Graduation Semester

**Graduation Project Report**

# *De-stereotyping Game Development Exercises*

2024

**Ties Anton Hendrik Poutsma**
Bachelor: Creative Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Graduation - Module 11 & Module 12

## *Abstract*

This project attempts to answer if a ‘neutral’ mini-game with programming features can attract potential students to a software development course, when they might be interested, but are swayed away due to stereotypes. Background research is performed on these stereotypes and the influence and influenceability they cause. The research shows that potential students are influenceable, and so a mini-game was ideated to make use of that opportunity. The ideated product is a three dimensional puzzle game made in the Unity game engine, with neutral styling, and a soft colour palette. The game features a game-within-a-game concept in order to create a gameplay layer that introduces players to programming. The gameplay layer is executed by granting players ‘developer access’ - players can directly access and control important gameplay variables. This access to these game variables is used to solve increasingly more difficult programming related puzzle challenges. Twenty playtests were performed to test the game’s overall experience, accessibility, and its impact as a tool to introduce players to programming as a concept. The playtests were overall experienced very positively and all playtesters were able to fully complete all puzzle elements of the game. When asked if they would have been more interested in programming as a concept, had they had this product as their first introduction to programming, the answer was a largely *yes*. The results gathered from playtesting indicate that a de-stereotyped minigame with developer access features can help attract prospective students to programming and by extension, to a software development course. The product is ready to be used on open days as an introduction to programming. Some future work should go into re-visiting the intuitiveness of existing gameplay features, solving some accessibility issues found in playtesting. Other than that, the main areas for future work stemming from this product are found in the form of product expansion. New levels, expanded developer access features and more challenging puzzles can greatly increase the product's effectiveness.

# *Contents*

# One *Introduction*

## 1.1) Stakeholders

**Title**: Thesis: "De-Stereotyping Game Dev Exercises"
**Student:** Ties Poutsma
**Supervisor:** Marcello Gomez Maureira
**Client:** Isabelle Kniestedt (from Grafisch Lyceum Utrecht)
**Critical Observer:** Angelika Mader

## 1.2) Situation and Challenge

This graduation project is a combination of research and software development work. The Grafisch Lyceum Utrecht offers a course on programming and game development. The staff of the Grafisch Lyceum in Utrecht has expressed a desire for a more demographically varied set of applicants. Finding diversity and a broader demographic of software developers is an important topic which has been worked on across the globe for a long time, and this project will add onto that. The GLU staff's idea was to find a bachelor student in their graduation phase, and have them work on a product that could help them attract different types of students.
They have created the task of creating a de-stereotyped game-design 'exercise' that can be displayed on open days of middle schools, where the teenagers there can be introduced to programming and game development in a stereotype-breaking way.
There are two parts to this task that provide challenge.

1. The de-stereotyping: The task is to create this exercise in a 'neutral' way, rid of elements that are typically attractive to stereotypical 'gamers' and potential programmers. Research will be done on these stereotypes, to figure out what the difference is between the stereotypes and reality and why there is room for changes. The findings will be adapted upon typical game-development exercises, and an appropriate 'exercise' that will be ideated and executed, with the goal for it to be put on display at an open day of the Lyceum. The eventual form factor of this 'exercise' that ended up being ideated is that of a minigame which combines basic puzzle game functionality as well as 'back-end' variable access, to simulate programming to a limited degree.
2. The intuitive and 'logical' integration of what would eventually become 'developer' functionality into the minigame. To give users the sensation of programming in real time, the minigame brings back-end variables directly into front-end game view. Those variables can be adjusted, and will directly correspond to the in-game behaviour of whatever objects' variable(s) have been changed. This is what sets the development phase of the project apart from a typical game development project. To give users the feeling that they are directly 'accessing' back-end code, and that they are really influencing the game that they are playing, one needs a fluid and intuitive

transition between the game itself and the representation of the 'developer' functionality added to the game.

An important aspect of this project is the effect it will have for the individual prospective students. The time to make choices that impact one's future when one is only a teenager can be confusing and daunting, and being swayed away from a software development course due to stereotypes or other influences in their life when the topic could be something that they might be really interested in or adept at would be a shame.

## 1.3) Research Objectives

To aid in creating a more realistic view on the reality of the diversity within these sets of people, the background research of this report will initially explore the types of stereotypes found in gamers and check those for accuracy when compared to the true distribution of gamers among us. This will be done by reviewing qualitative and quantitative data of multiple papers, where the current demographics of gamers will mainly be identified through quantitative data, and the current and past perceptions on gamer stereotypes will mainly be identified through qualitative data. Subsequently, the perception young adults have on these stereotypes will be laid out by examining qualitative tests, and then the effects these stereotypes have on those young adults - particularly in their academic choices - will be highlighted, also through the review of qualitative results.
The goal of the background research is to set the record straight and hopefully assist in reshaping the typical view on 'gamers' and showing the world that the stereotypes we attach to a 'gamer' do not have to be the truth. Additionally, the background research attempts to lay bare whether or not these stereotypes and peoples' perception of them is influenceable at all and how so. The hope is that this can lead to a more diverse distribution in the application of new students for ICT-related courses and an overall more accurate consensus on what a 'gamer' is and looks like.

The main question the background research will attempt to answer is: how does the influence that 'gamer stereotypes' have on young-adults affect their choices around academics?
A final, smaller part of the background research is done for answering the question: what kind of styling and theming should be used for the minigame.

## 1.4) Research Question

The overall research question that has stemmed from the challenge of this project is:

*Can a 'neutral' mini-game with programming features attract potential students to a software development course, when they might be interested, but are swayed away due to stereotypes?*

# Two Background Research

## 2.1) What stereotypes do we encounter for gamers?

From the ‘80s up until just after the start of the new millennium, when console and computer games were a new and upcoming phenomenon, the typical stereotype associated with these new ‘gamers’ was mainly that of a “physically pale and skinny, socially isolationist, introverted and awkward young man” [6]. “Picture a teen with his eyes glued to a flashy, big monitor - a basement dwelling figure who mindlessly mashes the buttons on his Atari controller”. This baseline stereotype of this ‘gamer’ has not changed too much over the years - especially in regards to their perceived social skills [1]. They are still seen as introverted and awkward, and the basement they stereotypically perform their gaming habits in has also remained unchanged, only the length of their gaming sessions has increased [1], [2].
The main change is in their stereotypical appearance, which has grown from an underweight young adult to be more overweight or obese, and they are much ‘lazier’ [7]. Socially they can sometimes be seen as more violent or offensive [6]. The gamers’ are often still related to antisocial loner archetypes, they seem more likely to continue living with their parents well into their adulthood and remain unmarried [1]. It certainly seems like the stereotypical ‘gamer’ is seen as someone who rejects the typical rules and values of our societies, they seem not to contribute to them, and they can be actively hostile to its other members.

The main big new thing is the diversification and addition of *more* stereotypical types of ‘gamers’. For example, the previously mentioned typical stereotypes are mainly of men. While women can also fit in these stereotypes, for them, a whole new stereotype also emerged: that of the very aptly named ‘Gamer Girl’ [8]. Gamers Girls are female gamers who - stereotypically - have a different motivation for gaming, are stereotypically ‘worse’ than men, have different interests both within game choices and IN-game preferences, are attention seeking, overly effeminate, and have highly sexualized interactions with their male counterparts, be it in-game or on a different related platform [4], [5], [7], [8]. Overall, the image of a typical gamer, whatever their gender, has mainly negative connotations around it, both in their lifestyle and in their perceived place in society.

## 2.2) How accurate are these stereotypes?

In reality, we see that gaming has become a much more mainstream activity when compared to the past. Just like watching television before it, it has become more widespread and common. The truth is, gamers are everywhere.
One of the first inaccuracies we see between gamer stereotypes and the reality is that gaming as a habit is in reality less clustered into specific demographic groups, with similar numbers of participants within the main participating age groups, ranging from younger individuals to middle aged ones. We also see that within the real demographics of gamers,

there is less marginalisation in income, employment status, ethnic group and gender diversity [1], [2, [3], a far cry from what the stereotypical sentiment of gamers suggests.
Besides the much more normalised demographics found in the reality of who gamers are, we also see a much more normalised *way* that gamers choose, experience and play games, compared to the attached stereotypes. For example, studies found that while playing their game, an individual gamer does on average not 'play' differently from another gamer. Inside the game, their interests are the same, they are triggered by the same mechanism and they are likely to have the same sense of motivation to solve the challenges that the game throws their way. There is in other words little difference between gaming 'styles' among different gamers in different demographics.

Despite all these softenings in the difference between 'perceived gamers' and 'real gamers', there are some groups in the gaming community that do still experience great disparities between how they view themselves, and how they are treated. We see these big differences in gamers' experiences when playing the game, versus when interacting with others in and outside of the game. The place where we see this difference is found in gaming habits across demographics, and they happen due to influence from outside factors.
For example, for online (communication based) games, a gamer may be treated as a (relative) equal by his or her teammates and/or opponents, yet see their attitudes towards them completely shift once their real identity is revealed, through voice chat or attached media platforms (think of a 'twitch stream') [4]. Similarly, outside of the actual games entirely, we see that being identified as a gamer can have effects on how a person is viewed [5].
Another one of the larger differences between gamers and their experiences is found between gender experiences. The majority of female gamers do not conform to the stereotypical 'gamer girl', yet the above mentioned different treatment gamers experience once parts of their IRL identity are revealed, is mainly applicable to women [4], [5], [8]. And this is despite the fact that women (like most gamers that do not demographically conform to the typical stereotype) do not *play* games differently from other gamers.
It seems that people tend to treat their anonymous online teammates or opponents in a very 'neutral' way, with not too many stereotypes attached, as long as they are not given reason to believe that those teammates are different. Yet then, once they find out that, say, their teammate turns out to be a woman in real life, they may treat them completely differently in game, be it by openly insulting them, removing them from online lobbies, quietly not working together with them during the game, or perhaps even the complete opposite: the female gamer is worshipped and overtly aided, even if it is detrimental to the teams' in-game success. Gamer stereotypes do still have effects on all players of games, despite the demographics not supporting said stereotypes.

## **2.3)** How do young adults 'see' gamers

The influence that shapes how young adults view gamers and gamer stereotypes is an interesting dance between young adults' perception and their likelihood to change their views over time.
One study finds that gamer stereotypes that are perceived by both teenage boys and teenage girls are gendered, but interestingly also still influenceable by exposure to gamers from certain genders [9]. Concretely, the study finds that exposure to a female gamer on teenage boys - who might initially be prejudiced against female gamers - did have their negative association of typical stereotypical female gamers be reduced significantly. The

study conducted 2 experiments, where in the first one a group of around 200 students were asked to give each item in a large given list of human characteristics a ‘rating’ of how much they related to each participant's idea of a ‘gamer’.
In the second experiment, a similar number of students were shown identical gameplay footage of a popular video game, with an overlay of either a male or a female gamer. They were then asked to attach a set of human characteristic attributes, similar to those of the first test, to whichever ‘gamer’ they saw on in their video. The two tests were compared, and in the results they found that ‘neutral’ gamer stereotypes are more compatible with typically male gender stereotypes rather than with typically female gender stereotypes. “Gamer stereotypes were generally consistent across the two samples, indicating that there is widespread agreement about what gamers are like.”
It seems that the teenagers did have a clear view of what a ‘gamer’ was supposed to be. “The evidence regarding the beneficial effects of exposure to female gamers was mixed. The incompatibility of female stereotypes and gamer stereotypes disappeared after exposure to a female gamer..” [8] Despite the seeming confidence of the teenagers’ assessment of ‘what a gamer is’ in the first test, they do appear to quite quickly change their views once they are exposed to a gamer that does not demographically conform to their previous perception.

A different study also tested what teenagers think gamers are. This study focused more on the different *types* of gamers, not on the overall characteristics of *a* gamer.
In this study [10], the students of Spanish high schools identify mainly two types of gamers.
The first type of gamer they identify is the escapist gamer - this is a gamer type that does not shy away from their IRL identity as a ‘gamer’ when prompted. They play games to unwind, hang out with friends, escape real life and/or to have something enjoyable to put effort into.
The second type of gamer they identify is the ashamed gamer - these mainly *do* shy away from an IRL identity as ‘gamer’, despite playing games regularly.
Additionally, and interestingly enough, these gamers are specifically pointed out by the students as gamers that they recognize, although as a much smaller percentage of all ‘gamers’. They mention professional gamers, who play as their job, who practise games to improve their skills, compete in professional matches and work on their branding on online platforms. They also describe celebrity gamers, who are similar to professional gamers in their focus on branding and the attachment of money to their gaming habit, but are different in the absence of a need to compete. It is replaced with an even larger amount of attention to detail in their personal branding: the celebrity gamer focuses on their digital and physical accessories (peripherals, chair or room decor) in order to improve their appearance. The teenagers seem to have a sense of admiration for both.

The main takeaway from these two studies is that how teenagers view gamers is majorly influenced by what they *see*. This is shown directly by the effect of the exposure of a gamer of a different gender in the first study, but it is also - albeit more subtly - shown in the second study. In the second study, the emergence of the ‘celebrity’ and ‘professional’ gamer is quite puzzling: in none of the other studies discussed in paper do these two even remotely come to the surface. We can infer that these two types of gamers are identified because this is what teenagers *see* when they interact with gaming platforms on the internet, despite it not accurately reflecting what a typical ‘real gamer’ is, on average.
So what we see when we compare the perception of teenagers to both a ‘standard gamer’ and to what they define as ‘types’ of gamers, is that they attach values to these ‘gamers’ that conform with stereotypes as mentioned in section I, and this being despite the fact that in

reality, real gamers are much more often than not, not the same as what the stereotype depicts, as per section II.
In spite of that, students are also malleable in their perception of stereotypes. Their main influence on what they perceive to be a ‘gamer’, what characteristics they should attach to ‘gamers’, and what types of gamers there are, is all dependent on what they witness and come into contact with. What they perceive to be a gamer can be altered by exposing them to different individuals that play games or perform tasks related to gaming.

## **2.4)** What influence does this have on their academic choices?

There are naturally many factors that affect academic decisions, and the influence of stereotypes (of all kinds) is no exception.
One study [11] was performed in response to the changes brought to our world by the CoViD-19 pandemic. In an attempt to aid in closing the gender gap in STEM studies, two Italian universities have been hosting an extracurricular summer camp for young women in order to attract them to ICT disciplines, called ‘Digital Girls’. For the sake of this background research the main focus will be on how effective this camp was in altering academic choices for these young women.
26% of girls had tried coding/programming before attending the summer camp. Among the girls aiming to continue their studies, it became clear that those who have already experienced coding are much more likely to continue their studies in an ICT sector than their peers at a staggering 45.5% vs. 10.8%. That result mainly highlights the importance of experiencing coding at school and how such an experience may positively and significantly impact students’ willingness to choose IT as a field study as a woman. Additionally, after the summer camp the percentage of girls that were motivated to continue their studies in an ICT field increased from 18.1% to 25.9%. The camp resulted in girls there saying they are 18% likely to continue in a STEM-study, where the current amount of girls enrolled there is around 1% of all european tertiary studying girls.
The main importance that the study shows is the willingness of young women to try or continue in a STEM or ICT field. The large percentage increases show that with just a small ‘push’, the women can recognise their liking, affinity or overall interest in the field.

Another study [12] examined the impact of gender stereotypes and their influencing of students’ academic choices at a University in Hungary. Their research question was: “What is the impact of gender stereotypes and demographic factors on students’ academic choices?”, and they performed their tests by conducting qualitative interviews with international students at undergraduate and postgraduate level. There were, as to be expected, many factors that were meaningful to the choice for education for the subjects. The factors that concern this background research are those relating to the influence of stereotypes. For the sake of transparency, the other categories were: ‘family background factors’, ‘university criteria factors’ and ‘mental position and its influence’. The fourth category was on ‘gender stereotyping factors’:
Initially, they asked the students to rank 26 personality characteristics, and to rank them according to how masculine or feminine they perceived that characteristic to be. This is a similar test to the one conducted in source nine [9] of this background research, but then with more of a focus on gender stereotypes, rather than gamer stereotypes.
Subsequently, the students mentioned that they chose their study according to their gender when prompted. They did not only choose their study based on capabilities and

qualifications, but also according to their societies' expectations for men and women. For example, some subjects mentioned that they believed that engineering would potentially be 'too complex' for women, and that it was a man's study, just like the profession of pilot, soldier or someone in the navy. Female majors were found to be dependent on their education but also their family, one of the participants mentioned. Professions mentioned as typically female were 'nurse' or 'teacher in the nursery'. The author of the background research mentions their shock when exposed to the results of their study. The study clearly suggests that stereotypical influence can majorly sway the choice of study for those who are in the position to make such a choice.

The main takeaway from these studies is not only that stereotypes *do* have (significant) influence on academic choices, but also that exposure to content of courses that might have once been unappealing in fact *can* influence young adults in their academic choices.
We can extrapolate this to the influence that gamer stereotypes have on young adults. Young adults can quite likely have their academic choices be affected by gamer stereotypes and their perception of gamers. However, exposure to academics that might be typically tied to 'gaming', such as ICT studies, can also positively influence the likelihood of young adults to choose such studies in the future.

## **2.5)** In summary

The goal of this background research was to assess the effect of gamer stereotypes on students' academic choices, particularly those in their teenage years. This has been achieved by dissecting the stereotypes of gamers, how 'well' represented they are in our society, how teenage students identify stereotypes and how they experience them, and finally if this influences teenagers in their academic choices.

The main perceived stereotype of a 'gamer' is still one of an introverted, lazy male with poor physical habits and health conditions, and there exist similar stereotypes in other demographics, such as the 'girl gamer' and her sexualised, (male-)attention seeking attitude. In reality, gamers are everywhere, and it has become a much more mainstream type of entertainment. Gamers can be almost anyone, and demographically, who gamers are has been vastly expanded. Teenagers' view on what gamers are and what the stereotypes of gamers are are mostly conforming to the stereotypes mentioned above, but most importantly, they are influenceable by what they *see*. When someone performs a gaming activity, teenagers are likely to point them out as a gamer, regardless of their perceived stereotypes. The perception that teenagers have of who 'gamers' are is malleable, and (long term) exposure to gamers of multiple demographics (just like what the reality of gamers is made up of) will likely diminish the severity and diversity of stereotypes attached to gamers. Alongside that, the choices that students from teenage to young adult years make in their academic choices are influenced by stereotypes, particularly gender stereotypes, and by extrapolation, also by the stereotypes of 'gamers'. Young adults were (often positively) influenced in their academic choice by exposure to coding and programming exercises, especially if they did not have prior coding experience. They would opt to continue with a study in an IT or STEM field more often. Their academic choice was directly influenced by exposure to course material from courses with typically stereotyped narratives attached to it.

We can conclude that stereotypes of gamers *do* influence the academic choices of young adults and teenagers, but also that both the stereotypes, their view on who ‘real’ gamers are *and* their sentiment towards academic courses is something influenceable *itself*.

It is possible to influence young (prospective) students and persuade them to enrol in an IT course, programming course or other STEM course, especially when suspected that they might be interested in such a course. Even if they might currently be swayed against opting for such a course due to their perceived stereotypes, a prolonged exposure to ‘gamers’ that accurately represent the real demographics behind gamers (one that has almost no limitations), can affect that stereotypical perception and change it entirely. The teenager may then, after having their perception change, make an academic choice that they would previously not have made.

In regards to the relevance of this influenceability towards this project's challenge, the main takeaway is that teenagers are influenceable towards what they encounter, despite previous judgments. This means that this project will use a neutral theming and styling approach to not evoke any negative judgments due to stereotypes, but with programming elements to tap into the influenceability. The opportunity this influenceability of teenagers provides will be used when introducing programming to teenagers through an interactive, de-stereotyped, minigame.

# Three *The Product*

## **3.1)** Ideation

### **3.1.1)** Base Game Choice

Many different types of games exist, from sprawling massively multiplayer online role playing games, to brain-breaking 4X strategy games, to high intensity shooter campaigns, to simply fun family oriented party games. Most games you'll find from these categories will be heavily themed (as they should be), but as per the challenge of this report, the goal for this product is to find a way to create a minigame that has enough styling to engage the user, but is also de-stereotyped in order to attract a broader demographic.
Unthemed core mechanics behind games can be categorised into *strategy* based games, *reflex* based games, and *experience* based games. One of the oldest games in our human history is the game of chess, a classic game with minimal styling, and very low theming. The medieval setting with kings, queens, knights and bishops to nominate otherwise arbitrary chess pieces with special game rules is the only theming and styling found in the game. Chess is a very classic *strategy* based game.

The minigame made for the challenge is also a strategy based game.

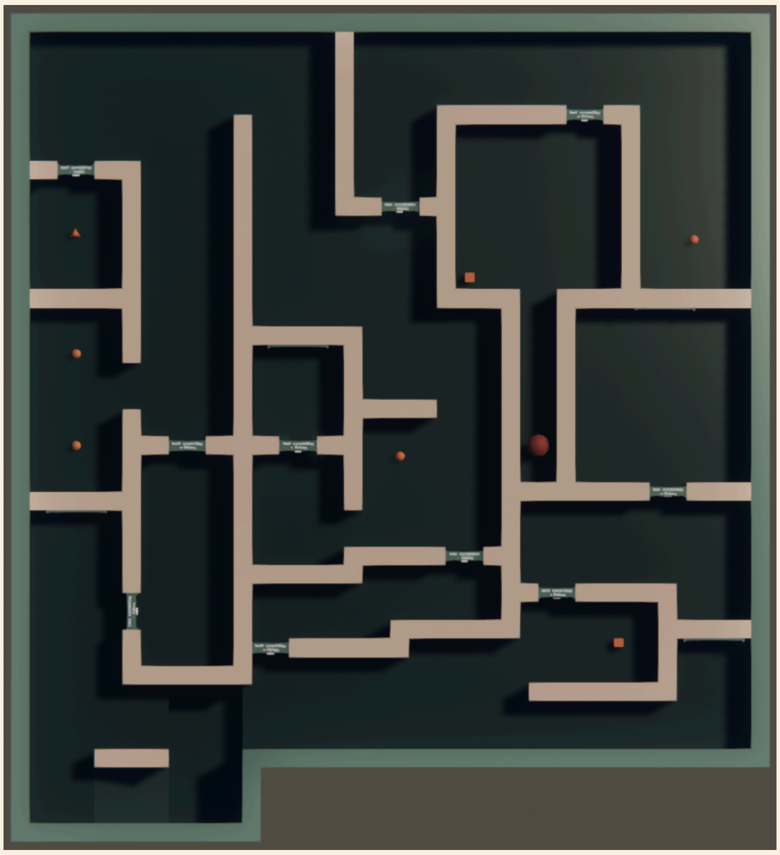
The choice of which of these three core gaming archetypes I wanted to use for the product came down to a few considerations.

**Experience based games:**
The main issue with choosing to make an experience based game for this project is what it takes to create a *good* experience based game. Good experience based games do not only require a good concept, but also simply a lot of work: 3D or 2D assets require a lot of time and effort (and likely multiple iterations), and to have a proper experience, the entire game needs visual and mechanical cohesion. That cohesion upgrades the requirement of quality of all other objects in the game, but also in the fluidity of their interactions, animation, physics collisions, and many other aspects. With a larger team, a clear vision on what to create, consistent effort, *and* more time, an experience based game is possible for this project (and perhaps even a great option). Unfortunately, those are not available. Scaling down the operation and creating an experience that lasts less than a minute is not sufficient for the successful completion of this project either. Ruling out an experience based game for this project is the best choice.

**Reflex based games:**
Workload wise, a reflex based game would be a much better option over an experience based game. The main issue with choosing a reflex based game however is found in the setting that the game will eventually be played and presented at. At a high school open day, a game that is fun for the player, but unintelligible for those passing by or even those who

are trying to pay attention will hinder the effectiveness of attracting a broader demographic. Reflex based games often have quickly changing camera angles, a lot of motion (and possibly motion blur), are fast paced and overall unsuitable for attracting passersby beyond an initial interest in whatever flashy instance happened on screen.
Additionally, integrating a programming-introduction mechanic is more difficult in a reflex based setting, over a strategy based setting.

**Strategy based games:**
Programming itself is a series of solving puzzles: solve the puzzle of *what* you want to create, then solve the puzzle of *how* you want to create it, and lastly solve the puzzle(s) of *fixing* the things that go wrong during the process. Using a strategy based game as a base concept makes the integration of programming elements and regular puzzle games possible: a logical combination.
There are other advantages to creating a strategy (puzzle) based game over one of the other two options: a puzzle game requires much less resources to complete successfully than an experience based game. A puzzle game also is way easier on the eyes for passersby. While it might not have the same blockbuster short-term impact as a reflex based game, it can have a longer lasting effect, which can cause passersby to stick around longer, and actually want to play the game themselves.



The choice made for this minigame is mainly a strategy based game. Solving puzzles is a typical strategy game element. Additionally, the game has some experience based gameplay design as well, found in the progression element of the game, and in the experience of growing a ‘stronger’ character with more developer functions. Experience is also found in the functions of destruction and creation that are enabled with the custom if-statement. Lastly, the game employs some limited reflex based design, particularly in the challenge of handling higher boost speeds that the player can adjust themselves. The obstacle of crossing the gap in the southwestern corner of the level is arguably also a reflex based challenge.

### 3.1.2) Game-within-a-game Idea

While the core game is a three dimensional puzzle game, using simple interactions between a rollable ball as a player, and a few different mesh shapes as keys to unlock corresponding doors. The integration of the programming aspect to introduce prospective students to programming is made possible due to the simplicity and elegance of the underlying puzzle aspect. The integrating mechanic was originally ideated as a ‘game-within-a-game’. There would be aspects of the game that would be limited in their space, with a ‘larger’, overarching game-space that has influence in the smaller game within.

Quickly, this ‘overarching’ game-space was ideated to be a sort of ‘developer-only’ setting, a ‘god-mode’ that could control aspects of the inner game in a way that would seem ‘unnatural’ by the standard of the inner game, but normal and possible in the greater game.
*“A level that has challenges for the hero that can’t be solved by the player, but can only be solved by the developer.”*, where the ‘player’ refers to the user-controllable ball in the puzzle, and the ‘developer’ to the user-controllable game controls of the greater game.

### **3.1.3)** ‘Developer Access’ Concept

Adding the programming aspect to this game-within-a-game idea was conceptualised when considering the real backend programming of the game: the doors, keys, and the player itself have real programmable attributes - why not allow *those* attributes to be adjusted by the ‘developer’? *“Don’t open the door with a key you find, but by directly changing the positional variables of the door itself.”* Here, the true core of the game's concept was born.

The very first prototype iterations of the project contained an elementary version of ‘developer access’. The base game contained some basic puzzle elements, and around 30% of the total screen was a large black bar with some buttons that influenced relevant game variables directly, such as enabling sideways movement, enabling the ‘boost’ function (was still called ‘sprint’ at that time) or directly collecting the first key in the scene.

The execution of the obstacles that can only be solved using developer access eventually evolved into three challenges: a gap in the floor that can only be crossed by turning up the boost speed of the player, a ‘broken’ door that can only be manually opened, and a lack of keys remaining in the scene, which calls for the retrieval of previously used keys from other doors - a function that is not enabled by default.

All of these challenges can only be solved by directly changing the state of game variables, which is made possible by the developer user interface, a straightforward panel with many different game variables which evolved from the black bar on the right side of the screen with just four buttons.

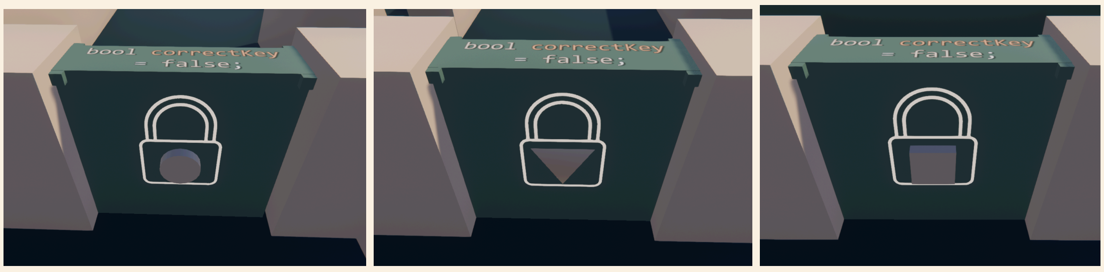
---

## **3.2)** Features

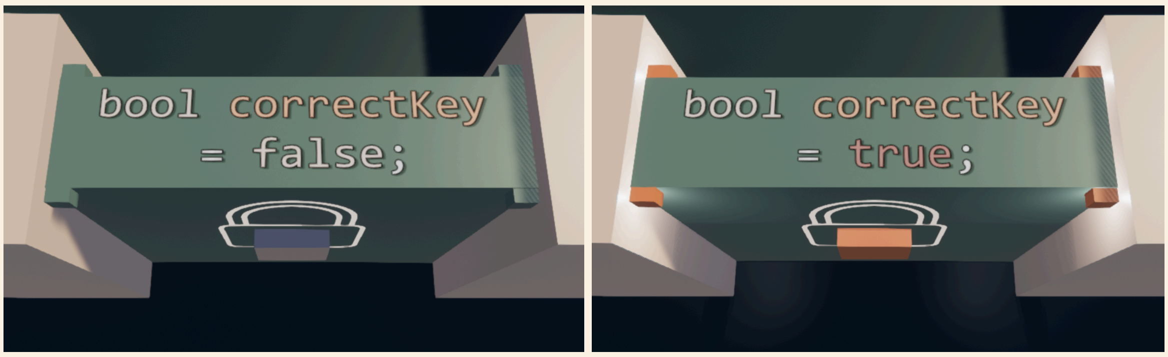
### **3.2.1)** Physical Objects

A number of different physical objects are found within the base game. As the absolute physical core, the game contains a flat floor, of about 20 meters by 20 meters. That floor is surrounded by outer walls, which cannot be interacted with in any way. The maze of the game consists of multiple thick walls that give the maze its shape, creating rooms, corridors and hallways. These walls also cannot be interacted with, except under one circumstance, which is revealed at the end of the gameplay cycle. To supplement the puzzle aspect of the game, some interactable objects are placed in the scene as well. In the following short sections, these interactable physical objects will have their functionality explained.

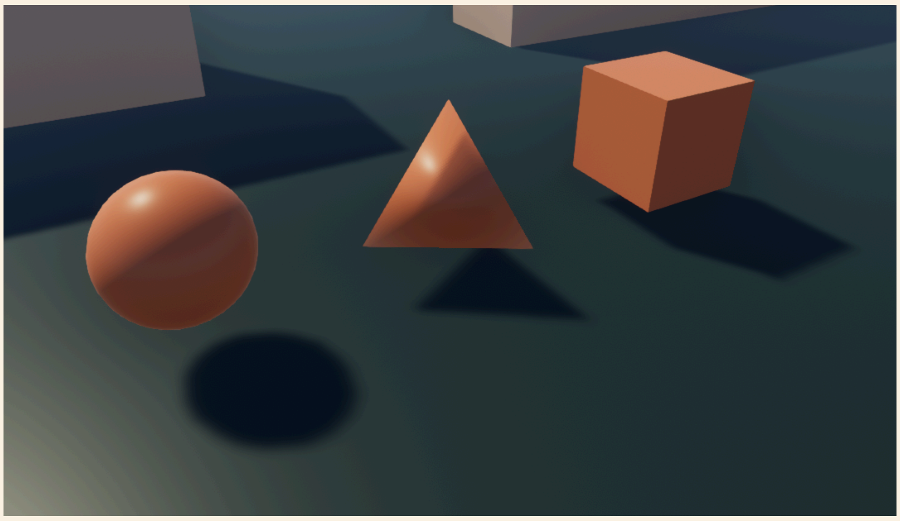
#### 3.2.1.1) Doors



The main puzzle aspect of the game comes in the form of doors that block the player's path. These doors have keys that the player can pick up and use to unlock the doors. Both the keys and the doors have three possible versions. The versions are identical, with one exception: a shape that symbolises the type of key needed to open them. between them.
There are three types of doors: a square door, a circle door and a triangle door.
The doors have a lock icon on the front, with the door's corresponding symbol in the center of that lock icon. On top of the doors is some text in a pseudocode format, that describes whether or not the player has brought the correct key in proximity to the door. If they have, then the symbol lights up, the lights on top of the door start shining, and the pseudocode boolean changes to 'true'.

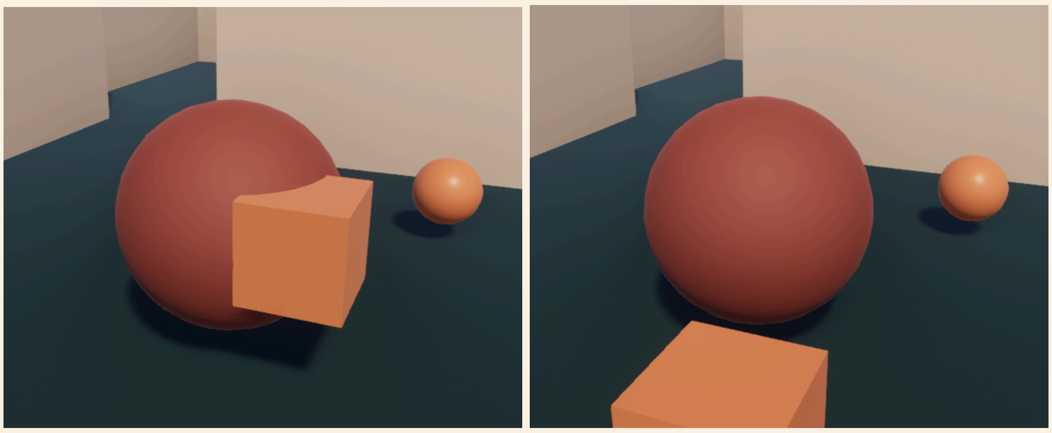


#### 3.2.1.2) Keys

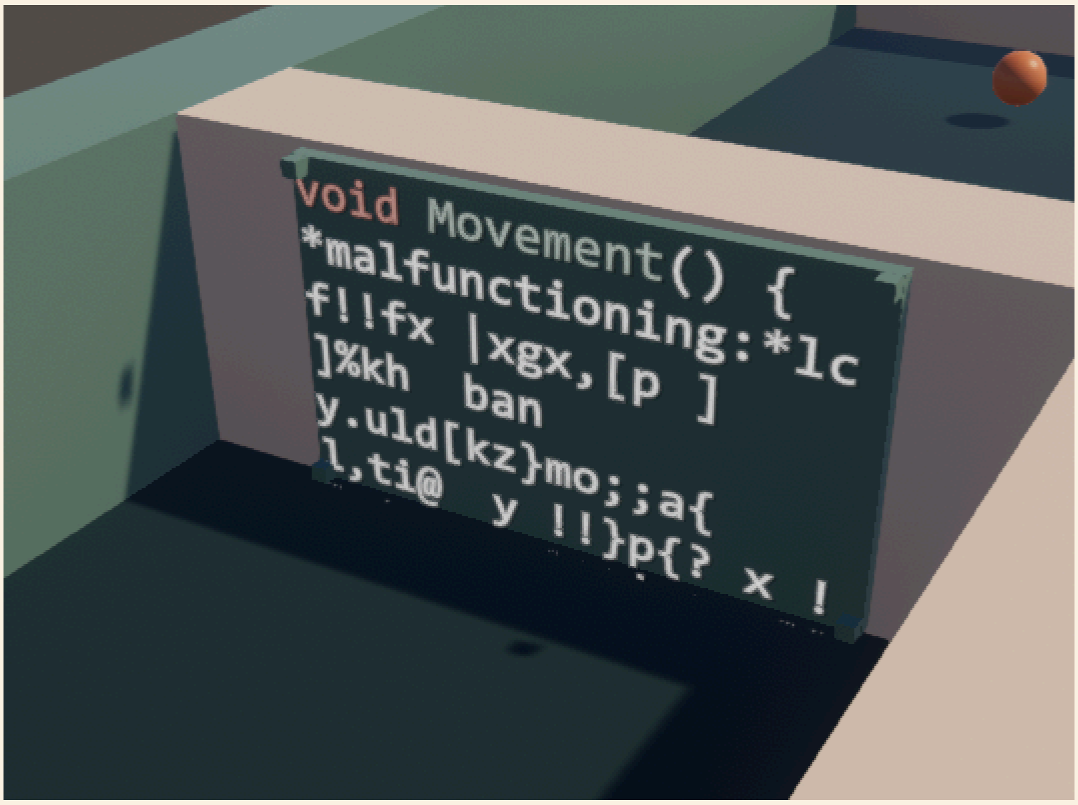


Like the doors, the keys come in three shapes: the square key, the circle key, and the triangle key.

They are represented by their different meshes, respectively a cube, a sphere and a tetrahedron (a three-sided pyramid). The keys can be picked up by the player and carried across the level. The player can then drop the keys, or use them to open the correct doors.

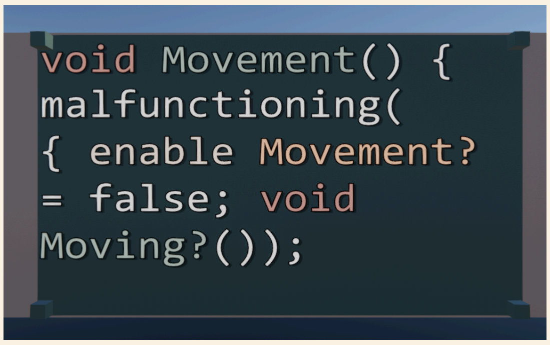


### 3.2.1.3) Screens

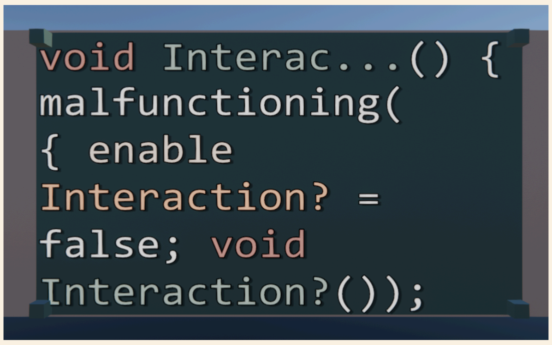


```
void Movement() {
*malfunctioning:*1c
f!!fx |xgx,[p ]
]%kh  ban
y.uld[kz}mo;;a{
1,ti@  y !!}p{? x !
```

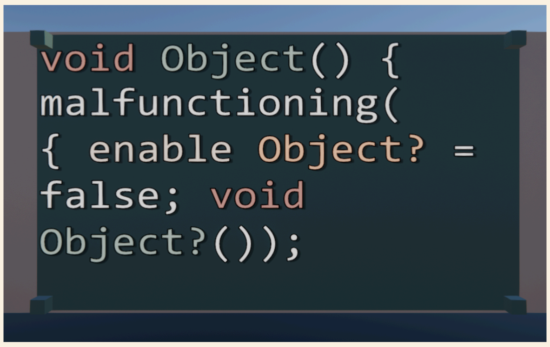
The third main object found in the game are the ‘screens’ that represent the four main developer functions. The screens have a text object with scrambling pseudocode on them, used to represent the ‘faulty’ aspect of their design. Additionally, the screens feature four small ‘lights’ on the top and bottom corners, in similar fashion to the doors. They light up when the player gets close to the screen.



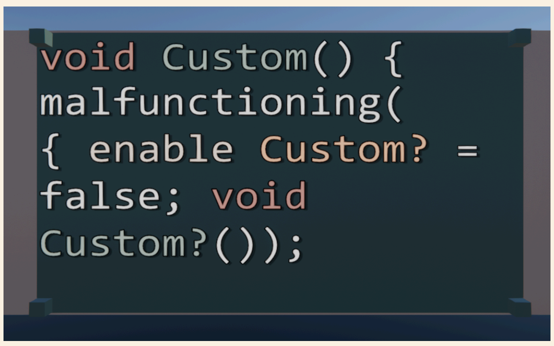
```
void Movement() {
malfunctioning(
{ enable Movement?
= false; void
Moving?());
```



```
void Interac...() {
malfunctioning(
{ enable
Interaction? =
false; void
Interaction?());
```



```
void Object() {
malfunctioning(
{ enable Object? =
false; void
Object?());
```

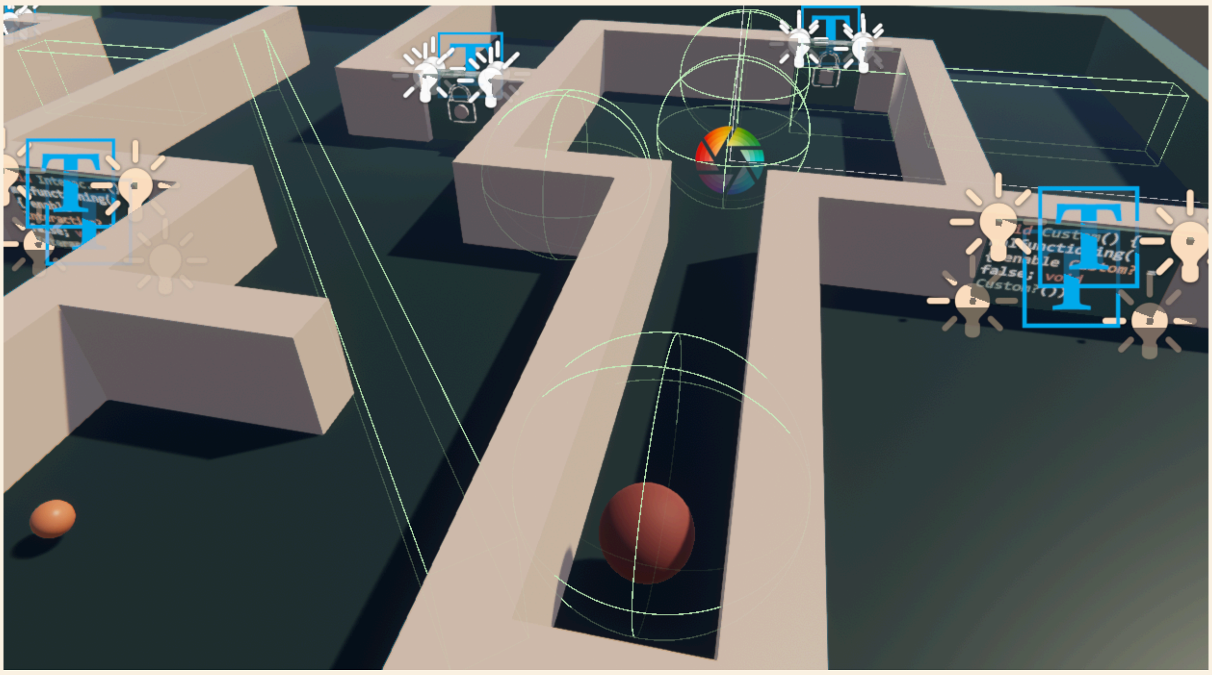


```
void Custom() {
malfunctioning(
{ enable Custom? =
false; void
Custom?());
```

**3.2.1.4) Trigger Zones**

The last main object that is found in the game is an invisible one: the Trigger Zones.

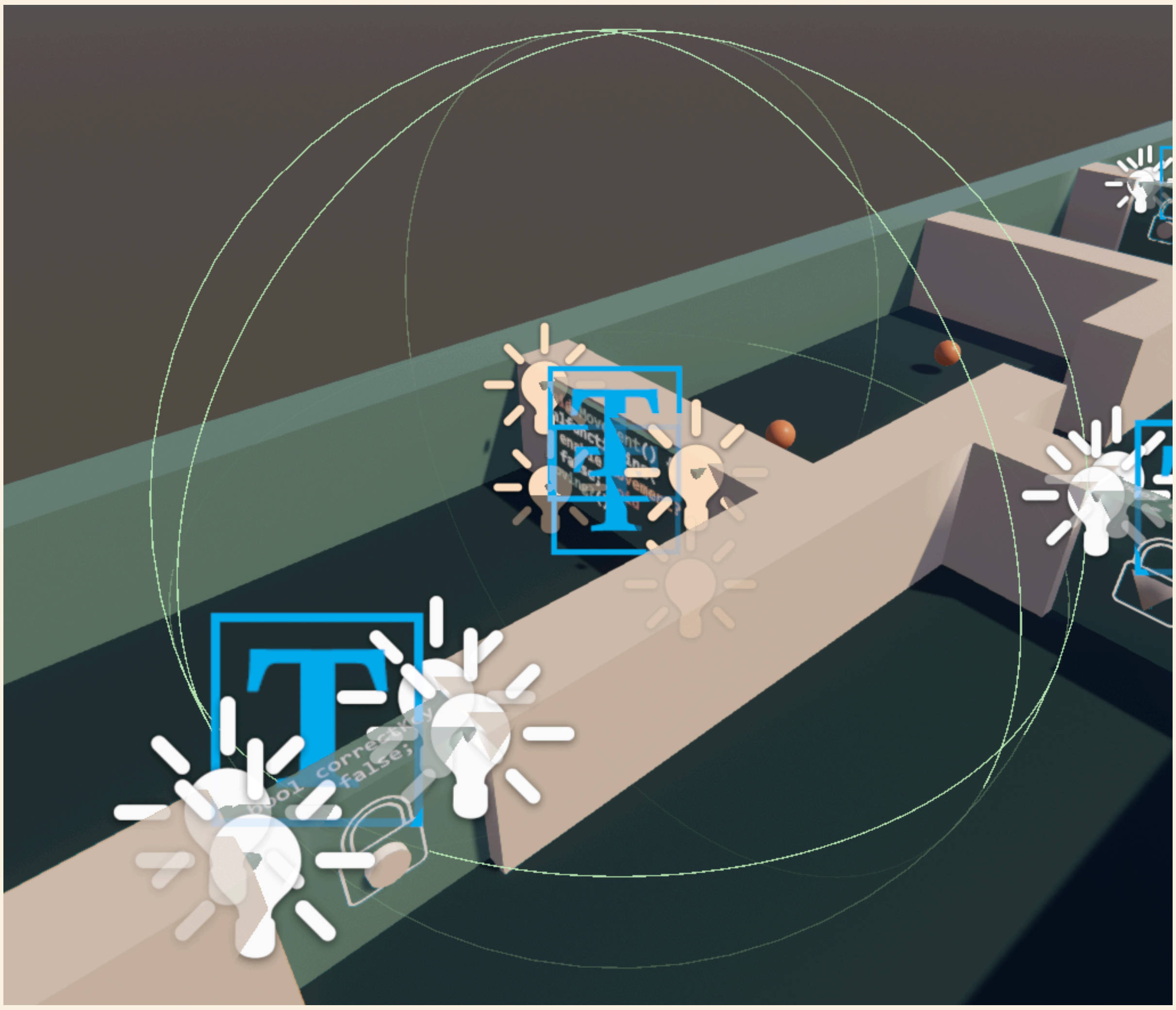
Two types of invisible Trigger Zones exist: ‘suggestion user interface trigger zones’ and ‘glitch trigger zones’. The screens found in the game have glitch trigger zones attached to them. Their function is very different from the suggestion UI Trigger Zones in the scene.



The eight suggestion UI trigger zones and their trigger colliders (in green).

A suggestion UI trigger zone is what causes an element of the user interface that introduces a game mechanic to either fade into the screen, or out of the screen.

For example, the collider around the player in the image above causes the UI element that explains the ‘W’ key to appear on screen. More about the Suggestion UI itself is found further in this section.



The radius of a Glitch Trigger Zone ^. At the edge, the glitch effect is minimal, but it gets worse the closer the player gets to the centre. More about the glitch effect further in this report.

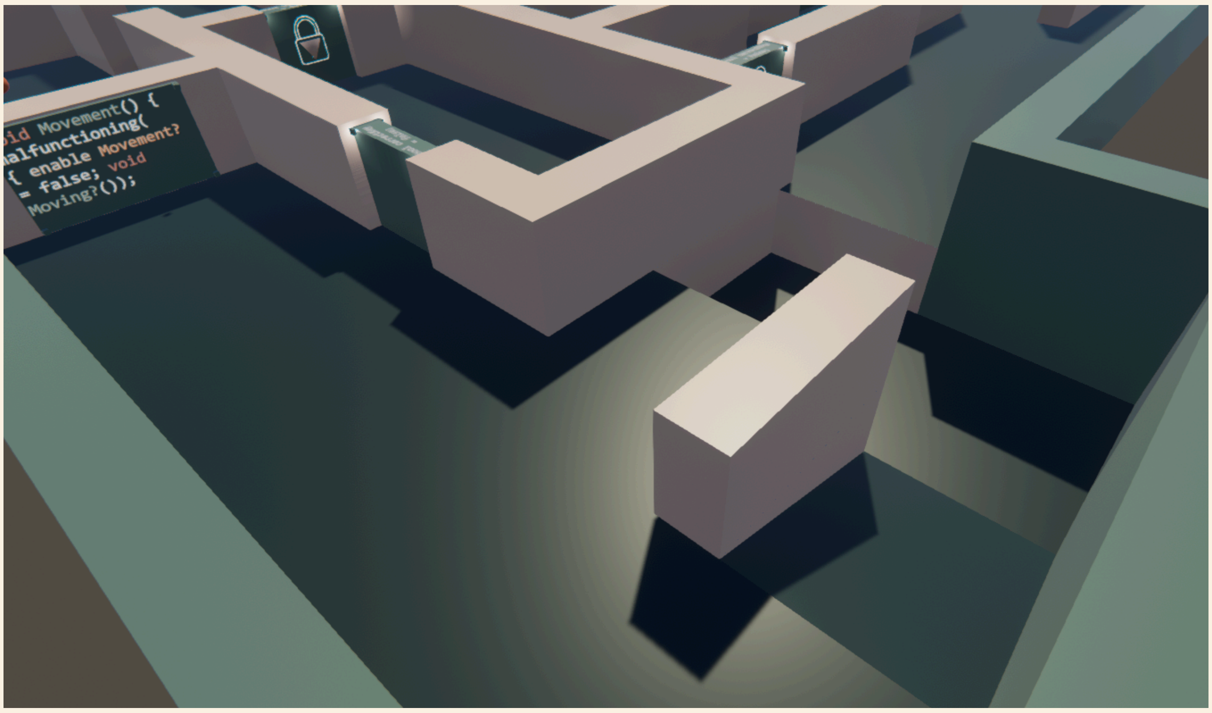
### 3.2.1.5) ‘Broken’ Door

bool {cnfoqmy. n?{qj = false;
bool ,ok;oe$qm&f &g = false;
bool #{, eqc % !f, = false;

The broken door is a special feature of the game which only appears in the game once. While it looks similar to any regular door, it can never be opened directly by the player. The only way to open this door is by using the developer UI.

The door has glitchy text instead of stable text on top, and the icon meshes shuffle and are randomly updated. Additionally, the broken door has a glitch trigger zone attached to it.

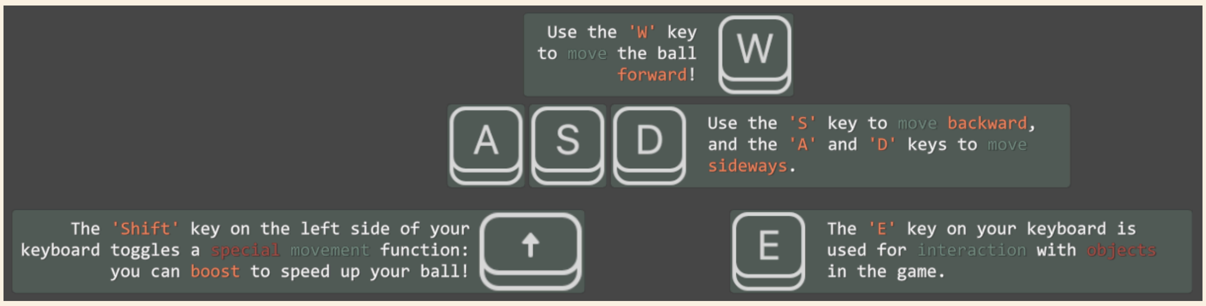
#### 3.2.1.6) Gap and Ramp



About halfway through the level, the player encounters a gap in the floor, with a ramp that lets the player get back up to the regular elevation of the level in case the player falls into the gap.
This gap is another obstacle that can only be overcome by using the developer user interface.

### 3.2.2) User Interface Features

#### 3.2.2.1) Suggestion User Interface



Use the 'W' key to move the ball forward!

Use the 'S' key to move backward, and the 'A' and 'D' keys to move sideways.

The 'Shift' key on the left side of your keyboard toggles a special movement function: you can boost to speed up your ball!

The 'E' key on your keyboard is used for interaction with objects in the game.

The suggestion user interface consists of four UI panels that pop up in the screen if their relevant Trigger Zones are interacted with.
The panels provide basic information of the controls of the game.

#### 3.2.2.2) Developer User Interface

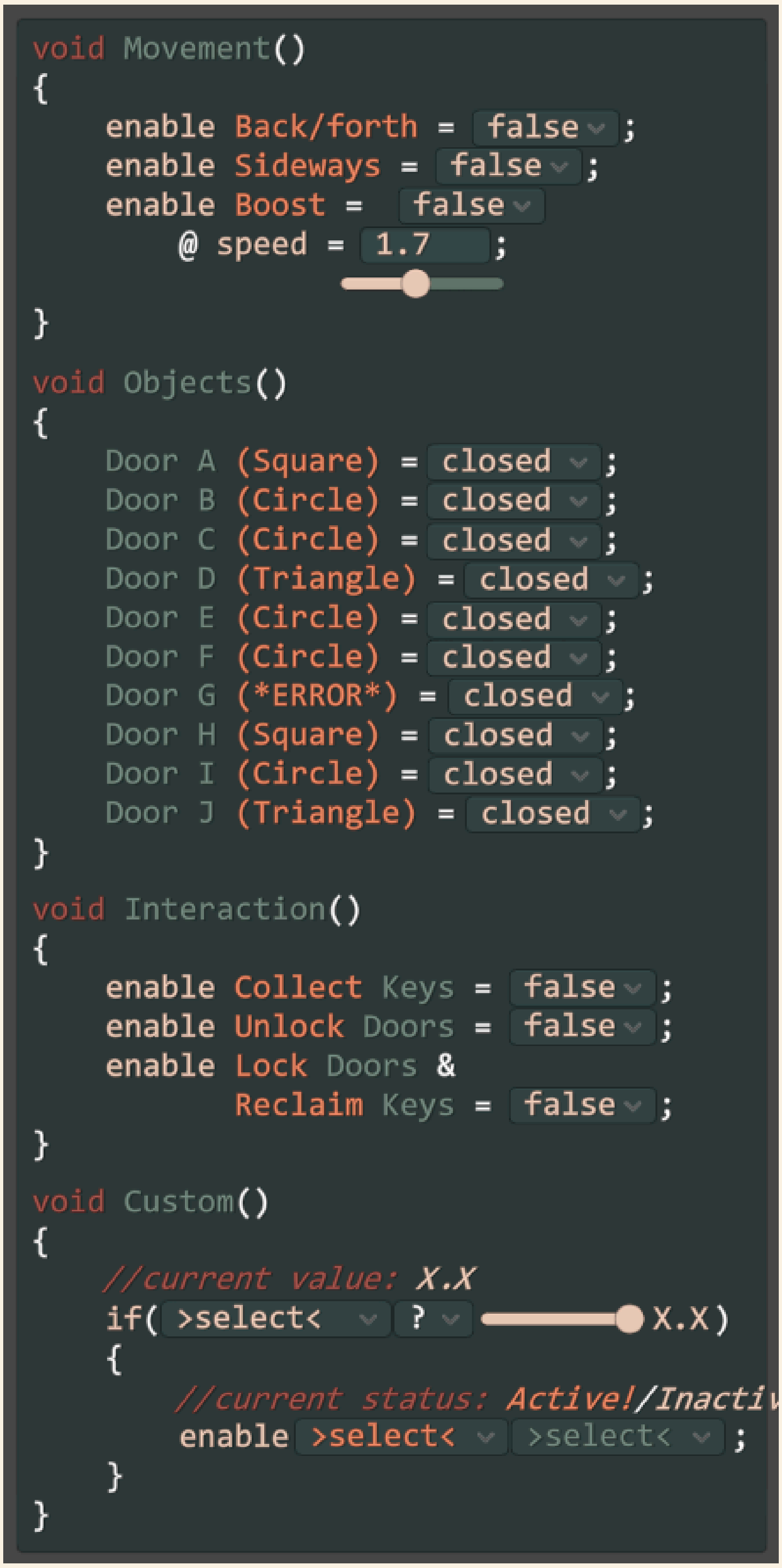
The developer User Interface consists of text in a Pseudocode style, that makes references to game variables. The variables can have their status adjusted through the dropdown items near the end of each line of pseudocode.

The interactions of the Dev UI will be expanded upon in the relevant tab of the Interactions part of this report.

While most of the ‘voids’ are pretty self explanatory in what game variable they adjust, the Custom() method is not.
It is a special method with multiple dropdown boxes that all interact with each other.
The user can write a custom ‘if-statement’ using the five dropdown menus, which will activate if all conditions are met.

For example, the user can make:
if (‘speed’ ‘>’ ‘1.0’)
{
 enable ‘break’ ‘walls’ ;
}

Which will, indeed allow the player to break through walls as long as the player's speed is higher than 1.0f.



```
void Movement()
{
    enable Back/forth = false ;
    enable Sideways = false ;
    enable Boost = false
        @ speed = 1.7 ;

}

void Objects()
{
    Door A (Square) = closed ;
    Door B (Circle) = closed ;
    Door C (Circle) = closed ;
    Door D (Triangle) = closed ;
    Door E (Circle) = closed ;
    Door F (Circle) = closed ;
    Door G (*ERROR*) = closed ;
    Door H (Square) = closed ;
    Door I (Circle) = closed ;
    Door J (Triangle) = closed ;
}

void Interaction()
{
    enable Collect Keys = false ;
    enable Unlock Doors = false ;
    enable Lock Doors &
           Reclaim Keys = false ;
}

void Custom()
{
    //current value: X.X
    if( >select< ? X.X)
    {
        //current status: Active!/Inacti
        enable >select< >select< ;
    }
}
```

## 3.3) Interactions

### 3.3.1) Physical Interactions

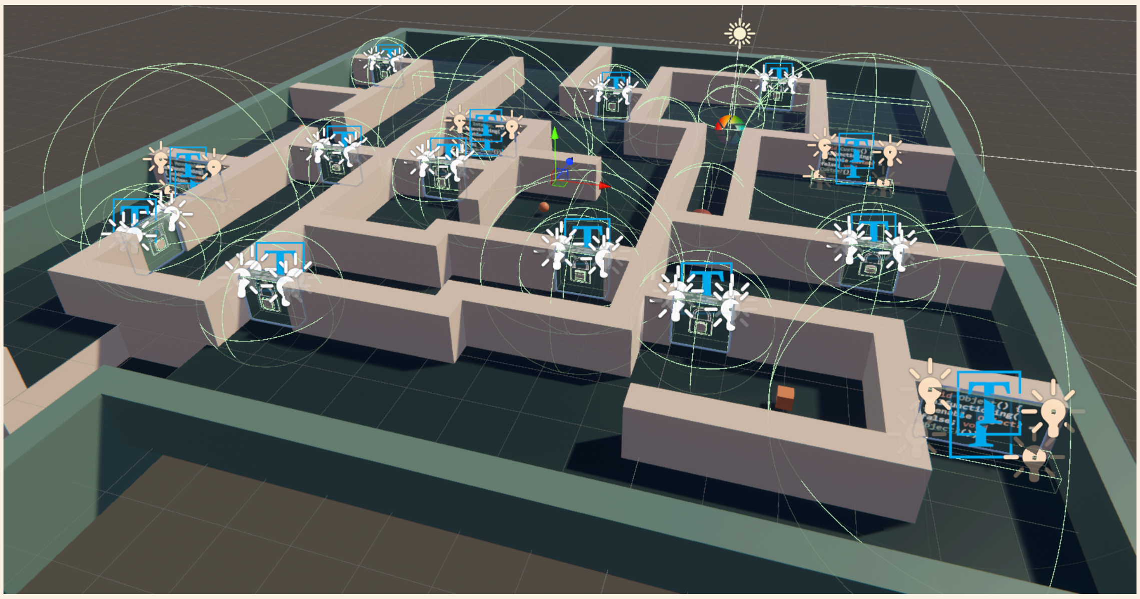
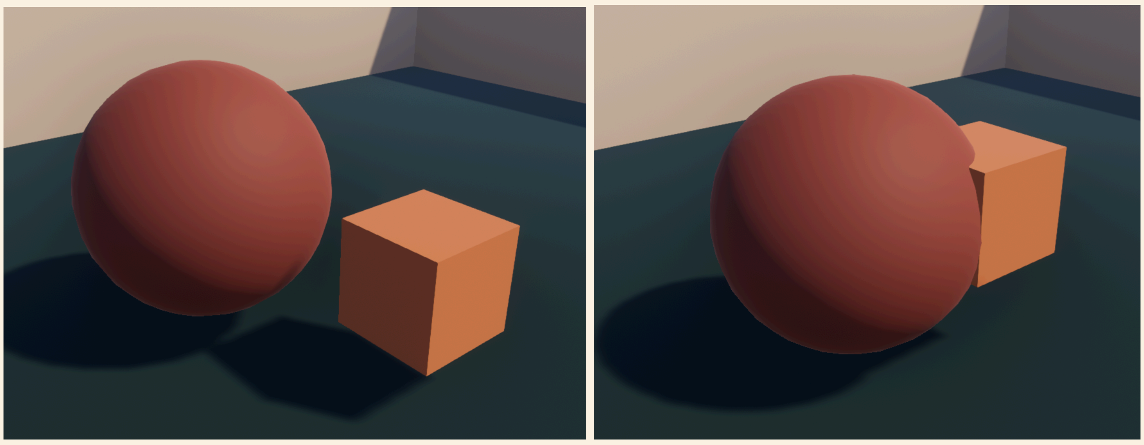


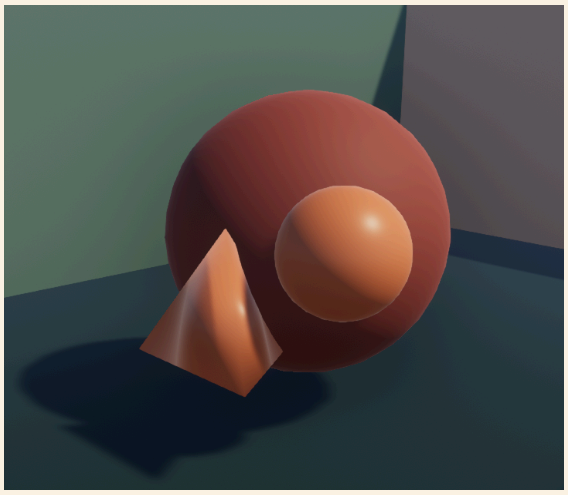
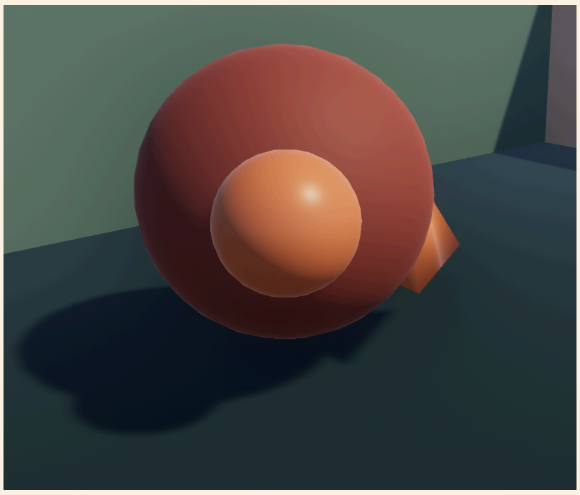
Image showing all colliders and triggers in the scene: everything that can be interacted with.

**3.3.1.1) Player and Keys**

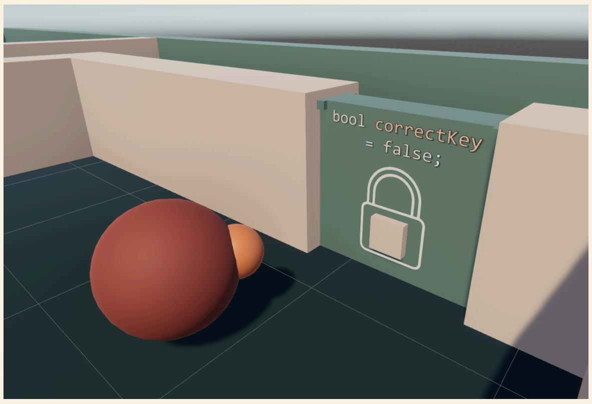
When the player touches a key, they will pick it up.

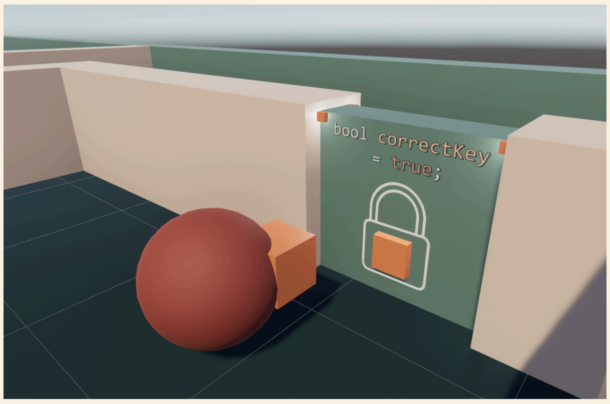


If the player is already holding a key, it won't pick up a second key. The player does not collide with the object, and goes straight through the second key.

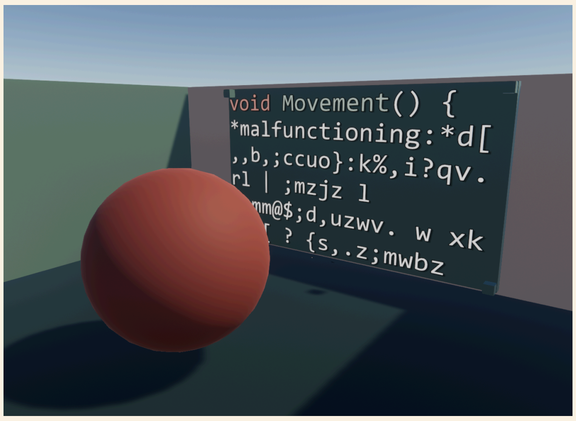
### 3.3.1.2) Player and Doors

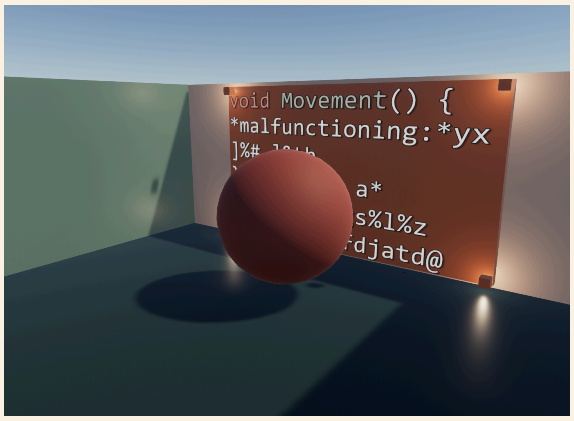




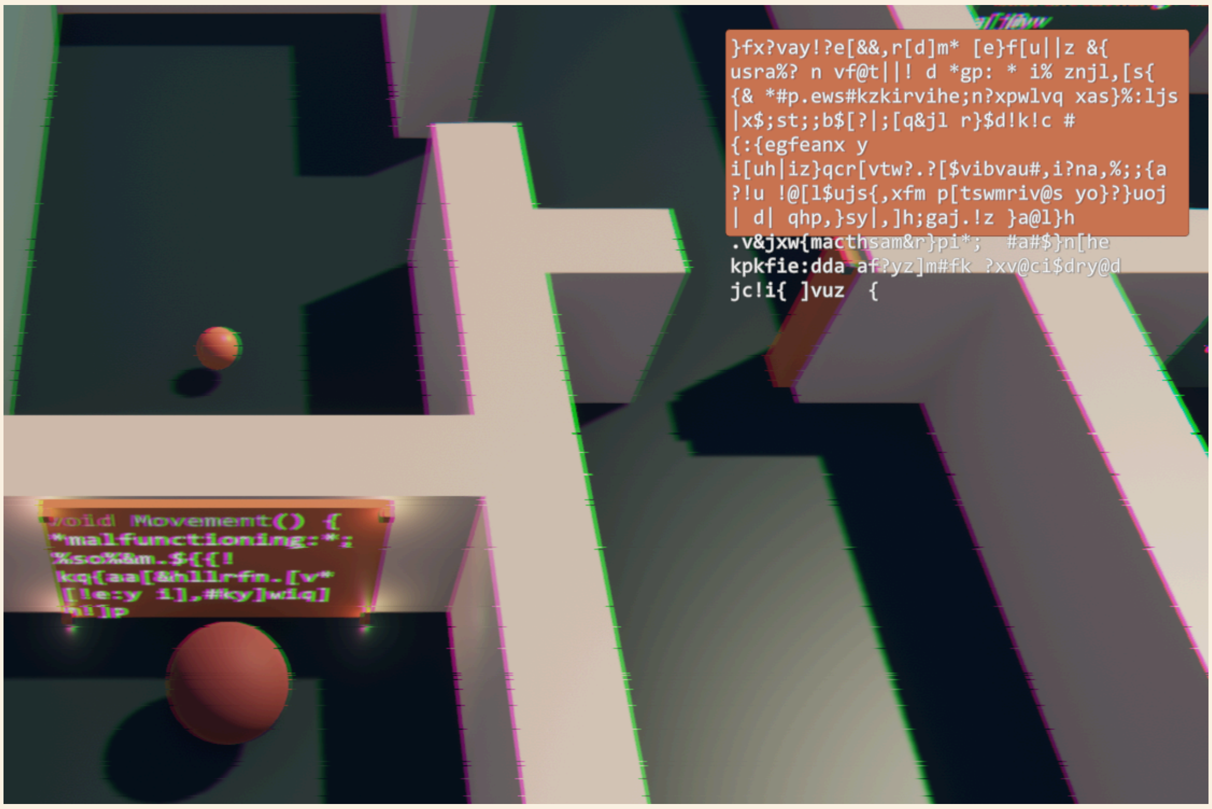
When within proximity, and while holding the correct key, the door lights up, turns its door icon mesh to orange, and displays that the ‘correctKey’ boolean is true. The player can then interact with the door using the interact key ‘E’ on the keyboard, to open the door.
If the player is not within proximity, or if the player is within proximity but while holding the wrong key, the door will remain idle, with its lights turned off, and the boolean remaining false.

### 3.3.1.3) Player and Screens





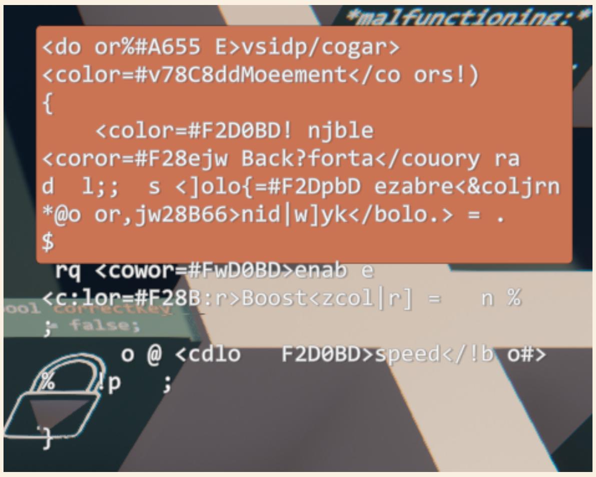
When within proximity, the screen lights up and its whole mesh is turned orange. Additionally, the randomised string of text is scrambling faster, and the relevant developer user interface block pops up on the players screen. The text of the block is still scrambling, and the background of the panel is orange.



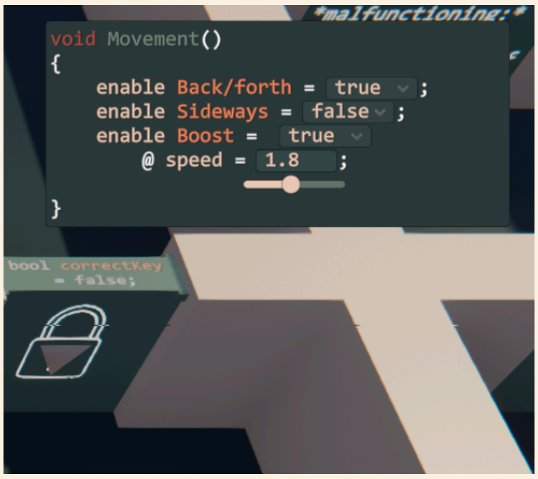
```
}fx?vay!?e[&&,r[d]m* [e}f[u||z &{
usra%? n vf@t||! d *gp: * i% znjl,[s{
{& *#p.ews#kzkirvihe;n?xpwlvq xas}%:ljs
|x$;st;;b$[?|;[q&jl r}$d!k!c #
{:{egfeanx y
i[uh|iz}qcr[vtw?.?[$vibvau#,i?na,%;;{a
?!u !@[l$ujs{,xfm p[tswmriv@s yo}?}uoj
| d| qhp,}sy|,]h;gaj.!z }a@l}h
.v&jxw{macthsam&r}pi*;  #a#$}n[he
kpkfie:dda af?yz]m#fk ?xv@ci$dry@d
jc!i{ ]vuz  {
```

```
void Movement() {
*malfunctioning:*;
%so%&m.${{!
kq{aa[&hllrfn.[v*
[le:y i],#ky]wiq]
hl]p
```

If the player presses the interact key ‘E’ while the screen is in this ‘activatable’ state, then the screen itself will disappear, and all focus will be on the scrambling text in the user interface. The scrambled text in the user interface will slowly unscramble, allowing the user to see the fully functional developer user interface come to life.



```
<do or%#A655 E>vsidp/cogar>
<color=#v78C8ddMoeement</co ors!)
{
    <color=#F2D0BD! njble
<coror=#F28ejw Back?forta</couory ra
d  l;;  s <]olo{=#F2DpbD ezabre<&coljrn
*@o or,jw28B66>nid|w]yk</bolo.> = .
$
 rq <cowor=#FwD0BD>enab e
<c:lor=#F28B:r>Boost<zcol|r] =   n %
;
      o @ <cdlo   F2D0BD>speed</!b o#>
%    !p   ;
}
```



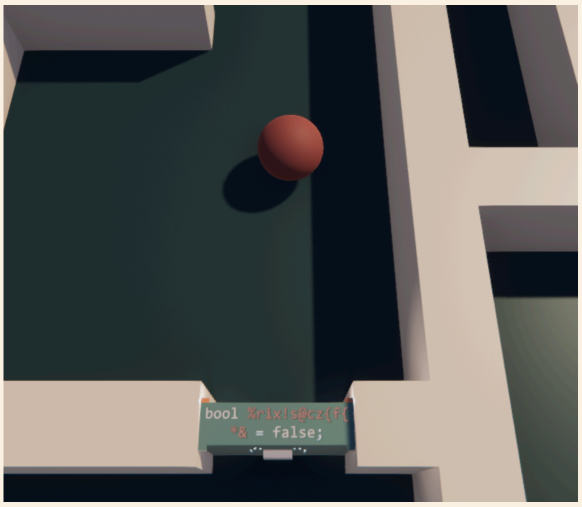
```
void Movement()
{
    enable Back/forth = true ;
    enable Sideways = false ;
    enable Boost = true
        @ speed = 1.8 ;
}
```

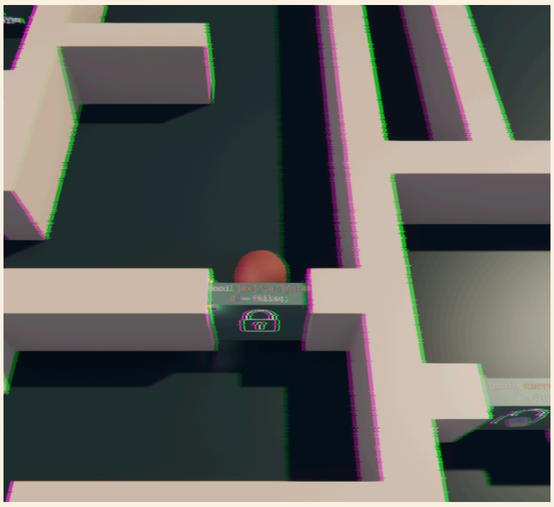
Once the text is fully unscrambled, the dropdown lists, input fields and sliders become available, and the Developer User Interface Block can be used!

### 3.3.1.4) Player and Trigger Zones

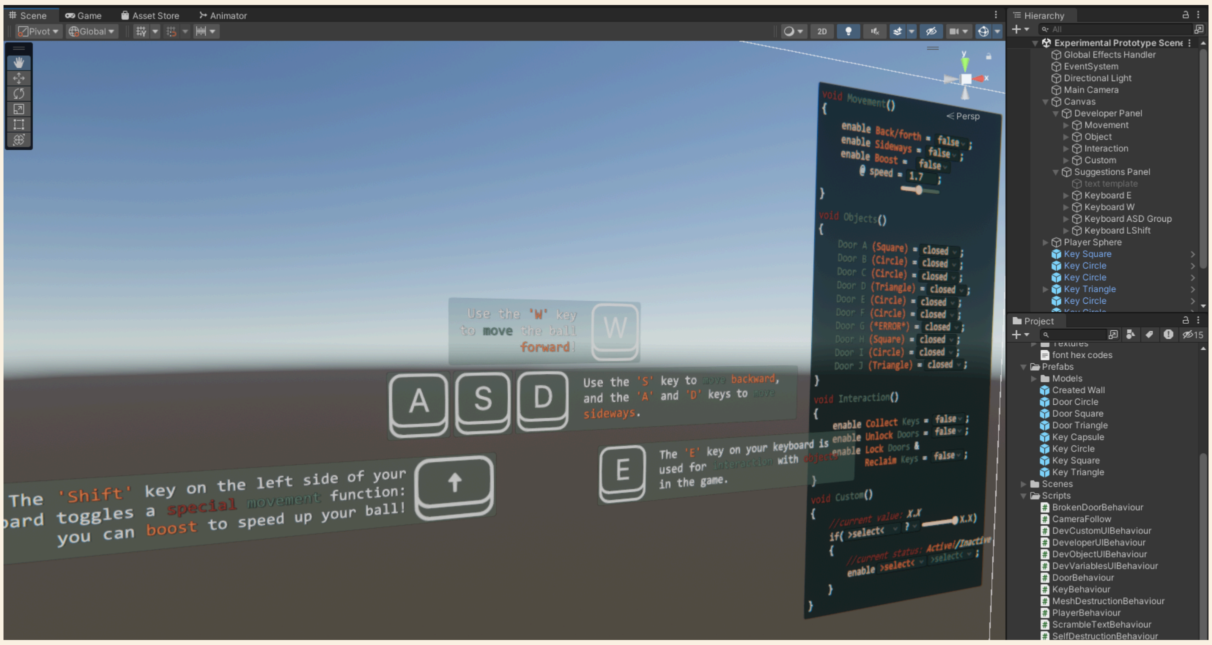
Like the Developer Screens, all Trigger Zones in the scene work with a collider, set to a trigger, that checks when something is touching it. There are two types of trigger zones, one for Suggestion UI elements, and one for glitch trigger zones.

The Trigger Zones of the Suggestion UI work in a very straightforward way. They only perform their task once, as their game object is set to inactive afterwards.
There are eight trigger zones in the scene that affect the suggestion UI. There are four elements to the suggestion UI, as will be described below in the relevant section of developer interactions. The interaction of the trigger zones is simple: one of the four elements of the suggestion UI will either be enabled, or disabled, with a short fading animation.
The Glitch Trigger Zones are more dynamic. They aren't interacting with the player only when their trigger is entered or exited, but rather they interact with the player every frame that the player is inside the trigger. As long as the player is inside the trigger, the glitch effects applied to the camera - 'scan line', 'colour drift' and 'horizontal shake' - will gradually be turned up, based on the proximity of the player and the centre of the trigger zone.





## 3.3.2) 'Developer' Interactions



```
void Movement()
{
    enable Back/forth = false ;
    enable Sideways = false ;
    enable Boost = false
      @ speed = 1.7 ;
}
void Objects()
{
    Door A (Square) = closed ;
    Door B (Circle) = closed ;
    Door C (Circle) = closed ;
    Door D (Triangle) = closed ;
    Door E (Circle) = closed ;
    Door F (Circle) = closed ;
    Door G (*ERROR*) = closed ;
    Door H (Square) = closed ;
    Door I (Circle) = closed ;
    Door J (Triangle) = closed ;
}
void Interaction()
{
    enable Collect Keys = false ;
    enable Unlock Doors = false ;
    enable Lock Doors &
      Reclaim Keys = false ;
}
void Custom()
{
    //current value: X.X
    if( >select< ? X.X)
    {
        //current status: Active/Inactive
        enable >select< >select< ;
    }
}
```

Use the 'W' key to move the ball forward!

Use the 'S' key to move backward, and the 'A' and 'D' keys to move sideways.

The 'E' key on your keyboard is used for interaction with objects in the game.

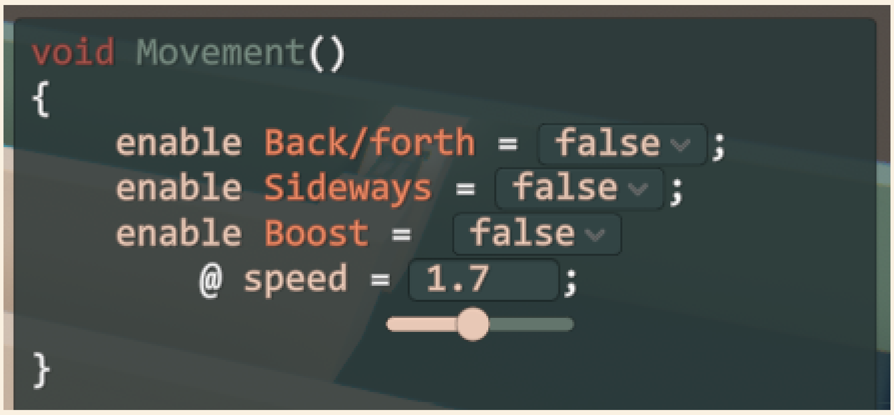
The 'Shift' key on the left side of your board toggles a special movement function: you can boost to speed up your ball!

Experimental Prototype Scene
Global Effects Handler
EventSystem
Directional Light
Main Camera
Canvas
Developer Panel
Movement
Object
Interaction
Custom
Suggestions Panel
text template
Keyboard E
Keyboard W
Keyboard ASD Group
Keyboard LShift
Player Sphere
Key Square
Key Circle
Key Circle
Key Triangle
Key Circle

font hex codes
Prefabs
Models
Created Wall
Door Circle
Door Square
Door Triangle
Key Capsule
Key Circle
Key Square
Key Triangle
Scenes
Scripts
BrokenDoorBehaviour
CameraFollow
DevCustomUIBehaviour
DeveloperUIBehaviour
DevObjectUIBehaviour
DevVariablesUIBehaviour
DoorBehaviour
KeyBehaviour
MeshDestructionBehaviour
PlayerBehaviour
ScrambleTextBehaviour
SelfDestructionBehaviour

All user interface elements as seen in the Unity Editor. ^

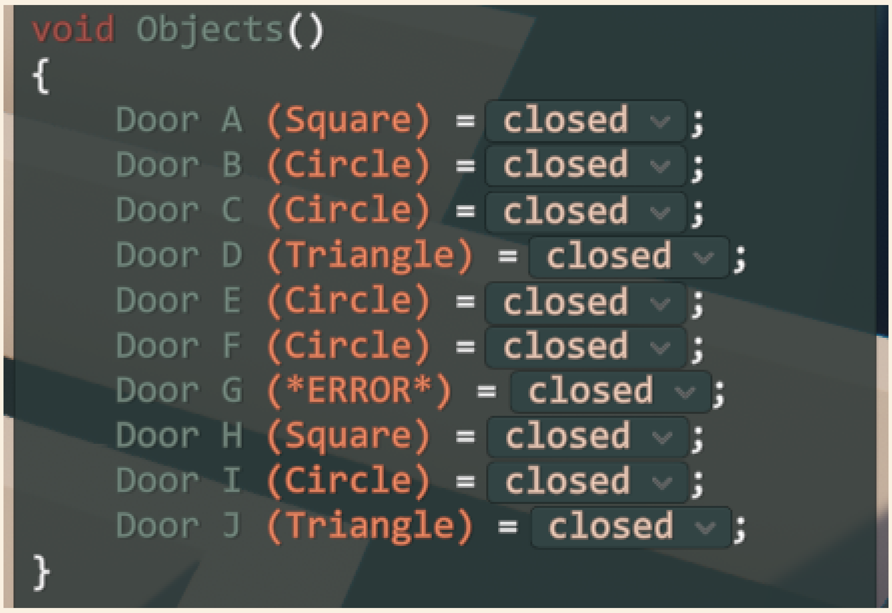
### 3.3.2.1) Developer UI: Movement Block



```
void Movement()
{
    enable Back/forth = false ;
    enable Sideways = false ;
    enable Boost =  false
        @ speed = 1.7 ;

}
```

The movement ‘block’ of the developer user interface has three boolean items and one adjustable float.
The Back/forth boolean enables or disables the player ball’s forward and backwards movement capabilities.
The Sideways boolean enables or disables the player ball’s left and right movement capabilities.
The Boost boolean enables or disables the ‘boost’ functionality of the player.
The Boost @ speed input field and slider determine the speed multiplier that is applied to the player's movement if the boost functionality is used.

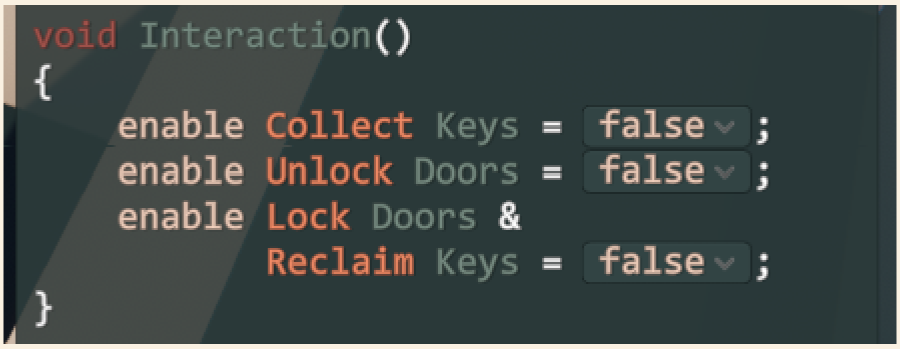
### 3.3.2.2) Developer UI: Objects Block



```
void Objects()
{
    Door A (Square) = closed ;
    Door B (Circle) = closed ;
    Door C (Circle) = closed ;
    Door D (Triangle) = closed ;
    Door E (Circle) = closed ;
    Door F (Circle) = closed ;
    Door G (*ERROR*) = closed ;
    Door H (Square) = closed ;
    Door I (Circle) = closed ;
    Door J (Triangle) = closed ;
}
```

There are nine functional doors in the scene and one broken door. They can all be opened and closed using this developer user interface block.
Each door has its own boolean, which is directly linked to the actual door. If the boolean is set to closed, the door will be closed. Conversely, if the boolean is set to open, the door will open.

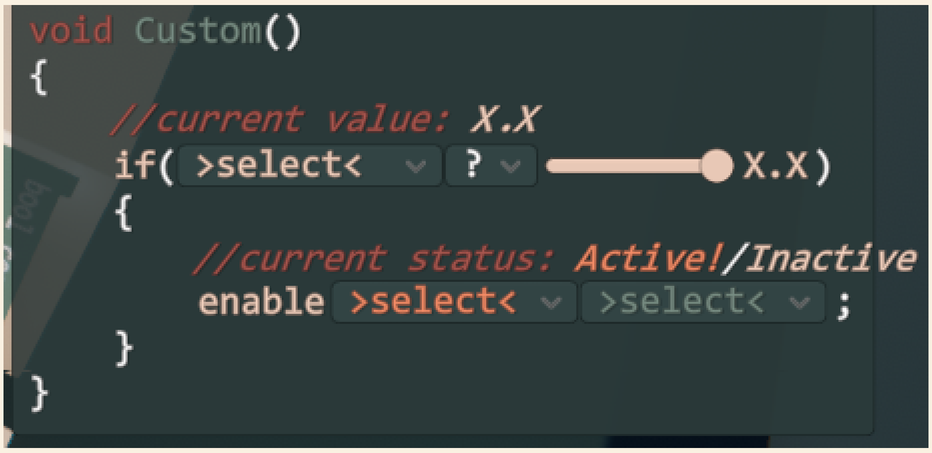
### 3.3.2.3) Developer UI: Interaction Block



```
void Interaction()
{
    enable Collect Keys = false ;
    enable Unlock Doors = false ;
    enable Lock Doors &
           Reclaim Keys = false ;
}
```

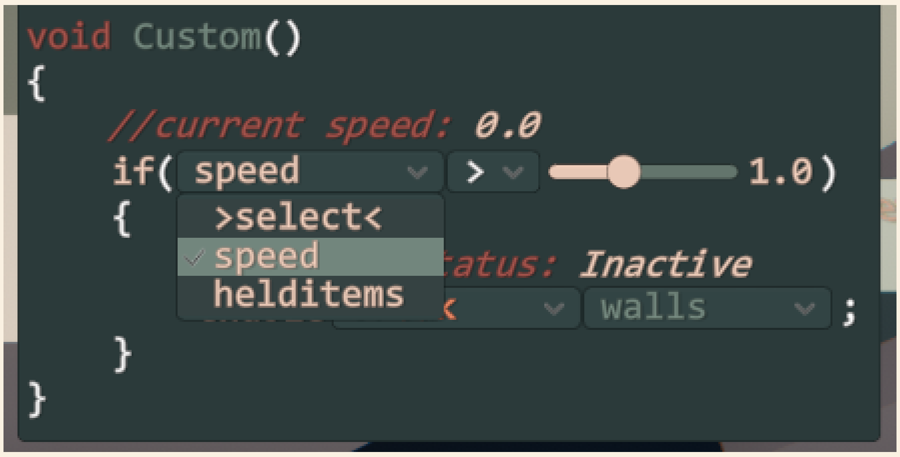
The Interactions block directly changes whether or not certain interactions in the game are allowed or not.
If the Collect Keys boolean is set to false, then the player will move straight through keys on the floor, as if they are already holding a key. The player cannot pick up and collect keys.
If the Unlock Doors boolean is set to false, the doors will not light up and be available for opening at all. If the player brings a key close to the door and pressing the interact button ‘E’ on their keyboard, then the key will merely be dropped in front of the player, on the ground.
If the Lock Doors & Reclaim Keys boolean is disabled, the player will not be able to close doors after they have been opened. If the boolean *is* enabled however, then the player can close doors again after they have been opened, and, importantly, retrieve the key that was stored inside the door. The key can then be used to re-open the door, or carried to another nearby door with the same key icon and opened instead.

### 3.3.2.4) Developer UI: Custom Block



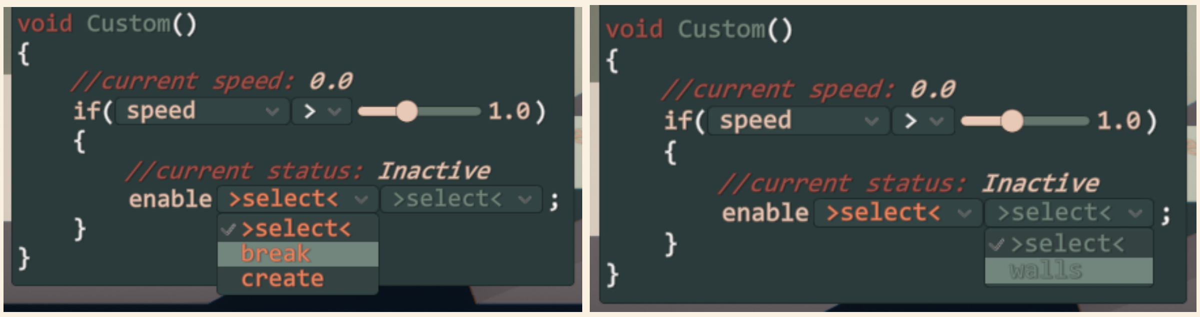
```
void Custom()
{
    //current value: X.X
    if( >select< ? X.X)
    {
        //current status: Active!/Inactive
        enable >select< >select< ;
    }
}
```

The Custom block of the developer user interface has the most diverse interactions with the game out of the four developer UI blocks.
The first line contains the conditional part of the custom if-statement.
As of version 1.0 of the minigame, the conditional has two options for the first part:



```
void Custom()
{
    //current speed: 0.0
    if( speed ⌄  > ⌄  1.0)
    {   >select<
        speed       tatus: Inactive
        helditems  <  ⌄   walls ⌄  ;
    }
}
```

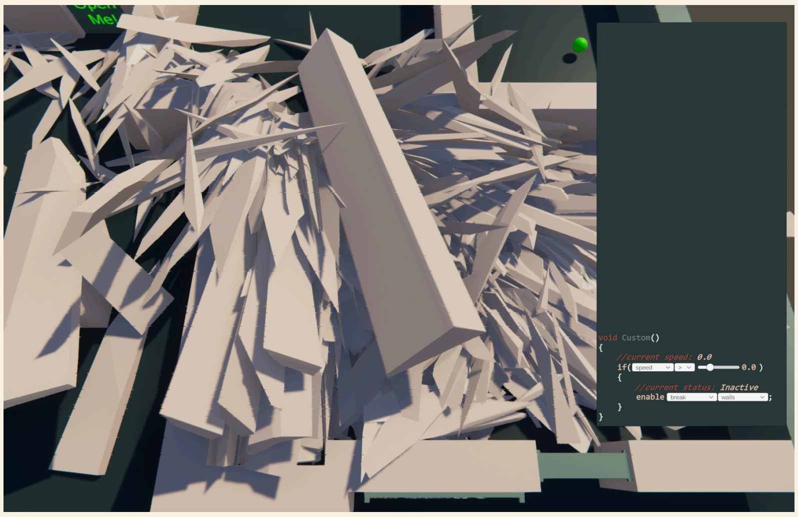
Either we check if the speed of the player is greater or less than a set number on the slider, or we check if the amount of items the player is holding is greater or less than the set number on the slider (where the amount of items the player can possibly hold is only zero or one).
In the second part of the custom if-statement, the user chooses what will happen, should the above condition be met. The two options available to be enabled are to either break or create, and *what* should be created or broken is chosen in the last dropdown: currently, the only things that can be created or destroyed are walls.



```
void Custom()
{
    //current speed: 0.0
    if( speed ⌄  > ⌄  1.0)
    {
        //current status: Inactive
        enable >select< ⌄  >select< ⌄ ;
    }       >select<
}           break
            create
```

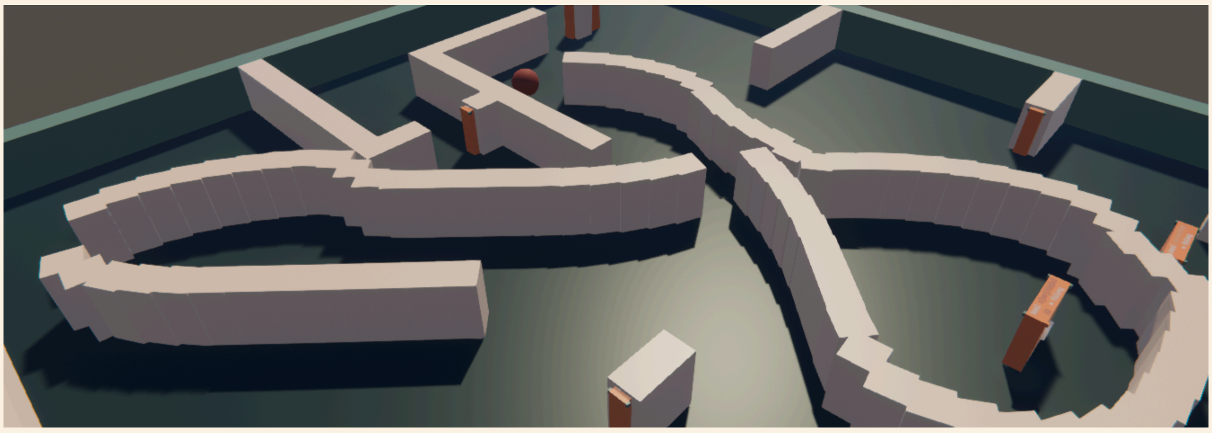
```
void Custom()
{
    //current speed: 0.0
    if( speed ⌄  > ⌄  1.0)
    {
        //current status: Inactive
        enable >select< ⌄  >select< ⌄ ;
    }                      >select<
}                          walls
```

If the player is able to break walls, then as soon as the player touches a wall, it will shatter into multiple pieces. When this happens, the colliders of the pieces are disabled to the player, and a short-term self-destruct function is enabled which causes the wall pieces to be completely destroyed from the game (to avoid overloading RAM or CPU usage) after a short delay. Smashing through the walls of the game leaves behind a sort of ‘path of destruction’, as shown here:



```
void Custom()
{
    //current speed: 0.0
    if( speed ⌄  > ⌄  0.0 )
    {
        //current status: Inactive
        enable break ⌄  walls ⌄ ;
    }
}
```

When creating walls, they are spawned at the player's location, facing the same way as the player. The walls create a sort of ‘path of creation’, as shown here:

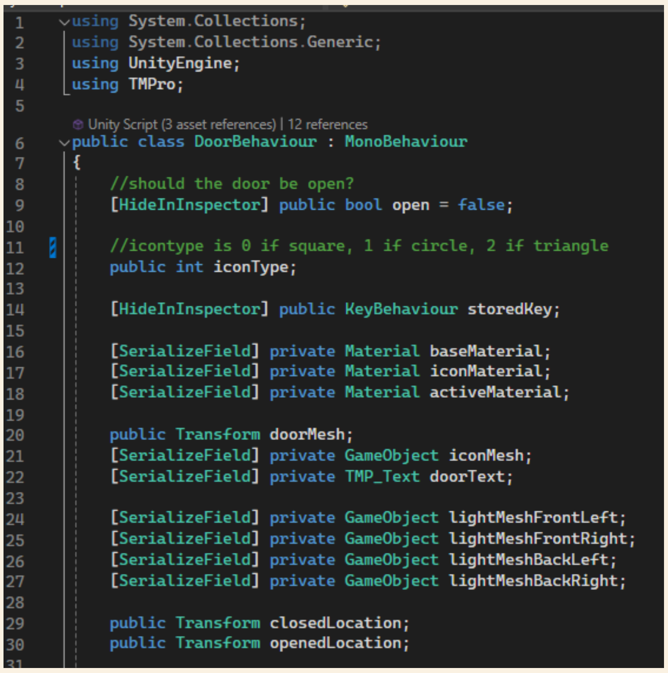


---

## **3.4)** Mechanics

*This section of the report covers all game mechanics, in reference to their C# scripts. The sheer amount of references, callbacks, and other connectivity in this linked web of scripts disallows a more fluid explanation structure as the rest of this report has. While this section will have enough structure for it to be easy to follow, it will not be structured with section numbers like the rest of the report.*
*All scripts were 100% created exclusively with the author’s programming ability, unless explicitly stated otherwise.*

**Doors**
The doors contain the DoorBehaviour C# script.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

Unity Script (3 asset references) | 12 references
public class DoorBehaviour : MonoBehaviour
{
    //should the door be open?
    [HideInInspector] public bool open = false;

    //icontype is 0 if square, 1 if circle, 2 if triangle
    public int iconType;

    [HideInInspector] public KeyBehaviour storedKey;

    [SerializeField] private Material baseMaterial;
    [SerializeField] private Material iconMaterial;
    [SerializeField] private Material activeMaterial;

    public Transform doorMesh;
    [SerializeField] private GameObject iconMesh;
    [SerializeField] private TMP_Text doorText;

    [SerializeField] private GameObject lightMeshFrontLeft;
    [SerializeField] private GameObject lightMeshFrontRight;
    [SerializeField] private GameObject lightMeshBackLeft;
    [SerializeField] private GameObject lightMeshBackRight;

    public Transform closedLocation;
    public Transform openedLocation;
```

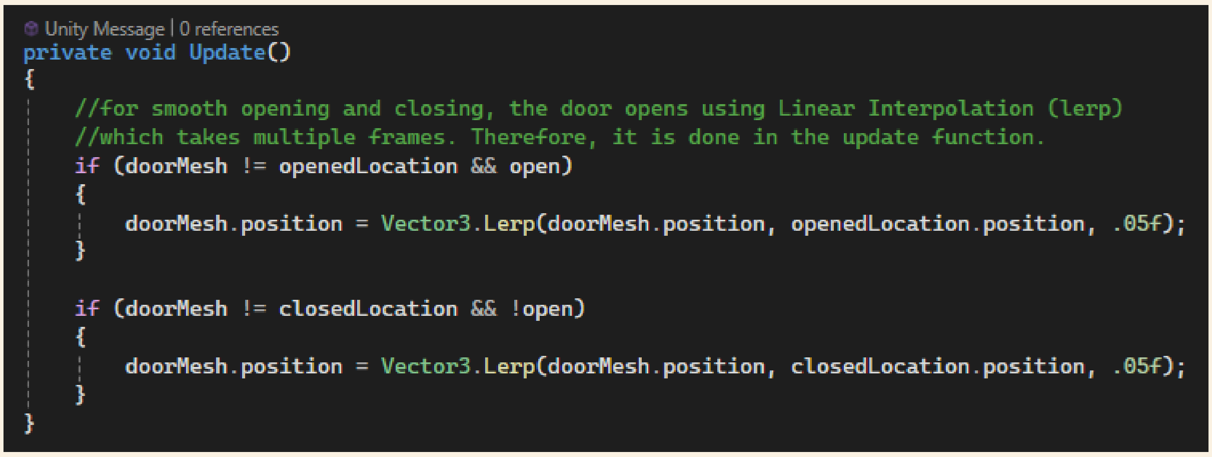
Main variables are the *open* boolean which depicts to other scripts whether or not this door is currently open.

The *iconType* integer, just like its key counterpart, depicts to other scripts whether or not this is a square, circle or triangle door.
The *storedKey* variable is used to determine whether or not a key is stored inside the door.
The material variables are used for colouring the doors when they are closed, open, or openable.

The doorMesh variable refers to the mesh of the door itself. This is the transform that has its location adjusted when the door is opened or closed. The closedLocation and openedLocation transform variables have empty GameObjects assigned to them that are located on the open and close locations in the scene. The doorMesh moves to those locations through linear interpolation to create a fast but smooth opening and closing animation.

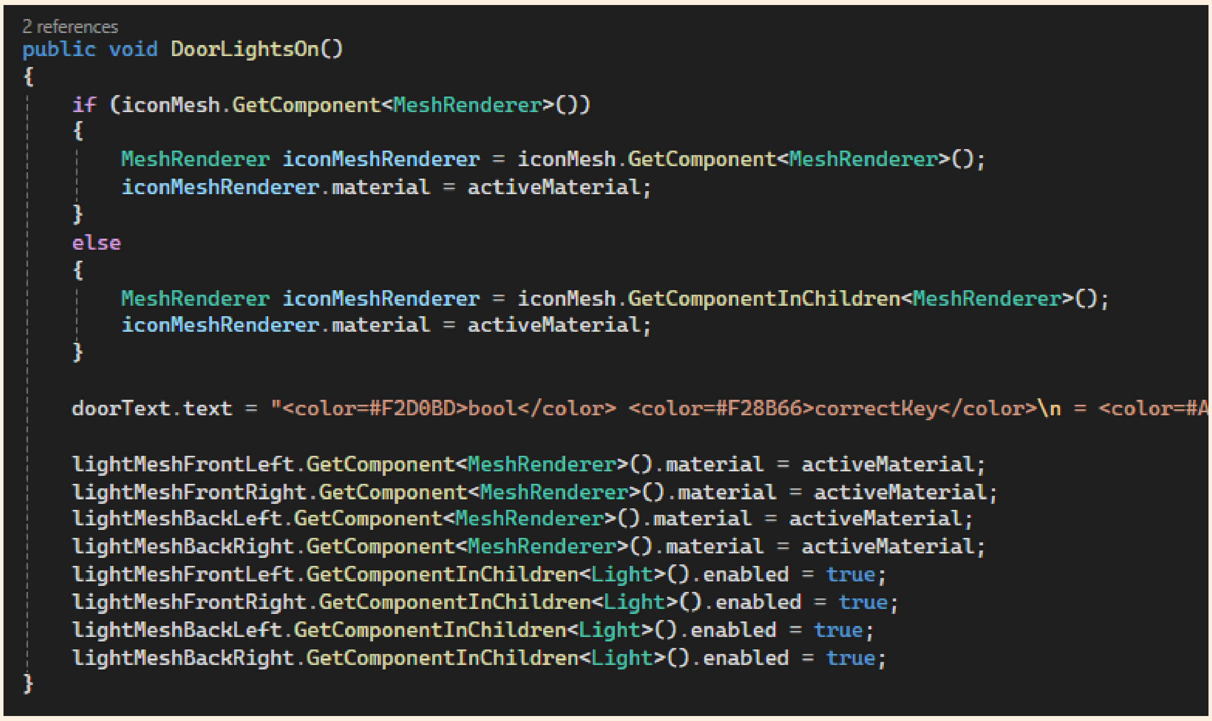
The lightMesh GameObjects refer to the four small lights on the corners of the top of the doors.

The DoorBehaviour script has multiple methods.
In the update function, the handling of the door opening and closing animation is done:



```
Unity Message | 0 references
private void Update()
{
    //for smooth opening and closing, the door opens using Linear Interpolation (lerp)
    //which takes multiple frames. Therefore, it is done in the update function.
    if (doorMesh != openedLocation && open)
    {
        doorMesh.position = Vector3.Lerp(doorMesh.position, openedLocation.position, .05f);
    }

    if (doorMesh != closedLocation && !open)
    {
        doorMesh.position = Vector3.Lerp(doorMesh.position, closedLocation.position, .05f);
    }
}
```

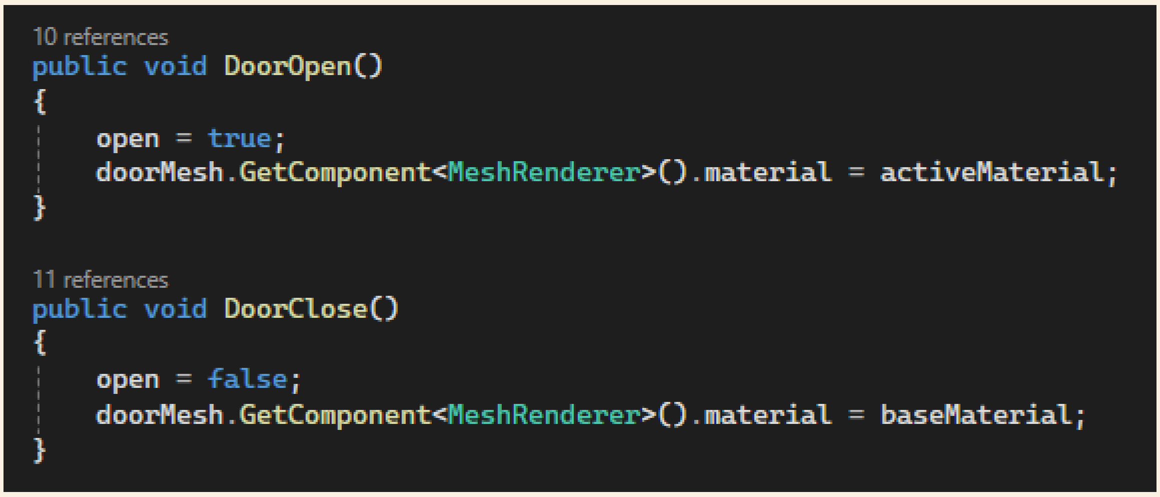
The DoorLightsOn() method is called when the door can be opened by a player holding the correct key:



```
2 references
public void DoorLightsOn()
{
    if (iconMesh.GetComponent<MeshRenderer>())
    {
        MeshRenderer iconMeshRenderer = iconMesh.GetComponent<MeshRenderer>();
        iconMeshRenderer.material = activeMaterial;
    }
    else
    {
        MeshRenderer iconMeshRenderer = iconMesh.GetComponentInChildren<MeshRenderer>();
        iconMeshRenderer.material = activeMaterial;
    }

    doorText.text = "<color=#F2D0BD>bool</color> <color=#F28B66>correctKey</color>\n = <color=#A

    lightMeshFrontLeft.GetComponent<MeshRenderer>().material = activeMaterial;
    lightMeshFrontRight.GetComponent<MeshRenderer>().material = activeMaterial;
    lightMeshBackLeft.GetComponent<MeshRenderer>().material = activeMaterial;
    lightMeshBackRight.GetComponent<MeshRenderer>().material = activeMaterial;
    lightMeshFrontLeft.GetComponentInChildren<Light>().enabled = true;
    lightMeshFrontRight.GetComponentInChildren<Light>().enabled = true;
    lightMeshBackLeft.GetComponentInChildren<Light>().enabled = true;
    lightMeshBackRight.GetComponentInChildren<Light>().enabled = true;
}
```

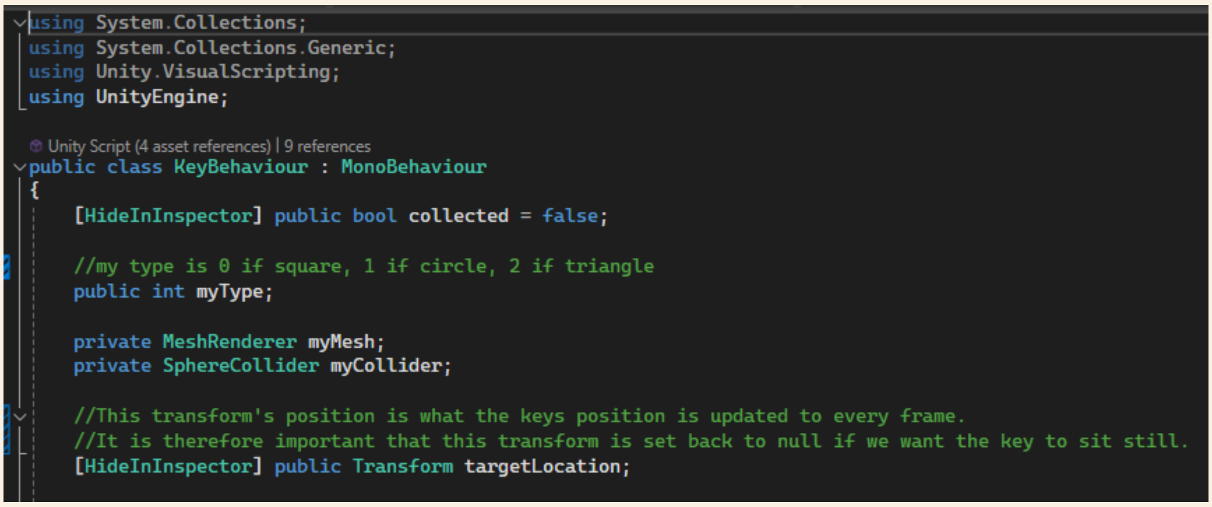
The DoorLightsOff() method is identical, reverting all ‘active’ states back to ‘idle’ ones.

Other than that, the DoorBehaviour script contains the important yet simple DoorOpen() and DoorClose() methods:



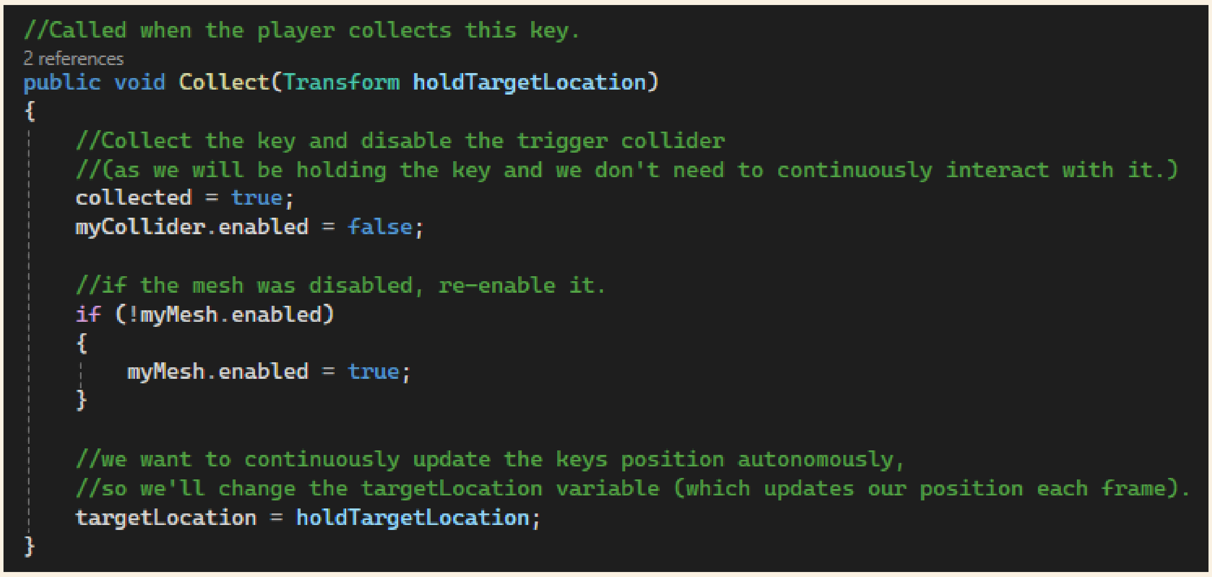
```
10 references
public void DoorOpen()
{
    open = true;
    doorMesh.GetComponent<MeshRenderer>().material = activeMaterial;
}

11 references
public void DoorClose()
{
    open = false;
    doorMesh.GetComponent<MeshRenderer>().material = baseMaterial;
}
```

**Keys**
They all have a KeyBehaviour C# script attached to them.



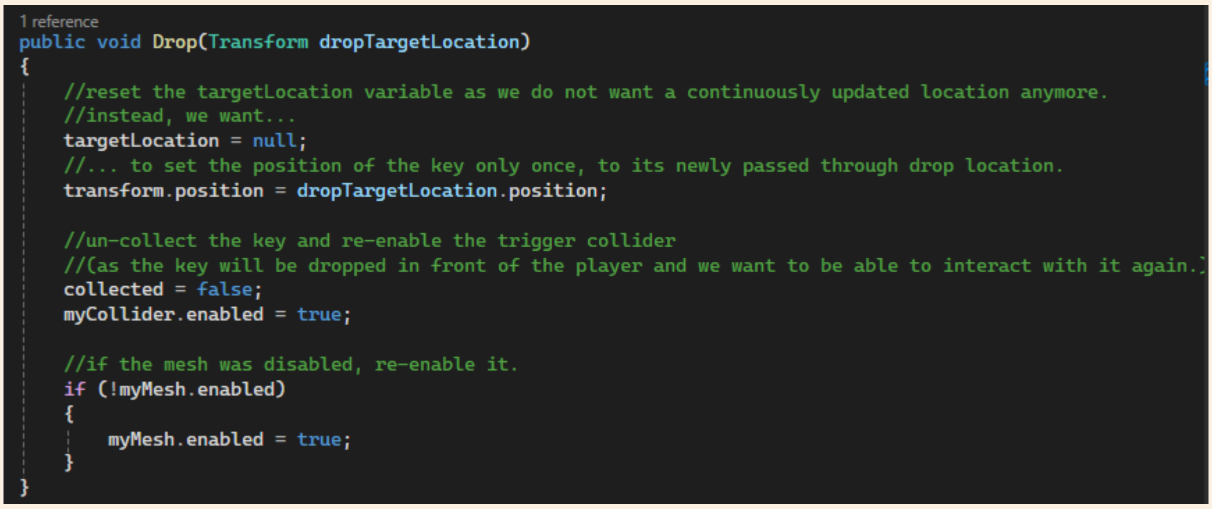
```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

Unity Script (4 asset references) | 9 references
public class KeyBehaviour : MonoBehaviour
{
    [HideInInspector] public bool collected = false;

    //my type is 0 if square, 1 if circle, 2 if triangle
    public int myType;

    private MeshRenderer myMesh;
    private SphereCollider myCollider;

    //This transform's position is what the keys position is updated to every frame.
    //It is therefore important that this transform is set back to null if we want the key to sit still.
    [HideInInspector] public Transform targetLocation;
```

The script contains some variables and multiple methods. They are the Collect(), Drop() and Store() methods, they are called when the player *collects* the key:



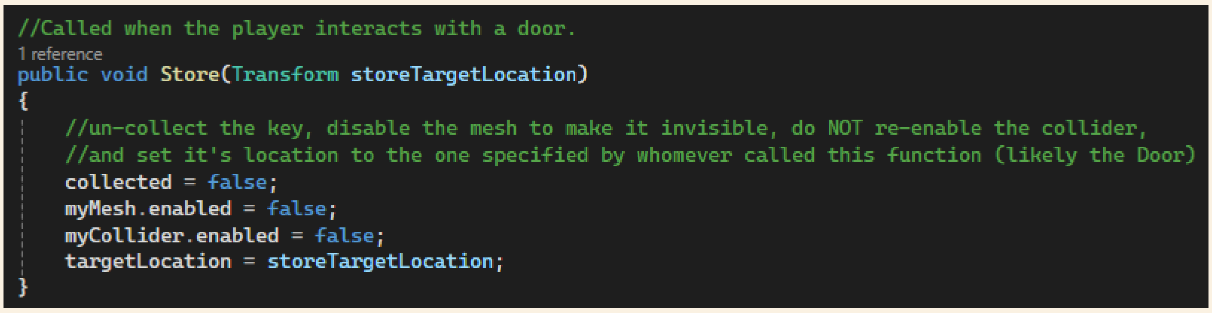
```
//Called when the player collects this key.
2 references
public void Collect(Transform holdTargetLocation)
{
    //Collect the key and disable the trigger collider
    //(as we will be holding the key and we don't need to continuously interact with it.)
    collected = true;
    myCollider.enabled = false;

    //if the mesh was disabled, re-enable it.
    if (!myMesh.enabled)
    {
        myMesh.enabled = true;
    }

    //we want to continuously update the keys position autonomously,
    //so we'll change the targetLocation variable (which updates our position each frame).
    targetLocation = holdTargetLocation;
}
```

When the player *drops* the key:



```
1 reference
public void Drop(Transform dropTargetLocation)
{
    //reset the targetLocation variable as we do not want a continuously updated location anymore.
    //instead, we want...
    targetLocation = null;
    //... to set the position of the key only once, to its newly passed through drop location.
    transform.position = dropTargetLocation.position;

    //un-collect the key and re-enable the trigger collider
    //(as the key will be dropped in front of the player and we want to be able to interact with it again.)
    collected = false;
    myCollider.enabled = true;

    //if the mesh was disabled, re-enable it.
    if (!myMesh.enabled)
    {
        myMesh.enabled = true;
    }
}
```

And when the player opens a door and needs to *store* the key inside that door:



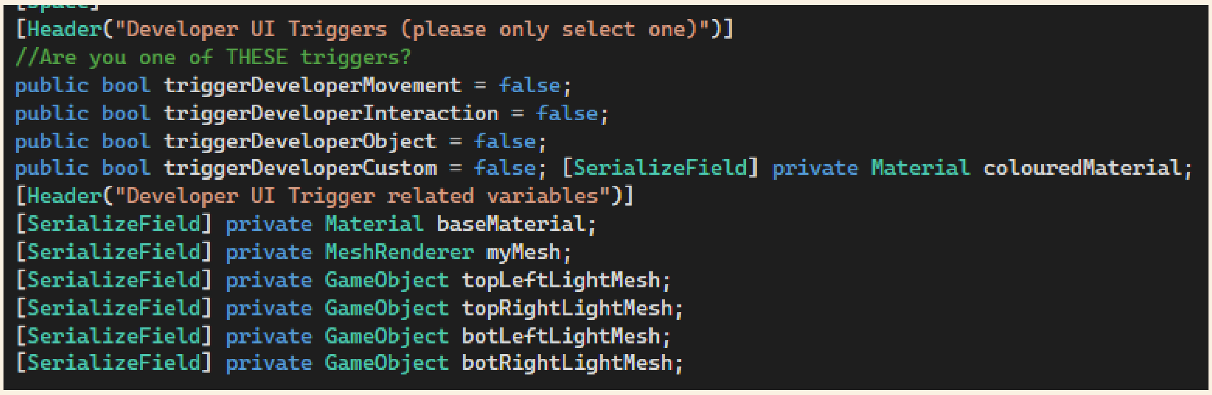
```
//Called when the player interacts with a door.
1 reference
public void Store(Transform storeTargetLocation)
{
    //un-collect the key, disable the mesh to make it invisible, do NOT re-enable the collider,
    //and set it's location to the one specified by whomever called this function (likely the Door)
    collected = false;
    myMesh.enabled = false;
    myCollider.enabled = false;
    targetLocation = storeTargetLocation;
}
```

Additionally, in the Update() function of the Key, the key's transform position is updated to the hold location (near the player's ball) if that variable is assigned.

The 'type' of key is visualised by the different meshes, and that is represented by a simple integer in the script. The *myType* integer is public, and it is what other scripts read to deduce whether the key in question is a square, circle or triangle.

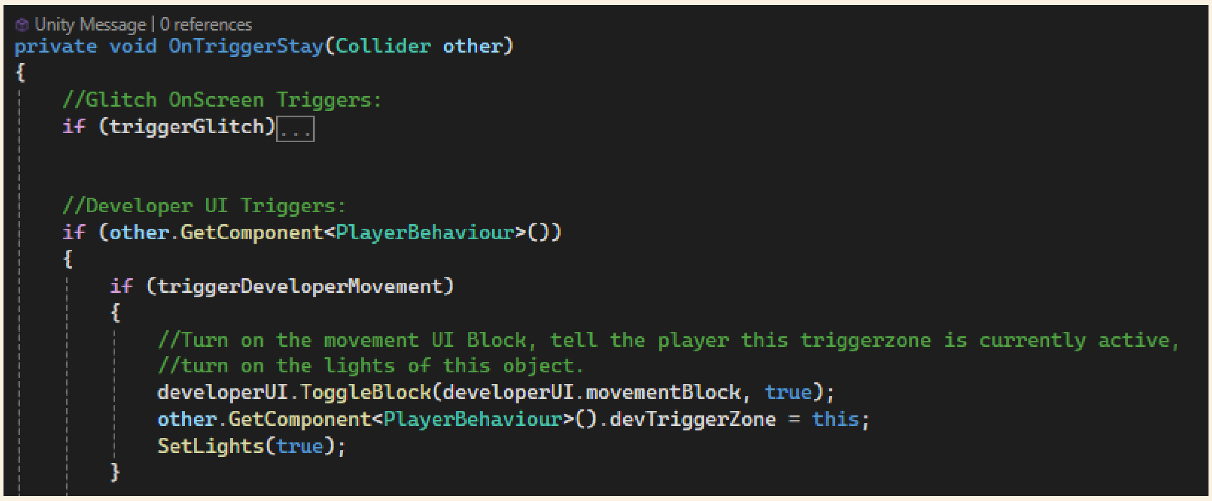
**Screens**

The screens use the TriggerZoneBehaviour scripts. The TriggerZoneBehaviour script is one of the larger scripts made for this game, and it will be expanded on fully later. The relevant variables of that script for the Glitchy Screens is:



```
[Header("Developer UI Triggers (please only select one)")]
//Are you one of THESE triggers?
public bool triggerDeveloperMovement = false;
public bool triggerDeveloperInteraction = false;
public bool triggerDeveloperObject = false;
public bool triggerDeveloperCustom = false; [SerializeField] private Material colouredMaterial;
[Header("Developer UI Trigger related variables")]
[SerializeField] private Material baseMaterial;
[SerializeField] private MeshRenderer myMesh;
[SerializeField] private GameObject topLeftLightMesh;
[SerializeField] private GameObject topRightLightMesh;
[SerializeField] private GameObject botLeftLightMesh;
[SerializeField] private GameObject botRightLightMesh;
```

One of the four booleans is checked on to signify which Glitchy Screen the one in question is. The Mesh related variables are used to show that the screen is 'active' when the player goes near it: the four lights will turn on and the mesh's material will turn orange.

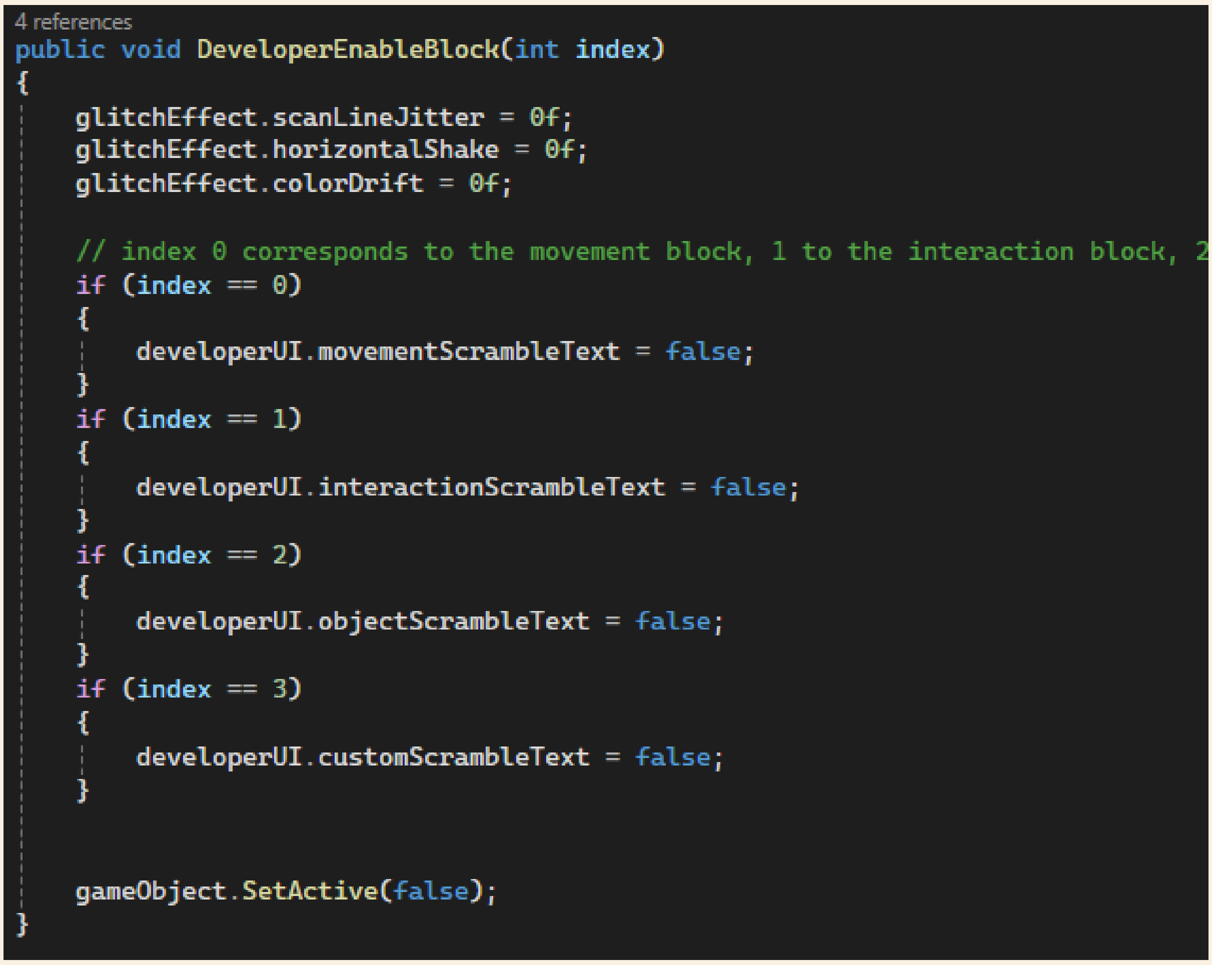


```
Unity Message | 0 references
private void OnTriggerStay(Collider other)
{
    //Glitch OnScreen Triggers:
    if (triggerGlitch)...


    //Developer UI Triggers:
    if (other.GetComponent<PlayerBehaviour>())
    {
        if (triggerDeveloperMovement)
        {
            //Turn on the movement UI Block, tell the player this triggerzone is currently active,
            //turn on the lights of this object.
            developerUI.ToggleBlock(developerUI.movementBlock, true);
            other.GetComponent<PlayerBehaviour>().devTriggerZone = this;
            SetLights(true);
        }
```

If the object that enters the trigger of the Glitchy Screen is a player, then the screen becomes visible on the user interface of the player.
Additionally, the lights turn on and the player can interact with the screen to unscramble the text and make the relevant developer functions available.

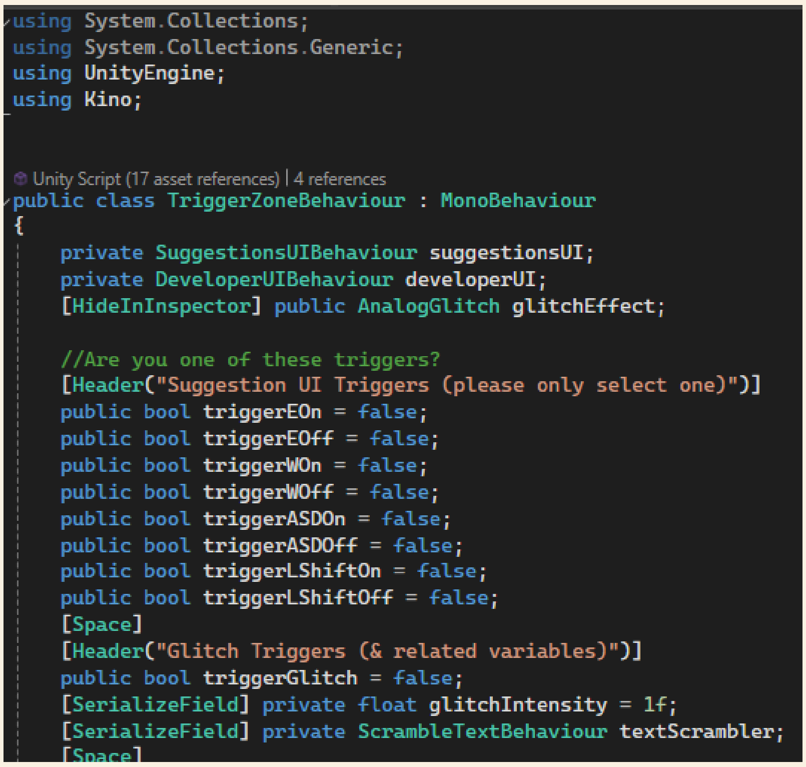
The script also contains a similar block of code in the OnTriggerExit method, which turns off the lights and disables the UI visibility of the relevant block.

If the screen is interacted with, it disables itself (visually), disables any latent glitchiness on the camera, and unscrambles the text on the UI.



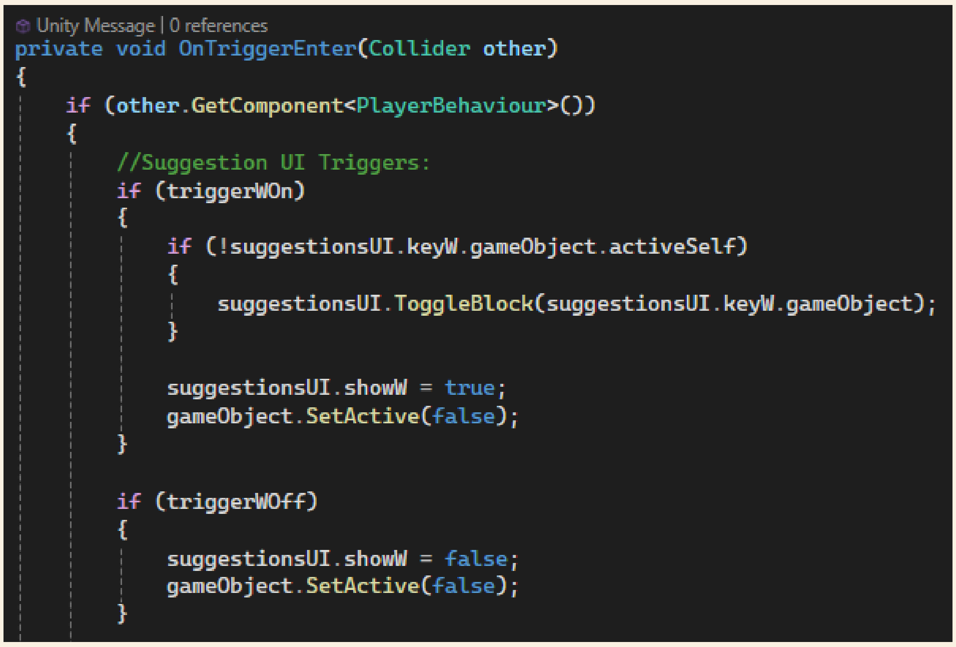
```
4 references
public void DeveloperEnableBlock(int index)
{
    glitchEffect.scanLineJitter = 0f;
    glitchEffect.horizontalShake = 0f;
    glitchEffect.colorDrift = 0f;

    // index 0 corresponds to the movement block, 1 to the interaction block, 2
    if (index == 0)
    {
        developerUI.movementScrambleText = false;
    }
    if (index == 1)
    {
        developerUI.interactionScrambleText = false;
    }
    if (index == 2)
    {
        developerUI.objectScrambleText = false;
    }
    if (index == 3)
    {
        developerUI.customScrambleText = false;
    }


    gameObject.SetActive(false);
}
```

### Trigger Zones



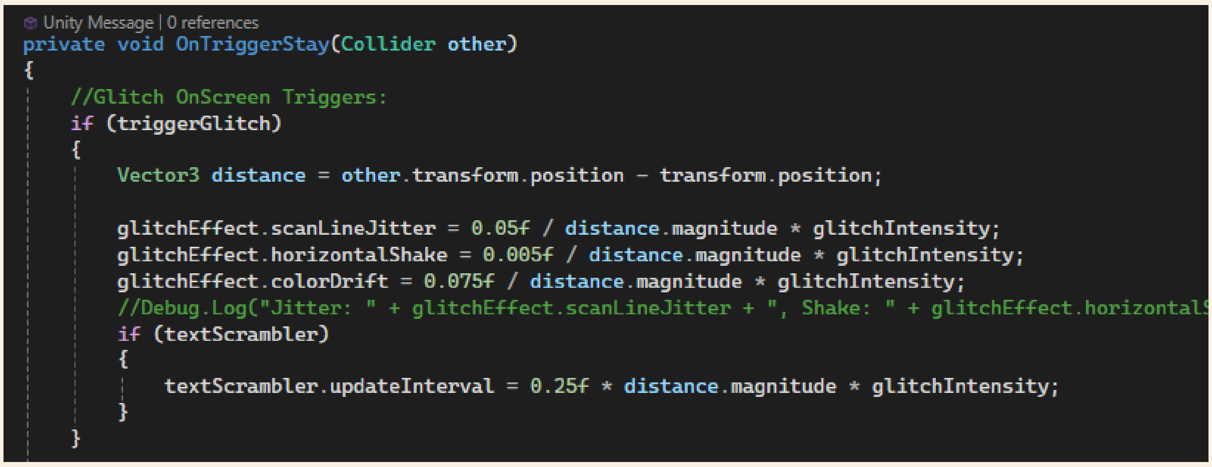
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Kino;


Unity Script (17 asset references) | 4 references
public class TriggerZoneBehaviour : MonoBehaviour
{
    private SuggestionsUIBehaviour suggestionsUI;
    private DeveloperUIBehaviour developerUI;
    [HideInInspector] public AnalogGlitch glitchEffect;

    //Are you one of these triggers?
    [Header("Suggestion UI Triggers (please only select one)")]
    public bool triggerEOn = false;
    public bool triggerEOff = false;
    public bool triggerWOn = false;
    public bool triggerWOff = false;
    public bool triggerASDOn = false;
    public bool triggerASDOff = false;
    public bool triggerLShiftOn = false;
    public bool triggerLShiftOff = false;
    [Space]
    [Header("Glitch Triggers (& related variables)")]
    public bool triggerGlitch = false;
    [SerializeField] private float glitchIntensity = 1f;
    [SerializeField] private ScrambleTextBehaviour textScrambler;
    [Space]
```

The Suggestion UI Trigger Zones have very simple functionality: When their trigger is entered by a player, the relevant suggestion UI element is either enabled of disabled, toggling their visibility.



```
Unity Message | 0 references
private void OnTriggerEnter(Collider other)
{
    if (other.GetComponent<PlayerBehaviour>())
    {
        //Suggestion UI Triggers:
        if (triggerWOn)
        {
            if (!suggestionsUI.keyW.gameObject.activeSelf)
            {
                suggestionsUI.ToggleBlock(suggestionsUI.keyW.gameObject);
            }

            suggestionsUI.showW = true;
            gameObject.SetActive(false);
        }

        if (triggerWOff)
        {
            suggestionsUI.showW = false;
            gameObject.SetActive(false);
        }
```

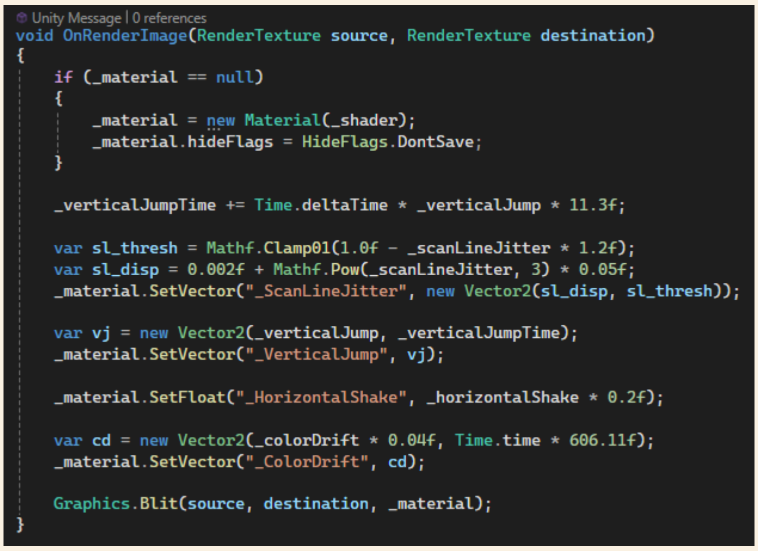
Glitch Trigger Zones also have very simple functionality. If a player enters its trigger, then the glitch intensity of the camera is increased in proportion to how close the player is to the centre of the trigger zone.



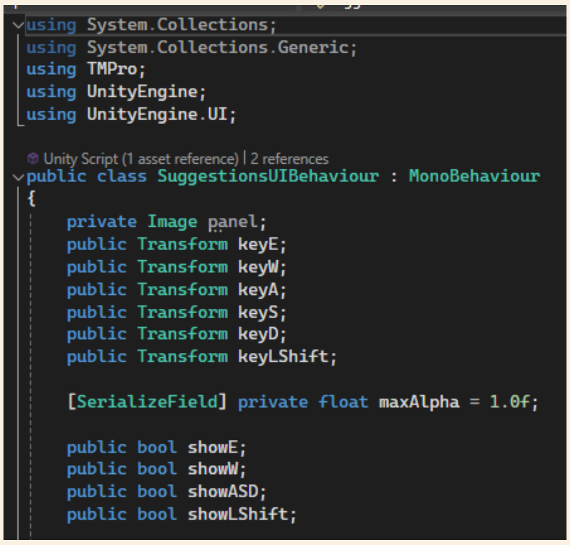
```
Unity Message | 0 references
private void OnTriggerStay(Collider other)
{
    //Glitch OnScreen Triggers:
    if (triggerGlitch)
    {
        Vector3 distance = other.transform.position - transform.position;

        glitchEffect.scanLineJitter = 0.05f / distance.magnitude * glitchIntensity;
        glitchEffect.horizontalShake = 0.005f / distance.magnitude * glitchIntensity;
        glitchEffect.colorDrift = 0.075f / distance.magnitude * glitchIntensity;
        //Debug.Log("Jitter: " + glitchEffect.scanLineJitter + ", Shake: " + glitchEffect.horizontalS
        if (textScrambler)
        {
            textScrambler.updateInterval = 0.25f * distance.magnitude * glitchIntensity;
        }
    }
```

The relevant glitch effect is found in the AnalogGlitch script, one of the few scripts used in this game that are not original to the project.

The AnalogGlitch script by Kino adjusts the graphics that are put out by a camera in the Unity game engine directly. There are a few options, but the ones used for this game are the Scan Line Jitter, the Horizontal Shake and the Color Drift.



```
Unity Message | 0 references
void OnRenderImage(RenderTexture source, RenderTexture destination)
{
    if (_material == null)
    {
        _material = new Material(_shader);
        _material.hideFlags = HideFlags.DontSave;
    }

    _verticalJumpTime += Time.deltaTime * _verticalJump * 11.3f;

    var sl_thresh = Mathf.Clamp01(1.0f - _scanLineJitter * 1.2f);
    var sl_disp = 0.002f + Mathf.Pow(_scanLineJitter, 3) * 0.05f;
    _material.SetVector("_ScanLineJitter", new Vector2(sl_disp, sl_thresh));

    var vj = new Vector2(_verticalJump, _verticalJumpTime);
    _material.SetVector("_VerticalJump", vj);

    _material.SetFloat("_HorizontalShake", _horizontalShake * 0.2f);

    var cd = new Vector2(_colorDrift * 0.04f, Time.time * 606.11f);
    _material.SetVector("_ColorDrift", cd);

    Graphics.Blit(source, destination, _material);
}
```

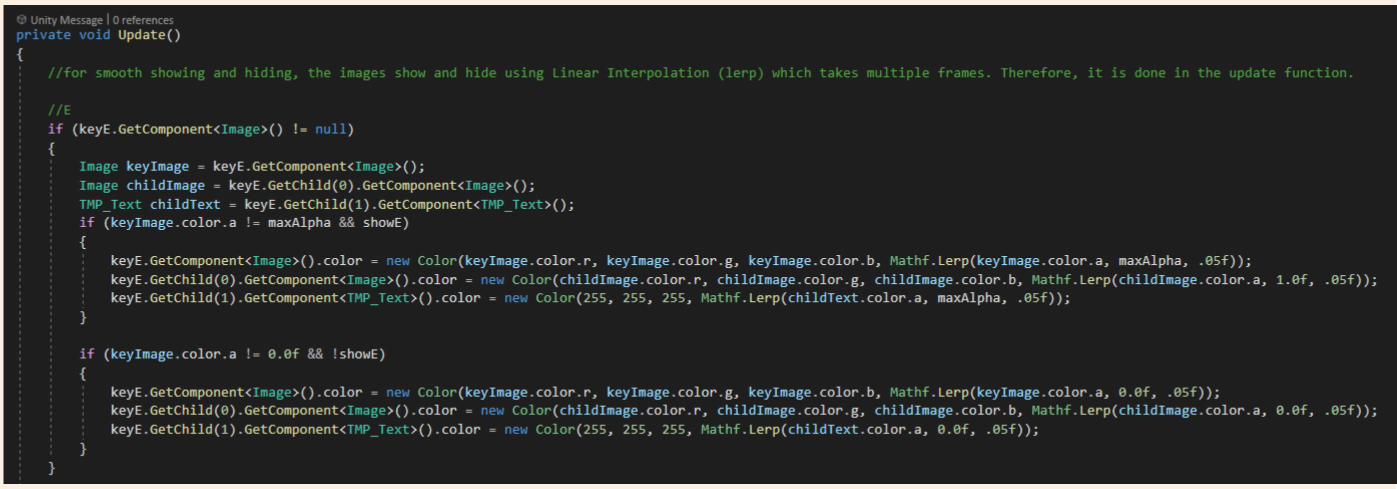


```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

Unity Script (1 asset reference) | 2 references
public class SuggestionsUIBehaviour : MonoBehaviour
{
    private Image panel;
    public Transform keyE;
    public Transform keyW;
    public Transform keyA;
    public Transform keyS;
    public Transform keyD;
    public Transform keyLShift;

    [SerializeField] private float maxAlpha = 1.0f;

    public bool showE;
    public bool showW;
    public bool showASD;
    public bool showLShift;
```

**Suggestion User Interface**

The SuggestionsUIBehaviour C# script is attached to the group object.

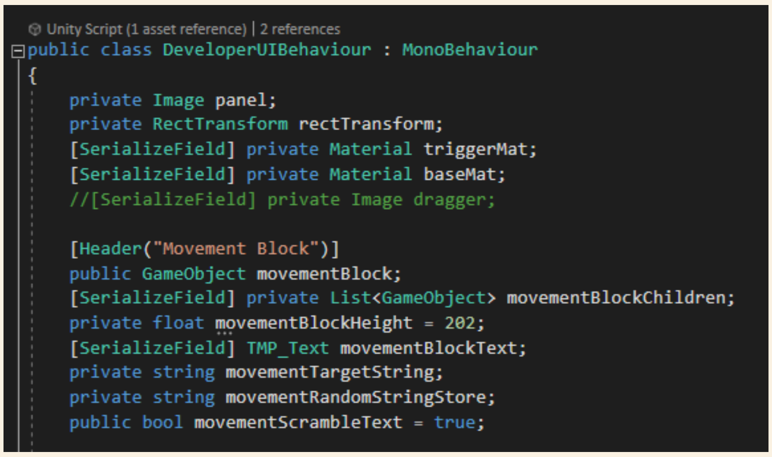
It handles the smooth enabling and disabling of the individual elements of the UI, using linear interpolation.



```
Unity Message | 0 references
private void Update()
{
    //for smooth showing and hiding, the images show and hide using Linear Interpolation (lerp) which takes multiple frames. Therefore, it is done in the update function.

    //E
    if (keyE.GetComponent<Image>() != null)
    {
        Image keyImage = keyE.GetComponent<Image>();
        Image childImage = keyE.GetChild(0).GetComponent<Image>();
        TMP_Text childText = keyE.GetChild(1).GetComponent<TMP_Text>();
        if (keyImage.color.a != maxAlpha && showE)
        {
            keyE.GetComponent<Image>().color = new Color(keyImage.color.r, keyImage.color.g, keyImage.color.b, Mathf.Lerp(keyImage.color.a, maxAlpha, .05f));
            keyE.GetChild(0).GetComponent<Image>().color = new Color(childImage.color.r, childImage.color.g, childImage.color.b, Mathf.Lerp(childImage.color.a, 1.0f, .05f));
            keyE.GetChild(1).GetComponent<TMP_Text>().color = new Color(255, 255, 255, Mathf.Lerp(childText.color.a, maxAlpha, .05f));
        }

        if (keyImage.color.a != 0.0f && !showE)
        {
            keyE.GetComponent<Image>().color = new Color(keyImage.color.r, keyImage.color.g, keyImage.color.b, Mathf.Lerp(keyImage.color.a, 0.0f, .05f));
            keyE.GetChild(0).GetComponent<Image>().color = new Color(childImage.color.r, childImage.color.g, childImage.color.b, Mathf.Lerp(childImage.color.a, 0.0f, .05f));
            keyE.GetChild(1).GetComponent<TMP_Text>().color = new Color(255, 255, 255, Mathf.Lerp(childText.color.a, 0.0f, .05f));
        }
    }
```

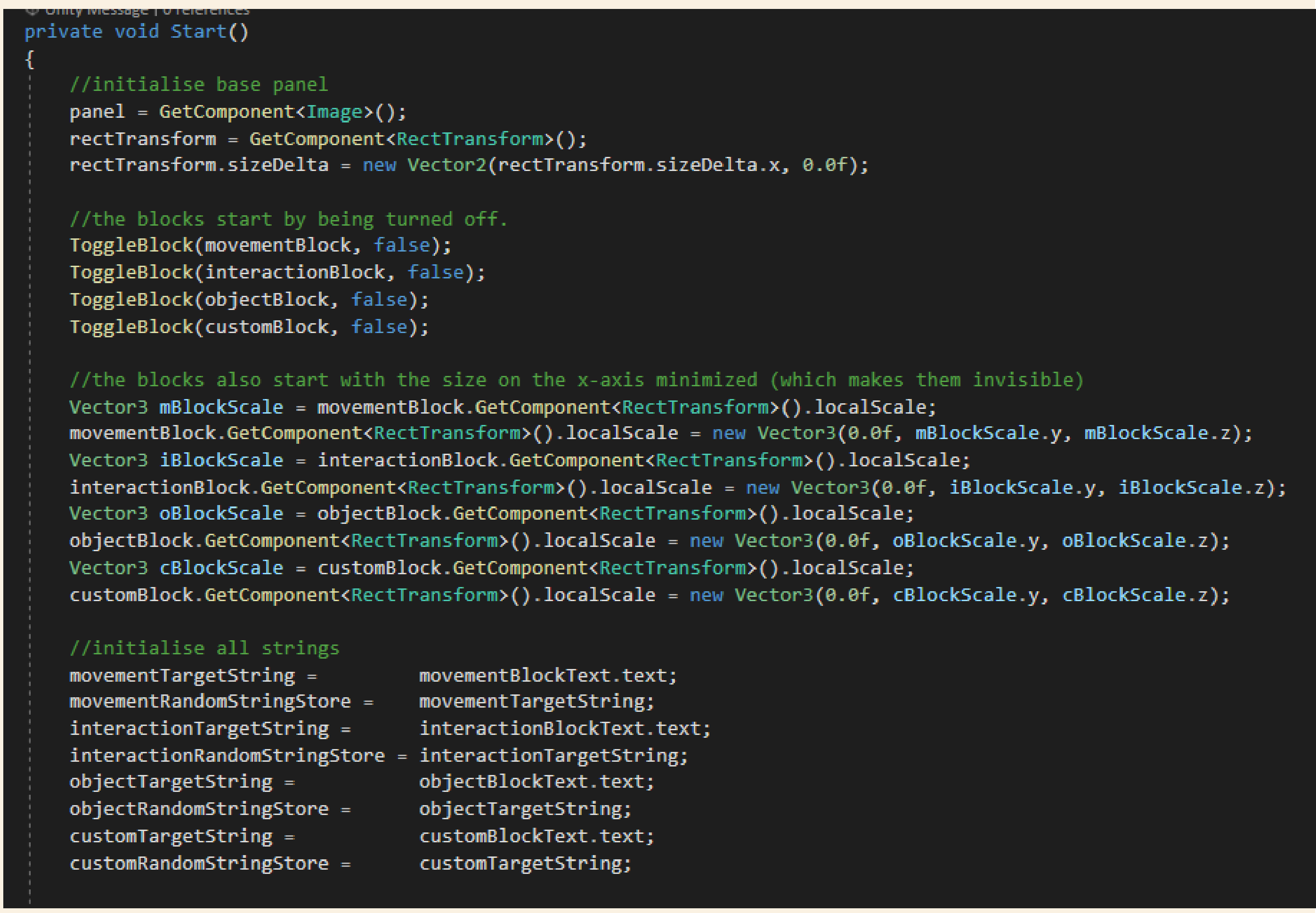
### Developer User Interface

The developer user interface uses multiple different C# scripts to regulate its behaviour.
On the group object, we find DeveloperUIBehvaiour



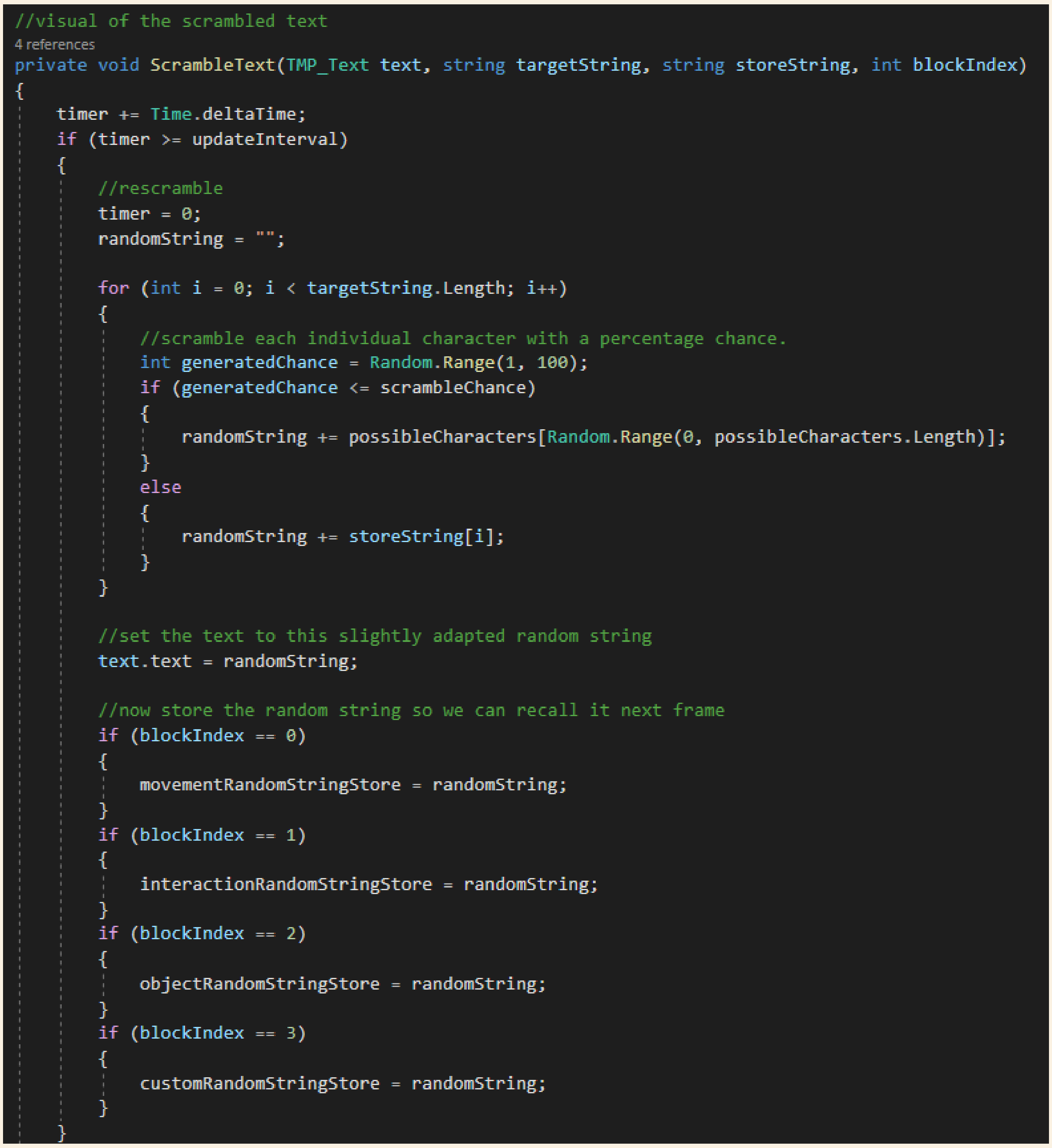
```
Unity Script (1 asset reference) | 2 references
public class DeveloperUIBehaviour : MonoBehaviour
{
    private Image panel;
    private RectTransform rectTransform;
    [SerializeField] private Material triggerMat;
    [SerializeField] private Material baseMat;
    //[SerializeField] private Image dragger;

    [Header("Movement Block")]
    public GameObject movementBlock;
    [SerializeField] private List<GameObject> movementBlockChildren;
    private float movementBlockHeight = 202;
    [SerializeField] TMP_Text movementBlockText;
    private string movementTargetString;
    private string movementRandomStringStore;
    public bool movementScrambleText = true;
```

It contains these blocks of variables that refer to the elements in the pseudocode.



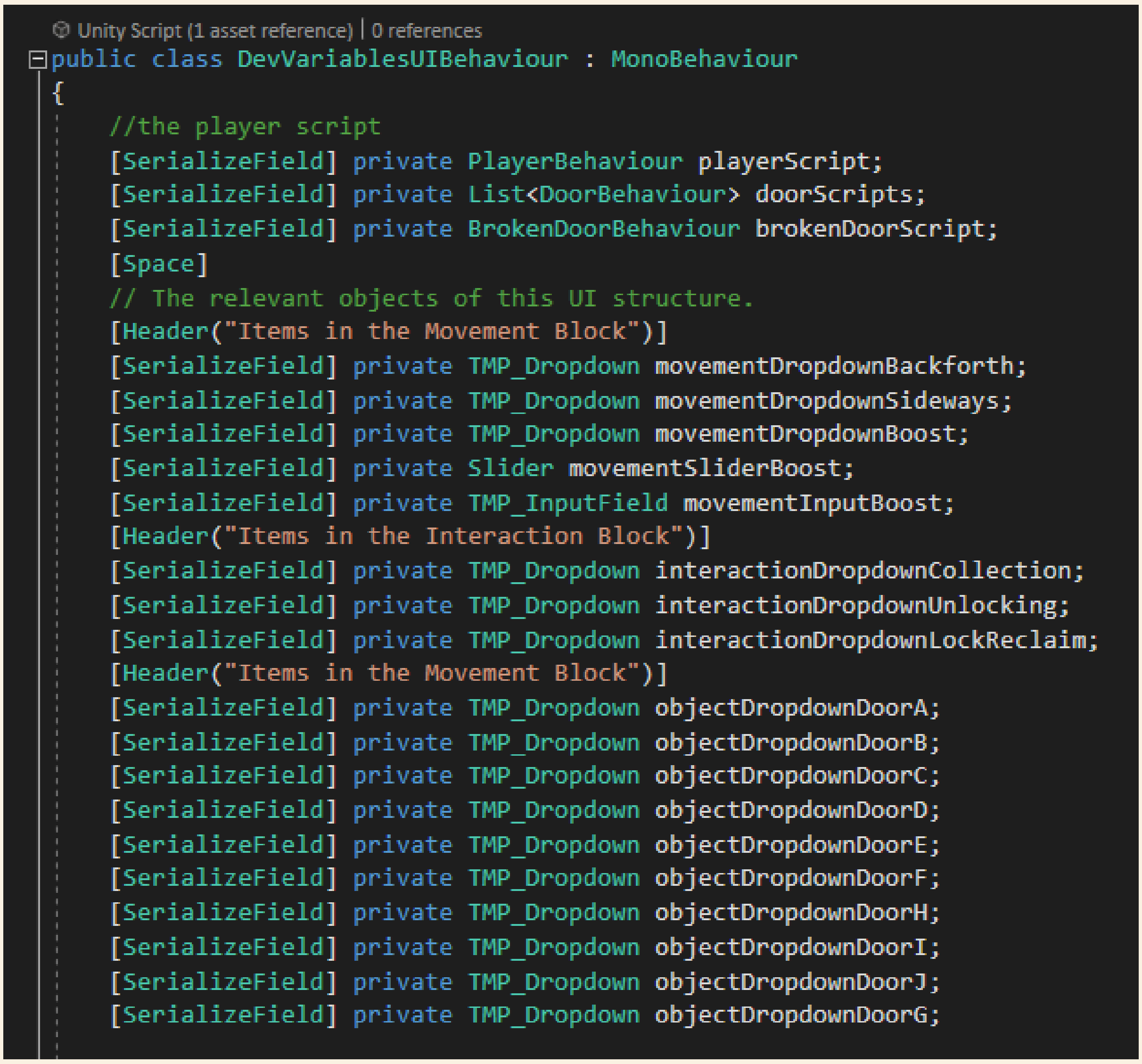
```
private void Start()
{
    //initialise base panel
    panel = GetComponent<Image>();
    rectTransform = GetComponent<RectTransform>();
    rectTransform.sizeDelta = new Vector2(rectTransform.sizeDelta.x, 0.0f);

    //the blocks start by being turned off.
    ToggleBlock(movementBlock, false);
    ToggleBlock(interactionBlock, false);
    ToggleBlock(objectBlock, false);
    ToggleBlock(customBlock, false);

    //the blocks also start with the size on the x-axis minimized (which makes them invisible)
    Vector3 mBlockScale = movementBlock.GetComponent<RectTransform>().localScale;
    movementBlock.GetComponent<RectTransform>().localScale = new Vector3(0.0f, mBlockScale.y, mBlockScale.z);
    Vector3 iBlockScale = interactionBlock.GetComponent<RectTransform>().localScale;
    interactionBlock.GetComponent<RectTransform>().localScale = new Vector3(0.0f, iBlockScale.y, iBlockScale.z);
    Vector3 oBlockScale = objectBlock.GetComponent<RectTransform>().localScale;
    objectBlock.GetComponent<RectTransform>().localScale = new Vector3(0.0f, oBlockScale.y, oBlockScale.z);
    Vector3 cBlockScale = customBlock.GetComponent<RectTransform>().localScale;
    customBlock.GetComponent<RectTransform>().localScale = new Vector3(0.0f, cBlockScale.y, cBlockScale.z);

    //initialise all strings
    movementTargetString =          movementBlockText.text;
    movementRandomStringStore =     movementTargetString;
    interactionTargetString =       interactionBlockText.text;
    interactionRandomStringStore =  interactionTargetString;
    objectTargetString =            objectBlockText.text;
    objectRandomStringStore =       objectTargetString;
    customTargetString =            customBlockText.text;
    customRandomStringStore =       customTargetString;
```

In its start function, it initialises the position and size of its pseudocode blocks, and sets their text strings, in preparation for scrambling.



```
//visual of the scrambled text
4 references
private void ScrambleText(TMP_Text text, string targetString, string storeString, int blockIndex)
{
    timer += Time.deltaTime;
    if (timer >= updateInterval)
    {
        //rescramble
        timer = 0;
        randomString = "";

        for (int i = 0; i < targetString.Length; i++)
        {
            //scramble each individual character with a percentage chance.
            int generatedChance = Random.Range(1, 100);
            if (generatedChance <= scrambleChance)
            {
                randomString += possibleCharacters[Random.Range(0, possibleCharacters.Length)];
            }
            else
            {
                randomString += storeString[i];
            }
        }

        //set the text to this slightly adapted random string
        text.text = randomString;

        //now store the random string so we can recall it next frame
        if (blockIndex == 0)
        {
            movementRandomStringStore = randomString;
        }
        if (blockIndex == 1)
        {
            interactionRandomStringStore = randomString;
        }
        if (blockIndex == 2)
        {
            objectRandomStringStore = randomString;
        }
        if (blockIndex == 3)
        {
            customRandomStringStore = randomString;
        }
    }
```

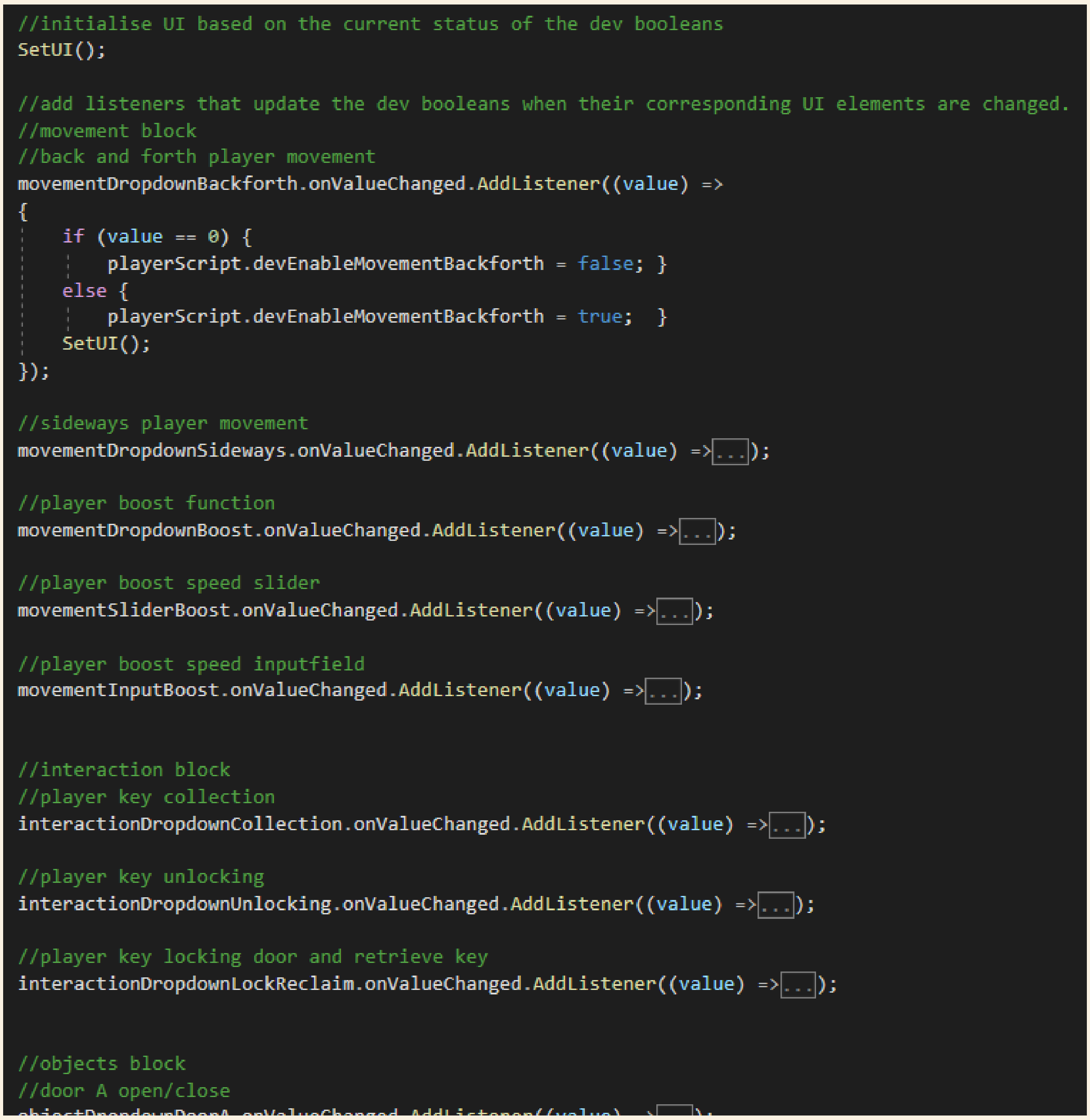
The scrambling function works like this. It randomly generates new characters and places those in the displayed string. The unscramble function is the same, except it doesn’t scramble towards random possible characters, but instead towards the target string. This gives it the illusion of slowly unscrambling.



```
Unity Script (1 asset reference) | 0 references
public class DevVariablesUIBehaviour : MonoBehaviour
{
    //the player script
    [SerializeField] private PlayerBehaviour playerScript;
    [SerializeField] private List<DoorBehaviour> doorScripts;
    [SerializeField] private BrokenDoorBehaviour brokenDoorScript;
    [Space]
    // The relevant objects of this UI structure.
    [Header("Items in the Movement Block")]
    [SerializeField] private TMP_Dropdown movementDropdownBackforth;
    [SerializeField] private TMP_Dropdown movementDropdownSideways;
    [SerializeField] private TMP_Dropdown movementDropdownBoost;
    [SerializeField] private Slider movementSliderBoost;
    [SerializeField] private TMP_InputField movementInputBoost;
    [Header("Items in the Interaction Block")]
    [SerializeField] private TMP_Dropdown interactionDropdownCollection;
    [SerializeField] private TMP_Dropdown interactionDropdownUnlocking;
    [SerializeField] private TMP_Dropdown interactionDropdownLockReclaim;
    [Header("Items in the Movement Block")]
    [SerializeField] private TMP_Dropdown objectDropdownDoorA;
    [SerializeField] private TMP_Dropdown objectDropdownDoorB;
    [SerializeField] private TMP_Dropdown objectDropdownDoorC;
    [SerializeField] private TMP_Dropdown objectDropdownDoorD;
    [SerializeField] private TMP_Dropdown objectDropdownDoorE;
    [SerializeField] private TMP_Dropdown objectDropdownDoorF;
    [SerializeField] private TMP_Dropdown objectDropdownDoorH;
    [SerializeField] private TMP_Dropdown objectDropdownDoorI;
    [SerializeField] private TMP_Dropdown objectDropdownDoorJ;
    [SerializeField] private TMP_Dropdown objectDropdownDoorG;
```

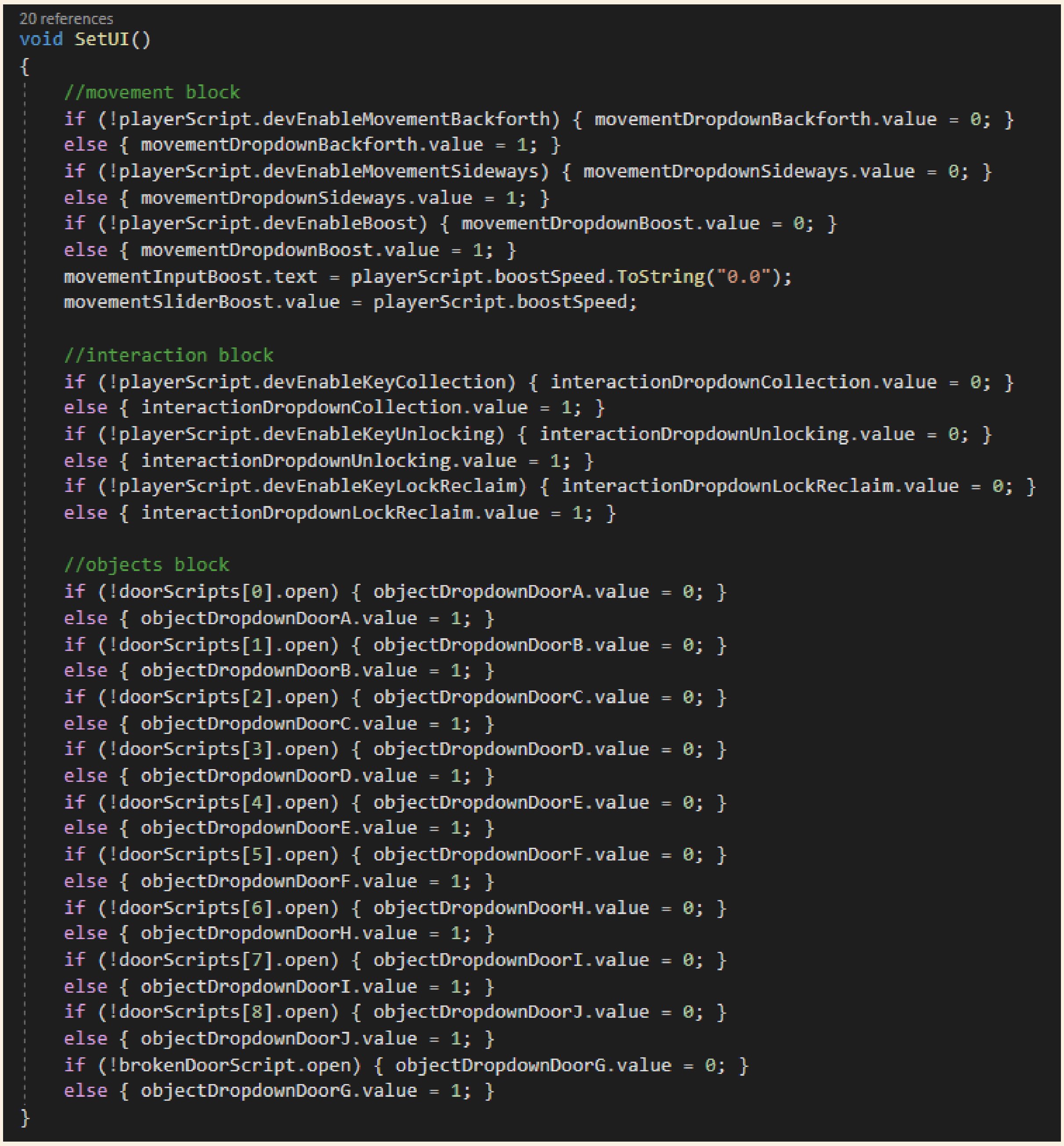
The group object also features the DevVariablesUIBehaviour.

It has references to every variable in the developer user interface.



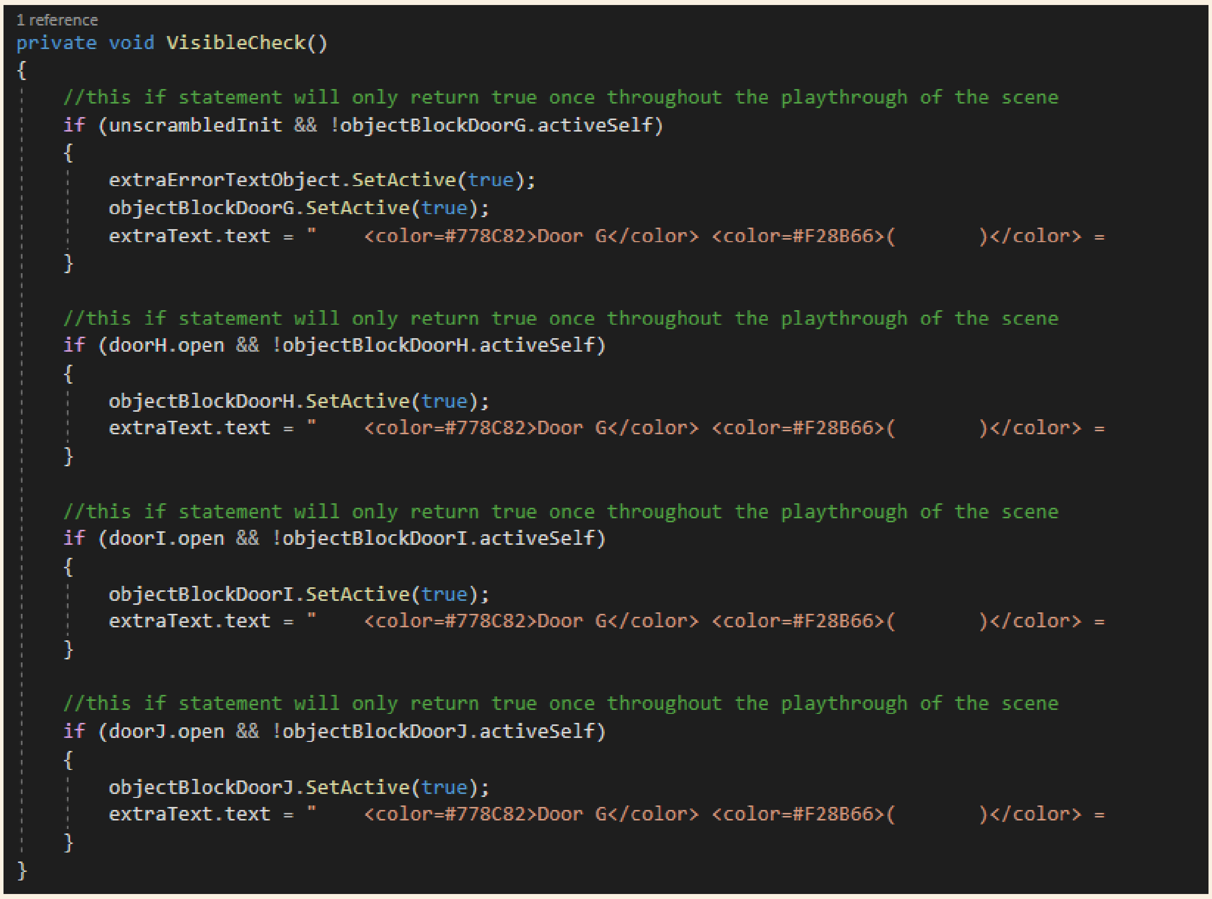
```
//initialise UI based on the current status of the dev booleans
SetUI();

//add listeners that update the dev booleans when their corresponding UI elements are changed.
//movement block
//back and forth player movement
movementDropdownBackforth.onValueChanged.AddListener((value) =>
{
    if (value == 0) {
        playerScript.devEnableMovementBackforth = false; }
    else {
        playerScript.devEnableMovementBackforth = true;  }
    SetUI();
});

//sideways player movement
movementDropdownSideways.onValueChanged.AddListener((value) =>...);

//player boost function
movementDropdownBoost.onValueChanged.AddListener((value) =>...);

//player boost speed slider
movementSliderBoost.onValueChanged.AddListener((value) =>...);

//player boost speed inputfield
movementInputBoost.onValueChanged.AddListener((value) =>...);


//interaction block
//player key collection
interactionDropdownCollection.onValueChanged.AddListener((value) =>...);

//player key unlocking
interactionDropdownUnlocking.onValueChanged.AddListener((value) =>...);

//player key locking door and retrieve key
interactionDropdownLockReclaim.onValueChanged.AddListener((value) =>...);


//objects block
//door A open/close
objectDropdownDoorA.onValueChanged.AddListener((value) =>...);
```

It adds listeners to all these variables, which change the status of the real game mechanics they are referring to.



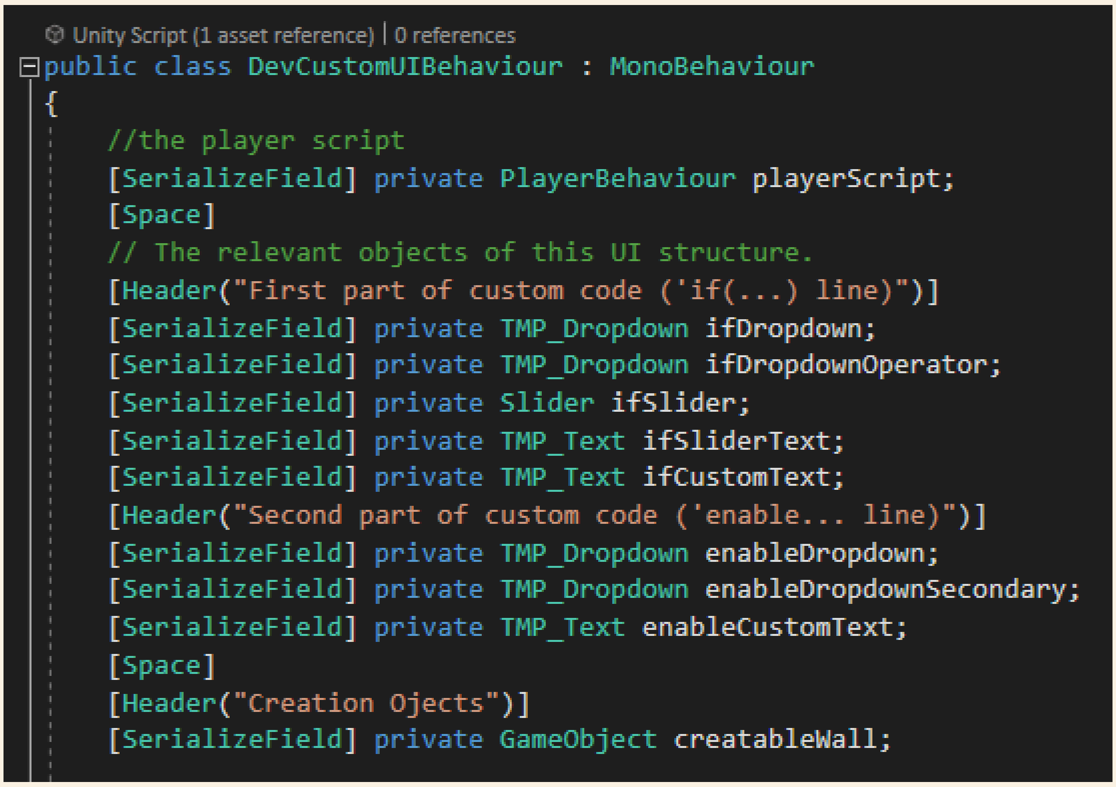
```
20 references
void SetUI()
{
    //movement block
    if (!playerScript.devEnableMovementBackforth) { movementDropdownBackforth.value = 0; }
    else { movementDropdownBackforth.value = 1; }
    if (!playerScript.devEnableMovementSideways) { movementDropdownSideways.value = 0; }
    else { movementDropdownSideways.value = 1; }
    if (!playerScript.devEnableBoost) { movementDropdownBoost.value = 0; }
    else { movementDropdownBoost.value = 1; }
    movementInputBoost.text = playerScript.boostSpeed.ToString("0.0");
    movementSliderBoost.value = playerScript.boostSpeed;

    //interaction block
    if (!playerScript.devEnableKeyCollection) { interactionDropdownCollection.value = 0; }
    else { interactionDropdownCollection.value = 1; }
    if (!playerScript.devEnableKeyUnlocking) { interactionDropdownUnlocking.value = 0; }
    else { interactionDropdownUnlocking.value = 1; }
    if (!playerScript.devEnableKeyLockReclaim) { interactionDropdownLockReclaim.value = 0; }
    else { interactionDropdownLockReclaim.value = 1; }

    //objects block
    if (!doorScripts[0].open) { objectDropdownDoorA.value = 0; }
    else { objectDropdownDoorA.value = 1; }
    if (!doorScripts[1].open) { objectDropdownDoorB.value = 0; }
    else { objectDropdownDoorB.value = 1; }
    if (!doorScripts[2].open) { objectDropdownDoorC.value = 0; }
    else { objectDropdownDoorC.value = 1; }
    if (!doorScripts[3].open) { objectDropdownDoorD.value = 0; }
    else { objectDropdownDoorD.value = 1; }
    if (!doorScripts[4].open) { objectDropdownDoorE.value = 0; }
    else { objectDropdownDoorE.value = 1; }
    if (!doorScripts[5].open) { objectDropdownDoorF.value = 0; }
    else { objectDropdownDoorF.value = 1; }
    if (!doorScripts[6].open) { objectDropdownDoorH.value = 0; }
    else { objectDropdownDoorH.value = 1; }
    if (!doorScripts[7].open) { objectDropdownDoorI.value = 0; }
    else { objectDropdownDoorI.value = 1; }
    if (!doorScripts[8].open) { objectDropdownDoorJ.value = 0; }
    else { objectDropdownDoorJ.value = 1; }
    if (!brokenDoorScript.open) { objectDropdownDoorG.value = 0; }
    else { objectDropdownDoorG.value = 1; }
}
```

Additionally, every time one of these variables changes, the developer user interface is updated to reflect the new changes.



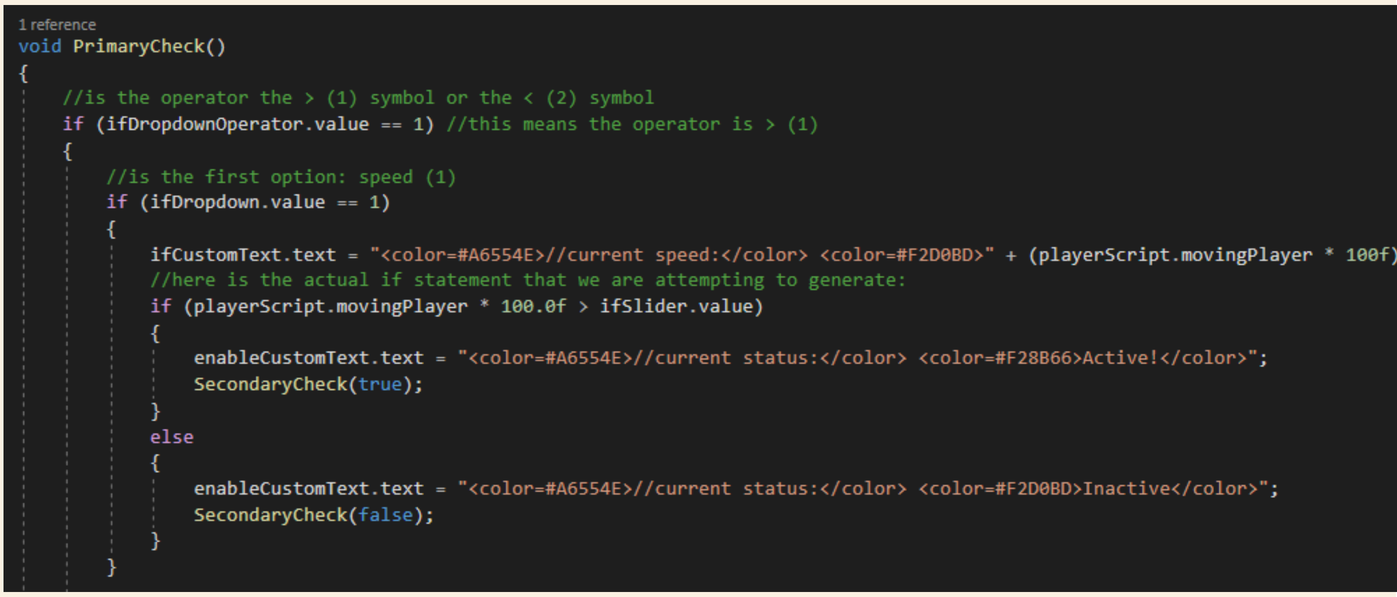
```
1 reference
private void VisibleCheck()
{
    //this if statement will only return true once throughout the playthrough of the scene
    if (unscrambledInit && !objectBlockDoorG.activeSelf)
    {
        extraErrorTextObject.SetActive(true);
        objectBlockDoorG.SetActive(true);
        extraText.text = "    <color=#778C82>Door G</color> <color=#F28B66>(       )</color> =
    }

    //this if statement will only return true once throughout the playthrough of the scene
    if (doorH.open && !objectBlockDoorH.activeSelf)
    {
        objectBlockDoorH.SetActive(true);
        extraText.text = "    <color=#778C82>Door G</color> <color=#F28B66>(       )</color> =
    }

    //this if statement will only return true once throughout the playthrough of the scene
    if (doorI.open && !objectBlockDoorI.activeSelf)
    {
        objectBlockDoorI.SetActive(true);
        extraText.text = "    <color=#778C82>Door G</color> <color=#F28B66>(       )</color> =
    }

    //this if statement will only return true once throughout the playthrough of the scene
    if (doorJ.open && !objectBlockDoorJ.activeSelf)
    {
        objectBlockDoorJ.SetActive(true);
        extraText.text = "    <color=#778C82>Door G</color> <color=#F28B66>(       )</color> =
    }
}
```

On the Object block, we also find the DevObjectUIBehaviour script. This script helps as an addition to the Dev UI Script: once the object block is visible, the goal is not to give access to every door in the scene immediately, only the doors the player has already opened. This script does precisely that, by displaying the status of new doors only after they've been opened first.



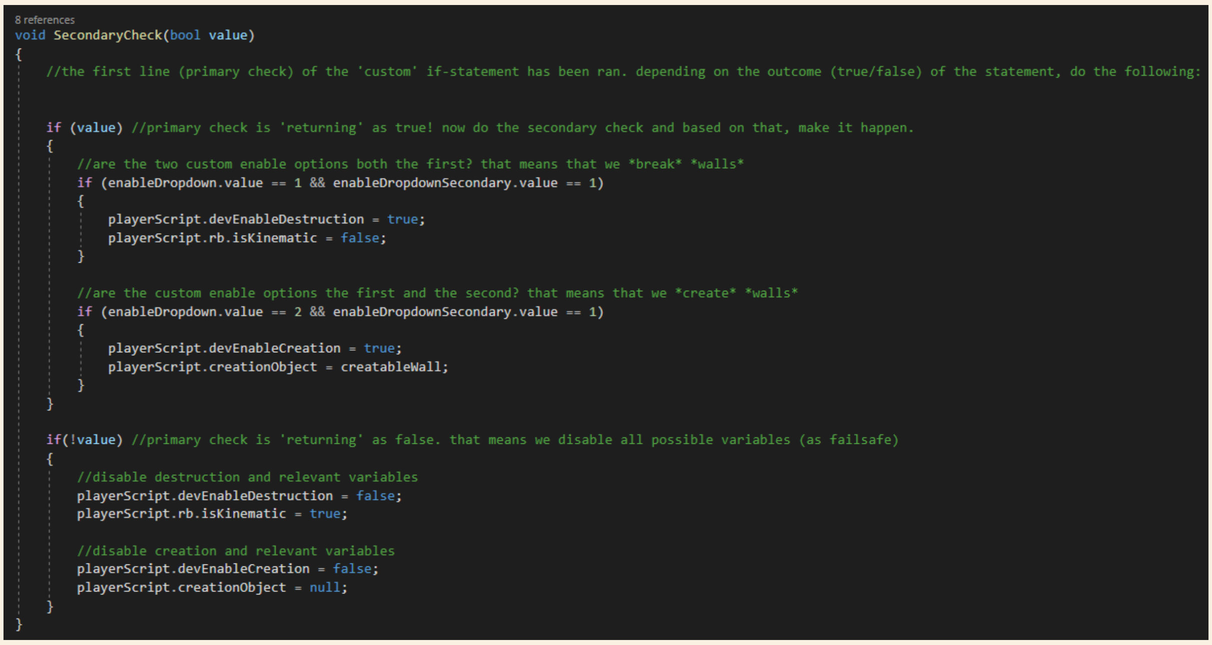
```
Unity Script (1 asset reference) | 0 references
public class DevCustomUIBehaviour : MonoBehaviour
{
    //the player script
    [SerializeField] private PlayerBehaviour playerScript;
    [Space]
    // The relevant objects of this UI structure.
    [Header("First part of custom code ('if(...) line)")]
    [SerializeField] private TMP_Dropdown ifDropdown;
    [SerializeField] private TMP_Dropdown ifDropdownOperator;
    [SerializeField] private Slider ifSlider;
    [SerializeField] private TMP_Text ifSliderText;
    [SerializeField] private TMP_Text ifCustomText;
    [Header("Second part of custom code ('enable... line)")]
    [SerializeField] private TMP_Dropdown enableDropdown;
    [SerializeField] private TMP_Dropdown enableDropdownSecondary;
    [SerializeField] private TMP_Text enableCustomText;
    [Space]
    [Header("Creation Ojects")]
    [SerializeField] private GameObject creatableWall;
```

On the Custom block we find the DevCustomUIBehaviour script.
This is one of the more complex scripts, as it handles the custom if-statement.



```
1 reference
void PrimaryCheck()
{
    //is the operator the > (1) symbol or the < (2) symbol
    if (ifDropdownOperator.value == 1) //this means the operator is > (1)
    {
        //is the first option: speed (1)
        if (ifDropdown.value == 1)
        {
            ifCustomText.text = "<color=#A6554E>//current speed:</color> <color=#F2D0BD>" + (playerScript.movingPlayer * 100f)
            //here is the actual if statement that we are attempting to generate:
            if (playerScript.movingPlayer * 100.0f > ifSlider.value)
            {
                enableCustomText.text = "<color=#A6554E>//current status:</color> <color=#F28B66>Active!</color>";
                SecondaryCheck(true);
            }
            else
            {
                enableCustomText.text = "<color=#A6554E>//current status:</color> <color=#F2D0BD>Inactive</color>";
                SecondaryCheck(false);
            }
        }
```

It initially performs a primary check to see if the first line of the if-statement is returning true.

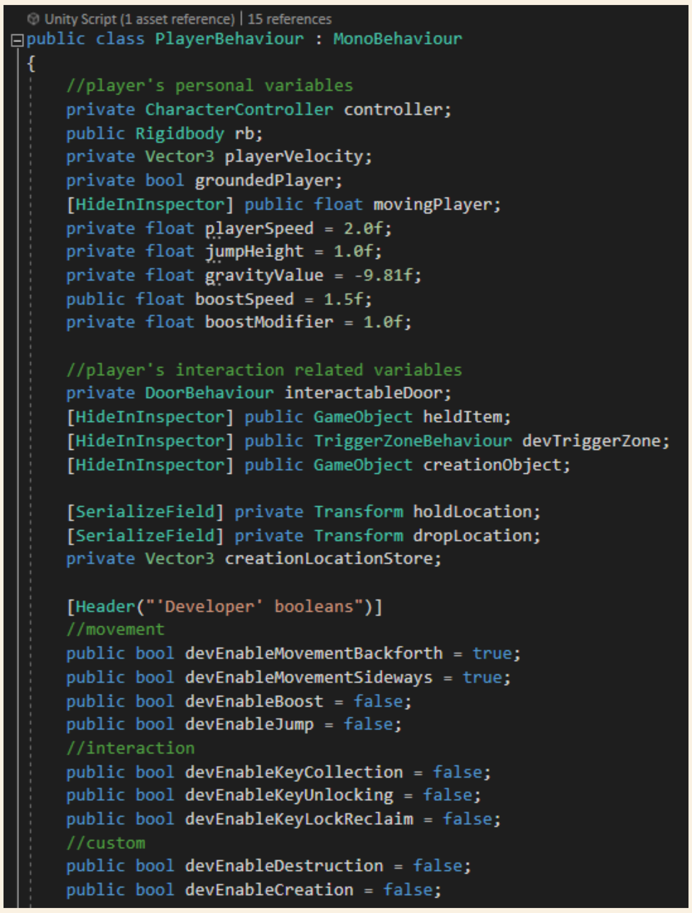


```
8 references
void SecondaryCheck(bool value)
{
    //the first line (primary check) of the 'custom' if-statement has been ran. depending on the outcome (true/false) of the statement, do the following:


    if (value) //primary check is 'returning' as true! now do the secondary check and based on that, make it happen.
    {
        //are the two custom enable options both the first? that means that we *break* *walls*
        if (enableDropdown.value == 1 && enableDropdownSecondary.value == 1)
        {
            playerScript.devEnableDestruction = true;
            playerScript.rb.isKinematic = false;
        }

        //are the custom enable options the first and the second? that means that we *create* *walls*
        if (enableDropdown.value == 2 && enableDropdownSecondary.value == 1)
        {
            playerScript.devEnableCreation = true;
            playerScript.creationObject = creatableWall;
        }
    }

    if(!value) //primary check is 'returning' as false. that means we disable all possible variables (as failsafe)
    {
        //disable destruction and relevant variables
        playerScript.devEnableDestruction = false;
        playerScript.rb.isKinematic = true;

        //disable creation and relevant variables
        playerScript.devEnableCreation = false;
        playerScript.creationObject = null;
    }
}
```

If the primary check returns either true or false (but not null or another type of error), then the secondary check will execute whichever code was created by the player in the Dev UI screen.

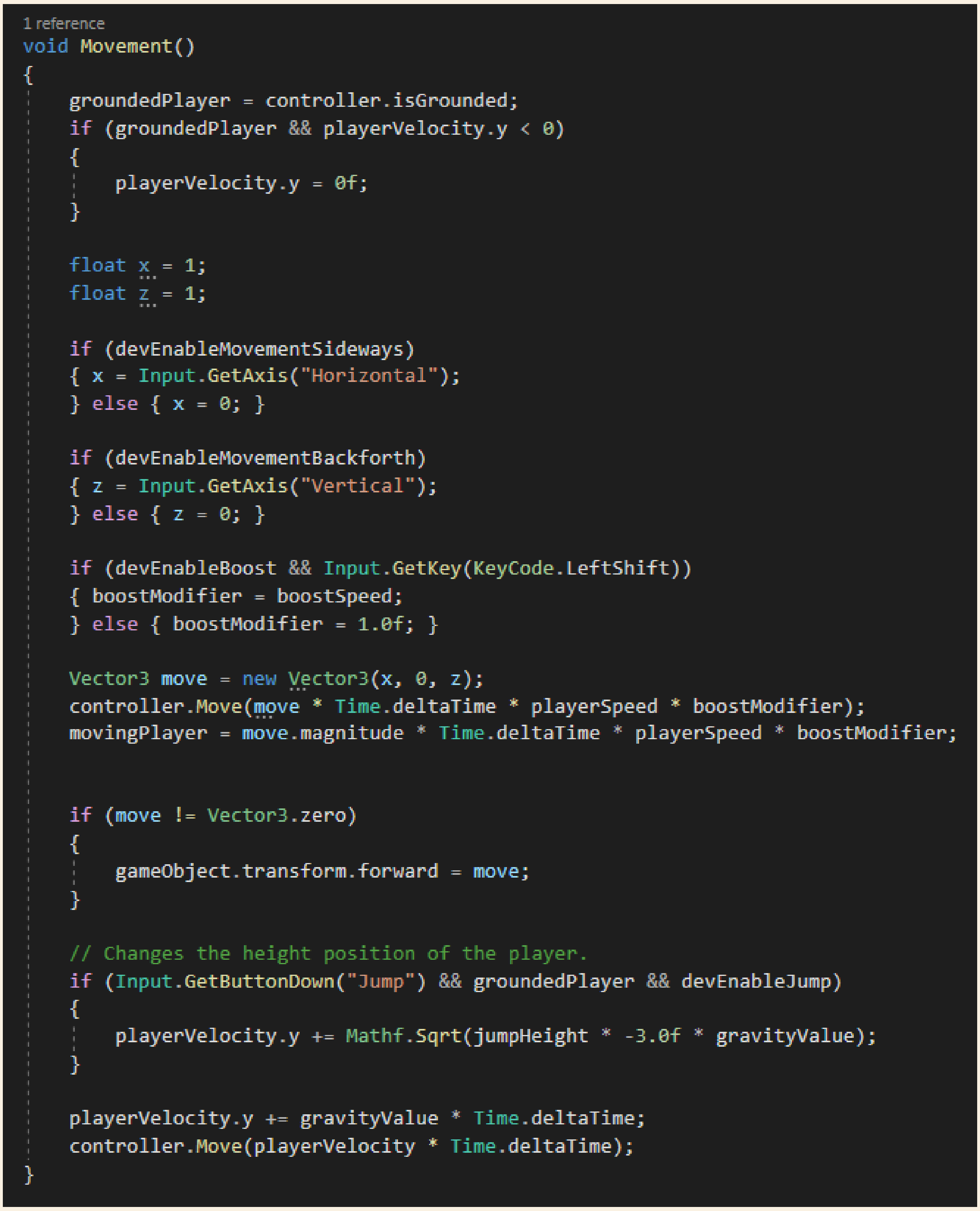
## Player

Of course, another important script is the actual player script. It contains multiple important functions.



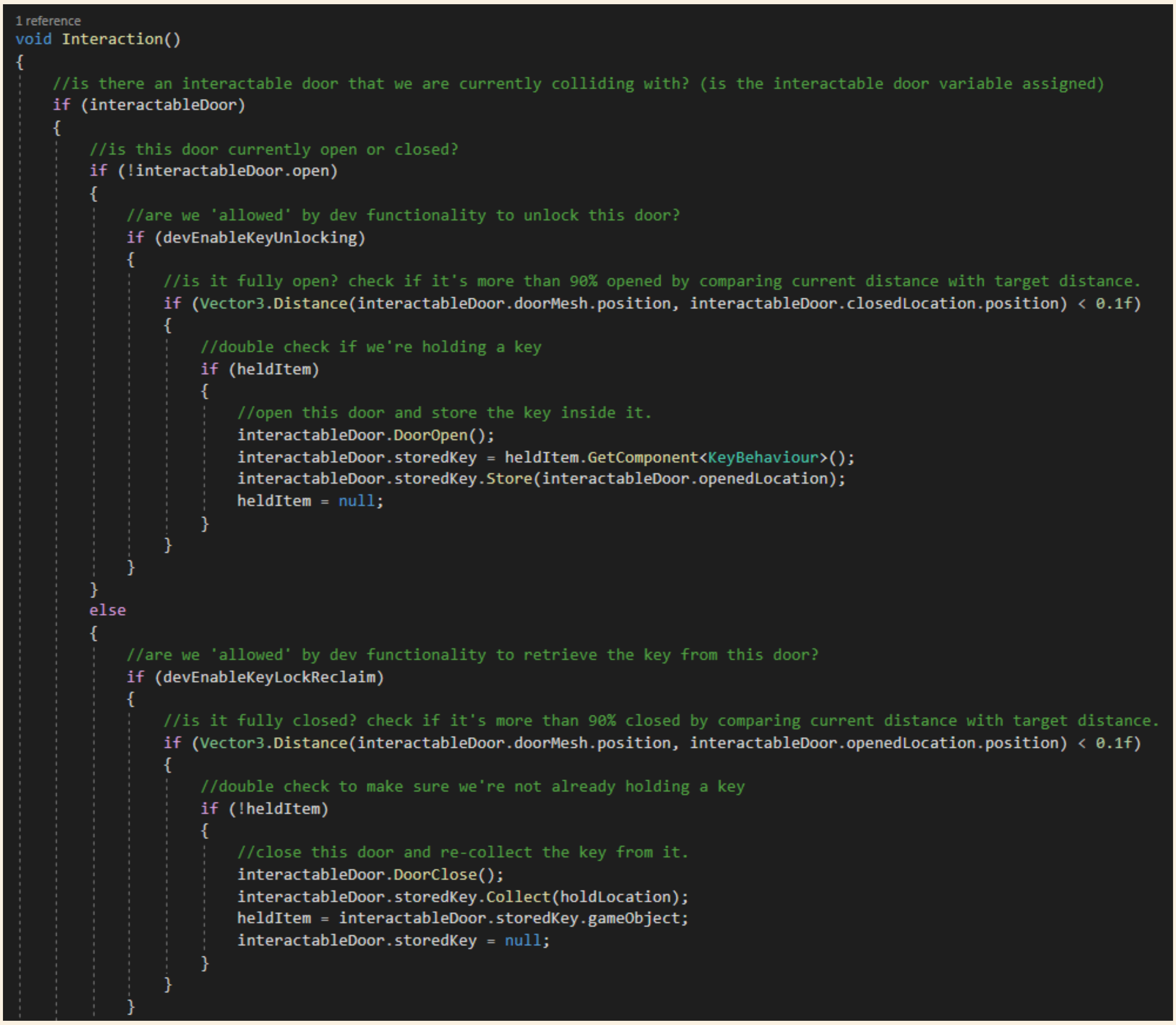
```
Unity Script (1 asset reference) | 15 references
public class PlayerBehaviour : MonoBehaviour
{
    //player's personal variables
    private CharacterController controller;
    public Rigidbody rb;
    private Vector3 playerVelocity;
    private bool groundedPlayer;
    [HideInInspector] public float movingPlayer;
    private float playerSpeed = 2.0f;
    private float jumpHeight = 1.0f;
    private float gravityValue = -9.81f;
    public float boostSpeed = 1.5f;
    private float boostModifier = 1.0f;

    //player's interaction related variables
    private DoorBehaviour interactableDoor;
    [HideInInspector] public GameObject heldItem;
    [HideInInspector] public TriggerZoneBehaviour devTriggerZone;
    [HideInInspector] public GameObject creationObject;

    [SerializeField] private Transform holdLocation;
    [SerializeField] private Transform dropLocation;
    private Vector3 creationLocationStore;

    [Header("'Developer' booleans")]
    //movement
    public bool devEnableMovementBackforth = true;
    public bool devEnableMovementSideways = true;
    public bool devEnableBoost = false;
    public bool devEnableJump = false;
    //interaction
    public bool devEnableKeyCollection = false;
    public bool devEnableKeyUnlocking = false;
    public bool devEnableKeyLockReclaim = false;
    //custom
    public bool devEnableDestruction = false;
    public bool devEnableCreation = false;
```

It contains almost all important variables that are adjusted by the developer user interface.



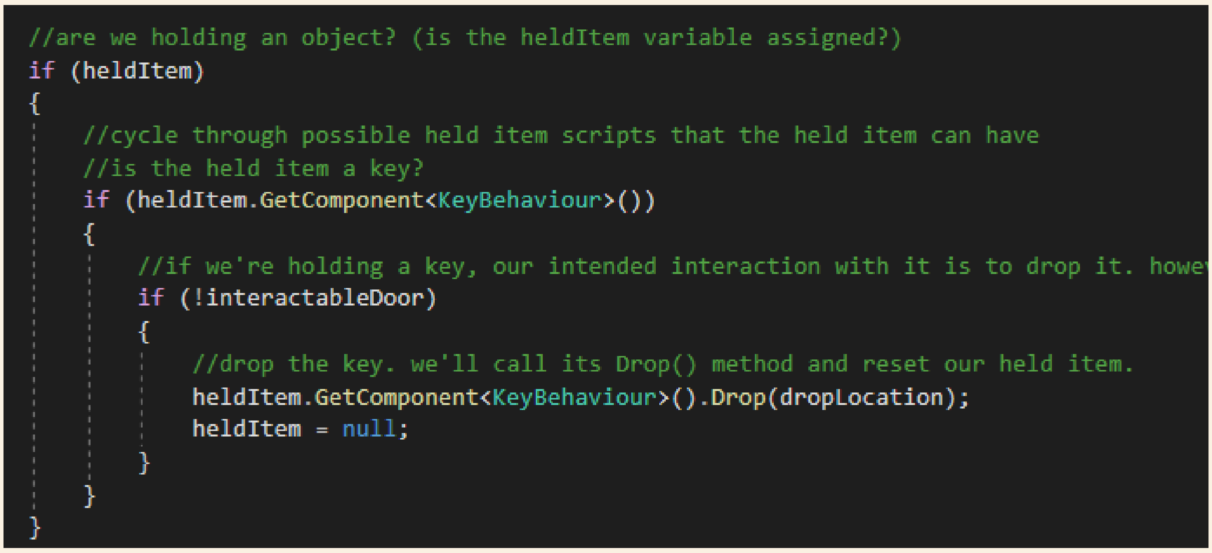
```
1 reference
void Movement()
{
    groundedPlayer = controller.isGrounded;
    if (groundedPlayer && playerVelocity.y < 0)
    {
        playerVelocity.y = 0f;
    }

    float x = 1;
    float z = 1;

    if (devEnableMovementSideways)
    { x = Input.GetAxis("Horizontal");
    } else { x = 0; }

    if (devEnableMovementBackforth)
    { z = Input.GetAxis("Vertical");
    } else { z = 0; }

    if (devEnableBoost && Input.GetKey(KeyCode.LeftShift))
    { boostModifier = boostSpeed;
    } else { boostModifier = 1.0f; }

    Vector3 move = new Vector3(x, 0, z);
    controller.Move(move * Time.deltaTime * playerSpeed * boostModifier);
    movingPlayer = move.magnitude * Time.deltaTime * playerSpeed * boostModifier;


    if (move != Vector3.zero)
    {
        gameObject.transform.forward = move;
    }

    // Changes the height position of the player.
    if (Input.GetButtonDown("Jump") && groundedPlayer && devEnableJump)
    {
        playerVelocity.y += Mathf.Sqrt(jumpHeight * -3.0f * gravityValue);
    }

    playerVelocity.y += gravityValue * Time.deltaTime;
    controller.Move(playerVelocity * Time.deltaTime);
}
```

Its movement method is called every frame, and handles movement. Note the devEnable booleans, they decide whether or not the player is allowed to perform certain actions.

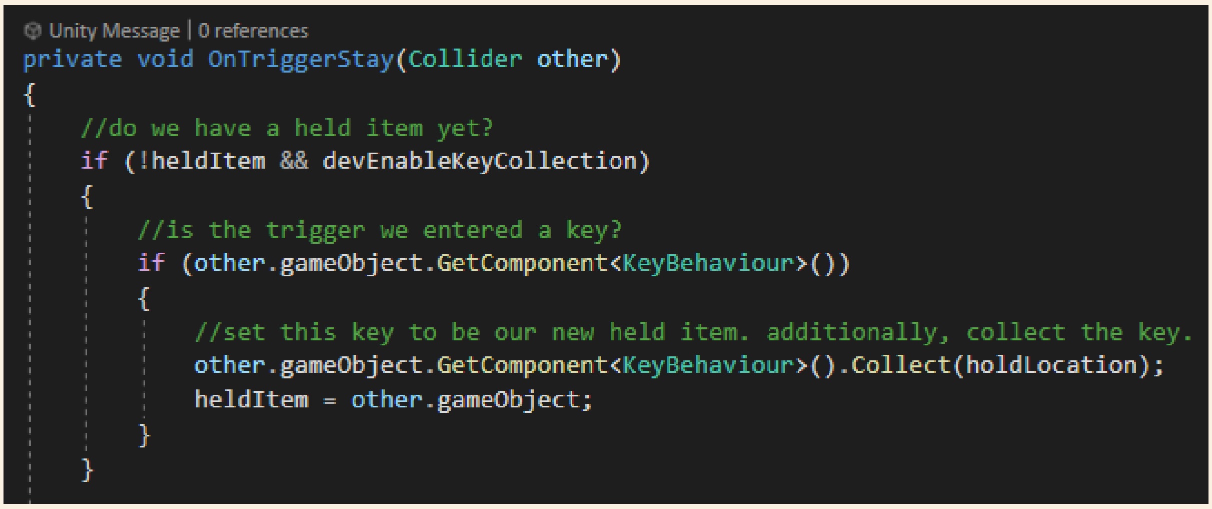


```
1 reference
void Interaction()
{
    //is there an interactable door that we are currently colliding with? (is the interactable door variable assigned)
    if (interactableDoor)
    {
        //is this door currently open or closed?
        if (!interactableDoor.open)
        {
            //are we 'allowed' by dev functionality to unlock this door?
            if (devEnableKeyUnlocking)
            {
                //is it fully open? check if it's more than 90% opened by comparing current distance with target distance.
                if (Vector3.Distance(interactableDoor.doorMesh.position, interactableDoor.closedLocation.position) < 0.1f)
                {
                    //double check if we're holding a key
                    if (heldItem)
                    {
                        //open this door and store the key inside it.
                        interactableDoor.DoorOpen();
                        interactableDoor.storedKey = heldItem.GetComponent<KeyBehaviour>();
                        interactableDoor.storedKey.Store(interactableDoor.openedLocation);
                        heldItem = null;
                    }
                }
            }
        }
        else
        {
            //are we 'allowed' by dev functionality to retrieve the key from this door?
            if (devEnableKeyLockReclaim)
            {
                //is it fully closed? check if it's more than 90% closed by comparing current distance with target distance.
                if (Vector3.Distance(interactableDoor.doorMesh.position, interactableDoor.openedLocation.position) < 0.1f)
                {
                    //double check to make sure we're not already holding a key
                    if (!heldItem)
                    {
                        //close this door and re-collect the key from it.
                        interactableDoor.DoorClose();
                        interactableDoor.storedKey.Collect(holdLocation);
                        heldItem = interactableDoor.storedKey.gameObject;
                        interactableDoor.storedKey = null;
                    }
                }
            }
```

The interaction method contains a long list of if statements: the interactions of the game are incredibly specific, and will only be possible in these specific scenarios.

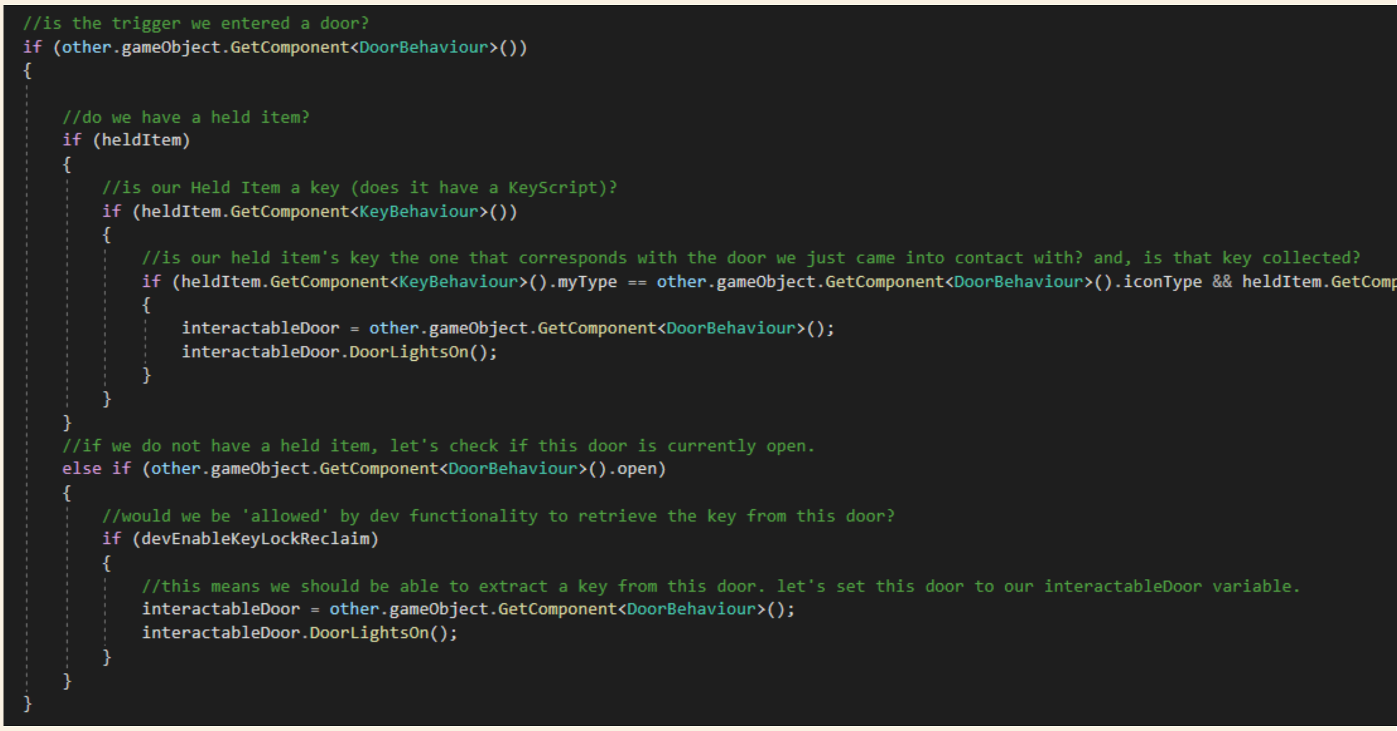


```
//are we holding an object? (is the heldItem variable assigned?)
if (heldItem)
{
    //cycle through possible held item scripts that the held item can have
    //is the held item a key?
    if (heldItem.GetComponent<KeyBehaviour>())
    {
        //if we're holding a key, our intended interaction with it is to drop it. howe
        if (!interactableDoor)
        {
            //drop the key. we'll call its Drop() method and reset our held item.
            heldItem.GetComponent<KeyBehaviour>().Drop(dropLocation);
            heldItem = null;
        }
    }
}
```



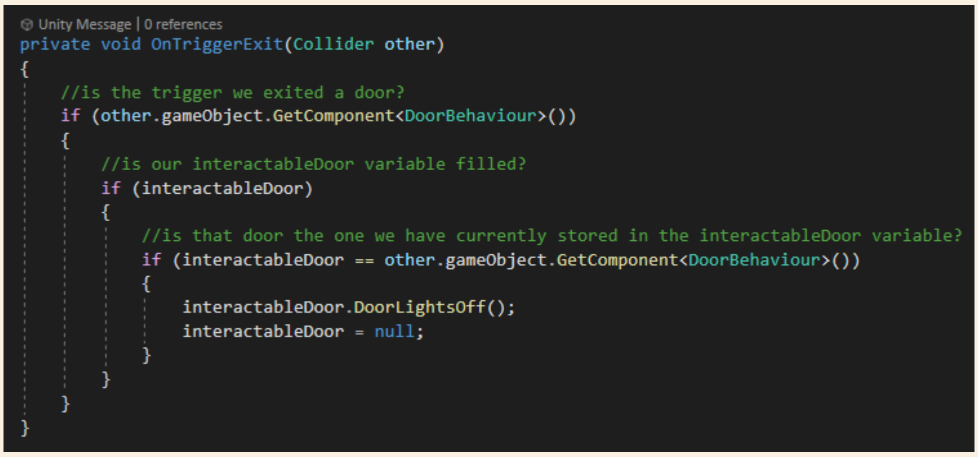
```
Unity Message | 0 references
private void OnTriggerStay(Collider other)
{
    //do we have a held item yet?
    if (!heldItem && devEnableKeyCollection)
    {
        //is the trigger we entered a key?
        if (other.gameObject.GetComponent<KeyBehaviour>())
        {
            //set this key to be our new held item. additionally, collect the key.
            other.gameObject.GetComponent<KeyBehaviour>().Collect(holdLocation);
            heldItem = other.gameObject;
        }
    }
```

If the player touches a key, it will be stored as the player's held item.



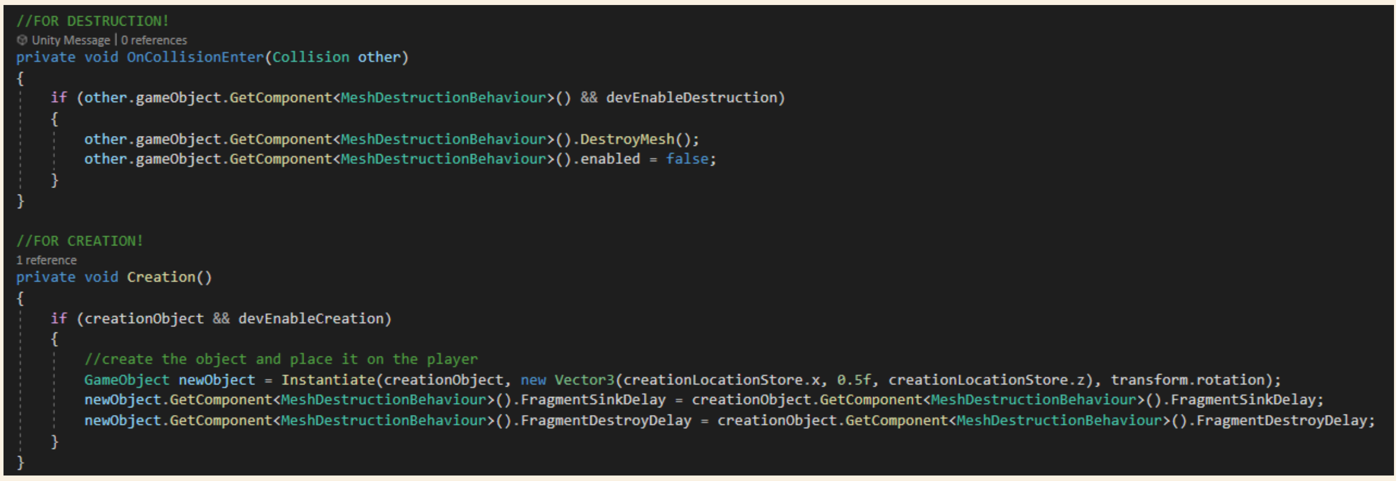
```
//is the trigger we entered a door?
if (other.gameObject.GetComponent<DoorBehaviour>())
{

    //do we have a held item?
    if (heldItem)
    {
        //is our Held Item a key (does it have a KeyScript)?
        if (heldItem.GetComponent<KeyBehaviour>())
        {
            //is our held item's key the one that corresponds with the door we just came into contact with? and, is that key collected?
            if (heldItem.GetComponent<KeyBehaviour>().myType == other.gameObject.GetComponent<DoorBehaviour>().iconType && heldItem.GetComp
            {
                interactableDoor = other.gameObject.GetComponent<DoorBehaviour>();
                interactableDoor.DoorLightsOn();
            }
        }
    }
    //if we do not have a held item, let's check if this door is currently open.
    else if (other.gameObject.GetComponent<DoorBehaviour>().open)
    {
        //would we be 'allowed' by dev functionality to retrieve the key from this door?
        if (devEnableKeyLockReclaim)
        {
            //this means we should be able to extract a key from this door. let's set this door to our interactableDoor variable.
            interactableDoor = other.gameObject.GetComponent<DoorBehaviour>();
            interactableDoor.DoorLightsOn();
        }
    }
}
```

If the player touches a door, and that door is interactable according to the game's rules, then it will light up.



```
Unity Message | 0 references
private void OnTriggerExit(Collider other)
{
    //is the trigger we exited a door?
    if (other.gameObject.GetComponent<DoorBehaviour>())
    {
        //is our interactableDoor variable filled?
        if (interactableDoor)
        {
            //is that door the one we have currently stored in the interactableDoor variable?
            if (interactableDoor == other.gameObject.GetComponent<DoorBehaviour>())
            {
                interactableDoor.DoorLightsOff();
                interactableDoor = null;
            }
        }
    }
}
```

If the player exits a door’s proximity, the door’s lights will turn off.



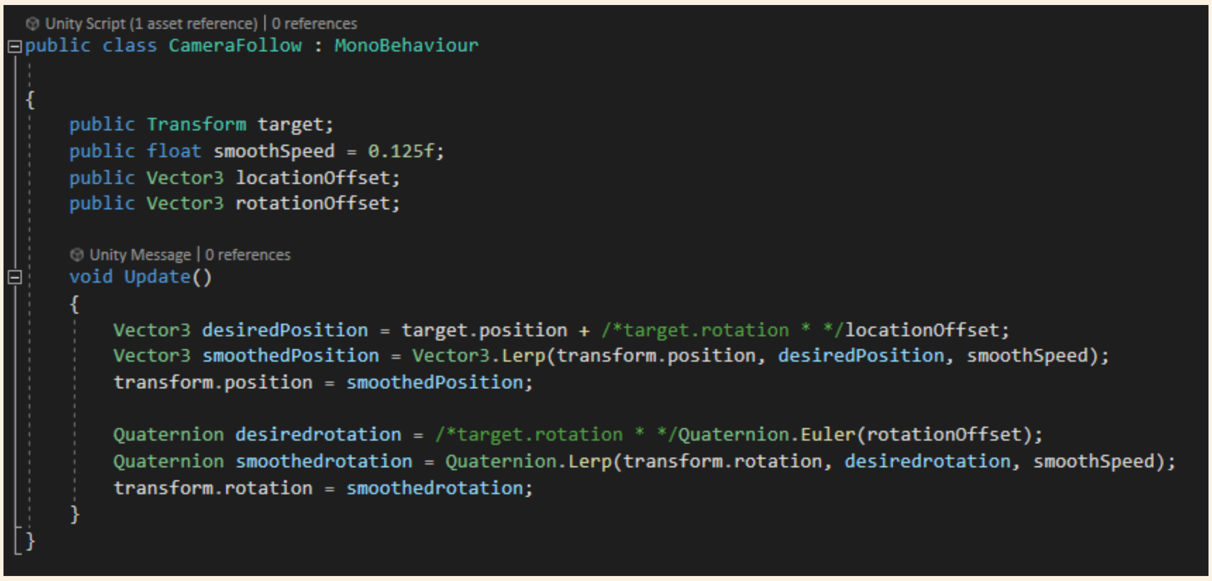
```
//FOR DESTRUCTION!
Unity Message | 0 references
private void OnCollisionEnter(Collision other)
{
    if (other.gameObject.GetComponent<MeshDestructionBehaviour>() && devEnableDestruction)
    {
        other.gameObject.GetComponent<MeshDestructionBehaviour>().DestroyMesh();
        other.gameObject.GetComponent<MeshDestructionBehaviour>().enabled = false;
    }
}

//FOR CREATION!
1 reference
private void Creation()
{
    if (creationObject && devEnableCreation)
    {
        //create the object and place it on the player
        GameObject newObject = Instantiate(creationObject, new Vector3(creationLocationStore.x, 0.5f, creationLocationStore.z), transform.rotation);
        newObject.GetComponent<MeshDestructionBehaviour>().FragmentSinkDelay = creationObject.GetComponent<MeshDestructionBehaviour>().FragmentSinkDelay;
        newObject.GetComponent<MeshDestructionBehaviour>().FragmentDestroyDelay = creationObject.GetComponent<MeshDestructionBehaviour>().FragmentDestroyDelay;
    }
}
```

The player can destroy or create walls using these functions.

### Other scripts

A few other scripts exist in the game.
Namely, the script which handles the broken door, which is similar to the regular door script. Main difference is that it can’t be opened with any keys. Additionally, it has some glitch effects added to the door, such as the scrambling boolean text, and the rotating symbols.

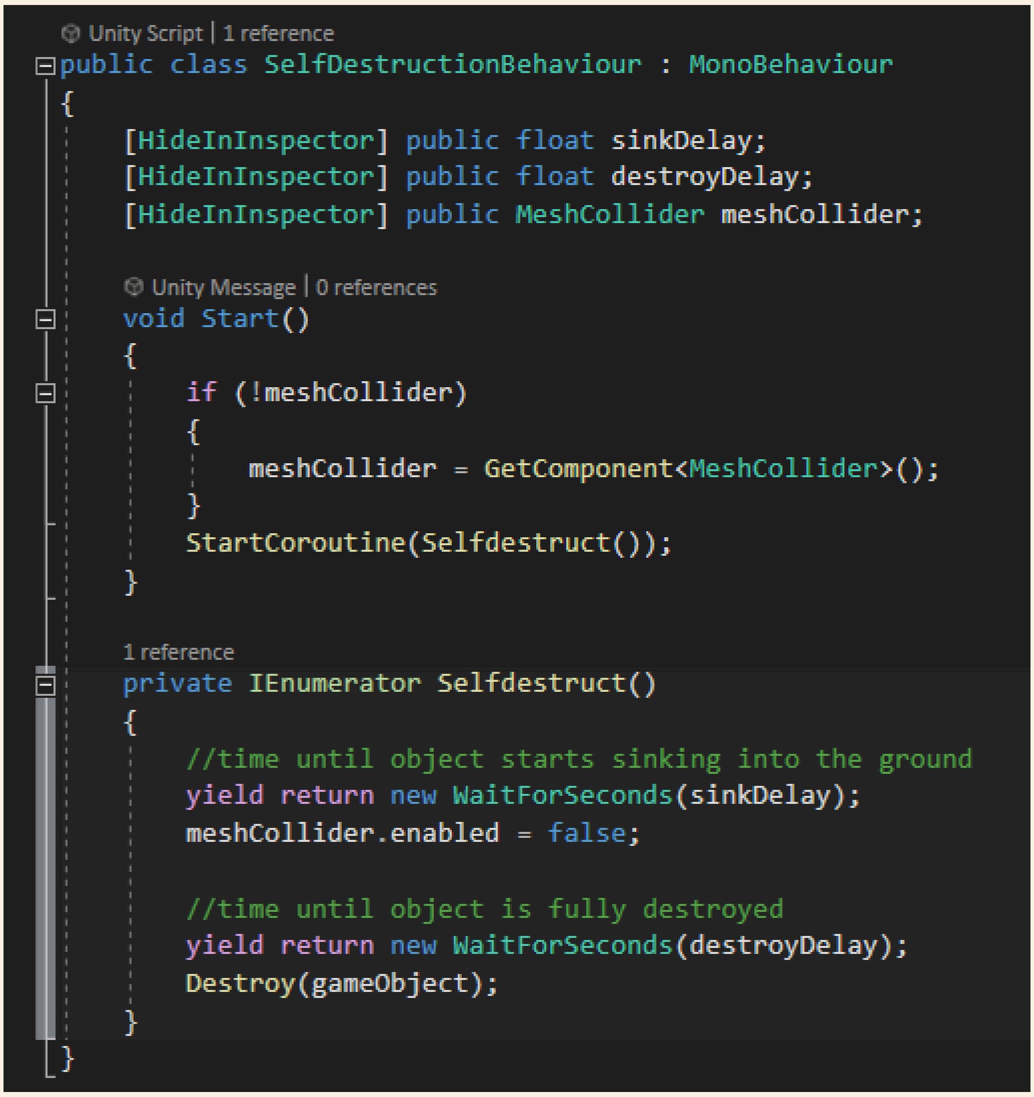
Another smaller script smoothes camera movement:



```
Unity Script (1 asset reference) | 0 references
public class CameraFollow : MonoBehaviour

{
    public Transform target;
    public float smoothSpeed = 0.125f;
    public Vector3 locationOffset;
    public Vector3 rotationOffset;

    Unity Message | 0 references
    void Update()
    {
        Vector3 desiredPosition = target.position + /*target.rotation * */locationOffset;
        Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition, smoothSpeed);
        transform.position = smoothedPosition;

        Quaternion desiredrotation = /*target.rotation * */Quaternion.Euler(rotationOffset);
        Quaternion smoothedrotation = Quaternion.Lerp(transform.rotation, desiredrotation, smoothSpeed);
        transform.rotation = smoothedrotation;
    }
}
```

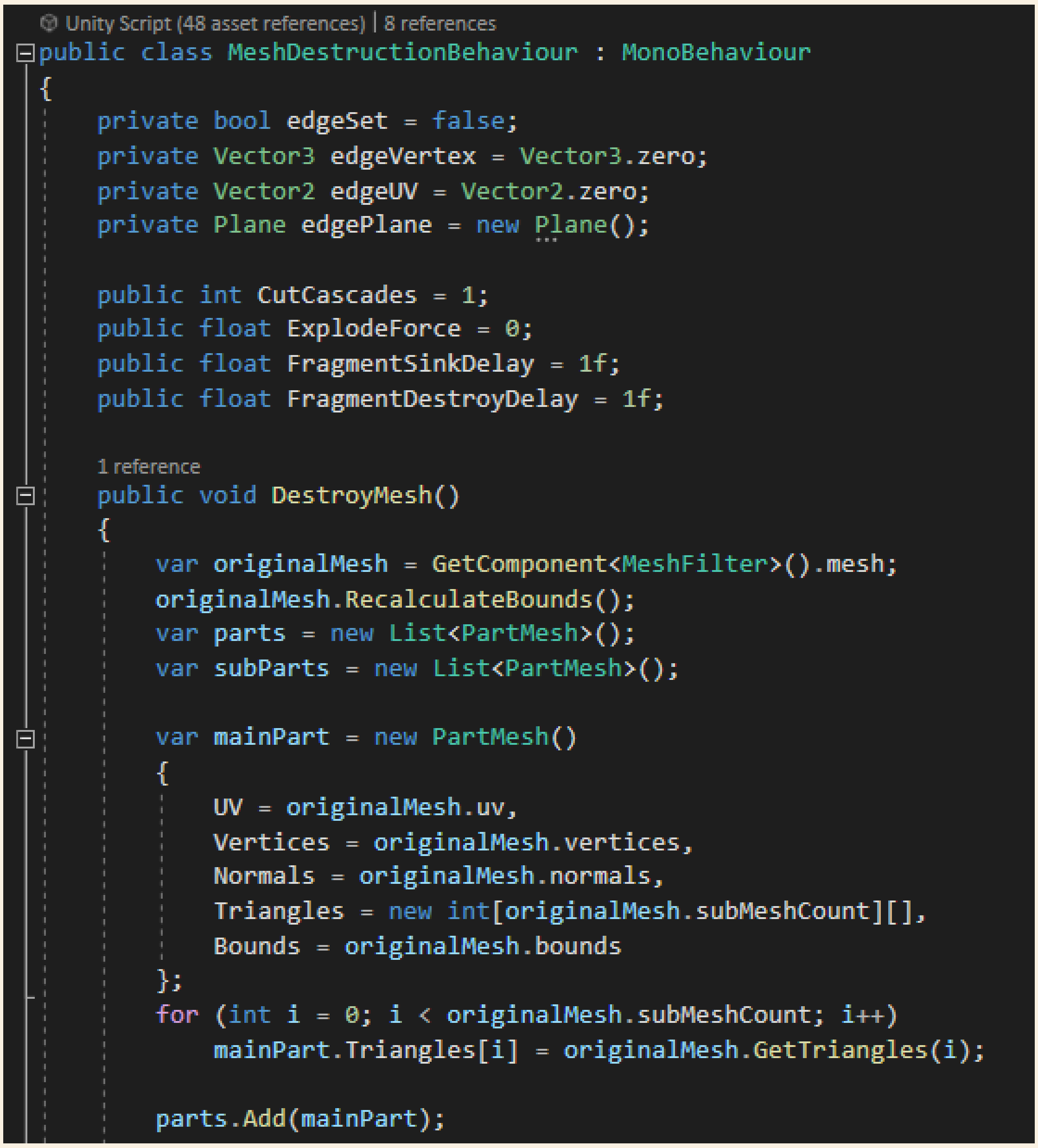
Using an offset and linear interpolation, the target location of the camera is smoothly eased in whenever the player moves.

The self destruction script causes the fragments of destroyed walls to fall through the floor and then self-destruct.



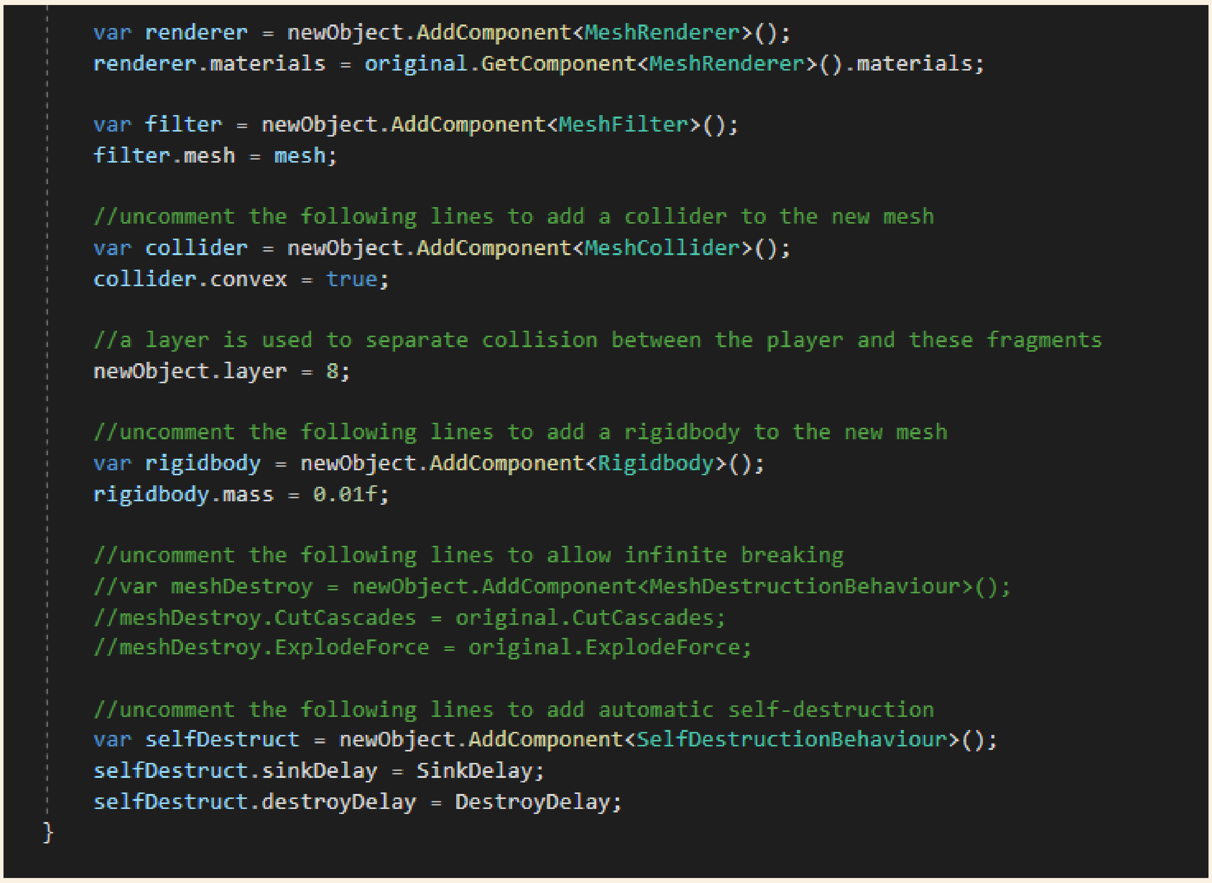
```
Unity Script | 1 reference
public class SelfDestructionBehaviour : MonoBehaviour
{
    [HideInInspector] public float sinkDelay;
    [HideInInspector] public float destroyDelay;
    [HideInInspector] public MeshCollider meshCollider;

    Unity Message | 0 references
    void Start()
    {
        if (!meshCollider)
        {
            meshCollider = GetComponent<MeshCollider>();
        }
        StartCoroutine(Selfdestruct());
    }

    1 reference
    private IEnumerator Selfdestruct()
    {
        //time until object starts sinking into the ground
        yield return new WaitForSeconds(sinkDelay);
        meshCollider.enabled = false;

        //time until object is fully destroyed
        yield return new WaitForSeconds(destroyDelay);
        Destroy(gameObject);
    }
}
```

Also, a specific scramble text script exists, which is used for the malfunctioning screens. It functions the exact same as the scrambling text of the developer user interface, but it has less other functions.



```
Unity Script (48 asset references) | 8 references
public class MeshDestructionBehaviour : MonoBehaviour
{
    private bool edgeSet = false;
    private Vector3 edgeVertex = Vector3.zero;
    private Vector2 edgeUV = Vector2.zero;
    private Plane edgePlane = new Plane();

    public int CutCascades = 1;
    public float ExplodeForce = 0;
    public float FragmentSinkDelay = 1f;
    public float FragmentDestroyDelay = 1f;

    1 reference
    public void DestroyMesh()
    {
        var originalMesh = GetComponent<MeshFilter>().mesh;
        originalMesh.RecalculateBounds();
        var parts = new List<PartMesh>();
        var subParts = new List<PartMesh>();

        var mainPart = new PartMesh()
        {
            UV = originalMesh.uv,
            Vertices = originalMesh.vertices,
            Normals = originalMesh.normals,
            Triangles = new int[originalMesh.subMeshCount][],
            Bounds = originalMesh.bounds
        };
        for (int i = 0; i < originalMesh.subMeshCount; i++)
            mainPart.Triangles[i] = originalMesh.GetTriangles(i);

        parts.Add(mainPart);
```

Lastly, the MeshDestructionBehaviour Script handles the destruction of walls. This script is one of two scripts that were not created originally by the author. It was originally made by Ditzel, and then adapted for this game.
It is a very large script. It calculates random points on the mesh in question, and slices the mesh in pieces. Then the mesh itself is destroyed, and replaced with the calculated pieces.



```
        var renderer = newObject.AddComponent<MeshRenderer>();
        renderer.materials = original.GetComponent<MeshRenderer>().materials;

        var filter = newObject.AddComponent<MeshFilter>();
        filter.mesh = mesh;

        //uncomment the following lines to add a collider to the new mesh
        var collider = newObject.AddComponent<MeshCollider>();
        collider.convex = true;

        //a layer is used to separate collision between the player and these fragments
        newObject.layer = 8;

        //uncomment the following lines to add a rigidbody to the new mesh
        var rigidbody = newObject.AddComponent<Rigidbody>();
        rigidbody.mass = 0.01f;

        //uncomment the following lines to allow infinite breaking
        //var meshDestroy = newObject.AddComponent<MeshDestructionBehaviour>();
        //meshDestroy.CutCascades = original.CutCascades;
        //meshDestroy.ExplodeForce = original.ExplodeForce;

        //uncomment the following lines to add automatic self-destruction
        var selfDestruct = newObject.AddComponent<SelfDestructionBehaviour>();
        selfDestruct.sinkDelay = SinkDelay;
        selfDestruct.destroyDelay = DestroyDelay;
    }
```

This is the code that was added to the script.
These additions serve to give the fragments collision, gravity interaction, and it adds the self destruct script to them, causes the fragments to disappear quickly after they are initially spawned.

## 3.5) Full Playthrough Description

A full playthrough is also available in video format, on the Author's Youtube Channel.
Link for digital readers of this paper: <[https://youtu.be/BLaHjGYnbJY](https://youtu.be/BLaHjGYnbJY)>

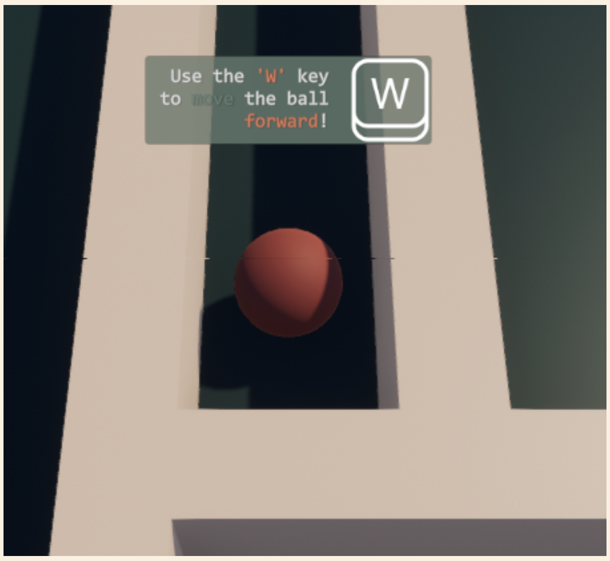
### 3.5.1) Initial Stage

bool correctKey = false;

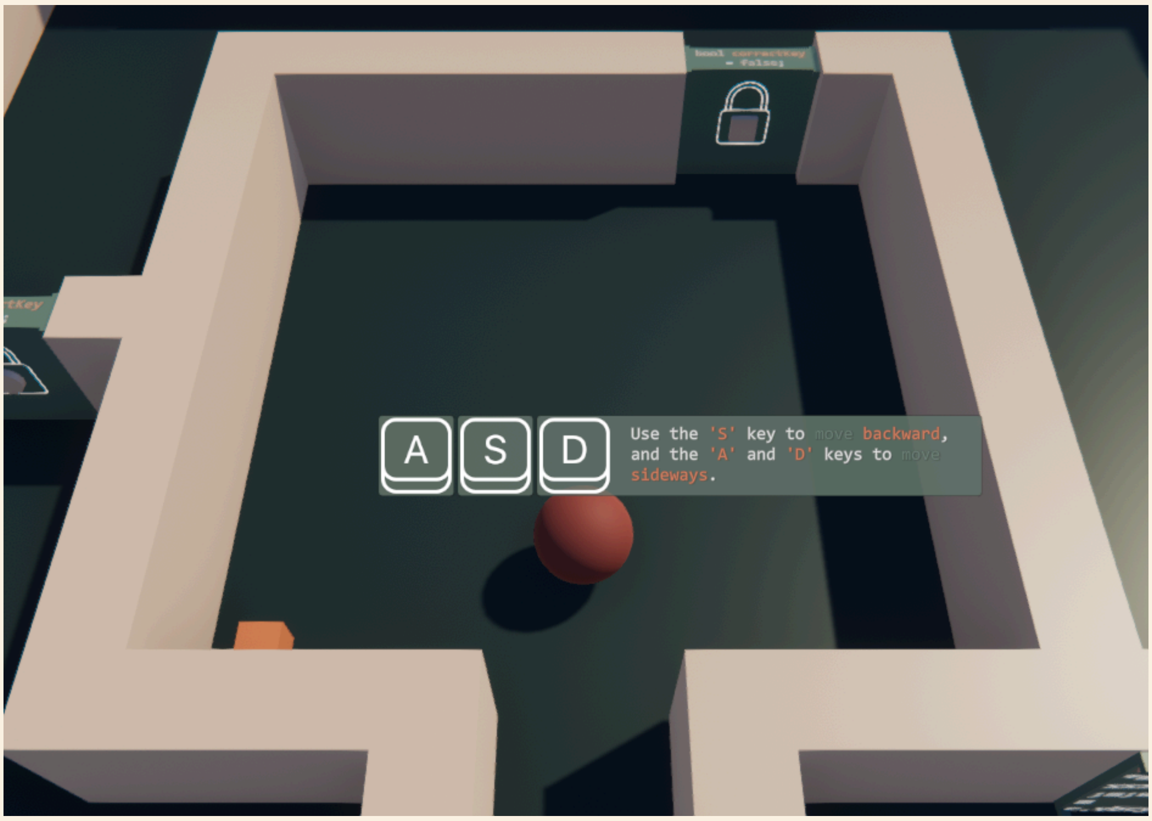
Image showing the first room. ^

#### 3.5.1.1) Game Controls

When the user first starts this game, they'll be welcomed to the maze with its dark green floor and off-white walls. The player will notice their ball in the center of the screen, and a UI pop-up which tells them to use the 'W' key to move their ball forwards.

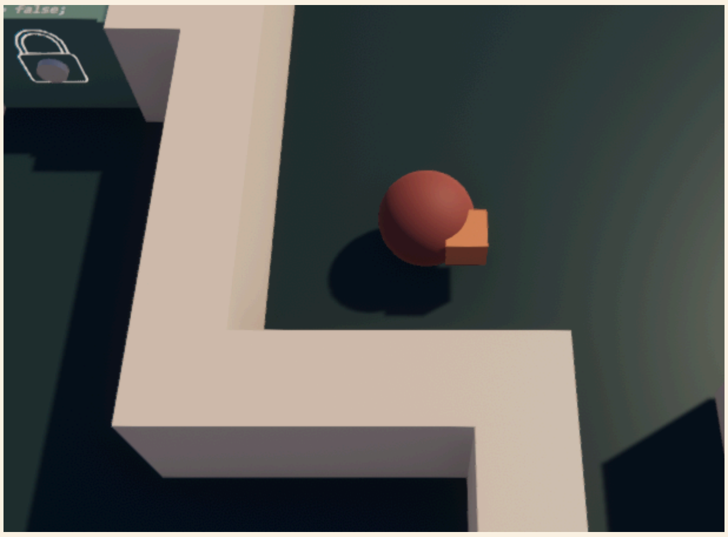


Once the player moves forward a little bit, they'll enter the larger place of the first room.
The 'W' UI element fades out, and a new UI element appears instead:

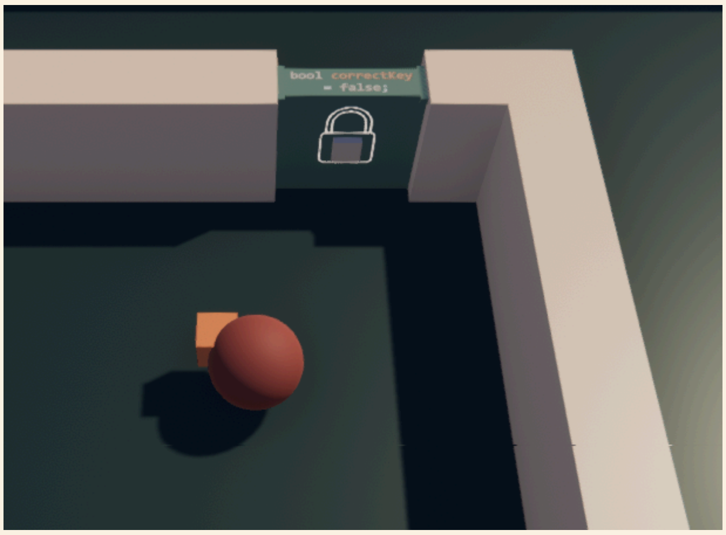


Use the 'S' key to move backward, and the 'A' and 'D' keys to move sideways.

By using the A, S, D and the W key, the player has full two dimension freedom over where to roll their ball.
At this stage, they can explore the room they are in.
The main two things they'll notice are the door in the northeastern corner of the room, and the strange cube in the southwestern part of the room.
If they move towards the cube, and touch it, they'll pick it up.



Their next step will likely be to bring the cube towards the door on the other side of the room.

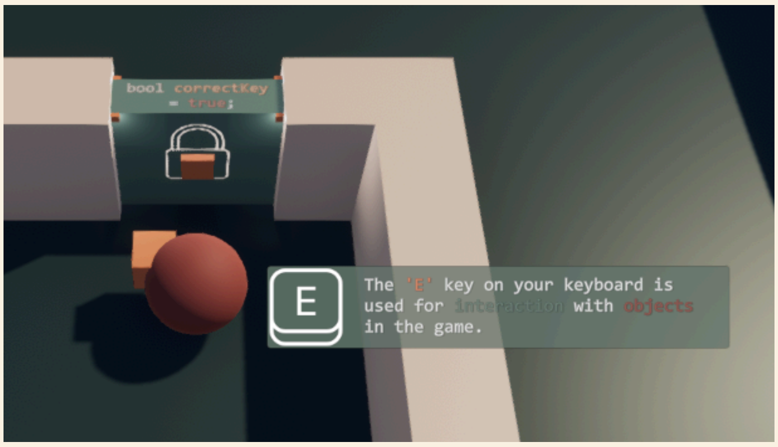


```
bool correctKey
= false;
```

Once they are here, they'll notice that the door lights up when they get close to it while holding the cube.

#### 3.5.1.2) Base Interactions

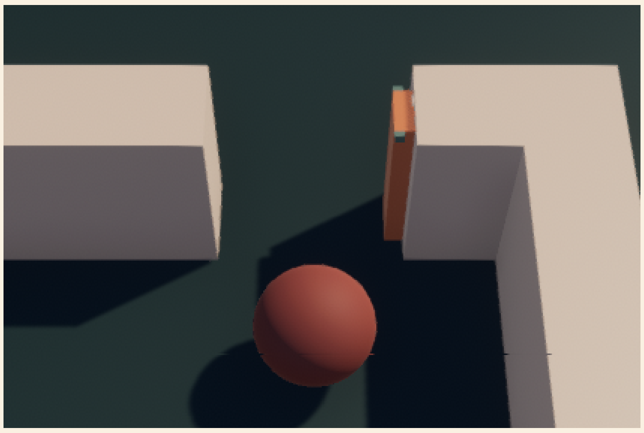
At this point, the player will discover that the cube serves as a key to open the door. Additionally, a new UI element pops up, instructing the player to use the 'E' key on their keyboard to open the door.



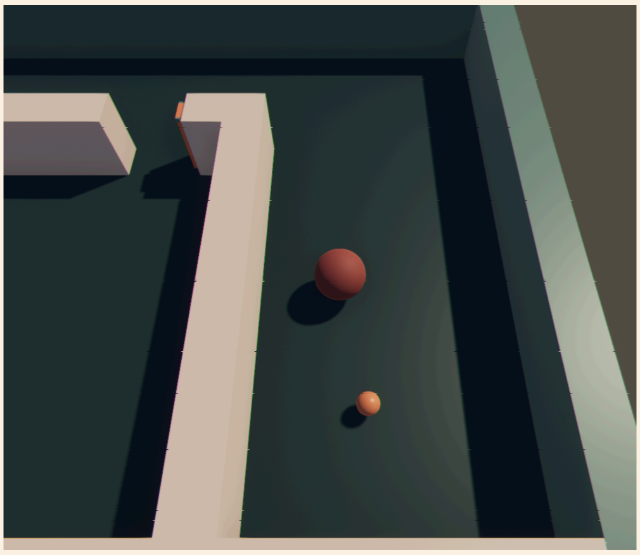
```
bool correctKey
= true;
```

The 'E' key on your keyboard is used for interaction with objects in the game.

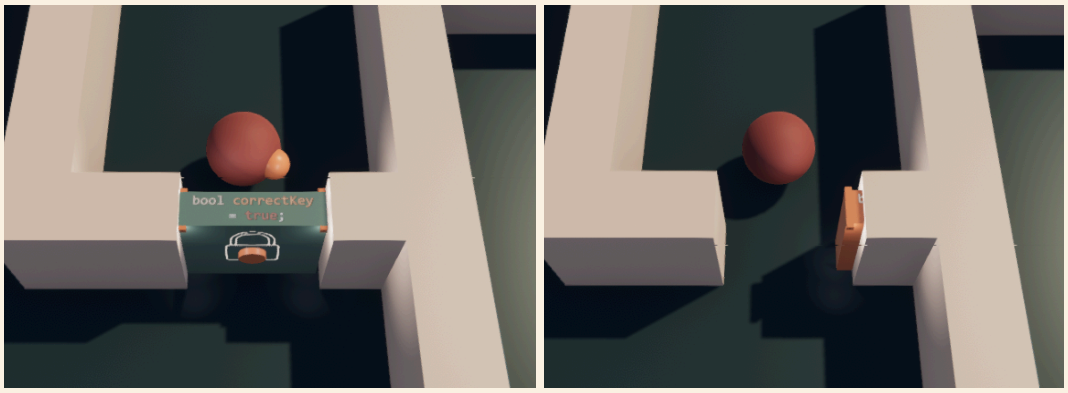
Pressing the 'E' key consumes the key and opens the door.



The player has solved the very first part of the game, and is out of the first room.
In the second room, they'll follow a similar gameplay pattern.

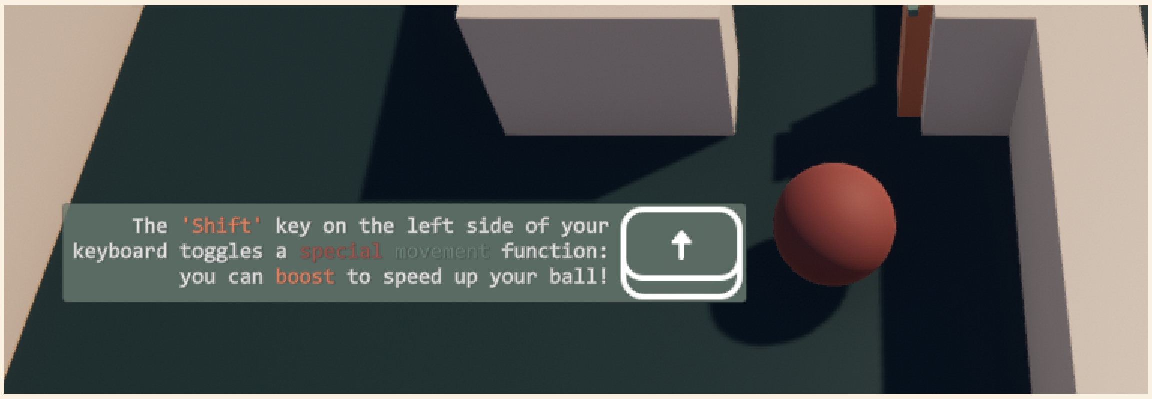


The player finds a new object, this time a small sphere.
After picking it up, they'll bring it to the next door, on the other side of the 'room'.



```
bool correctKey
= true;
```

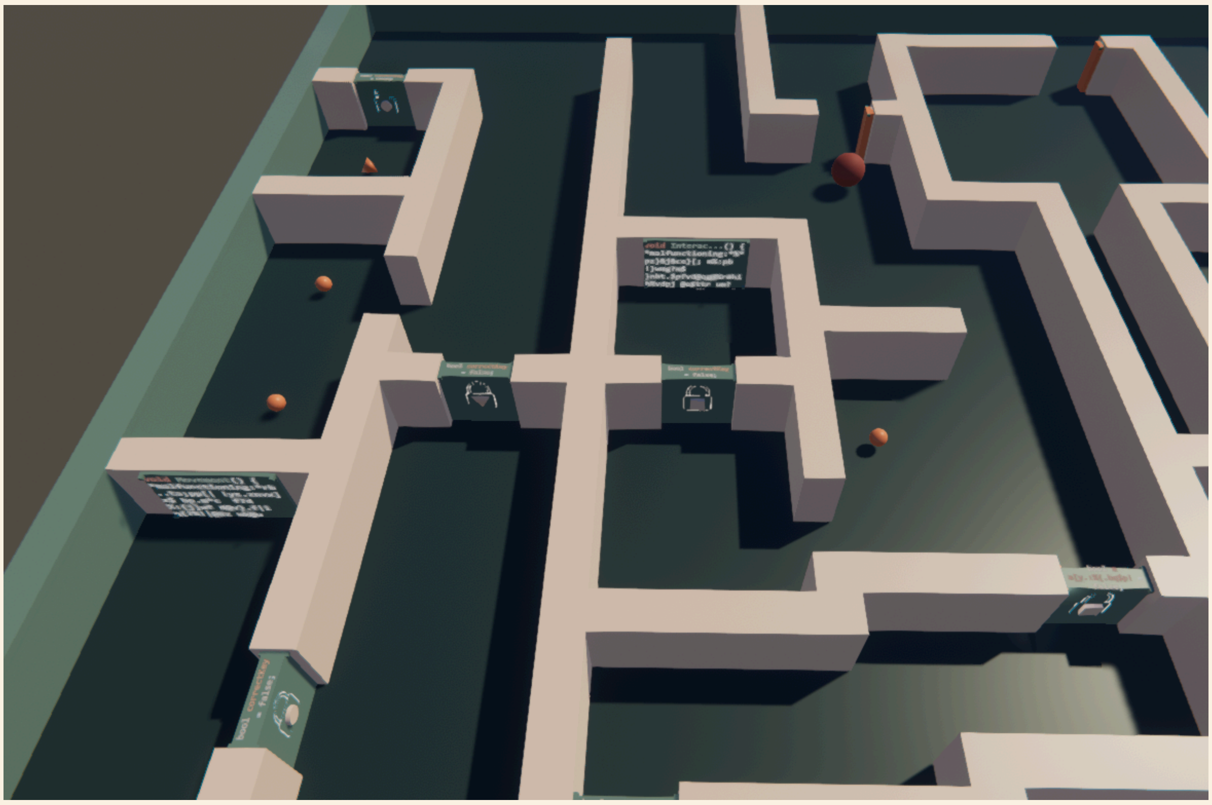
After moving through this door, the last UI element that describes game functions pops up. It lets the player know that they can move faster throughout the level if they press the 'shift' key. It works as a 'boost' function.



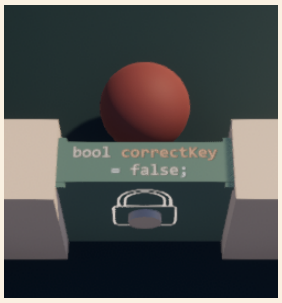
The 'Shift' key on the left side of your keyboard toggles a special movement function: you can boost to speed up your ball!

At this point, the player has completed the introductory stages of the minigame.

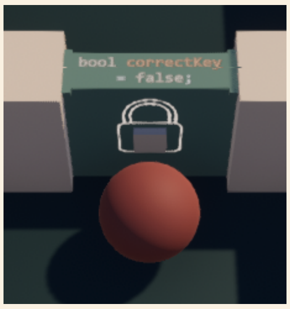
**3.5.1.3) Keys & Doors Puzzle**
The next part of the game is less linear. The player finds themselves in a larger part of the maze, with multiple doors presented to them.



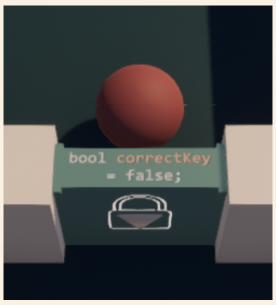
The player can use their newly unlocked boost speed to explore this part of the game.
One thing the player will notice is that the only available keys at this part of the level are in the sphere shape, whereas there are multiple doors, all with different key requirements.



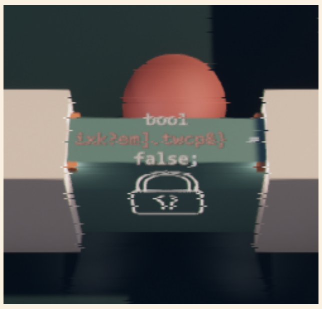
```
bool correctKey
= false;
```



```
bool correctKey
= false;
```

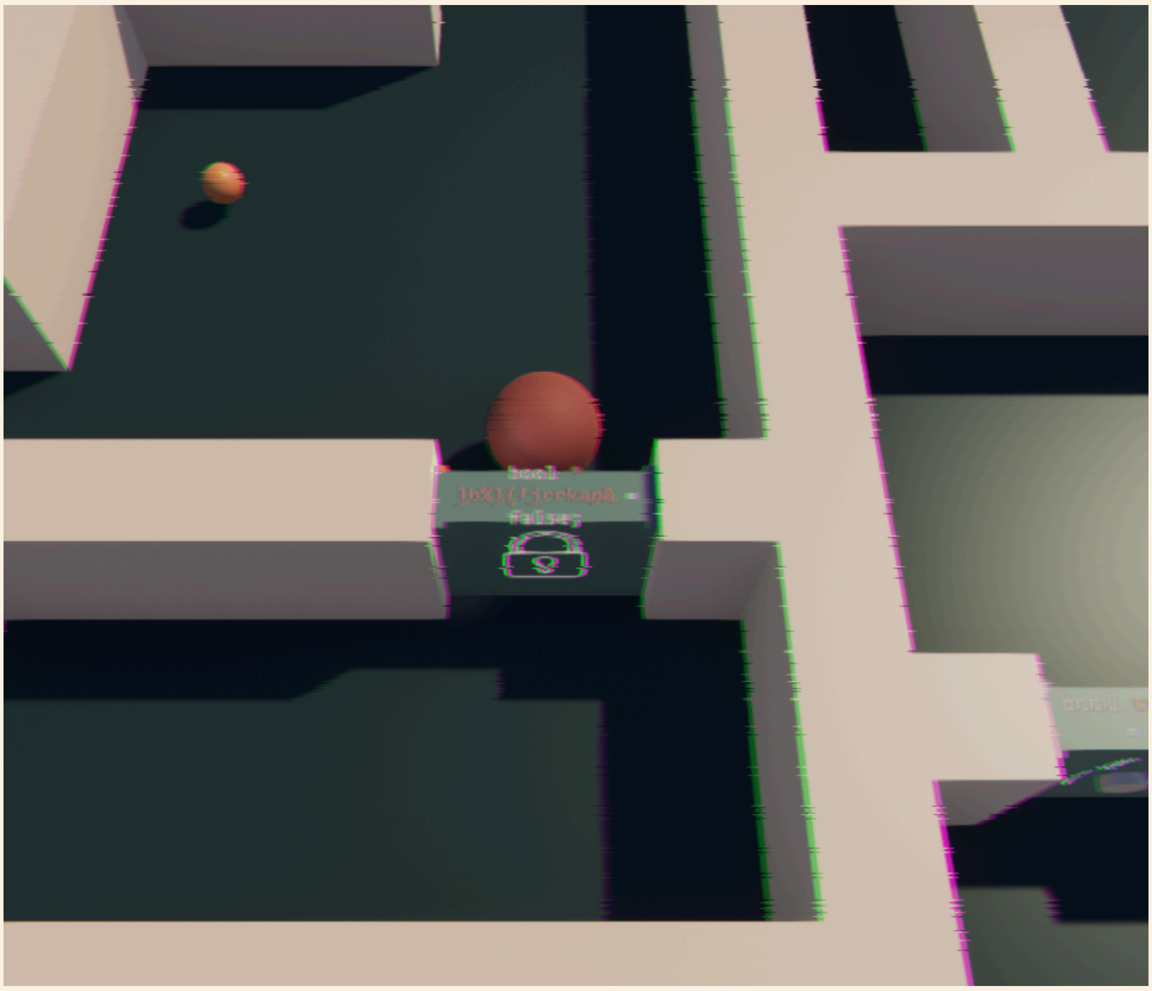


```
bool correctKey
= false;
```

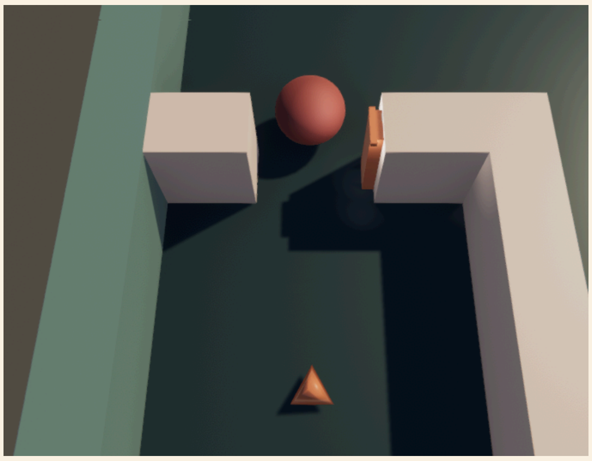


```
bool
false;
```

Additionally, the player will notice a fourth door shown above, with which something seems to be going wrong. This door causes many visual issues. The boolean text on top of the door is scrambling, the icon on the door that depicts its required key is changing seemingly randomly, and the game-screen starts shaking, colour drifting, and jittering. The player will also notice that the door is not openable with any of the keys in the scene, nor by any other method.



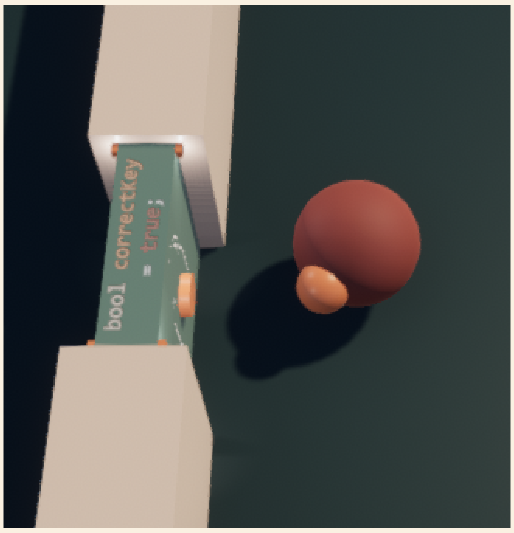
Since the only type of key available at this part of the game are sphere keys, that's what the player will pick up. Logically, the only door that can be opened is the sphere door on the northwestern side of the maze.

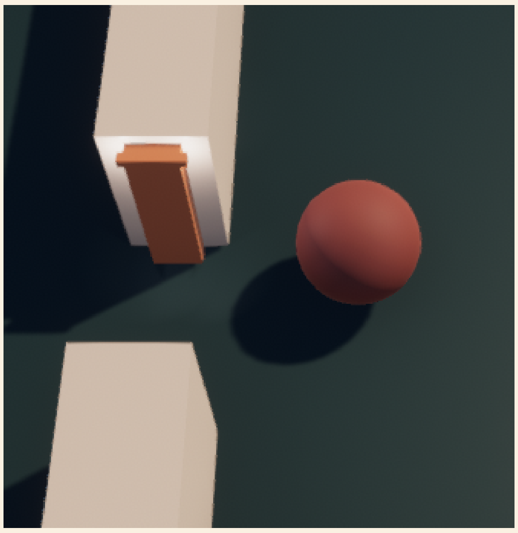


bool correctKey
= true;

Inside the small room that the player has now unlocked, they find a key in the shape that they were yet to encounter up until that point: a triangle key.
With it, naturally, the player is able to open the triangle door near the center-western part of the maze.

The last action the player will take before they ‘complete’ the first act of the game is by grabbing another one of the sphere keys lying around and opening the next door in the western corridor.





## 3.5.2) Second Stage

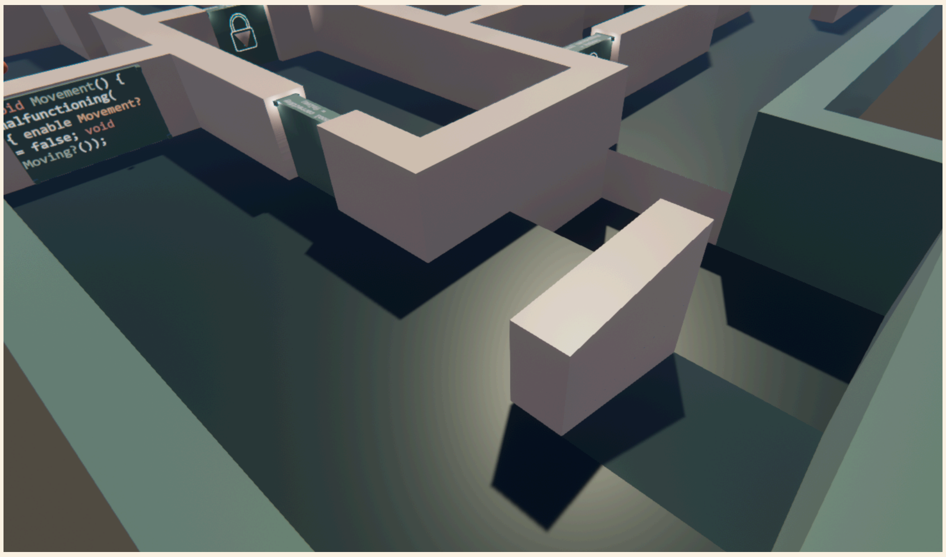
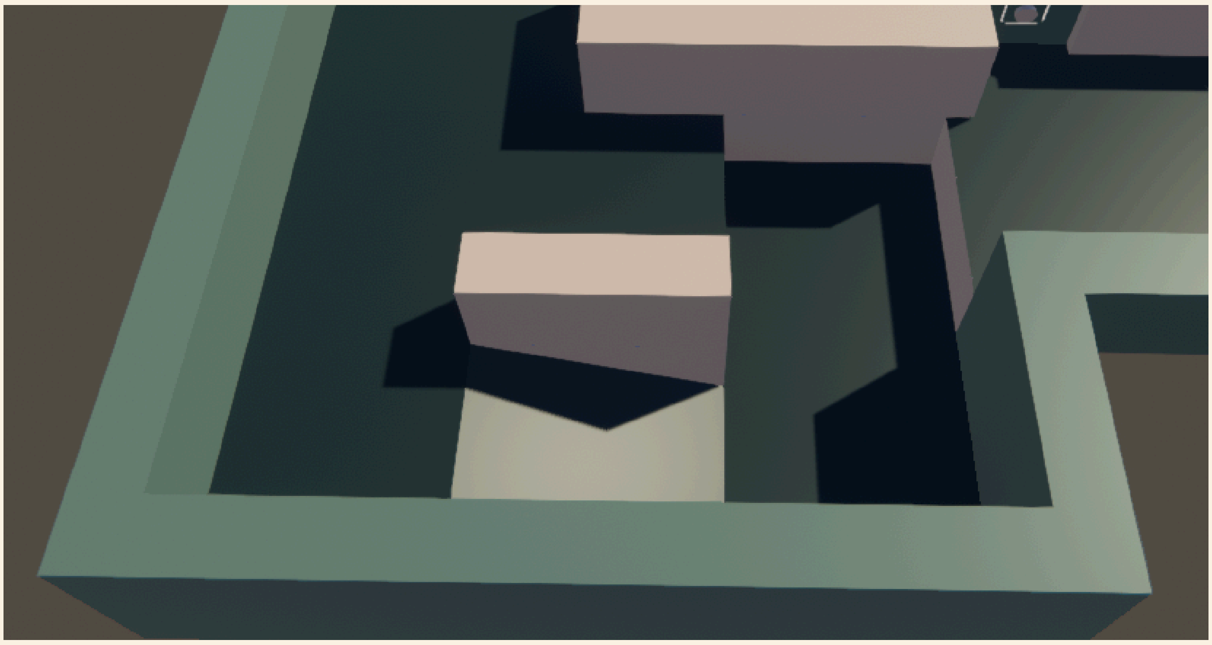


Image showing the movement screen and the gap. ^

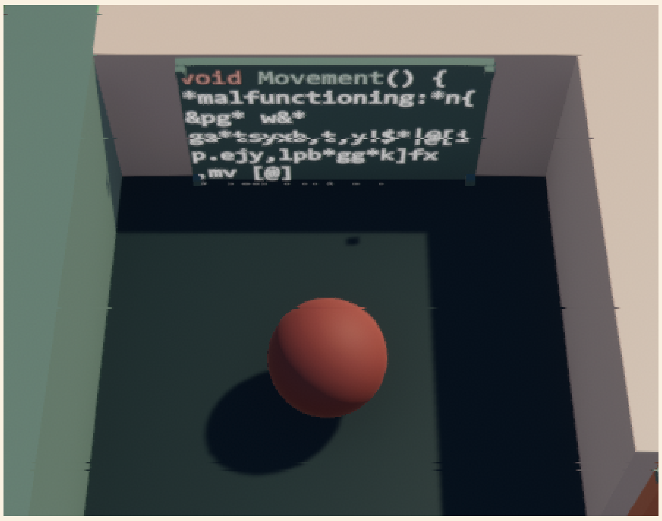
**3.5.2.1) Dev UI - Movement & The Gap**

At this point of the game, the player has started the second stage of the game. The player has now gotten the hang of the base game controls and the base puzzle system with the keys and doors. There were a few doors in the maze for which they didn’t have the key to open them, mainly a square door near the center of the maze. There was one leftover sphere key for which there was seemingly no corresponding door. The player has also already had its first encounter with an obstacle they weren’t able to solve: the glitching door - which seemed to not respond to anything they threw at it.

When the player moves into the corridor on the southwestern side of the maze, they’ll encounter two new game elements. The main one is the gap in the floor, which completely blocks progression throughout the maze: if the player can't cross the gap, they won't be able to continue.

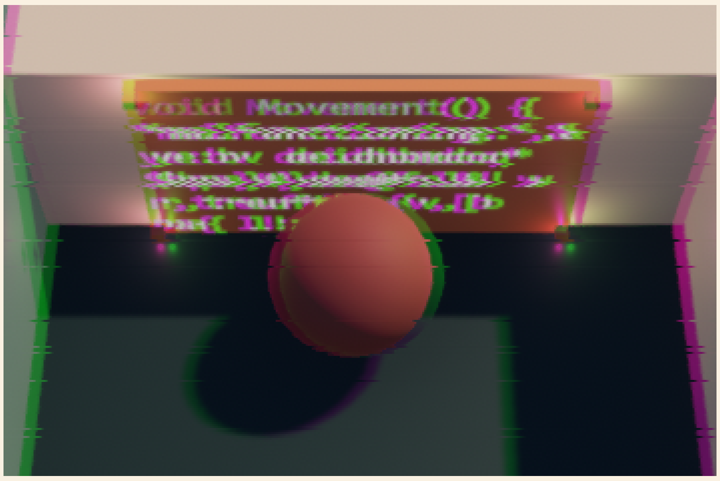


Additionally, the player will encounter a 'screen' on one of the walls of the corridor.

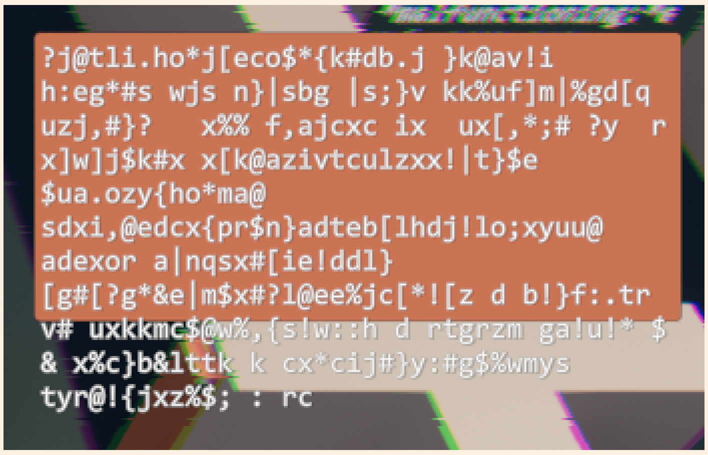


```
void Movement() {
*malfunctioning:*n{
&pg* w&*
ga*tsyxb,t,y!$*|@[i
p.ejy,lpb*gg*k]fx
,mv [@]
```

Like the glitching door, there seem to be all sorts of things wrong with this screen. The text on the screen is scrambling, and proximity to the screen causes the real game's screen to start glitching, just like the glitching door.
When the player moves very close to the screen, an interesting interaction occurs:

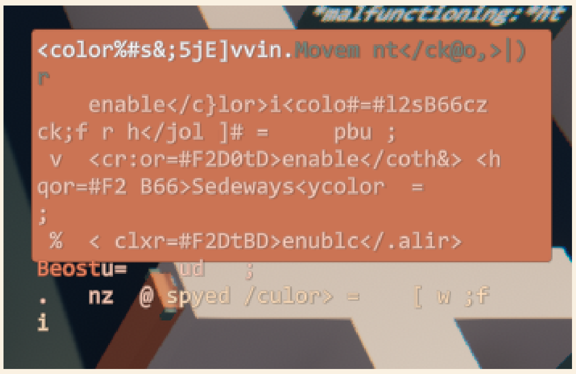


The glitching effects seem to increase to the highest possible levels, and the screen turns orange - and the lights surrounding the screen turn on as well, just as the lights on a door would if the player is holding the correct key and is close to said door.
Moreover, on the players user interface, something very weird happens:

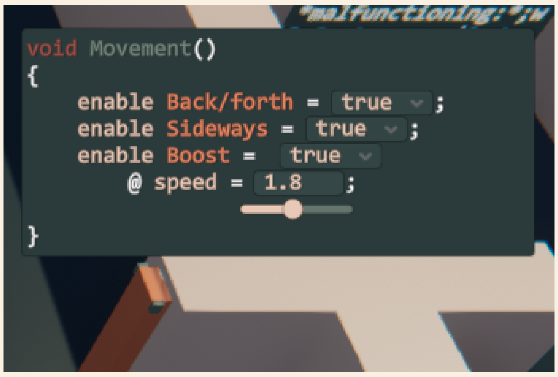


```
?j@tli.ho*j[eco$*{k#db.j }k@av!i
h:eg*#s wjs n}|sbg |s;}v kk%uf]m|%gd[q
uzj,#}?   x%% f,ajcxc ix  ux[,*;# ?y  r
x]w]j$k#x x[k@azivtculzxx!|t}$e
$ua.ozy{ho*ma@
sdxi,@edcx{pr$n}adteb[lhdj!lo;xyuu@
adexor a|nqsx#[ie!ddl}
[g#[?g*&e|m$x#?l@ee%jc[*![z d b!}f:.tr
v# uxkkmc$@w%,{s!w::h d rtgrzm ga!u!* $
& x%c}b&lttk k cx*cij#}y:#g$%wmys
tyr@!{jxz%$; : rc
```

On the corner of the players screen, this panel pops up, with scrambling, ineligible text.
The player can interact with the screen now, which causes the physical screen to be consumed and disappear, and the text on the panel to slowly unscramble - revealing the text which lied hidden behind it:



```
<color%#s&;5jE]vvin.Movem nt</ck@o,>|)
r
    enable</c}lor>i<colo#=#l2sB66cz
ck;f r h</jol ]# =      pbu ;
 v  <cr:or=#F2D0tD>enable</coth&> <h
qor=#F2 B66>Sedeways<ycolor  =
;
 %  < clxr=#F2DtBD>enublc</.alir>
Beostu=    ud    ;
.   nz  @ spyed /culor> =     [ w ;f
i
```

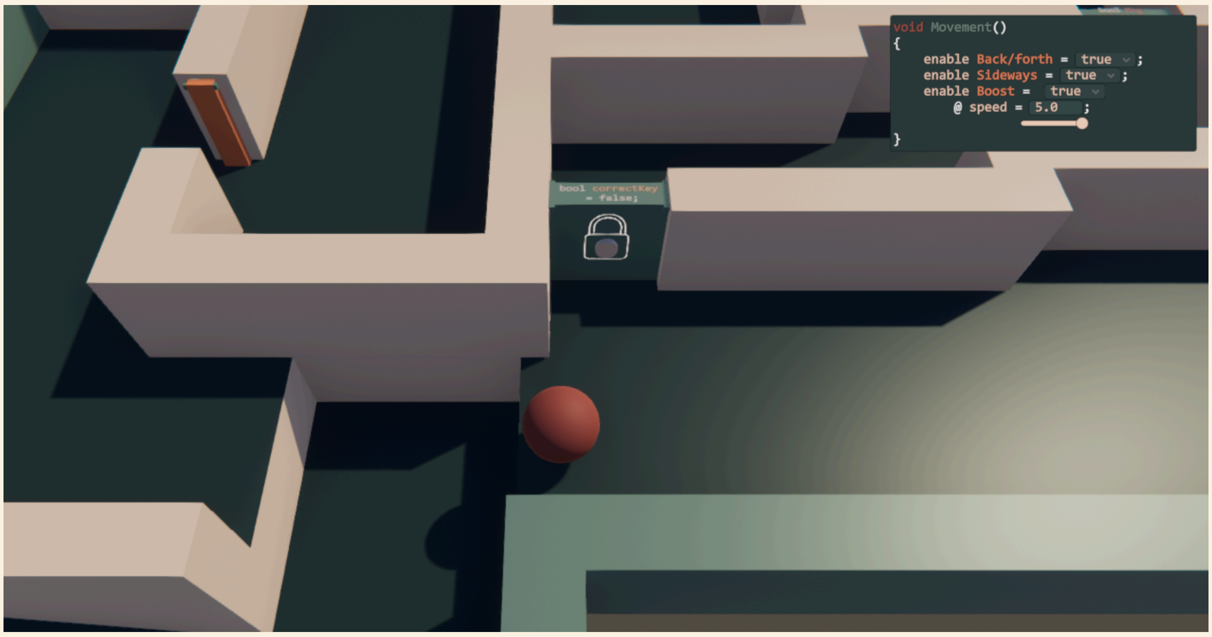


```
void Movement()
{
    enable Back/forth = true ;
    enable Sideways = true ;
    enable Boost = true
        @ speed = 1.8 ;
}
```

What lies beneath the ineligible, scrambled text is pseudocode. If the player tries to interact with the screen by clicking on the dropdown menus, they'll notice that they can change the variables that are described in the pseudocode.

If they then try to move around with the changed variables, it seems that the changes they made do have effect! If they disable back/forth or sideways, then their ball will be unable to move in those respective directions!
While those changes are not exactly helpful with the situation the player is in, the Boost Speed variable can be. If the player slides the slider up to the maximum, then their boost speed is significantly increased. Additionally, crafty players might figure out that the input field above the slider is also adjustable. Players can enter any value they desire, and completely break the game with it.

If the player chooses not to break the game, but rather set the boost speed to a reasonable, but higher setting - from 4.0 to around 10.0 works best - then they'll be able to solve the challenge of crossing the gap:
With a boost speed as mentioned above, the player can simply boost over the gap, clearing this first 'developer only' obstacle.

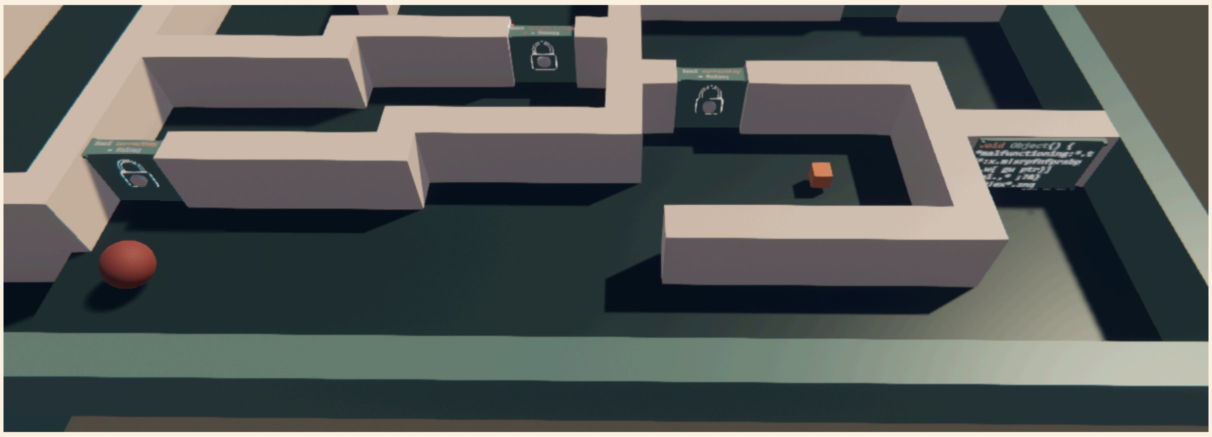


```
void Movement()
{
    enable Back/forth = true ;
    enable Sideways = true ;
    enable Boost =  true
        @ speed = 5.0 ;
}
```

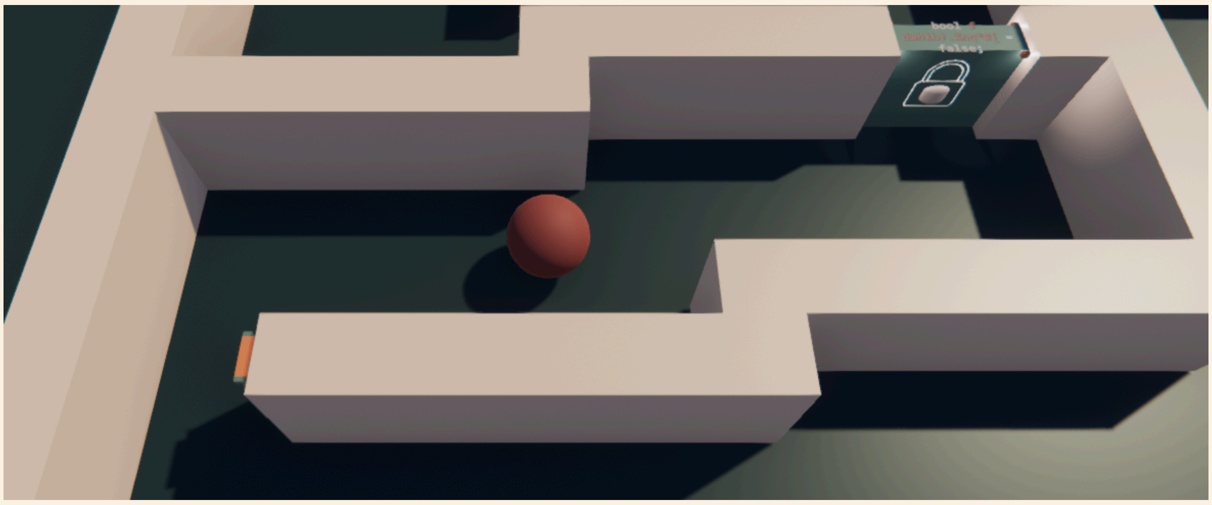
```
bool correctKey
  = false;
```

### 3.5.2.2) Dev UI - Objects & The Broken Door

After the player crosses the gap, they can explore the southern corridor of the maze.



The first door on the western side of this corridor can be opened with one of the sphere keys that was left in excess in the previous section of the game.

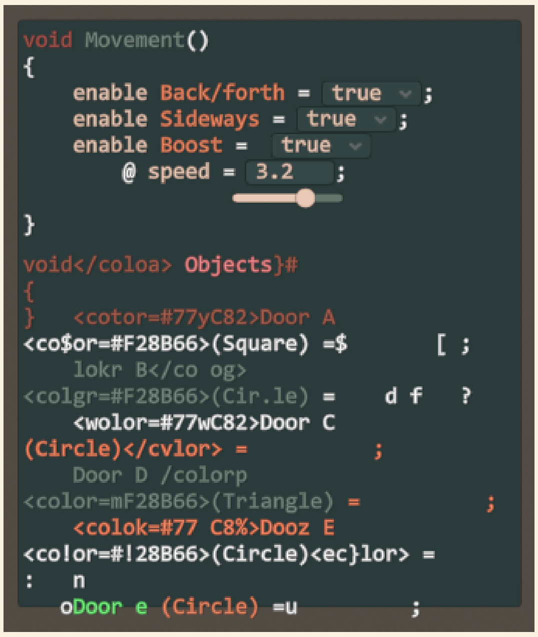


Upon doing that though, the player once again comes face to face with the broken door - still unable to be opened.

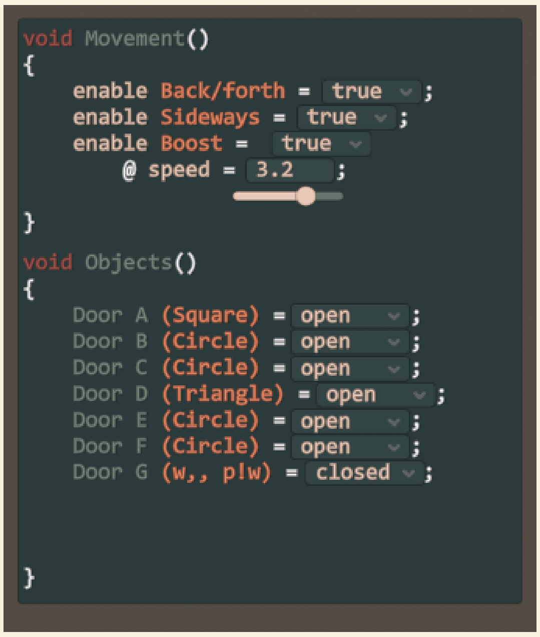
The player might recognize at this point that the only way to get past this broken door is by employing a similar strategy as with the gap: they need to access the games variables and change the doors' status directly.

The second malfunctioning screen of the eastern side of the corridor is exactly where they will find that possibility.

Upon interacting with it, the developer access UI is expanded, and unscrambling reveals the following pseudocode:

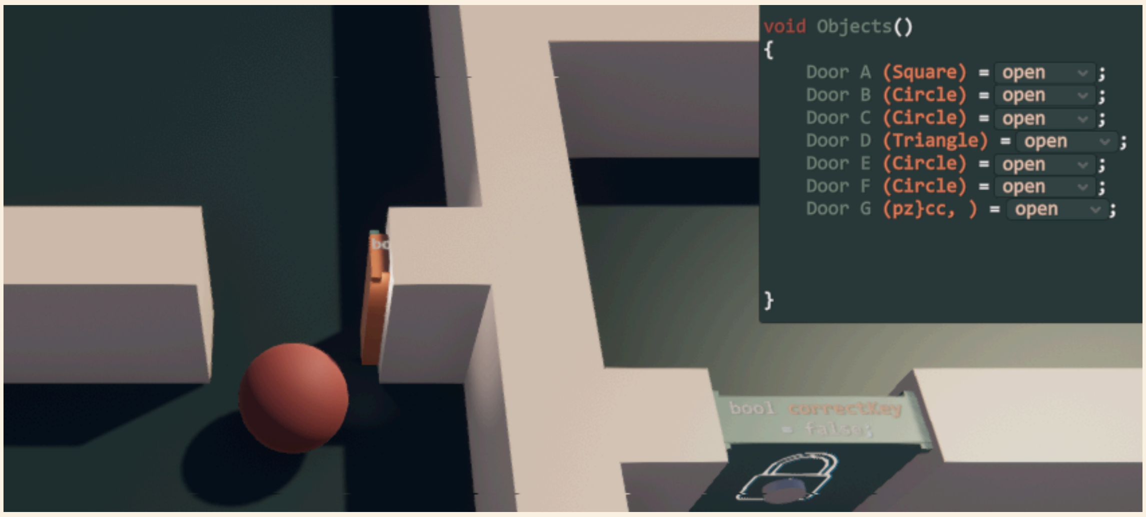


```
void Movement()
{
    enable Back/forth = true ;
    enable Sideways = true ;
    enable Boost = true
        @ speed = 3.2 ;

}

void</coloa> Objects}#
{
}   <cotor=#77yC82>Door A
<co$or=#F28B66>(Square) =$        [ ;
    lokr B</co og>
<colgr=#F28B66>(Cir.le) =    d f   ?
    <wolor=#77wC82>Door C
(Circle)</cvlor> =          ;
    Door D /colorp
<color=mF28B66>(Triangle) =           ;
    <colok=#77 C8%>Dooz E
<co!or=#!28B66>(Circle)<ec}lor> =
:   n
   oDoor e (Circle) =u         ;
```



```
void Movement()
{
    enable Back/forth = true ;
    enable Sideways = true ;
    enable Boost = true
        @ speed = 3.2 ;

}

void Objects()
{
    Door A (Square) = open ;
    Door B (Circle) = open ;
    Door C (Circle) = open ;
    Door D (Triangle) = open ;
    Door E (Circle) = open ;
    Door F (Circle) = open ;
    Door G (w,, p!w) = closed ;

}
```

The player will notice that Door G is closed, and has some interesting scrambled text describing the door type. If the player sets the variable to 'open', then the broken door will open up, and no screen glitching will occur if the player comes near it.



```
void Objects()
{
    Door A (Square) = open ;
    Door B (Circle) = open ;
    Door C (Circle) = open ;
    Door D (Triangle) = open ;
    Door E (Circle) = open ;
    Door F (Circle) = open ;
    Door G (pz}cc, ) = open ;

}
```

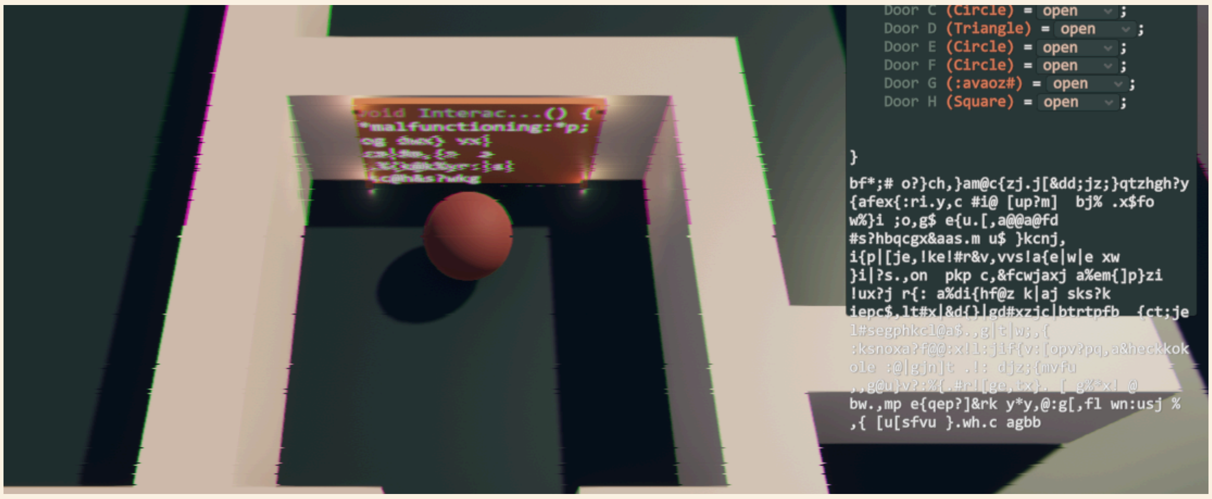
The player has solved the second 'developer only' obstacle.

### 3.5.2.3) Dev UI - Interaction & Reclaiming Keys

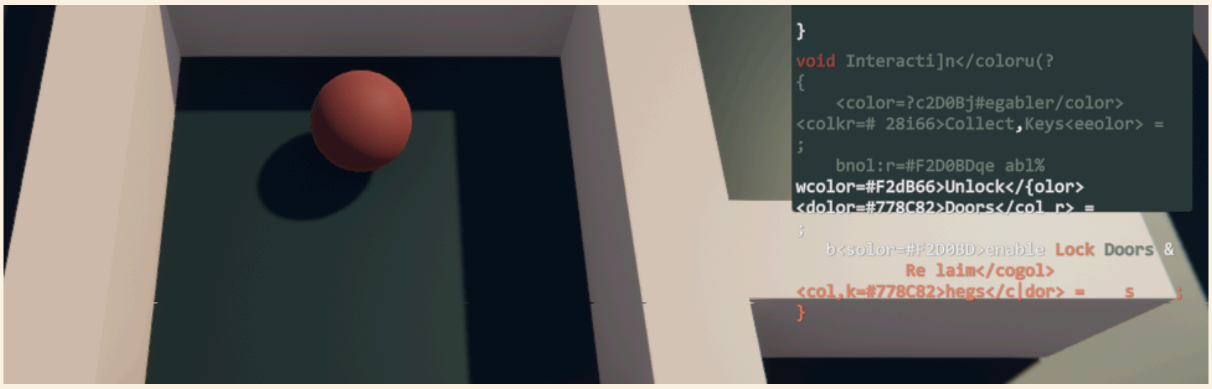
The next problem the player encounters is quite straightforward. There are two more doors left in the scene that need to be opened, but there are no more keys left in the scene.

Once again, this is an issue that can only be solved by fidgeting with the backend of the game through the developer user interface panel.

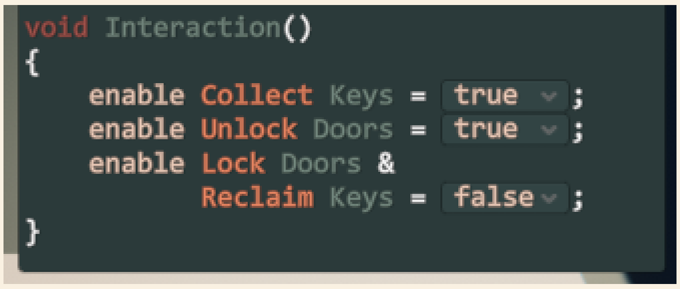
The next section of the Dev UI is unlocked by interacting with the screen that refers to ‘interaction’.



```
    Door C (Circle) = open    ;
    Door D (Triangle) = open    ;
    Door E (Circle) = open    ;
    Door F (Circle) = open    ;
    Door G (:avaoz#) = open    ;
    Door H (Square) = open    ;


}
bf*;# o?}ch,}am@c{zj.j[&dd;jz;}qtzhgh?y
{afex{:ri.y,c #i@ [up?m]  bj% .x$fo
w%}i ;o,g$ e{u.[,a@@a@fd
#s?hbqcgx&aas.m u$ }kcnj,
i{p|[je,!ke!#r&v,vvs!a{e|w|e xw
}i|?s.,on  pkp c,&fcwjaxj a%em{]p}zi
!ux?j r{: a%di{hf@z k|aj sks?k
iepc$,lt#x|&d{}|gd#xzjc|btrtpfb  {ct;je
l#segphkcl@a$.,g|t|w;,{
:ksnoxa?f@@:x!l:jif{v:[opv?pq,a&heckkok
ole :@|gjn]t .!: djz;{mvfu
,,g@u}v?:%{.#r![ge,tx}. [ g%*x! @
bw.,mp e{qep?]&rk y*y,@:g[,fl wn:usj %
,{ [u[sfvu }.wh.c agbb
```



```
}
void Interacti]n</coloru(?
{
    <color=?c2D0Bj#egabler/color>
<colkr=# 28i66>Collect,Keys<eeolor> =
;
    bnol:r=#F2D0BDqe abl%
wcolor=#F2dB66>Unlock</{olor>
<dolor=#778C82>Doors</col r> =
;
   b<solor=#F2D0BD>enable Lock Doors &
           Re laim</cogol>
<col,k=#778C82>hegs</c|dor> =    s    ;
}
```



```
void Interaction()
{
    enable Collect Keys = true  ;
    enable Unlock Doors = true  ;
    enable Lock Doors &
           Reclaim Keys = false ;
}
```

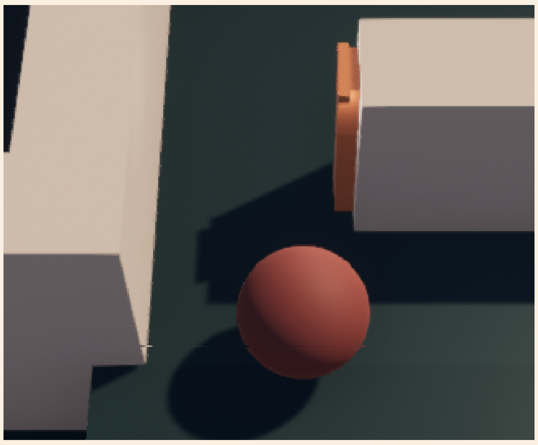
Three game interaction variables show themselves, and they can be enabled and disabled.

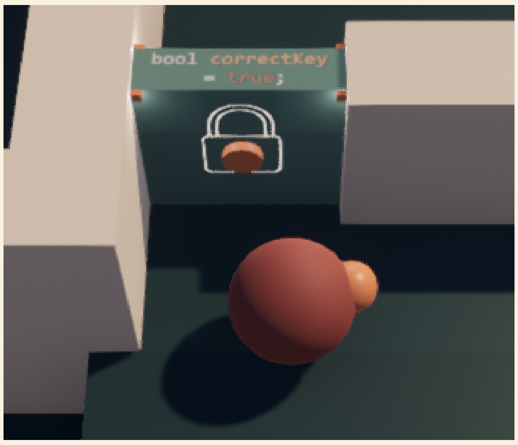
Collect Keys refers to the ability of the player to pick up keys, hold them, and drop them on the floor in a different location in the scene.

Unlock Doors is even more straightforward than the previous interaction: it refers to the player's ability to open doors with corresponding keys.

The Lock Doors & Reclaim Keys function sticks out. It is disabled by default.

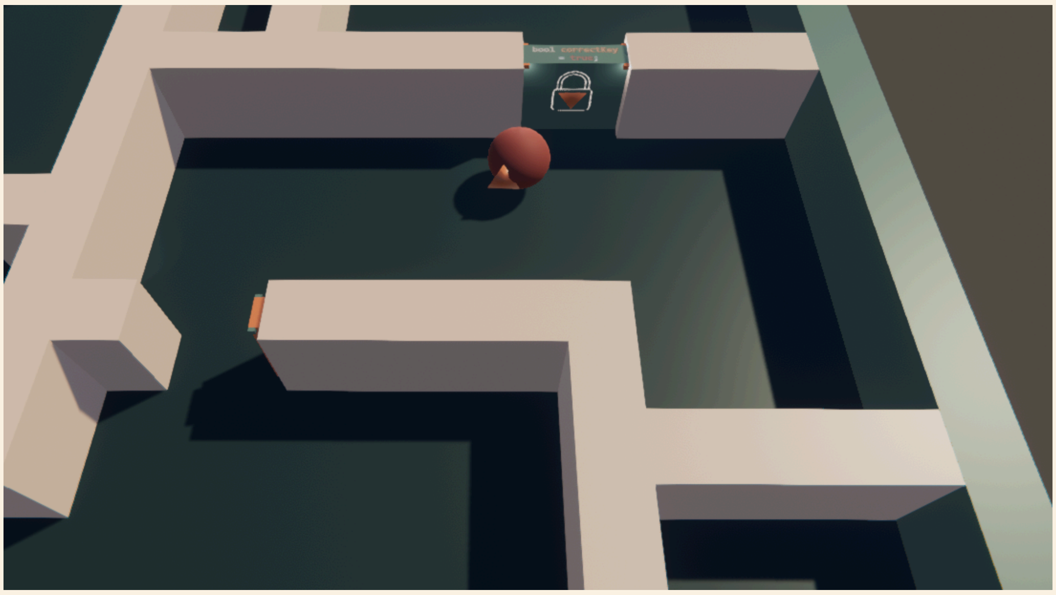
If the player enables this interaction and reads the function text properly, they’ll be able to figure out that they can now move to a door they had previously opened, and now re-close that door. Doing that reclaims the key that was used to open that door, and the key is stored as the player's new held item.





```
bool correctKey
= true;
```

With this new power, the player is able to open the last two doors by reclaiming the two needed keys: one sphere key and one triangle key.



By doing this, the player has completed the last ‘developer only’ obstacle.
By making use of programming related, game altering, functional pseudocode, the player has been able to traverse through the maze in its entirety. They have, at this point, effectively completed all the challenges found in the minigame. And still, there is one more ‘secret’ to unlock. It is found when the player interacts with the final malfunctioning screen, which they encounter as soon as they enter the final room of the maze.

## 3.5.3) Final Stage

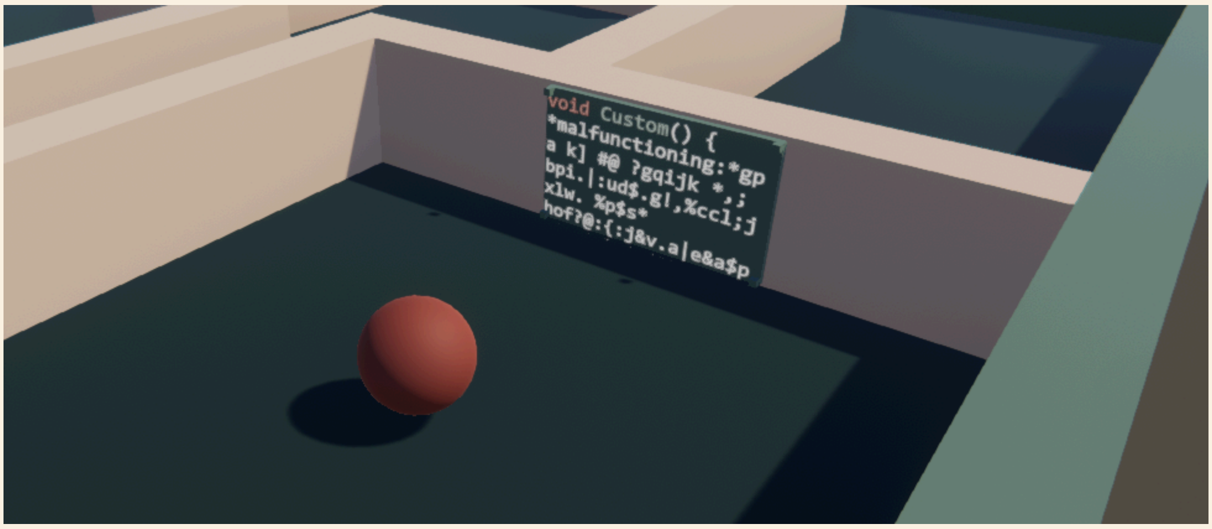
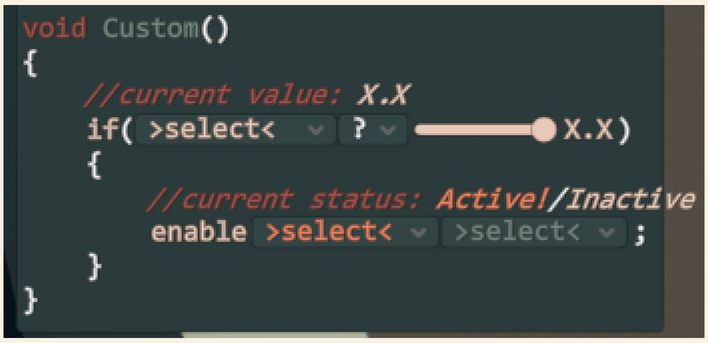


Image showing the custom screen in the final room. ^

**3.5.3.1) Dev UI - The Custom If-Statement**

Upon interacting with the final screen in the maze, the Dev UI is expanded one last time. This time however, the pseudocode that appears takes form in an unfamiliar way:

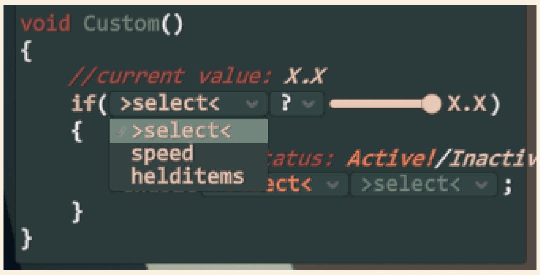


```
void Custom()
{
    //current value: X.X
    if( >select< ? X.X)
    {
        //current status: Active!/Inactive
        enable >select< >select< ;
    }
}
```

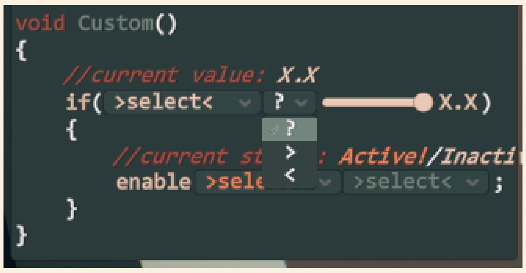
Instead of the singular variables attached to simple object or interaction descriptions, this piece of pseudocode contains many more variables that the player can adjust.

Players familiar with programming will notice that the structure of the code is very reminiscent of what is called an ‘if-statement’. If-statements are used in programming code to perform conditional checks, and to provide responses to those checks. If-statements are one of the most important programming functions - in all disciplines.

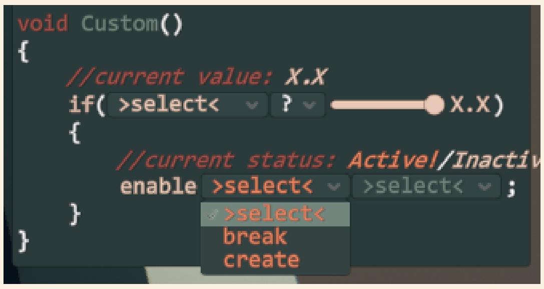
The player can fidget with the options. Here are all the options the player has:



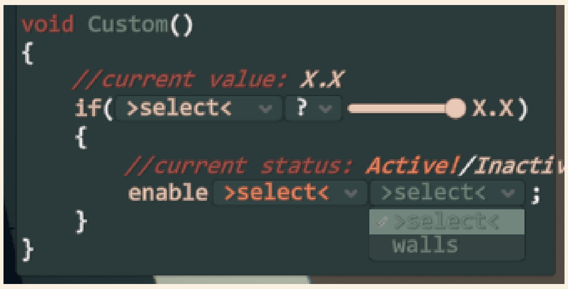
```
void Custom()
{
    //current value: X.X
    if( >select< ⌄ ? ⌄ X.X)
    {           tatus: Active!/Inactiv
                ect< ⌄ >select< ⌄ ;
    }
}
```



```
void Custom()
{
    //current value: X.X
    if( >select< ⌄ ? ⌄ X.X)
    {
        //current st   : Active!/Inacti
        enable >sele    ⌄ >select< ⌄ ;
    }
}
```

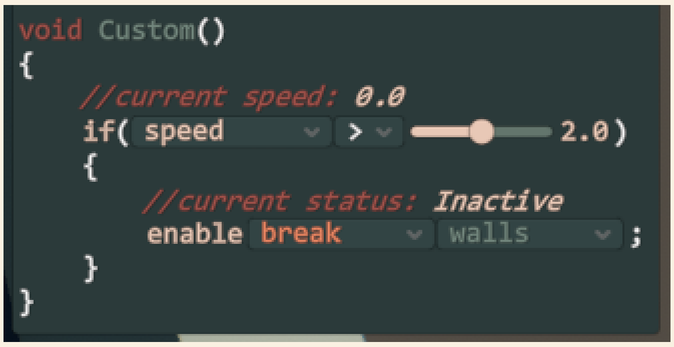


```
void Custom()
{
    //current value: X.X
    if( >select< ⌄ ? ⌄ X.X)
    {
        //current status: Active!/Inactiv
        enable >select< ⌄ >select< ⌄ ;
    }
}
```



```
void Custom()
{
    //current value: X.X
    if( >select< ⌄ ? ⌄ X.X)
    {
        //current status: Active!/Inactiv
        enable >select< ⌄ >select< ⌄ ;
    }
}
```

In real if-statements, there are no dropdowns. A developer has full freedom to write whatever conditional check they deem appropriate for their if-statement. In this program, that is not possible. Therefore, a few dropdown options are used, limiting the options of this if-statement to ones that can actually be executed in this minigame.
The *//comments* in the if-statement are used to help the player understand what kind of data they're currently checking with their if-statement, and show whether or not their newly enabled custom function is active or not.
After some fidgeting, the player will likely come up with a configuration that looks somewhat like this:

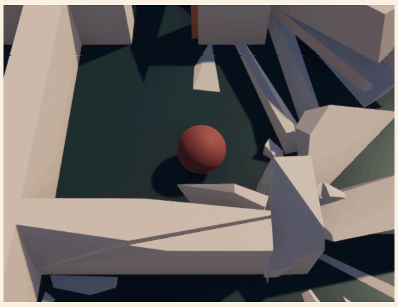


```
void Custom()
{
    //current speed: 0.0
    if( speed ⌄ > ⌄ 2.0)
    {
        //current status: Inactive
        enable break ⌄ walls ⌄ ;
    }
}
```

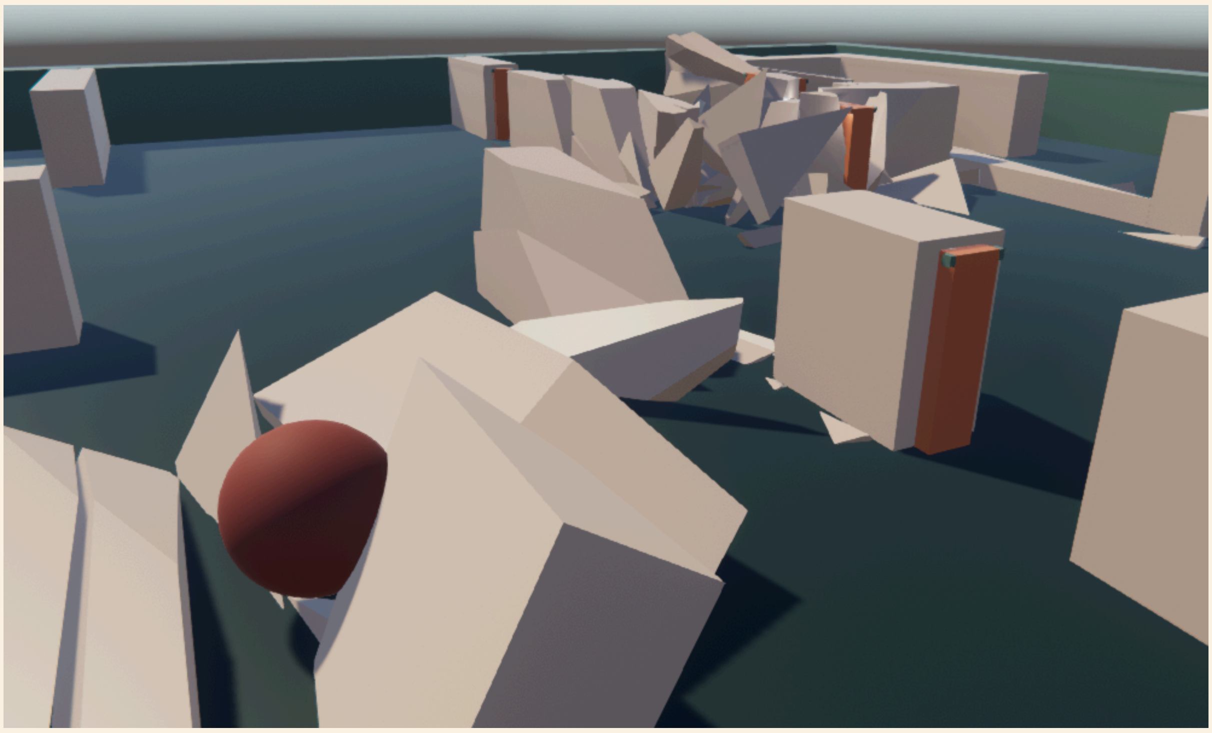
Reading this statement out loud gives:
Condition line: *"If the speed of the player is higher than 2.0 'metres' per second,"*
Execution line: *"Then, enable the functionality of breaking walls."*

And that line does exactly what the player will expect it to do.

### 3.5.3.2) Breaking Walls

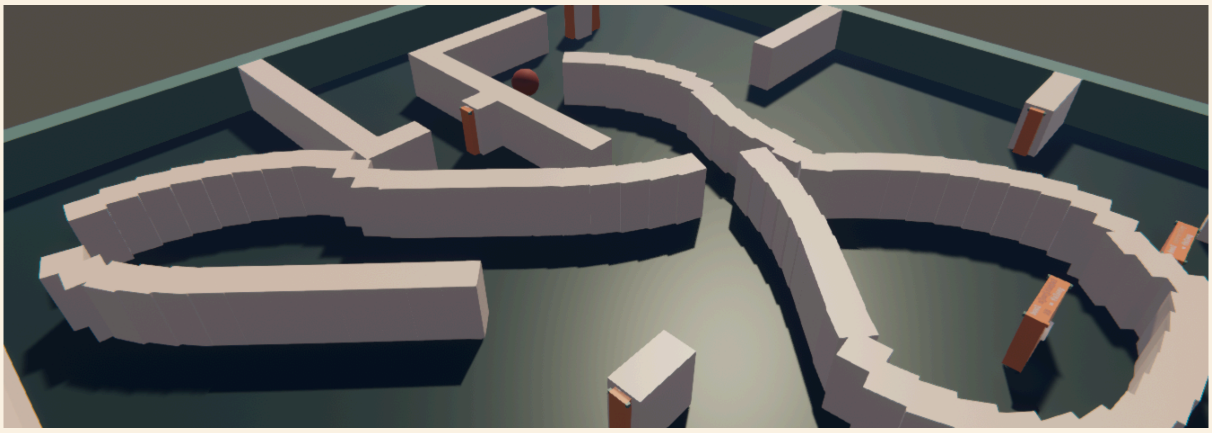


If the if-statement conditional is active, and the execution functionality chosen is breaking walls, then the player can smash through walls.
If a wall is broken, it no longer has collision, and it shatters into smaller, randomly generated fragments.



After a few seconds, the fragments drop through the floor, and are fully destroyed and removed from the scene.

**3.5.3.3) Creating Walls**



The opposite of destruction is also possible. If the ‘create walls’ functionality is enabled, then the player will spawn small, half a meter by half a meter walls behind themselves as they move. The player can play around a bit, and use this functionality to ‘create’ the foundation of their own level.
Created walls can also be destroyed, should the player change the enabled functionality to destroy walls once again.

**3.5.3.4) Other Possible Functions**

As the player has fun finishing the game by destroying the maze and making a design of their own, they might wonder what other functionality this if-statement could offer. While the creation and destruction of walls are currently the only functions that are made for this minigame, other options could include the creation and destruction of doors, keys, and perhaps even screens. Additionally, a function that can be enabled with the custom if-statement could be a jumping function. This can open up many new layers to the game, making the game not just two dimensional in the maze's shape, but three dimensional by adding height and platforms to the game.

As with a real if-statement, the only limits are those of the developer's imagination.

# Four *Iterative Timeline*

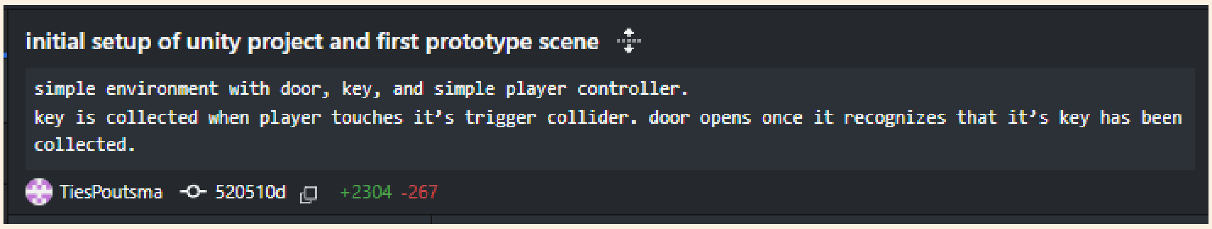
## **4.1)** Initial Prototyping

### **4.1.1)** Base Game Concept

As explained in the ideation section, the base game concept is a three dimensional puzzle game. In the very first stages of prototyping, this concept was used to guide the creation of the very first gameplay elements. Using simple interactions between a rollable ball as a player and keys to unlock corresponding doors was the first idea for the prototype - it stemmed from the ‘strategy-game’ core game concept. A rollable ball was chosen, as the direct control over a ‘player’ object was theorised to be more interactive and engaging than a game where the only interaction is clicking and dragging on elements in the screen. The integration of the programming aspect was not fully defined at all at this point of the iterative process. There was, however, already an idea of having the integrating mechanic be a 'game-within-a-game’. The base puzzle game would be limited in its space, with a ‘larger’, overarching game-mechanic that would have influence over the smaller game within.

### **4.1.2)** Base Functionality

At this point in the timeline, the first task at hand was transitioning from the background research to an actual product concept. After building a base unity project and setting up a github system to upload and sync iterations, the first actions made for the minigame itself were the addition of a floor, a simple room, and a rollable player ball.
Next up, a door and key were made. For a long time in the iterative process, the keys were not distinguished by shapes, but by colours. Additionally, there was exactly one key for each door. It was possible to pick up keys, and drop them in front of the player.


**initial setup of unity project and first prototype scene**

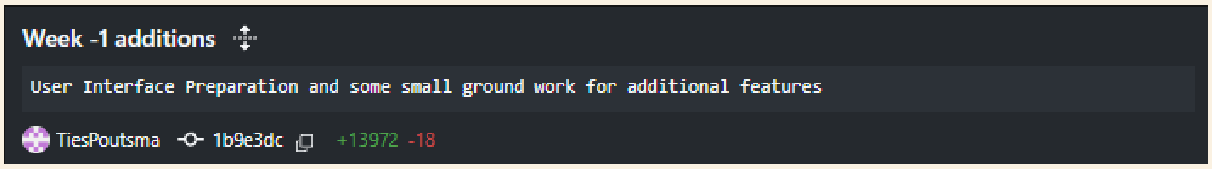
```
simple environment with door, key, and simple player controller.
key is collected when player touches it's trigger collider. door opens once it recognizes that it's key has been
collected.
```

TiesPoutsma 520510d +2304 -267

In this initial setup, doors were openable by proximity only. A door could only be opened if the player was holding the correct key near it, and if that was the case, the door would open automatically.

### **4.1.3)** Prototype Developer User Interface

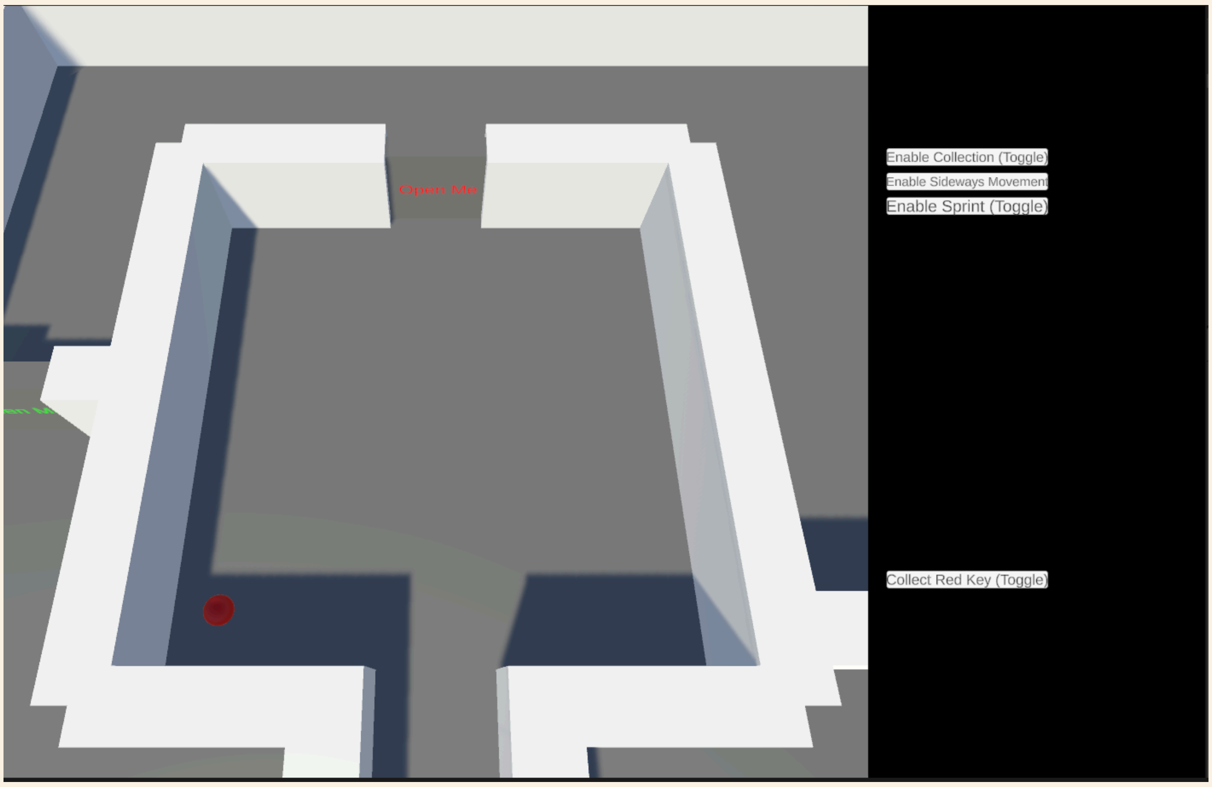
The next part of the initial prototyping was developing the prototype of the Developer User Interface. The ‘overarching’ game-space that was originally ideated, came to be a sort of ‘developer-only’ setting. A function of the game that could control aspects of the inner game in a way that would seem ‘unnatural’ by the standards of the inner game, but possible in the greater game.


Week -1 additions

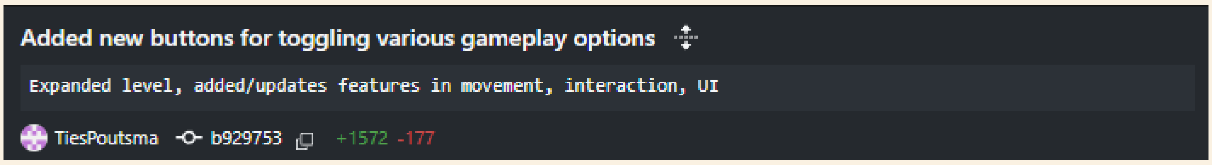
```
User Interface Preparation and some small ground work for additional features
```

TiesPoutsma 1b9e3dc +13972 -18

It was realised in the Unity project by casting a large black panel over the right side of the screen, leaving room for some user interface buttons that could affect the base game.



Consider the real backend programming of the game: the doors, keys, and the player itself have real programmable attributes - if *those* attributes are allowed to be adjusted by the 'developer', then that opens the door to interesting and special puzzle elements that make use of programming related styling.


Added new buttons for toggling various gameplay options

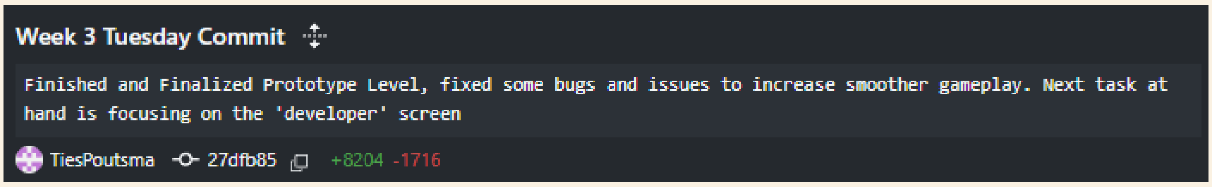
```
Expanded level, added/updates features in movement, interaction, UI
```

TiesPoutsma b929753 +1572 -177

Around 30% of the total screen was this large black bar with some buttons that influenced relevant game variables directly, such as enabling sideways movement, enabling the 'boost' function (was still called 'sprint' at that time) or directly collecting the first key in the scene.
In this initial prototype, the 'developer user interface' did not have any puzzle related elements added to it yet.

## 4.2) Level Design

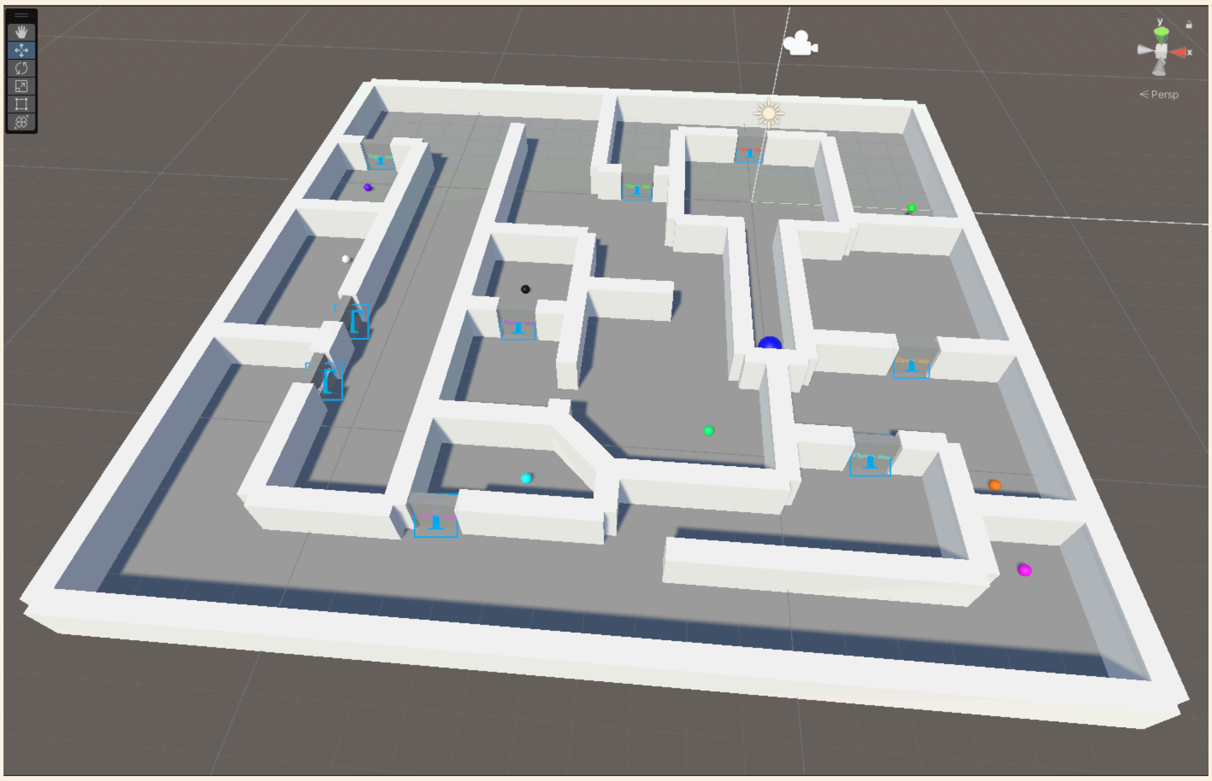
The next part of iteration was expanding the level itself.


Week 3 Tuesday Commit

```
Finished and Finalized Prototype Level, fixed some bugs and issues to increase smoother gameplay. Next task at
hand is focusing on the 'developer' screen
```

TiesPoutsma 27dfb85 +8204 -1716

The maze was expanded to be much greater than just the first room. The expanded level also featured multiple new doors and keys in various corresponding colours.

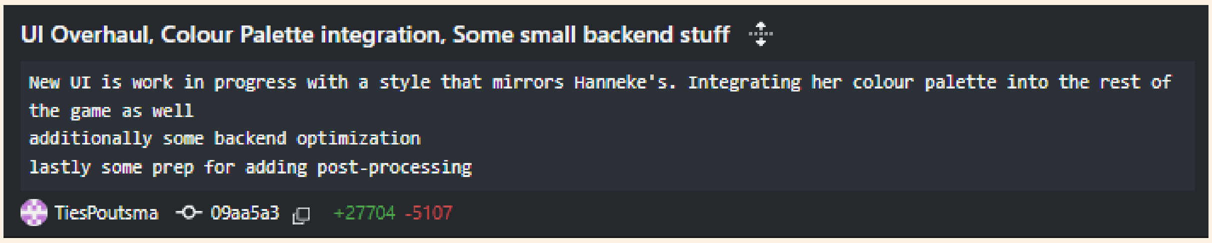


The placement of keys and doors created a simple yet fun puzzle: the maze was not linear by any means, requiring backtracking and intelligently using the position of keys to open all doors and get to the final room.
There was no special function in place if the player completed the maze in its entirety.

Throughout the development of the project, scripts and other backend functionality was revisited multiple times to ensure smooth and functional gameplay. This update included such revisions. In this case, some bugs were fixed relating to the picking up and dropping of keys, the smoothness of the animation of doors opening and closing, and some general semantics and naming styles in the scripts themselves.

## 4.3) Stylisation and Style Integration into Dev UI

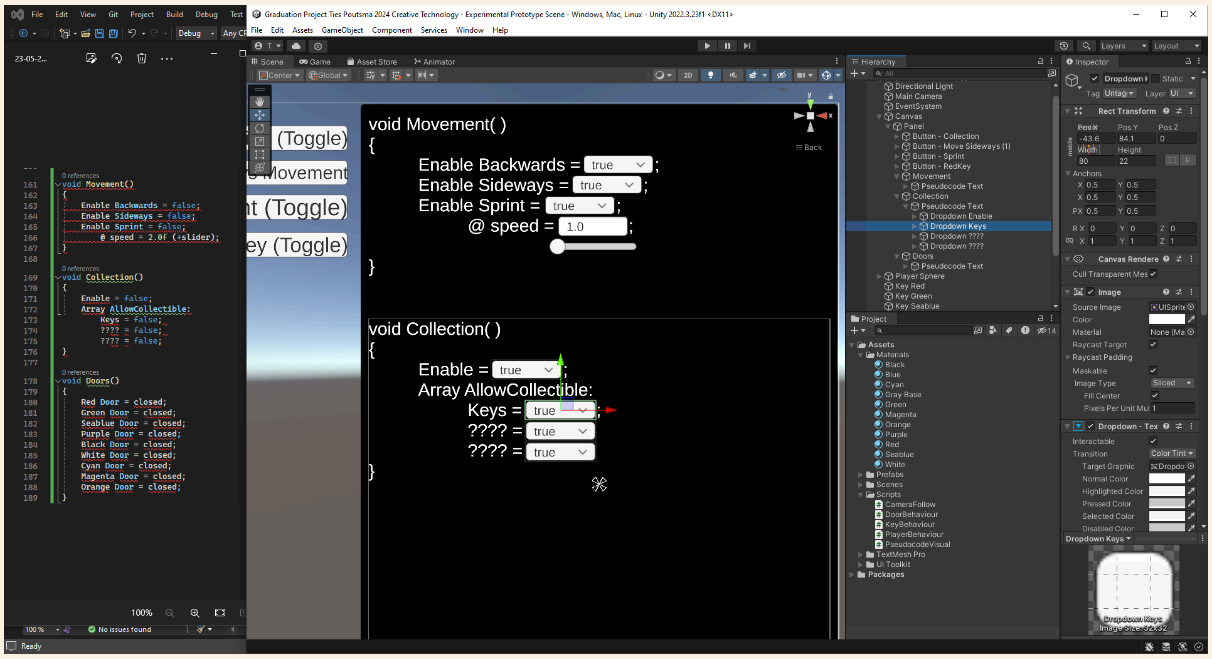
Next on the docket: a larger update that added the colour palette to the game and did some work towards making the developer user interface functional with more game mechanics.



**UI Overhaul, Colour Palette integration, Some small backend stuff**

```
New UI is work in progress with a style that mirrors Hanneke's. Integrating her colour palette into the rest of
the game as well
additionally some backend optimization
lastly some prep for adding post-processing
```

TiesPoutsma 09aa5a3 +27704 -5107



Graduation Project Ties Poutsma 2024 Creative Technology - Experimental Prototype Scene - Windows, Mac, Linux - Unity 2022.3.23f1 <DX11>

```
void Movement()
{
    Enable Backwards = false;
    Enable Sideways = false;
    Enable Sprint = false;
        @ speed = 2.0f (+slider);
}

void Collection()
{
    Enable = false;
    Array AllowCollectible:
        Keys = false;
        ???? = false;
        ???? = false;
}

void Doors()
{
    Red Door = closed;
    Green Door = closed;
    Seablue Door = closed;
    Purple Door = closed;
    Black Door = closed;
    White Door = closed;
    Cyan Door = closed;
    Magenta Door = closed;
    Orange Door = closed;
}
```

```
void Movement( )
{
    Enable Backwards = true ;
    Enable Sideways = true ;
    Enable Sprint = true ;
        @ speed = 1.0 ;
}

void Collection( )
{
    Enable = true ;
    Array AllowCollectible:
        Keys = true ;
        ???? = true
        ???? = true
}
```

The image above shows the creation of the expanded and improved Dev UI ^.

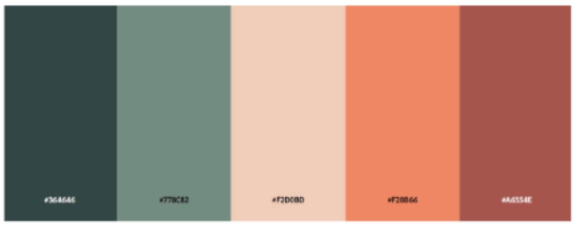


*Figure 33: Palettes for the fourth mood board - 1 of 2. LTR: #364646 #778C82 #F2D0BD #F28B66 #A6554E.*

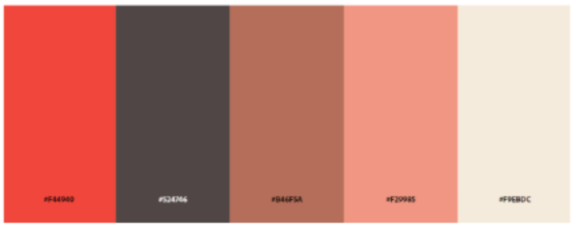
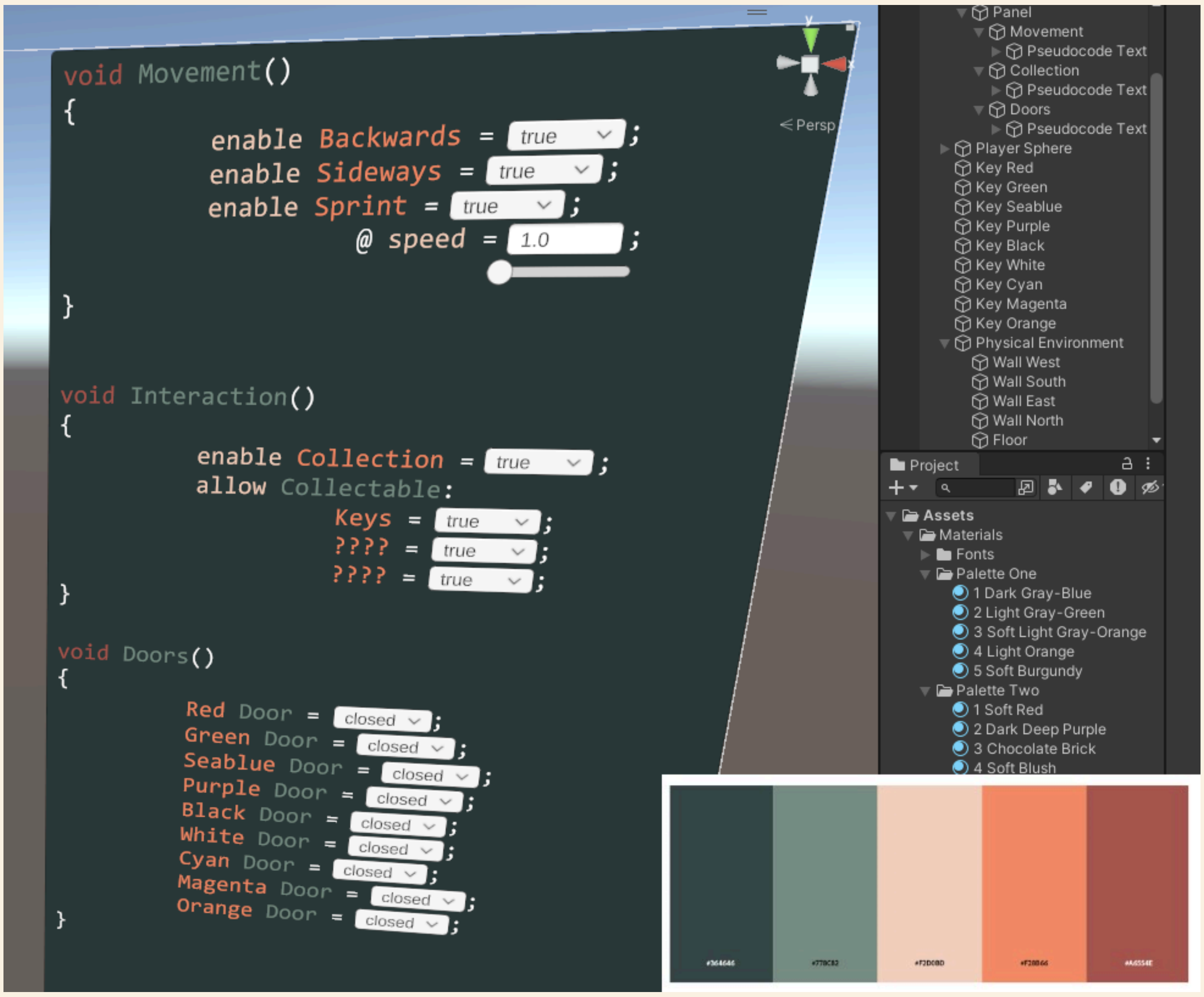


*Figure 34: Palettes for the fourth mood board - 2 of 2. LTR: #F44940 #524746 #B46F5A #F29985 #F9EBDC.*

These are the two colour palettes imported for the minigame. They come from the counterpart research paper of this project - Goud's *"Using graphic design to attract female high school students to the field of creative coding"* [13].

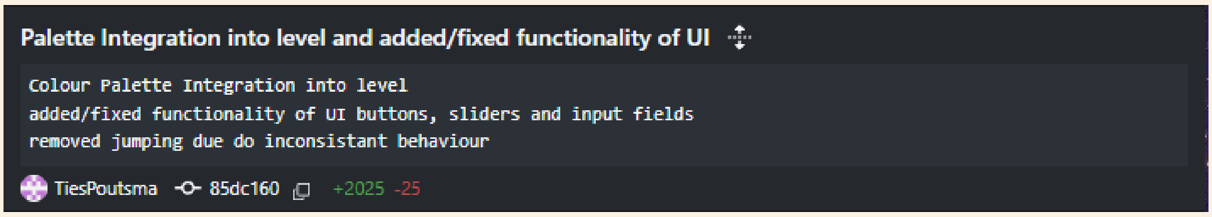
In the end, the first palette is the one that was chosen to be used. There were plans to use the second one as well, but for the sake of visual clarity, conciseness, and to avoid colour clashing, the first one was used for the whole game.

In this update, the Dev UI steps away from generic functions, and moves to a clearer structure that refers to game mechanics that were in place in the game at that time: player movement, the collection of keys, and the doors in the scene.



```
void Movement()
{
        enable Backwards = true ;
        enable Sideways = true ;
        enable Sprint = true ;
                  @ speed = 1.0 ;
}

void Interaction()
{
        enable Collection = true ;
        allow Collectable:
                  Keys = true ;
                  ???? = true ;
                  ???? = true ;
}

void Doors()
{
        Red Door = closed ;
        Green Door = closed ;
        Seablue Door = closed ;
        Purple Door = closed ;
        Black Door = closed ;
        White Door = closed ;
        Cyan Door = closed ;
        Magenta Door = closed ;
        Orange Door = closed ;
}
```

After updating the content of the text and adding the sliders, input-fields and dropdown menus, font of the text was changed to the same font as the one used in Visual Studio 2022, the main programming tool Unity uses. Additionally, the colour palette was integrated into the panel and the text itself.



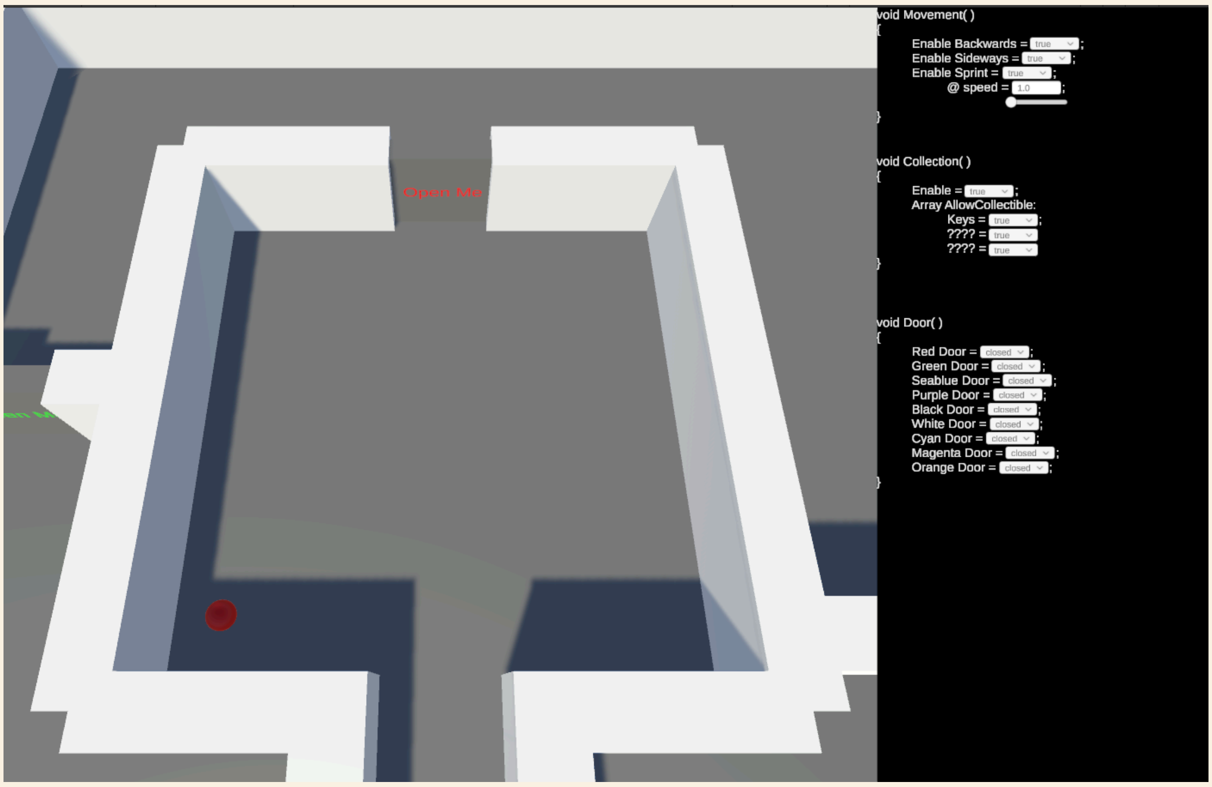
**Palette Integration into level and added/fixed functionality of UI**

```
Colour Palette Integration into level
added/fixed functionality of UI buttons, sliders and input fields
removed jumping due do inconsistant behaviour
```

TiesPoutsma 85dc160 +2025 -25

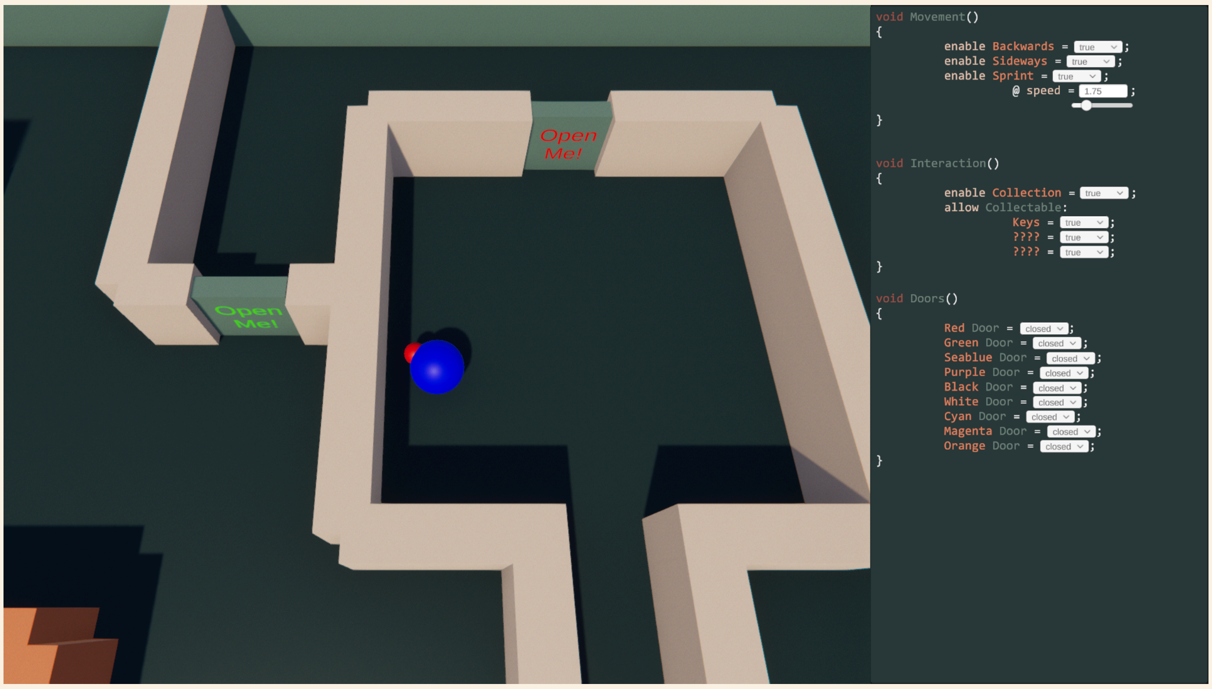
The colour palette was also integrated into the general level elements.
This is also the update where post-processing effects were added to the camera such as Ambient Occlusion, Vignetting and Colour Toning with the ACES Colour Space.

## Before:



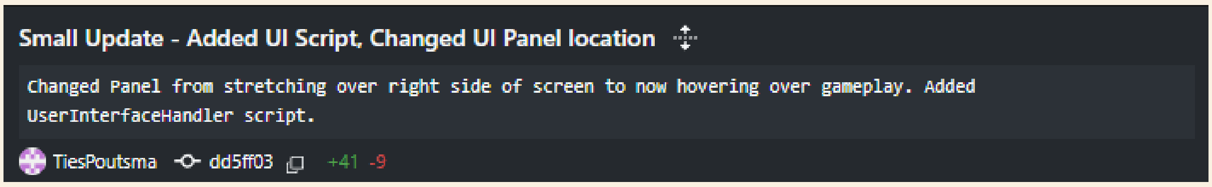
```
void Movement( )
{
        Enable Backwards = true ;
        Enable Sideways = true ;
        Enable Sprint = true ;
                @ speed = 1.0 ;
}

void Collection( )
{
        Enable = true ;
        Array AllowCollectible:
                Keys = true ;
                ???? = true
                ???? = true
}

void Door( )
{
        Red Door = closed ;
        Green Door = closed ;
        Seablue Door = closed ;
        Purple Door = closed ;
        Black Door = closed ;
        White Door = closed ;
        Cyan Door = closed ;
        Magenta Door = closed ;
        Orange Door = closed ;
}
```

## After:



```
void Movement()
{
        enable Backwards = true ;
        enable Sideways = true ;
        enable Sprint = true ;
                @ speed = 1.75 ;
}

void Interaction()
{
        enable Collection = true ;
        allow Collectable:
                Keys = true ;
                ???? = true ;
                ???? = true ;
}

void Doors()
{
        Red Door = closed ;
        Green Door = closed ;
        Seablue Door = closed ;
        Purple Door = closed ;
        Black Door = closed ;
        White Door = closed ;
        Cyan Door = closed ;
        Magenta Door = closed ;
        Orange Door = closed ;
}
```

The last step made with this update was changing the position and size of the Developer User Interface panel.

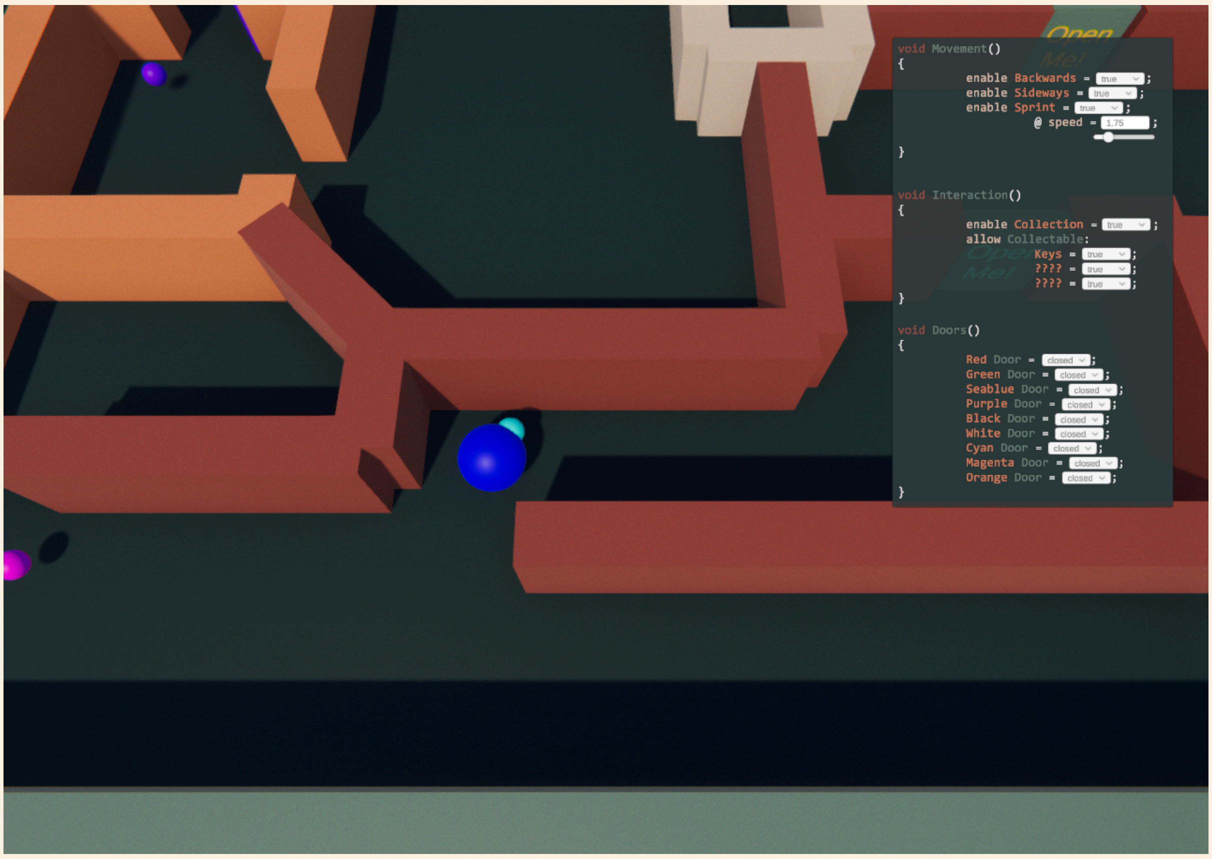


**Small Update - Added UI Script, Changed UI Panel location**

```
Changed Panel from stretching over right side of screen to now hovering over gameplay. Added
UserInterfaceHandler script.
```

TiesPoutsma dd5ff03 +41 -9

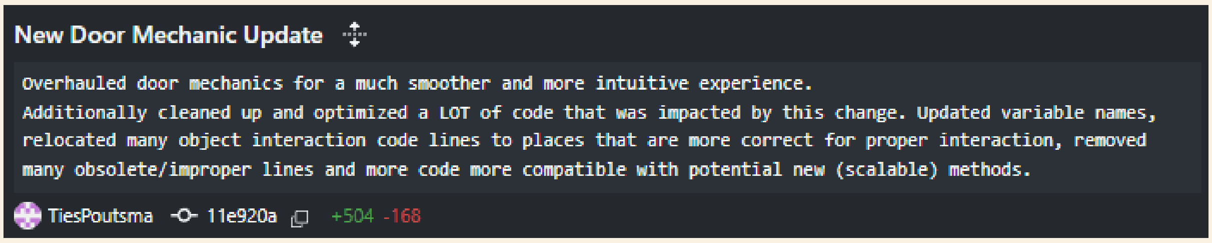
At this point in time, the game looked like this:



```
void Movement()
{
        enable Backwards = true;
        enable Sideways = true;
        enable Sprint = true;
                @ speed = 1.75;
}

void Interaction()
{
        enable Collection = true;
        allow Collectable:
                Keys = true;
                ???? = true;
                ???? = true;
}

void Doors()
{
        Red Door = closed;
        Green Door = closed;
        Seablue Door = closed;
        Purple Door = closed;
        Black Door = closed;
        White Door = closed;
        Cyan Door = closed;
        Magenta Door = closed;
        Orange Door = closed;
}
```

## 4.4) Updated Door Mechanics

After updating the styling of the game and creating a more solid foundation upon which to expand the game on, it was time to look into creating a more fluid gameplay experience, to allow the creation of more complex puzzles afterwards.



**New Door Mechanic Update**

```
Overhauled door mechanics for a much smoother and more intuitive experience.
Additionally cleaned up and optimized a LOT of code that was impacted by this change. Updated variable names,
relocated many object interaction code lines to places that are more correct for proper interaction, removed
many obsolete/improper lines and more code more compatible with potential new (scalable) methods.
```

TiesPoutsma 11e920a +504 -168

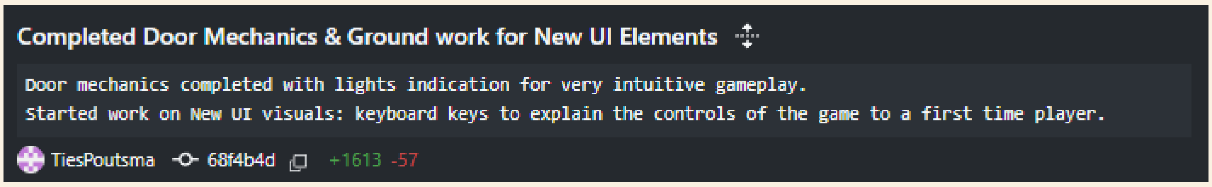
A big step in creating this fluid gameplay was updating the functionality of the doors. Up until this point, doors were only openable by *holding* the key and moving in close proximity. If the player then let go of the key or moved out of proximity of the door, the door would close again.

Not having a way to permanently open doors was a big issue: it didn't allow any backtracking throughout the level, forcing a more linear maze style. That was not in line with the creative vision behind the game. The way doors worked had to change.

With this change, the mechanics of the doors were updated to now no longer open automatically. Instead, the player interacts with the door using the ‘interact’ key ‘E’ that was also used for dropping held keys. If the player attempts to interact with a door, and they are holding the correct key, then the interaction happens: the key is consumed, being stored inside the door, and the door opens. If the player then presses the interact key again, the door will close, and the key will be restored and placed back into the player's holdItem space.

As is mentioned in the description of the gitHub update, in this update a lot of code impacted by the new door mechanics was cleaned up and optimised. All relevant backend scripts had their structures overhauled, which included: updated variable names, relocation of many object interaction code lines to places that are more correct for proper interaction, removal of many obsolete/improper lines and the updating of code to be more compatible with potential new (scalable) methods.

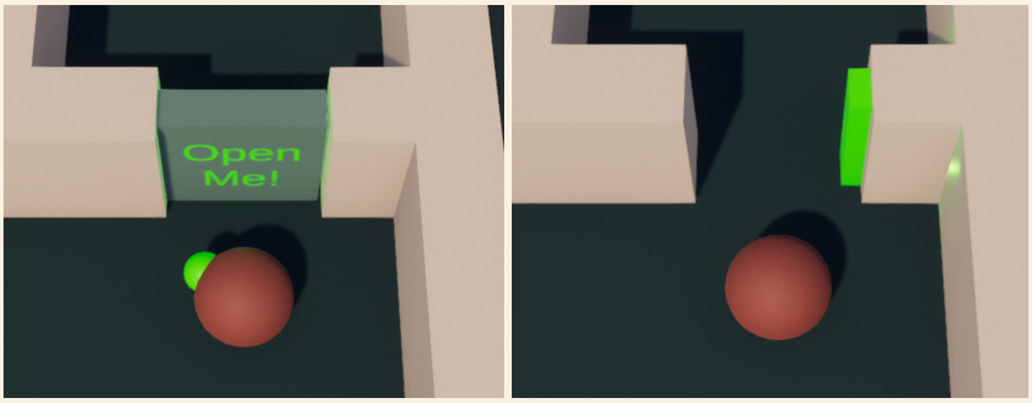
Lastly, in a smaller, second update, new visuals were also added to the updated doors.


Completed Door Mechanics & Ground work for New UI Elements

```
Door mechanics completed with lights indication for very intuitive gameplay.
Started work on New UI visuals: keyboard keys to explain the controls of the game to a first time player.
```

TiesPoutsma 68f4b4d +1613 -57

Now, coloured lights were added in the same colour as the door text. These lights would light up if the player is within proximity of the door while holding the correct key: that means that the lights light up if the player can open the door if they interact with it.
If they indeed do interact with the door and open it, the door changes colour in its entirety, taking on the same colour as the lights and text.

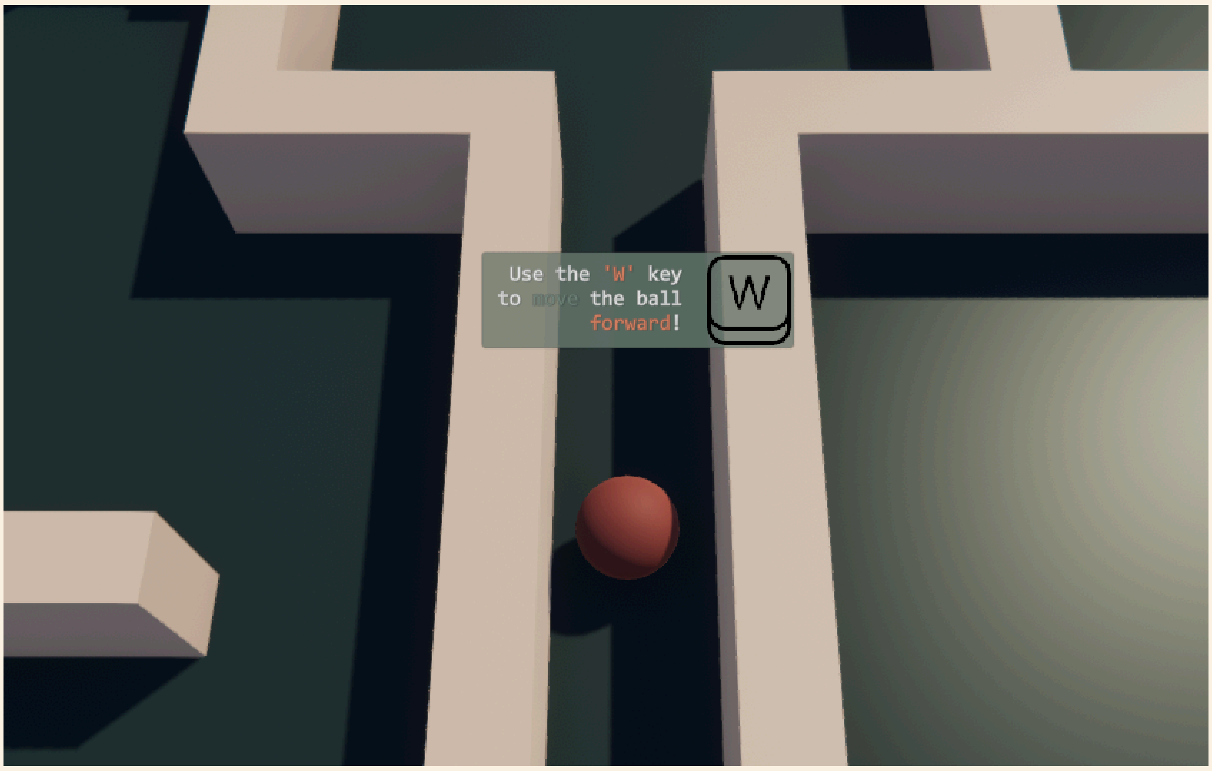


With this smaller update, some groundwork was laid for the next step of the game: adding user interface elements that give the player a concise but brief explanation of how to operate the game’s functions with their keyboard. This would end up becoming the Suggestion User Interface.

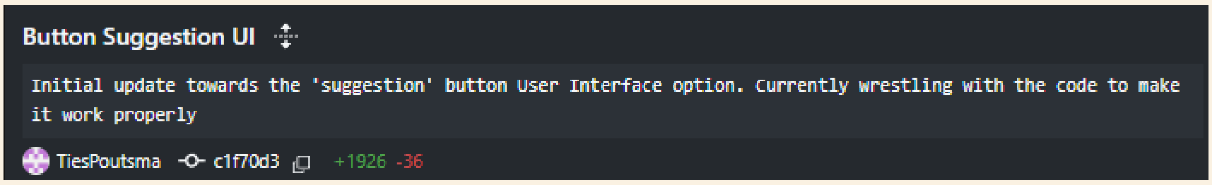
## 4.5) Suggestion User Interface

The need for the Suggestion User Interface came from some unofficial ‘playtests’. Users had a lot of trouble figuring out how to properly operate the game. It was virtually impossible to know that the ‘E’ key was used for interactions, nor was it clear that the ‘left-shift’ key on the keyboard could allow the user to boost (or ‘sprint’, as it was called at that time).
It was clear that something was needed to let users know the controls of the game.

### 4.5.1) Suggestion UI Elements



The idea took shape as simple user interface pop-up panels, with an image of the relevant keyboard key, and some short text explaining what the key does when pressed.



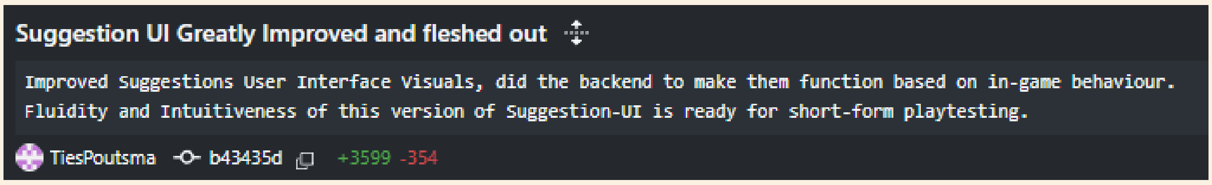
Button Suggestion UI

```
Initial update towards the 'suggestion' button User Interface option. Currently wrestling with the code to make
it work properly
```

TiesPoutsma c1f70d3 +1926 -36

The update tried hiding and showing the images using .anim animations, but Unity’s Animator was not suitable for the animation style I wanted. To make the panels fade in and out neatly, the alpha values of the panels were changed directly in the Suggestion UI script.
A different approach was needed.

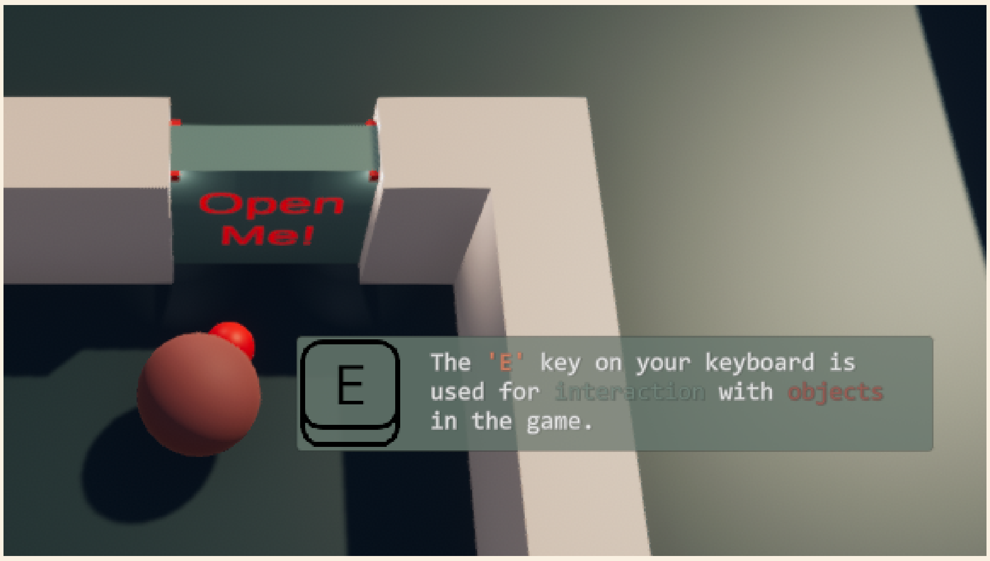
### **4.5.1)** Trigger Zones


**Suggestion UI Greatly Improved and fleshed out**

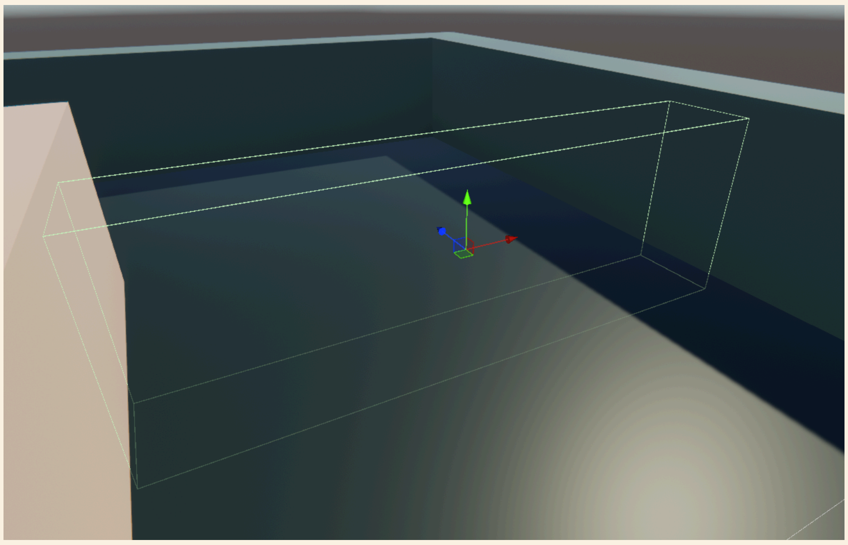
```
Improved Suggestions User Interface Visuals, did the backend to make them function based on in-game behaviour.
Fluidity and Intuitiveness of this version of Suggestion-UI is ready for short-form playtesting.
```

TiesPoutsma b43435d +3599 -354

To make these UI pop-ups appear and disappear smoothly, trigger zones were developed. As per the section on trigger zones in the Features section, trigger zones are invisible gameobjects with a collider set to ‘trigger’. It causes them to lose collisions, but still register when other colliders enter their zone. This ‘trigger’ moment will enable or disable a chosen Suggestion UI element. In the image below, a trigger zone is placed near the first door, alerting players that they can press the ‘E’ key to interact with, and open the door.


Open Me!
E
The 'E' key on your keyboard is used for interaction with objects in the game.

Trigger zones do not have to be spherical. They can also have a rectangle shape, such as the one below: this one causes the ‘E’ key explanation pop-up to fade out.

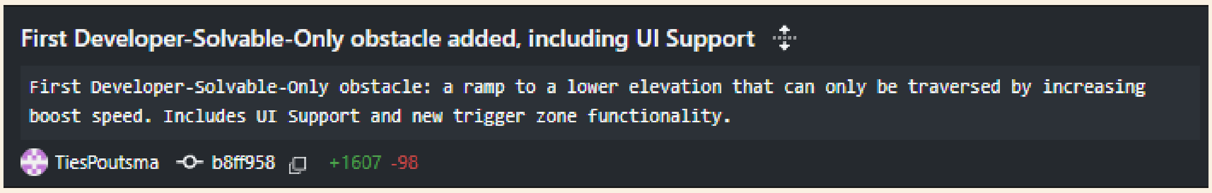


Once a trigger zone is activated once, it will disable itself after completing its animation.

## 4.6) Developer User Interface

With these elements in place, it was now time to develop and add the first obstacle into the game that could only be solved using the Developer User Interface.

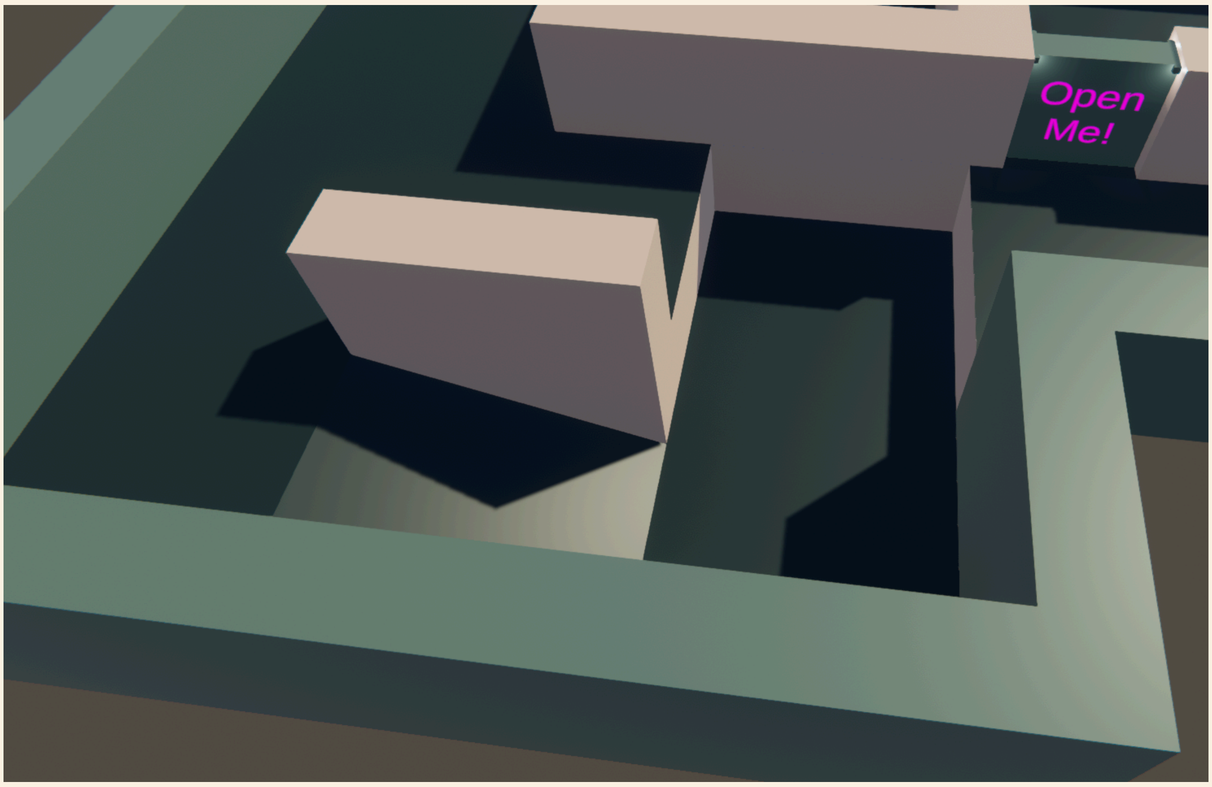
### 4.6.1) Level Layout Update: Gap & Ramp


First Developer-Solvable-Only obstacle added, including UI Support

```
First Developer-Solvable-Only obstacle: a ramp to a lower elevation that can only be traversed by increasing
boost speed. Includes UI Support and new trigger zone functionality.
```

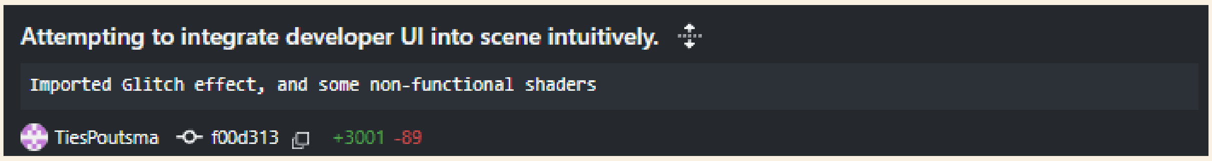
TiesPoutsma b8ff958 +1607 -98

This first obstacle was conceptualised as a gap in the floor: it's something that cannot be traversed using any of the available elements of the game. Except for the functionality of the Dev UI.



The idea behind the gap was that if the player has a boost speed above 4.0, then they can simply boost over the gap. The developer user interface has a function that allows the adjustment of the boost speed of the player, at that point from 0.0 to 5.0. If the player uses that, then they can cross the gap.
If the player makes a mistake, or has a boost speed of too low a calibre, then they will fall into the gap, and must use the added ramp to get back up to the main elevation level.
With this update the developer user interface panels containing the movement, collection and door functions were hidden. If the player moves close to a trigger zone near the gap, then the corresponding panel will pop back up permanently.

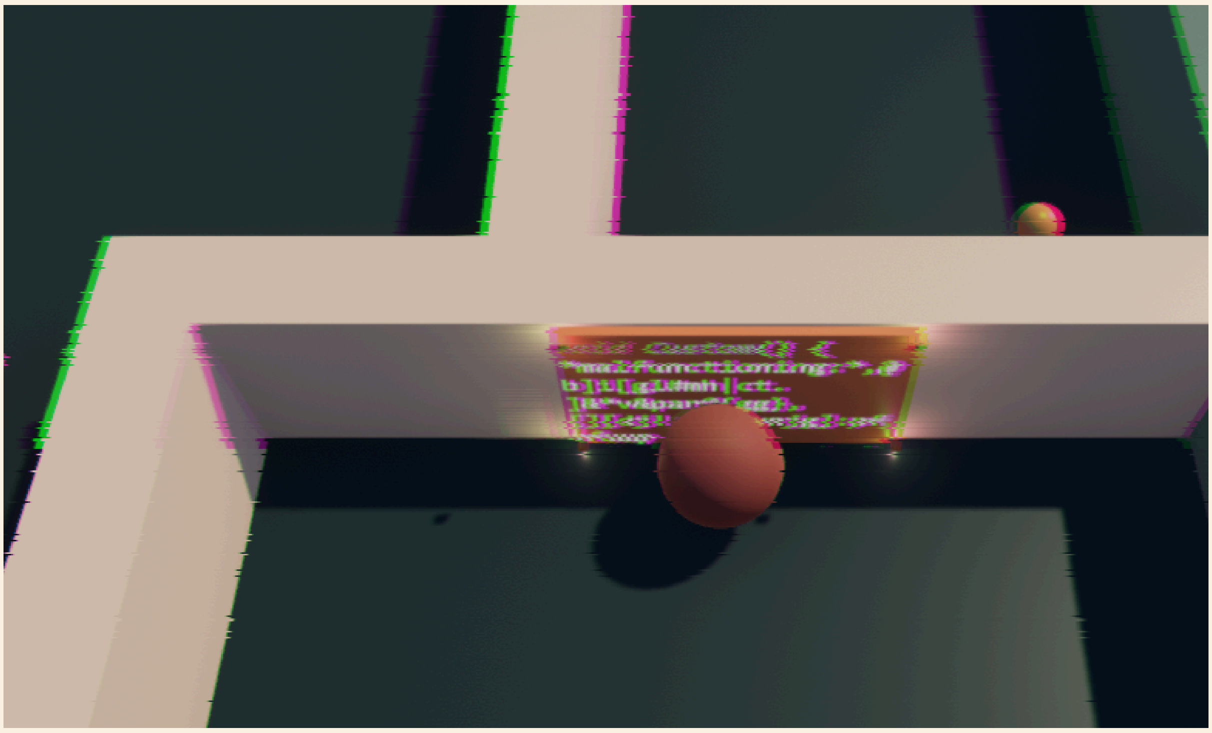
### **4.6.2)** Glitch Effect (& Unused Camera Shaders)


Attempting to integrate developer UI into scene intuitively.

```
Imported Glitch effect, and some non-functional shaders
```

TiesPoutsma f00d313 +3001 -89

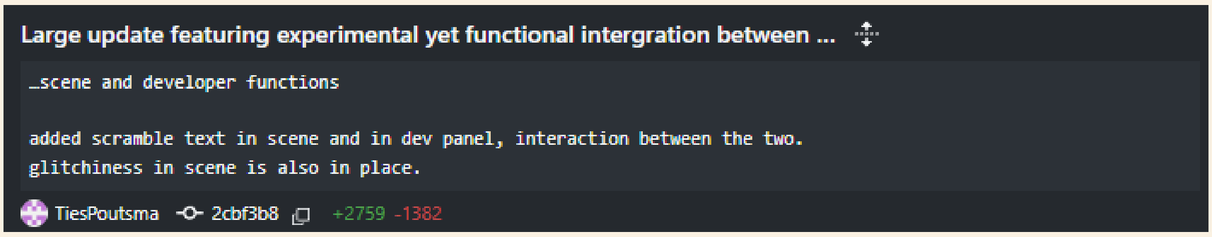
A question popped up at this point:
*'What is the thematic* reason *that the player would somehow be able to adjust the backend variables of the game?'*
The question held a lot of weight. It made no sense game-wise for the user to suddenly be able to adjust important game variables that have a massive impact on the playability of the game. Despite the game being purposefully mainly unthemed, it made sense to include some theming into the game to solve this problem. The initial idea behind this was to add a sense of 'breaking' the game. Some shaders were implemented that made parts of the screen look as if they were glitching, broken and otherwise performing like they shouldn't.
That 'broken' gameplay would serve as a bridge between the base game and the Dev UI: it would be possible for the player to access variables of the game itself, because the game was fundamentally 'broken', and the code lying underneath it was exposed.



Those imported shaders ended up not functioning properly with the minigame, so they were shelved, and another 'game-breaking' thematic was needed. Eventually the 'Glitch Effect' by Kino was used. It contained a few glitching camera effects, of which three were implemented into the game: the Screen Shake, Scan Line Jitter, and the Colour Drift.

### **4.6.3)** Scramble Text

The glitch effect was great at making it seem like the program was breaking, but it wasn't everything the game needed styling wise. A second styling feature designed for this game is found in both the malfunctioning screens that were added in this update and in the developer user interface that was already in place. This styling feature is the 'Scramble Text'.

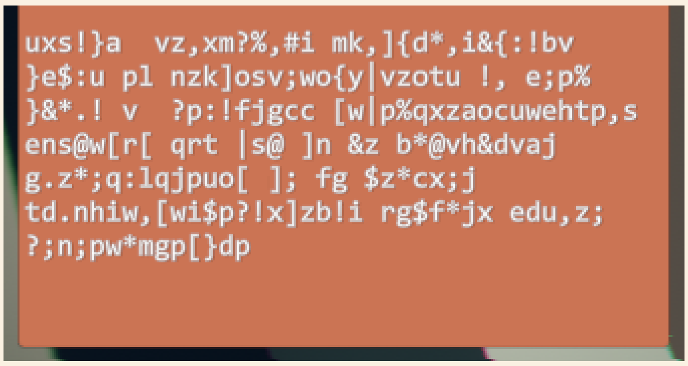
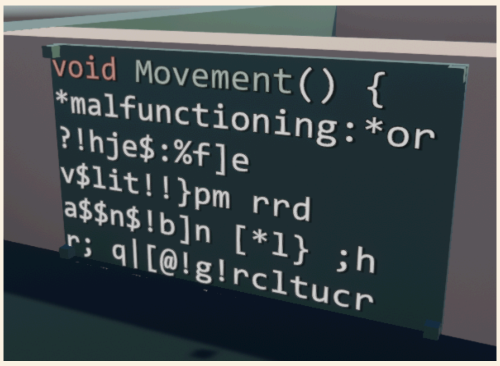


Large update featuring experimental yet functional intergration between …

```
…scene and developer functions

added scramble text in scene and in dev panel, interaction between the two.
glitchiness in scene is also in place.
```

TiesPoutsma 2cbf3b8 +2759 -1382

The scrambling text had a simple concept: it would take a correct string of displayable information, and change that string to randomised letters or symbols. The individual characters are then continuously randomised, creating the appearance of a string that has completely lost all control.


uxs!}a  vz,xm?%,#i mk,]{d*,i&{:!bv
}e$:u pl nzk]osv;wo{y|vzotu !, e;p%
}&*.! v  ?p:!fjgcc [w|p%qxzaocuwehtp,s
ens@w[r[ qrt |s@ ]n &z b*@vh&dvaj
g.z*;q:lqjpuo[ ]; fg $z*cx;j
td.nhiw,[wi$p?!x]zb!i rg$f*jx edu,z;
?;n;pw*mgp[}dp


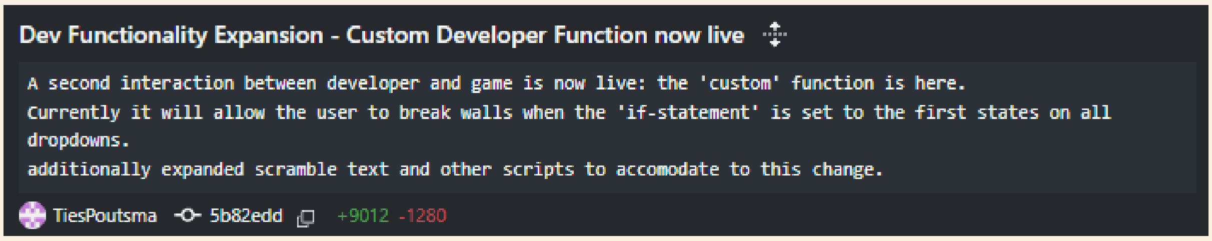
With the scrambling developer user interface and the scrambling physical malfunctioning screens in the game being connected, the link between the Dev UI and the obstacles in the game that can only be solved with it was finally bridged. Thematically, it made sense for the user to access developer variables: they were playing in a ‘broken’ and ‘glitching’ game, which they had to traverse and fix themselves.

---

## 4.7) Custom Developer Function

It wasn’t clear yet at this point in the timeline how many more ‘developer-only obstacles’ would be developed, or what they would look like. There was, however, one other concept theorised.
Something that felt very unsatisfying gameplay wise was completing the game. There was no type of victory screen or any other type of functionality in place that indicated that the end of the level was reached.
The idea to solve this: allow the player to write their own code, and give them full creative freedom over the rest of the game.

### 4.7.1) Breaking Walls

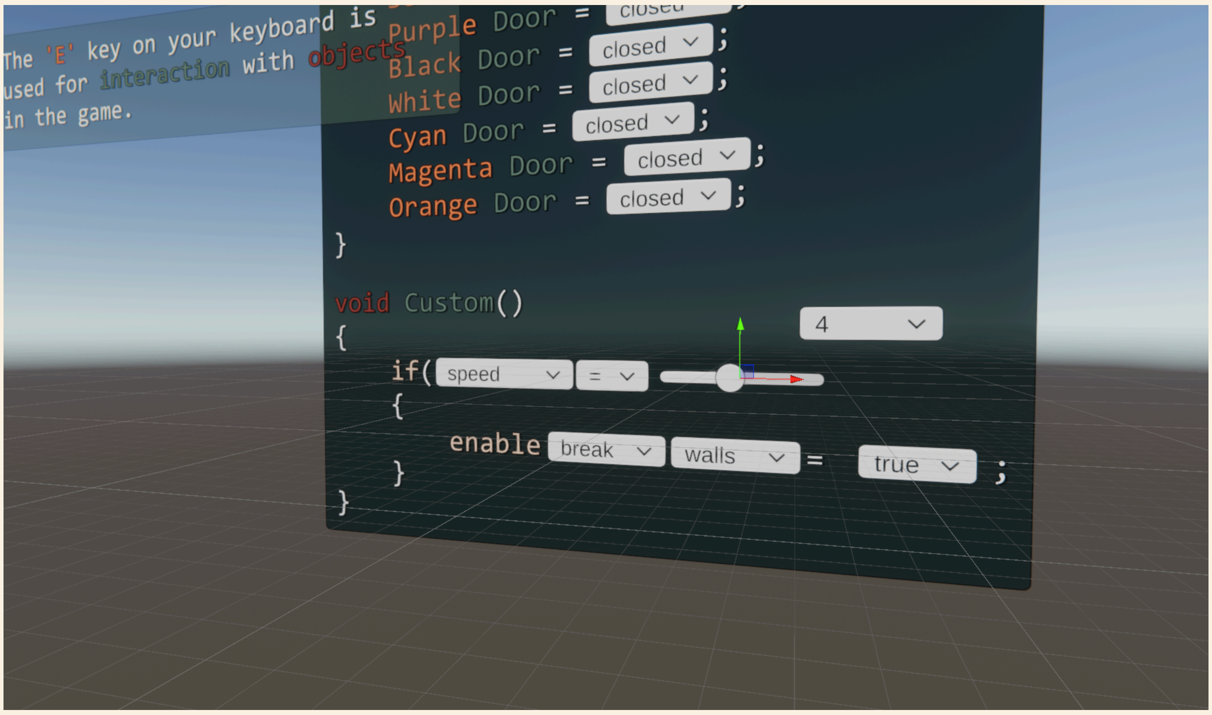


Dev Functionality Expansion - Custom Developer Function now live

```
A second interaction between developer and game is now live: the 'custom' function is here.
Currently it will allow the user to break walls when the 'if-statement' is set to the first states on all
dropdowns.
additionally expanded scramble text and other scripts to accomodate to this change.
```

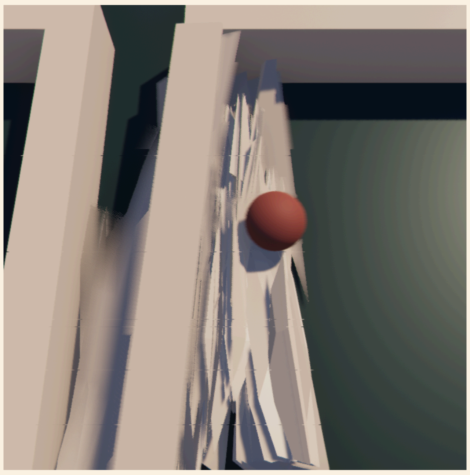
TiesPoutsma 5b82edd +9012 -1280

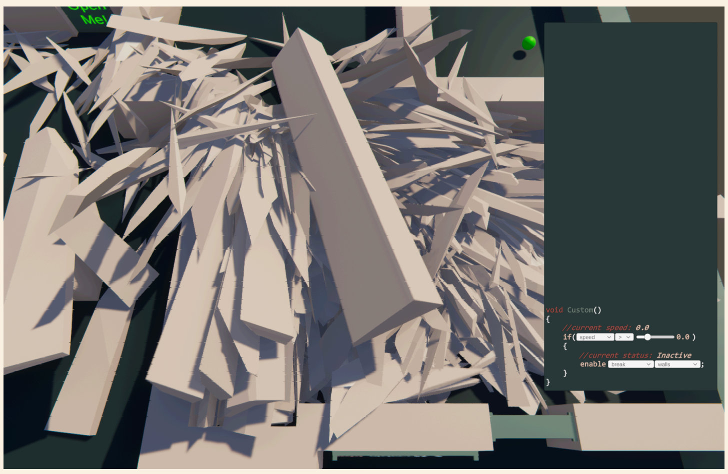
While full creative freedom would be a bit out of the scope of the project, a reasonable compromise could be made.



```
The 'E' key on your keyboard is
used for interaction with objects
in the game.

Purple Door = closed ;
Black Door = closed ;
White Door = closed ;
Cyan Door = closed ;
Magenta Door = closed ;
Orange Door = closed ;
}

void Custom()
{
    if( speed = 4
    {
        enable break walls = true ;
    }
}
```

A ‘Custom’ option was added to the developer user interface. For a more traditional programming format, an ‘if-statement’ was chosen to represent the function that is extremely common, yet very valuable in programming.
As with a regular if statement, it consists of two parts: the condition, and the execution.
The first line of the if-statement refers to the condition: *if ( speed = 4 )* is an example shown in the screenshot above.
The next line refers to the execution. Should the first line return *‘true’*, then the execution line will be called.
The first execution line created for this custom if-statement reads: *enable break walls = true;*
As one might expect, this indeed allowed the player to smash through walls, using the Mesh Destruction Script by ditzel.



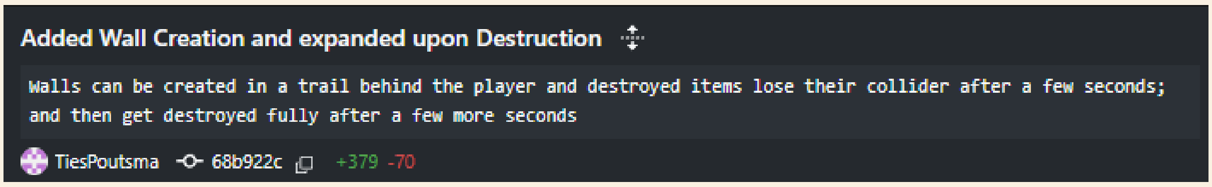


```
void Custom()
{
    //current speed: 0.0
    if( speed > 0.0 )
    {
        //current status: Inactive
        enable break walls ;
    }
}
```

The script was quite chaotic, and needed some tweaking to properly function with the minigame.

## 4.7.2) Creating Walls

A change was made to destruction in this next update. Instead of having wall fragments scattered around the level permanently, they would fall through the floor after a few seconds, and be fully destroyed (removed from the scene) after that.

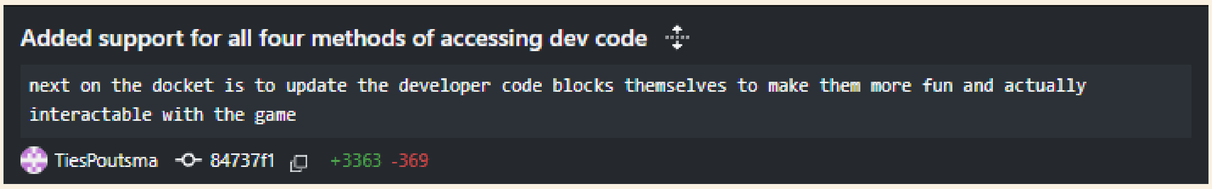

Added Wall Creation and expanded upon Destruction

```
Walls can be created in a trail behind the player and destroyed items lose their collider after a few seconds;
and then get destroyed fully after a few more seconds
```

TiesPoutsma 68b922c +379 -70

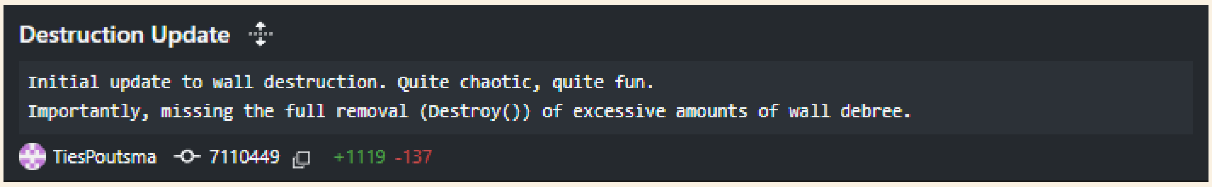
To give the player something else to do besides only breaking walls, a second option was added: wall creation. If the player selects this option using the custom if-statement, then walls will be created in a path behind the player. These walls are also breakable, should the player return their if-statement to enable wall breaking.

## 4.7.3) Additional Updates


Added support for all four methods of accessing dev code

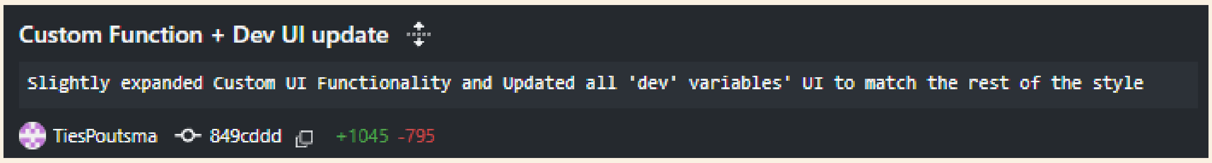
```
next on the docket is to update the developer code blocks themselves to make them more fun and actually
interactable with the game
```

TiesPoutsma 84737f1 +3363 -369


Destruction Update

```
Initial update to wall destruction. Quite chaotic, quite fun.
Importantly, missing the full removal (Destroy()) of excessive amounts of wall debree.
```

TiesPoutsma 7110449 +1119 -137


Custom Function + Dev UI update

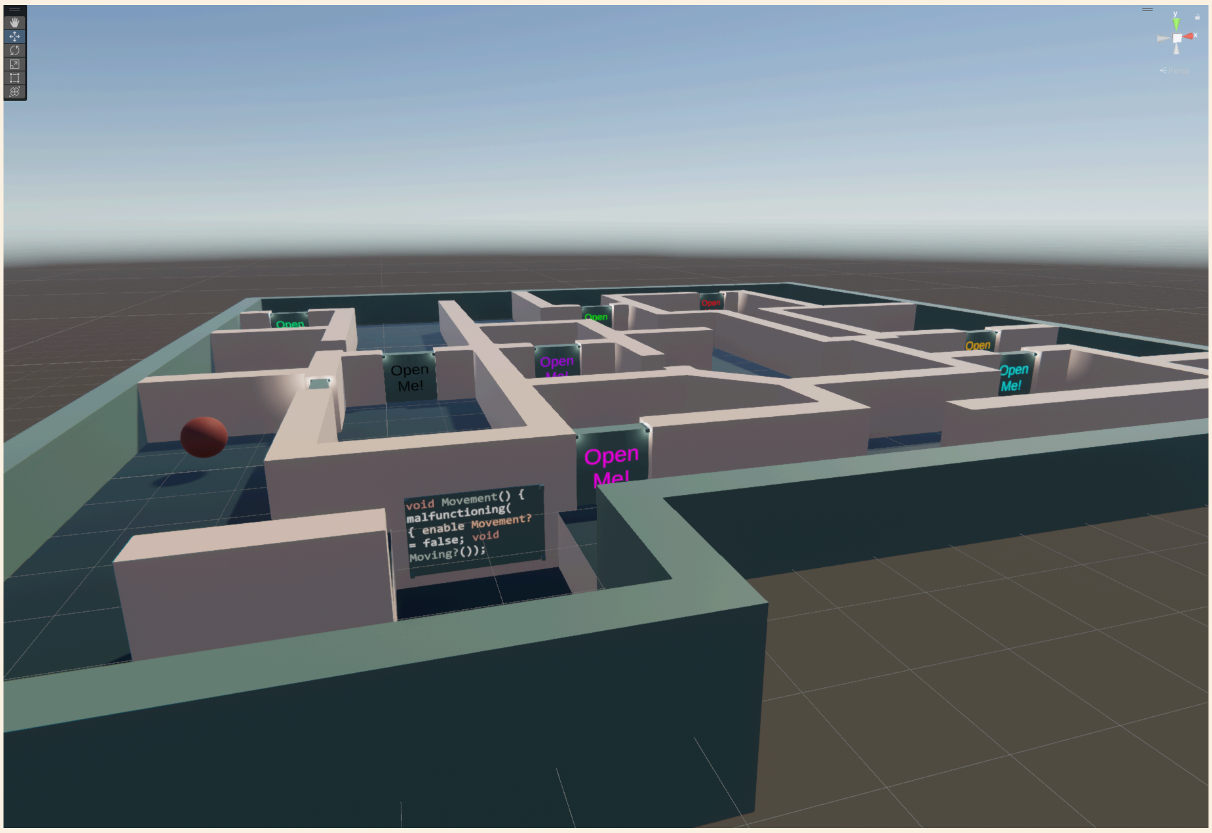
```
Slightly expanded Custom UI Functionality and Updated all 'dev' variables' UI to match the rest of the style
```

TiesPoutsma 849cddd +1045 -795

A few smaller updates were made for the sake of optimisation and styling. An important change was the backend expansion of the Developer User Interface. The expansion featured new physical malfunctioning screens added, making all four sections of the developer user interface interactable and accessible. The second and third sections of the Dev UI did not have developer-only obstacles at this point, though.
The destruction functionality was also updated, removing recursion issues which would crash the game, improving collision behaviour between broken mesh pieces and the player, and improving framerate issues when breaking a wall for the first time.

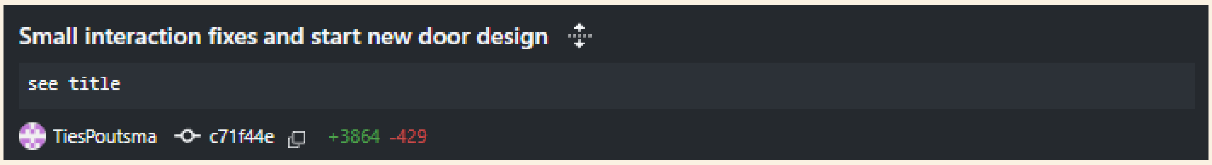
## 4.8) Final Style Update

At this point, the minigame was starting to really take shape. Next on the docket was the overall styling of the game.



```
void Movement() {
malfunctioning(
{ enable Movement?
= false; void
Moving?());
```

There were still some inconsistencies with the game's styling, namely the doors which still had many varying colours.



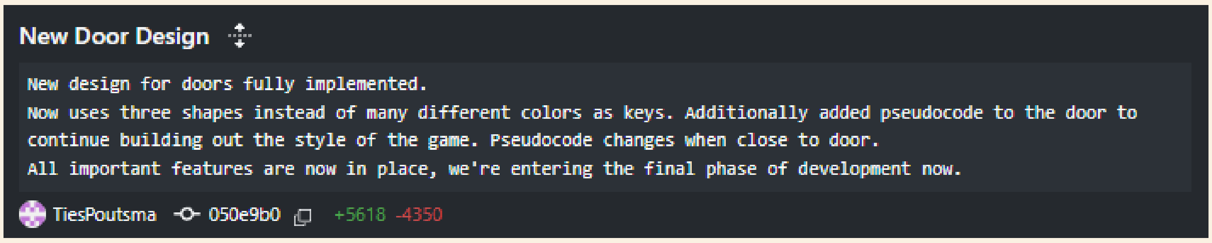
**Small interaction fixes and start new door design**

```
see title
```

TiesPoutsma c71f44e +3864 -429

After some initial backend fixes, it was time to tackle the door design.

### 4.8.1) New Door and Key Design

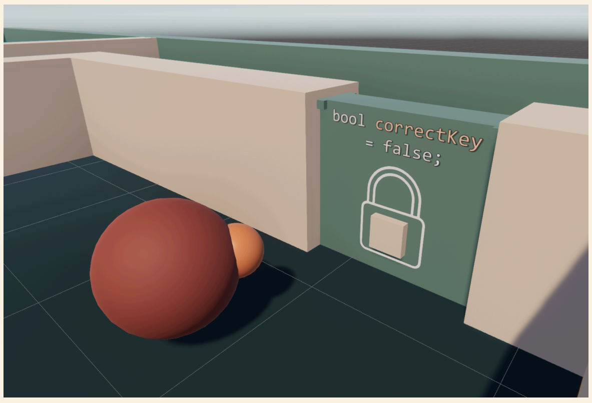


**New Door Design**

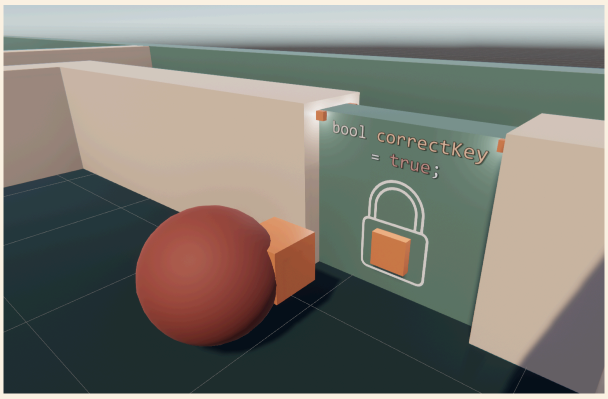
```
New design for doors fully implemented.
Now uses three shapes instead of many different colors as keys. Additionally added pseudocode to the door to
continue building out the style of the game. Pseudocode changes when close to door.
All important features are now in place, we're entering the final phase of development now.
```

TiesPoutsma 050e9b0 +5618 -4350

The main change made to the doors was the removal of the many different colours in favour of three different shapes: a square, a circle, and a capsule.



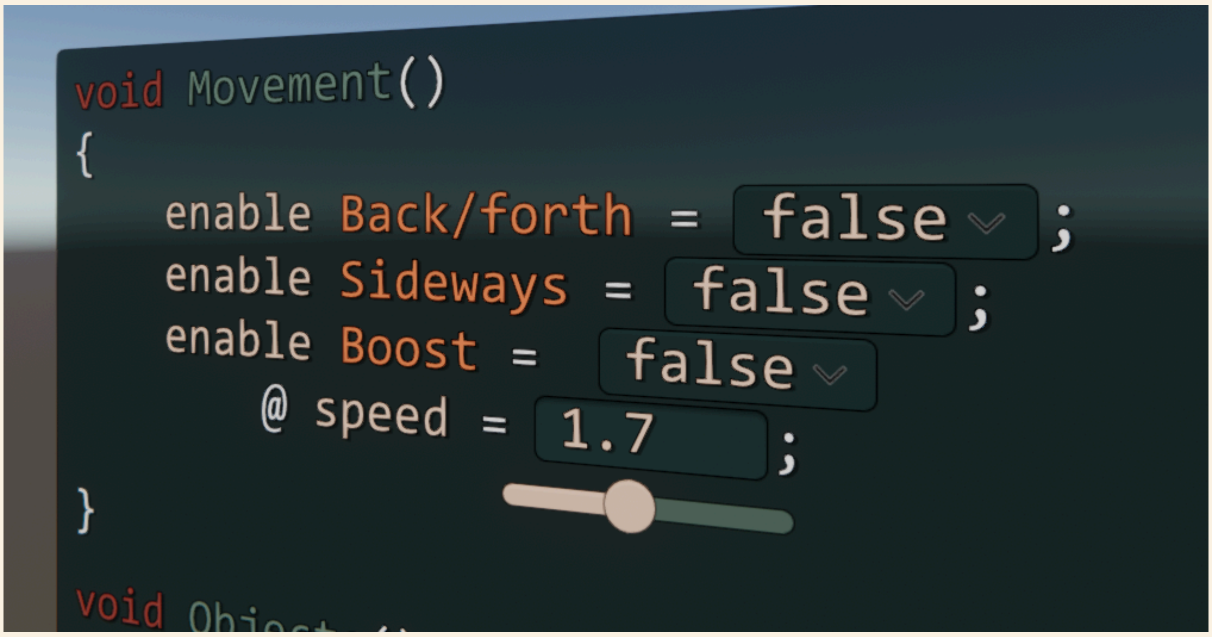
```
bool correctKey
= false;
```



```
bool correctKey
= true;
```

The visuals of the door were also overhauled. There was now a lock icon on the doors, and a shape in the middle of the door. The shape referred to the type of key that needed to be collected for the door to be openable.
Alongside that, pseudocode was added to the door, which referred to the key type that belongs to the door as well. This pseudocode was added to expand on the pseudocode theming and overall theming of the game.
If the player came into proximity of the door while holding the correct key, multiple things would happen. The four lights on top of the door would turn on, shining with a white light. The symbol in the middle of the lock icon would go from its base off-white colour to orange, the same colour as the keys now had. And lastly, the pseudocode boolean would spell out that it returned true, instead of false.

### **4.8.2)** Developer UI Variables Style Update



```
void Movement()
{
    enable Back/forth = false ;
    enable Sideways = false ;
    enable Boost = false
        @ speed = 1.7 ;
}

void Object
```

In the same update, the developer user interface had a styling update as well. Now, not only the text content had the Visual Studio 2022 font and the game’s colour palette, but the variables of the developer user interface were updated to match the rest of the panel.

### **4.8.3)** Preparation for Final Update

With the new door design, new key shapes, updated Dev UI, and an expanded and improved backend, the game had all the pieces it needed for a massive, final update that put all those pieces in their place.

For the final update, the main things to be tackled were:

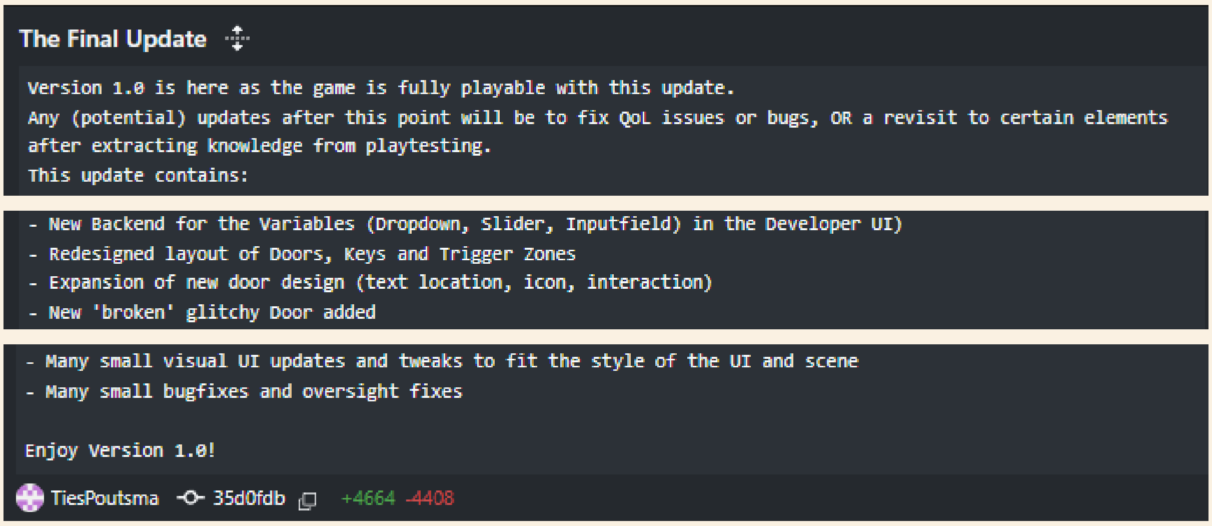
Revisiting the Developer User Interface one last time and build a script which makes the variables of the UI respond properly to the real game variables and vice versa.

Design two new developer-only obstacles that serve as a continuation of the objective to make the player work with the Dev UI to solve obstacles.

Redesign the level where needed, to accommodate for these changes.

With these ideas in mind, development on the final update started.

## 4.9) The Final Update



**The Final Update**

```
Version 1.0 is here as the game is fully playable with this update.
Any (potential) updates after this point will be to fix QoL issues or bugs, OR a revisit to certain elements
after extracting knowledge from playtesting.
This update contains:
```

```
- New Backend for the Variables (Dropdown, Slider, Inputfield) in the Developer UI)
- Redesigned layout of Doors, Keys and Trigger Zones
- Expansion of new door design (text location, icon, interaction)
- New 'broken' glitchy Door added
```

```
- Many small visual UI updates and tweaks to fit the style of the UI and scene
- Many small bugfixes and oversight fixes

Enjoy Version 1.0!
```

TiesPoutsma 35d0fdb +4664 -4408

### 4.9.1) Developer UI Variables Mechanics Overhaul

To fully flesh out the game and make sure that all interactions between the variables of the Dev UI and the actual game variables worked properly, a new script was written that directly linked all dropdown menus, sliders and input-fields of the Dev UI with their relevant real variables. This entire overhaul is not visible on the front end of the game whatsoever, yet still it is extremely important.
Previously, the variables of the Dev UI 'communicated' through Unity's built in User Interface callback functions. Those proved to be horribly inconsistent, and oftentimes were not even possible properly at all.

The DevVariablesUIBehaviour C# script solved these issues.

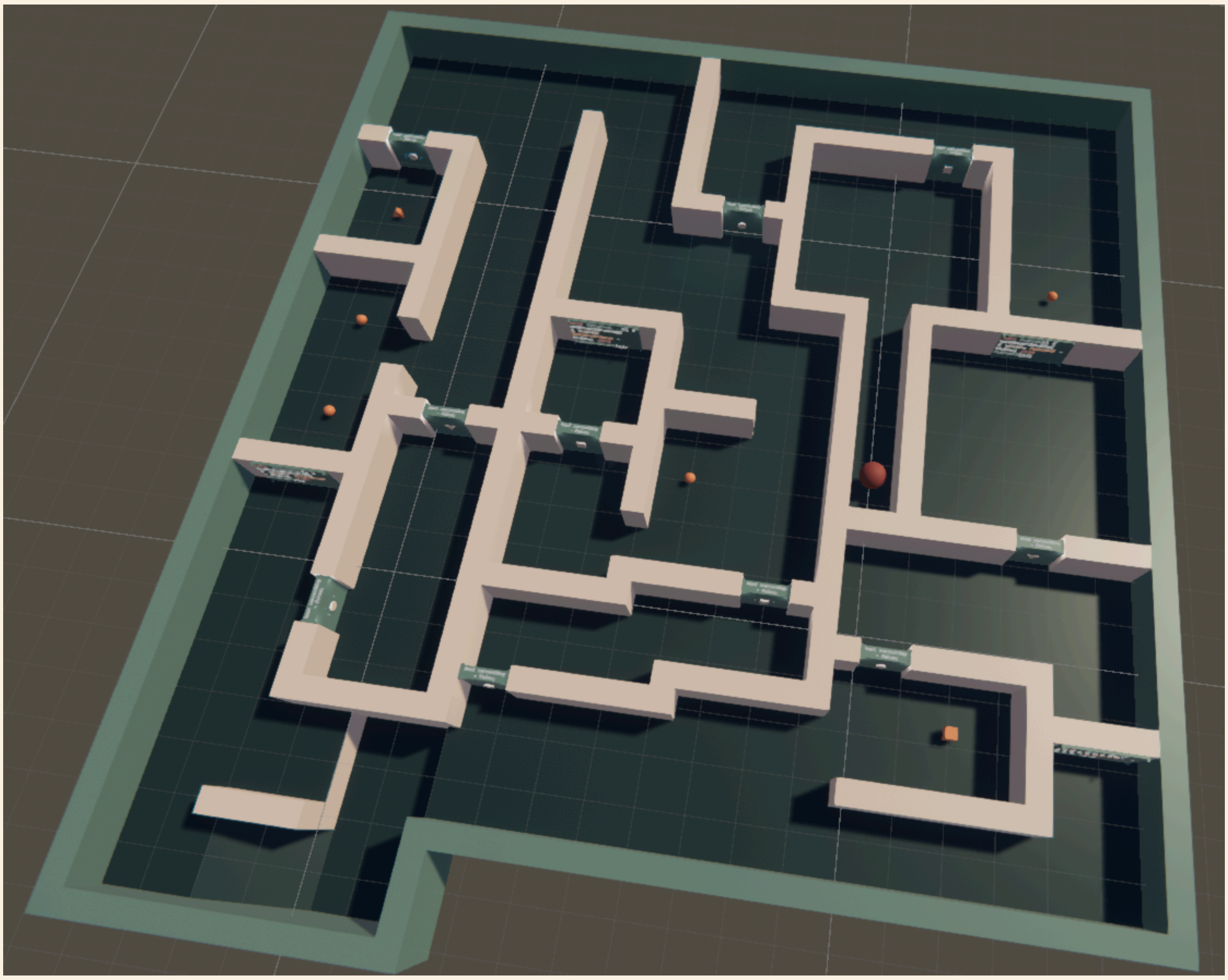
With a fully functional developer user interface in place, it was time to design the final two 'dev-only' obstacles for the game.

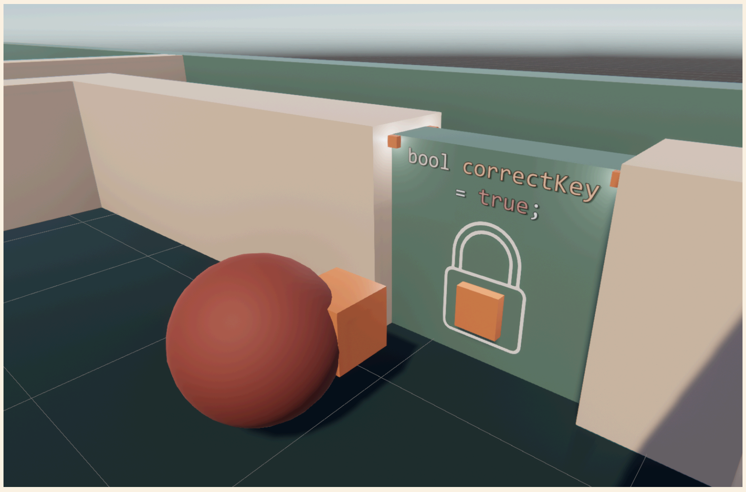
### 4.9.2) Redesigned Level Layout: Doors, Keys, Screens, Trigger Zones

The new door design added a challenge in the level design. The amount of possible key types to open the doors in the scene was reduced from nine keys for nine doors, to three key types for nine doors. This meant that players could simply collect their keys from previously opened doors to easily complete the entire game. It killed all variability from the game, and needed a solution. By cleverly choosing which door types and key types to put where, and by disabling the key reclamation feature, the game was back to its original puzzle vision, with no simple linear solution.

Screens were also shuffled around a bit, to create a healthier balance between base game puzzle obstacles, and developer-only obstacles. The movement screen was still placed near the ramp, but on a different wall, so that the player cannot accidentally stumble upon it. It must be actively sought out.
The same thing was done for the new Object and Interaction screens. The Object screen was placed in the southeastern corner of the level, and the Interaction screen was placed in

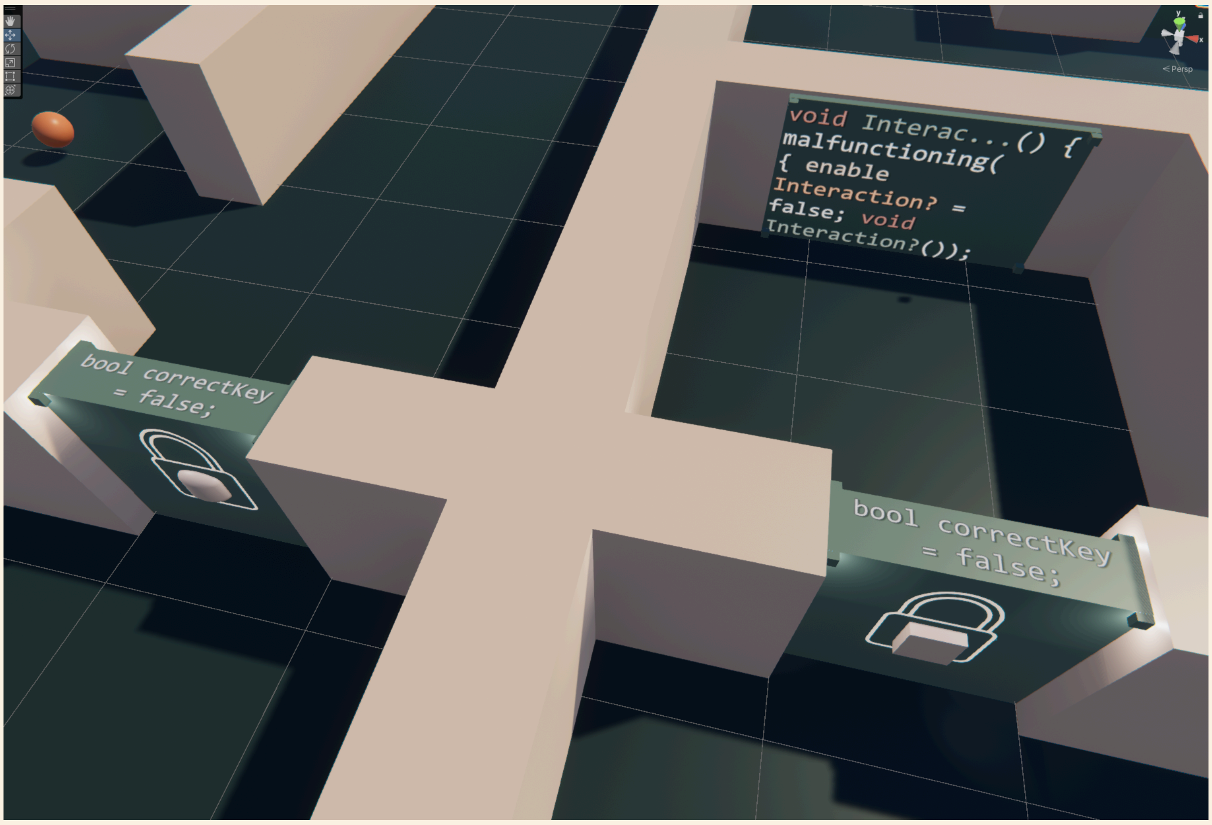
the middle of the maze, but behind a locked door - one that could be seen from the start of the game, but only opened much further up in the gameplay process.
Lastly some trigger zones for both the glitch effects and for the suggestion user interface were shuffled around a little, once again to provide balance in intensity of information in case of the Suggestion UI, and balance in glitchiness in regards to the glitch effect.
This is a screenshot of the final layout of the minigame:





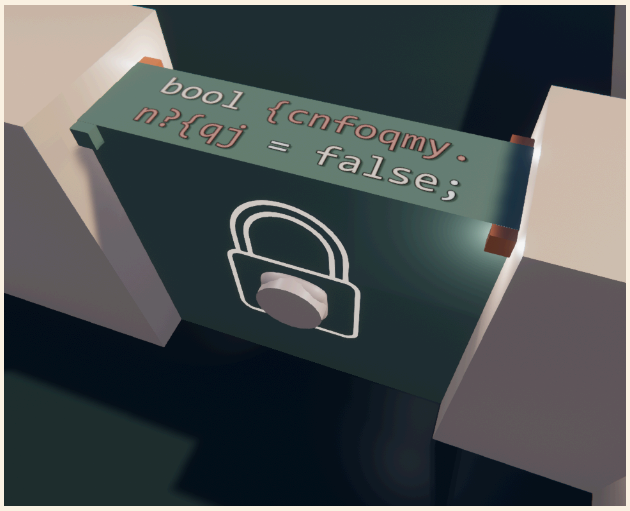
### 4.9.3) New Door Design Finalisation

In the final update, the new door design was revisited one last time. In the previous update, the doors had the pseudocode boolean text on the front of the doors. This turned out to be difficult to see if the player was coming from the backside of the door, due to the game's camera angle. To solve this, the icon and symbol were moved up on the door a little bit, and the pseudocode boolean text was changed to be on the topside of the door, rather than on the front.



Additionally, as mentioned, the reclamation availability of keys from doors was disabled. This change is also what stemmed one of the developer-only challenges: the last two doors in the scene did not have keys lying around in the scene for them to be opened with. Instead, with the developer user interface, the player could re-enable the key retrieval function, and retrieve previously used keys from other doors, and use those keys to open the final two doors.
The player would have to interact with the Interaction screen in the middle of the scene, behind a locked square door. To get to that door, however, the player must solve one more 'developer-only' obstacle first.

### **4.9.4)** New 'Broken'/Glitchy Door



This is the point where the elusive 'broken' door was first conceptualised and made.
Sitting pretty much dead center in the middle of the maze, this door cannot be opened by any means using the game's functions.

The only way to open it, is by directly adjusting its open/closed status in the developer user interface, in the Objects block.
The Objects block is unlocked by interacting with the objects screen in the southeastern side of the maze.

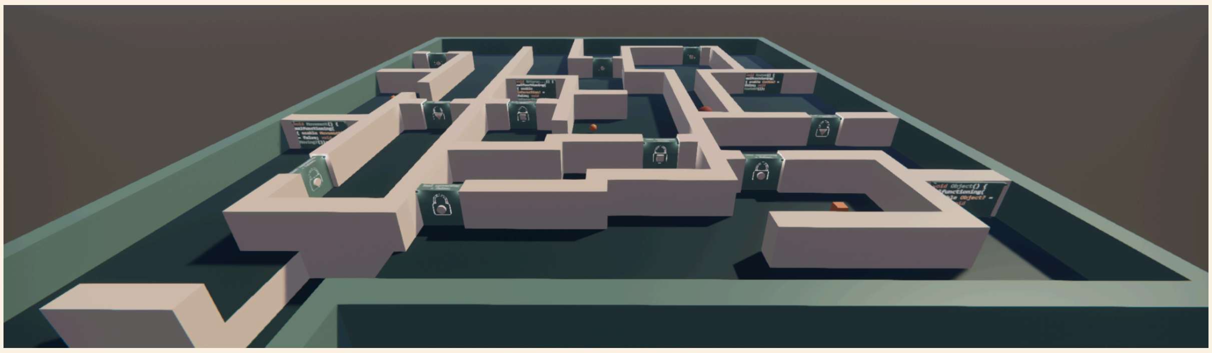
### **4.9.5)** Small oversight/bug/inconsistency fixes

The final changes made to the game for the 1.0 release were a large amount of ‘dots-on-the-i’.

These contain, but are not limited to:

- Backend semantics
- Backend optimisation
- Removal of deprecated code
- Public/private variable consistency
- Post processing optimisation
- Camera angles
- Lighting intensity and rotations
- Scramble text sizing and scramble intensity
- Glitch effect intensity
- Wall, door, key, and screen placement - with pixel perfect precision
- Wall destruction and creation optimisation
- Adding script functionality to Destroy() leftover objects in the scene
- Quality assurance
- General bug testing
- Collision testing
- Visual clarity testing
- Some more smaller updates

At this point, the mini-game was ready for its version 1.0 release.
This is the game version that was used for three initial playtests. After those playtests, a few more small changes were made, and version 1.1 was created.
The playtesting section of this report will expand on those changes.



# Five *Playtesting*

## 5.1) In General

### 5.1.1) Plan

The main information sought after can be found in the answer to these two questions:

- How do playtesters experience the product itself?
    - What is their experience in terms of in-game progression?
    - How do they experience the game controls and game functionality?
    - Do they have any issues with visibility or accessibility of game features?
- How do playtesters view the product in regards to the research question?
    - Beginner/unskilled programmers
    - Intermediate programmers
    - Experienced programmers

Playtests were set up with these questions in mind.
Individual playtest sessions had a session time of 15 to 30 minutes with the possibility of overtime in case of interesting and useful discussions.
The individual playtesting participants were briefed about the session itself and its goals before starting. They were told that:

- They would be playing a three dimensional puzzle game.
- They were asked to think out loud.
- They were told that the author would not interact with them during the session, unless they felt that they were completely stuck and needed help solving the puzzle.
- The author would monitor and observe their gameplay choices and experience, and their overall behaviour throughout the session.

Additionally, the individual playtesters were told that there would be a short debriefing interview after their completion of the product playthrough.

### 5.1.2) Audience

The targeted audience for playtesting is a varied mix of individuals that are around 5-15 years older than the targeted audience for the eventual product itself. Additionally, the individuals targeted for the playtests will have either low, medium or high pre-existing skill with programming (object based programming).
At the end of the playtesting rounds, twenty playtests were performed.
From the final playtesting audience, five playtesters were beginners or unskilled programmers, eight were intermediate level programmers, and seven were experienced programmers.

## **5.2)** Results

### **5.2.1)** Initial Playtests and Product Version 1.1

After the release of the 1.0 version of the product, three initial playtests were performed. These playtests had slightly less structure than the subsequent ones, and were targeted slightly more towards game functionality rather than the research question - in order to find and target any specific issues that the developer might have overlooked.

A summary of noteworthy results from these three playtests:

**Game controls/functionality and progression:**
All three playtesters were able to complete the entire minigame without any help.
Most players did not take too long with understanding the base game controls and the first few puzzle functions.
The first obstacle was found in the developer screens. It took a playtester quite some time to figure out that the physical screens in the game could not only be interacted with, but that it was required for progressing in the game.
The most time consuming obstacle for all three playtesters was found in solving the third developer-only obstacle. Playtesters had trouble understanding that ‘reclaim keys’ meant that they could close doors and reclaim the key that was used to open them.
One playtester remarks that their favourite moment in the game was the moment they solved the first developer only obstacle - after unlocking the movement block of the developer screen, they figured out that they needed to increase their boost seeds to cross the gap in the level.

**Game accessibility/visibility:**
One of the more common accessibility issues that came to light from these playtests was found in the key and door system. The three icon shapes used in version 1.0 were a cube/square, a sphere/circle and a capsule. Playtesters were confused with the three, as the capsule and sphere shapes are quite similar at first glance.
Another issue was found in the phrasing of the ‘reclaim keys’ function. It was unclear what game functionality the line was referring to, and it took playtesters quite some time and trial and error to figure out.
An accessibility problem - that was originally believed to be an unsolvable game engine limitation - was found in an issue with the game’s built-in behaviour of user interface and keyboard interactions: after the user interacts with the user interface, the highlighted button or dropdown item can also be interacted with using the keyboard. The space bar and the arrow keys or WASD keys will change which UI item is highlighted and what their contents are. A playtester does not expect this, and is unpleasantly surprised to find their user interface elements changed when they didn’t mean to.

**Overall Experience**
Playtesters like the feeling of gaining an increasing amount of control over the game as it goes on. Subsequently, being able to fully destroy and create walls at the end of the game

felt good and was fun. A common remark was that breaking and creating walls was a great feature that most playtesters enjoyed, mentioning that it felt rewarding and satisfying.
One playtester also remarks that because the custom code if-statement doesn't suggest anything but lets the user construct their own code, the creation/destruction features feel more like something the playtester 'discovers', instead of it being handed to them. Additionally, if they would have only been allowed to destroy walls, it would have just felt gimmicky, but being able to also create walls adds fulfilment and gives the game proper closure. Another positive remark was found in the scrambling text on the developer user interface: the unscrambling effect was experienced as very satisfying.
Playtesters are almost universally very positive about the product.

**Thoughts on the products' 'introduction to programming' aspect:**
Playtesters remark their understanding of the goal of the product and express their satisfaction with the games 'developer access' features. None of the three initial playtesters are beginner level or unskilled programmers. For playtesters in that category, a special question is formed. When asked *"if you found yourself back in time before your first introduction to programming, and you instead had this product as your first programming related encounter, would you have been interested in programming as a concept (and how so)?"*, the answer was a resounding 'yes' from all three. One describes that they "would have been able to understand programming better through the use of this product". Other remarks include:

- "In regards to the 'custom code', if an unskilled programmer is able to understand the if-statement, they'll understand programming in general."
- "I could see a software development company being interested in this product."
- "If I were an unskilled programmer it would take me some time to figure out how to comprehend the pseudocode, but I would still be able to understand it."

**Changes made for version 1.1**
After the initial playtests, some small tweaks were done to solve some of the issues that came up - mainly accessibility issues that clearly hindered the gameplay experience.
These changes were:

- The capsule key and capsule door have their meshes changed to a triangle (tetrahedron).
- The 'Reclaim Keys' text has been changed to 'Unlock Doors & Reclaim Keys'.
- The intractability between the user interface and the keyboard has been disabled.

With those changes implemented, Version 1.1 of the product was ready for the main playtest sessions.

### **5.2.2)** Main Playtests

In the main playtesting sessions, the questions and focus was similar to those of the initial playtests. With the changes of the 1.1 release though, most playtesters were able to complete the game much more smoothly.

#### 5.2.2.1) Overall Experience

Overall, all seventeen main playtesters were very positive about their experience.

Many playtesters remarked both out loud during the playtest and during the debriefing interview that they had fun, and were enjoying themselves.

Gameplay wise, playtesters largely were able to solve the initial door&key puzzle stage of the game with relative ease. If a playtester had trouble during this stage, it was often due to forgetting a previously explained game mechanic. Interestingly, the playtesters would largely blame themselves if something like that happened, rather than putting it on a lacking intuitiveness of the game's mechanics.

Some playtesters remarked that the developer user interface could feel overwhelming at their first encounter with it. Oftentimes those same playtesters also remarked that once they were able to figure out how to make the Dev UI work, they actually felt like it was easier to navigate than they expected.

Other situations or remarks that came up multiple times, in no particular order, were:

- Many playtesters tried pressing the spacebar in hopes of finding a jump function.
- Almost all playtesters remarked that they liked breaking walls at the end of the game, and that it felt very satisfying.
- Many playtesters spent a reasonable amount of time trying to interact with the broken door, before either realising it wouldn’t react, or simply giving up.
- Only a handful of playtesters got stuck in the game so harshly that they asked the author for a hint. (Those same playtesters were all able to solve their issues without the author *explicitly* explaining to them what to do.)
- Many playtesters remarked that solving the first developer-only obstacle (the gap in the floor) by increasing their boost speed was one of their favourite moments of playing the minigame.
- Most playtesters were able to solve all puzzles autonomously, both the base game puzzles and developer-only obstacles.
- A number of playtesters remarked that the fluidity and smoothness of the game greatly increased their enjoyment and excitement over the game.
- A number of playtesters specifically remarked that they liked the styling of the game.

One last general remark:

There was little difference in the speed at which playtesters solved the game in regards to their skill level at programming. Overall, the beginner or unskilled programmers were slightly slower than the others, but both the experienced and intermediate level programmers completed the game in similar time frames. On average, the completion time of the game was around five to ten minutes, with some playtesters taking a little longer, around fifteen minutes.

**5.2.2.2) Controls/functionality and progression**

The most common functionality and control issues were found in playtesters either forgetting what some previously explained keys would do, or in some cases not immediately understanding that they could use the mouse to interact with the developer user interface.

On average, most playtesters were able to understand and use the game’s functions and controls well. There is some room for additional clarity though, perhaps through the use of more suggestion UI panels, or by adding additional intuitive light-up elements to objects in the scene.

For example: if a player is standing in front of a developer screen for a long time without interacting with it, the suggestion UI element that shows the ‘E’ key could fade in again.

Game progression wise, the points where players were stuck the longest were the second door, the first dev-only obstacle, but mainly, the third dev-only obstacle. As mentioned before, players were only really stuck at the second door and the first dev-only obstacle because they had either forgotten or misunderstood that they had to press their ‘E’ key to interact with the relevant game element.

At the third dev-only obstacle however, some players were stuck for minutes at a time. They tried many things to get the door that was next in the maze to open, a handful of playtesters even tried brute-forcing multiple variable combinations, in hopes of somehow solving a hidden puzzle.

In the end, oftentimes after reading the functions of the Interaction block of the Dev UI out loud, they would understand that they could reclaim keys back from doors they had previously opened. They’d then move to one of the previous doors, press the ‘E’ key, and use their newly reclaimed key to open the second-to-last or last door of the maze.

One final issue that playtesting uncovered about game progression, is that some playtesters did not understand that them destroying the entire level with the power of the custom if-statement meant that the game was over. They would make sure that they had destroyed all possible walls, looking for a clue for the game to continue. At that point, the author had to step in and let the playtester in question know that the game was over.

#### 5.2.2.3) Accessibility/visibility

Most accessibility in the game was very clear. Some remarks, in no particular order, include:

- It was sometimes difficult to read the pseudocode on top of the doors.
- Some playtesters never noticed the shape on the front of doors, and/or didn’t realise that the shape referred to the key needed to open that door.
- Almost none of the playtesters were able to figure out that the boost speed could be set to any number the playtester could think of if they used the input field instead of the slider.
- None of the playtesters were able to make the connection that the glitching pseudocode and randomly jittering icon on the front of the broken door referred to the fact that it did not have a ‘correctKey’ in the scene at all.

#### 5.2.2.4) ‘Introduction to programming’ aspect

Of all twenty playtesters (including the initial playtesters), five were beginner or unskilled programmers, eight were intermediate level programmers and seven were experienced programmers. As mentioned before, every single playtester was able to complete the game in its entirety eventually.

All playtesters were asked the same question near the end of the debriefing interview:

*“If this was your first encounter with programming, would you have been more interested in programming as a concept than you might have been if it weren’t and how so?”*

Surprisingly, *all* playtesters answered this question with a yes, some, even more excitedly than others.

Remarks that stemmed from this question were, in no particular order:

- The product inspires a playtester to try out programming - they specifically mention that it is way more engaging to learn programming via this minigame over a book, because the product is fun to use.
- That same playtester also remarks that the product makes programming seem less intimidating to them, and that they would love to have an expanded version of the product with more levels and puzzles to teach them programming.

- A playtester remarks that if this program were their first interaction with programming, they might have had some programs with the english language (playtester is not native to the language).
- Many playtesters remark that the programming aspect of the game is easy to pick up.
- A playtester remarks that they can be forgetful about programming terms, and they dislike that they have to look up programming terms when getting back into making a program - they then remark that they like how this was not an issue with this product.
- Multiple playtesters remark that they especially like the custom if-statement - especially the experienced programmers remark how it is a great addition to the game as it refers to a real programming function and looks more like real code over pseudocode.
- Multiple playtesters remark that they like how the product saves them from the frustrating experience of bug-fixing: they like the immediate impact from their actions.
- Especially the beginner and/or unskilled programmers are positive about the product's programming introduction aspects.
- One playtester remarks how the moment they realised that game development is possible for them as well was the moment the programming world opened up for them - and that this product would have helped that get that realisation sooner.
- *“I feel like I am cheating!”*
- Overall, a majority of playtesters remark that they would have been more interested in programming as a concept compared to their true first interaction with it, had they had this product instead.

# **Six** *Ethical Considerations*

*Unlike the rest of this report, in part Six - this part, the author can speak in first person.*

## **6.1)** Code of Ethics

### **6.1.1)** Ethical Landscape

The first ethical dilemma I encountered which arises in this graduation project stems from the conflict found in the task of 'de-stereotyping' this project. The de-stereotyping of this project is one of the main ethical requirements needed to address the project's overarching need, and yet there is a counterbalance that also demands attention: a great challenge is found in balancing the avoidance of stereotypes against losing authenticity. Creating a de-stereotyped game-development exercise may lead to a situation where the exercise becomes so neutral that it loses authenticity, personality, and character, and ultimately fails to engage the prospective students genuinely.

What we currently see as typical game-dev exercises is that they are quite heavily themed: 'make a random generator that creates a dungeon layout', 'create a fantasy game item inventory slot system' or 'design a small-scale 1v1 shooting simulator'. All heavily themed.

For the challenge, I currently am looking at reducing such types of theming by identifying the 'core' gaming values that lie underneath these themed exercises. I would then return to those principles, and create the minigame based on them.

What makes a game that plays in a dungeon setting interesting, is the *strategic* aspect of the game, just like the challenge of required *reflexes* for the 1v1 shooter, and the *experience* of a sprawling fantasy game. When attempting to return to those enticing game principles, there is, as I expanded upon in the first section, a risk in removing so much theming that the game development exercise becomes so uninteresting that it fails to captivate a prospective student.

On one hand, there is a need to remove stereotypical elements of an exercise to attract a more diverse audience. On the other hand, there's a risk of creating an exercise that feels forced or disconnected from the reality of game development, potentially failing to provide an accurate representation of the field. While I'll try my hardest to make it work for this project, a perfect balance between these two requirements is not truly possible. If one wishes to design a truly de-stereotyped minigame, it *will* lose a very large amount of authenticity. Because to create a game that is even remotely interesting, fun, or challenging, one *needs* a certain amount of theming to at least spark some sensations in the player. On the other side of the coin, if one were to invest into the theming of a game development exercise ad maximum, it would be so themed that it would lose its broader appeal, and would likely only be engaging to the original designer.

So, paradoxically, maximum investment in de-stereotyping would prove detrimental and maximum investment in theming would prove detrimental, yet attempting to find a balance between the two also has an ethical issue: if we don't adhere fully to one side, then can we truly say that we are 'de-stereotyping' or 'theming' the project at all?

The second ethical dilemma I identified in this graduation project is found when diving into the challenge of navigating unconscious bias in the minigame's design. Designing a de-stereotyped minigame requires careful consideration to avoid unintentional biases. As a designer in general, but specifically for this project - with the goal being to create a minigame that is de-stereotyped, with reduced theming so as to not cater specifically to a stereotyped group of potential prospective students - a designer will want to avoid biases when executing on this challenge. In the context of designing the minigame to attract a more diverse audience, several types of biases can occur, such as a cultural bias, where unintentional inclusion of elements that may favour a particular culture or background can make the exercise more relatable for some groups while potentially alienating others. Or a gender bias, where despite efforts to de-stereotype, the exercise might still unintentionally favour or discourage participation based on gender, reinforcing traditional gender roles in game development, even on a smaller scale. It's challenging to completely eliminate all unconscious biases during the design process. Other biases that can influence design are those such as a socioeconomic bias, where certain aspects of a game-dev exercise may assume a certain level of economic privilege or access to resources, potentially excluding individuals from lower socio-economic backgrounds. For example, when high end computing is required for an exercise but the student cannot access a powerful enough laptop or computer for themselves to work on the exercise properly. Similarly we can find problems in an educational bias where design elements that assume a certain level of prior education or exposure to programming may unintentionally disadvantage individuals who haven't had the same opportunities. Despite my best efforts, there may be aspects of the exercise that inadvertently reinforce certain biases.

Interestingly enough, in the act of de-stereotyping itself, one must make several assumptions on the demographics of the (larger) target audience to properly de-stereotype in the first place. So, despite the fact that a goal of the project is to avoid unconscious biases in my design, I must *consciously* make assumptions on my demographics in order to de-stereotype the minigame's theme? Problematically, the issue with *unconscious* bias is that it occurs despite the developer/author's efforts not to reinforce them.

The dilemma here is found in the proper navigation of this challenge. One can acknowledge the limitations of the developer/author and the design process, strive to minimise bias and create an inclusive environment for all potential applicants. Despite that, by doing all of the above, the developer/author *will* subject themselves to falling for unconscious biases, simply because they are unconscious biases.

### **6.1.2)** Code of Ethics for this Project

For this project I can identify a couple of relevant moral principles. I use the Institute of Electrical and Electronics Engineers's Code of Ethics [14] and identify some myself. The descriptions of the principles below are not directly from IEEE, they are similar but adapted by me. IEEE includes a broad range of principles, some of the key aspects of the IEEE Code of Ethics that are specifically relevant in the context of creating a de-stereotyped game development exercise in my project are:

1. **Integrity and Honesty (I),** *upholding integrity involves being honest and straightforward in all professional interactions and activities.*
   The principle of integrity is relevant to the project as I attempt to navigate the balance between de-stereotyping and maintaining authenticity. Being honest about the

intentions behind removing stereotypical elements and ensuring that the exercise accurately represents the core principles of game development is crucial. Integrity should guide the project - to avoid creating a disingenuous or forced experience for students, creating ‘trust’ and authenticity in the learning process made with these game-dev exercises is of great importance, and why I should consider this principle when planning my project.

2. **Inclusivity and Diversity (II.),** *a principle that emphasises the importance of promoting diversity and ensuring inclusivity in the project.*
   Inclusivity and diversity are - as mentioned in the previous outline topic - critical in the context of the project. By aiming to attract a more diverse audience to game development, there is alignment with the principle of inclusivity. It involves creating an environment where individuals from various backgrounds, cultures, genders, and socioeconomic statuses feel welcome and represented. This translates in the project to the proper de-stereotyping and ‘un-theming’ of the game-dev exercises. Considering inclusivity helps avoid unintentional biases and ensures a broad appeal for prospective students, it is important that I consider this principle when planning my project.
3. **Respect for Individuals (II.),** *respecting individuals involves treating all people with dignity and recognizing the value of diverse perspectives.*
   This principle is mainly on addressing biases, especially gender bias, cultural bias, and the other potential sources of discrimination outlined in the previous outline topic - additionally this principle can be seen as an extension of the previous principle. Respecting individuals means acknowledging their backgrounds, experiences, and identities without reinforcing stereotypes. It guides the project to create exercises that resonate with a wide range of students, adding to create an environment where everyone feels respected and valued, which I consider important for the end goal of this project.

Additionally, while not part of the (more general) IEEE code of ethics [14], I identify this moral principle:

4. **Accessibility,** *ensuring accessibility involves making technology and information available and usable to all individuals, regardless of their abilities or resources.*
   Accessibility is a key principle in the context of socioeconomic and educational biases. It's important that I design my minigame so that I don't assume a specific level of economic privilege or prior education. Considering accessibility ensures that the project doesn't unintentionally disadvantage students who may have varying levels of resources or educational backgrounds. This principle promotes fairness and equal opportunity for all potential applicants, which is why I consider it important for my project.

From the KIVI [15]:

5. **Technical Competence (VI.),** *maintain and enhance our technical competence. We are familiar with our own limitations and we shall make others aware of these limitations.*
   Honest and accurate reflection on my personal work and achievements is very important. Making mistakes or working suboptimally is completely normal and natural, but it is my responsibility to reflect on my work properly to learn from those mistakes in turn, and to use the knowledge gained from my reflection to re-optimize my working methods and attitude towards the development of my project.

## **6.2)** Design Engagement by Ethical Decision Making

### **6.2.1)** Creative Technology’s Ethical Cycle

Before this course I had never considered CreaTe’s design cycle to be special at all, I assumed it was the main way to design generally. Personally I like that Van de Poel & Royakkers’ design cycle starts with a problem analysis. It suits CreaTe’s design cycle and my own. If one encounters something they don’t like - no matter the context - the first step is to ask, why is there a problem with this? After that, it’s time to figure out what to do about it. Sometimes that can be as simple as asking someone a question or imploring them to perform a task, but sometimes one has to really sit back and properly design *how* exactly they’re going to get that house-wide in-your-ceiling custom-LED-strip hooked up to sync with the background colours of that Netflix show you like to watch. CreaTe’s approach to problem solving is exactly that: take effort and consider multiple approaches, cycle through possible solutions until you find the one which makes sense and ‘*clicks’* - and then execute it. In CreaTe it’s certainly not as if there is no moral analysis at all, but it usually isn’t the item up on the docket. The case of my graduation project is different from the usual design cycle in Creative Technology. My project’s main objective is direct product design, by request of my client. She has asked me to create a game-design ‘exercise’ - which after iteration became the ‘minigame’ that we’ve been discussing in this report so far - that can be displayed on open days of middle schools, where the kids there can be introduced to programming and game development in a fun way. What makes the request special is the challenge to create this minigame with ‘neutral’ styling, rid of elements that are typically more attractive to stereotypical ‘gamers’ and (potential) programmers.

### **6.2.2)** Van de Poel & Royakkers Ethical Cycle [16]

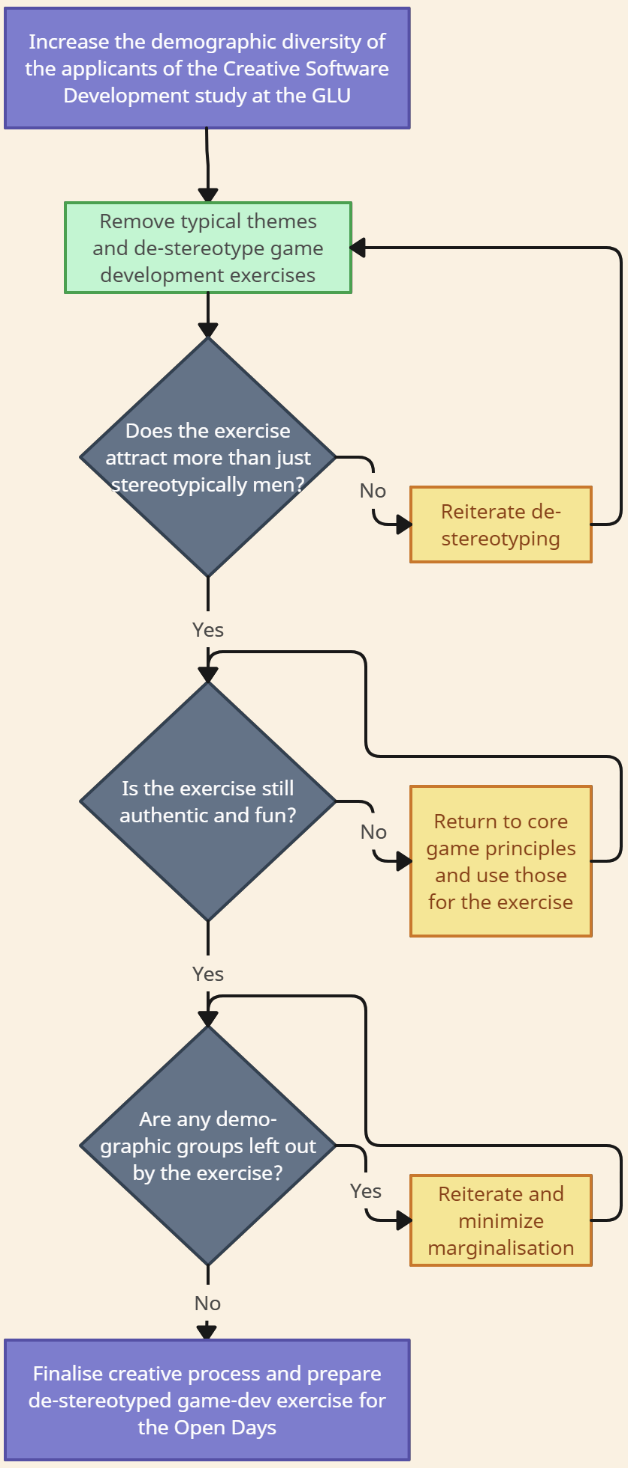
The very nature of that request is different from typical problem solving, because the proper approach method was not clear initially. Stereotypes and the reality behind them often have some ambiguity between them, and the problem solving process for the task has therefore been different. The very first step of Van de Poel & Royakkers ethical cycle asks me to recognise that there is a moral or ethical issue associated with the technology that I am going to develop. There are multiple, but interestingly one of the main ones is found in the fact that I do not have the time or resources to complete a multi year analysis of gamer, programmer, teenager and their interests’ stereotypes and realities. Therefore, the product that I end up making will, despite my efforts, be flawed morally and ethically in what I have been tasked to do.Wherever possible, I do incorporate both CreaTe’s design process and an ethical cycle that is basically as Van de Poel&Royakkers’, just in a limited capacity.
Two other ethical dilemmas concerning my graduation project have been discussed in section two of this report: de-stereotyping while maintaining authenticity and navigating unconscious biases in design.
For the third step in Van de Poel & Royakkers ethical cycle, the options for actions I chose to undergo for this project was ‘depth’ based. I did not create multiple loose prototypes, but instead focused on improving and refining a single prototype. Where necessary, I would reiterate and occasionally trace back multiple steps in development to revise the design and approach of the prototype.
The fourth step and the fifth step of the ethical cycle suggest that I assess my prototype using ethical principles and assess that ethical framework too. To reflect on for example its

inclusivity, at one specific point in the design process this directly came up when working on the integration of the ‘developer’ functionality into the base game. To immerse the ‘developer’ functionality’s pseudo-code style, I created scrambling random text on screen, to mimic a glitching interface. Additionally, I added post-processing effects to the camera that made the camera shake, drop colour bleeding on object edges and horizontal line jitter when the player got close to the glitchy text in the game. This is where I took a step back to assess inclusivity. The jitter and glitchy effects were quite drastic, potentially excluding people with seizure sensitivity (for the camera effects) or dyslexia (for the glitchy text) when the effects got too intense.

As step six in the ethical cycle suggests, I greatly reduced the intensity of the effects and they are way less jarring. The effects are (at this stage of development) not *replaced* though, so there is still a chance I am excluding people of the above mentioned demographic from experiencing this minigame to its fullest. I will (re-)assess this in step seven and eight; they will come in the future of my project's progress. Soon I will finish development and start playtesting. I will reflect on the outcome and use the information for my project accordingly.

## 6.3) Applied Ethics

### 6.3.1) On Fledderman's ethical design problem-solving techniques


Increase the demographic diversity of the applicants of the Creative Software Development study at the GLU
Remove typical themes and de-stereotype game development exercises
Does the exercise attract more than just stereotypically men?
No
Reiterate de-stereotyping
Yes
Is the exercise still authentic and fun?
No
Return to core game principles and use those for the exercise
Yes
Are any demo-graphic groups left out by the exercise?
Yes
Reiterate and minimize marginalisation
No
Finalise creative process and prepare de-stereotyped game-dev exercise for the Open Days

Regard the figure on the right of this paragraph:
This flowchart represents the ethical aspects of the creative process that I will partake in when designing the de-stereotyped game development exercise.
We start with the challenge of this graduation project: the GLU in Utrecht wants a more demographically diverse set of applicants for its creative game development course.
That will be achieved by me designing a de-stereotyped game development exercise which is to be displayed on an open day of the GLU. The product will be a very elementary look at game development, by looking at the essence of what makes games - and its exercises - fun: the strategic, reflexive and experience based concepts beneath typical themes.
The goal of that is to attract prospective students who may not fall into the category of typical applicants: male young adults with gaming as their main hobby.

The flowchart follows simple questions going off of Fleddermans flowcharting [17], by which we determine if the product is coming along the way we want it to:
After the initial de-stereotyping process, we check if the exercise is already attractive to a more diverse group of prospective potential students. If it isn't, reiterate the de-stereotyping. If it is, we come to the next step:
Is the exercise actually still captivating, engaging, fun, yet still authentic? If it isn't, reiterate by returning to the core game principles of reflexive, experience and strategic gaming. If it is, we continue to the last important step:
And any demographic groups now left out by the created exercise? If so, reiterate the product, minimise marginalisation so long as there is enough time left for the finalisation of this graduation project.
If the amount of demographic groups left out is minimal, then finalise the creative process and prepare the de-stereotyped game development exercise for the Open Days of the GLU and hopefully witness how more types of prospective students find out how interesting and how fun software development can be.

### **6.3.2)** On Markkula's ethical risk sweeping principle [18]

In Markkula's toolkit, we find the concept of ethical risk sweeping. It's a technique that helps identify potential ethical risks in a project. Applied in the context of my project, the steps look as follows:

*One: Identify risks*

Most of these ethical risks have been discussed at some length in this report already. I identify: a risk of unintended unconscious bias, a risk of misrepresentation (due to assumptions made when trying to take unconscious bias into account), a risk of lost authenticity (which leads to poor engagement) or a risk of unequal access (due to socioeconomic background or physical ability).

*Two: Assess impact*

Unintended unconscious bias: High impact - could compromise the initial goal of inclusivity.

Misrepresentation: Medium impact - could lead to misconceptions in the design.

Lost authenticity: Very high impact - could compromise the goal of captivating the prospective students.

Unequal Access: High impact - could lead to shallow representation due to a lack of demographic diversity in prospective students that try the minigame.

*Three: Evaluate likelihood*

Unintended unconscious bias: Medium likelihood - this project has a focus on making sure this doesn't happen as one of its core requirements and principles. That said, unconscious issues are unconscious for a reason.

Misrepresentation: Low likelihood - minimizable with reasonably careful design.

Lost authenticity: Low likelihood - balance of de-stereotyping versus losing authenticity is one of the main challenges of this project's execution.

Unequal Access: High likelihood - the project will not directly adapt to accommodate people who would have difficulty getting access to it.

*Four: Prioritise risks*

Unintended unconscious bias: Medium priority - important and will be focused on during execution.

Misrepresentation: Low priority - will be focused on, but not with high frequency or intensity.

Lost authenticity: High priority - focus on this risk is part of the core design of this project.

Unequal Access: Low priority - while impact and likelihood of this ethical issue is high, it is not within this project's resources to focus on or solve. This would be something to focus on in future work.

*Five: Develop mitigation strategies:*

Unintended unconscious bias: Conduct bias reviews with diverse focus groups to identify and mitigate biases.

Misrepresentation: Ensure that core gaming principles are present while minimising excessive theming.

Lost authenticity: Conduct playtests to identify and wring out less engaging parts of the project.

Unequal Access: Where possible, design the game to run on minimal computing resources and potentially make the minigame accessible on multiple platforms.

### **6.3.3)** On ethical theories

*One: Utilitarianism:*

Utilitarianism suggests that the right thing to do, is that which brings the most happiness of benefit to the greatest group of people.

In Engineering, that would refer to the right thing to do being the creation of products or systems that help the most people possible.

*Two: Rights-Based Ethics:*

Rights-based ethics is an ethical theory which suggests that all individuals have certain basic rights, and that these should be respected and protected. In rights-based ethics these rights are seen as inherent to all humans, regardless of the circumstances.

In Engineering, it would mean having your design choices consider the rights of all the people that will be affected by your technology.

*Three: Duty-Based Ethics (Deontology):*

Duty-Based Ethics is more of a theory that focuses on following certain moral rules and duties, no matter what the outcome or consequences may be.

In Engineering, this would mean sticking to specific ethical principles or rules, whether or not that will actually result in what the engineer was aiming towards, or whether or not this protects individual rights or increases overall happiness.

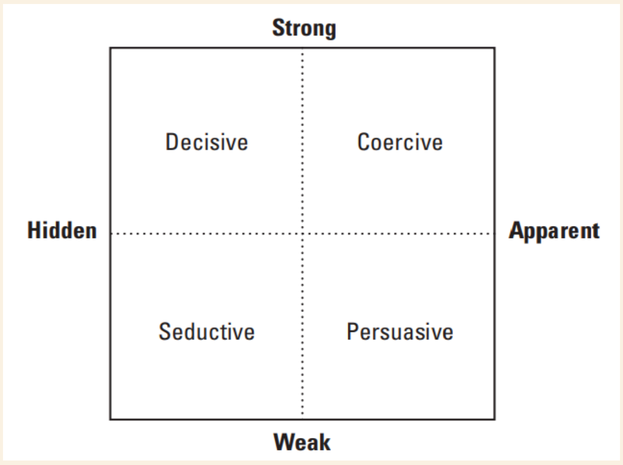
*Applied to my project:*

For my GP, in regards to Deontology, the main objection is that it can lead to very rigid or inflexible decision-making, due to the requirement of constantly considering the ethical principles the engineer is sticking to. While I don’t consider my GP to be one that will directly contest any ethical principles that at least I myself adhere to, I can see how people with different political and social views might have an issue with it. My task of trying to sway away from a stereotyped (typically male as my research so far has indicated) type of person that would be interested in my clients course, to a more neutral approach, where anyone with a more typical baseline interest in programming will (hopefully) be attracted too, might not sit well with people or cultures that still place heavy enforcements on gender differences and other such prejudiced social requirements. Fortunately, those cultures or people are not directly related to the setup and execution of my GP, so I will at the very least at that point not have issues with this moral objection ‘they’ might experience. If that becomes a problem when my GP has finished, will be a question for the future. I personally don’t see it as worrisome.

## **6.4)** Theoretical Discussion

### **6.4.1)** Driver for change

The project serves as a "driver for change" by challenging and reshaping common practices and behaviours in the field of game development education, and by truly trying to make a change: this change being in the demographics of applicants of a software development course. The project attempts to alter traditional game-dev exercises, shifting them towards inclusivity, diversity, and ethical considerations. By de-stereotyping game development exercises, the project aims to change the narrative around software development, encouraging a broader, more varied participation in the game development field, contributing to a more inclusive and representative learning environment.



Regard the following figure on the right:
Tromp, Hekkert & Verbeek define how individuals experience influence being exerted on themselves, and they define it using the axes of *force* and *salience*: *“A design can exert influence that can vary from weak to strong (force), and a design can exert influence that can vary from an implicit to a more explicit manner (salience). Based on these two dimensions of exerting influence, we distinguish four types of influence: coercive, persuasive, seductive, and decisive influence. A product can coerce, persuade, seduce, or decide for somebody.”* [19]

A de-stereotyped game development exercise, in the way that I will be creating and presenting it - for development-inexperienced young adults on an open day for a game-dev course in higher education - will have a highly implicit influence by nature.
A heavily themed and stereotyped exercise will on the contrary be highly apparent. It will by design attempt to capture the attention of those nearby, and try to captivate them with spectacular visuals, fantastical stories or lightning fast reflex-tests. When all of these themes are removed, and the core game principles are laid bare, then attracting bystanders will be much more implicit. It is much more likely that the bystanders think their motivation to try out the game-dev exercise is intrinsic, but it isn't. It is the opposite: a de-stereotyped exercise is more likely to speak to a bystanders core principles too, instead of appealing to themes and fantasies they may already know the like or don’t like.
Returning to the model of Tromp, Hekkert & Verbeek: the product I will be developing in my project is implicit and hidden in nature, and will likely have a weak effect on the axis of *force*. The product is supposed to speak to the hidden feelings the prospective student might have, where programming, software development and game development and their interesting applications are designed to subtly speak to bystanders. That said, the product will of course never force a bystander to make a decision. Bystanders might be attracted to try out the game-dev exercise, but it’s eventually up to them. The product that I will design with my project will be seductive in nature.

### **6.4.2)** Synthesis

The project faces some ethical dilemmas as discussed in section two, where we seek to strike a balance between de-stereotyping and authenticity, and between navigating unconscious bias versus making assumptions on groups of people. My Code of Ethics to handle ethical issues with my project contains focus on integrity, honesty, inclusivity, diversity, respect for individuals, accessibility and a focus on maintaining technical competence through honest and reflective reiteration. I compare the ethical cycle of Creative Technology and Van de Poel en Royakkers, and go through the latter one step by step. I identify the importance of problem analysis, moral and ethical considerations, and iterative design. I highlight the effectiveness of the ethical cycle on my project by expanding on its effect on my project: I made adjustments to minimise exclusion by changing intense visual effects in the project, and I have planned further assessment during playtesting. For the applied ethics I outline the ethical design process using a graph in Fledderman's problem-solving style. The graph shows how to ensure that the minigame appeals to the desired diverse demographics while maintaining the authenticity it requires to keep the prospective students engaged. The iterative process checks on inclusivity and minimises marginalisation. Next up, I discuss Markkula's ethical risk sweeping principle to identify and mitigate potential ethical risks in the project. The identified risks include unintended unconscious bias, lost authenticity, misrepresentation, and unequal access. I assess the risks for their impact, likelihood, priority and I define strategies to address them. On ethical theories, I discuss utilitarianism, rights-based ethics and duty-based ethics. I dive deeper into deontology and its relation to the project and how it can lead to very rigid or inflexible decision-making. Using Tromp, Hekkert, and Verbeek's model, I inferred that the de-stereotyped minigame will exert implicit and weak influence, to appeal to intrinsic motivations rather than explicit themes. The goal of the minigame and its intended design is to subtly engage prospective students by highlighting the core principles of game development. The seductive approach looks to create a more inclusive and representative learning environment, without coercion. The project serves as a 'driver for change' by challenging the current field of game development education.

### **6.4.3)** Limitations

The project faced some limitations. The ethical issues mentioned in this report were assessed comprehensively, but the limited scope means some potential biases and accessibility issues can certainly occur during the project's development. Additionally, while the project aims to create an inclusive game development exercise, it is challenging to cater to all demographics and individual preferences within a single minigame.

### **6.4.4)** Future Work

Future work should involve more extensive and diverse focus groups to further identify and mitigate unconscious biases. Additional resources should be allocated to develop multiple prototypes instead of just one 'depth based' prototype, and thorough playtesting across various demographics should be conducted. Improving the accessibility of the minigame to ensure it can be experienced by as many demographics as possible is of great importance.

*Artificial Intelligence Statement*

*For part six of this report, text generative artificial intelligence aid was used to a very limited capacity. Text generative AI was only used for text formatting, basic suggestive writing style, and limited summarisation of existing rapportage.*

# Seven *Conclusion*

This project attempted to answer if a ‘neutral’ mini-game with programming features can attract potential students to a software development course, when they might be interested, but are swayed away due to stereotypes.
The challenge of this project has two parts. The first is the de-stereotyping of the mini-game, by returning to base gaming concepts and removing heavy theming. The second challenge is the intuitive integration of gameplay elements that introduce programming aspects to the player.
For the first part of the challenge, background research was done on the stereotypes mentioned in the research question, the influence they have, and the influenceability of those stereotypes. The research shows that the stereotypes do have an effect, but, potential students are still influenceable by what they encounter, see and experience.
To use the opportunity that influenceability brings, a mini-game was ideated. By returning to base gaming concepts, the decision was made to create a three dimensional puzzle game in the Unity game engine, with neutral styling, a soft colour palette, and strategy as its core underlying game concept.
The mini-game features a game-within-a-game concept in order to create a gameplay layer that introduces players to programming. The gameplay layer is created by granting players ‘developer access’ - players can directly access and control important gameplay variables.
After solving basic puzzle elements, the player encounters obstacles that seem unsolvable.
The access to the game’s variables is used to solve these increasingly more difficult programming related puzzle challenges. If the player is able to complete the game, they are rewarded with a fun challenge: a piece of programming pseudocode pops up, which bears close similarity to a real common programming function. This custom if-statement gives players an even deeper look into the concept of programming.

Twenty playtests were performed, with playtesters that possess varying programming skills. They range from no skill to high skill, and they tested the game’s overall experience, accessibility, and its impact as a tool to introduce players to programming as a concept.
The playtests were overall experienced very positively and all playtesters were able to fully complete all puzzle elements of the game.
When asked if they would have been more interested in programming as a concept, had they had this product as their first introduction to programming, the answer was largely a *yes*. The results gathered from playtesting indicate that a de-stereotyped minigame with developer access features can help attract prospective students to programming and by extension, to a software development course. The product is ready to be used on open days as an introduction to programming.
Some future work should go into re-visiting the intuitiveness of existing gameplay features, solving some accessibility issues found in playtesting. Other than that, the main areas for future work stemming from this product are found in the form of product expansion. New levels, expanded developer access features and more challenging puzzles can greatly increase the product's effectiveness

# References

[1] B. Engelstätter and M. R. Ward. (2022). *Video games become more mainstream* [Online]. Available: https://doi.org/10.1016/j.entcom.2022.100494

[2] D. Williams et al. (2008). *Who plays, how much, and why? Debunking the stereotypical gamer profile* [Online]. Available: https://academic.oup.com/jcmc/article/13/4/993/4583542

[3] M. D. Griffiths et al. (2003). *Breaking the Stereotype: The Case of Online Gaming* [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/12650566/

[4] K. C. Chappetta et al. (2020). *Gaming roles versus gender roles in online gameplay* [Online]. Available: https://doi.org/10.1080/1369118X.2020.1764608

[5] J. Kivijärvi and M. Katila. (2021). *Becoming a Gamer: Performative Construction of Gendered Gamer Identities* [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/15554120211042260

[6] R. Kowert and J. Oldmeadow. (2012). *The Stereotype of Online Gamers: New Characterization or Recycled Prototype?* [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/12168.23066.pdf

[7] B. Paaßen et al. (2016). *What is a True Gamer? The Male Gamer Stereotype and the Marginalization of Women in Video Game Culture* [Online]. Available: https://link.springer.com/article/10.1007/s11199-016-0678-y

[8] R. L. Harrison et al. (2016). *Gamer Girls: Navigating a subculture of gender inequality* [Online]. Available: https://www.researchgate.net/publication/311463026_Gamer_Girls_Navigating_a_Subculture_of_Gender_Inequality

[9] T. Morgenroth et al. (2020). *The Gendered Nature and Malleability of Gamer Stereotypes* [Online]. Available: https://www.liebertpub.com/doi/full/10.1089/cyber.2019.0577\

[10] J. Vilasís-Pamos. (2020). *How do teens define what it means to be a gamer? Mapping teens' video game practices and cultural imaginaries from a gender and sociocultural perspective* [Online]. Available: https://doi.org/10.1080/1369118X.2021.1883705

[11] F. Faenza et al. (2021). *The Digital Girls Response to Pandemic: Impacts of in Presence and Online Extracurricular Activities on Girls Future Academic Choices* [Online]. Available: https://www.mdpi.com/2227-7102/11/11/715

[12] S. S. Taher. (2021). *The influence of gender stereotyping and demographic factors on academic choice: The case of the University of Debrecen* [Online]. Available: https://files.eric.ed.gov/fulltext/EJ1351189.pd

[13] H. Goud. (2024). *Using graphic design to attract female high school students to the field of creative coding* [Online]. Available: https://essay.utwente.nl/98473/1/Goud_BA_EEMCS.pdf

[14] *IEEE's Code of Ethics* (2020) [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

[15] *KIVI's Code of Ethics* (2018) [Online]. Available: https://www.kivi.nl/uploads/media/5a587110c2160/2018-01%20Code%20of%20Ethics.pdf

[16] Royakkers et al. (2007) *The Ethical Cycle* [Online] Available: https://www.researchgate.net/publication/225126411_The_Ethical_Cycle
[17] Fledderman. (2012) *Ethical Problem-Solving Techniques* [Online] Available: https://www.iqytechnicalcollege.com/Engineering%20Ethics_Fleddermann.pdf
[18] Markkula. (2024) *Ethical Risk Sweeping* [Online] Available: https://www.scu.edu/ethics/ethics-resources/ethical-decision-making/
[19] Tromp et al. (2011) *Design for Socially Responsible Behaviour* [Online] Available: https://www.jstor.org/stable/41261940