Learning Unreal Engine 5 Through Designing and Implementing a 3D Puzzle Platformer Mechanic and Level

Introduction

As an upcoming game designer, my current skills mainly revolve around the Unity game engine to prototype and script game features. Most of the projects I've worked on so far have been using Unity and have developed my skills in C# programming and the tool in general. A lot of companies worldwide use Unity to develop their games, which makes my skills in Unity practical.
A lot of other companies, however, use other engines for their game development, such as Unreal Engine 5 (UE5). Unreal Engine 5 is a popular game engine that provides a lot of built-in tooling and allows developers to create very realistic-looking games. One of the many differences with Unity is the blueprints system. Programming behaviour in UE5 is different because it doesn't require the user to write code like most other game engines. Instead, it makes use of node-based programming.

Having more than no experience in UE5 could open up more opportunities in the future. In this project, I will design and prototype a 3D platformer game mechanic and level to learn the basics of Unreal Engine 5. This will also be done using the Double Diamond design thinking framework to guide my process (Design Council, 2025).

Discover Phase

Existing 3D Puzzle Platformers and Their Mechanics

To create an effective 3D puzzle platformer, the core mechanics must be carefully designed. The best mechanics are versatile, meaning they can be applied in numerous ways for both platforming and puzzle-solving, often leading to unexpected and creative solutions. Think of the Portal Gun in *Portal 2*, which isn't just for reaching high ledges but also for redirecting energy, solving complex physics puzzles, and bypassing obstacles. Similarly, the "Bash" ability in *Ori* serves as a crucial tool for movement, combat, and environmental puzzle-solving by allowing players to redirect projectiles and themselves. This versatility deepens the gameplay, encourages player creativity, and minimises the need for an overwhelming array of single-use tools.

Even when dealing with abstract concepts like time or perspective, a strong mechanic maintains physical and spatial coherence. It feels grounded within the game's established physics and spatial rules, ensuring that actions have predictable and understandable outcomes within the game world. *Braid's* Time Rewind, while magical, consistently affects objects, allowing players to understand what will and won't rewind.

In *Fez*, the Perspective Shifting mechanic genuinely rotates the environment, making previously aligned elements physically connect. This coherence reduces frustration, allows players to form accurate mental models of the game world, and makes puzzle solutions feel logical rather than arbitrary.

A good mechanic also encourages player agency and experimentation. The design should invite players to try different approaches and even "fail forward," where a failed attempt still provides valuable information or a clearer understanding of the puzzle. *Super Mario 64*'s 360-degree analogue movement in its open levels perfectly illustrates this, as players are encouraged to experiment with various jumps and routes to reach objectives.

Finally, clear visual and auditory feedback is paramount. Players need immediate confirmation when they use a mechanic, whether it's a platform moving, a light activating, or a distinct sound effect. Think of the satisfying "pop" of a portal opening in *Portal 2* or the visual snap of *Fez*'s world rotating. This feedback confirms player input, helps them identify cause and effect, and subtly guides them toward correct solutions. When all these elements align, a 3D puzzle platformer mechanic becomes elegant, versatile, clearly communicated, and intrinsically linked to both the puzzle-solving and traversal aspects of the game. It empowers the player to creatively interact with the 3D space, leading to satisfying solutions and a deep sense of accomplishment.

Unreal Engine 5 blueprints tutorials/guides

To effectively develop proficiency in Unreal Engine 5, especially in visual scripting through Blueprints and level blocking techniques, the following free learning resources have been identified. These include official documentation, community tutorials, and curated video content designed to support both beginners and intermediate users.

Epic Games Official Resources

- Blueprints Visual Scripting in Unreal Engine 5
    - This official documentation introduces the core concepts of Blueprints, Unreal’s powerful visual scripting system.
    - https://docs.unrealengine.com/5.0/en-US/blueprint-visual-scripting-in-unreal-engine
- Unreal Engine Online Learning Portal
    - Offers dozens of structured free courses on topics including Blueprints, level design, lighting, and more.
    - https://www.unrealengine.com/en-US/onlinelearning-courses

- UE5 - Level Design Fundamentals
    - A page dedicated to explaining the fundamentals of level design with examples.
    - https://dev.epicgames.com/community/learning/tutorials/3VKJ/unreal-engine-fortnite-level-design-fundamentals

YouTube-Based Community Tutorials

- Smart Poly – Unreal Engine 5 Beginner Tutorials:
    - A popular beginner-friendly YouTube playlist that teaches Blueprints.
    - Unreal Engine 5 | Blueprint For Beginners (2023)
- WorldofLevelDesign - 3 Methods for Blocking Out Environments and Level Designs in UE5
    - This video provides a real-world example of blocking out environments for level design.
    - UE5: 3 Methods for Blocking Out Environments and Level Designs in UE5

Define Phase

Using the research done in the previous section, design criteria will be distilled. These criteria will serve as a guideline to evaluate the ideas generated in the ideation phase.
Considering this project's primary purpose is for me to practice using Unreal Engine 5 (UE5), the main criterion that matters is feasibility. However, with the NAF criteria in mind, the novelty and appeal criteria will also be considered (*The NAF Technique – TICON*, n.d.).
To keep track of the relative importance of these design criteria, a weight percentage will be attributed to each of them to determine the impact each criterion will have on the final scoring of the ideas.

Design Criteria

- Feasibility: Considering the current level of experience with UE5 blueprints, the game mechanic needs to be simple to prototype within the time frame.
- Novelty: The uniqueness of the idea compared to other existing mechanics established in the Discover phase.
- Attractiveness: This considers how well the idea solves the problem. This would encompass the aspects of a mechanic that were mentioned in the Discover section:
    - Versatility of applications
    - Player experimentation
    - Intuitive controls

Criterion	Weight
Feasibility	0.8
Novelty	0.1

Attractiveness	0.1

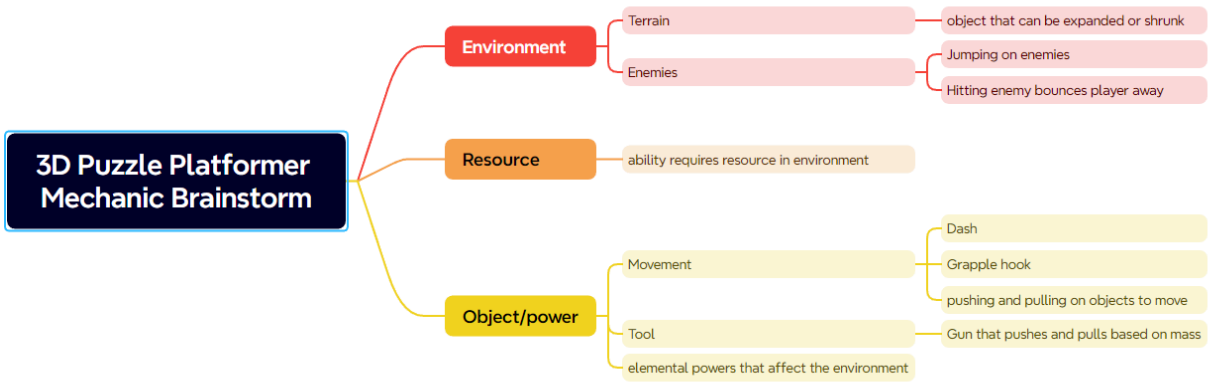
The feasibility of the feature is the most important criterion by far because the project's aim is to learn UE5 basics by implementing a mechanic. With the level of experience in UE5 considered, the mechanic must be highly skilled to compensate.
The other criteria matter in the sense that they may break a tie should multiple ideas have similar feasibility.

Game Mechanic - Develop Phase

Since the level design will depend on the chosen mechanic, the game mechanic will first be ideated and prototyped before starting the process for the level.

Ideating Game Mechanics

A mind-mapping method was used to brainstorm a few ideas:


3D Puzzle Platformer Mechanic Brainstorm
Environment
Terrain
object that can be expanded or shrunk
Enemies
Jumping on enemies
Hitting enemy bounces player away
Resource
ability requires resource in environment
Object/power
Movement
Dash
Grapple hook
pushing and pulling on objects to move
Tool
Gun that pushes and pulls based on mass
elemental powers that affect the environment

Out of this brainstorming mindmap, a few concrete ideas emerged. The best 3 were chosen to be elaborated:

1. Shrinkable/expandable object which requires expanding/shrinking another.
    - A distinct type of object that the player could shrink or expand from a distance, but requires doing the opposite to another object. These special objects would physically interact with the environment. Puzzles would revolve around shrinking and expanding the right object in the right order to solve.
2. A dash ability that needs to be recharged through a pickup.
    - A dash ability that would propel the player in a direction, but with a limited number of uses. More uses could be gained by collecting specific pickups. The dash would be used in traversing the level by carefully considering the remaining uses and ways to get more dashes.

3. Hitting objects/enemies causes the player to bounce away.
    - Certain objects or enemies would push the player away if they were hit by the player. The idea here is that carefully manipulating the player position as well as the object/enemy position could make for interesting puzzles.

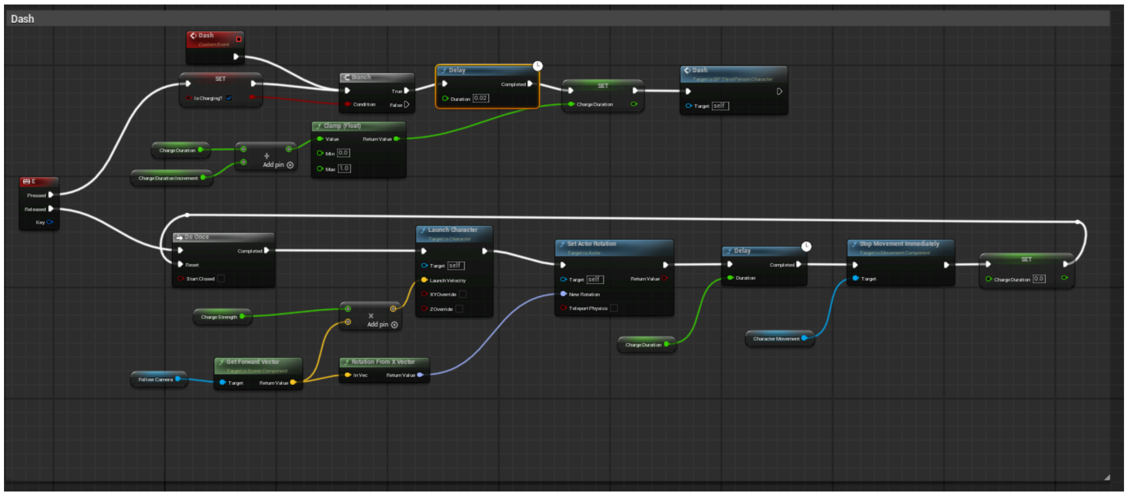
These ideas were evaluated based on the NAF criteria established in the Define Phase. The ratings and weighted scores can be observed in the following table:

		Options					
		Shrinkable/expandable objects		Dash with dash charge pickup		Object that bounce	
Criteria	Weight	Rating	Score	Rating	Score	Rating	Score
Feasibility	0.8	5	4	9	7.2	7	5.6
Novelty	0.1	6	0.6	2	0.2	7	0.7
Attractivene ss	0.1	7	0.7	4	0.4	6	0.6
Total	1		5.3		7.8		6.9

The dash ability idea seemed the simplest and straightforward to experiment with and implement. Compared to the other two ideas, it seemed to depend less on more complicated systems such as selecting objects for the shrinkable/expandable objects idea. Idea 3 was also promising it relied a bit too much on a sturdy physics system to implement as someone of my UE5 experience.

Iteration 1

The first step was to create a project in UE5 using the 3rd-person template made by Unreal Engine, which provides a movable character with a jump and a basic test level.
Using the “How to Make a Simple Dash Ability in Unreal Engine 5” video by Gorka Games (Gorka Games, 2022) as a guide for implementing the basic dash functionality, I adapted it to be quicker and more responsive, as well as implemented a charging functionality, which I was able to figure out myself with a few searches online.
The charging mechanic was a fun additional mechanic I came up with while implementing the dash. I wanted to try my hand at implementing something without a tutorial, and it turned out to be a fun addition to the system, so I kept it in. It works by holding down the dash button, which increases the distance of the charge the longer it is held down(u to a maximum).
This video gave me a good starting point for figuring out the very basics and trying to implement something that worked.

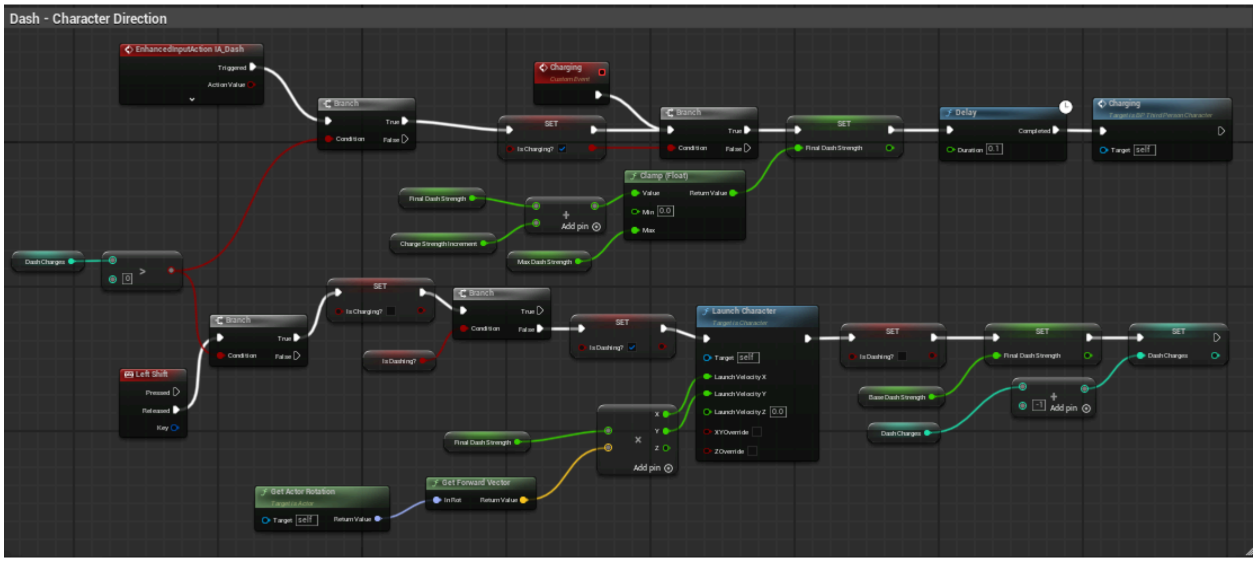


However, the dash didn't work exactly like I was expecting. Increasing the launch force too much made the character uncontrollably slide, was overall hard to control and frustrating to use. My first solution to this problem involved stopping all movement of the character after a delay using a node. While this helped make the dash more responsive, the changes made to the system for this solution ended up making the charging mechanic inconsistent. I tried various things to fix it, but ultimately couldn't find a solution.
Lastly, this version of the dash worked by using the camera direction to aim the dash, which also meant you could dash vertically. However, it feels more like a first-person perspective rather than a third-person perspective. Looking into existing game showed that most abilities like dashes go in the direction the character is facing, not the camera.

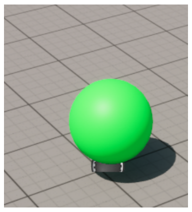
Iteration 2

Through the help of a different tutorial for implementing a dash called "HOW TO DASH | Unreal Engine 5 Tutorial" by The Average Dev (The Average Dev, 2024), this iteration fixes the sliding issue using different character controller settings (separate braking friction is off, and falling lateral friction is set to 5), which turns out to be much better than the previous iteration in terms of consistency of the dash

The dash direction is also set to the character's forward vector this time which makes the controls feel more in line with third person games.



The Dash Charge resource was also added along with the dash charge pickups, which means the player can only dash when they have more than 0 dash charges.
Dash Charges can be gained by picking up Charge Orbs, which grant 1 charge on pickup.

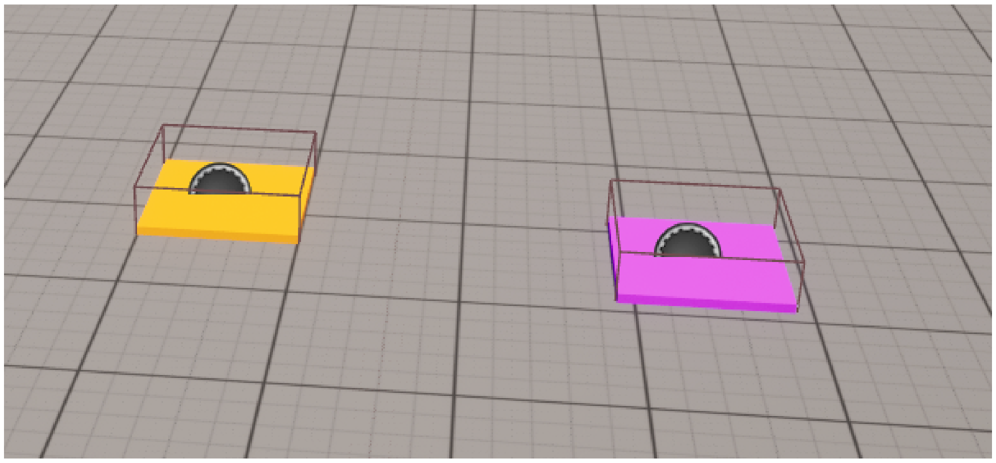


Iteration 3

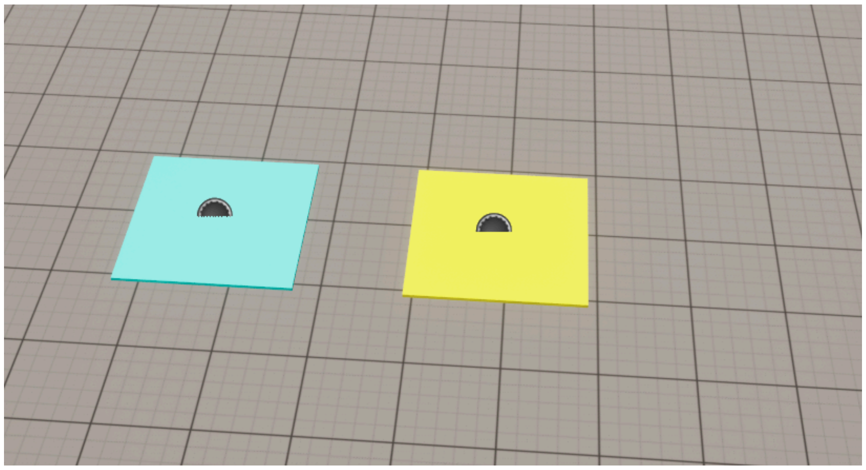
Having only the dash as a mechanic turned out not to be enough to create an interesting level. Various simple mechanics were therefore implemented.

Created two bounce pad variations to add more interactivity to levels.

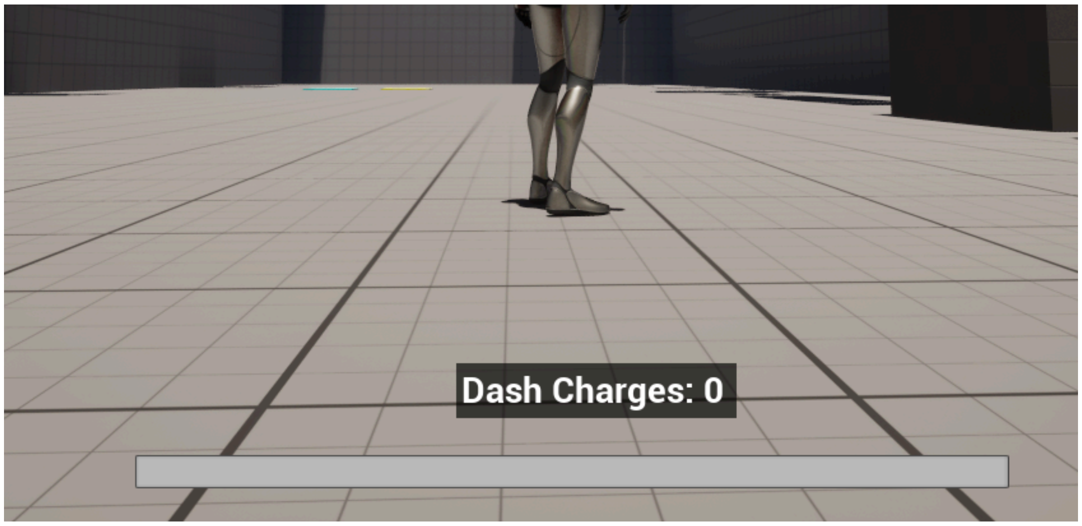
- The static bounce pad (orange in the image) gives a fixed vertical thrust when the player collides with it. The player also retains their horizontal momentum when hitting the bounce pad.
- The dynamic bounce pad (pink in the image) transfers the velocity of the player into a vertical thrust, which means the player will be launched upwards depending on how fast they hit the bounce pad. In this case, all the player’s horizontal momentum is negated.



Added horizontal and vertical moving platforms of different colours so the player can easily tell them apart.



Added checkpoints which allow the player to teleport back to the last checkpoint they touched by pressing R. This felt essential for making a platformer level.
Added UI elements in the form of a Dash Charge counter and charge bar for the charged dash.

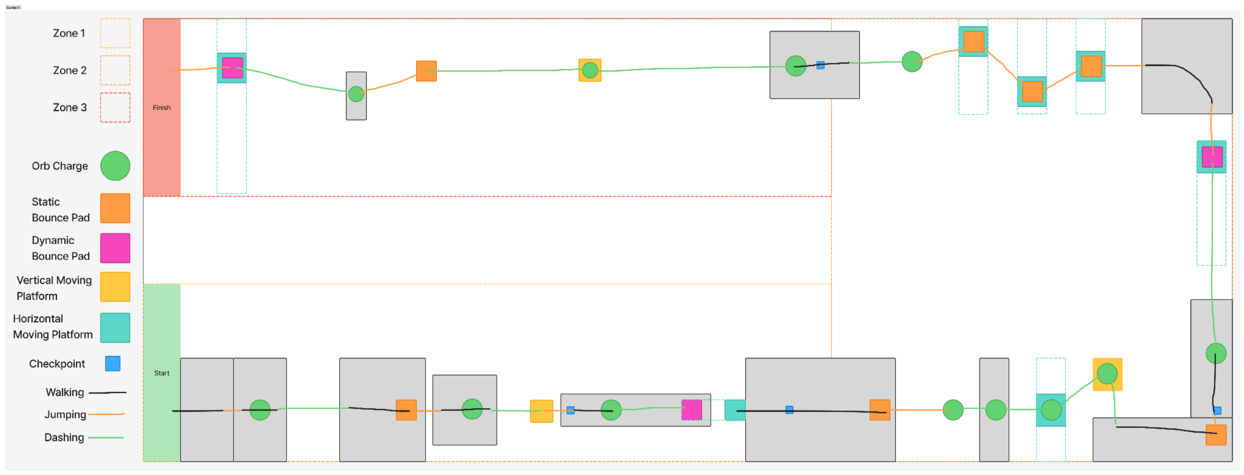


Level Design - Develop Phase

The process for developing the level had to be shortened due to the game mechanics taking longer than expected to develop. This means that the ideation phase was skipped, and the prototyping of the level was started straight away without considering possible ideas.

Iteration 1

Iteration 1 was done in Figma to get a general sense of the level flow and how some of the challenges would fit within the space. The following diagram is a top down perspective of this level.



The level is split into 3 zones, going from easiest to hardest. Zone 1 is meant to introduce the player to the mechanics one by one. The challenges are also fairly easy to match the player's expected lower skill level in using the tools at hand.
Zone 2 ups the challenges by combining the mechanics. The challenges should stay relatively easy as long as the player figures out the solution.
In Zone 3, the player is challenged harder by making the solutions about their timing and distance gauging.

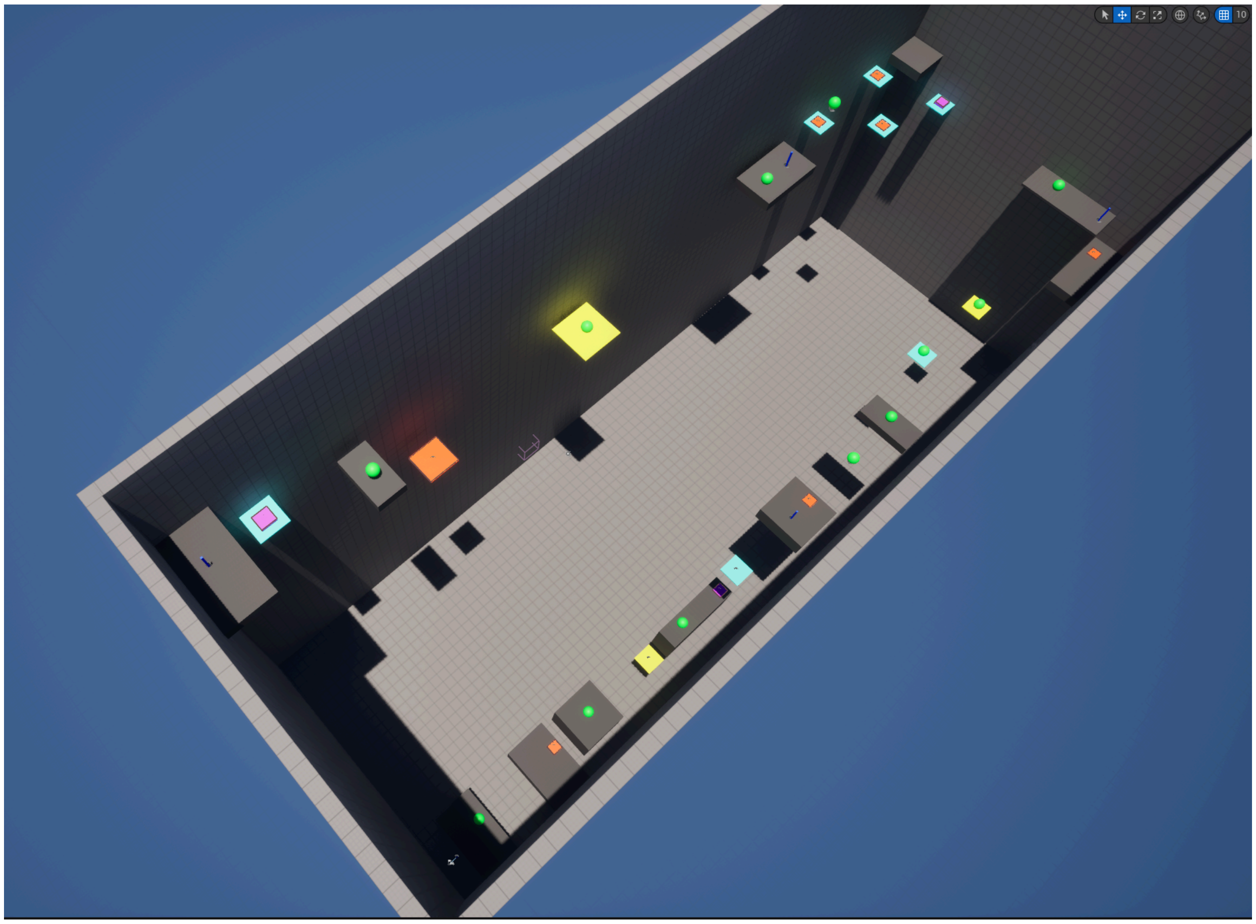
While it may be hard to tell, the level would be going upwards continuously as the player progresses all the way to the end. This was done for one main reason: to prevent the player from charge-dashing through challenges(since it only works horizontally). This is also supported by the many mechanics to move vertically, like the two bounce pads and the vertical moving platform.
While there is no fall damage, moving higher and higher also gives a sense of intensity to the playe,r which matches the difficulty increase.

To determine the locations of the checkpoints, the expected skill level and the relative difficulty of the challenges were considered. While the first challenges are relatively easy for someone

who has completed the level before, a new player may be more likely to make repeated mistakes, which can lead to frustration when having to redo a section constantly.
As the player progresses through the challenges, checkpoints become less available between harder challenges, which forces the player to redo challenges they completed before.
Overall, the challenges between checkpoints are considered hard enough to complete in one go for the expected skill level of the player, even if they have to redo it multiple times. The goal is to have the player avoid redoing sections that have become too easy, as this can lead to boredom and frustration.

Iteration 2

In this iteration, the prototype made in Figma is recreated in Unreal Engine using the modelling tool to create the shapes. This is one of the level blockout techniques suggested in the WorldofLevelDesign video from the Discover Phase.



Of course, some adjustments had to be made since implementing and testing the level in 3D revealed some issues with the initial Figma layout. Overall, this implementation remained faithful to the Figma layout.

The process for the implementation involved creating the challenges one by one, starting from the beginning of the level, and running the level to make sure it was playable as intended. Some

efforts were made to make sure the player couldn’t “cheat” their way unexpectedly through a challenge, but some may still be there. This is in large part due to the way the charge orbs currently work, since they can provide an infinite supply of charges with a max of 3. Ideally, the orbs would have disappeared on pickup and reappeared when respawning, but due to a lack of skill in blueprints, this was not able to be implemented.
This means the player may be able to spam-dash their way through challenges in some areas.

Deliver Phase

The final product is a build of the Unreal Engine 5 project, which consists of the level shown above that was designed around the mechanics developed earlier in the project.
Below you will find links to a playable build and a video showcase of the level:

Final video showcase: https://youtu.be/Yv7BIVOxl0c

Mechanics implementation showcase: https://youtu.be/kozkY0LMJzw

Playable build: https://adecazenove.itch.io/ue5-learning-project

References

Design Council. (2025). *The Double Diamond*. Www.designcouncil.org.uk; Design Council. https://www.designcouncil.org.uk/our-resources/the-double-diamond/

Gorka Games. (2022, September 8). *How to Make a Simple Dash Ability in Unreal Engine 5*. YouTube. https://www.youtube.com/watch?v=Ok-Sb_NxZlw

The Average Dev. (2024, April 6). *HOW TO DASH | Unreal Engine 5 Tutorial*. YouTube. https://www.youtube.com/watch?v=TkSiasE0p_I

The NAF Technique – TICON. (n.d.). https://www.creativityteaching.eu/the-naf-technique/