**Dank Heist: A Unity-based Multiplayer Bank Heist Game**

**Technical Report**

Lucy Brownless (lcb34), Joe Coker (jc2086), Joseph Denby (jd744), Marin-Gabriel Stoica (ms2247), Ameen Taha (akt22)



**Abstract**

*We set out to produce a quality video game using Unity Engine. In planning, we agreed that our game should meet a level of complexity that best shows our capabilities. Game features included animation, level design, artificial intelligence, network multiplayer among others to deliver on an immersive and enjoyable experience.*

*We achieved much of what we set out to do but acknowledge that the project fell short on fully realizing our ambition.*

**1. Introduction**

This videogame started as a passion project for most of our group members, as we always wanted to create something memorable and test our skills, and so it became our final year project to hit two goals at once. We also all enjoy videogames and have solid knowledge and expertise in video games culture, which provided us with an edge on understanding subtle mechanics and functionalities. Our best bet on making a game was through the usage of Unity, a powerful and practical tool engine for game development, which had many useful integrated features to make quality of life easier.

Now we had to pick a game idea to work on, and fortunately we had a rough idea on the concept: a multiplayer 3D bank heist. One of our major sources of inspiration was a game called payday2, which in many ways held a great number of similar ideas to our own.

At this point we had 2 more core features we had to implement to succeed, which were also arguably the most hard/ tedious to implement: multiplayer and logical behaviours. They both represented a big obstacle and many bug/problems which had to be accounted for, but they also provided great satisfaction and flavour to the game, as the gameplay will reveal.

**2. Background**

To create a comprehensive and sophisticated 3D multiplayer game, an in-depth exploration of the game development field was undertaken. This investigation focused on the Unity Game Engine, its capabilities, and its prevalence in the industry, the growing popularity of multiplayer games, and an examination of games with comparable themes to our game.

**2.1 Unity Game Engine**

As the second most utilized game engine, Unity is responsible for the development of approximately 13% of games in the current marketplace [1]. This prevalence highlights the engine's utility and offers an extensive foundation of source material for research purposes. Unity's popularity is largely attributable to its cross-platform availability, enabling developers to effortlessly deploy their games on a wide range of platforms, including Windows, MacOS, Android, and iOS [2][3]. Furthermore, the engine boasts a robust 3D graphics engine, physics engine, animation system, and scripting tools. For instance, the InputManager tool [4] has been employed in the present project to facilitate the association of inputs with corresponding actions in a menu.

An analysis of popular Unity-based games, such as Rust, Pillars of Eternity, and Risk of Rain 2, was conducted to glean insights into the development process. Singh and Kaur [5] offer a valuable perspective on game development in Unity, detailing core aspects of the Unity editor, such as the inspector and game view.

**2.2 Multiplayer Game Development**

The popularity of multiplayer games has surged, with a 2022 Unity report indicating that 72% of gamers engage in multiplayer games [6]. Consequently, the current project has adopted a multiplayer format to align with these trends. Multiplayer games facilitate interaction and collaborative or competitive play among users. The multiplayer landscape comprises various formats, including online multiplayer, massively multiplayer online (MMOs), and local multiplayer. The present project employs online multiplayer, accommodating up to four players from across the world to collaborate. This feature was implemented using Unity's integrated Netcode tool [7]. To further contextualize the project, a comparable game incorporating online multiplayer was examined.

**2.3 Payday 2**

Payday 2, a well-established first-person shooter video game that offers players the opportunity to participate in various heists, including bank robberies, either independently or in teams of up to four, was evaluated as a reference point for the current project. This analysis was motivated by the thematic similarity between the two games. Payday 2 has received positive reviews from both users and critics, achieving an 8/10 rating from IGN [8], indicating that the core heist theme and shooter-style gameplay are attractive to players. These elements were incorporated into our game to create a bank heist experience in which the player must evade the police while collecting as much cash loot as possible. Additionally, Payday 2 inspired the use of masks within our game, offering users more control and flexibility over the appearance of their characters.

**2.4 Enemy Behaviour**

In order to have dynamic gameplay, the current project incorporates enemy police behaviour. To optimize the implementation of this behaviour, an investigation of enemy behaviour in other games was conducted. Typically, games employ group AI script(s) to dictate overarching behaviour patterns [9]. Considerations for this aspect of the project included shooting, player pursuit, and a fear factor to introduce an element of unpredictability in the behaviour of individual police characters.

**3. Aims**

Upon conducting a comprehensive analysis of the project requirements, we developed an initial plan in which the workload was divided into three distinct stages.

* The first stage, referred to as the "skeleton", focused on essential features that that required complex systems to operate. These features were networking functionality, artificial intelligence for non-player characters, and a character controller.
* The second stage, known as the "muscle", involved all other essential features that did not require complex systems to operate, including but not limited to level design, firearm scripting, animations and user interface.
* The third and final stage, labelled the "skin", consisted of non-essential, supplementary features designed to enhance the game, such as grenades and a variety of law enforcement and civilian classes. This plan provided us with a clear understanding of the project's direction, with the primary goal of creating a multiplayer first-person shooter that offered a range of weapon and non-player classes for players to interact with based on their preferred gameplay style.

In hindsight, we realized that our initial goals were overly ambitious given our resources and capabilities. Despite our careful consideration of multiple factors during the planning phase, we underestimated the complexity introduced by the networking feature. As a result, we adjusted our objectives toward a less complex style of gameplay, ensuring the seamless functionality of both the skeleton and muscle stages.

**3.1 Networking**

The inclusion of multiplayer functionality was a crucial aspect of our game design, as shared experiences with friends can significantly enhance gameplay. With advancements in technologies like Photon [10] and Netcode [7], implementing online multiplayer has become more accessible. Our team opted for the Netcode package due to its seamless integration with Unity Engine and support for Unity Gaming Services [11].

Netcode facilitated the creation of game lobbies, network abstractions defining active play sessions. Lobbies are created with a designated host and an identifier code. Players are assigned to suitable rooms based on latency and player count, and once a lobby is full, it becomes inaccessible to others. Networked game events are then processed by the host player and shared with the others in the active session.

The initial implementation of lobbies was simplified by Unity's comprehensive documentation [7]. We developed a "Lobby Manager" class to handle connections to Unity services and player assignments to open lobbies. However, challenges arose when client-side animations were not properly synchronized, resulting in stiff, floating character representations. To resolve this issue, we utilized the "[ServerRpc]" attribute to communicate client-side events to the host, ensuring synchronization.

In summary, the successful integration of multiplayer functionality with Netcode and Unity Engine has established a robust foundation for our game. The implementation of lobbies and synchronized client-side animations demonstrates the potential of this system for creating engaging, shared gameplay experiences. Although networking and gameplay events have not been fully integrated, we are optimistic about future improvements to the system. The current implementation's reliability and potential inspire confidence in the multiplayer system's ability to provide an immersive and enjoyable experience for players.

**3.2 AI**

Non-player character (NPC) artificial intelligence (AI) is a key component in modern video game development, allowing game developers to create more immersive and engaging game worlds where NPCs behave in realistic and interesting ways. The method to control the NPCs we selected is called a behaviour trees, which are designed as hierarchical structures, where each level of the hierarchy represents a different level of abstraction in the NPC's behaviour.

At the root of each tree there would be a "root node" which would represent the NPCs top-level behaviour. The root node branches out into "child nodes" which would be more specific behaviours. Each node is a specific action like for example "retreat" and the tree would evaluate each node/action in a specific order based on the NPC's current state are world conditions. Once evaluated, each node would produce a weight, a value between 0-1, which represents how favourable each action is.

During the development of our system, we were pleased with the outcomes we achieved for several reasons. Firstly, our modular design facilitated the creation of new scripts, which we could easily add to the appropriate layer in the behaviour tree, streamlining the process of building complex behaviour trees. In addition, we implemented GUIDs to manage assets in Unity, which improved our workflow and minimized the risk of errors in our game projects. However, due to time constraints, we were unable to implement an automatic system for adding new nodes to the tree, necessitating the manual addition of nodes to the source code. Moreover, we encountered some UI problems where the trees would load incorrectly, resulting in a visually disorganized layout. Nonetheless, the trees continued to function as intended.

**3.3 Level Design**

In order to incorporate this feature in our game, we began by designing a floorplan for the bank manually. This design had to take into account the bank's size, the position of the vaults, and potential player routes. A level that is too small would make the gameplay too brief, while a level that is too large could become tedious for the players. To avoid these pitfalls, we decided to have two vaults located at opposite ends of the map, with different routes connecting them. Multiple pathways through rooms and hallways were included to allow players to traverse the map strategically, based on the positioning of enemy cops. The vaults were placed far from the entrance/exit door to prevent gameplay from being trivial and encourage players to explore the map.

To ensure the best possible level design, we went through several iterations of the design process. The final floorplan can be found in the corpus. We then implemented this feature into Unity by constructing walls and floors and then decorating each room by hand. Our team carefully considered where objects should be placed in the rooms to provide optimal gameplay. We placed desks and boxes in strategic locations to offer cover for players and cops while still allowing for traversable through-routes. To maintain the theme of a bank, we ensured that the level design remained realistic and immersive to the player. Finally, we filled the vaults with collectable money and different types of interactable prefabs, such as gold bars, large stacks, small stacks, and medium stacks of notes.

**3.4 Sound Design**

The implementation of this feature is crucial to provide a more immersive experience for players. To achieve this, our team first created an Audio Mixer to handle all sound inputs. The Audio Mixer contained three channels - Master Volume, Music Volume, and Effects Volume - to control the different types of audio in the game. We then created an Audio object in the scene view that held multiple Audio Source components, each connected to its corresponding Audio Channel. Music clips were linked to the Music channel, while effects like gun and menu sounds were linked to the Effects channel. The Master channel controlled both other channels simultaneously.

To allow players to save their audio settings, we used ScriptableObject, a serializable Unity class that can store significant amounts of data independently of scripts. We created the AudioProfile class, which is marked to appear in the asset menu. Each creation of a profile instantiates a new instance of the AudioProfile class, which contains the audio mixer's settings for each user. The User Interface accesses get/set methods and a save method to display this data via sliders and buttons.

Multiple scripts throughout the project needed to be modified to ensure that audio clips played at the appropriate times. For instance, the Weapon script was adjusted so that the gun sound effect played whenever the player fired a bullet, while the InteractableMoney script played a "money" audio clip whenever the player interacted with the money.

**3.5 User Interface**

This section outlines our development of a visually appealing and user-friendly User Interface (UI) for our game. Utilizing Unity's UI features, we created various menus and in-game elements, addressing challenges and improving the overall user experience. We detail the design and implementation of the Start, Options, Sound, and Score menus, as well as key in-game UI components.

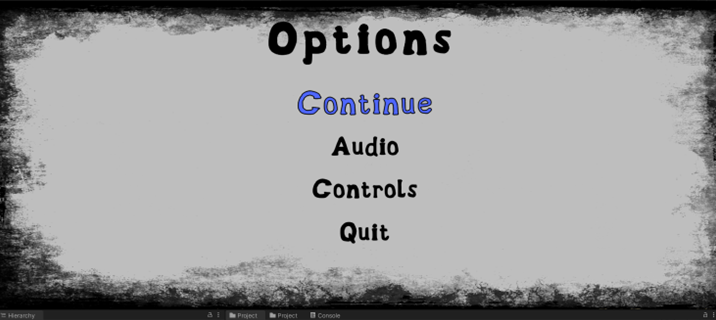
**3.5.1 Start Menu**



*Figure 1 - Start menu*

The Start Menu allows players to input their name and begin the game. The 'Play' button fetches the input, assigns it to a variable, and instantiates the player character, while the 'Settings' button links to the Options Menu.

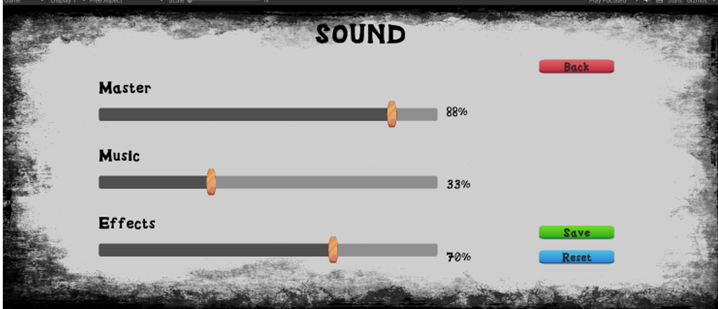
**3.5.2 Options Menu**



*Figure 4 - Option Menu, with user hovering over the Continue button*

The Options Menu comprises four buttons: Continue, Audio, Controls, and Exit. Players can return to the Start Menu, access the Sound Menu, or quit the game from this menu.

**3.5.3 Sound Menu**



*Figure 5 – Sound Menu*

We resolved an issue with inaudible audio at around 50% slider position by converting linear values to logarithmic and adjusting the signal voltage. The Save and Reset buttons enable players to modify and restore volume settings, while the Back button returns to the Options Menu.

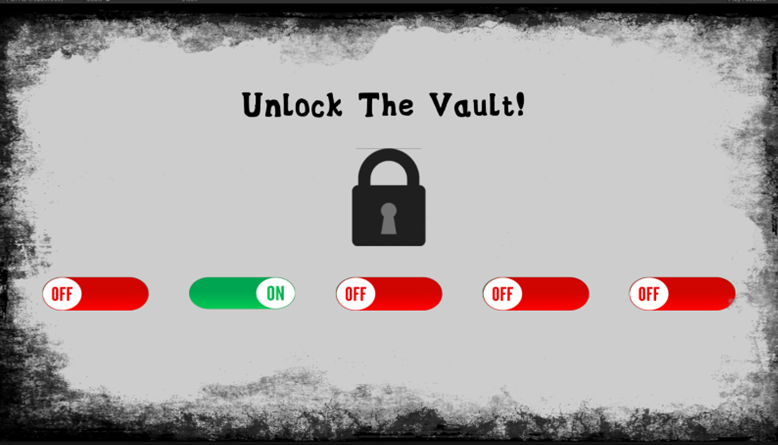
**3.5.4 Score Menu**



*Figure 6 - Scoreboard*

Activated at the game's end, the Score Menu displays total money collected, player name, and ranking. It uses PlayerPrefs to store and retrieve scores across gameplay sessions, sorting them in descending order for a ranking system.

**3.5.5 Minigame**



*Figure 7 - Locked Menu*

Players interact with vault doors to enable a minigame screen where they toggle switches to unlock the vault. Switches are randomized, and upon successful completion, the lock image changes to signify the unlocked state. A Coroutine briefly disables the vault door before returning the player to the main game.

**3.5.6 Player Health**

A red bar above the player's head represents their health, positioned by retrieving the player's onscreen coordinates and adjusting the y-value. A slider object reflects changes to the player's health value in the Character script, decreasing when the player takes damage.

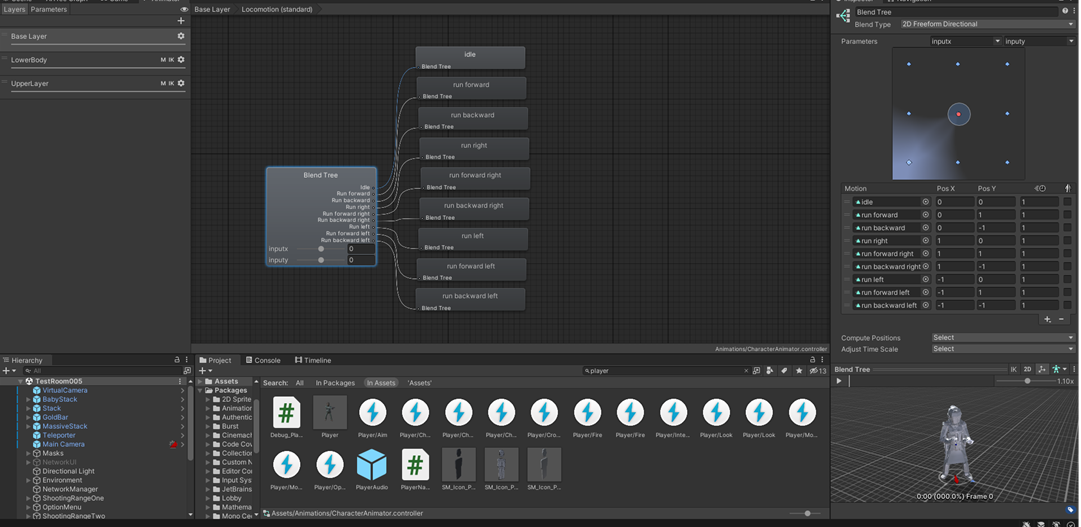
**3.5.6 In-Game UI Elements**



*Figure 9 - Wanted Level, Weapon Selected, Ammo, Health, Money Collected*

* Top right: Displays 'money collected,' fetching the money variable from the Character script and showing the amount picked up for three seconds.
* Top left: Shows the 'wanted level,' increasing as enemies are defeated. The GameManager class provides the wanted level value, and the badge animates and changes colour to indicate rising difficulty.
* Bottom left: Exhibits the player's weapons and ammo count. The currently selected weapon's ammo and clip variables are retrieved from the Weapon script and displayed in the AmmoUI.

**3.6 Animation**



*Figure 10 – Animation window*

Animations were incorporated into our game to depict player movements, such as walking, running, crouching, and firing. Unity's built-in animation features were used to achieve this. Specifically, animations for moving forward, backward, left, and right in both running and walking styles were required for the player movement. Initially, transitions were linked to the CharacterLocomotion script to ensure that each movement animation aligned with the user's input and direction of motion. However, this approach resulted in clunky-looking transitions between animations. To overcome this issue, a blend tree was employed to smoothly combine multiple animations by incorporating aspects of all of them simultaneously. Each animation's degree of contribution to the final result depended on the motion parameters. The following positions were assigned to each animation:

* Idle (no movement) – X:0 Y:0,
* Forward – X:0 Y:1,
* Backward - X:0 Y: -1,
* Left - X: -1 Y:0,
* Right - X:1 Y:0,
* Forward Left - X: -1 Y:1,
* Forward Right - X:1 Y:1,
* Backward Left - X: -1 Y: -1,
* Backward Right - X:1 Y: -1.

The animation blending employs linear interpolation, calculating each animation's average weights in the blend tree based on the user's x and y inputs. Two blend trees were used, one for running and the other for walking.

During the development of crouching animations, we encountered an issue where the animations did not contain the player character holding a weapon, which was a requirement for our project. To resolve this issue, masking was utilized to discard parts of the animation and display only the specified segments. To achieve this, layers were used, including a base layer that held the entire animation, an UpperBody layer containing the upper part of the character model's body animation, and a LowerBody layer containing the lower part. A blend tree for the crouching animations was designed in the UpperBody layer, blending the top half of the standard animations for running in any direction with the bottom half crouching animations. This method spliced the animations together, enabling a smooth transition. Similarly, the same process was employed with aiming animations, splicing them together with the standard LowerBody layer but in the opposite way.

To enable the animations to interact with other components of the game, we integrated parameters within our scripts, utilizing Boolean checks to determine the current state of the player, such as crouching, aiming or dead, and correspondingly instructed the animations to transition between these various states and blend trees.

**8. Conclusions**

**8.1 Overview**

As a group we developed a fun and immersive video game that includes a variety of features that demonstrate major aspects of modern game design.

**8.2 Final Product**

In summary, our project has successfully developed a video game that encompasses a diverse range of features. Through remote control of human characters and interaction with in-game systems such as doors, masks, money, and NPCs, players are provided with an engaging and immersive gaming experience. The incorporation of modular behaviour trees has contributed to the dynamic nature of the game, allowing for distinct behaviours of computer-controlled characters. Additionally, the smooth and realistic feel of character movement further enhances the overall gameplay experience.

The user interface provides clear and intuitive critical feedback while maintaining the game's aesthetic appeal. Considerable thought was given to balancing the size and functionality of the bank level, ensuring minimal dead-ends to facilitate smoother gameplay runs while avoiding repetitiveness. Lastly, the character customisation option available in the lobby, where each player can choose a unique mask. This adds a whimsical element to the game. Overall, our project has delivered a well-balanced and engaging experience that meets the needs of its intended audience.

**8.3 Improvements**

**8.3.1 AI tree**

We created an elegant solution to NPC decision making and it works. However, newly created nodes must be added manually to the source code. This is problematic for two main reasons. Firstly, each developed node takes additional time to integrate, time that otherwise would be better served elsewhere. Reason two, constant tampering of the source code allows for errors and mistakes to occur in the system. Given more time, we would have addressed node creation within the system itself. Node implementation would be integrated automatically and systematically.

**8.3.2 Rooms**

One problem we sought to address was the impact on performance in our AI algorithms. We expected thousands of objects in the final bank level. Some algorithms would require evaluating these objects and so encompassing them all simultaneously would be both impractical and unnecessary. We solved this issue by defining a Room class. These classes would define a zone on our map to which we could assign these objects and keep track of characters that are within their boundaries. This allowed NPCs to consider only elements within the room's influence.

Current implementation requires us as developers to manually instantiate an instance of a room and configure the boundaries. On a map the size of our bank level, this is very tedious and prone to potential error. It may be worth considering an alternative a approach to the problem. If not, an improvement to implementing new rooms would be required.

**8.3.3 Weapons**

Originally, we planned on having a multitude of different weapons to suit different play styles. Due to technical difficulties, for example animation changes, and time constraints, we scoped it down to a simple pistol.

**Acknowledgements**

We would like to thank Elena Botoeva for supervising our project.

**Bibliography**

[1] Toftedahl, M. (2019) *Which are the* *most commonly used game engines?*, *Game Developer*. Game Development. Available at: <https://www.gamedeveloper.com/production/which-are-the-most-commonly-used-game-engines-> (Accessed: March 29, 2023).

[2] *What makes unity so popular in game development?* (2022) *Arnia*. Arnia Software. Available at: <https://www.arnia.com/what-makes-unity-so-popular-in-game-development/> (Accessed: March 29, 2023).

[3] *Multiplatform game development for the future*. Unity. Available at: <https://unity.com/solutions/multiplatform> (Accessed: March 29, 2023).

[4] *Input manager* (2021) *Unity*. Available at: <https://docs.unity3d.com/Manual/class-InputManager.html> (Accessed: March 29, 2023).

[5] Singh, S. and Kaur, A. (2022) “Game development using Unity Game Engine,” *2022 3rd International Conference on Computing, Analytics and Networks (ICAN)* [Preprint]. Available at: <https://doi.org/10.1109/ican56228.2022.10007155>.

[6] Efendioğlu, A. (2022) *Unity report states 77% of gamers play multiplayer games*, *Game Industry News*. mobidictum. Available at: <https://mobidictum.com/game-industry/unity-report-states-77-percent-gamers-play-multiplayer/> (Accessed: March 29, 2023).

[7] *About Netcode for gameobjects: Unity Multiplayer Networking* (no date) *Unity*. Unity. Available at: <https://docs-multiplayer.unity3d.com/netcode/current/about> (Accessed: March 29, 2023).

[8] Ingenito, V. (2013) *Payday 2 review*, *IGN*. Available at: <https://www.ign.com/articles/2013/08/13/payday-2-review> (Accessed: March 29, 2023).

[9] Bart VossenBloggerMay 04, 2015 (2015) *Enemy design and enemy AI for Melee Combat Systems*, *Game Developer*. Available at: <https://www.gamedeveloper.com/design/enemy-design-and-enemy-ai-for-melee-combat-systems> (Accessed: March 29, 2023).

[10] *Introduction | Photon Engine* (no date) *Photon*. Photon. Available at: [Introduction | Photon Engine](https://doc.photonengine.com/pun/current/getting-started/pun-intro) (Accessed: April 3, 2023).

[11] *Unity Gaming Services: Live Game Development Platform*. Unity. Available at: <https://unity.com/solutions/gaming-services> (Accessed: April 3, 2023).