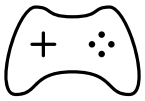
Abertay University



Using Orthogonal Unit Differentiation to Design Tools for Weapon Balancing in First-Person Shooter Games

Morgan Gareth Skillicorn

BA (Hons) Game Design and Production

School of Design and Informatics

Abertay University

May 2023

5874 words

Table of Contents

Table of Figures

Abstract

This project focuses on the design and development of tools within the Unity game engine to test the effectiveness of orthogonal unit differentiation (OUD) as an approach to weapon balancing in shooter games. The main objective was to explore various game balancing techniques and apply them specifically to weapon balancing. Through an extensive literature review, a solid theoretical foundation was established, revealing that fully automated balancing systems have had limited success. Consequently, a hybrid approach, combining manual and automatic tools, was deemed optimal for enhancing the efficiency of weapon balancing. Despite the project's successes, certain limitations were encountered, such as the absence of user testing. Nonetheless, the findings demonstrate that OUD is a viable method for achieving balanced gun mechanics. By working in parallel with manual tools, the integration of automatic tools proved to be an efficient and effective means of weapon balancing. This project contributes to the broader understanding of game balancing techniques and highlights the potential benefits of combining manual and automated approaches in achieving optimal game balance.

1. Introduction

Orthogonal unit differentiation (OUD) is a design principle that focuses on creating unique and distinguishable gameplay elements. The term *orthogonal unit differentiation* is used to describe the relationship between units in a video game, which are composed of actions, abilities, and characteristics. For instance, in the context of a game, a car unit is orthogonally distinct from a boat as the two cannot traverse the same areas and remain useful in the presence of the other. In contrast, a car and a motorcycle are not orthogonally different, given that they both travel on land and can perform similar functions. This principle has been utilized in the gaming industry for many years to compare game units, yet it is proposed to extend its application through research into the balancing of guns and weapons in first-person shooter games.

The task of weapon balancing has been difficult to automate, due to its reliance on human intuition as well as the influence of external factors. The dominant strategies of a game, also known as the *meta*, are frequently a subject of discussion as balance changes take effect and players gravitate toward the most effective tools available. If a player is presented with a choice of several weapons and consistently selects one over the others, it may be inferred that there is a strictly dominated strategy and orthogonal unit differentiation could be a solution.

Through the creation and development of a toolkit for the Unity editor, research into the application of tools for weapon balancing is planned. The role of orthogonal unit differentiation in informing game designers about potential improvements to their game's weapon balance will be investigated.

This project intends to employ a range of methods to abstract weapon data and present it to the user with varying degrees of complexity. The toolkit development within this research will contribute to an understanding of tool and system design processes, with the ultimate goal of creating commercially viable systems.

The primary objective of this project is to design and develop a weapon balancing system for a first-person shooter game that leverages orthogonal unit differentiation to balance gameplay systems. The research will delve into the current state of the industry regarding game balancing, with a particular emphasis on weapons and the creation of supportive tools. Development of a rudimentary first-person shooter will provide context for the research, facilitating the design and development of a commercially viable, packageable asset to be reviewed and reflected upon throughout and at the conclusion of the research project.

2. Related Work (Literature Review)

2.1 Weapon Balancing in First-Person Shooter Games

2.1.1 Importance of Weapon Balancing

Weapon balancing is a crucial aspect of first-person shooter (FPS) games to ensure fair competition and enjoyable gameplay. This section will review the current literature on the importance of weapon balancing, including how it impacts player satisfaction, retention, and the game's overall success. The importance of balancing will have a great effect on the importance of tools to balance games making this a useful area to review.

Weapon balancing is integral to the enjoyment of FPS games ensuring fair competition. Well-balanced weapons contribute to the overall appeal and fairness of FPS games, resulting in a more enjoyable experience for players. Weapon balancing greatly contributes to the level of enjoyment players have of FPS games and ensures fair competition. Numerous studies have been conducted to explore the impact of weapon balancing in FPS games, including those by (Zagal, 2013) and (Barr, 2008). (Zagal, 2013) mentions in their report that the “grind” mechanics employed in Call of Duty 4 can put off some players as the balance tips in the favour of players that are more highly invested. Sometimes this imbalance is a problem that must be resolved and at other times is a conscious design choice that must be carefully considered in the overall landscape of the game to ensure that the overall experience does not fall out of balance (Zagal, 2013).

Balanced weapons lead to increased player satisfaction as the player feels that skill is the determining factor in their success (Chen, 2017). Chen looks at this in some detail their study into balancing the matchmaking systems of competitive multiplayer online battle arena (MOBA) style games stating, “matching players with too wide skill gaps leads to completely dominant wins that bore the winners and discourage the losers”. Subsequently, a well-balanced game can lead to higher player retention rates, as players are more likely to continue playing a game they find fair and enjoyable (Adams, 2014). Although this report focuses on the development of tools for the balancing of FPS weapons, not matchmaking, this idea still applies and is important to consider how a tool could ensure gameplay is fair.

2.1.2 Approaches to Weapon Balancing

Various methods have been employed by game developers to achieve balanced weapon systems. This section will discuss traditional approaches, such as playtesting and manual adjusting, as well as more recent techniques, like data-driven and procedural generation methods.

The significance of traditional approaches such as playtesting and iterative design to balance weapons in FPS games is discussed in (Hullet, 2012). However, it is also well analysed in this report that games are becoming increasingly more complicated and that “traditional playtesting is no longer able to provide sufficient coverage of all possible gameplay states”. While still being a very important tool it is worth noting that rarely do any of these balancing techniques work independently. Playtesting is an essential tool for gathering user feedback to refine weapon balance, ensuring a fair and enjoyable gameplay experience. It is recommended that playtesting is involved in development from the very beginning however this may not always be practical when developing complex multiplayer systems (Fullerton, 2014).

In some cases, developers manually adjust weapon parameters to create balance within the game, as described by (Hunicke, 2005) these changes are often felt by players even when very minor. This is a very interesting report as it goes into adjusting the player experience during gameplay. It is mentioned that players feel cheated when gameplay elements are adjusted during gameplay however when

these changes are made without the player being aware they do not feel this way. This could differ with weapon systems that take place in an online environment but it is worth noting how dynamic difficulty changes could be applied to overused or underused weapons to make them more viable in an FPS setting.

By using modelling techniques developers can mimic “the metagame” of a given player base, generating synthetic data to forecast how players might play a game. This modelling can then be used to balance the game before more effectively (Kavanagh, 2021). Balancing is able to be executed more effectively and efficiently as designers are able to work with vast amounts of data based on a model that is accurate to real player data and make changes as appropriate. This then allows for balancing to be done with less negative impact on players and less manual testing involved.

Although this research focuses more on manual game balancing using advanced tools and data abstraction it is worth considering these alternative balancing methods and how they might work alongside the artifacts of this research. A hybrid model looks increasingly more feasible when reviewing the existing literature on the topic of gameplay balancing.

2.2 Orthogonal Unit Differentiation (OUD)

2.2.1 Concept and Benefits

This section will explore the concept of OUD and its benefits, particularly in the context of weapon design and balancing in FPS games.

In most explanations of OUD, a two-axis graph is used with the X and Y axis being orthogonally differentiated. It is important to consider that the axis of a graph is not a unit but functionality of a unit and that units can span the area on the graph. By adding more axis to this graph in the form of a radar graph it is possible to begin building a complex model of OUD as a design principle (Smith, 2003).

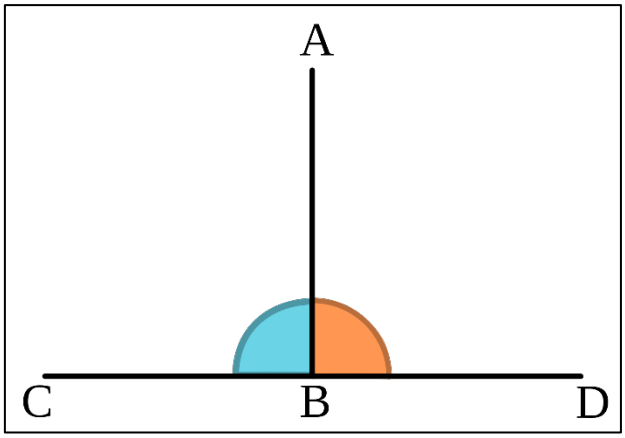


Figure 1 The line segments AB and CD are orthogonal to each other. No amount of X provides movement along Y axis.

In (Makin, 2017) OUD as a game design principal is formally tested using the example of StarCraft II. The article looks at whether the present units are orthogonally differentiated and deconstructs the vast parameters of each unit to find where the linear dependencies are. As a result of this it is possible to draw conclusions as to whether the units are theoretically balanced (Tekinbas, 2004).

It is intended that this research will explore the orthogonality of an array of parameters contained in a weapon object. The practicality of comparing more than two units’ parameters must be considered making a radar graph a suitable data visualisation tool. It is worth noting that radar graphs have been shown to be one of the least efficient graphs for users attaining data however they can visualise overall

trends in data very well. For this reason, they should not be used to deliver raw data but instead used to compare the data of multiple units and to test their orthogonality (Abeynayake, 2023).

OUD as a design concept benefits a game in many ways, firstly by adding strategic depth to player decision making. By designing units that are orthogonally different players are forced to weigh up the pros and cons of each unit and evaluate the effectiveness of it in a given situation. This is discussed thoroughly in Harvey Smith's 2003 GDC talk (Smith, 2003). This principle could be applied to weapon classes and force players into picking a high damage sniper for long range or a lower damage but faster firing submachine gun in a different situation. OUD encourages replayability by having strictly different units that excel in different areas. This is exampled very well by Call of Duty Modern Warfare 2 (Activision, 2019) where players level up different guns in the multiplayer to unlock various new attachments and skins. There are various different play styles to the game and often you can find players repeating the same game mode, working through different archetypes, and still succeeding in match despite using orthogonally differentiated weapon archetypes.

2.2.2 OUD in Game Design

OUD helps prevent strictly dominated strategies arising (Smith, 2003). In the popular game Rainbow Six: Siege, the use of light machine guns (LMGs) became the strictly dominated strategy as the damage dealt by LMGs paired with Siege's unique "single headshot kill" feature meant the operator Finka was picked in 75% of the ranked matches played at a high level. Ubisoft states that "there are a few components that we agree are tuned a little high, specifically LMGs and Finka" (Ubisoft, 2022). Considering the game features 67 playable operators, over 30 of which are attackers, using human intuition it is obvious that this is an issue. This is something that OUD could potentially display graphically before it ever became a problem.

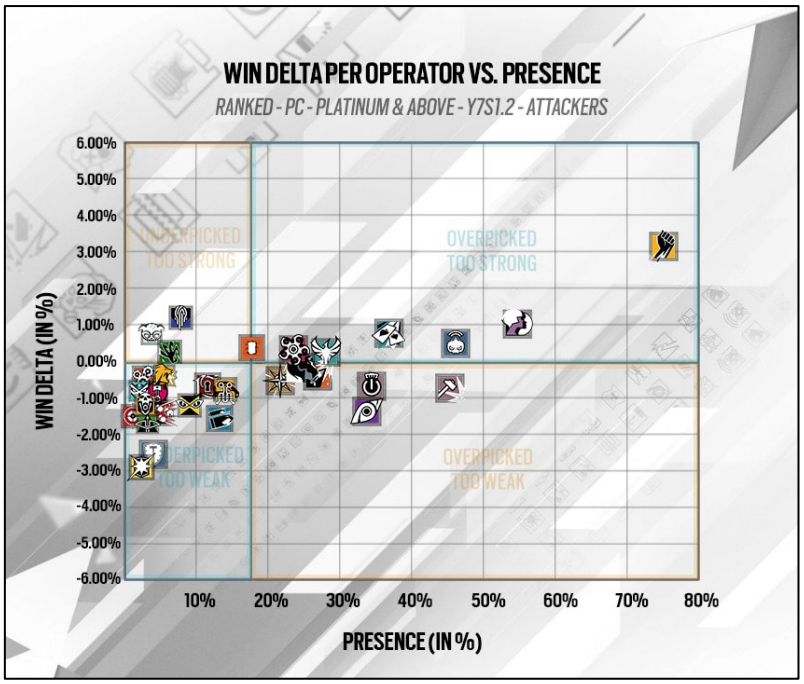


Figure 2 The win delta of operators in R6 Siege showing the operator "Finka" being played in 75% of ranked games, in a game with over 30 operators, above the platinum rank with a positive win delta.

Humans are able to balance systems using intuition and it is clear that the fundamental principles of balancing through OUD are clearly something that humans understand subconsciously. As (Smith, 2003) mentions perhaps one of the most basic implementations of OUD is in the game Rock Paper

Scissors. The game is nothing more than an implementation of OUD and each “weapon” is balanced perfectly to fit with every other weapon making no single weapon better than any other.

2.3 Tools for Weapon Balancing

2.3.1 Designing Tools

Prebuilt software for weapon balancing in FPS games is scarce and so most developers end up having to create their own solutions, wasting time that could be invested elsewhere. Developing editor tools is nothing new and can be a great way to improve efficiency in a game pipeline. “Design Games for Architecture” (Westre, 2013) is a book that teaches the fundamentals of creating tools in Unity. The research project will use the Unity engine making this reading a very useful resource. The specifics of designing tools are talked about as a game, “tools that a designer “plays” like video games”.

2.3.2 Machine Learning and Artificial Intelligence in Weapon Balancing

“Procedural weapon generation for unreal tournament III” (Gravina, 2015) is a study where weapons were balanced using a procedural model to fully automate the task of weapon balancing in an existing commercial shooter game. This study draws a lot of similarities to that of this research project however it differs in the methods it uses to define “balance”. Gravina considers a weapon balanced if the number of kills by a bot in a simulation closely meets the value of a complex formula. While this may seem balanced it doesn’t account for user agency and does not suggest the weapons are orthogonally differentiated.

The use of dynamic game balancing through automated systems and the impact it has on user satisfaction is analysed in (Andrade, 2021). This study shows that agents that implement a dynamic balance and perform closer to the level of a real user provide the most player satisfaction. It is clear that players want balance. AI systems are able to deliver balance more closely, although not in all areas, making a hybrid approach more desirable.

2.4 Related Fields and Their Contributions

2.4.1 Design and Art

A particularly poignant quote “The perception of balance is more powerful than balance itself” (Kaplan, 2017) discusses the importance of the player’s perception of balance. In a forum post, Kaplan informs players of Overwatch that balancing is an ever-evolving cycle to change the strictly dominated strategy. In 2019 People Make Games released a documentary highlighting the use of guns in Wolfenstein: Enemy Territory and it was apparent during beta testing on the game that there was an apparent imbalance in two guns, the MP40 and Thompson, used by the opposing teams despite them both sharing identical metrics (People Make Games, 2019). It became apparent that the players’ response to an audio file for one of the guns having a deeper tone led players to play more aggressively as they thought they had a better gun. In turn this led to skewed metrics and an issue with game balancing that was entirely driven by a placebo effect. This placebo effect exists and for this reason it is important not to analyse individual datapoints and metrics in a vacuum. Like all fair experiments, when balancing gameplay, testing individual variables and maintaining a constant will lead to better results.

3. Implementation

This was a technical project and the implementation explored a large area of technical design, systems design and tools design. This implementation section will break down each development stage and go into detail on the processes and design choices that were made.

3.1 FPS Controller and Weapon System

3.1.1 Developing a Movement Controller

Initially, a basic movement controller was developed, utilizing the WASD keys for character movement and a separate camera object that could be controlled with the mouse. It was crucial to create a simple controller to ensure the compatibility of the universal weapon system. The weapon controller was then attached to the camera, allowing it to rotate with the player's movements as seen in figure 3.

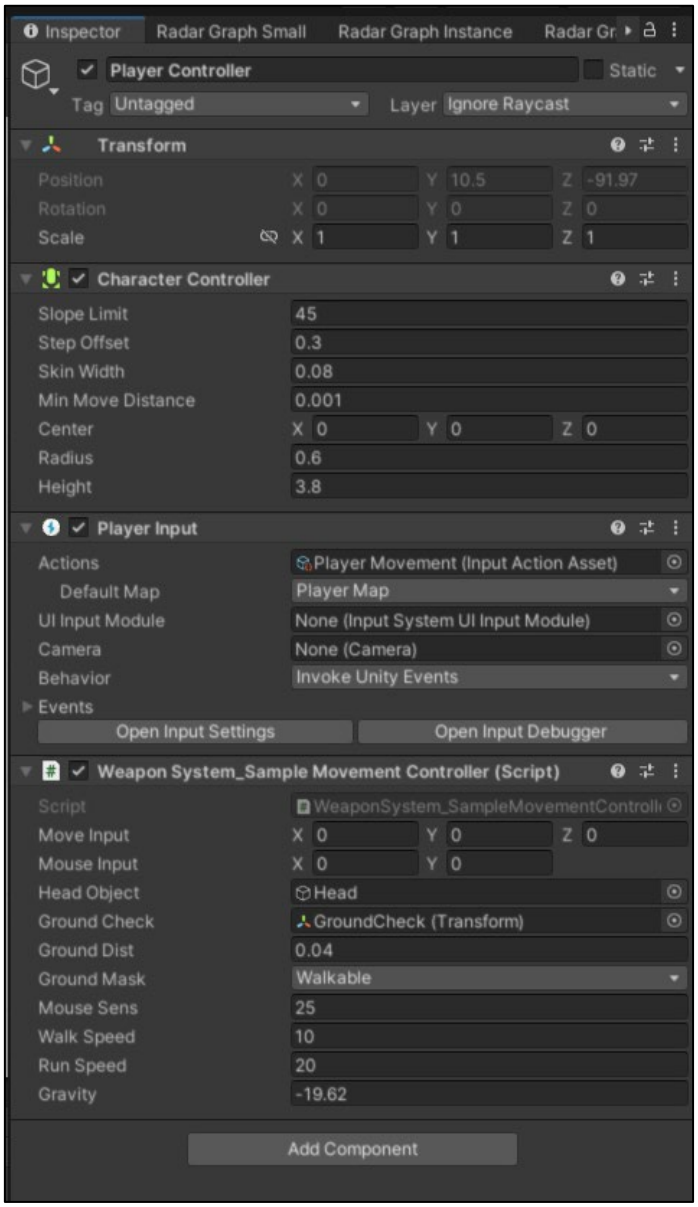


Figure 3 Player controller as seen in the inspector window. The controller sensitivity and gravity can be adjusted to be suitable for a variety of games.

3.1.2 Designing a Universal Weapon System

The weapon system had to be universal in order to appeal to a larger user base. It is intended that this system will be released as a Unity asset making it suitable for implementation in a number of potential projects. In order for the weapon system to fit a project universally it consists of two simple game objects that can be dropped onto any first-person controller in order to add weapon functionality. This simple implementation allows for a standardised way of managing weapon logic and intuitive management of weapon parameters. Figure 4 shows the carefully considered hierarchical structure of the player controller and weapon system.

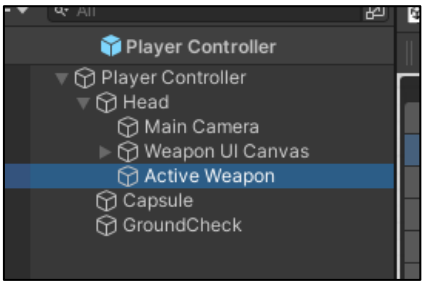


Figure 4 Hierarchical structure of the player controller prefab with the weapon system implemented.

3.1.3 Raycasting and Registering Gunshots

In order to register shots fired by the weapons raycasting was used. Built into the weapon controller is a system that fires a raycast based on the range parameters of the current weapon. The developer using the system has the option to apply the raycast origin to whatever object they feel most appropriate for the given system. For the purposes of this project and the demo implementation, the raycast will be fired from the camera as is common practice in many competitive FPS games. Performing raycasts also allows for physical collisions that can be used to call functions when bullets land on their targets to manipulate health. The raycast happens in the fixed update class for performance reasons and to ensure physics collisions happen reliably. An inverted layer mask is also used to ensure shots are only registered on objects with the relevant tag assigned to it.

```
Unity Message | 0 references
private void FixedUpdate()
{
    int layerMask = 1 << 2;
    layerMask = ~layerMask;

    RaycastHit hit;

    #region firing update
    if (firing)
    {
        if (Physics.Raycast(fpsCamera.transform.position,
                fpsCamera.transform.TransformDirection(Vector3.forward),
                out hit, Mathf.Infinity, layerMask))
        {
            shootTimer += Time.deltaTime;

            float fireRate = currentWeapon.parameters.fireRate / 60;
            fireRate = 1 / fireRate;

            if (shootTimer > fireRate)
            {
                shootTimer = shootTimer - fireRate;

                if (HasAmmo() && !stillReloading)
                {
                    if (currentWeapon.parameters.automatic)
                    {
                        Debug.DrawRay(fpsCamera.transform.position,
                            fpsCamera.transform.TransformDirection(Vector3.forward) * hit.distance,
                            Color.red);

                        RegisterShot();
```

Figure 5 Code snippet from the weapon system showing the raycast implementation. The snippet shows the timing system used to delay raycasts with a check for ammo. The ray is also drawn as a gizmo in the editor for debugging purposes.

3.1.4 Firing Systems and Reloading

The ammunition system is straightforward, however very important as it is a key area to consider during the balancing stage. When a shot is fired, the ammo count in the current magazine is decremented by one. Once the magazine count has reached zero the player must then reload. When the reload function is executed the capacity of the magazine is decremented from the ammo reserve until this also reaches zero. When designing the reload function it became apparent that the reload speed of a weapon is another important metric that can be balanced using the OUD system. When reloading an animation is played in order to immerse the player and this animation could be any length of time. A system was added that takes the desired reload time and then uses a formula to retarget the keyframes in the animation clip to the specified time frame, seen in figure 6.

```
reloadTimer += Time.deltaTime;
GetComponentInChildren<Animator>().speed = currentWeapon.parameters.reloadSpeed / reloadTimer;
```

Figure 6 Recalculating the speed of the animation clip based on desired reload time.

3.1.5 Recoil

Recoil control is one of the primary distinguishable traits between the skill levels of players in an FPS game. For that reason, the balancing of recoil relies on a robust and customisable recoil system. The recoil system designed for this project features the following parameters for each weapon's hip fire and aimed states:

- recoil
    - Base recoil X
    - Base recoil Y
    - Recoil random X
    - Recoil random Y
    - Recoil snap
    - Recoil return

The system works by taking the base value of X and Y, multiplying them by the corresponding randomness value and then generating a new value based on that using a random range. This predictable randomness allows for customisable recoil patterns that can be controlled but still perform in a similar way to an actual gun. The recoil return variable returns the gun to the original position at a constant rate after firing which can be disabled depending on the implementation. The snap is used when adding recoil after a shot to give the illusion of kick on a weapon in a more realistic manner. As an example, the formula used to apply horizontal recoil and update the gun and camera position is as follows:

$$H = RAND((B * -R),(B * R))$$

H = horizontal recoil B = base recoil R = recoil randomness

3.2 Storing Weapon Parameters with Scriptable Objects

3.2.1 The Benefits of a Universal Data Structure

The weapon data in this project is stored as a Unity scriptable object consisting of a custom data structure with every changeable parameter in it. This solution allows data to be stored and accessed in a uniform, predictable manner. It also means that when adding a new weapon, the developer can simply fill out the variables for the object and add the desired art assets to it without having to

manually create several prefab game objects. As a tools design task this is also an effective way to create a tool that looks polished and can be manipulated easily by someone who does not fully understand the specific tool but does have knowledge of the Unity editor. The tool for creating weapon objects can be seen in figure 8.

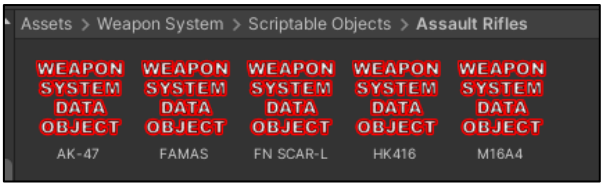


Figure 7 Example of weapon scriptable objects containing assault rifle weapon data.

3.2.2 Creating and Editing Scriptable Objects

Scriptable objects ability to be sorted in a Unity project makes them very useful for storing weapon data. The weapon controller has a variable to store the current primary and secondary weapons and by simply changing this the player swaps weapons. Subsequent tools developed to balance weapons can capitalise on this easy-to-modify setup and edit weapons as appropriate.

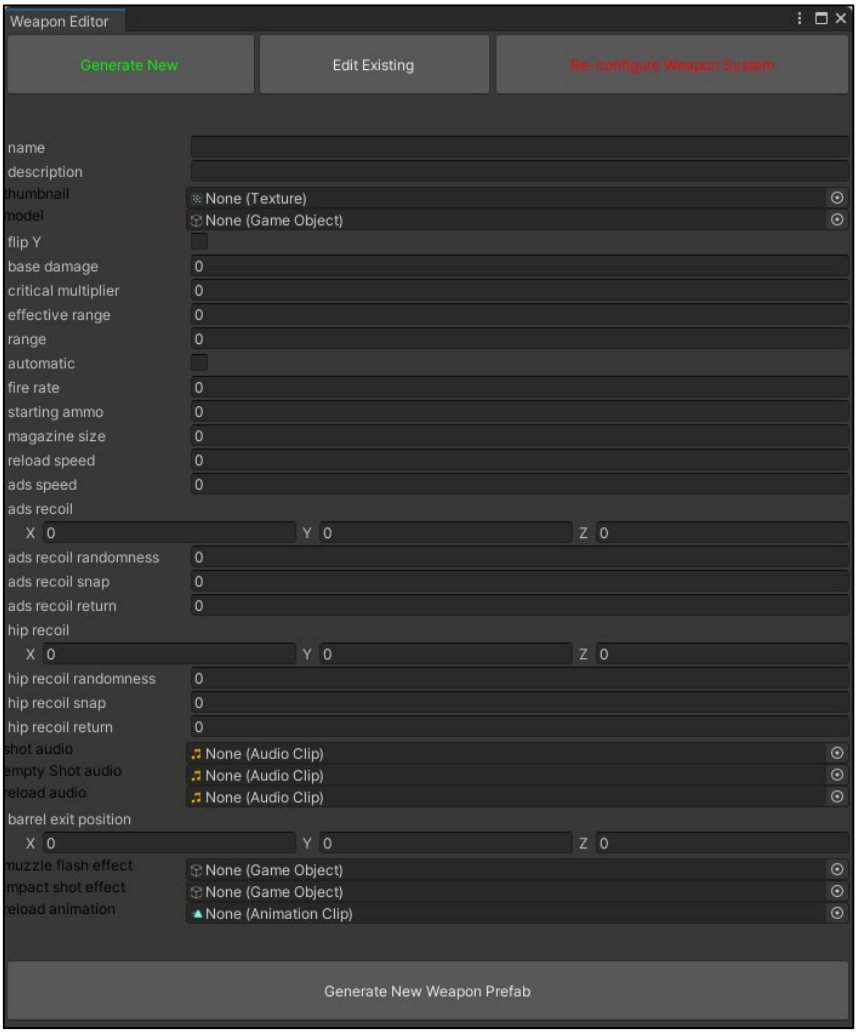


Figure 8 Custom tool designed to generate new weapon data objects as well as manually edit existing weapons. Here the different paramaters for a weapon can be edited for balancing purposes.

3.3 Designing Tools in the Unity Editor

3.3.1 Graph Renderer

During the initial design stages of this project, it became evident that the need to display and visualise abstract data in the Unity editor would be important. After considerable research it became clear that there was no good existing solution to render complex graphs in this way. The answer to this problem would inevitably be to create a custom package using a low-level graphics library that became known as Graphs++. This package features custom graph tools that can be directly integrated into a Unity UI class to display radar graphs currently although the intention would be to expand this tool to cater to other types of graphs in the future. A single graph consists of a custom data structure that contains multiple datasets, consisting of multiple data points. This structure allows for multiple datasets to be displayed and compared at once in a uniform way. The manner in which these graphs are rendered allows them to be responsive to the window size and expandable to any number of datapoints.

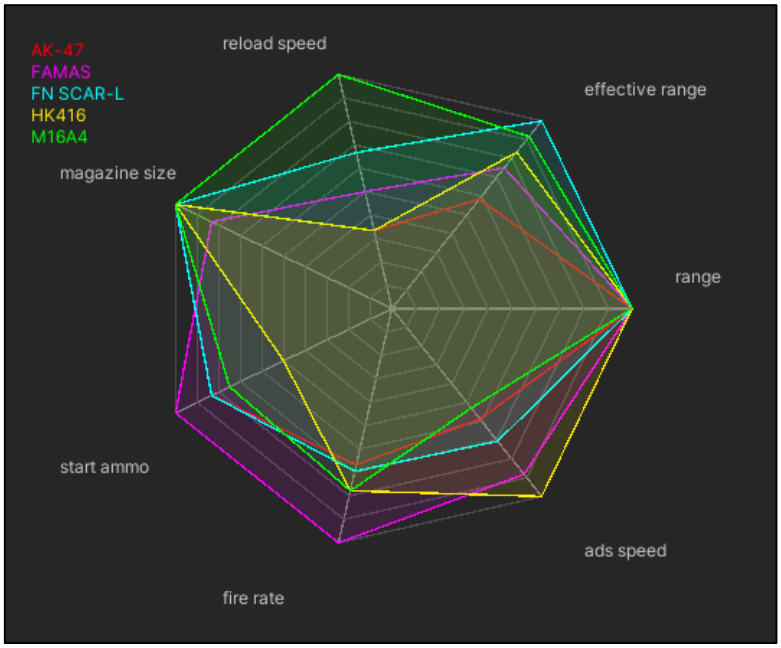


Figure 9 Graphs++ radar graph implemented in a UI with multiple datasets. The data being displayed shows five assault rifles being compared by seven attributes. No weapon in this comparison is statistically dominant as none of them have overlapping convex hulls.

3.3.2 Weapon Balancing Dashboard

Perhaps the most important tool developed as part of this project is the balancing dashboard. This dashboard acts as a user interface for the designer/developer that uses this system in their game. The dashboard features a basic view and an advanced view. Both views feature a large radar graph in the centre and a list of all the weapons found in the project directory sorted by archetype. The weapon archetype must exist in the weapon's description. Users can then add and remove weapons from the comparison to balance them against each other. The graph axis scale based on the data being visualised. At the bottom of the screen is a list of suggestions that the system recommends based on a set of rules. The primary rule being tested is that no weapon can be strictly dominant meaning if the convex hull of one data set encompasses another a warning will be flagged. Convex hull refers to the line created by each dataset on a radar graph. Simply, if every parameter of a dataset is lower than that of another dataset the convex hull is fully encompassed.

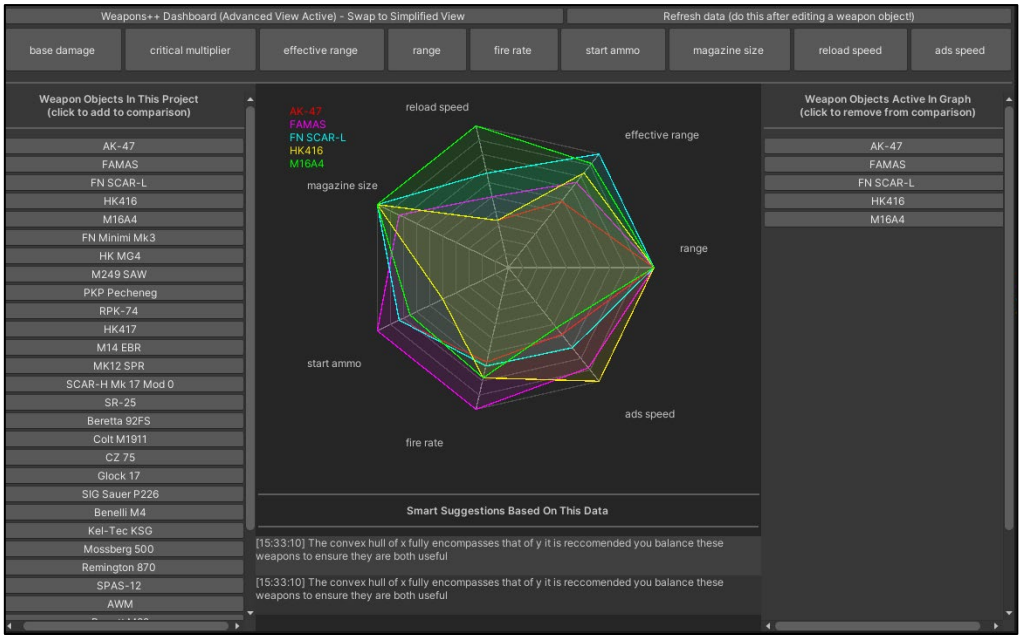


Figure 10 The weapon balancing dashboard set to advanced displaying a custom set of parameters from the assault rifle archetype.

3.4 Data Analysis Tools

3.4.1 Simplified Dashboard View

The simplified dashboard view features a selection of pre-set graphs for less adept users to easily compare the various key characteristics of a weapon. These are things such as recoil, damage, and handling analysis each featuring various relevant datapoints to balance. The user can then add weapons from the complete list on the left to a comparison on the right and visualise the data on the radar graph.

3.4.2 Advanced Dashboard View

The advanced view is the same as the simplified view for the most part however instead of having pre-set graphs the user is able to toggle on and off individual datapoints, allowing them to build a custom graph that caters to the balancing objectives the have for their specific project. Again they are able to add and remove weapons from a list and visualise the data on a radar graph.

3.4.3 Automatic Balancing Suggestions

The automatic balancing section, stylised as *smart suggestions*, is designed to inform the user of potential issues in their current weapon balancing setup. Predefined checks are built into the dashboard that calls functions in the Graphs++ class and tests the data against a set of rules. The primary rule used to check the current balance is whether or not a weapon strictly dominates any other weapon. If a weapon's convex hull fully encompasses the convex hull of another weapon, an alert is flagged for the user. The idea is they will be made aware that a weapon is either overpowered or underpowered and be able to adjust accordingly. This set of rules could be expanded in a number of ways to automate balancing. Automation must be carefully considered as the tool is designed to be implemented in a variety of projects and a user may intentionally have created a weapon that appears to be imbalanced.

3.5 Balancing Weapons

3.5.1 Creating Weapons Based on Real Weapons

In order to test the solution a simple firing range was developed with a multitude of guns added. These guns are based on real guns with all the initial data values being set true to life. As this is not an art project, sample assets were used that do not reflect the look of the actual weapon. By inputting this real data into the demo, it is possible to build a rough guide as to where the various weapons should sit in the balance of the demo and then adjust them to enhance gameplay afterward.

3.5.2 Using OUD Tools to Analyse and Iterate

Once a rough balance had been reached inputting predefined weapons more detailed balancing could take place in order to test the system. Sample data was used that was known to be imbalanced so the system could identify this and recommend changes as a test. The system was then used and tested in an iterative manner to carefully implement balancing across a variety of weapons and weapon archetypes.

4. Evaluation

4.1 Achievement of Objectives

The literature review and research provided a comprehensive understanding of weapon balancing techniques and gameplay balancing as a whole. The project explored various existing titles and research in the field allowing for a strong theoretical understanding prior to the design and development of any research artefacts.

The tool created as part of this research allows game designers to rapidly iterate balancing tasks for weapons in a first-person shooter. Data can be visualised and manipulated in real time making balancing more efficient and reliable. The project demonstrates the creation of a tool in Unity for the practical application of the theory outlined throughout this report and puts the knowledge gained into practice. Overall this was a successful way of practically researching the topic.

4.2 Research Evaluation

4.2.1 Research Strengths

The literature review provided valuable insights into the existing techniques used in industry and methods that have been proven ineffective previously. The research demonstrated a deep understanding of the subject matter and helped to inform the development of the tool. The implemented tool's functional capabilities enhance the game design processes and help to achieve a well-balanced weapon system more efficiently.

The addition of visual tools helped facilitate efficiency, providing the user with feedback and aiding them in game balancing. The visual representation of data enhanced the tools usability and effectiveness allowing the usefulness of OUD to be tested more rigorously.

The project artifacts have commercial viability and could be developed further, ultimately being released as a full tool for Unity. This commercial viability showcases the findings of the project and the success in researching the initial questions asked.

4.2.2 Research Limitations

The research project faced some challenges. One of which being the ability to test the usability of the tool in existing games. This was due to the data and tools used by commercial developers being proprietary. This limited the research to open-source solutions and ultimately the development of a basic first-person shooter to serve as a test project.

Due to the nature of the project, there were issues regarding the projects potential scope. As the project was completed by a single person it was not practical to develop a comprehensive first-person shooter with polished gun models and various systems to provide context for balancing. It was difficult to test the solution in an environment where the targets are unable to react and ideally the research would continue further and be implemented in a live service game to test it more fully. The exclusion of multiplayer elements may have resulted in a narrow perspective and hidden possible issues with the balancing of weapons. Possible improvements could include feedback providing the user with damage per second calculations and recoil pattern charts.

4.3 Findings

4.3.1 The Usefulness of OUD

It is clear that OUD is a valuable approach to balancing game units, specifically weapons. However, it must be said that this assumes the game units are separated only by quantitative data as qualitative data cannot be compared visually on a radar graph. The tool was able to aid in the balancing of an

array of weapons making them all orthogonally different and ensuring that there was no dominant strategy. The visual tools developed for the project played a crucial role in leveraging the usefulness of OUD. By using graphical representations of data, designers are able to visualise the parameters of weapons and compare them with the parameters of other weapons easily. This visualisation enhances the balancing workflow and facilitates the game design process.

4.3.2 Tool Usability

Due to the project scope, the weapon system is rudimentary. Because of this the implementation of new weapons is complex. When giving feedback on the tool one person mentioned that it is complex to set up and because of this people could be put off using it. Once the tool is set up it is intuitive to use so more thought should be put into rectifying this. One possible solution could be a setup wizard to automate the creation of weapon prefabs, in the first instance before any balancing takes place, to make repetitive tasks such as importing models more streamlined.

4.3.3 Using Graphical User Interfaces (GUIs)

This project uses various visual tools extensively, one of which being the Unity editor GUI. The project makes good use of the tools available in Unity and expands on them where appropriate. The development of a graph extension is a clear example of using tools in conjunction with OUD to balance gameplay units. With more careful planning the Unity editor could have been used more efficiently with cleaner code and more in-depth tools such as the wizard mentioned previously.

4.4 Methods

The project primarily made use of the Unity engine for the development and implementation of the tool. There were some challenges due to the lack of any formal design stage such as the organisation of code and source control solutions. A large portion of the code base has been refactored although more work is needed to develop a tool that is commercially ready.

A notable method employed was the development of a graph rendering tool that has provided incredibly useful for manipulating data and analysing it in real time. The inclusion of this renderer enhanced the usability of the tool and helped to form an understanding of OUD for weapon balancing.

Overall, the methods used in the project, particularly the integration of the tool within Unity and the development of visual aids, contributed to the successful achievement of the project's objectives. The project intended to employ a range of methods to abstract weapon data and provide it to the user with varying degrees of complexity. The tools developed do this and provide the user with the ability to implement the theoretics of OUD into their project with relative ease.

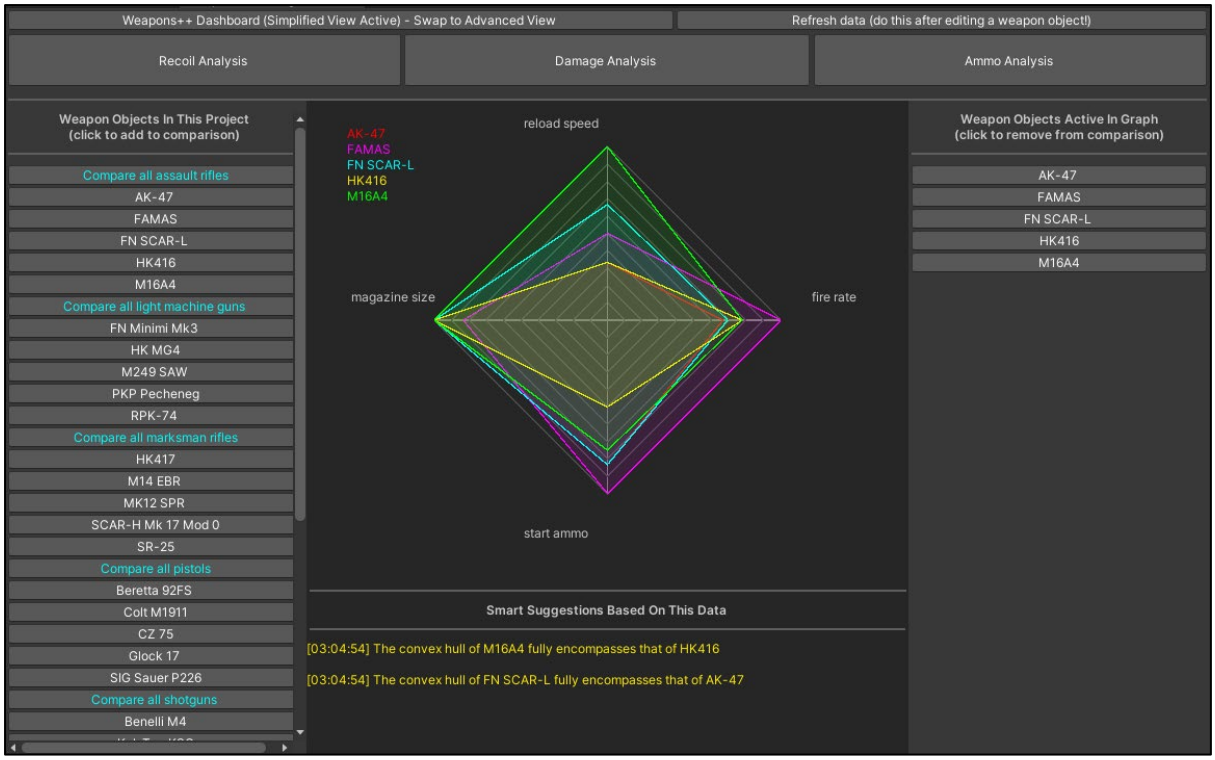


Figure 11 Smart suggestions tells the designer in this view that two of the weapons are unbalanced and there are strictly dominating strategies.

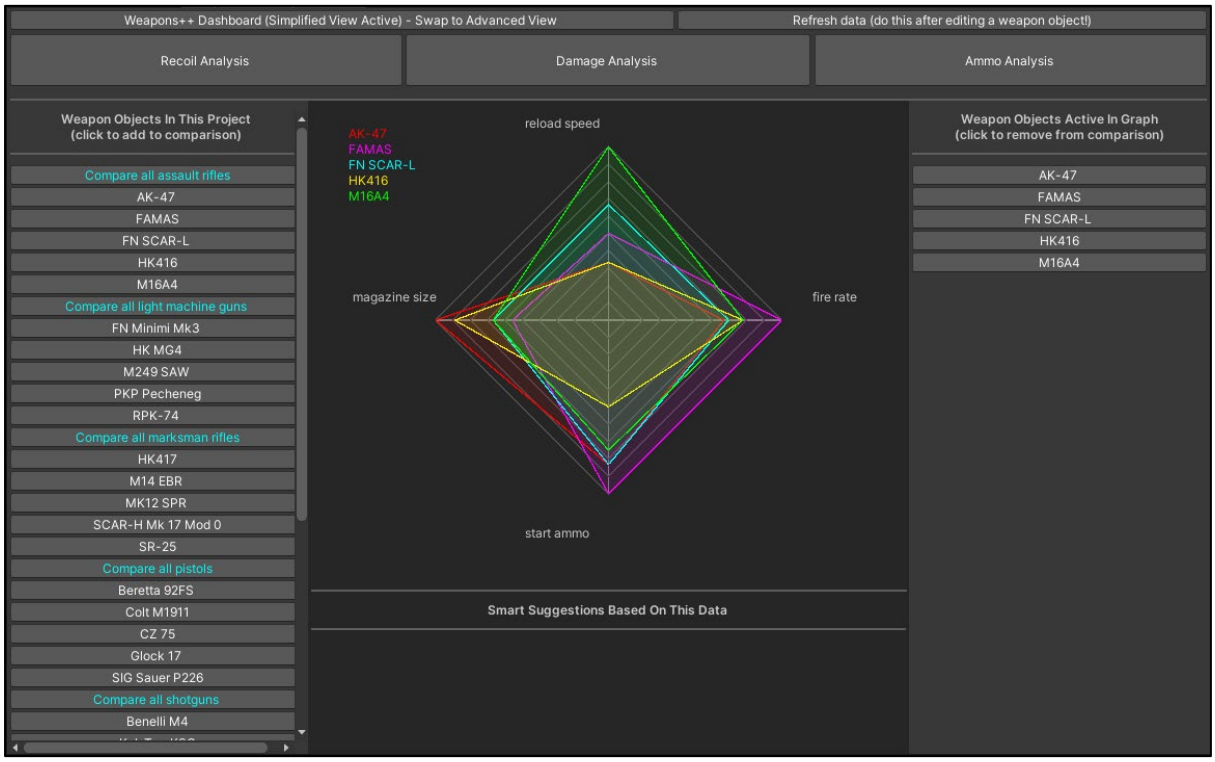


Figure 12 In this view the magazine size of the AK-47 and the HK416 have been increased to improve the balance of the game making each weapon orthogonally different.

5. Conclusion

In conclusion this project aimed to explore the use of orthogonal unit differentiation as a method for balancing weapons in first-person shooter games. It investigated the various balancing techniques and a tool was developed for the real-time manipulation of data to achieve suitable game balance. Through a comprehensive literature review, the project gained insights into existing industry practice and a theoretical foundation for the design and development of a weapon balancing tool.

The developed tool demonstrated success in achieving game balance through a hybrid approach. It enabled designers to manipulate weapon parameters and visualise their relationship with other weapons in real time. Although the tool is user friendly to an extent more work needs to be done to automate repetitive tasks before it could be released commercially. The hybrid approach that was reached automates the abstraction of data for a human to analyse and balance accordingly.

The research evaluation identifies several strengths and weaknesses with the project, highlighting both its success and opportunities for improvement. If the project was to be repeated it is likely that the same results would be reached however the artifacts developed would be of a higher standard and explore the use of OUD more deeply.

The findings indicate that OUD is a useful framework for weapon balancing when used in a hybrid model for games that rely on quantitative data. As some qualitative traits of a weapons, such as the animations and audio assets could influence a player's perception of the weapon, OUD cannot be used with complete confidence for the analysis of qualitative data.

By comparing figures 11 and 12 the usefulness of OUD and data visualisation can be seen. The pre-programed rules have detected a strictly dominating strategy and informed the designer of this. The designer is then able to re-balance the weapons using the radar graph to inform their decisions.

In summary, this project successfully addressed its objectives by researching weapon balancing techniques, developing a tool for real-time data manipulation, and evaluating its effectiveness. The findings contribute to the understanding of game balancing and provide insights into the application of the OUD approach. The project's achievements and identified strengths lay the foundation for further research and the potential commercialisation of the developed tool. With continued refinement and expansion, this project has the potential to make significant contributions to the field of game design and balancing techniques, as well as providing a structure for future developers of similar tools.

References

Abeynayake, H. G. (2023). Efficacy of information extraction from bar, line, circular, bubble and radar graphs. *Applied Ergonomics, 109*. doi:10.1016/j.apergo.2023.103996

Activision. (2019). *Call of Duty MW2.*

Adams, E. (2014). *Fundamentals of Game Design.* Pearson Education.

Andrade, G. R. (2021). Dynamic Game Balancing: an Evaluation of User Satisfaction. AAAI Conference on Artificial Intelligence.

Barr, P. (2008). *Video Game Values, Play as Human-Computer Interaction.* Wellington: Victoria University of Wellington.

Britannica. (2023). *Orthogonality*. Retrieved May 7, 2023, from https://www.britannica.com/science/orthogonality

Chen, Z. S. (2017). Player Skill Decomposition in Multiplayer Online Battle Arenas. Cornell University.

Fullerton, T. (2014). *Game Design Workshop: A Playcentric Approach to Creating Innovative Games* (3rd ed.). CRC Press.

Gravina, D. (2015). Procedural weapons generation for unreal tournament III. *IEEE Games Entertainment Media Conference (GEM)*, 1-8.

Hullet, K. N. (2012). Empirical analysis of user data in game software development. ACM.

Hunicke, R. (2005). The case for dynamic difficulty adjustment in games. Valencia: ResearchGate.

Kaplan, J. (2017). *Overwatch Forum*. Retrieved 03 20, 2023, from https://us.battle.net/forums/en/overwatch/topic/20757436132?page=3#post-47

Kavanagh, W. (2021). *Using probabilistic model checking to balance games.* Glasgow: University of Glasgow. doi:10.5525/gla.thesis.82618

Makin, O. B. (2017). Orthogonal analysis of StarCraft II for game balance. ACM.

People Make Games. (2019). Why the Sound of a Gun Had to Be Nerfed in Wolfenstein: Enemy Territory. United Kingdom: People Make Games. Retrieved from https://www.youtube.com/watch?v=RDxiuHdR_T4

Smith, H. (2003). Orthogonal Unit Differentiation GDC Talk.

Tekinbas, K. Z. (2004). *Rules of Play: Game Design Fundamentals.* MIT Press.

Ubisoft. (2022). *Y7S2 PRE-SEASON DESIGNER'S NOTES*. Retrieved May 7, 2023, from https://www.ubisoft.com/en-us/game/rainbow-six/siege/news-updates/68bhOPoegAwcwysIJ7rOui/y7s2-preseason-designers-notes

Westre, A. (2013). *Design Games for Architecture.* New York: Routledge.

Zagal, J. B. (2013). Dark Patterns in the Design of Games.