com/en/fusion/current/class_fusion_1_1_network_rigidbody2_d.html#details [Accessed 09/04 2024].
GAMES, F. G. 2020. Photon Networking 2 - Installing And Setting Up.
GITLAB. Available: [https://about.gitlab.com/topics/version-control/] [Accessed 03/03 2024].
GRIND, M. A. 2023. What is a good ping and FPS for online gaming? *What is a good ping and FPS for online gaming?* Quora.
HASBRO.INC. 2017. *BeyBlade Burst* [Online]. Available: https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://apps.apple.com/us/app/beyblade-burst-app/id1143164520&ved=2ahUKEwjkob6T_KGFAxVKUkEAHc_wDbYQFnoECB4QAQ&usg=AOvVaw0S5KcOsRAwcsZXiAP-KybT [Accessed].
IKEDA, A. 2023. *UNRAVELING THE MYSTERIES OF PROCEDURAL GENERATION IN GAMING* [Online]. Available: https://tokengamer.io/unraveling-the-mysteries-of-procedural-generation-in-gaming/#:~:text=Advantages%20of%20Procedural%20Generation%20in%20Gaming&text=Reduced%20Development%20Resources%3A%20Instead%20of,creating%20a%20more%20immersive%20experience. [Accessed].
JEANFABRE 2015. *Photon rigibody2dview for jumping and running?* Playmaker Forum.
JOEBAIN 2014. what do PhotonView's Observe Options mean. Unity Forum.
JOKINIEMI, T. 2014. *Unity Networking - Developing a single player game into a multiplayer game* [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/78214/Jokiniemi_Topias.pdf?sequence=1 [Accessed].
KRASSEN CINDY, A. S. 01 Jan 2018. *Games of Social Control: A Sociological Study of Addiction to Massively Multiplayer Online Role-Playing Games* [Online]. Available: https://typeset.io/questions/what-percentage-of-gamers-prefer-multiplayer-games-2xcwwahjok# [Accessed 04/01/2024 2024].
LINKEDIN. 2024. *How do you make your game stand out in the market?* [Online]. Available: https://www.linkedin.com/advice/3/how-do-you-make-your-game-stand-out-market-skills-gaming-industry [Accessed 19/01 2024].
MARTINEZ, E. 2021. *Intro to Multiplayer in Unity with Photon* [Online]. Available: https://medium.com/codex/intro-to-multiplayer-in-unity-with-photon-eea70ca054fe [Accessed 9/12 2023].
MICROSOFT. 2009. *Networking Basics: Peer-to-peer vs. server-based networks* [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-essentials-sbs/cc527483(v=ws.10)?redirectedfrom=MSDN [Accessed 19/01 2024].
PANDEY, H. 2022. *Peer-to-peer vs client-server architecture for multiplayer games* [Online]. Available: https://blog.hathora.dev/peer-to-peer-vs-client-server-architecture/ [Accessed].

PHOTONENGINE. *Showcase of Photon Projects* [Online]. Available: https://www.photonengine.com/realtime/showcase [Accessed].
PHOTONENGINE. 2024a. *Photon Initial Setup* [Online]. Available: https://doc.photonengine.com/pun/current/getting-started/initial-setup [Accessed 22/11 2023].
PHOTONENGINE. 2024b. *Region Configuration* [Online]. Available: https://doc.photonengine.com/pun/current/connection-and-authentication/regions [Accessed 19/01 2024].
PHOTONENGINEFUSION. *Introduction to PhotonEngine* [Online]. Available: https://doc.photonengine.com/fusion/current/fusion-intro [Accessed].
PHOTONREALTIME. 2024. *Photon Pun* [Online]. Available: https://www.photonengine.com/pun [Accessed 07/04/2024 2024].
QUAN, D. 2021. Multiplayer game development with Unity and Photon PUN : a case study: Magic Maze.
SERVERS. 2024. *Differences between peer to peer and dedicated game server hosting* [Online]. Available: https://www.servers.com/news/blog/differences-between-peer-to-peer-and-dedicated-game-server-hosting [Accessed 01/04/2024 2024].
THENULLREFERENCE 2018. *multiplayer with PUN2 or Netcode ?* Unity Forums.
TRAINOR-FOGLEMAN, E. 2024. Available: https://www.evercast.us/blog/unity-vs-unreal-engine [Accessed 09/04 2024].
TYROLLER, J. 2024. *This Problem Changes Your Perspective On Game Dev* [Online]. Available: https://www.youtube.com/watch?v=o5K0uqhxgsE [Accessed].