The Fine Line Between Skill and Cheating in Video Games

Wizardry and Steamworks

October 8, 2022

Abstract

There is an increasing interest in video games relying heavily on the psychology of players with regards to monetization as well as pitching games to be accepted as a form of artistic expression. As follows, a need has arisen to implement security measures in order to prevent game scams and there are even discussions of political fibre concerning the overspending of young video game players. In the midsts of the video game discussion, the authors approach video games from a tributary angle and with a desire to draw attention to a frequently omitted fine line distinction between player skill and cheating. It has been found that several methods of circumventing game difficulty is common to multiple games of the same genre that have been left undocumented. Some of the methods of circumventing game difficulty are in a direct relation with more fundamental concepts of computer security such as command injections, Time of Check To Time of Use or attacks based on race conditions. The paper aims to document various methods that players employ when playing while highlighting the fundamental principles that the exploits hinge on as well as providing real-case examples for each category.

1 Introduction

From the inception of video games into mainstream media, a parallel movement was started to reduce game difficulty in order to allow video game players to succeed without expending the effort that would be required to win the

game through skill. Whether reducing game difficulty meant cracker groups inserting "trainers" into pirated games that granted video game players extra benefits or whether players discovered with minimal knowledge various programming bugs and their consequences that provided an advantage in a video game, there has always been a fine line that would, as the authors would like to claim, frequently and unjustly distinguish between "video game player skill" and "video game cheating".

Game companies acting under the constraint of profits, particularly with regards to games as a service, interpret "cheating" loosely and end up implementing security measures that tend to raise false positives. In some cases the mitigation tools have been said to infringe on the fundamental rights of players (Sharp, 2006). To add to the smelting pot of problems, games have been known to become over the years an echelon for job recruitment, particularly given the interest of job recruiters in relation to the skill required to defeat or win the game (Nieborg, 2004; Melis, Hare, & Tomasello, 2006). Game companies frequently use blanket bans to prohibit or penalize a player and tend to coalesce skill or talent with tampering with the game code. This article will focus on "game techniques" with the limitation that the software and the hardware that the game runs on is left untampered with (Kolasinski, Blake, Anthony, Price, & Smith, 1979). This article stands to highlight the distinctions between "skill" and "cheating" as particular to the mechanics of the game itself that have been used to gain an advantage.

The techniques used to gain advantages in games are applicable to an entire game genre and are closely related to fundamental principles of computer security. It has been identified that singularity events are to be discovered in some games where, due to the architectural disposition of objects, Non-playable Characters (NPCs) as well as other game assets, the outcome of the game is predictable given some previous knowledge of the game.

2 Outline of the Paper

With the hopes of the Introduction 1 having laid out the goals of the paper, the authors would like to offer an outline of the paper (Chapter 2) and then proceed and provide examples of a universal game cheats or techniques that are ubiquitous to all games in the section "Ubiquitous Game Cheats" (Chapter 3). Some subchapters of Chapter 3, namely "Ubiquitous Game Cheats" target a more technically inclined audience but practical examples

are provided in order to make the article palatable to a wider audience. The "Conclusions" chapter 4 sums up the findings, discusses the findings and provides avenues to be explored.

3 Ubiquitous Game Cheats

Games can be distinguished as "online games" where certain parts of the game are backed by a server on the Internet, perhaps as an evolution of Multi-User Dungeon (MUD) games (Heron, 2013) on Buletin-Board Systems (BBS), and "single player games" where the entire game, even if benefiting from a LAN multiplayer play mode, is entirely in the possession of the customer. Nevertheless, the authors would like to argue that most of the methodologies presented in the following subchapters pertain to the game genres or are even more fundamental such that the methodologies can be applied to a broad range of games regardless whether that means online, multiplayer or single player games.

3.1 Memory Manipulation

Borrowing from software security and with particular mentions to memory injections (Zalewski, 2001), time-of-check-to-time-of-use (TOCTTOU) (Jinpeng & Calton, 2005; Merrill, 2003) exploits as derivations of the "confused deputy" problem (Hardy, 1988) as a consequence of race conditions, memory manipulation applies to both a client-server game model and the single player game model. The unifying technique lies within the fact that various game parameters such as the in-game state of the player will be stored in memory. By identifying the exact memory location where a variable lies, the value corresponding to the variable can be altered thereby granting the player an advantage. The technique relies on scanning the entire memory allocated to a game, identifying a value corresponding to a memory address to be changed, changing the value by playing the game and then repeating as many times as necessary till the list of memory addresses containing the sought value cannot be further reduced. After the memory addresses can no longer be reduced, the value at the various found memory locations is changed to be favourable to the player.

The same technique can apply to "server-backed games" or "online games" when certain variables are not checked for consistency by the server. For in-

Figure 1: Scanning memory in "The Sims Mobile" by "Electronic Arts" in order to alter the value of the energy bar and grant the player a limitless amount of energy.

stance, "The Sims Mobile" by "Electronic Arts" (Figure 1) on Android has been found to track in-game currency but leaves the tracking of in-game "energy" up to the client such that an attacker can control their own energy and the server will blindly accept the change. In many cases, tracking all the variables of players in an online game is unfeasible due to the network bandwidth requirements such that manipulating some variables on the client might be possible.

When the attack can be performed in a client-server online game, the attack pattern can be categorized under the TOCTTOU category of exploits where the game client will alter a variable and the server will trust the game client when the variable is checked. Conversely, whenever the attack fails, the server thwarts the TOCTTOU attack by maintaining a copy of the variable effectively performing in general computer security terms a transaction (Seltzer, 1992). On the outlines, the actual attack performed is a memory injection where a buffer is updated to contain a different value or different values. Furthermore a buffer or integer overflow could trigger a

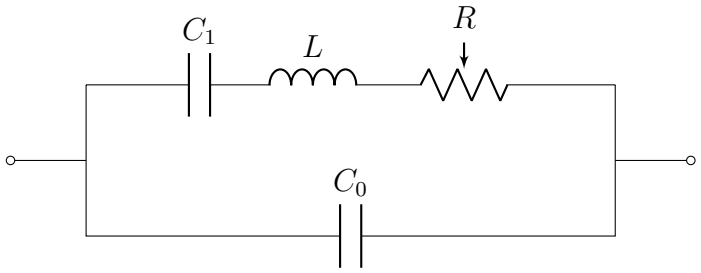

Figure 2: An RLC circuit equivalent of a crystal oscillator that can be used to vary the frequency in order to slow or speed up games relying on the oscillator as a clock source.

confused deputy event where a player could, for example, become immortal due to a variable overflowing and being interpreted as infinity.

3.2 Time Manipulation

Whether a platform benefits from a real-time clock (RTC) or time is backed by CPU ticks, most games require a notion of time such that time itself becomes an attack vector in video games. Older platforms, back and before the "Commodore Amiga" or the "Atari" relied heavily on the CPU to keep track of time by timing the distance between CPU operations. A common exploit was to use a software called a CPU jammer - effectively some software that incurred a lot of CPU cycles such that the CPU had trouble maintaining a consistent distance between CPU ticks and effectively slowing down time thereby allowing players more time to react to challenges. Games produced in the past, even when the x86 architecture was gaining grounds are only playable today by using a slowdown mechanism or the game will play at the CPU clocking speed which is faster than the game was intended to be played.

The GameBoy DMG by "Nintendo" handheld console relied on a crystal oscillator that was built into the PCB that ticked at $\approx 4.194304MHz$ and would provide a means to measure time. The crystal oscillator could be physically manipulated such that the game could be sped up or slowed down effectively providing time manipulation as a feature of a game.

In Figure 2 an circuit is presented that is the RLC functional equivalent of a piezoelectric quartz crystal with the added advantage that the value of the potentiometer R can be modified at will by the player. In electronic

terms, the equations governing an RLC circuit are as follows:

$$f = \frac{1}{2\pi * \sqrt{L * C}}$$

for the frequency f and:

$$Q = \frac{1}{R}\sqrt{\frac{L}{C}}$$

for the quality factor of the oscillator. Since variable potentiometers are the cheapest components to obtain, it is desirable to combine the formula in order to express the frequency f to depend on the resistance R that can be varied whilst other parameters will remain fixed. From the Q-factor, we have that:

$$\begin{aligned} Q &= \frac{1}{R}\sqrt{\frac{L}{C}} \\ Q^2 &= \frac{1}{R^2} * \frac{L}{C} \\ \frac{L}{C} &= Q^2 * R^2 \\ \frac{L * C}{C} &= C * Q^2 * R^2 \\ L * C &= C^2 * Q^2 * R^2 \\ \sqrt{L * C} &= \sqrt{C^2 * Q^2 * R^2} \end{aligned}$$

substituting in the formula for the frequency:

$$f = \frac{1}{2\pi * \sqrt{C^2 * Q^2 * R^2}}$$

that can then be further reduced to:

$$f = \frac{1}{2\pi * C * Q * R}$$

and then solved for R:

$$\begin{aligned} \frac{1}{f} &= 2\pi * C * Q * R \\ R &= \frac{1}{2\pi * f * C * Q} \end{aligned}$$

Since small components are usually preferred, particularly in tight enclosures such as the GameBoy DMG, small values are picked for C and since stable oscillators are desired high values for Q are typically chosen. Let $C = 1pF$, let $Q = 20000$ and. the typical GameBoy DMG clock frequency of $4.194304MHz$ such that:

$$\begin{aligned} R &\approx \frac{1}{2 * \pi * 4.19Hz * 1e^{-12}F * 2e^{4}} \\ &\approx 1.8973\Omega \end{aligned}$$

Meaning that the variable resistor would have to have its lock point at 1.8973Ω in order to emulate the GameBoy DMG clock frequency of $\approx 4.194304MHz$ when all games will play at their design speed. Looking back at the derived formula for the clock frequency f relative to the resistence R:

$$f(R) = \frac{1}{2\pi * C * Q * R}$$

It becomes clear that:

$$\lim_{R \to \infty} f(R) \Rightarrow f \mapsto 0$$

such that for very large resistor values the games will be slowed down.

Conversely,

$$\lim_{R \to 0} f(R) \Rightarrow f \mapsto \infty$$

signifying that very small resistor values will speed the game up.

Whether intentional or unintentional the distinction between cheating and skill can be made by differentiating between games that allow time manipulation as a game control or the intent of the player to hack the game in order to gain an advantage in the games.

3.3 Saving and Reloading

In this section a distinction can be made between save game manipulation; that is, the player using tools to alter save games and a player just saving the

game. The former case is an actual game hack and can possibly refer back to "trainers" released by crack groups that allowed variables to be manipulated within save game files. This was mainly performed by analyzing save game file formats and then locating how the variables are disposed.

Nevertheless, the latter "hack" is a "skill" based hack based on the player's observation that restoring a game nullifies a series of events. In technical terms, once a game is loaded, the whole transition of states between the state where the game has been saved and up to the point of reloading is completely cancelled. The "exploit" hinges on the fact that an entire continuation is eliminated and then the game is rewound to a previous state. Borrowing from Hoare logic (Hoare, 1969), only `emp` could follow from a death or an irreversible event such that reloading the game is the only option available to a player. One example illustrated in Figure 3 where saving and reloading games can be considered a cheat would be in games where mini-games are embedded in a larger game and are required to progress the game itself; whether that would be gambling to gain currency, playing mini-games to open doors, etc.

```
procedure PLAY(minigame)
end procedure
save ← game state
repeat
    result ← PLAY(minigame)
    if result ⇒ loss then
        game state ← save
    else
        save ← game state
    end if
until game state is desirable
```

Figure 3: The simplest algorithm to win any mini-game in any single-player game just by saving and reloading the game.

The algorithm can be repeated indefinitely and guarantees that no losses will be incurred. In particular, the algorithm is perhaps the only solution to guarantee a win in situations where a game can lead to locking up in case the player fails to win a task at hand. Some classical examples include games where players must lock-pick locks or else the lock gets permanently

jammed ("Thief" developed by "Eidos-Montreal", "Deus Ex" developed by "Eidos", etc...), the player runs out of money and cannot hedge a bet to play blackjack ("Leisure Suit Larry", the blackjack mini-game where the player needs to accumulate money to organize a marriage to be seen in Figure 4) and the game leads to a point of no return.

Figure 4: The game of Blackjack to be found in "Leisure Suit Larry in the Land of the Lounge Lizzards" developed by "Sierra On-Line" can be won without failure by following the algorithm illustrated in Figure 3.

3.4 Ethical Guidelines

Computer games with narratively rich backgrounds are created by people from various backgrounds and have varied notions of morality that share similar ethical reasoning patterns. By consequence, the game design and narrative patterns generated by game developers tend to ease the progression of a video game player whenever the player takes decisions that are following the same patterns that the developers based the narrative of the game upon. Even the most libertine of games, starting from old school pen and paper "Dungeons and Dragons" cannot allow a completely free expression or the

game could be made to end or deadlock very early in the narrative due to players performing acts that destroy the game itself. Such instances could include attacking or killing NPCs that are crucial to the game progression, not playing the game at all, or any other actions that tear at the fabric of the game rather than following the path established by the narrative.

A game player is more than often expected to follow an ethical path where certain acts are favourable to others and, although the former sounds general, there is a wide variety of games where a player is rewarded by performing noble deeds rather than being random and chaotic with the choices within the narrative of the game. More than often, once a player strays from the general ethical framework of the game, the game forcibly adjusts the path of the player by either increasing the difficulty or penalizing the player.

3.5 Macros

There is no unifying policy on using macros in games and end-user license agreements (EULA) vary between games and game companies. With the eldest form of macro being the "autofire" feature implemented by most hardware game controllers to whole scripts written in AutoIt (Cano, 2016), there is a very broad gradient between what could be considered cheating or just a quality of life change. For most "online" games where other competing players would be affected, there usually exists an unifying EULA clause stating that a game provider may cease service to any user at its own discretion.

Concepts such as "Clickjacking" follow through as a separate attack vector using various software that allow players to perform repetitive clicks in order to gain an advantage. The debate whether macros are to be considered cheating or not gets even more complex when it is to be considered that hardware developers have integrated the creation of complex macros as a feature available within the hardware itself (Tan & Ng, 2014).

An extreme example of macros are aim bots for First-Person Shooter (FPS) games, as portrayed in Figure 5, where some software is designed to perform a series of Optimal Character Recognition (OCR) and key injection operations in order allow a player to always aim perfectly at a target. Since the game is not being tampered with, the methods of detecting players that use aiming bots relies heavily on statistical analysis and can only confer a probability threshold based on which the game administrators decide whether the player is to be marked as a cheater (Yu, Hammerla, Yan, & Andras, 2012). Fairness is discussed in loose terms of a player having an unfair advantage

Figure 5: AimBot used in First-Person Shooter games where players use some external software that rotates the camera to always be pointing precisely at enemies.

over other players but good knowledge and practice of a game can similarly be considered an advantage.

3.6 Exploiting Architectural Invariants

The French philosopher Gilles Deleuze (Deleuze, 2014) mentions a road roundabout explaining that given its nature, a road roundabout has the effect of controlling drivers by restricting their moment patterns to a circular motion in the roundabout. Deleuze names the consequence of architecture "Societies of Control" with references given to the French philosopher Michel Foucault. Albeit the subject being complex, when the concept of architecture restricting movement is reduced to the layout of a game scene, a lot of the free variables are reduced to a smaller set of irreducible variables depending on the game design. Going back to one of the earliest commercially successful games, namely "Pong" (a simplification of table tennis) on the Atari 2600, released as part of the "Video Game Olympics" collection, the game can be won indefinitely and without failure simply by positioning the bar at the right position on the screen. The ball is cast from the middle of

the field and has the exact same initial start vector up to the point that the game is either always won by the player or the game is a draw ending up in an endless exchange of the ball. The problem of movement restrictions in "Pong" is reduced enough to be explained with some Eucledian geometry and basic Newtonian physics.

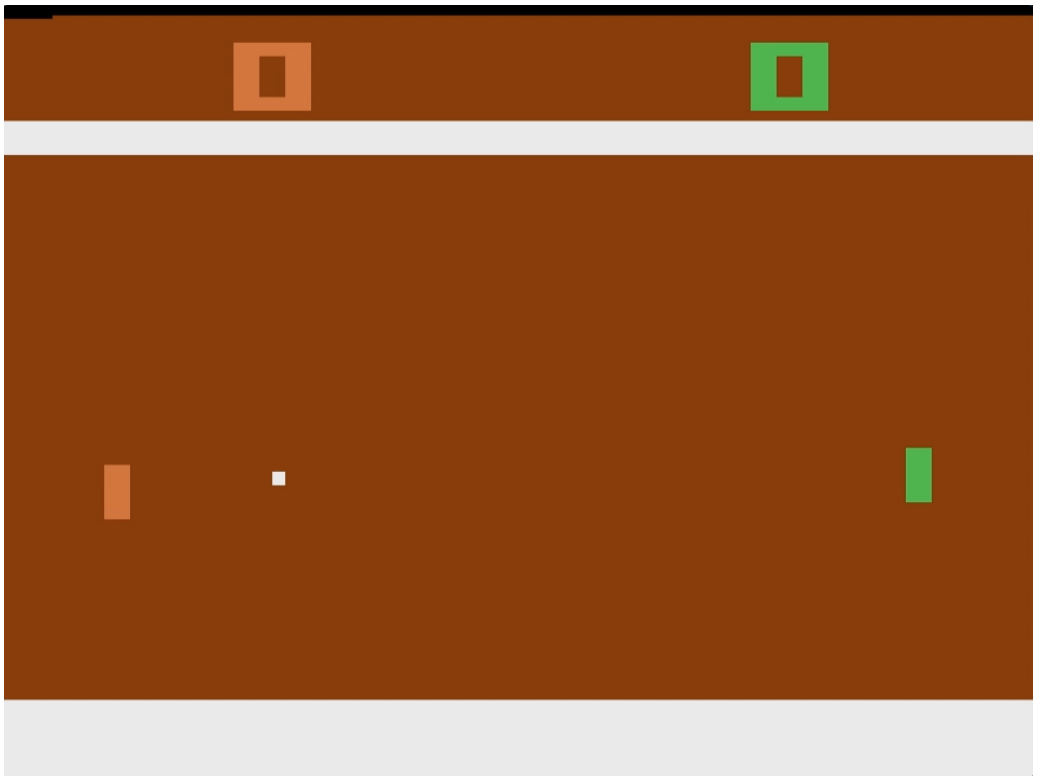

Figure 6: Atari 2600 "Video Game Olympics - Pong" game avoiding the trivial case stalemate by deliberately shifting the position of the computer controlled paddle (on the left) lower than half of the width of the paddle.

As the Atari 2600 game "Video Game Olympics" stands there are several simple rules that govern the gameplay: the ball speed remains constant throughout the game, the ball bounces off the upper and lower boundaries according to Newtonian mechanics (ie: the angle of incidence is equal to the angle of reflection) and the ball bounces off the paddles as a function of the point where the ball hits the paddle in relation to the closest edge of the paddle; in case the ball hits the very centre of the paddle, the ball is sent

back at 90° relative to the paddle. At the very least, these rules allow the player to deadlock the game by matching the angles such that the exchange between the player and the computer controlled paddle never ends.

Assuming the rules of the game, the angle of reflection on the player paddle θ_p and the angle of reflection on the computer controlled paddle θ_c are:

$$\begin{aligned} \theta_p &= \frac{\frac{d_p}{l_p} - \frac{1}{2}}{2 * \pi} \\ \theta_c &= \frac{\frac{d_c}{l_c} - \frac{1}{2}}{2 * \pi} \end{aligned}$$

where d is the distance to the closest edge of the paddle and l is the length of the paddle. Both paddles are of equal length such that $l_p = l_c$. It follows from the system of equations that $\theta_p = \theta_c$ when $d_p = d_c$. In other words, if and only if both angles of reflection are equal then the point of impact must have occurred at the same distance from the edge for any given paddle. However, θ_p can be equal to θ_c since the upper and lower edges follow the law of reflection. If and only if $\theta_p = \theta_c$ then it follows that $d_p = d_c$ such that a stalemate in terms of the game can be achieved when the ball hits at the same distance from either edge on both the player and computer paddles. In the most trivial case, when the point of impact is right in the middle of the paddles $d_p = \frac{l_p}{2}$ both angles are 0 and the ball is exchanged horizontally between the directly opposing paddles. There are no other variables to consider as per the rules of the game and a stalemate where the game never ends is possible via the equations.

On the Atari 2600, the game is sufficiently wise to eliminate the most trivial of condition, when the paddles directly face each other across the field and exchange the ball horizontally (Firgure 6). However, the possibility of an endless game whenever the ball bounces off a wall is still possible. After a sufficiently long exchange of the ball across the field between the players, the game decides to cancel the game. One parallel can be drawn between "Pong" game to chess, that has been proven to always end in a draw given ideal players (Newell, Shaw, & Simon, 1958; Bhattacharya, 2011) or to "Tic-Tac-Toe" that would always end in a draw given ideally proficient players.

As per the goals of the paper it is indeterminate at best to decide whether a player winning or achieving a stalemate by observation is to be considered "cheating" or just defeating the game; especially since there is no manipula-

tion of the game code, no hacks or cracks have been applied such that the stalemate is the result of the game design and its invariants.

A more complex example of the architecture of a game being exploited would be "Super Mario Bros" published by "Nintendo" in 1985 where a player can escape certain levels simply by breaking the upper wall and then completing the entire level by avoiding all obstacles (Figure 7). Perhaps avoiding all other obstacles sets the player at a disadvantage by skipping game content such that breaking the upper wall and completing the level is not considered a cheat.

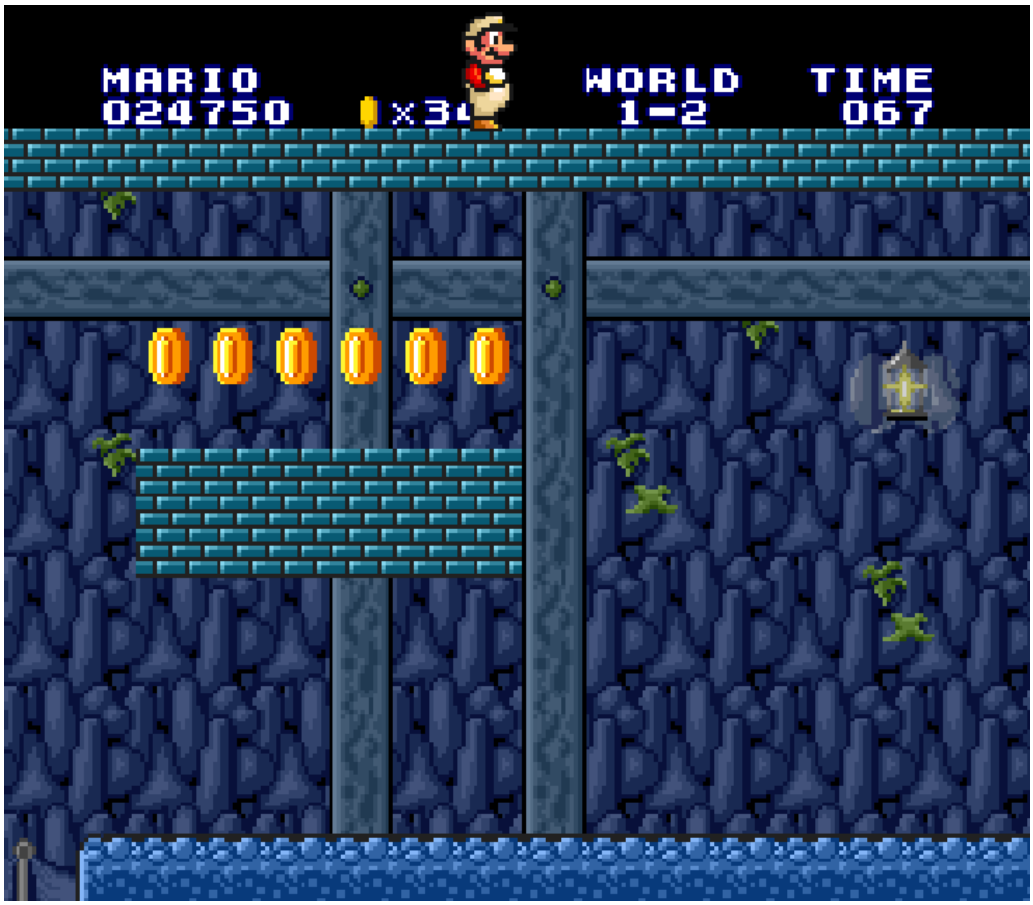

Figure 7: Skipping an entire level in "Super Mario Bros" by "Nintendo" just by breaking the upper wall and then walking all the way to the end of the level while avoiding all perils.

While the "Pong" example seems like an omission, the ability to break upper walls and skip levels in "Super Mario Bros" seems intentional, or, at the very least, something that has been observed by the developers but has been left unchanged in the final release of the game. Nevertheless, in both examples, the player uses the layout or geometry of the scene as leverage in order to gain an advantage.

Figure 8: Escaping the scene geometry in "Hobo: Tough Life", either by reaching areas that were not meant to be explored (left) or by using an in-game object to force the player through a wall (right), leads to the ability to travel quicker between various points of interest.

In "Hobo: Tough Life" by "Perun Creative" (Figure 8) a more emphasized example can be observed where a player can escape the scene geometry and travel within buildings or underneath the scene. Escaping the scene can be performed by picking up an object and forcing the player through the wall or just by reaching areas of the game that were never intended to be reached. Most likely, for the sake of reducing game requirements, some of the walls have been set to phantom and without any collision detection such that the player is able to traverse some of the walls. Otherwise, the player can be forced through the wall in a constrained space where the force that pushes the player through the wall is greater than the opposing force exerted by the wall as per the second law of Newtonian mechanics. As per the rightmost image in Figure 8, the player manages to push through the wall by narrowing the available space using a fence. This is most likely due to the physical resistance of the wall being lower than the resistance of the fence that gradually increases the force exerted upon the player and by transmission of

the force, the wall that the player stands against. Whether the player escapes the scene via boundaries that are set as phantom or the player forces through boundaries by exploiting the inaccuracies of the physics engine, the exploit is generically labeled as a “wall hack”.

Figure 9: A screen shot of the game “Days Gone” developed by ”Bend Studios” and published by ”PlayStation Mobile, Inc.” where an estimate of the time of day within the game is entirely possible by knowing the compass direction of the player and the position of the sun.

When developers use game engines to create games, the game engines tend to reproduce the physical properties of the real world such that it would be fair to note that in-game observations or deductions can be leveraged from the real world. Taking as an example a screenshot from the game “Days Gone” developed by ”Bend Studios” and published by ”PlayStation Mobile, Inc.” illustrated in Figure 9 it seems to be possible to estimate that the time of day is morning just by knowing the compass orientation of the player relative to the sun. It is known from the map that the player in the screenshot is travelling north and it can be observed that the shadow cast by the player is towards the left-hand side (west) with the sun on the right-hand side of the player (east) which allows us to infer that the game environment is depicting

the morning. Albeit a trivial observation, the case stands to show that game environments sometimes mimic the real world, that, in turn constrains the possible outcomes within a game such that we can say that it is known that certain outcomes are not possible. It is possible to say that due to the game following the real world environment, a player in "Days Gone" should prefer to travel westward with the sun behind the player in order to cast their shadow in front and to easily spot objects in front of the player as they are illuminated by the sun.

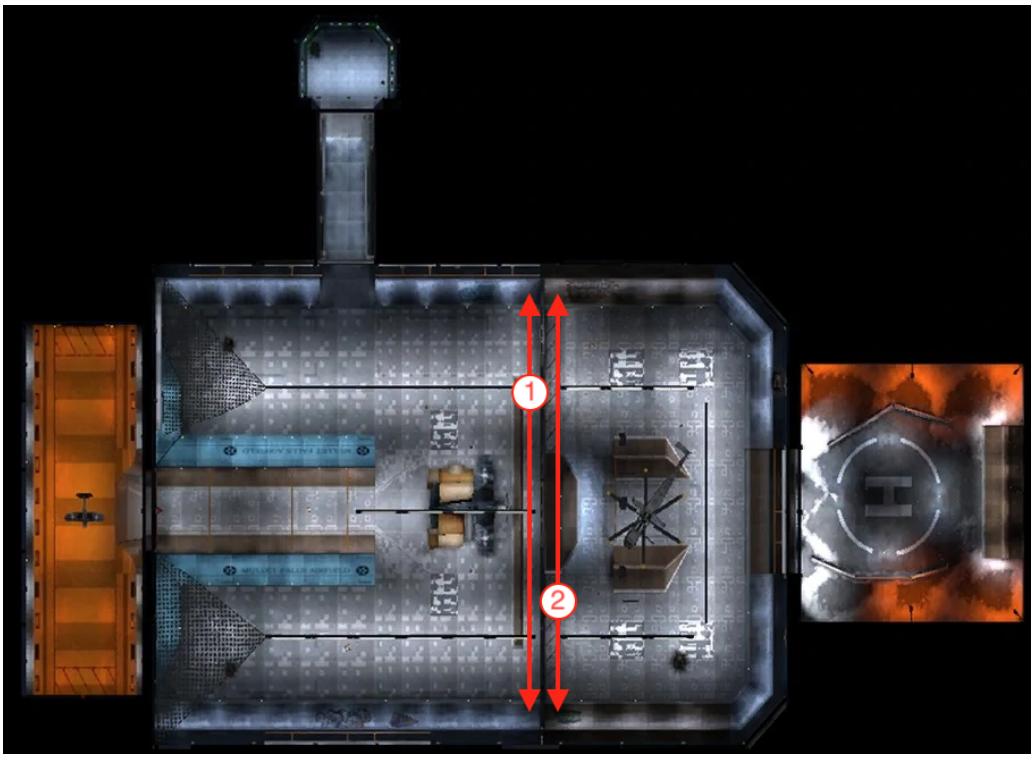

Figure 10: "Tony Hawks Pro Skater 2" where it seems likely that two points of oscillating equilibrium on the paths marked (1) and (2) are a direct consequence of the layout of the level and the rules of the game.

As a final yet intriguing example that was investigated by the authors, it seems that letting the skater in "Tony Hawks Pro Skater 2" (developed by "Neversoft" and published by "Activision") roam without user intervention will invariably end up with the skater oscillating like a pendulum on the two possible paths illustrated in Figure 10. Perhaps circling back to Deleuze,

it is the best guess of the authors that the reason for the existence of the two points of equilibrium lies within the architecture of the level itself. It so happens that due to the disposition of various obstacles and objects as they are distributed within the level, after a certain amount of random collisions, the skater ends up in either states of equilibrium.

The game was left running running for a large amount of time and it was observed that the skater ends up on the same possible paths. Various experiments have been carried out such as placing the player at random locations within the level and then letting go of the controls as well as testing the game on two different platforms in order to rule out implementation details. The amount of time till either of the two points varies greatly and either equilibrium point can be reached in a time span of minutes and up to hours.

3.7 Pathfinding Exploits

Related to the "Exploiting Architectural Invariants" in Chapter 3.6, players have been known to confront in-game NPCs and exploit constants in the pathfinding algorithms in order to gain an advantage. A good example would be the game "PacMan" (a maze game developed and released by "Namco" in 1980) where, due to the pathfinding algorithm of the four types of ghosts, the player can either hide from the ghosts indefinitely or figure out an algorithmic way of flawlessly winning the game without failure. Depending on the "scatter" and "chase" states of the ghosts as well as the targeting algorithm that chases the player in "PacMan", a few invariants of the game are generated that a player can exploit. More than likely, the invariants are undesired consequences of the chosen algorithms and the programming. The "PacMan" example illustrated in Figure 11 is a direct consequence of a number overflow condition that changes the components of movement vectors of the ghosts (Retro Game Mechanics Explained, 2019).

One could argue that the "PacMan" exploits are to be considered cheating but it is important to note that there is no special action required on the behalf of the player other than a good knowledge of the game. A different example shows up in a more recent game, "Warframe" by "Digital Extremes", where players exploit the pathfinding of attacking NPCs, camp within a section of the endless survival mission "Ophelia" and are able to gather resources indefinitely.

When the special spot was discovered by players (illustrated in Figure 12),

Figure 11: A "safe" place in "PacMan" where a player (yellow ball) will never be reached by the ghosts due to the algorithms governing the movement of the ghosts. Conditioned by the movement rules, the ghosts end up looping on certain paths that will never reach the player.

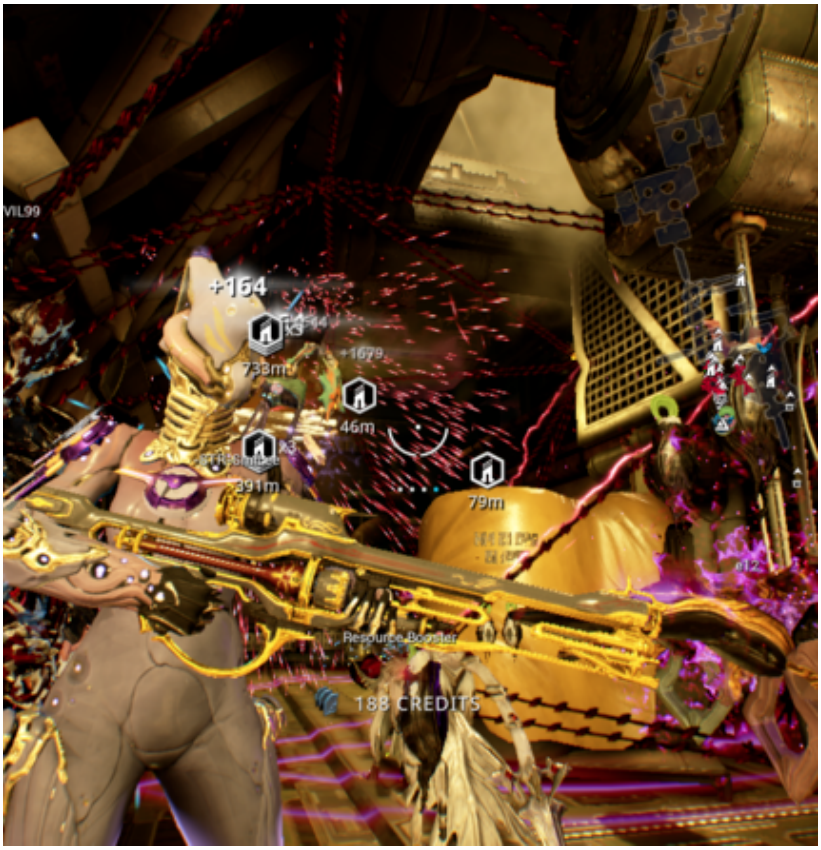

Figure 12: A camping spot used by players in the game named "Warframe" by "Digital Extremes" where the players are able to amass resources indefinitely without being killed by the NPCs attacking them.

the anti-cheat system was triggered often and players would end up banned

from the game. The ban emitted was detected by an automated anti-cheat mechanism that counted the number of resources acquired and did not detect what the players were doing. In that sense, the ban was a false positive (a "trade ban") and the threshold was removed by "Digital Extremes" via an update.

After a while, an attempt by "Digital Extremes" was made to address the problem yet the fix was local to the "Ophelia" mission and, as the authors would like to argue, as laid out in Chapter 3.6, the problem pertains more to the procedural algorithm that generates the level and creates the enclosure within the "Ophelia" mission. As illustrated in Figure 13, a new camping spot similar to the camping spot within the "Ophelia" mission is generated inside the "V Prime" mission in "Warframe", where players are able to camp again within an enclosed space while being shielded from attacking NPCs.

Players have been known in other games to drag NPCs around the in-game world into positions that have them at an advantage. The action of dragging an NPC is known as "kiting" and have been performed at large in various games. One example of kiting took place in a game named "World of Warcraft" by "Blizzard Entertainment" in 2005 where players belonging to the in-game guild named "Khadars Rage" managed to kite a high level NPC "Kazzak" into one of the in-game capitals "Stormwind" (Figure 14). The manoeuvre lead "Kazzak" to unleash a barrage of high level attacks against all the other players in "Stormwind" and the players responsible with the kiting were banned by "Blizzard Entertainment" from the game.

It stands to mention that there were was no hacking involved in kiting "Kazzak" and that the NPC was just dragged across the in-game world just by following the rules of the game itself, ie: generating sufficient threat to keep the NPC running after the players well beyond the intended area of movement of the NPC. The reason for the ban, in this particular case, was most likely backed by the EULA labelling the players as being disruptive to the game server or using too many resources but in particular because the kiting may have affected players other than the players performing the kiting. Since Massively Multiplayer Oriented Role Playing Games (MMORPG) game design seems to follow the principle of threat generation to determine how long an enemy NPC follows the player, kiting is ubiquitous to all games following the same game design model such that "World of Warcraft" by "Blizzard Entertainment" cannot be considered a singular example.

Another example that is closely related to both pathfinding and game architecture is the notion itself of "taxiing", "running", "ferrying" or "carry-

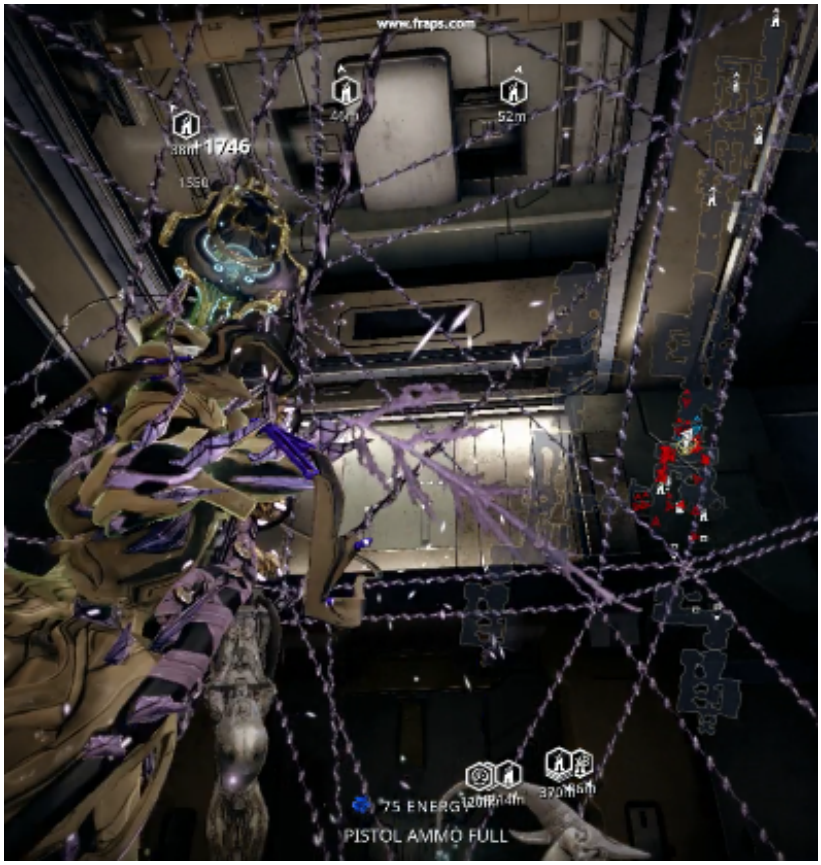

Figure 13: A different camping spot yet undiscovered by players in “Warfame” in the “V Prime” mission by “Digital Extremes” following the fix for the previous spot on “Ophelia” in Figure 12.

ing” that involves a high level player assisting a low-level player by skipping through game content in order to reach a well-established destination. The person performing the “taxiing” invites the low-level player to the party and then runs through multiple areas of the game, dragging the low-level player along, by completing tasks that are rewarded to the entire party. Since the low-level player is part of the party, the award applies to the low-level player as well as to the person performing the “taxiing”.

One example of “taxiing” that stands aside is to be found in “GuildWars” by “NCSoft” where a large amount of game content can be skipped by going from the starter area to a key area of the game where maximum level apparel

Figure 14: A high-level NPC named “Kazzak” getting dragged into the human capital “Stormwind” by the “Khadars Rage” guild.

can be purchased. The “Droknar’s Forge” run, as illustrated in Figure 15 was famous for being difficult to accomplish. The run contained high level NPCs as well as environmental perils that made running from “Beacon’s Perch” to “Droknar’s Forge” difficult. The advantage of hiring a player to perform the “Droknar’s Forge” run was that a starting player could avoid “Ascension” that required a large part of the game to be completed. The “Droknar’s Forge” run was the preferred choice for players that created a second game character and did not want to complete the entire game again but wanted to reach the trading hub at “Droknar’s Forge”. Even though the “Droknar’s Forge” run became famous, the run was not tampered with by “NCSoft” such that the “Droknar’s Forge” run became part of the game content that

Figure 15: The “Droknar’s Forge” taxiing path in the “Guildwars” game by “NCSoft” leading from “Becon’s Perch” to “Droknar’s Forge” thereby skipping the need to complete “Ascension” within the game.

may have not been anticipated by the game developers.

3.8 Animations and Cutscenes

Animations and cutscenes that are displayed at certain times during gameplay can become opportunities that affect the outcome of the game. Whenever a cutscene takes place, the game progress is temporarily suspended in order to play the animation. While the animation is running, the players become temporarily immune to all damage and the number of frames for which the animation unfolds is named by gamers ”inviciframes”. The effect of becoming temporarily immune to all damage can scale depending on whether the game developers intend to display an entire movie or whether a short animation is to be played. In some cases, when a player is NPCs and manages to escape and trigger a cutscene, the game allows the player to

progress even without completing the fight.

Figure 16: Grabbing and enemy in “Cadillacs and Dinosaurs” by “Capcom” and then throwing them triggers a short animation during which the player becomes invulnerable.

As an example from the arcade era in the “Cadillacs and Dinosaurs” by “Capcom” game, grabbing one of the attacking NPCs as illustrated in Figure 16 and then throwing them on the ground will grant the player a short temporary invulnerability as the animation plays out. The grabbing and throwing action can thereby be exploited by a skillfull player by performing the same action in succession in order to avoid incoming damage.

A similar example can be found in a more recent game “Red Dead Redemption II” by “Rockstar Games” where petting a dog grants the player a temporary invulnerability as seen in Figure 17. At times, in other games, entering a cutscene will cancel all other tasks and allow the player to continue into the next sequence of the game without penalties such as in the game “Styx” by “Cyanide Studio”.

In any case, animations have proven to be an avenue of bypassing dan-

Figure 17: Patting a dog repetitively in "Red Dead Redemption II" by Rockstar Games grants temporary invulnerability to incoming attacks.

gerous situations where both animations and cutscenes can be exploited to gain an advantage.

4 Conclusions

It is clear that influencing the game by tampering with the software or hardware is indeed a "hack" yet there are many other cases where the player has been following the game rules as established by the game developers. In many cases, the techniques used to exploit games are common to many other games and are a result of the game design patterns being applied in perpetuity. It becomes clear that "wallhacks", "taxiing", exploiting pathfinding, cutscene

exploits or even "saving and reloading" the game are to be distinguished from direct memory manipulations of games because the former rely on a good knowledge of the game contrasted to directly modifying the software or hardware. All the ubiquitous techniques apply too all games regardless whether the games are single player, multiplayer or provided as service and the techniques are applicable to a whole sets of video game genres. The decision of video game maintainers in case of multiplayer or online games to bar access to certain players employing these techniques are most of the time empirical at best due to being very difficult to prove. Online game maintainers set certain thresholds in automated checks that when exceeded, a player is considered to be cheating such that false positives are expected.

References

Bhattacharya, P. (2011). Deep blue-search algorithms. *Indian Institute of Technology Bombay*.

Cano, N. (2016). *Game hacking: developing autonomous bots for online games*. No Starch Press.

Deleuze, G. (2014). *Pourparlers. 1972-1990*. Minuit.

Hardy, N. (1988, October). The confused deputy: (or why capabilities might have been invented). *SIGOPS Oper. Syst. Rev.*, *22*, 36–38. Retrieved from `http://dl.acm.org/citation.cfm?id=54289.871709`

Heron, M. (2013). "likely to be eaten by a grue"—the relevance of text games in the modern era. *The Computer Games Journal*, *2*(1), 55–67.

Hoare, C. A. R. (1969, October). An axiomatic basis for computer programming. *Commun. ACM*, *12*, 576–580. Retrieved from `http://doi.acm.org/10.1145/363235.363259` doi: http://doi.acm.org/10.1145/363235.363259

Jinpeng, W., & Calton, P. (2005). Tocttou vulnerabilities in unix-style file systems: an anatomical study. In *Proceedings of the 4th conference on usenix conference on file and storage technologies - volume 4* (pp. 12–12). Berkeley, CA, USA: USENIX Association. Retrieved from `http://dl.acm.org/citation.cfm?id=1251028.1251040`

Kolasinski, W., Blake, J., Anthony, J., Price, W., & Smith, E. (1979). Simulation of cosmic-ray induced soft errors and latchup in integrated-circuit computer memories. *IEEE Transactions on Nuclear Science*, *26*(6), 5087–5091.

Melis, A. P., Hare, B., & Tomasello, M. (2006). Chimpanzees recruit the best collaborators. *Science*, *311*(5765), 1297–1300.

Merrill, H. (2003, Jan). *Re: Error handling.* http://www.mail-archive.com/dbi-users@perl.org/msg15388.html.

Newell, A., Shaw, J. C., & Simon, H. A. (1958). Chess-playing programs and the problem of complexity. *IBM Journal of Research and Development*, *2*(4), 320–335.

Nieborg, D. (2004). *America's army: More than a game.* https://www.academia.edu/download/8173013/isaga_nieborg.pdf.

Retro Game Mechanics Explained. (2019). *Pac-man ghost ai explained* (Vol. 11). https://www.youtube.com/watch?v=ataGotQ7ir8.

Seltzer, M. I. (1992). *File system performance and transaction support.*

Sharp, J. E. (2006). There oughta be a law: Crafting effective weapons in the war against spyware. *Hous. L. Rev.*, *43*, 879.

Tan, M.-L., & Ng, C. A. (2014, April). *Programmable computer mouse* (No. US8711095B2). Retrieved 2020-12-07, from https://patents.google.com/patent/US8711095B2/en

Yu, S.-Y., Hammerla, N., Yan, J., & Andras, P. (2012). A statistical aimbot detection method for online fps games. In *The 2012 international joint conference on neural networks (ijcnn)* (pp. 1–8).

Zalewski, M. (2001, May). *Delivering signals for fun and profit.* http://lcamtuf.coredump.cx/signals.txt.