Securing 2D Games Against Speedrunning Video Forgeries: System Design Proposal

Megan Prakash (meganp@mit.edu)
6.857 Final Paper
Spring 2021

Abstract

“Speedrunning” is an increasingly popular activity within the video game space. Players, known as “speedrunners,” attempt to complete an entire video game or video game level in as little time as possible. The recent rise of streaming platforms and a culture of consuming video game content has caused speedrunning to rocket upwards in popularity and become competitive on a worldwide scale, where top players become known across the entire community. As a result, fake speedrun techniques have proliferated. In this paper I propose a system design to augment a 2D video game such that (1) the game is resistant to the replay and editing attacks we describe, and (2) gameplay and streaming experience is not compromised for the user or the streaming platform.

Background

“Speedrunning” is an increasingly popular activity within the video game space. Players, known as “speedrunners,” attempt to complete an entire video game or video game level in as little time as possible. This often requires the player to execute extremely precise controller inputs or take advantage of bugs in the game source code, which they use to evade the game’s obstacles extremely rapidly. For example, it takes most players at least 10 hours to complete Super Mario 64[1], but the speedrun world record tracked by speedrun.com is currently 1 hour, 38 minutes, and 21 seconds, as of writing this paper [2].

An integral part of speedrunning culture is players recording and sharing videos of their speedruns. The recent rise of streaming platforms and a culture of consuming video game content has caused speedrunning to rocket upwards in popularity and become competitive on

[1] https://howlongtobeat.com/game.php?id=9364
[2] https://www.speedrun.com/sm64

a worldwide scale [3], where top players become known across the entire community. As a result, fake speedrun techniques have proliferated.[4]

Some of the most common methods for creating a fake speedrun video are (1) source code hacks, where the user changes the source code of a game to make it easier to speedrun, (2) tool-assisted speedruns, where the user uses software or a physical robot to send scripted controller input to the game, and (3) video fakes, where the speedrun video shared on a streaming platform has been spliced from multiple video clips, and/or a "live" stream video was in fact recorded previously[5].
In this paper, I focus on the third technique: video fakes. The current "state of the art" technique for detecting speedrun video fakes is essentially for communities to rely on human moderators and audience members to identify inconsistencies in the video. For example, if two video clips have been spliced together, there may be discontinuities in the game graphics at the point of the splice. However, it is not guaranteed that discontinuities will be identified or that they will even exist in a spliced video. Additionally, if a malicious speedrunner claims to be streaming live but in fact is streaming a pre-recorded video, there is no way to verify this. Players often try to demonstrate the integrity of their video gameplay with good-faith measures, such as showing their computer's clock within the screen, but this is not sufficient for establishing the legitimacy of their speedrun videos.

Some game developers are now implementing software-based strategies for preventing speedrun forgeries within their games that rely on online systems. Given this paradigm, I will propose addressing speedrun video fakes by implementing additional security measures within a game that interface with a trusted verification server. In particular, I focus on the space of 2D games, which I will define in the next section.

Problem Statement

In this paper I propose a system design to augment a 2D video game such that (1) the game is resistant to the replay and editing attacks we describe, and (2) gameplay and streaming experience is not compromised for the user or the streaming platform. For live-streamed gameplay, our design adds the ability for a streaming platform to request attestation from the player, who must prove the liveness of their streamed gameplay. For uploaded speedrun recordings, our design adds a subtle augmentation to the background of the 2D game that allows the video platform to computationally verify the integrity of the recording, i.e. detecting if the recording has been edited.

---

[3] https://www.wired.com/2016/01/speedruns/
[4] https://www.polygon.com/2017/5/22/15675028/speedrunning-cheats
[5] https://arstechnica.com/gaming/2019/12/how-the-scourge-of-cheating-is-changing-speedrunning/

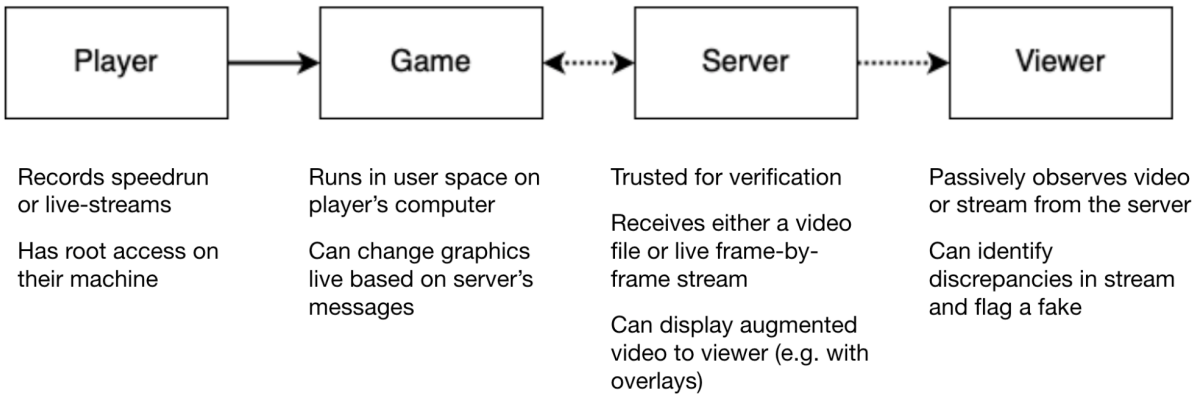
I intend to achieve enough resistance to the attacks such that the attacks are “sufficiently inconvenient” for a malicious player to achieve. We define “sufficiently inconvenient” to mean that the player would have to resort to hacking the game source code in order to circumvent the security measures.

I define a “2D game” as a game that has a static background plane and a foreground plane with dynamic gameplay. An example is below:

The mountain and sky in the background are a static image that scrolls as the player’s character moves forward. Superimposed on the background are the gameplay elements, which are dynamic and interactive.

Definitions

The actors in the system are the player, the game, the server, and the viewer.



- **Game**: 2D appearance, for example a platformer side-scrolling game (show figures). Contains a background plane and foreground elements. The background plane contains a static image that translates across the screen.
- **Streaming platform**: Consumes video streams frame-by-frame as uploaded in real time to the website, then displays them to stream viewers. The platform has the ability to augment the frames before displaying them to the viewers, commonly used to overlay live chat transcripts and other interactive elements.
- **Player**: plays the game using their keyboard and mouse and attempts to complete the entire game or level as rapidly as possible, often taking advantage of narrow windows of opportunity or very precise maneuvers. A malicious player launching the attacks we describe will (1) record multiple gameplay videos, then splice them together, and/or (2) claim to be streaming live from their real-time gameplay, but in reality is streaming a recorded video to the platform.
- **Viewers**: passively observe gameplay with the ability to comment live and view the video or recorded stream after the gameplay event. Viewers have an interest in preventing speedrun forgery, as described above.

Assumptions

- The player is assumed to be potentially malicious and has root access to their computer. The game developer, server, and viewer are considered trusted.
- Perfect security isn't possible, but we can make it "sufficiently inconvenient" to forge a video that passes verification by the server
- The streaming site has access to trusted third party users. It can serve as a root of trust.

As a note, capturing user keypresses to verify that a video corresponds to a legal set of game inputs and was contiguously does NOT guarantee the video's authenticity. Tool-assisted speedruns, as mentioned earlier, are very common and in fact are their own realm of speedrun communities. Even though tool-assisted speedruns are outside the scope of this paper, I discard keypress captures as a possible solution because it is so easily circumvented by a malicious player trying to forge a speedrun.

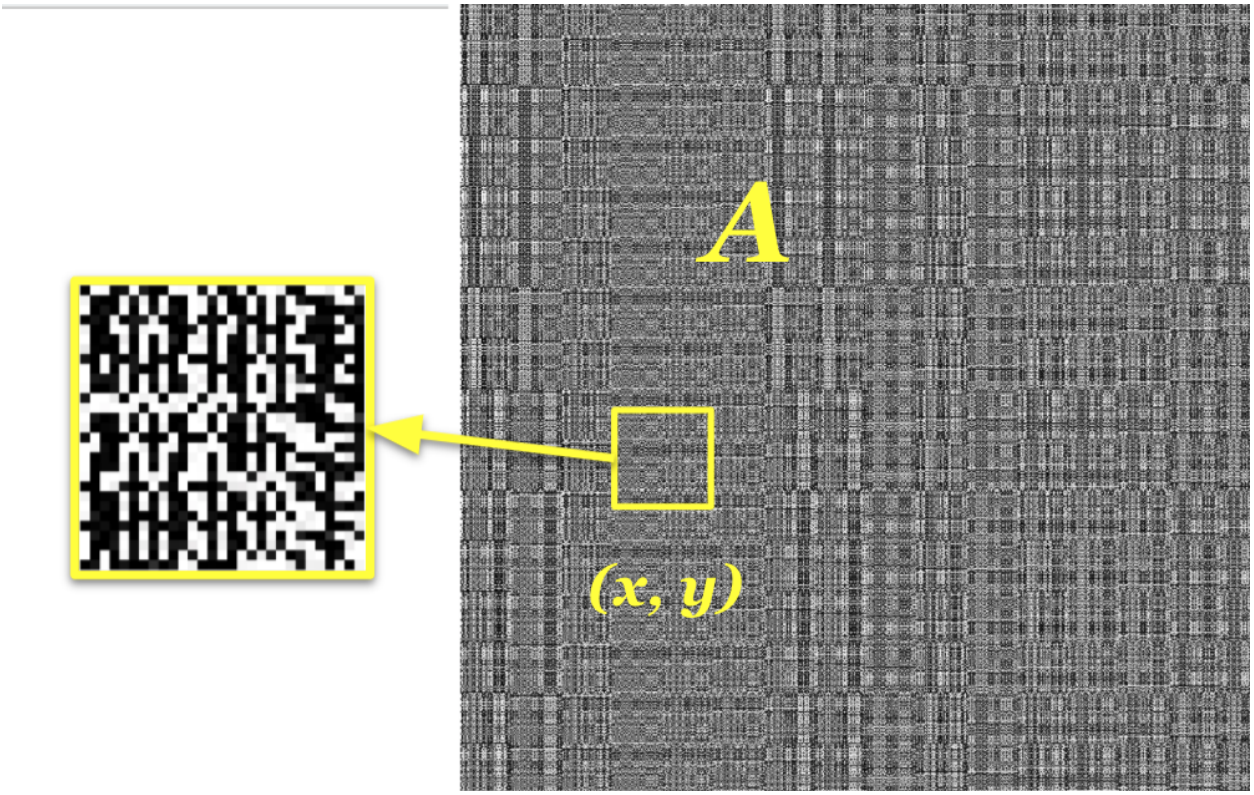
Proposal

I propose the addition of self-identifying dot patterns to the background of a 2D game, which will encode attestation information and are difficult to remove from a video. This is done with the purpose of adding methods to detect the **integrity** and **liveness** of a streamed video.

Position-coding patterns

Position-coding patterns (here referred to as “dot patterns”)[678]are a technology that have existed for multiple decades and were originally developed for multimodal interaction with printed paper. A dot pattern generally consists of a random-appearing matrix of nearly-imperceptible dots, printed onto a sheet of paper. Any sample of the dots, e.g. captured by a camera mounted on a pen, can be used to identify (1) which page is being observed, and (2) where on the page the sample is located.

In the following image, array ***A*** is an array of marks comprising a dot pattern. The subsample shown is taken from location (x, y) within A. Given the correct decoding algorithm for the generated pattern **A**, it is possible to decode the subsample and retrieve the identity of **A** and the location (x, y)



[6] https://link.springer.com/article/10.1007/s00138-007-0093-z
[7] https://arxiv.org/pdf/0706.0869.pdf
[8] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7487653

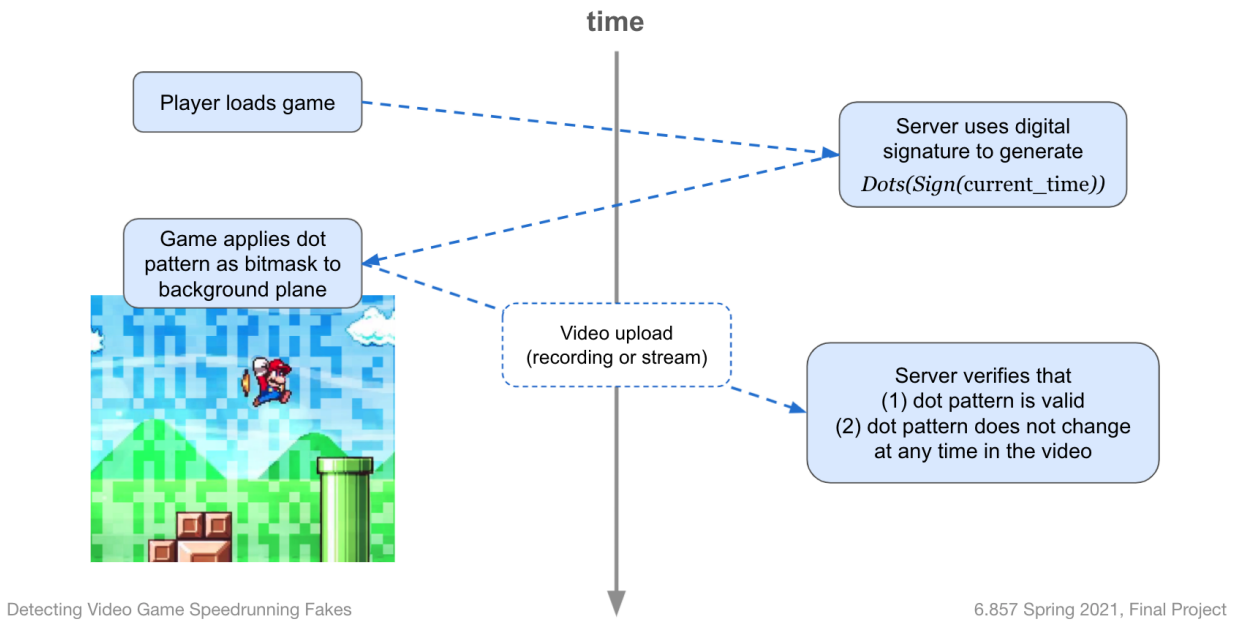
Given this property, it is possible to use a dot pattern to encode an arbitrary message that can be decoded using any sufficiently large subsample. I will refer to this operation as generating array **A** from message **msg** using **A** = *Dots(***msg***)*.

I propose using dot patterns to **create unique encodings of single gameplay events**, then watermarking the background of a 2D game with a corresponding dot pattern each time gameplay begins. Any third party can subsequently watch a gameplay video and decode the background pattern in order to identify the gameplay event. This occurs as follows:

Pattern generation scheme:

1. Player loads the game. At loading time, the game phones the verification server.
2. The server takes the current timestamp and signs it to make *Sign(current_time)*. Then, it encodes it in a dot pattern using *Dots(Sign(current_time))*. Finally, it sends this dot pattern back to the game.
3. The game loads the background image into memory, performs some randomization of it (e.g. move elements around, change the color values), then applies the server's dot pattern as a mask to the background plane such that the dots are near-invisible but can still be identified.
4. When the server consumes the gameplay video, either as a recording or a stream, it can decode the dot pattern in each frame. Part of the reason this is easy for the server is because the dot pattern is decodable using *any* subsample, so the server can decode it no matter how the background has scrolled or how the foreground is occluding it.

See figure on next page:



Security strength

The server identifies forgery as follows:

- If the dot pattern changes at any time, then the video has a splice.
- If the timestamp is earlier than the start time of a claimed livestream, then the video wasn't live.
- If the dots aren't correctly occluded by the foreground, then the dots have been added after the video was recorded.

It is plausible that, since the server has a method for identifying the dots, the same method can be used by a player to identify dots in their video, remove them, then copy dots onto the video from a different video clip. A potential countermeasure is for the game to add dots to the background in a lossy way that makes them impossible to remove.

Game security:

- The dot pattern scrolls with the game, so it is not possible for the player to capture it and reuse it throughout a live stream
- The game needs to store the base background image on disk, as part of its source. However, the version of the image with the dots superimposed is held in memory, making it more difficult for the user to find and modify.

Limits of the system

Implementing this design in a video game takes cooperation from the game developer. Though game companies have recently demonstrated interest in anti-forgery measures, such measures are only available to newly created games. Meanwhile, Super Mario 64 was published in 1996 and remains an extremely popular game in the speedrunning community.

Additionally, as mentioned in the introduction, video fakes are only one aspect of speedrunning forgery.

Hope you have a wonderful summer. Be well!
Thanks,
Megan