ROBERT GORDON UNIVERSITY ABERDEEN

School of Computing

Honours Report

EoSDPointdevice : a Touhou accessibility mod

Pierre Boeglen 2022

This report is submitted as part of the requirements for the degree of

BSc (Hons) in Computer Science

at Robert Gordon University, Aberdeen, Scotland

I confirm that the work contained in this Honours project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.

Signed : Pierre Boeglen Date : 23/04/2022

Abstract

Touhou Koumakyou : Embodiment of Scarlet Devil (EoSD) is a shoot'em up video game released for Windows in 2002. It is the 6th instalment of the Touhou franchise, and it is the first game of the series to be released on Windows – the previous ones were for the PC-98.

This increased visibility that came with releasing on a more famous platform has made EoSD the game that made Touhou so popular, most notably with its witty characters and its amazing soundtrack.

However, due to the high difficulty innate to its genre, only few people interested in Touhou are actually able to enjoy the games.

The vanilla version works with an arcade life system: if the player gets hit, they come back right where they died, but they lose a life. If all lives are lost, the player gets sent back to the very beginning of the game.

Therefore, I wanted to create a software that would give many fans-to-be a chance to have fun with the series' most popular game.

Table of Contents

Introduction

This report will trace the history of my Honors Project, EoSDPointdevice, which has been in the works for several months. Trial and error was a common theme throughout these months: due to the low documentation available on all the obscure topics I had the chance to explore, a lot of my initial ideas simply did not work – making this project an opportunity to learn important lessons in adaptability.

The report will start with a literature review, exploring the concepts of accessibility in video games, the documentation (or lack thereof) about Touhou 6, and the possible options that can be used in reverse engineering.

Then, the design and methodology will be studied. The gameplay of the game will be detailed, and my initial strategies for implementing the Pointdevice mode will be explained.

Next will be my implementation of the solution – all the main ideas will be explained in detail, why they failed, and how the final one worked.

Finally, I will evaluate the final product, comparing it to what was defined in the requirements earlier this year.

The relevant appendices will be made available at the end of the report, after the references, and they will also be found in the supplementary materials dropbox.

Literature Review

Introduction

The Touhou series is mostly known for its huge online presence, not only in Japan but on the western internet as well. So many of its characters, soundtracks, locations are incredibly iconic – and there are mountains of fan-created content, from simple written fictions, to manga, to full-blown short animated series or video games. But the media that shows the scale of the fandom the best is probably the music: the Touhou wiki (2012) records over 6,200 fan-made Touhou music arrangement CDs. Not mere tracks, but full-blown albums, with proper releases.

However, at its core, the Touhou Project is something much less ambitious: a series of Bullet Hell games, created by a single developer known as ZUN. "Bullet Hell" is a subgenre of Shoot'em up, with the particularity that the game's screen tends to be filled with a lot of slow-moving bullets, rather than a few faster ones (Whitehead J. 2007 p. 18). This puts the gameplay emphasis on slow, precise movement as well as pattern recognition, instead of relying on quick reflexes and reaction time.

The series started in August 1997, with "The Highly Responsive to Prayers" (Team Shangai Alice, 1997), a game released for the Japanese PC-9800, a platform that was in competition with Windows in Japan at the time. For ZUN, that competition did not last long, as all games starting after Touhou 5 – that is, a total of 23 out of the 28 games – were instead published on windows. The first game on Windows, Touhou 6: Koumakyou - The Embodiment of Scarlet Devil, is the one that I will study throughout this project and is also the most popular Touhou game.

As "Bullet Hell" implies, these games are hellishly hard. This is a big reason why the media and fan content surrounding the games are much more popular than the games: the most invested players even coined the term "Secondary" to refer to fans who are not interested in the original games – that is how big that part of the fandom is. Numbers might help put things into perspective: on the website Moriya Shrine, the website where western fans can find the games, Touhou 6 – again, the most popular title – was downloaded 256 thousand times (Nitorium, 2018). Meanwhile, on Youtube, some of the most popular Touhou music tracks have up to several dozen million views – Bad Apple !!'s remix by nomico has 56 million views (Kasidid2, 2009), U.N. Owen was Her? has 11 million views (IAmTehChibeh, 2008).

We could argue that the number on Moriya Shrine is so low because Japanese fans would get the games somewhere else, but the same is true of Youtube: in Japan, nicovideo.jp is the main video broadcasting website. Indeed, on nicovideo, the Bad Apple!! remix has 26 million views (あにら, 2009). This means that the regional audiences overlap is negligible, and that the secondary content really is that much more popular than the games.

The difficulty is not the only factor explaining why so few fans actually played the games – personal taste in game genres, for example, are another reason – but it is a factor, nonetheless. How can these people be helped? How can the Touhou 6 experience be made more accessible, without losing sight of the original, developer-intended experience? This is the main problematic that this project is meant to answer.

Please do note that, due to the nature of the project, the literature on this project is heavily centred around online fan-made multimedia, and there is therefore a lack of relevant academic literature.

Accessibility in demanding video games

One of the easiest ways to do that would be to implement an easier, more accessible way to play the game.

The challenges of easy modes

The possible conflict in the developer's vision

One key issue that can be pointed out in debates about whether or not games should include easier modes is that easier modes could go against the artistic vision imagined by the developers. One of the clearest examples of that is the infamously difficult Dark Souls franchise: its director, Hidetaka Miyazaki, went on record (Dayus, 2019) saying that the difficulty of the games was the way it was because he wanted a unified experience: he wanted all players to have gone through the same challenges and struggles.

However, some games are not designed around a given difficulty, or even around a few prebuilt difficulty options. Some games are fully customisable – Minecraft, for example, offers 6 difficulty options, from the godlike "Creative" mode to the extremely punishing "Hardcore" mode. But it goes further beyond and has dozens of "game rules" that change how various parts of the game interact with the environment. And that is not all – fan made content can add extra difficulty options (Damen, 2020).

This works perfectly with the sandbox nature of Minecraft. However, this will not necessarily be a good option with any game – in games where the challenge is key, giving too much freedom might be a bad idea since the players should not have to feel like they are asked to design the game themselves.

A study (Alexander et al. 2012) on the effects of game difficulty on the player found that players tend to underestimate themselves, and to give themselves an easier experience than what they can handle, or than what they would enjoy the most. Therefore, in a game as heavily reliant on bullet patterns as Touhou, giving too much freedom in changing things like bullet speed could easily make the experience boring.

The player's reaction to difficulty

A "Flappy Bird"-like game was created for the aforementioned study, called Start-Surf. Flappy bird (dotGears, 2013) is a mobile game where the user touches the screen to give vertical momentum to a small bird. The player must guide the bird through as many gates, made of two vertical pipes starting at the top and bottom of the screen, as possible. The key is to balance the vertical momentum: too much and the bird will hit the top pipe, too little and it will hit the bottom pipe. Hitting a pipe means losing.

Start-Surf also consists of guiding a player character through gates, and the player must collect points by passing right in the middle of the gates' openings.

The study tested how much players enjoyed Star-Surf depending on many parameters: the self-assessed level of each player (casual or experimented), the chosen difficulty level, the perceived difficulty. The study also included a comparison between static and dynamic difficulty scaling, but that it is not relevant for this project. That is because Star-Surf is procedurally generated based on very few parameters: the player speed, the space between gates and the height of the opening in a game – whereas Touhou is about hand-crafted intricate bullet patterns.

The other tested parameters are however much more relevant: no matter the self-assessed level, players picked a difficulty below their abilities. They all showed skill greater than what was needed to complete the game, it is clear that players tend to underestimate themselves. This is even true post-game, as once again the player's perceived performance was below their actual performance. This is the case even though the perceived difficulty of the game is below the actual difficulty (Figure 1).

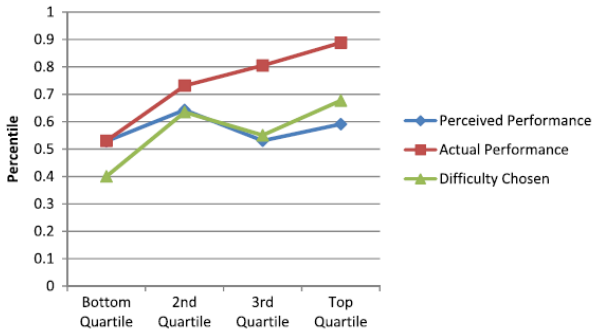


Figure 1

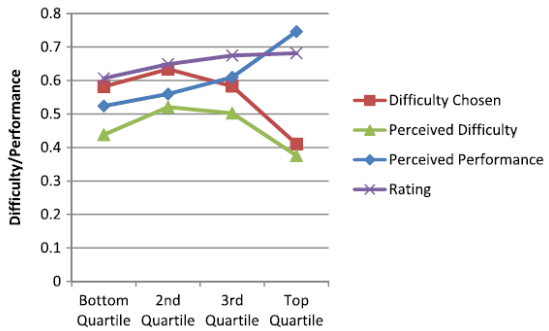


Figure 2

As Figure 2 indicates, the closest the chosen and perceived difficulties are, the higher the rating (the player enjoyment) and the perceived performance are. This means that in order to make the players feel good about their results, it is important to have difficulty options that match a level of experience. The study's authors concluded that there were several steps that could be taken in order to perfect the difficulty system of a game:

First, we can name the difficulties after levels of experience: "easy", "normal", and "hard" are very subjective terms. However, "casual" and "experienced" are a more objective, precise way of measuring a player's ease with games. This is a small change, but it is recommended, nonetheless.

It is also useful to monitor the performance of players. In order to consistently keep the experience interesting, it is possible to give the players a heads up if they are performing particularly well or poorly by reminding them that other difficulty settings exist. The gameplay experience should be neither boring nor too frustrating. However, the difficulty change should only be a suggestion, not a dynamic change – in a game about pattern learning, consistency is key, changing patterns between playthroughs would definitely break that rule.

A solution: accessibility modes

Clears up confusion on both sides

The study presented a lot of good ideas, but there needs to be more thought put into them before implementing them. While they would certainly improve the gaming experience of the players, they could still conflict with developer vision.

So, the solution is simply to separate the developer-intended experience and the accessibility-focused mode. By making it clear that the latter only exists to offer the experience to people who would otherwise be unable to enjoy it, we can respect both the artistic vision and the need for accessibility.

Mark Brown, from the YouTube channel Game Maker's Toolkit, recorded a video essay (Brown, 2018) talking about the idea of accessibility modes in detail.

He starts this video by highlighting the accessibility versus intended vision dilemma, before quickly explaining that, in the end, it's a matter of communication. Options like cheat codes left no confusion whatsoever: they were not the intended way to play. The same is true of mods, or of other pieces of software that are not contained in the game or officially approved. This is the relevant form of accessibility option in this project, as I seek to deliver the mode through a patch.

This is how the sci-fi survival horror game SOMA (Frictional games, 2015) worked in its accessibility mode: an extra piece of downloadable content (DLC) containing a "Safe mode" (Brown, 2018) is available for players who feel like they need it. This mode makes the monsters unable to kill you while keeping their threatening behaviour (and, most importantly, the atmosphere of the game) intact. By

making this mode optional DLC, instead of a classic update that just adds the Safe mode to the difficulty selection menu, the developers make it clear that this is not the intended experience, and players have to make a conscious choice to get out of the beaten tracks.

However, this separation alone is not enough for a Touhou game, seeing as, in the western community, patches are extremely common – most importantly for translations. Indeed, some games have over 30 languages implemented through translation patches (Touhou Patch Center, 2021). This means that adding another layer of explanation to ensure clear communication is advisable.

How to design an « accessibility mode »

I have explained why clear communication is important in presenting difficulty levels, particularly accessibility modes who typically are the furthest modes from the intended experience. Let us take a look at a few more important guidelines, then bring up examples and judge how well they implemented the mode based on the previously established criteria:

Other basic guidelines

By making aspects of a game easier or more accessible, it is very possible that some content will be lost. However, efforts should be put into making sure that as much content as possible is still kept accessible: easier difficulties should be there to offer everyone the opportunity to enjoy the game, not to punish people for not picking high enough of a difficulty.

Another thing worth noticing is that the player shouldn't be shamed or insulted for picking an easier difficulty. Calling them, essentially, a "spineless baby" (Figure 3) the way Wolfenstein: The New Order (MachineGames, 2014) does is really unnecessary. This is an extreme example, but it shows how important clear communication is. Why insult the player when you could just explain that the very easy mode is not the intended way to play?

Figure 3

Other examples

Cuphead (Studio MDHR, 2017) is another case of poorly handled easy mode: entire phases are removed in the boss battles. The extremely high-quality animations are a huge part of why the game is so appealing, and by removing boss phases, easy mode players are deprived of some of these visuals. Not only that, but the last area of the game also requires the player to have beaten every boss in normal mode, depriving the player of a fulfilling conclusion to the game they have likely struggled with for many hours.

Furi's (The Game Bakers, 2016) Promenade difficulty is a similar situation: many boss phases are removed, and boss phases are important content in a boss rush game. At least, the easier mode does not lock you out of an ending.

Much like SOMA, the accessibility feature for Outer Wilds' (Mobius Studio, 2019) DLC "Echoes of the Eye" is not about making the game easier – it is about making it less scary. The execution is simple and effective – the threatening situations you may encounter are less dangerous, and the scary parts do not play some of the loudest, most ominous sound effects. It is the presentation that is notably good: the feature is presented as a toggleable "Reduced frights" setting, the game lets you know that the feature is available when booting it for the first time, and when you try to toggle it on, the game lets you know that these parts are meant to be scary and that the option is only there for people who cannot proceed further due to the scares being too much to handle.

Celeste's (Extremely OK Games, 2018) "Assist Mode" shines in both presentation and execution: the game displays a dialogue box very explicitly stating the role of Assist mode and that it is not the intended experience. In terms of functionality, it takes the form of a new section in the settings menu that lets the player meddle with various global variables (Figure 4).

Figure 4

The settings are very customisable: the game speed can be changed in steps of 10%, the “Air dashes” let you use your dash move in the air either one extra time, or infinitely many times. This lets any player fine tune the game until, ideally, they find themself in their flow zone (Stratakis, 2019) (Figure 5).

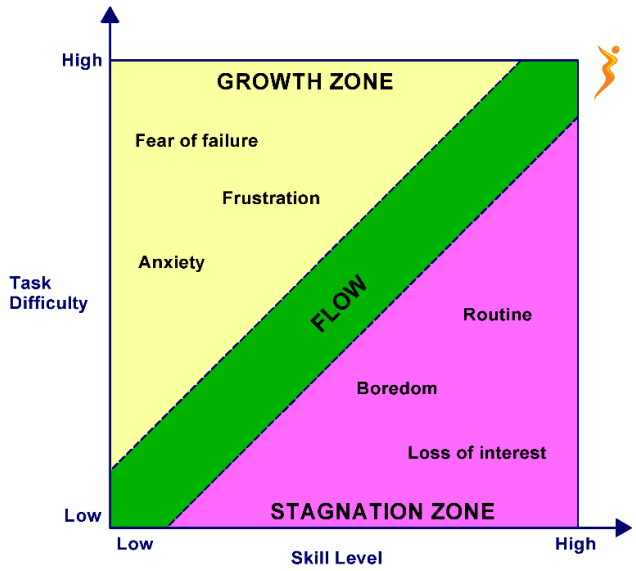


Figure 5

Now, let us take a look at how well Touhou 6 handled its easy mode. The implementation is quite poor, unfortunately: much like Cuphead, you are locked out of a proper ending if you play in easy mode. Not only that, but all other modes kick you out of the game after using 3 continues: continues can be used when you lose all your current lives, and using one locks you out of good endings (but at least you can still access the last stage if you play on normal or higher). However, the extra bonus stage is only accessible with a good ending.

All of these design decisions were deliberate, and in line with the arcade-like spirit of the game, but it still is a shame that they make the game so inaccessible.

There are many ways Touhou 6 could be made more accessible: toggleable invincibility, infinite bombs (bombs are a powerful consumable resource that clears the screen of bullets for a few seconds), increased player damage… but I believe that implementing a system that the developer has used in a sequel will give a result that is closer to the intended experience, closer than if the game’s balance was meddled with.

This system appeared in the 15th title, Touhou Kanjuden : Legacy of Lunatic Kingdom (Team Shangai Alice, 2015). It is called the “Pointdevice” mode: essentially, it separates each stage and boss fight into a dozen chapters. Each chapter contains a few basic enemy waves, or one boss phase. There is no life counter: if the player gets hit, they are thrown back to the start of the chapter, never losing more than a minute or so of progress. This greatly attenuates how punishing death is: by default, losing all lives would kick you back to the main menu, having to restart from stage 1. Pointdevice mode offers a way to fully enjoy the intricate patterns of each chapter, a way to figure out how they work without the pressure of potentially losing dozens of minutes of progress. And at the same time, the experience stays very rewarding, because the players have proved to themselves that they have the skill to beat the game without getting hit even once. Hopefully, that can lead them to trying out the intended mode so they can get the true “Touhou experience”, but it remains rewarding even as a standalone experience.

Touhou 6's architecture

The core of the project is figuring out how to implement such a mode. In order to do it, technical knowledge about the game is required. However, there is little public information about Touhou 6's architecture (Touhou Wiki, 2020). What we do know is that it is the first game released for Windows, in summer 2002, initially directed for Windows 98 through XP.

The same custom-built engine was used for Touhou 6 through 9. It uses the DirectX API set to interact with the computer.

The DirectX8 API

DirectX is a collection of APIs related to multimedia, specifically games and videos, for Windows. It is mostly known for Direct3D, a 3D rendering API competitor to OpenGL (Silicon Graphics, 1994), one of the biggest 2D and 3D rendering API. The explicitly required DirectX components are Direct3D, and DirectSound. However, it is likely the game also uses components like Direct2D and DirectWrite, for example in its user interface. The first version of DirectX was released in 1995 (Microsoft Wiki, 2007), and the one used in Touhou 6 is DirectX 8, which first released in late 2000 (Touhou Wiki, 2020).

Touhou patching

That is currently all that is publicly available about the technical side of Touhou 6. There exists a large patching community (Touhou Patch Center, 2021) that created a tool called THCRAP : TouHou Community Reliant Auto Patcher. However, this community is focused on providing translation patches to the games – their tools are not able to modify the gameplay.

This means that, for the implementation of the mod to be possible, Touhou needs to be reverse engineered.

The options in reverse engineering software

Reverse engineering described the process of figuring out how a piece of technology works by starting from the finished product. In computer science, in the context of this project, it refers to examining a compiled executable file in order to broadly learn the structure of the software – the variables and their addresses, the function calls, etc.

The tools I plan to use in order to achieve this are Ghidra (NSA, 2019) and Cheat Engine (Dark Byte, 2000).

Ghidra

Ghidra is an open-source reverse engineering software used by the NSA whose existence was revealed on Wikileaks in 2017. The USA government then made the tool public in 2019.

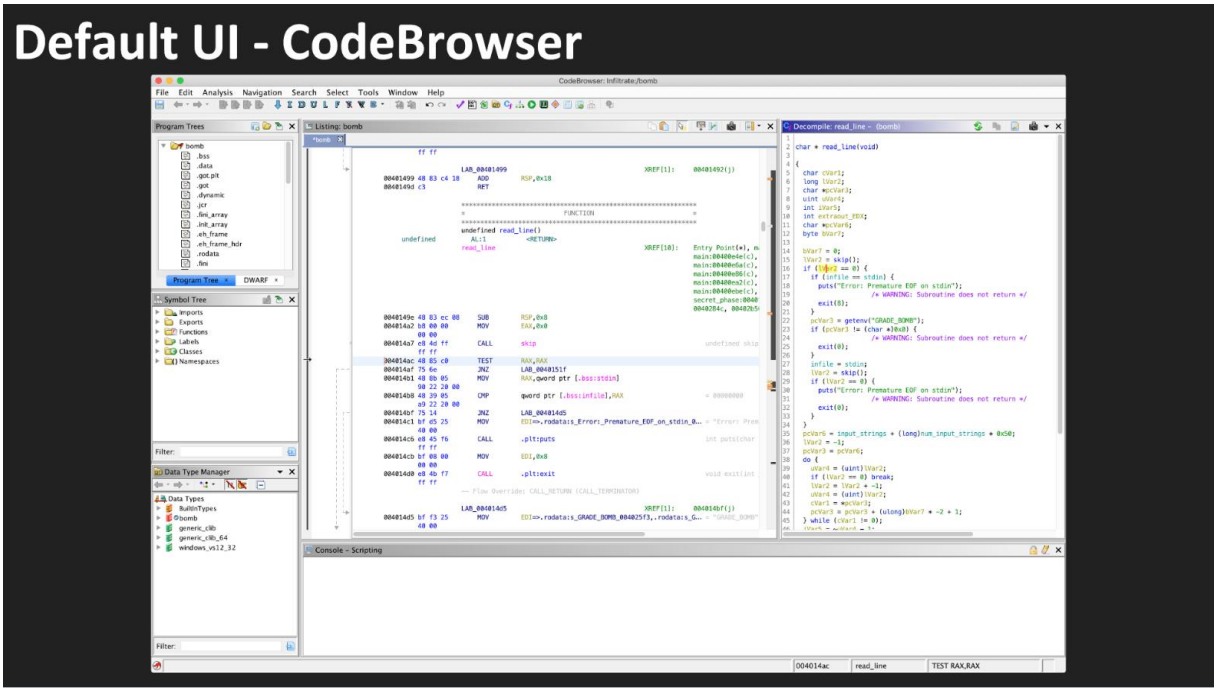


Figure 6

Figure 6 is a screenshot of the default interface of Ghidra, taken from a course about the tool (Bulazel and Blackthorne, 2019). The central window is the disassembler: the assembly instructions that were read from the executable binary.

The right column is the decompiler: it generates C/C++ code that corresponds to the assembly instructions.

The right column contains three windows: the program tree, which highlights the section of the executable that is currently being examined.

In the middle right is the symbol tree. A symbol is essentially the humanly readable version for a code element – for example, a variable. The symbol does not only contain the variable name: it also holds data about the structure of the variable in terms of memory. For the sake of clarity, even though the following example might be inaccurate, we could say that the symbol is what makes the difference between an int32 and a string containing 4 chars: both of these variables are stored on 4 bytes. The disassembler and decompiler will detect a 4-bytes variable, but it is up to the user to figure out what actual type it represents. This is what the symbol tree is for: managing the symbols the user identified throughout the software.

Finally, below that is the data type manager. Just like symbols, complex data types like structs or classes are also essentially lost during compilation. The data type manager allows the user to manage the data types they have identified.

The use of Ghidra can be complemented with a software that allows the user to dissect a process' memory in real time, Cheat Engine

Cheat Engine

This open-source software first released in 2000 (Dark Byte, 2018), although the project's GitHub page (Dark Byte, 2021) only tracks releases from 2015 forward. Even in 2021 the software is still being improved upon, with its 7.3 release being out in August.

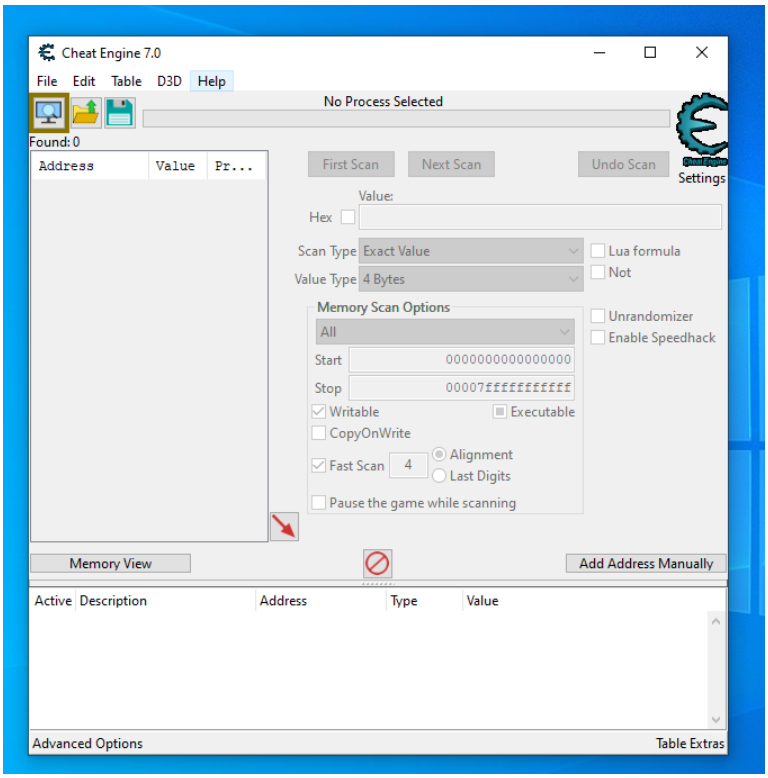


Figure 7

The software is very easy to use. Look at figure 7: with the top left button, we can select a running process. Then, by filling a number in the Value field and pressing First Scan, the software will list all memory addresses that currently contain the given value. By changing that value and pressing next scan, we will obtain all the addresses from the previous list whose stored values have changed to the next number. This process can be repeated as many times as required until we can identify the address containing the value that we seek to modify or identify. Once found, we can change or put restraints on that address's value.

An example of cheat engine with Notepad

In this example, the addresses containing the cursor's horizontal position will be identified. First, I set it to a given number – say, 29. The first scan gave me a list of 30 addresses that point to a variable storing 29. After changing the value and re-scanning, I only have 2 addresses left. Both of these address update in real time as I move the cursor, which prove that they are indeed related to the horizontal position. One is likely the actual position, while the other is the displayed value in the UI. Figure 8 below illustrates this process.

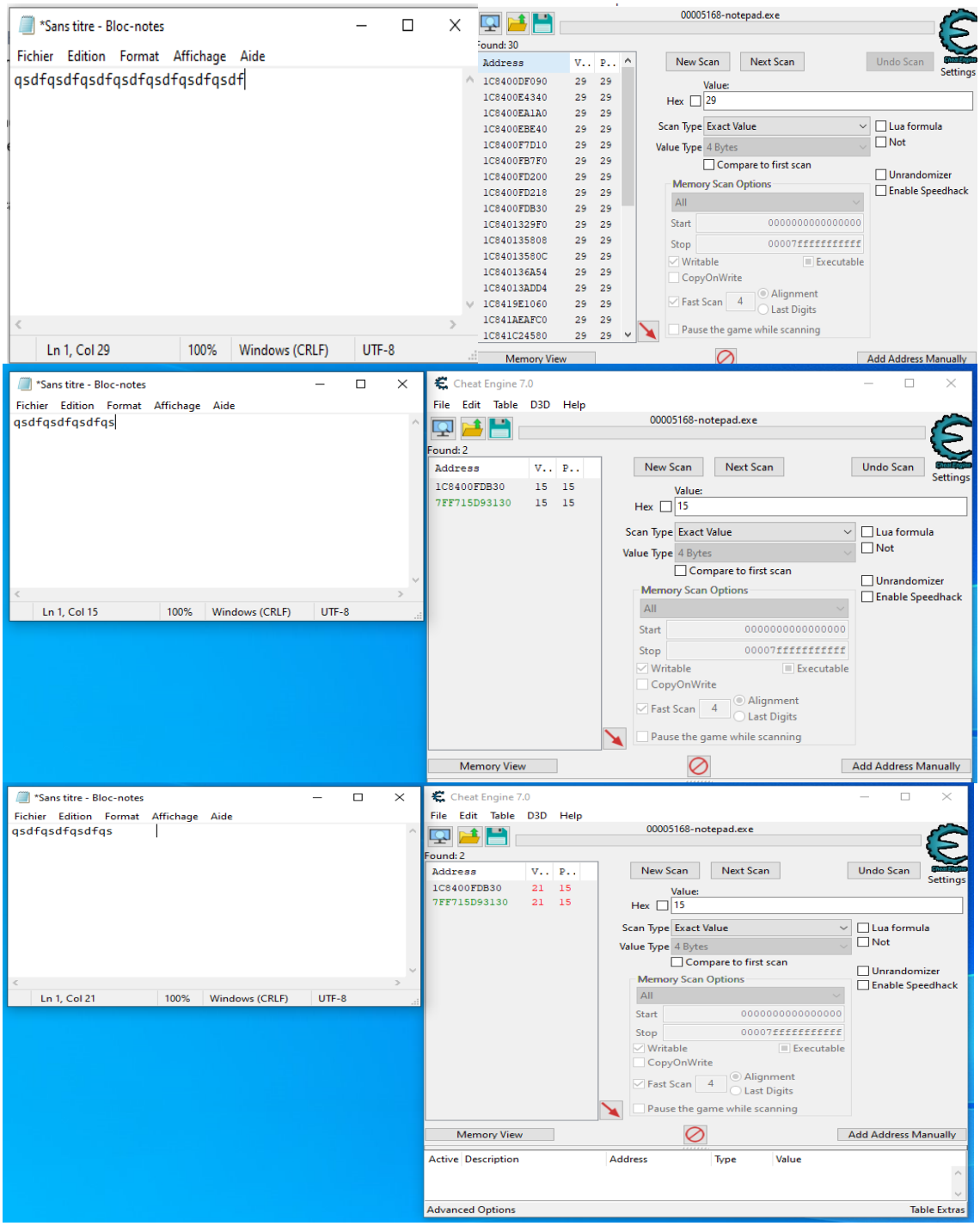


Figure 8

Conclusion

In this review, I explored how and why accessibility modes should be created. I then explained how I could implement such mode in the game Touhou 6, before exploring the tools I plan on using in order to achieve that. If there is one thing this review is lacking, it would be technical details about the game itself: hopefully, this project can end up being a step towards addressing the absence of such literature.

Design / Methodology

How the vanilla game works

First and foremost, the Touhou games fit within the Bullet Hell genre. This is a subgenre of Shoot'em up. In Shoot'em up games, the playable character is typically a plane or a spaceship, progressing through an automatically scrolling level where enemies appear to shoot down the player. The player is able to shoot back, and the main gameplay loop consists of positioning yourself in such a way that the enemy shots don't hit you while your shots hit them. Galaga (Namco, 1981) is one of the most famous examples of the genre.

The subgenre Bullet Hell describes games where the enemies fire a massive amount of slower projectiles, shifting the focus of the gameplay towards reading bullet trajectory and finding safe spots and routes within the intricate patterns. As a result, quick reflexes are not as useful as they are in other Shoot'em up.

The high projectile number that can appear on screen at once can create a "curtain of bullets" slowly descending upon the player, which is where the subgenre gets its Japanese name from. In-game, Touhou is indeed referred to as a 弾幕, pronounced "danmaku", literally meaning "curtain of bullets".

The player can pick a character between Reimu and Marisa. This mainly changes the banter before boss encounters, though it also changes some boss attacks. Reimu is meant to be easier to play, with a wider shot spread, but weaker, while Marisa is faster and more precise. Marisa is harder to handle, but rewards the skilled player with more damage, meaning faster boss kills. Reimu is simply the opposite. Each character has access to two unique secondary shot types, typically referred to as A and B within the community. This means there are essentially four characters: Reimu A and B, and Marisa A and B. The secondary shots are not fired from the character, but from two orbs that fly with them, to their left and right. The orbs are shifted in front of the character when the player is focusing.

Focusing is a very common mechanic in Bullet Hell games that allow the character to slow down. This is what allows for more precise movement. In later games, focusing also reveals the hitbox of the player character, who is much smaller than the actual sprite, being only about 4 by 4 pixels. However, the hitbox is never revealed in Touhou 6.

The game is split in 7 stages. The 6 first ones make up the main game. Difficulty selection is available for them, though the 6th stage is not available on Easy mode. The other difficulties are Normal, Hard and Lunatic.

The player starts the game at stage 1, then proceeds all the way to stage 6. If the player dies, they lose a life and reappear right where they died and lose a bit of power, no further penalty. If all lives are lost, a continue can be used. Using a continue clears your current score, grants you your starting lives back, gives you maximum power, and locks you out of the good endings. It also prevents you from recording a replay of your game.

The last one is the extra stage, a difficult one that can be attempted individually, without having to play all the previous ones. It is unlocked after beating the main game on Normal difficulty or higher without using a continue.

A stage is composed of several waves of enemy, and some boss fights. Typically, the stage starts with some enemies, then a boss fight with few phases near the halfway point, then some more enemies, then another, longer boss fight. The 6th stage does not have a second set of enemy waves, instead going straight from the first boss encounter to the second.

Enemies sometimes drop items: point items and power items. Point items simply increase your score – they give more points the higher the difficulty, and the higher the player is on-screen. A point item collected in normal mode at the top of the screen will grant 100 000 points, whereas the same item collected at the bottom of the screen would only grant around 30 000 points.

Power items are what make your shots more powerful, mainly by increasing the number of shots fired (though Marisa B's lasers are simply made stronger, instead of increasing the number of lasers).

Some mid-stage bosses also drop extra lives and bombs. The other way to gain lives is to reach a given score: 10, 20, 40 and 60 million points. For bombs, on the other hand, the only other way to regain them is to die.

Bombs are a consumable item that immediately clears the screen of every bullet, grants the player a short period of invincibility, and deals massive damage to everything on screen. They are very rare, but they can get the player out of any sticky situation whatsoever. They are essentially extra lives that reward players that are good enough to know when they're going to get hit.

This was all just the basic gameplay, though – things that anyone would be able to describe after playing a few times. Little is documented about the truly inner workings of Touhou 6. It is known that the game was created for Windows 98, using DirectX 8 components and libraries to function. Everything else was created by ZUN: the artwork, the soundtrack, even the engine was custom-made.

With how little is officially documented, most of the relevant technical knowledge about the game was found using Cheat Engine and Ghidra. It was obvious from the start that some variables that could be found with Cheat Engine were going to play key roles within the project – for example, the variable that indicates whenever the player dies.

How Pointdevice mode would work

The name "Pointdevice mode" was coined in Touhou 15. Simply put, it implemented a checkpoint system in the game. This game mode felt so much more rewarding to play, it was exactly what I wanted to implement in Touhou 6.

The key feature is that dying sends the player back to a previous checkpoint, instead of detracting a life from a counter. This really is just that simple in concept - the crux of the project ended up being the exploration of different strategies that could implement such a system.

The rest of the project is about properly defining that mode for the users: by clarifying the distinction between accessibility and easy modes, by explaining that the software does not offer the intended experience.

Implementation

First idea: decompiling the entire source code

Cheat engine allows the user to scan for values within a process, as shown in the notepad example above. Touhou 6's memory was split into two parts: first, the actual code itself – the algorithmic parts of the functions. Then, the variables. These variables are what Cheat Engine detected.

But Cheat Engine can do more than scan values and change them: it can list the instructions that modified these values. Working that way, it was possible to get the address of the Assembly instruction that changed a value, and by looking up that address in Ghidra, one would find what function the instruction was part of. Then, it was possible to start guessing what the function did, what meaning the less obvious variables actually held. Little by little, Touhou 6 as a whole could be figured out. Once everything was identified, the result would have been a large C file that compiles into the Touhou 6 executable. From there, the game would have been modifiable in essentially any way whatsoever. However, that proved to be way too difficult. Too much guessing was involved, and not enough time was available. Another option was quickly considered: code injections.

Second idea: using "array of byte" injections with Cheat Engine

The most feasible way to run a code injection was array of byte (AOB) injections, using Cheat Engine.

The Assembly instructions were defined with bytes. An AOB injection consists of finding a unique array of bytes that would be used as a beacon, so that the code injection can find where to be injected. With the previous method, several useful variables were identified, most notably one that counted the number of lives the player currently had. From that variable, it was possible to identify which instruction removed a life upon death. This was where the injection needed to go.

Once that instruction was identified, I could use Cheat Engine to replace it with a Jump instruction that would point towards some custom code. Actually creating the custom code was the unfeasible part. The idea was to call the function that loads the last enemy wave. With Cheat Engine and Ghidra, I was able to identify that function, but the number of parameters was extremely high. It was necessary to have every right parameter on the stack, in the right order, and all the registers had to be in the right state. Perhaps this could have been feasible with more Assembly experience, but time to obtain such experience was lacking. Once again, another plan had to be made.

Third idea: creating a software to handle savestates

There was but one other context where custom checkpoints are commonplace: emulation. In video game emulation, they are typically called "savestates". In virtual machine management software, they are often called "snapshots", but they are one and the same. They both consist of saving the exact state of an environment, be it a computer's or a console's OS, in a way that can be reloaded in order to return the machine to that state. This is alarmingly similar to what using a checkpoint does: it is essentially a return to a previous point in time.

In order to bring this idea to life, two things were required: the ability to read the memory of a process, and the ability to write into the memory of a process. Both were granted by the Windows.h C++ libraries. Obviously, when working with such low-level concepts, a lot of care had to be put into the actual code, but again, the library offered a lot of functions to handle that.

While C++ did make for a more enjoyable coding experience than Assembly, there was still one problematic part with the idea itself: since the software reads from and writes into a different process, most issues with the code would make the target process crash instead of triggering a C++ error. This also meant that there was no way to know if that idea would even work until it did work.

And as it stood, the software was indeed not working. As I talked about that idea with a friend of mine, thread contexts were mentioned. This was the last piece of the puzzle. In retrospect, it makes sense that simply dumping the memory and forcing it back into the process would not work: information like the location of the instruction pointers would be lost. This information is exactly what the thread contexts are. And once again, there were C++ libraries to get and set thread contexts. Once they were implemented within my code, the software actually worked. Save states could be created and reloaded by typing S (save) and L (load) in the command line interface.

Automating the saving and the reloading was the last step. The reloading was simple enough: in the software's main while loop, instead of waiting to read a character from the CLI, the number of total deaths was read. If it ever hit 1, a reloading to the previous checkpoint is triggered. This was done using the same function that reads the entire memory of the process for the savestate, but this time, simply one address, the one containing the death counter, was read.

The saving automation was more complex. During the early Cheat Engine tests, no variable that tracked the progress within a stage was found, something else had to be used – that thing ended up being the Health value of the enemies.

Let us imagine this scenario: the first 10 enemies of a stage have 40hp. Then, there is a second wave with stronger, 80hp enemies. Finally, there is a boss with 1000hp.

In order to create a checkpoint at each of these sections, an array containing 40, 80 and 1000 could be used. When the game is started, within the main while loop, the value of the address of the enemy's health is read. If it is equal to 40, a save is created, and the array's index is incremented so that the loop now looks for 80. Once 80 is found, meaning the second wave of enemies have appeared, a save is created, and the loop now seeks 1000. This system works very well with a game based on scripted enemy waves.

When testing, most features worked as intended, but there were a few noticeable hiccups, nonetheless. The most notable example is with the saving automation – the health point system was more complex than anticipated. Several enemies can be on screen at once, and their health is dynamically allocated in an array. This means that if a boss appears before the previous enemy disappears, the boss health might be stored within the second variable of the array. Therefore, there were some cases where the hp address had to be padded in order to access the right variable. Since the enemy waves, paths and attacks are fully scripted, the required padding was consistent between several runs, making this task a lot easier to fix.

Besides one function, whose content is based on a Stackoverflow response (KNIGHT J, 2014), all of my code is original, and all the answers to my other problems were found by looking up the official documentation.

Testing & Results

The software has not been shared with other players yet. Since the focus of the project is accessibility, a clear graphical user interface would have to be expected, instead of an intimidating command line interface – and more time would have been needed to implement a GUI. No GUI can be found on the requirements list, however – that is because the project's artifact was initially planned to be a patcher software that would edit the game's executable.

Evaluation

Original requirements

	Must Have	Should Have	Could Have	Will Not Have
Functionnal Requirements	**F1: The patch must be able to run on the english version of the game found on the website Moriyashrine**	**F2: The new mode should not remove the ability to « deathbomb » (the ability to use a bomb for a few frames after getting hit to prevent the lost of a life)**	F3: The new mode could replace extra life drops with bomb drops	**F4: The patch will not implement the score, bomb and life chapter high-score rewards that are present in Touhou 15**
	F5: The new mode must separate the game stages & boss fights into short chapters	*F6: The « explanation » described in requirement NF5 should come in the form of a disclaimer pop-up message displayed in the game engine before the difficulty selection.*	F7: The new mode could reset the stage's music to the music progress at the start of the current chapter upon death (since some stage enemy waves are timed with the music)	
	F8: The new mode must be toggleable from the Extra stage selection menu	F9: The new mode should offer the option to increase the difficulty in between stages if a stage was beaten deathless.		
	F10: The choice to use the new mode must be given right before the difficulty selection menu.	F11: The « difficulty change offer » described in requirement F9 should be displayed using the same dialogue box renderer as the pause menu		
	F12: The new mode must be compatible with the other functions of the game, like the replay mode			
	F13: The new mode must make it so that a death sends you back to the start of the chapter instead of decrementing the life count by one			

Non-functionnal Requirements	*NF1: The patch must be easy to install by coming in the form of an executable that will modify the user's game executable.*	**NF2:The new mode should still offer a challenging but less punishing experience.**		**NF3: The patch will not come in the form of a dll file**
	NF4: The patch must be developed using the reverse-engineered source code of the original game's executable	**NF5: The patch should actively give a clear explanation that it offers an optional, more accessible experience, and not the intended one.**		**NF6: The patch will not have to be supported by THCRAP, the most used Touhou translation patching software**

Requirements evaluation

The requirements written in bold font were fully met. The ones in italic were not met, but their role was fulfilled in some other way. The ones in a normal font were not met.

F6, F8, F10, F13 and NF1 are all the italic requirements.

F6: The software does give a clear explanation that it offers an accessible experience, not the intended one. However, that disclaimer is not rendered using the in-game engine. Instead, it is a Windows pop-up.

F8 and F10: the new mode is not selectable within the game, as modifying the game itself proved to be too difficult. Instead, the new mode is automatically turned on whenever EoSDPointdevice is ran, before the start of the game.

F13: the new mode does even more than simply send you back to the start of the stage – it sends you back to the start of the "chapter", as per described in F5. In that sense, F13 was beyond met.

NF1: The patch does come in the form of an executable, and it easy to use, but it does not modify the user's game executable. It has to be executed each time a new game is started.

F7 was not met, but that is actually a good thing –enemy waves are actually only timed with music in Touhou 15. That is not the case with Touhou 6: mid-stage boss fights do not interrupt the music, and do not take a specific amount of time, which means the music could have been desynchronized from the enemies during the second half of the stage if the player took longer or shorter than intended to beat the mid-stage boss. Besides, keeping the flow of the music instead of interrupting it constantly every death might make for a better experience with the short Touhou 6 levels (this was tolerable in Touhou 15, which had much longer levels)

Self-evaluation

I am quite happy with the results of this project. Low-level memory management and interaction are such entropic parts of a computing environments that I did not believe creating savestates of a singular process would be possible. To me, it only could have worked with a memory dump of the entire RAM. But I had lost faith in the other solutions, so I tried, and it worked. This part truly is the technical feat of the project in my opinion: it essentially made me realise that a savestate manager software could exist for potentially many, if not every, singleplayer offline PC game.

Conclusion & Future work

I think the project, though light in amount of produced content due to the high quantity of trial and error, was quite successful. Not only does the software make the game more accessible, but it also keeps the game experience rewarding. A player that managed to beat the game on Pointdevice mode proved that they were able to beat the game perfectly! The life system was implemented because the game would be too punishing if merely one mistake made you lose everything. Pointdevice mode does not allow the player to make any mistake, but simultaneously gives them an occasion to try again immediately. It both makes the experience more accessible and arguably more rewarding.

There are many directions in which the project could expand. First, it could be completed with a proper GUI. This would make distribution possible, and this is the most obvious future work imaginable.

Then, further work could be done on the reverse engineering of the game. With Cheat Engine and Ghidra, I identified many variables and addresses -not enough to understand the entire game, but enough to get a fair start. With more experience on Ghidra, more understanding on how exactly it decompiles things like structures, a lot of progress could be made.

However, the very idea of creating savestates of single processes is very interesting to me on its own, even outside the context of gaming. With little modifications, I believe the software could list all available processes and save the state of whichever process the user wants.

There are few things I would change about the project if I were to do it again. With the power of retrospection, it would be easy to say that, were I to restart the project, I would have started with the functional idea from the get-go, but that is not the point. I do believe that I had a good reasoning, and I went from the ideas that sounded the most to the least feasible based on my previous computer science knowledge, and based on what I learned during my literature review.

References

Literature review references

[1] ALEXANDER, J.T. Et Al., 2012. An investigation of the effects of game difficulty on player enjoyment. Elsevier. [online]. Available from: https://www.sciencedirect.com/science/article/abs/pii/S1875952112000134 [Accessed 10 October 2021].

[2] あにら., 2009. [Touhou] Bad Apple!! PV [Shadow]. [online video]. 26 October. Available from: https://www.nicovideo.jp/watch/sm8628149 [Accessed 19/11/2021].

[3] BROWN, M., 2018. What Makes Celeste's Assist Mode Special. [online video]. 21 February. Available from: https://www.youtube.com/watch?v=NInNVEHj_G4 [Accessed 1 October 2021].

[4] BULAZEL, A. BLACKTHORNE, J., 2019. Three Heads Are Better Than One: Mastering NSA's Ghidra Reverse Engineering Tool . [online lecture]. INFILTRATE 2019. RPISEC. 14 May 2019. Available from: https://github.com/0xAlexei/INFILTRATE2019/blob/master/INFILTRATE%20Ghidra%20Slides.pdf [Accessed 5 October 2021].

[5] DAMEN, F., 2020. So I added a new difficulty to Minecraft.... [online video]. 13 June. Available from: https://www.youtube.com/watch?v=5C2wdLLTbTI [Accessed 19/11/2021].

[6] DARK BYTE, 2000. Cheat Engine.

[7] DARK BYTE, 2018. When was Cheat Engine first released?. [online]. 25 September 2018. Cheat Engine Forum. Available from: https://www.cheatengine.org/forum/viewtopic.php?t=608620&sid=d8d58343632df820e9643727eabb3b70 [Accessed 19/11/2021].

[8] DARK BYTE, 2021. GitHub – cheat-engine/cheat-engine: Cheat Engine. A development environment focused on modding. [online]. Available from: https://github.com/cheat-engine/cheat-engine [Accessed 19/11/2021].

[9] DAYUS, O., 2019. Here's Why Dark Souls, Bloodborne, And Sekiro Don't Have Difficulty Options. Gamespot. [online]. Available from: https://www.gamespot.com/articles/heres-why-dark-souls-bloodborne-and-sekiro-dont-ha/1100-6459827/ [Accessed 19/11/2021].

[10] DOTGEARS, 2013. Flappy Bird. Hanoi: dotGears.

[11] EXTREMELY OK GAMES, 2018. Celeste. Vancouver: Extremely OK Games.

[12] FRICTIONAL GAMES AB, 2015. SOMA. Malmö: Frictional Games AB.

[13] IAMTEHCHIBEH., 2008. UN Owen Was Her- Full version. [online video]. 28 November. Available from: https://www.youtube.com/watch?v=8jJZA-O_B78 [Accessed 19/11/2021].

[14] KASIDID2., 2009. 【東方】Bad Apple!! ＰＶ【影絵】. [online video]. 27 October. Available from: https://www.youtube.com/watch?v=FtutLA63Cp8 [Accessed 19/11/2021].

[15] MACHINEGAMES, 2014. Wolfenstein: The New Order. Rockville: Bethesda Softworks.

[16] MICROSOFT, 2000, DirectX 8.0 Documentation. [software documentation]. 12 November 2000.

[17] MICROSOFT WIKIA, 2007. DirectX. [online]. Available from: https://microsoft.fandom.com/wiki/DirectX [Accessed 19/11/2021].

[18] MOBIUS DIGITAL, 2019. Outer Wilds. West Hollywood: Annapurna Interactive.

[19] NITORIUM, 2018. Touhou 6: Koumakyou - The Embodiment of Scarlet Devil. [online]. Place of publication unknown: publisher unknown. Available from: https://moriyashrine.org/files/file/13-touhou-6-koumakyou-the-embodiment-of-scarlet-devil/ [Accessed 19/11/2021].

[20] NSA, 2019. Ghidra. Fort Meade: NSA.

[21] ORLAND, K., 2017. “Safe Mode” lets you explore Soma’s horror without risk of death. ARSTechnica. [online]. Available from: https://arstechnica.com/gaming/2017/12/somas-new-safe-mode-takes-the-survival-out-of-survival-horror/ [Accessed 19/11/2021].

[22] SILICON GRAPHICS & KHRONOS GROUP, 1994. OpenGL. Sunnyvale: Silicon Graphics.

[23] STRATAKIS, M., 2019. How often do you experience Flow?. [online]. Available from: https://innobatics.com/flow/ [Accessed 19/11/2021].

[24] STUDIO MDHR, 2017. Cuphead. Oakville: Studio MDHR.

[25] TEAM SHANGAI ALICE, 1997. Touhou Reiiden~ The Highly Responsive to Prayers. Tokyo: Team Shangai Alice.

[26] TEAM SHANGAI ALICE, 2015. Touhou Kanjuden ~ Legacy of Lunatic Kingdom. Tokyo: Team Shangai Alice.

[27] THE GAME BAKERS, 2016. Furi. Montpellier: The Game Bakers.

[28] TOUHOU PATCH CENTER, 2021. Touhou Patch Center:Tutorial. [online]. Available from: https://www.thpatch.net/wiki/Touhou_Patch_Center:Tutorial [Accessed 19/11/2021].

[29] TOUHOU WIKI, 2012. Category:Arrangement CDs. [online]. Available from: https://en.touhouwiki.net/wiki/Category:Arrangement_CDs [Accessed 19/11/2021].

[30] TOUHOU WIKI, 2020. Embodiment of Scarlet Devil. [online]. Available from: https://en.touhouwiki.net/wiki/Embodiment_of_Scarlet_Devil#Concept [Accessed 19/11/2021].

[31] WHITEHEAD J, 2007. Game Genres: Shmups. [PowerPoint presentation]. Sante Cruz: Foundation of Interactive Game Design. Available from: https://portalspdf.com/4770030789 [Accessed 19/11/2021].

Other references

[1] NAMCO, 1981. Galaga.

[2] KNIGHT J, 2014, windows - How to get the starting/base address of a process in C++? [Online] Available from: https://stackoverflow.com/questions/11564148/how-to-get-the-starting-base-address-of-a-process-in-c

Appendices

Ethics Form

STUDENT PROJECT ETHICAL REVIEW (SPER) FORM

The aim of the University's *Research Ethics Policy* is to establish and promote good ethical practice in the conduct of academic research. The questionnaire is intended to enable researchers to undertake an initial self-assessment of ethical issues in their research. Ethical conduct is not primarily a matter of following fixed rules; it depends on researchers developing a considered, flexible and thoughtful practice.

The questionnaire aims to engage researchers discursively with the ethical dimensions of their work and potential ethical issues, and the main focus of any subsequent review is not to 'approve' or 'disapprove' of a project but to make sure that this process has taken place.

The *Research Ethics Policy* is available at
www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060

Student Name	Pierre Boeglen
Supervisor	Mark Snaith
Project Title	Reverse engineering the game Touhou 6: Koumakyou - The Embodiment of Scarlet Devil in order to patch in a checkpoint system
Course of Study	Honors project CM4105
School/Department	School of Computing

Part 1 : Descriptive Questions			
1	Does the research involve, or does information in the research relate to:	Yes	No
	(a) individual human subjects		X
	(b) groups (e.g. families, communities, crowds)		X
	(c) organisations		X
	(d) animals?		X
	Please provide further details:		
	-		
2	Will the research deal with information which is private or confidential?	Yes X	No
	Please provide further details:		
	-The source code for a video game will be exposed to me. However, it will never be made publicly available : only a patch will.		

Part 2: The Impact of the Research			
3	In the process of doing the research, is there any potential for harm to be done to, or costs to be imposed on	Yes	No
	(a) research participants?		X
	(b) research subjects?		X
	(c) you, as the researcher?		X
	(d) third parties?		X
	Please state what you believe are the implications of the research:		
	-		
4	When the research is complete, could negative consequences follow:	Yes	No
	(a) for research subjects		X
	(b) or elsewhere?		X
	Please state what you believe are the consequences of the research:		
	-		

Part 3: Ethical Procedures			
5	Does the research require informed consent or approval from:	Yes	No
	(a) research participants?		X
	(b) research subjects		X
	(c) external bodies	X	
	If you answered yes to any of the above, please explain your answer: **An ethical review proved to be necessary in order to validate the reverse engineering of the game**		
6	Are there reasons why research subjects may need safeguards or protection?	Yes	No
			X
	If you answered yes to the above, please state the reasons and indicate the measures to be		
7	Has PVG membership status been considered?		
	(a) PVG membership is not required.		X
	(b) PVG membership is required for working with children.		X
	(c) PVG membership is required for working with protected adults.		X
	(d) PVG membership is required for working with both children and protected		X
	If you answered yes to (b), (c) or (d) above, please give details:		

8	Are specified procedures or safeguards required for recording, management, or storage of data?	Yes	No
		X	
	If you answered yes to the above, please outline the likely undertakings: **The reverse engineered source code will be stored in a private OneDrive directory that shall only be shared with the relevant university staff. The patch itself, however, poses no issues and will simply be stored on a git repository.**		

Part 4: The Research Relationship			
9	Does the research require you to give or make undertakings to research participants or subjects about the use of data?	Yes	No
		X	
	If you answered yes to the above, please outline the likely undertakings: **At the end of the project, a form will be passed to the students (and staff) of RGU. If they wish, they will be able to evaluate the checkpoint patch and give their thoughts on it. The form will not be made available to the Touhou or the gaming online communities.**		
10	Is the research likely to be affected by the relationship with a sponsor, funder or employer?	Yes	No
			X
	If you answered yes to the above, please identify how the research may be affected:		

Part 5: Other Issues			
11	Are there any other ethical issues not covered by this form which you believe you should raise?	Yes	No
		X	
	The checkpoint patch needs to follow the fan-made content guidelines as defined by Team Shangai Alice (translation here)		

Statement by Student I believe that the information I have given in this form is correct, and that I have addressed the ethical issues as fully as possible at this stage.			
Signature	Pierre Boeglen	Date	06/10/2021

If any ethical issues arise during the course of the research, students should complete a further Student Project Ethical Review (SPER) form.

The *Research Ethics Policy* is available at
www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060

Part 6: To be completed by the supervisor			
12	Does the research have potentially negative implications for the University?	Yes X	No
	If you answered yes to the above, please explain your answer:		
	Possible legal/ethical implications with reverse-engineering an existing game. Will likely need advice from ethics committee as to whether this can proceed, and if so under what conditions.		
13	Are any potential conflicts of interest likely to arise in the course of the research?	Yes	No X
	If you answered yes to the above, please identify the potential conflicts:		
14	Are you satisfied that the student has engaged adequately with the ethical implications of the work? [In signifying agreement, supervisors are accepting part of the ethical responsibility for the project]	Yes X	No
	If you answered no to the above, please identify the potential issues:		
15	**Appraisal:** Please select one of the following		
	The research project should proceed in its present form – no further action is required		
	The research project requires ethical approval by the School Ethics Review Panel	X	
	The research project needs to be returned to the student for modification prior to further action		
	The research project requires ethical review by an external body. If this applies please give details		
	Title of External Body providing ethical review		
	Address of External Body		
	Anticipated date when External Body may consider project		

Affirmation by Supervisor			
I have read the student's responses and have discussed ethical issues arising with the student. I can confirm that, to the best of my understanding, the information presented by the student is correct and appropriate to allow an informed judgement on whether further ethical approval is required.			
Signature	M. Snaith	**Date**	7 October 2021

Poster

EoSDPointdevice

Project by Pierre Boeglen, supervised by Mark Snaith

Introduction

Touhou Koumakyou : Embodiment of Scarlet Devil (EoSD) is a shoot'em up video game released for Windows in 2002. It is the 6th installment of the Touhou franchise, and it is the first game of the series to be released on Windows – the previous ones were for the PC-98.

This increased visibility has made EoSD the game that made Touhou so popular, with its witty characters and its amazing soundtrack.

However, due to the high difficulty innate to its genre, only few people interested in Touhou are actually able to enjoy the games.

The vanilla version works with an arcade life system: if the player gets hit, they come back right where they died, but they lose a life. If all lives are lost, the player gets sent back to the very beginning of the game.

Therefore, I wanted to create a software that would give many fans-to-be a chance to have fun with the series' most popular game.

Project Aim

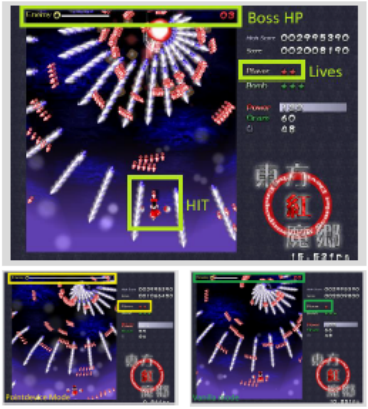
The objective was to replace the vanilla system with checkpoints, scattered throughout the levels and the boss fights.

Touhou 15 (TEAM SHANGAI ALICE, 2015) implemented such a system, "Pointdevice mode", which is what the project is named after.

Methods

A lot of trial and error went into this project. It all relied on low-level memory manipulation: therefore, Cheat Engine(DARKBYTE, 2000) was used a lot in order to start getting an idea of how the game was working behind the scenes. Once some key variables were located, Ghidra (NSA, 2019) was used to understand how these variables were modified – for example, how exactly a life was removed upon death. A strong understanding of the death mechanic was required to proceed with the project.

Figures and Results



The first idea was to create a code injection using Cheat Engine, by intercepting the instruction that removes a life and replacing it with a redirection towards custom code that would reload the previous enemy wave. This would require coding in Assembly.

Unfortunately, Ghidra revealed that the functions that load enemy waves were way too complex to be coded in Assembly, Much more time to reverse engineer the game, let alone to learn more Assembly, would have been needed. Therefore, the focus was shifted towards Savestates.

Savestates are a feature most often found in emulation: they create a copy of a program's state, a copy that can be reloaded in order to essentially go back in time to a previous point.

This strategy ended up working: using many low-level Windows C++ libraries, it was possible to read and write memory and thread contexts.

Conclusion

I am happy with the results of the project. I had lost faith more than a few times, I had tried a lot of ideas that did not work, but in the end, it was the simplest yet least intuitive one that worked. I would have never thought that reloading the state of an app, let alone a game, would have been feasible, given how entropic the low-level environments of computers are.

Now, I can only hope that this project will give some dubious fans the final push they wanted to finally get to enjoy the Touhou games.

Acknowledgments

I would like to thank Mr. Snaith, for guiding me throughout the project, and the Touhou community as well as the content creator Game Maker Toolkit (BROWN, M., 2018) for inspiring me to create accessibility for a beloved yet esoteric game.

My gratitude extends towards Darkbyte, the Cheat Engine creator, for making such a useful program open-source, as well as Wikileaks for leaking Ghidra, forcing the NSA to release it open-source.

Finally, I would like to thank ZUN for creating such a great game series essentially on his own, and for being so acceptant of fan projects. Such benevolence is really rare in Japan when it comes to copyrights.

References

NSA, 2019. Ghidra. Fort Meade: NSA.

DARK BYTE, 2000. Cheat Engine.

BROWN, M., 2018. What Makes Celeste's Assist Mode Special. [online video]. 21 February. Available from: https://www.youtube.com/watch?v=NInNVEHj_G4 [Accessed 1 October 2021].

TEAM SHANGAI ALICE, 2015. Touhou Kanjuden ~ Legacy of Lunatic Kingdom. Tokyo: Team Shangai Alice.

ROBERT GORDON UNIVERSITY ABERDEEN
SCHOOL OF COMPUTING

Honors Project

Project log

24/09 : EoSD patch draft ethics form & project proposal

01/10 : review of the drafts

07/10 : sent the updates drafts

11/10 : searched litterature

21/10 : drafted litt review

09/11 : commitee approved

19/11 : submitted litt. review, sent updated ethics form & proposal

26/11 : drafted requirements

03/12 : worked on requirements, looked up stuff on CE & GH

10/12 : worked on requirements

15/12 : submitted requirements

--BREAK--

24/01 : started work on cheat engine, identifying as many vars as possible

31/01 : started work on Ghidra, learned the workings of the program

07/02 : started research on PoC, kept progressing on ghidra

10/02 : identified PoC goal - simply showcasing a checkpoint

15/02 : found a way to create a checkpoint, only works on boss battles though

20/02 : submitted that as PoC

28/02 : realised the ghidra methos was unrealistic, started researching savestates

09/03 : decided to go for the winthread method : memory dump reload

15/03 : memory dump reload working, onto thread context reset

21/03 : fixed c++ issue with thread context, drafted final report

28/03 : created git repo

04/04 : started work on poster

11/04 : finished poster, progress on final report

18/04 : reworked poster

27/04 : presentation

28/04 : finished report

Detailed Project Proposal

First Name:	Pierre
Last Name:	Boeglen
Student Number:	2006259
Supervisor:	Mark Snaith

Defining your Project

1.1 Project title

Help: *a brief statement about what you are actually going to do.*

Reverse engineering the game Touhou 6: Koumakyou - The Embodiment of Scarlet Devil in order to patch in a checkpoint system

1.2 Background

Help: *Provide the background to your project. This section should highlight the main topics in the area you are going to research. Essentially what is the project about, what has been done before and why is this project important? ~500 words*

1. Introduction to the Touhou Project

The Touhou Project is a series of independent "bullet hell" shoot'em up games (or STGs), made by Team Shanghai Alice, which is composed of only one member, "ZUN". He is the one behind everything about these games: the code, the art, the story, and last but not least, the music. "Bullet hell" refers to STGs where a very high number of bullets are on screen, they rely on the player's ability to find patterns to safely sneak through, not on pure reaction speed.

The first Touhou game, "The Highly Responsive to Prayers", was released in 1997 for the Japanese PC-9800. The four next games were also released on that system, but Touhou 6 (or "EoSD") brought the series to Windows in 2002. While the 5 first instalments only sold a few hundred physical copies at most during conventions, it is Touhou 6 that gave the series the push towards its huge popularity: it is still active nowadays, as we can see with Touhou 18 came out in May 2021. Not only are there 18 main titles; there are also 12 spin-off games.

The games typically follow an arcade style: if you get hit, you do not lose any progress in the stage, but you lose a life. If you lose all lives, you lose the whole game and need to restart (or use a continue, locking you out of the "good ending"). The games all last 6 stages, and each stage lasts a few minutes.

2. Embodiment of Scarlet Devil

Let us go into more details about Touhou 6. It was a "clean slate" for the series, both in terms of story and of technology. Compared to the first 5 games, the graphics look much more detailed, the music uses a MIDI format rather than chiptunes, the bullet patterns are much more intricate and interesting. It also introduced many of the series' most popular characters, making it the most famous Touhou game by far.

3. The community

It's impossible to talk about Touhou without mentioning the huge fan community. The game series has spawned a massive amount of fan content: art, memes, music remixes, manga, even fully fledged fan games or animated series. One other form of fan content is patches – the addition of small features to the game. For example, a Focus Hitbox Patch. EoSD does not display your hitbox. In all the following games, "focusing" (making your character move slower to help you get through complex patterns) also highlights the exact portion of your characters that can actually get hit, your "hitbox". The hitbox is much smaller than the character sprite, so it is useful to be able to see it separately. EoSD does not have that feature, so people made a patch to implement it.

4. My project

My goal is to create a patch for EoSD to implement the checkpoint system from Touhou 15. That game offers an alternative play mode, "Pointdevice mode" where each stage is split in chapters. There is no life system: if you get hit, you get sent back to the beginning of the chapter with no penalty. This makes the game much more accessible to people who do not wish to throw themselves at the game many times in arcade mode in order to train later stages – it makes the game into a "die and retry" situation where experimenting is encouraged, and where the result will be a perfect no-death completion.

Note: throughout this proposal, I will refer to "Touhou 6" and "EoSD" interchangeably. The same applies for "checkpoint system" and "Pointdevice mode".

1.3 Motivation

Help: *To whom is this project important? A project must address a question/problem that generates a small piece of new knowledge/solution. This new knowledge/solution must be important to a named group or to a specific client (such as a company, an academic audience, policy makers, people with disabilities) to make it worthwhile carrying out. This is the* ***motivation*** *for your project. In this section you should address who will benefit from your findings and how they will benefit. ~300 words*

Example 1: If you intend to demonstrate that a mobile application that automates class registers at RGU will be more efficient than paper-based registers - the group who would be interested in knowing/applying these findings would be both academic and administrative staff at RGU and they would benefit by time saved and a reduction in their administrative workload.

Example 2: You are demonstrating that a particular 3D model design increases realism in 3D environments. The group that would be interested would be games designers or developers of 3D virtual environment applications. The would benefit from producing more realistic environments that could increase sales of their products.

Example 3: You have designed a new network topology for IrishOil plc's new Aberdeen headquarters. The interested group would clearly be IrishOil. They would benefit from easier maintenance and improved security of their computer network.

I can picture this project having one main benefits:

The patch will offer a new, more accessible way to play a very famous franchise's most popular game. This means that players all over the world will benefit from the project – from those who do not have time to spend learning the game by heart, to those who do not have the physical capacities or the attention span to intensely focus and make precise movement for 30+ minutes. This does include, but is not limited to, people with disabilities.

Accessibility is especially an issue with the Touhou franchise because of its huge online popularity. The size and dedication of the fandom cannot be overstated: for example, the Touhou wiki records over 6,200 fan music albums. Entire albums, not single tracks. Most of these albums were even commercially available to a certain degree, because the copyright around the Touhou franchise is quite lenient. The point is, the fandom is huge, but the games are known for being very tough and punishing. This means that thousands, maybe millions of fans are missing out on their favorite franchise's source material, because its high difficulty makes it inaccessible. The patch would let most of these people discover the Touhou experience.

1.4 Aim & Objectives

Help: *Outline what are the main things your project is going to do and what steps or milestones will be used to achieve this aim. The Aim is unlikely to change throughout your project; however, the objectives are likely to adapt to your ongoing research and development. In particular it is highly likely that you may wish to split objectives into sub-objectives as work progresses. A good clear set of objectives give you something to evaluate your final project against.*

Example : For the timetable app outlined above
Aim: To create a functioning attendance application that efficiently automates the taking of class registers.
Objective 1: study existing register system in place at RGU and identify weaknesses
Objective 2: research existing automation technology's and identify and evaluate those that may be appropriate to taking in class registers
Objective 3: Implement chosen technologies to create prototype application
Objective 4: Conduct user trials to evaluate capabilities of prototype application
Objective 5: Create a refined application incorporating feedback from user trials

Aim: Reverse engineer the game Touhou 6 in order to obtain a codebase on top of which a Pointdevice mode patch will be implemented

Objective 1: Learn more about the game on a technical level

Objective 2: Learn about the reverse engineering landscape

Objective 3: Obtain a codebase for the game

Objective 4: Learn about EoSD patches

Objective 5: Create the patch

Objective 6: Find out if players find the concept useful, well executed

1.5 Key Techniques

Help: *Perform some initial research into the area and outline what techniques you may research in further detail here. The techniques you cover here should include references to the papers where you have sourced the information. The techniques mentioned here are very likely to become the section headers in your literature review.*

Technical details about EoSD:

- Uses the DirectX API to interact with PC components (https://www.windowscentral.com/what-directx-why-does-matter-gaming)
- Runs on a custom engine that ended up being used for the next 3 games as well (https://en.touhouwiki.net/wiki/Embodiment_of_Scarlet_Devil#Concept)

Reverse engineering software:

- Ghidra can be used for disassembly and decompilation (https://github.com/0xAlexei/INFILTRATE2019/blob/master/INFILTRATE%20Ghidra%20Slides.pdf)
- Cheat engine is used to dissect a process' memory in real time, it allows one to study addresses as well as assembly instructions and function calls (https://www.cheatengine.org/aboutce.php)

Patching Touhou games:

- THCRAP, the "Touhou Community Reliant Automatic Patcher" is mostly used to apply translation patches, but also other types of patches. This means that this is an option worth researching into, but that there will not be much documentation outside of translation. (https://gist.github.com/WindowDump/e007516524b7488eccf74a020b3c7977, https://www.thpatch.net/wiki/Touhou_Patch_Center:Download)

1.6 Legal, Social, Ethical, Professional and Security issues

Help: *Here you should discuss any legal, social, profession and security issues that you believe may occur during the course of your project. It is not acceptable to write none in this box, all projects, regardless of focus will have to address issues in one, or more, of these categories. This is an extremely important part of your honours project to which there is no correct answer, this section must be fully discussed with your Honours Supervisor.*

Example 1 : In the class register example above – there would be a Legal and Security issue with the gathering and storage of student data. There may be a social constraint as you may be relying on a user to have access to a specific technology. There will need to be consideration of user accessibility.

Example 2 : A 3D model design may have ethical considerations in its evaluation. What if your model made users feel nauseous? Social constrains may again be access to technology or accessibility issues.

Example 3 : You network design need to adhere to specific company policies. You would need to consider the possibility that your design could be wrong, compromising the company's security.

Touhou is a special case in the context of independent games copyright. Despite being the full copyrights owner, despite not wanting large companies to have a role in his games, ZUN is still very supportive of fan work. For example, he refuses to ship his games with official translations, out of fear that the company would "set in stone" the text, but has no issues with fan translations.

https://touhou-project.news/guideline/

This page gives a detailed guideline that fan content creators must follow. This is an extract of the translation (found here, and double-checked with DeepL):

Prohibited Activities

* Producing material that damages the Touhou Project image or brand.

* Producing material that infringes the rights of others.

* Producing material that misleads others into thinking it is official Touhou Project material.

* Producing material using game assets other than screenshots and gameplay videos.

* Posting content that includes material from any of the endings.

* Producing material that involves transmitting personal thoughts using original content and/or derivative works.

* Reusing material from another derivative work without the permission of the copyright holder.

* Other acts that are judged to be significantly maladapted according to social conventions, such as excessive sexual expression or content that discriminates specific individuals, race, etc.

* Processing ZUN's photo without permission

As a patch is a form of fan-created content, it needs to follow these rules as well. This is the verification that the patch indeed follows the rules:

The patch would not damage the brand, it would not in fringe on anyone's rights, it would not use game assets or spoil any of the endings. It would not involve transmitting my personal thoughts, reuse other fan made content, contain explicit/offensive content or use ZUN's photos. However, it could potentially appear to be official material, so steps will be taken to assure that no confusion

is possible – for example with a message on the title screen explaining that the Pointdevice mode is not official and is added by a fan.

With all these rules respected, I believe there should be no issues with the patch.

The reverse engineering part is where the problem lies. There are two factors that make reverse engineering the game acceptable:

- The lack of End user license agreement in the Touhou games
- The fact that the game was obtained through legal means

However, the rules listed above make it clear that distributing, as the game assets would clearly need to be used.

This means that there will be no issues distributing the patch itself, the end goal of the reverse engineering, but that I cannot make the source code public.

Therefore, the source code will be stored on a private OneDrive directory that will only be shared to the project supervisor and markers. The patch itself will be stored on a git repository.

The testing of the patch shall only be carried out within the RGU staff and student community.

1.7 Project Plan

Help: *This is the project plan as to how you will go about achieving the objectives of the project.*

Example: In the class register example above the research plan may involve:
Collecting and analysing paper-based registers in a given class on five occasions.
Identifying the error rate average on these occasions
Researching existing automation techniques
Designing and implementing a mobile application that automatically records attendance in class.
Deploying the application in the class on five occasions.
Identifying the error rate average of the mobile application on these occasions.
Comparison of data and summary of findings.

- Research technical details about EoSD (DirectX API, custom engine)
- Research what software is used for reverse engineering (Ghidra, Cheat engine)
- Reverse engineer the game for the codebase
- Research the technologies used for patching executables (THCRAP, *.dll files)
- Create the patch
- Set up forms for the RGU students (and staff) to give feedback about the idea and its execution

1.8 Ethics Form

You must include in your signed ethics form in this submission or you will not be able to continue the project.

Source code
https://github.com/2006259/eosd-pointdevice