How does designing a game around a diegetic interface influence traditional game design choices and how is the player experience affected?

Tyler Evans

Contents

1. Introduction

‘Dead End’ is a first-person zombie survival game where the player must venture through an abandoned city to repair their broken-down truck. The game has been developed as an outcome on the research of diegetic interfaces in interactive media. The game explores how traditional interfaces, such as the heads-up display (HUD) and other user interfaces can be displayed diegetically, investigating the impact on traditional game traditional game design choices and exploring how the player experience, primarily relating to usability, is impacted. Desk research has been conducted looking at related academic literature on the impact of the HUD for various elements of the player experience, as well as investigating how previous interactive media have implemented diegetic systems. From this research, a set of functional and non-functional requirements have been detailed in order to establish the features of the artefact. The design and development of the game is then outlined, explaining the main features of the project and justifying the choices made. A series of user testing was conducted to determine how the system matches the intended outcomes and to verify the impact on the user experience. The resultant artefact was determined successful in providing a diegetic interface that was usable, and critically enjoyable, for players.

2. Background

A greater level of immersion in video games is something that is constantly pursued by players. While great improvements have been achieved in the level of graphical fidelity and lighting quality in video games across the years as technical limitations are greatly reduced, it sometimes may not be enough for some players. As a result, players often look to enhance their experience by disabling the HUD. Most video games during gameplay feature a main action scene, where the player can interact with their avatar, enemies or other targets. A HUD is often superimposed onto the main action scene, providing key contextual information about the player’s status, the game’s current condition, tutorial messages and more depending on the game’s genre (Caroux, L. and Isbister, K., 2016). Despite the importance of the HUD in most cases, some players might find that the HUD clutters the visual interface to the point where it detracts from the immersive experience. This effect has garnered a great amount of attention online. Videos like *German Trench Assault (No Hud Immersion) – Battlefield 1* (Gunnar001, 2016) have amassed upwards of ten million views showcasing HUD-less multiplayer gameplay, and articles like *Games That Are Best Played With No HUD* (Lewis, E., 2025) provide a list of games where a HUD-less experience can be enjoyed by players seeking a greater level immersion. Some games may also offer ‘diegetic’ elements within their interface. Diegetic interfaces relate to the narrative of the game, where elements can be seen and interacted by the player-character, as opposed to non-diegetic interfaces, like the game’s HUD (Cairns, et al., 2015). It is therefore necessary to evaluate the impact of the HUD on usability and immersion, and to determine if a diegetic experience can deliver an enjoyable and engaging experience.

2.1. Literature Review

The HUD has been frequently discussed in academic literature, with studies primarily looking at player performance, user experience and immersion. Caroux, P. and Isbister, K. (2016) conducted two experiments in their research, the first of which looking at the relationship between player expertise and HUD usage in two different genres, while the second experiment identified how HUD composition influenced player experience according to player expertise and game genre. Both experiments were conducted in a first-person shooter (FPS) or a real-time strategy (RTS) game. Overall, it was determined that a permanent HUD within the visual interface of a virtual environment will improve the understanding of the environment by the user. The composition and spatial organisation of elements also had importance, with elements being displayed in a line at the bottom of the visual interface being most preferred amongst players. This study outlined the importance of the HUD overall to player experience, especially in more HUD-dependent experiences like RTS games. Caroux, L. et al (2021) looks at the physical and semantic characteristics of the HUD in relation to player performance and experience, conducting experiments changing the size, colour and composition of the HUD. It found that the presence or absence of elements of information necessary for the game's main task did have a significant effect on the player's experience, with performance being diminished. This again shows the importance of the HUD when it is used in a manner where the game is dependent on its use. However, other studies have shown that removing the HUD can be beneficial to the player experience, especially regarding immersion. Cairns, P. et al. (2015) conducted a study where participants played through a level of a FPS game with either a diegetic interface without a HUD, or a non-diegetic interface with the game's original HUD. The study found that the removal of the HUD did not present an obstacle for initial engagement and enjoyment, and the removal of the HUD increased immersion for expert players. This has been backed up by Roysid, H.A., Pangestu, A.Y., and Akbar, M.I (2021), performing a similar study and applying it to two separate role-playing games (RPG). They too found that turning off the HUD had an impact on increasing the level of immersion.

A frequent trend across studies was that when a game is played without the HUD, the level of immersion the player experiences improve. This comes at the trade off in the level of understanding the player may have for the game's environment. Novice players may also struggle more without the input hints that a game's HUD may provide when performing certain tasks. Despite these limitations, information may be displayed diegetically without requiring a HUD. The previous studies mentioned all performed their experiments with pre-existing games in which the HUD is a purpose-built component of the game. If diegetic interfaces are to be explored further, then research should identify how diegetic elements can be designed on their own as opposed to disabling the traditional HUD-based interface, as well as reviewing feedback from participants. Some studies have performed research identifying how certain information could be displayed through diegesis. Peacocke, M. et al. (2015) conducted a study where the ammunition display in an FPS game were displayed in five different capacities, three of which were on the HUD and two of which were diegetic. It was found that players performed the best when the ammunition was

presented using a diegetic “number-in-game” display; most participants also preferred this form of display. In addition to this, Köhle, K. et al. (2021) performed research which used three different health indicators in a virtual reality (VR) shooter game and gathered diegetic interfaces would be preferred in singleplayer and story-driven experiences, whereas non-diegetic interfaces may be useful in competitive multiplayer games. Albeit in VR, it is still worthy to note that diegetic interfaces can be worthwhile in delivering a consistent and immersive experience. However, research by Pfister, L. and Ghellal, S. (2018) highlight the importance of making sure diegetic interfaces are understandable and do not confuse the player. They experimented the use of a diegetic interface versus a non-diegetic interface in a 2D platformer, concluding that the non-diegetic interface was perceived on a higher level of immersion. It was noted that participants did not understand the diegetic elements clearly, and some elements may have been unnoticed. Across all of this research, it is apparent that diegetic interfaces provide more immersive experiences, as long as it is delivered in an understandable and effective manner. It is therefore important to cross-reference existing interactive media and see how previous products have implemented diegetic elements in their interfaces.

2.2. Previous Interactive Media

There are many different examples of interactive media that have incorporated diegetic systems within their visual interfaces, presenting information consistently and clear way. *Dead Space* (Motive, 2023) is a survival-horror game which fully implements a diegetic interface throughout the gameplay experience. The game does not feature a permanent HUD, and the player-character’s health is displayed on their spine, along with ammunition displays appearing on the character’s weapon when aiming. Because the game is science-fiction, it can get around limitations regarding menus and excuse them as in-game holograms, but as they do fit within the game world and the player-character is able to interact with them, this does fit the definition of diegesis. *Fallout 4* (Bethesda Game Studios, 2015), albeit not a fully diegetic experience, does provide some diegetic elements within the game. The ‘Pip-Boy’ is an in-game wearable device that the player-character interacts with to manage their health, inventory and more. This cleverly incorporates traditional menus into an in-world device and provides a strong basis for a method to manage these elements diegetically. Lastly, *Metro Exodus* (4A Games, 2019) features some diegetic elements to convey certain information to the player, such as with an in-game wristwatch conveying compass direction and timekeeping or gas masks becoming cracked or fogging to convey certain dangers.

2.3. Research Motivation

The motivation for this research is to expand on the work done by previous researchers and look to fill in some of the gaps from these studies. Earlier studies have looked at the effects on the player experience when the HUD is removed, but these are often for games that are dependent on the HUD for its experience. Other studies have looked at specific methods of displaying certain kinds of information diegetically, such as the player’s health or ammunition levels, but it is worth noting that these studies have only focused on new visual methods for displaying this

information and have not explored how they could also be displayed sonically. Previous research also seems to have only focused on HUD replacements, rather than looking at how certain game mechanics, such as inventory management or crafting systems, could be redesigned using a diegetic interface without having to use menus. It is evident that there is a clear gap between these studies to explore more about how certain information can be displayed without the HUD through visuals and audio, as well as how game mechanics could be implemented diegetically.

3. Requirements

As an outcome of the desk research, which involved looking at relevant academic literature and previous interactive media artefacts, a set of functional and non-functional requirements were set out for the artefact. Note that the following requirements were frequently adjusted as the project developed depending on the progress and changes in scope.

3.1. Functional Requirements

Requirement	**Description**
FR1	The game should include a small but detailed 3D environment.
FR2	The game should feature enemies.
FR3	The 3D environment should feature enemies and items.
FR4	The 3D environment should include an exit.
FR5	The game should have a first-person player-character.
FR6	The player’s health should be displayed diegetically.
FR7	Other elements of the player’s status should be displayed diegetically.
FR8	The player should have the ability to equip and use weapons.
FR9	The game should have an inventory system.
FR10	The player should be able to pick up and use items.
FR11	The game should have a crafting system.
FR12	The game should include a sufficient level of crafting recipes to facilitate a small level of progression.
FR13	The game’s mechanics (inventory, crafting, etc.) should be implemented diegetically with relevant models, animations and sound.
FR14	The game should include a journal to allow the player to view objectives and log recipes.

The role of the functional requirements was to facilitate an experience with a diegetic interface. Therefore, the player-character needed to include different variables that

enables this. A first-person perspective was chosen over a third-person perspective as to allow the player to see from the eyes of the main-character. While not impossible in a third-person perspective, a first-person perspective makes it easier to display the player's condition diegetically, such as health, movement or ammunition levels. To allow these variables to be changed, the player needed to encounter enemies within the game's environment. In addition to this, traditional survival mechanics, like inventory and crafting systems, were features that were ideal to display diegetically. These mechanics needed to be adjoined with an item system, where the player can pick up and use items. To support all of these mechanics, the game's environment needed to be the correct size and detail to place enough enemies to challenge the player, as well as to place items in a consistent manner that facilitates a small level of progression. Lastly, the journal draws inspiration from Fallout 4's 'Pip-Boy,' where the player-character can view their objectives and recipes as they continually update throughout the game.

3.2. Non-Functional Requirements

Requirement	**Description**
NFR1	The game should be intuitive and easy to learn.
NFR2	The game should be immersive (consistent visuals, fluid mechanics)
NFR3	The game should be reliable and free from bugs.

Creating an artefact that consists of a fully diegetic experience has many limitations that can lead to a possibly frustrating experience where the player may find difficulty in learning the controls and mechanics. Therefore, the control scheme needs to be simple and easy to learn, feedback mechanisms need to be consistent and understandable and game mechanics should be intuitive. A game with consistent visuals and fluid mechanics would be ideal to allow the player to become engrossed within the game's environment and fully interact with its systems. A reliable and bug-free experience was also preferred to ensure that the player can play through the experience without intervention from unexpected errors. While a game without bugs cannot be guaranteed, user testing would outline any potential errors present within the experience.

4. Design and Development

The solution is titled *Dead End*, a zombie survival game made in the Unity game engine, including a complete diegetic interface without featuring a HUD. The player must venture into an abandoned city riddled with zombies to find parts to repair their broken-down truck. The player can pick up and use items, combine them together to craft new items, and store them in their inventory. The following section outlines the major features of the experience and provides justification for its implementation.

4.1. Player Condition

Player health in most games is usually represented by an element on the HUD, such as with a ‘health bar.’ This can sometimes be used in parallel with a diegetic form of displaying the player’s health, visually through the use of a blood overlay, or sonically through heart-beat or heavy breathing sound effects. In Dead End, the HUD is not an option, so more emphasis has been placed on health being displayed diegetically. As seen in Figure 1, Dead End uses a red vignette overlay to signify low health. In addition, a heart-beat sound effect begins to play when the player is at a low health. This method of displaying the player’s health was chosen due to its conventional use in other games, and the auditory and visual effects paired with each other should easily signify to player’s that they are at a low health.

Figure 1. Low Health Overlay.

The script, ‘PlayerManager’ manages all of the variables and functions associated with the player character. Inside this script, the player has an integer value called ‘playerHealth,’ which equals 100. Inside of the ‘Update’ function, which is called every frame, the player health is checked in an if statement to see if it goes below 20. Once it does, the heart-beat sound effect begins playing. In addition to this, a function is called in the ‘CanvasManager’ script, which displays the overlay on the canvas. Inside this script, the health is also checked to see if it goes below 50. If it does, the overlay is enabled, and depending on the health’s value, the opacity of the overlay changes. An inverse lerp function is utilised to achieve this (see Figure 2).

```
1 reference
public void DisplayInjuredOverlay(int health)
{
    if (health < 50)
    {
        injuredOverlay.enabled = true;

        // The following block was generated by ChatGPT to change opacity of the injured overlay based on health
        // Author: ChatGPT
        // Date: 18/02/26
        // Location: https://chatgpt.com/
        float alpha = 1f - Mathf.InverseLerp(0f, 50f, health);
        Color c = injuredOverlay.color;
        c.a = alpha;
        injuredOverlay.color = c;
    }
    else
    {
        injuredOverlay.enabled = false;
    }
}
```

Figure 2. Implementation of the 'DisplayInjuredOverlay' function in 'CanvasManager.cs.'

4.2. Item System and Outlines

The item system is an essential part of the project, as it is the basis for the rest of the main mechanics to function. Because of this, the item system evolved a lot through the course of development due to its complexity and foundational nature to the rest of the project. To begin with, the player needs to complete the simple task of picking up an item. In the early stages of development, placeholder shapes were used to test this functionality (see Figure 3). To do this, a raycast needs to be fired from the direction of the camera to the object. To differentiate objects that can be picked up versus regular game objects, a layer called ‘Pickupable’ is used. Afterwards, the object that the raycast hits is checked to see if it is within this layer. If it is, the item is parented to a game object called ‘HoldPoint,’ which is attached to the first-person controller. The item transform is then updated to match the hold point’s transform and the item’s physics within the rigidbody component are disabled. This can be seen in Figure 4.

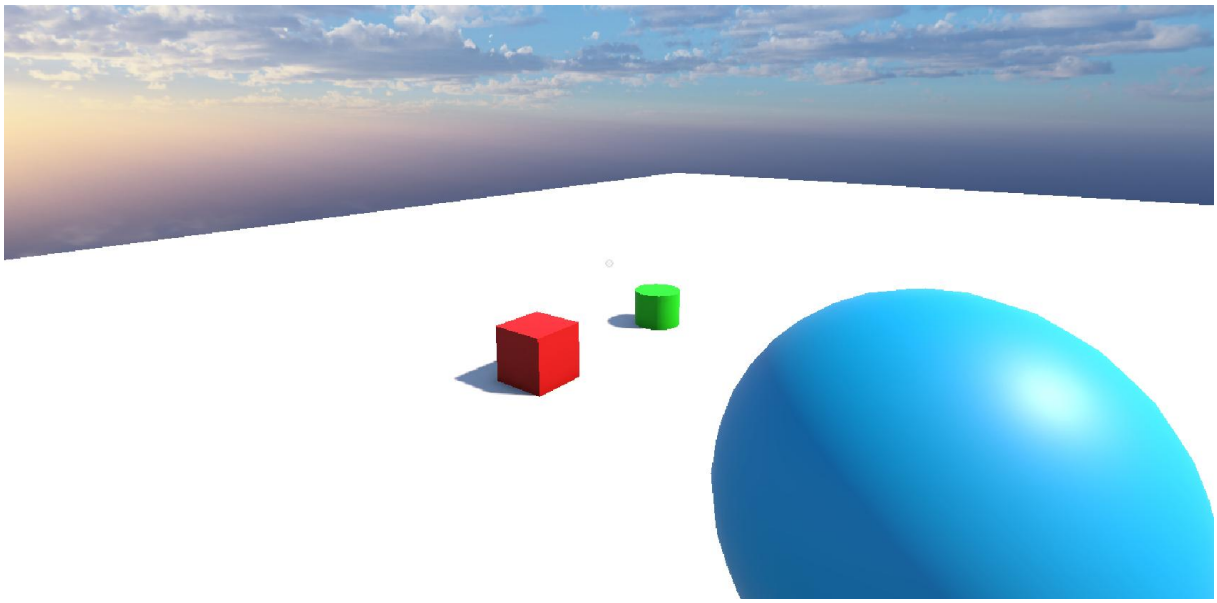

Figure 3. Early demonstration of picking up items with a blue ball being held.

```
1 reference
void HoldItem()
{
    RaycastHit hit;

    if (Physics.Raycast(camPos.position, camPos.forward, out hit, 5.0f))
    {
        if (hit.transform.gameObject.layer == LayerMask.NameToLayer("Pickupable"))
        {
            Pickupable pickup = hit.collider.GetComponent<Pickupable>();

            if (pickup != null)
            {
                pickup.PickUp(holdPoint);
                currentHeldItem = pickup;
            }
        }
    }
}
```

Figure 4. Early implementation of the 'HoldItem' function in 'PlayerManager.cs'

This early implementation of the item system exposed some potential usability problems. During this phase of development, a crosshair was needed to in order to precisely pick up items. However, the use of a crosshair in this project is not permitted as it is displayed on the HUD. Therefore, this problem needed to be addressed through another method. Because of this, the asset ‘Quick Outline’ from Unity’s asset store was used to display a white outline around the border of an item whenever the camera is looking at it (see Figure 5). This method can be found in other games where items can be picked up, especially in VR, and provides useful visual feedback to make this interaction easier for the player. In addition to the outline, interactable items have a small glow to differentiate themselves from general props within the game world.

Figure 5. Item outline.

The game consists of 26 unique items that they can interact with. Each of these items have their own prefab to allow items to have consistent behaviour and variables across all instances of the item. Some items have their own unique functionality and therefore require their own scripts to execute these interactions. Therefore, there is one main class called ‘Pickupable.’ Any items that require their own functionality will inherit from this parent class as to allow the item to be picked up as normal. Most child classes override the ‘ItemFunction’ function, which is checked in the ‘PlayerManager’ script when an item is being held. This allows items, such as weapons or healing items, to be interacted with when a button is pressed. The following sections outline specific items with unique interactions.

4.2.1. Melee Weapons

To counteract the zombies that are present in the game, there is a need for the player to use weapons. There are four melee weapons that the player can use. These include the Stone Knife, Reinforced Knife, Stone Axe and Reinforced Axe. Both the knives and axes have their own script, respectively called ‘Knife’ and ‘Axe,’ and both override ‘ItemFunction’ to allow the player to use the weapon when the left mouse button is pressed. In both scripts, a raycast is fired outwards from the direction of the camera. The raycast is then checked to see which layer the weapon hits. If the default layer is hit, a sound effect is played. If an item is hit, a force is added to the item to make it move, along with a sound effect. If a zombie is hit, then a function is called in the zombie script to damage the zombie. Depending on the weapon, this does a different amount of damage. Both scripts use coroutines to play the respective animations for knives and axes, as well as align the raycast code with the animation. A Boolean variable called ‘canHit’ is used in both scripts to allow the full animation to play and to stop the user from repeatedly attacking with the weapon.

4.2.2. Ranged Weapons and Ammunition

There are two ranged weapons that the player can use, including the handgun and the shotgun. These both have their own respective scripts, called ‘Handgun’ and ‘Shotgun’. Both scripts shoot when the left mouse button is pressed. This too shoots a raycast in the direction of the camera and checks the layer which it hits. If the default layer is hit, a bullet impact game object is instantiated on the surface of the object to provide visual feedback about direction of the bullet. If an item is hit, a stronger force is added to the item to make it move, again to improve the feedback of using the weapon. If the zombie layer is hit, then the function to damage the zombie is called. The raycasts on the shotgun fires at half the distance compared to the handgun. As opposed to the single shot from a handgun, the shotgun disperses eight pellets. This is done by iterating through a loop 8 times and adding a random spread for each pellet. In addition to shooting, each gun can aim using right click. This moves the weapon from the ‘HoldPoint’ game object to another game object called ‘AimPoint’. This ensures that the weapon’s sight is aligned with the centre of the player’s view, as to compensate for the lack of a crosshair (see Figure 6).

Figure 6. Aimed shotgun.

When a gun is out of ammunition, an empty gun sound effect is played. Each gun has a variable, called ‘magAmmo,’ which is used to check the number of bullets left in a gun’s magazine. The player manager also has its own variables, called ‘handgunAmmo’ and ‘shotgunAmmo’, and checks the overall amount of ammo that the player has for the respective gun. When the player presses the ‘R’ key, an if statement checks if the player has ammunition to reload. If so, the reload animation is played, the gun’s magazine is filled, and the number overall bullets for that gun is subtracted. This process for the handgun can be seen in Figure 7. The handgun has 12 bullets in a magazine, while the shotgun has 6 bullets. The shotgun code differs slightly as a different animation is played depending on how many shotgun bullets need to be reloaded. To acquire more ammunition, the respective handgun and shotgun ammunition items can be interacted with to add to the player’s overall ammunition count.

```
if (Input.GetKeyDown(KeyCode.R) && !isReloading && playerManager.handgunAmmo > 0 && magAmmo < 12)
{
    canShoot = false;
    isReloading = true;
    StartCoroutine(ReloadAnimation());

    int bulletsNeeded = 12 - magAmmo;
    int bulletsToLoad = Mathf.Min(bulletsNeeded, playerManager.handgunAmmo);

    magAmmo += bulletsToLoad;
    playerManager.handgunAmmo -= bulletsToLoad;
}
```

Figure 7. Implementation of reloading in ‘Handgun.cs.’

4.2.3. Healing and Food Items

There are two healing items and two food items that can be found in the experience. These items were included as enemies will certainly damage the player through the

experience, therefore balancing the difficulty to make the experience more enjoyable. The bandages provide the player with 50 points of health, provided that they are not already at full health. The med kit functions the same but provides the player with a full 100 points of health. Raw meat can be eaten which will subtract 20 points of health from the player, while cooked meat can be ate and provide the player with 50 points of health. The player can interact with these items by pressing the left mouse button, where they will add or subtract the amount of health depending on the item, play the associated sound effect and then be destroyed.

4.2.4. Campfire

The campfire is a unique item which can be interacted with. However, this item cannot be picked up. The campfire can either be active or inactive, and only one campfire can be active at a time. When the campfire is activated, the player's spawn point is recorded, so that when the player dies, they will respawn at the position of their activated campfire. This feature was included as the map of the game is a moderate size, and players may find it useful to set their spawn point as to not traverse the entire length of the map when a zombie kills them.

4.3. Inventory System

To allow the user to store the items that they collect, an inventory system is required. Diegetic inventory systems are not common in existing interactive media, and therefore a lot of thought was required into how this system could be implemented. An early concept for the inventory was a physics-based inventory system. The user would place their backpack on the ground and physically drop any items inside of it. The user would then be able to pick up the backpack and place it anywhere they required. Similar systems can be seen in *Cairn* (The Game Bakers, 2026), where items are freely placed within the space of the backpack, albeit represented two-dimensionally and in a non-diegetic menu. Another approach was a grid-based system where the player's camera would look inside the backpack and be able to interact with an inventory. This grid-based system would follow a system similar to extraction shooters, where items of different sizes can be placed inside of a grid. Figure 8 contains a depiction of how these interactions might have worked.

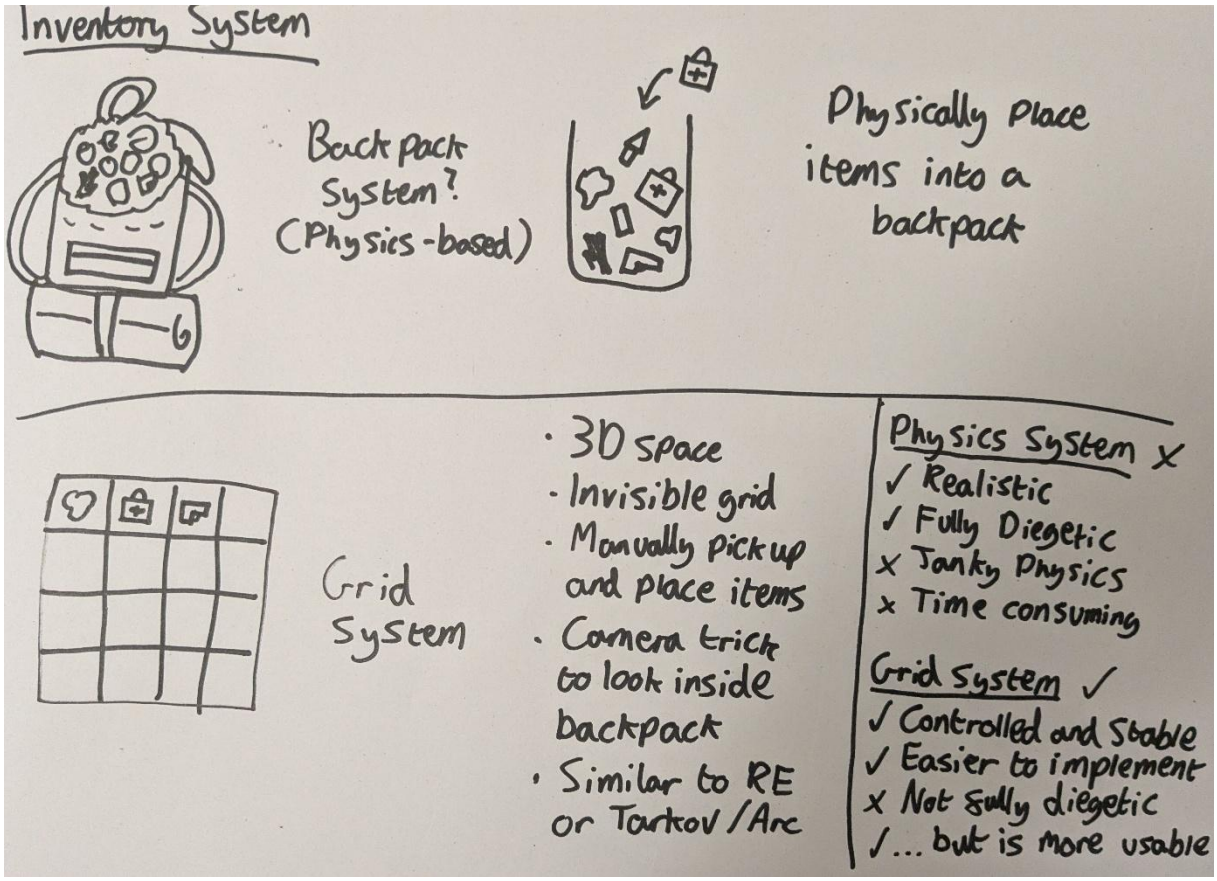

Figure 8. Sketch showing potential designs to the inventory system.

While a physics-based system would have been interesting to explore further, it was concluded that this system would be too technically complex and that the physics may be unreliable when multiple items are stacked upon one another in a tightly confined space. A grid-based system like the one pictured in Figure 8 would not have fully been diegetic as it would effectively be done through a menu. As a result, a new outcome that was inspired by both of these previous approaches was chosen. The idea of placing the backpack was retained along with the backpack being represented as a 3x3 grid, where the player can place items in any of the inventory slots. This retained the idea of placing the backpack on the ground, while also removing the concern of unpredictable physics interactions. This grid would be displayed in the game world instead of a hidden menu and therefore remain fully diegetic. This system is inspired by other inventory systems that can be seen in *Peak* (Aggro Crab, 2025), where a backpack is physically placed on the ground, but the inventory itself is represented in the game world rather than in an interface on the HUD. Figure 9 shows an early implementation of this system with the use of placeholder shapes representing the items and inventory slots.

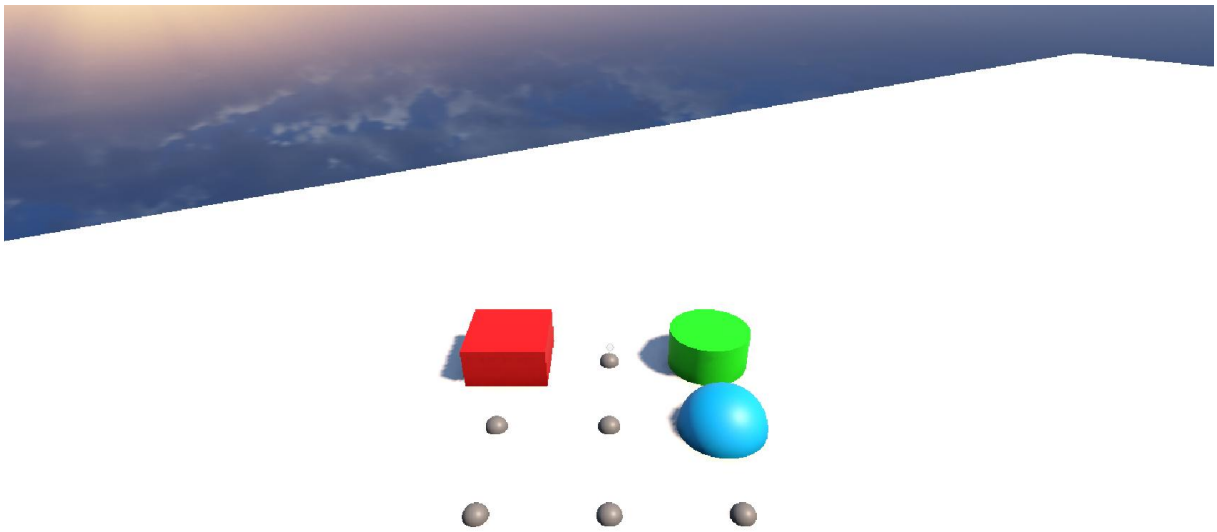

Figure 9. Early demonstration of the inventory system.

The implementation of this system is similar to picking up items. The player can place the backpack by pressing the ‘Tab’ key. If the player is holding an item and presses the ‘E’ key while looking at an empty inventory slot, the item will be placed into the inventory slot. This is achieved in the ‘PlayerManager’ script by firing a raycast from the direction of the player’s camera and checking if the raycast hits the ‘InventorySlot’ layer. If it does, it then checks whether the inventory slot is empty or not by seeing how many children are within the inventory slot’s game object. If it is empty, then the item is stored by parenting it to the inventory slot. This implementation can be seen in Figure 10. If the player wishes to pick up an item from the backpack, then this can be done like any other item in the game world.

```
private bool StoreItem()
{
    RaycastHit hit;

    if (Physics.Raycast(camPos.position, camPos.forward, out hit, 3.0f, inventoryLayer))
    {
        if (hit.transform.gameObject.layer == LayerMask.NameToLayer("InventorySlot"))
        {
            Transform inventorySlotPoint = hit.transform;

            if (inventorySlotPoint.parent.childCount == 1)
            {
                currentHeldItem.Store(inventorySlotPoint.parent);

                currentHeldItem = null;

                AudioManager.instance.PlaySound("Store Item");

                return true;
            }
        }
    }

    return false;
}
```

Figure 10. Implementation of the 'StoreItem' function in 'PlayerManager.cs.'

Further refinement was required to the visuals of the backpack system in order to improve the usability of the system for the player. The early implementation of the system in Figure 8 was difficult to interact with, with a small inventory slot that required a lot of precision to access along with a general lack of visual feedback. The inventory could have been displayed similarly to *Sons of the Forest* (Endnight Games, 2023), where a backpack is displayed flat on the ground with items atop of it. However, a simpler implementation was preferred, and therefore a grid-based system using outlines, as seen with picking up items, provided a system that was easy to interact with and gave consistent visual feedback to inform the player of the slot they are interacting with. The inventory slots appear with a black outline as default. When the player is holding an item and looking at an empty slot, the slot appears with a white outline to signify that an item can be placed. When an item is placed, the outline is no longer present, with the item now appearing in the slot. These visuals were paired with a backpack model that acted similar to items, visually informing the player that they are interacting with the inventory. When the user presses the ‘E’ key on the backpack object, this inventory is closed, offering another way to interact with the inventory. While this system is inspired by a number of different implementations in previous interactive media, it is unique in how it diegetically displays the inventory in the game world and does so in a way that is intuitive and logical.

Figure 11. Demonstration of the inventory system

4.4. Crafting System

In addition to the item and inventory systems, the crafting system exists to facilitate a level of progression throughout the game, with the player being able to craft weapons and upgrade them, create healing items to assist in their experience, and craft ammunition for the weapons they find. The crafting system is also another system that needed to be displayed diegetically in order to fit the requirements of the project. Therefore, a thoughtful design was required to ensure the interaction made sense. A

number of different approaches were considered as part of this design process. One approach was another physics-based system, where the player would physically drop items in the vicinity of one another. If the items had a crafting recipe, then they would combine to create the new item. This idea was abandoned because it may lead to unintended items being crafted and it would not have felt reliable due a lack of player agency over the interaction. Figure 12 contains a depiction for how this interaction may have worked.

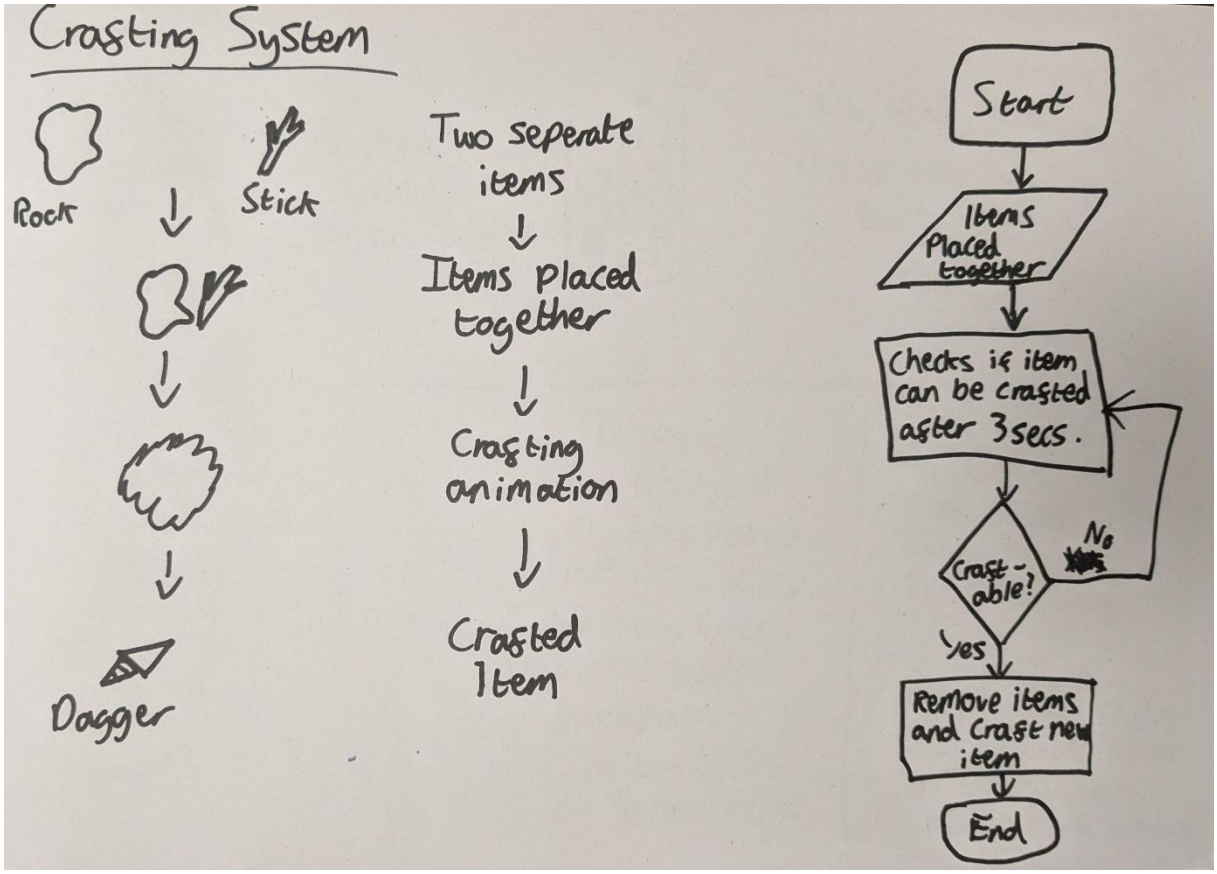

Figure 12. Sketch showing a potential design to the crafting system.

Ultimately, these ideas were scrapped in favour of a combination style crafting system, seen in games like *Resident Evil 7: Biohazard* (Capcom, 2017), where two items can be combined together to create a new item. Whereas traditional systems would have this interaction inside of an inventory's menu, Dead End does this interaction diegetically within the game world. This system was preferred as the interaction for the player would not be too complex given only two items need to be combined together and therefore should be easy to learn. Despite the interaction itself not being too complex, the implementation required a lot of research as to allow the system to distinguish between items and craft the respective recipes. Therefore, some discussion with generative AI was used to determine an appropriate solution for this complex problem. This exchange led to the use of scriptable objects within Unity. The game contains two scriptable objects, with one being used for items and the other being used for recipes. Item scriptable objects contain data relating to the item's name, the item's prefab, a Boolean called 'isEssential,' and another Boolean called 'isQuestItem'. 'isEssential' determines whether the item can be removed when

the player dies and ‘isQuestItem’ determines whether the item is a quest item. Recipe scriptable objects contain data relating to crafting recipes. It determines which two items can be combined together and the resulting item. A Boolean, ‘isCrafted’, determines whether this recipe has been crafted or not. These scriptable objects are useful as each item contains information that allows them to be distinguished from one another and therefore be used for crafting. The script ‘CraftingManager’ contains an array with each recipe. In this script, the function ‘TryCombine’ loops through the array of recipes and determines whether two items can be crafted. If so, it returns the result of the crafting recipe. This implementation can be seen in Figure 13. In the ‘PlayerManager’ this function is referenced in the ‘CombineItem’ function. This function is only executed when the player presses the ‘E’ key. It fires a raycast at objects within the ‘Pickupable’ layer and determines whether the two objects can be combined together. If they can, then the new item will be instantiated, and the two initial objects will be removed.

```
public Item TryCombine(Item item1, Item item2, bool isChecking)
{
    foreach (var recipe in recipes)
    {
        bool match =
            (recipe.itemA == item1 && recipe.itemB == item2) ||
            (recipe.itemA == item2 && recipe.itemB == item1);

        if (match)
        {
            if (recipe.isCrafted == false && !isChecking)
            {
                recipe.isCrafted = true;
                AudioManager.instance.PlaySound("New Recipe");
            }

            return recipe.result;
        }
    }

    return null;
}
```

Figure 13. Implementation of the 'TryCombine' function in 'CraftingManager.cs.'

In addition to the inventory system, the crafting system required additional refinement to enhance the user experience and provide greater visual feedback to when two items can be crafted together. In ‘CheckItemCrafting,’ the function fires a raycast at objects in the ‘Pickupable’ layer. It then checks if the item the player is looking at can be combined with the item they are holding. If so, the item the player is holding will start simulating a shaking effect to show that an item can be combined, in addition to a white outline around the item that can be combined with (see Figure 14).

Figure 14. Demonstration of the crafting system.

4.5. Zombie AI

Enemy AI is required to provide the experience with the necessary challenge of defeating enemies as they progress through the game world, while also encouraging the interaction of the previous mechanics. The experience consists of 24 zombies placed around the game's map, with a total of 15 different variants of models. Enemies use Unity's AI navigation tools to traverse through the game's environment. Each zombie has their own Nav Mesh Agent component which determines specific characteristics of the zombie. A Nav Mesh Surface has been baked to allow agents to traverse the map and avoid any obstacles (see Figure 15).

Figure 15. View of the NavMesh in the Dead End's environment from the Unity Editor.

The 'Zombie' script contains all of the behaviour for the enemies. It uses a finite-state machine to determine a zombie's behaviour. Zombies can have three states, including 'PASSIVE,' 'CHASE' and 'DEAD.' By default, a zombie is in a passive state. Each zombie has at least two idle positions, where the zombie will continuously walk between these two positions. The zombie will wait at each idle position for 5 seconds before walking to the next position. This is done so that the player can determine a zombie's path and try and sneak around them if they wished. While the zombie is in a passive state, a function called 'PlayerDetected' will be continuously executed. This function contains 3 Boolean variables, 'isInAngle,' 'isInRange' and 'isNotHidden.'

‘isInAngle’ changes to true when the player is within the enemy’s cone of vision. ‘isInRange’ changes to true when the distance between the zombie and the player is within the specified detection range. ‘isNotHidden’ is true when the raycast shot from the direction of the zombie to the player is not obstructed. When all three Boolean variables are true, the player is detected and the zombie will change to the chase state. This function can be seen in Figure 16.

```
bool PlayerDetected()
{
    isInAngle = false;
    isInRange = false;
    isNotHidden = false;

    if (Vector3.Distance(transform.position, player.transform.position) < detectionRange)
    {
        isInRange = true;
    }

    RaycastHit hit;

    if (Physics.Raycast(transform.position, player.transform.position - transform.position, out hit, detectionRange))
    {
        isNotHidden = true;
    }

    Vector3 side1 = player.transform.position - transform.position;
    Vector3 side2 = transform.forward;

    float angle = Vector3.SignedAngle(side1, side2, Vector3.up);

    if (angle < detectionAngle && angle > -1 * detectionAngle)
    {
        isInAngle = true;
    }

    if (isInRange && isNotHidden && isInAngle)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Figure 16. Implementation of the 'PlayerDetected' function in 'Zombie.cs.'

When the zombie is in the chase state, the zombie will continuously chase the player by setting the agent’s destination to the player’s position. If the zombie is within a certain range of the player, it will attempt to attack and damage the player. The attack animation will play, and the player will be damaged by a random number of health points between 10 and 20. If the player reaches a certain distance away from the zombie, the zombie will change back to the passive state and walk back to their idle positions. The zombie has an integer variable called ‘zombieHealth’ which equals 100. After the player has damaged it enough and this value reaches 0, the zombie will switch into the dead state. This then triggers the death animation, which then removes the zombie game object from the environment.

Lastly, the zombie has a number of sound effects that are played as a result of associated behaviours. The ‘Zombie’ script has 5 different arrays for sound clips. When the zombie is at an idle position, detects the player, attacks the player, gets damaged by the player and gets killed by the player, a random sound clip from the respective array will be played. This gives the zombies variety in the sounds they make in addition to the variants of zombies within the game.

4.6. Game World and Item Placement

The game world has the important role of facilitating all the interactions to take place. The world needed to be big enough in scope in order to allow these interactions to take place, while also small enough to ensure that the player does not get uninterested while traversing through the city's environment. In terms of aesthetics, an abandoned city was determined as the best to deliver a consistent visual experience that aligns with the game's theme. When designing the game's environment, it was important to recognise the potential player progression. While the environment is not too complex, it was important to ensure the game had some non-linearity to provide the player with free rein over the path they choose. Therefore, there are multiple paths that the player could progress through, with the roads of the environment signifying these different pathways. All of the game's items needed to convincingly fit inside the environment, so it was important that items were placed accordingly in the right areas. At the beginning of the game, the player is placed in a forest. The forest has a number of starter crafting items, such as sticks, logs and rocks. It is intended that this allows the player to create their first weapon and collect crafting materials before they venture into the main city. As they explore further into the city, they will find more powerful weapons, and acquire crafting materials, such as the metal scrap, which will allow them to upgrade their starter weapons. The game features three small buildings that they can explore, with each building containing a firearm and other important items. The city also features two small park areas, so that the player does not need to venture back to the forest at the beginning of the game's map to acquire certain items. In regard to quest items, these items are distributed evenly throughout the map so that the player needs to explore the city sufficiently to find these items. However, it was important to ensure that these quest items are not too difficult to find, so therefore multiple of these items were placed to ensure that they could be found. Enemies are placed with the intention of scaling the difficulty upward as the player progresses through the city. At the beginning of the environment, there is a light number of zombies that the player should be able to deal with using starter equipment. Deeper into the city, there is a higher density of zombies to deal with, by which the player should have acquired more powerful weaponry.

4.7. Journal and Game Completion

The journal was the final major implementation in the game's development. Despite the previous interactions working as intended, it would be difficult for the player to know what their objective is and to remember crafting recipes or even know that they can craft items together at all. Because of this, the journal was implemented to provide the player with important information regarding their main objective and provide them with a recording of the crafting recipes they have found. This was inspired by *Fallout 4*'s 'Pip-Boy' (Bethesda Game Studios, 2015), where the player's objectives and other information was displayed diegetically through a device on the player-character's wrist. The journal provides a diegetic way of displaying crucial game-related information to the player through a physical item in the game world. The journal can be picked up as a regular item and be interacted with by pressing the left mouse button. The journal's overlay is then displayed, as seen in Figure 17. The

left page shows a diary entry, providing some context about the player's position, and the items that they must find in the environment. The right page displays a list of the recipes that they have crafted. When the player hasn't crafted any items, the text will prompt the player to craft a weapon before entering the city.

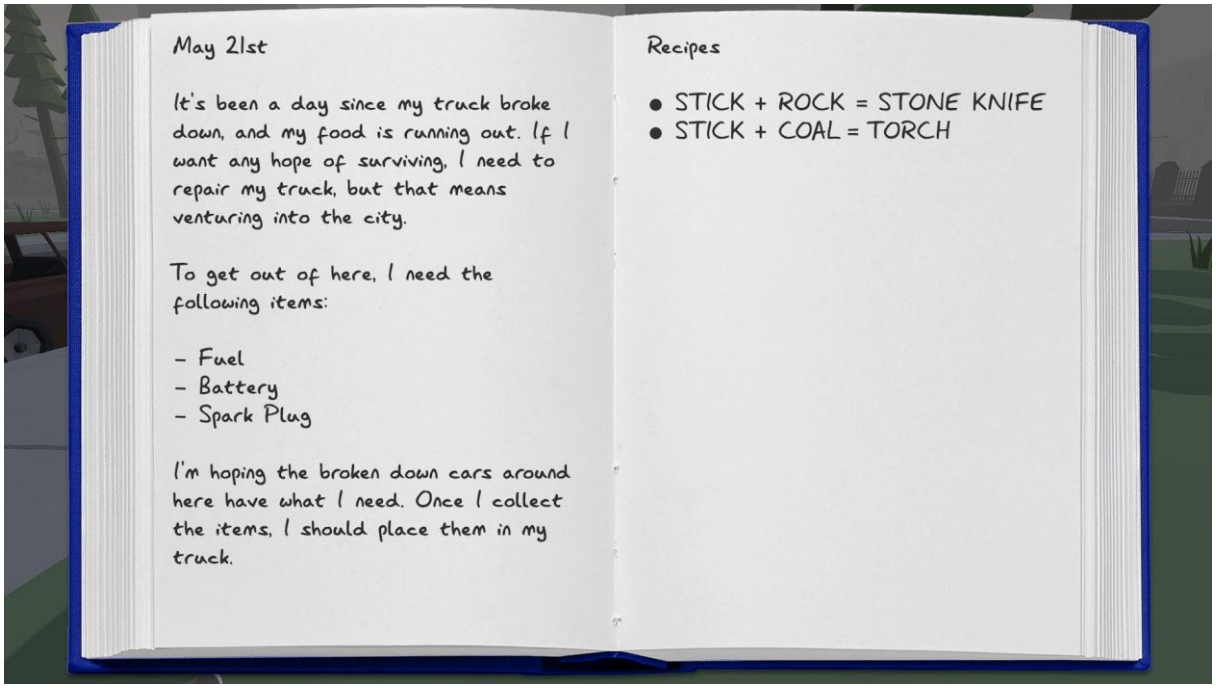

Figure 17. View of the journal's overlay.

To update the journal's overlay, the recipe array from the crafting manager is accessed. Each crafting recipe has a Boolean, called 'isCrafted,' which is switched to true when the player has first crafted an item using the related recipe. Each time the journal is displayed, the array is accessed to check which items have been crafted. This then updates the text of the journal. If none of the recipes have been used, then the default text is displayed. Any time the player holds an item that is a quest item, this information is passed to the 'UpdateJournalOverlay' function, which updates the objective to show a strikethrough on the associated quest item, indicating that this item has been collected. This implementation can be seen in Figure 18.

```
public void DisplayJournalOverlay()
{
    recipeListText.text = "Recipes\n\n";

    foreach (var recipe in craftingManager.recipes)
    {
        if (recipe.isCrafted == true)
        {
            recipeListText.text += "• " + recipe.itemA.itemName.ToUpper() + " + " + recipe.itemB.itemName.ToUpper() + " = " + recipe.result.itemName.ToUpper() + "\n";
        }
    }

    if (recipeListText.text == "Recipes\n\n")
    {
        recipeListText.text += "I haven't crafted any items yet. Maybe I should try craft a weapon before entering the city.";
    }

    journalOverlay.SetActive(!journalOverlay.activeInHierarchy);
}

1 reference
public void UpdateJournalOverlay(Item questItem)
{
    if (questItem.itemName == "fuel" && !fuelConfirmation.enabled)
    {
        AudioManager.instance.PlaySound("New Recipe");
        fuelConfirmation.enabled = true;
    }
    else if (questItem.itemName == "car battery" && !batteryConfirmation.enabled)
    {
        AudioManager.instance.PlaySound("New Recipe");
        batteryConfirmation.enabled = true;
    }
    else if (questItem.itemName == "spark plug" && !sparkPlugConfirmation.enabled)
    {
        AudioManager.instance.PlaySound("New Recipe");
        sparkPlugConfirmation.enabled = true;
    }
}
```

Figure 18. Implementation of the 'DisplayJournalOverlay' and 'UpdateJournalOverlay' functions in 'CanvasManager.cs.'

5. Evaluation

Once the development of the game was completed, a series of user testing was conducted with six participants. The chosen six participants had an adept level of experience playing games as to ensure expertise did not produce a barrier in the playthrough of the game. Participants were asked to read through an information sheet outlining details of the research and then complete a participant consent form. Once this was completed, the participant was tasked with playing through the experience from start to finish. Once they completed the game, they were asked to complete an online questionnaire about their experience playing. The questionnaire consisted of a mixture of quantitative questions, answered with a Likert scale from 1 to 7, and qualitative questions. These questions related to the difficulty of the game, the intuitiveness of the game's mechanics, the understanding of the player-character's condition, and their enjoyment of the game as a whole.

When asked about the game's difficulty on a Likert scale from 1 to 7, a value of 3 was the mode and the median. As a whole, the game was found relatively easy, with mainly enemy interaction highlighted as the biggest challenge, citing particular difficulty when it came to damaging zombies with melee weapons. Some issues were noted with the game's mechanics, such as difficulty picking up smaller items and the lack of hints for crafting. When asked about the difficulty regarding the game's controls, all players found the controls easy to learn, with most participants answering with 2 on a scale of difficulty from 1 to 7. The item system, involving picking up items and using them, was concluded as intuitive apart from one outlier. Most participants rated the system as a 6, with a single outlier rating it a 3. The outline around highlighted items was a feature that participants thought helped a lot. However, smaller items may have been trickier to pick up due to having a smaller

collider and therefore requiring greater precision. One participant found it difficult to differentiate items from regular props in the game world. In regard to the inventory system, specifically involving placing the backpack and storing items, all participants found this interaction intuitive. Four participants rated the system a 7, and two participants rated it a 6. Participants found that the controls for opening and closing the backpack made sense, and that the grid-based system was intuitive and easy to learn. Participants highlighted the effect of making the experience slower and more methodical, and the emphasis on inventory management. When asked about the crafting system, participants found it slightly intuitive, with a modal and median response of 5. A lot of participants talked about difficulty in knowing how to initially craft items at first, but once this hurdle was overcome it was found easier to use this interaction. Some participants wished for improved visual feedback on items that could be crafted, as well as asking for crafting to take place within the inventory. When queried about the journal, participants mostly found it useful, with a majority of responses recording a 6. Many participants found great use in being able to see the recipes they obtained, while also being able to keep track of their objective. One participant wished that the journal provided a few more hints on what items could be crafted. Two participants mentioned how the journal took up inventory space, with one participant discarding it altogether to acquire more room. When asked about the player condition, responses were mixed, with a median of 4.5 and a mode of 3 and 5. Some participants found the health overlay too subtle and requested the need for some numerical feedback about the player's health. Some participants also found difficulty in knowing how much ammo was left despite the audio feedback. When asked about the enjoyment of the experience, participants were satisfied overall, with a mode and median of 6. Participants found themselves engaged with the experience as a whole and were engaged with the game's mechanics. Some participants noted the reward in finding new crafting recipes and others found killing zombies satisfying. When asked about any suggested improvements, players wished for a clearer crafting system, the ability to distract enemies, improved player mobility, and greater visual feedback for the player's status.

Overall, the user testing outlined the general success of the project, while highlighting some important factors to note. The responses about the game's difficulty were expected; a moderate difficulty was intended during development. The result regarding the control scheme was slightly unexpected, as previous studies concluded that diegetic interfaces came with the trade-off of having difficult controls (Cairns, 2015; Caroux and Isbister, 2016), but this may have come down to a consistent and conventional control scheme which the experienced players who participated in the study did not have difficulty with. The game's item system was generally received as intuitive, which could be additionally accredited to the controls and the use of outlines. Reception to the inventory system was also positive and outlined a successful implementation of traditional inventory systems in a diegetic format. The crafting system had room for improvement. More could have been done to allow players to discover new recipes easier, such as through the form of visual feedback, or through a use of hints written in the journal. That being said, the journal successfully counteracted some of the issues that were discovered in the design of the crafting system and provided players with necessary context and information

relating to the game's objectives and recipes. The visibility of the player's condition would be improved with a stronger indication of a player's low health, in addition to a numerical health indicator, albeit a little more difficult to display diegetically. Despite some small shortcomings, participants still found great enjoyment with the game's mechanics, indicating that the diegetic systems did not detract from the enjoyment of the experience.

6. Reflection

A diegetic experience called 'Dead End' was created using the Unity game engine. This game successfully implements a diegetic interface in a manner which does not detract from the player experience and provides a foundation for how traditional game design choices could be adapted to provide player's with more immersive experiences. Academic literature outlined the impact of the HUD and diegetic interfaces on the player experience but primarily looked into games that were already dependent on the HUD's contextual interface, or specific ways of communicating traditional HUD-based information diegetically. 'Dead End' expands on existing research by exploring how usual game mechanics, like inventory and crafting systems, can be adapted for use in a diegetic system. The design and implementation of these mechanics was an overall accomplishment, with user testing mostly achieving the desired outcomes. Item systems and inventory systems were designed well as to facilitate simple and understandable interactions, and while the crafting system had room for improvement, the desired functionality was still accomplished. Information regarding the player condition took inspiration from previous interactive media in regard to the injured overlay, and although small alterations could be made, it does communicate the player's health. Sound effects also communicate interactions effectively in order to improve the game's feedback systems. The game's mechanics all integrated together, in addition to the design of the environment and the implementation of enemy behaviour, to deliver a cohesive and consistent experience that was enjoyable for players. The game could be expanded further with the integration of more items and crafting recipes, while also refining the current systems further to control any of the limitations of a diegetic interface.

7. Bibliography

Aggro Crab. (2025). *Peak*. [Game]. Available at: https://peakpeakpeak.com/. [Accessed 11 May 2026].

Bethesda Game Studios. (2015). *Fallout 4*. [Game]. Available at: https://fallout.bethesda.net/en/games/fallout-4. [Accessed 7 May 2026].

Cairns, P. et al. (2015). Removing the HUD: The Impact of Non-Diegetic Game Elements and Expertise on Player Involvement. *CHI PLAY '15: Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, pp. 13-22. Available at: https://doi.org/10.1145/2793107.2793120.

Capcom. (2017). *Resident Evil 7: Biohazard.* [Game]. Available at: https://www.residentevil.com/7/us/. [Accessed 11 May 2026].

Caroux, L. and Isbister, K. (2016). Influence of heads-up displays' characteristics on user experience in video games. *International Journal of Human-Computer Studies*, 87, pp. 65-79. Available at: https://doi.org/10.1016/j.ijhcs.2015.11.001.

Caroux, L. et al. (2022). Heads-up displays in action video games: the effects of physical and semantic characteristics on player performance and experience. *Behaviour & Information Technology*, 42(10), pp. 1466-1486. Available at: https://doi.org/10.1080/0144929X.2022.2081609.

Gunnar001. (2016). German Trench Assault (No Hud Immersion) – Battlefield 1. [Video]. Available at: https://www.youtube.com/watch?v=hCYjzwSsEh8 [Accessed 6 May 2026].

Köhle, K. et al. (2021). Diegetic and Non-diegetic Health Interfaces in VR Shooter Games. *Human-Computer Interaction – INTERACT 2021*, pp. 3-11. Available at: https://doi.org/10.1007/978-3-030-85613-7_1.

Lewis, E. (2025). *Games That Are Best Played With No HUD.* [Online]. GameRant. Last updated: 29 September 2025. Available at: https://gamerant.com/games-best-played-better-with-no-hud/ [Accessed 7 May 2026].

Motive. (2023). *Dead Space.* [Game]. Available at: https://www.ea.com/en-gb/games/dead-space. [Accessed 7 May 2026].

Peacocke, M. et al. (2015). Evaluating the effectiveness of HUDs and diegetic ammo displays in first-person shooter games. *2015 IEEE Games Entertainment Media Conference (GEM)*, pp. 1-8. Available at: https://doi.org/10.1109/GEM.2015.7377211.

Pfister, L. and Ghellal, S. (2018). Exploring the influence of non-diegetic and diegetic elements on the immersion of 2D games. *OzCHI '18: Proceedings of the 30th Australian Conference on Computer-Human Interaction*, pp. 490-494. Available at: https://doi.org/10.1145/3292147.3292190.

Rosyid, H.A., Pangestu, A.Y. and Akbar, M.I. (2021). Can Diegetic User Interface Improves Immersion in Role-Playing Games? *2021 7th International Conference on Eletrical, Electronics and Information Engineering (ICEEIE)*, pp. 200-204. Available at: doi:10.1109/ICEEIE52663.2021.9616732.

The Game Bakers. (2026). *Cairn*. [Game]. Available at: https://store.steampowered.com/app/1588550/Cairn/. [Accessed 11 May 2026].

4A Games. (2019). *Metro Exodus*. [Game]. Available at: https://www.deepsilver.com/gb/games/metro-exodus. [Accessed 7 May 2026].