We Call This Controller Skip: AI for Speedrunning

Michael Cook[1], M Charity[2], Maren Awiszus[3], Filippo Carnovalini[4] and Alexander Dockhorn[5]

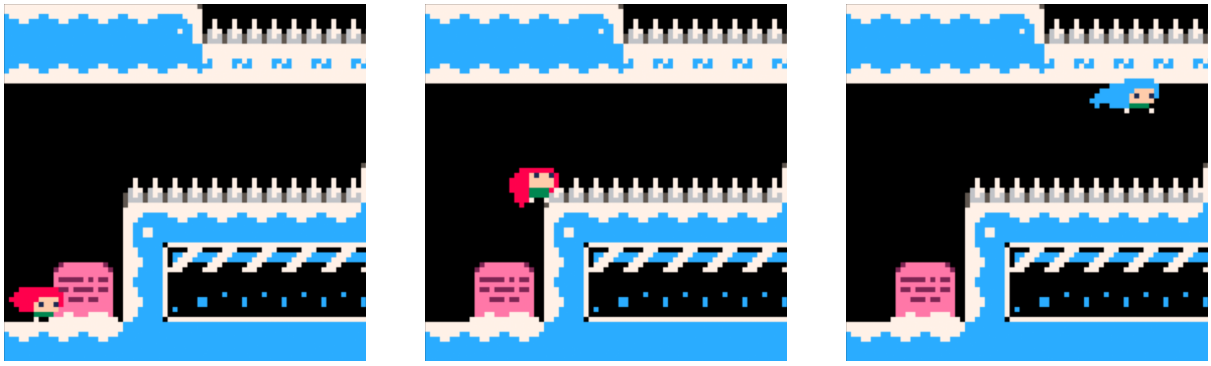
[1]Department of Informatics, King's College London, UK
[2]Department of Computer Science, University of Richmond, USA
[3]Viscom SE, Hannover, Germany
[4]CCLab, Artificial Intelligence Lab, Vrije Universiteit Brussel, Belgium
[5]SDU Metaverse Lab, University of Southern Denmark, Denmark

Abstract

Speedrunning, a community centred around the practice of playing games as fast as possible and often under constraints, is a rich and vibrant part of modern games culture. It is also an area largely untouched by game AI research, despite it having its own culture of technical innovation and research. In this paper, we provide an overview of modern speedrunning, describe research areas already pursued by the community, and report on exploratory work investigating how AI research might be applied to simple speedrunning processes. We close by suggesting new research questions this space reveals for us, and suggesting ways that we can peacefully and respectfully work alongside and as a part of the speedrunning community.

Keywords

general game playing, speedrunning, game playing agents, heuristic search, reinforcement learning



(a) The player dashes up from the floor below towards the corner.

(b) They pass through the spikes from below, touching the floor.

(c) They can then jump off the spikes and dash across.

Figure 1: A *spike jump*, a basic speedrunning technique in the game *Celeste Classic* [1]. This example is demonstrated in our instrumented version of *Celeste Tech Training* [2] written in PICO-8.

1. Introduction

Speedrunning is the art and science of playing games quickly [3]. Speedruns require a high familiarity with a specific game and its various speedrun routes, the knowledge and manipulation of glitches, bugs, and vulnerabilities in the game's code, and a great deal of player skill to execute. A speedrun's goal is typically to finish a game as fast as possible, but it is often subject to additional constraints that change the goal or how it must be achieved.

Traditional speedrunning emerged in the 1990s, as games began tracking player times and offering target times to beat. *DOOM* offered target times set by the developers for each level and also allowed

Joint AIIDE Workshop on Experimental Artificial Intelligence in Games and Intelligent Narrative Technologies, November 10-11, 2025, Edmonton, Canada.

mike@possibilityspace.org (M. Cook); mcharit2@richmond.edu (M. Charity); mcharit2@richmond.edu (M. Awiszus); mcharit2@richmond.edu (F. Carnovalini); mcharit2@richmond.edu (A. Dockhorn)

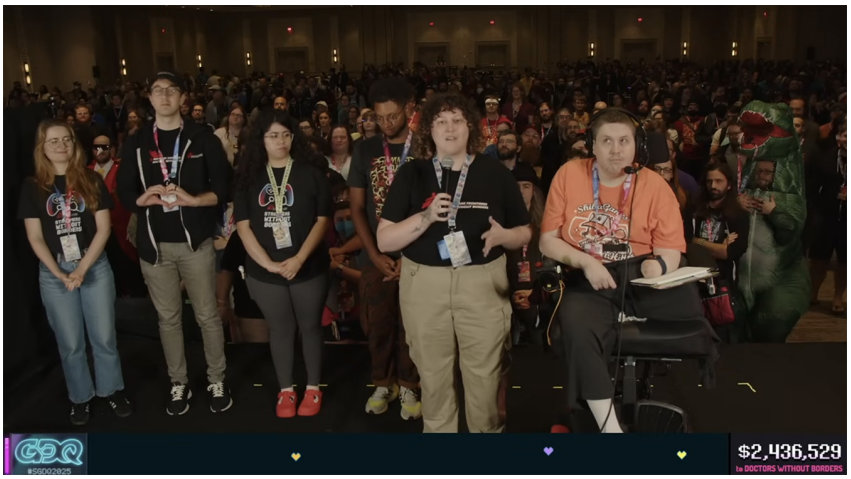


Figure 2: A screenshot from the finale of *Summer Games Done Quick 2025*. The total raised can be seen in the bottom-right of the screen. The person speaking, second from right, is from *Doctors Without Borders*, thanking the viewers with the crowd behind them.

players to record their performance and store it as a small file containing only player inputs, meaning they could be shared online easily, and other players could review and watch them in-game [1].

Today, speedrunning is a major subculture within videogames. The most popular website for organising and sharing speedruns is speedrun.com[2], which hosts over 5.4 million speedruns, with 2.4 million registered users. The community hosts regular charity marathons such as *Games Done Quick*[3], which has raised over $50 million by livestreaming speedruns and soliciting donations. Speedrunners are often engaged in game development too, either being invited as testers for games, or as a part of the design community for mods, fangames and standalone titles. It is increasingly common for games to directly support speedrunners through custom features, such as in-game timers and other quality-of-life features that make running a game easier [4].

The speedrunning community also has an interest in technical innovation and problem-solving. Tool-assisted speedruns, randomisers and glitch hunting (all of which we describe at length later) are aspects of speedrunning that have led to the design of custom tools and hacks that incorporate ideas from procedural generation, program analysis, verification and more [5]. Communities for speedrunning a single videogame will often develop bespoke tools for modifying a game to make it easier to speedrun, or easier to investigate and discover things about.

Despite the incredible growth of speedrunning over the past three decades, its enormous impact on the culture of games and game development, and its passionate technical community, speedrunning has seen little exploration by AI researchers. In this paper, we outline the opportunities we see in this space for AI researchers to engage with the speedrunning community, to make contributions (without detracting from or devaluing the spirit of speedrunning), and to find exciting new research questions that challenge existing AI techniques. To support this, we also report on exploratory work we undertook to probe different approaches to writing AI speedrunners for the 'fantasy console' game engine PICO-8[4], and identify areas of future work that could build on this in the future.

2. Background and Related Work

This section introduces some speedrunning concepts, in particular how categories work, the technical problems and challenges involved in speedrunning, and related work in the area. We also contrast the traditional idea of game-playing AI with the challenge of speedrunning, to highlight the gap in research.

[1] Original demo files for DOOM runs made in 1994 can still be found online: http://cd.textfiles.com/doomcompanion/
[2] https://www.speedrun.com/
[3] https://gamesdonequick.com/
[4] https://www.lexaloffle.com/pico-8.php

Due to the relatively small number of existing papers on speedrunning (which we discuss) this section is longer than usual, to provide a good grounding for all readers.

2.1. Speedrun Categories

A typical speedrun will target a specific game (for modern games, sometimes a specific *version* of a specific game) and aim to complete it in the fastest time possible. 'Complete' is usually defined as seeing the credits roll at the end of the game, or the last moment that control is taken away from the player. This traditional format of speedrunning is sometimes referred to as an 'Any%' (pronounced 'any percent') run – that is, reaching the end by any means. This is known as a *category* of run. Games commonly have many categories of speedrun within them, each with different goals and constraints that the community has deemed meaningfully different and interesting [6].

Speedrun categories are defined by a community on a case-by-case basis. Although a single person can define a category, categories typically only become official once multiple runners express an interest in competing in it. A common category is '100%', which usually requires the player to reach the end of the game having completed most optional objectives. The 100% speedrun for *Super Metroid* requires the player to finish the game with all items collected[5]. Some games will have categories that are specific to a particular game. *Super Mario 64*, for example, has categories for 120 Stars (which is every star in the game, analogous to a 100% category) as well as 70 Stars, 16 Stars, 1 Star, and 0 Stars[6]. The number and variety of categories a game needs is negotiated by the community and shifts over time.

Categories can also change the way in which the game is played, putting restrictions on the game or the player. Common category modifiers require the runner to avoid using certain techniques, such as 'No Out Of Bounds' which prevents the runner from using any techniques which push themselves outside of the level's 'bounds' (which can mean different things in different games; usually the 2D or 3D space the player is intended to be in)[7]. Categories can also change the game itself. For games with procedural or random elements, a common category is a 'Seeded' run where all players play on the same random seed, meaning that nondeterministic elements are fixed and the same for all runners[8]. This changes the nature of the speedrun, mitigating some of the randomness, enabling the community to collectively plan strategies, and placing more emphasis on the execution of a strategy.

2.2. Speedrun Technology

The speedrunning community is not just made up of speedrunners – many different people make contributions to the community. Some of those are developers of tools that help speedrunners practice or investigate games. For example, the introduction of a *cutscene removal* mod for *Final Fantasy IX* reduced the length of the speedrun by several hours, making it more viable for speedrunners to practice and increasing interest in the category[9].

One of the best-known examples of technical innovation in the speedrunning scene is in *tool-assisted speedruns* or TAS [7]. These are automated speedruns that execute a series of instructions at exact times, pre-programmed by human authors. TAS can be seen as a hybrid speedrun, where the exact sequence and timing of inputs is discovered by a human, but the perfect execution of the instructions is done automatically by software. TAS are considered their own category of speedrunning work, but have a close connection to speedrunning and are often showcased alongside regular speedruns at events. Since they are speedruns that can only be executed with perfect timing and reactions, they represent a theoretical best-case for human speedruns. A human speedrun can only match a TAS, never exceed it (if a human exceeded a TAS, they could simply take those inputs and record them as the new best TAS).

The development of new TAS strategies also reveals exploits in a game that speedrunners may be able to use elsewhere, or eventually learn to execute manually. Because TAS runs are often made and

[5]https://www.speedrun.com/supermetroid
[6]https://www.speedrun.com/sm64
[7]For example, Portal: https://www.speedrun.com/portal
[8]For example, Minecraft: https://www.speedrun.com/mc
[9]https://www.speedrun.com/ff9

Figure 3: A screenshot of a Hollow Knight randomiser, showing different settings including allowable skips (top-left) and item types to randomise (right). Screenshot taken from a video by twitch.tv/skurrypls.

tweaked frame-by-frame, they can pay attention to minor details and find exploits that would not be discovered through normal play. This is related to the act of *glitch hunting* in speedrun communities, where players explore aspects of a game they believe may be vulnerable to manipulation. This can involve trying strategies that have caused glitches in other games (since games often share engines or implementations).

It is important to emphasise that TAS runs are *not* guided by AI search – they are made manually, and tweaked by hand to find the optimal route through a game. Their superhuman execution means that extremely unusual feats can be achieved, such as a TAS of *Super Mario World* which can use pixel- and frame-perfect tricks to manipulate registers and write code into the memory of the Super Nintendo console, then execute it. Another TAS author was able to find a route that played four different *Super Mario* games (1, 2, 3, and Lost Levels) simultaneously, using the *same* inputs to all four games, and completing all four games at the exact same moment[10]. This delight in pushing both hardware and software to do improbable things is a good example of how creative and fun the speedrunning community can be.

2.3. Randomisers

Randomisers are modifications to a game that change static game elements into dynamic ones. Common types of randomiser swap around enemy types or locations, or swap the locations of items. More extreme randomisers can change where doors or exits lead to, what the goal of the game is, or how the game looks or sounds. Randomisers generally fall into two categories: playability-preserving and -non-preserving. In the latter case, the randomiser makes no claim that any particular use of the randomiser is a completable game, or that it adheres to a particular difficulty level. These are often used for entertainment value outside of speedrunning.

In speedrunning, randomisers are more often playability-preserving. These randomisers come with internal logic, often customisable, that changes how the randomisation affects the game. For example, in the *Hollow Knight* randomiser, there is specific logic to ensure that unlockable abilities are never placed behind obstacles that require that ability to traverse (more generally: they ensure that keys are never behind the locks they open). More complex randomisers can take into account specific speedrunning techniques, meaning that they can place some items in out-of-reach areas if the speedrunner has stated they know how to use glitches or tricks to access them. A screenshot of a Hollow Knight randomiser can be seen in Fig. 3.

The most prominent example of speedrunning being covered in game AI research is the 2022 paper by

[10]https://tasvideos.org/2636M

Mawhorter et al. [5] in which they argue that the randomiser community – which is not exclusively a speedrunning community but which has a healthy overlap with it – are essentially procedural generation researchers. Many randomiser mods and tools require solving complex problems in procedural level design, as well as human-computer interaction problems in surfacing the tools to non-expert users.

2.4. Game AI and Speedrunnning

Game AI research has already extensively examined the problem of playing a game automatically. One of the most famous examples of this is Robin Baumgarten's A* Mario bot, which has become a recurring example of game AI research creating impressive gameplay [8]. Baumgarten's bot was created for an AI Platformer competition where success was measured in distance travelled, not time taken, but this nevertheless results in behaviour which looks quite similar to a TAS speedrun in many ways – performing last-second saves and risky maneuvers to get through a level efficiently.

This practice of designing AI agents to play games differs from the task of speedrunning, even in the sense of a TAS run, in several important ways. First, research into such agents is normally more interested in tackling a general problem rather than a specific game. The AI Platformer competition, which Baumgarten's bot entered and won, was not challenging entrants to complete a specific Super Mario Bros. level, or even game, from start to finish; instead, it was interested in the general task of controlling a game character in a platforming environment *similar to* Super Mario Bros. This means such agents do not use strategies specific to a particular game, and in most cases are not playing on specific hardware or software either. The PAC-MAN AI competitions that ran in the 2010s, for example, could not reach the game's infamous 'kill screen' because they were run on a researcher's reimplementation of the game [9].

Game-playing AI agents often have their set of available actions restricted in some way. Researchers' goals when creating AI agents are usually either: to provide a realistic opponent for humans; to mimic average human performance for player modelling; or to compete with and exceed human performance. In all three cases, there is an expectation that the agent will generally avoid using glitched or unusual behaviour, and changes are sometimes made to the AI agent to reflect this. In solving Sokoban puzzles, for example, the best AI solvers ignore player control and instead abstract the game to be a problem of moving boxes instead [10]. This cuts out a large part of the gameplay space so as to focus on what is seen as most relevant, which is useful for the research problem being tackled. However it is missing out on potentially important space that could yield new techniques or speedrunning strategies.

Existing work in game AI related specifically to speedrunning is rare. In [11] Groß et al. attempt to formalise the process of *routing* in speedruns. Routing is the process by which speedrunners organise a particular path through a game. Where multiple paths are possible, it may be more optimal to complete tasks in a certain order. Routes can also change based on the emergence of new techniques, and change between categories. In [12] dos Santos Junior et al. present their work training a PAC-MAN reinforcement learning agent specifically in the context of speedrunning. Most academic scholarship relating to speedrunning is not in technical computer science but in sociological study of the field and its culture [13].

3. Challenges in AI for Speedrunning

Speedrunning is a vibrant community of people that has existed for decades, composed of huge numbers of players playing similarly huge numbers of games. It is important for us as AI researchers to engage respectfully with this space, and to focus on clear communication about our goals, both to establish what is interesting about this space for researchers, and to clarify to the existing community what we hope to bring to the space. We discuss engagement with the speedrunning community as a point of future work as well.

We propose that a core goal for work in this space is the implementation of game-playing agents that can perform speedrunning tasks such as running games (seen or unseen) and glitch discovery. This is distinct from traditional game-playing AI agents, and brings with it its own unique challenges and

research problems. Aside from contributions to speedrunning, these problems and challenges also offer a platform to push AI research further too, just as game-playing AI competitions have in the past.

3.1. Inputs and Hardware

Speedruns are typically run on specific versions of original game software. These are sometimes emulated, but they are always based on real versions of the game released by its original developer. In contrast, game-playing agents are usually developed in a specialised research environment (such as machine learning 'gyms' [14]) or with an API that allows direct access to both inputs and outputs of the game (such as the MicroRTS [15]) challenge, which allows an agent to query precise information about the game state).

Game-playing agents are also usually provided with an *action space* which defines all possible actions within the game, from which they can select a move at a particular timestep. These action sets may be simplified to focus the search towards legal moves, which might block an agent from either discovering or using glitches in their solution. For example, in speedruns for *Resident Evil 7*, the graphics settings menu must be accessed to change the Frames Per Second cap multiple times during a run. To our knowledge, no game-playing AI agent has ever had access to game settings as part of its action set.

The rate at which an AI agent updates is usually chosen based on a reasonable reaction window to avoid the agent being polled for decisions at too high a rate, which would slow down performance. However, many speedrunning glitches require rapid sequences of inputs or require inputs to be sent at a specific frame in the game's execution (so-called 'frame-perfect' tricks). Super Mario Bros. runs at 59.97 frames per second, but the AI Mario Competition polls the agent 25 times per second - meaning the agent would not be able to perform a frame-perfect trick in the original game.

3.2. Glitch Discovery

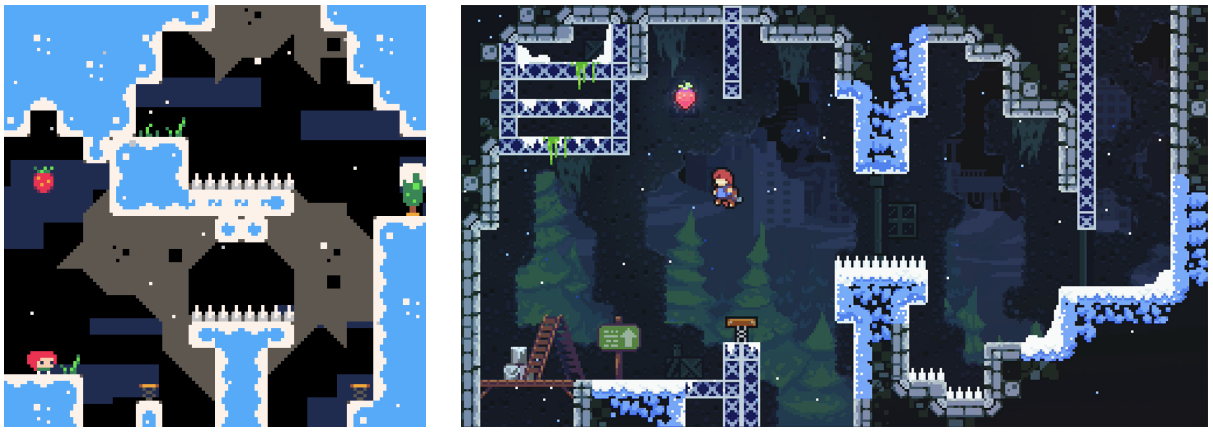
Using *known* glitches presents one set of problems for researchers, but discovering *new* glitches is its own challenge. This has an overlap with tools for quality assurance and testing for games, which has been tackled by both AI researchers [16] and game developers [17]. One key difference between glitch hunting for speedruns and for game testing is that testing is mostly concerned with errors that may affect the playability or experience of the player. However, glitches that are useful to speedrunners may be entirely benign or unimportant from a quality assurance perspective.

Another key challenge in glitch discovery is the gap between *identifying* a glitch, and finding a *use* for it. Glitches are often discovered accidentally, and their utility is not always obvious to speedrunners. Glitches can also be discovered but considered too difficult to replicate or perform. For example, many tricks used by speedrunners of *Super Mario Bros.* were considered to be 'TAS-only' upon their discovery; that is, they required such precise inputs that they were only feasible for an automated tool-assisted speedrun. While this is still true of some tricks, others have become possible either through the increased skill ceiling of speedrunners, or by alternative methods being discovered for triggering them [18].

Thus, glitch discovery is a thorny problem for AI researchers: it relies on writing AI agents that perform in ways that run counter to most existing game-playing AI; it requires deep exploration to decide whether a glitch is useful (or even *potentially* useful); and it involves player/runner modelling to determine if it is feasible for humans to perform. Of course, an AI system does not need to perform all of these tasks independently to be useful – AI techniques for glitch discovery may act as tools for existing human glitch hunters to use.

4. Case Study: Instrumenting PICO-8

In the development of this paper, we prototyped three agent-based setups using the PICO-8 game engine as a target, to begin to probe some of these questions and identify the challenges present in developing AI tooling for speedrunning. PICO-8 is a 'fantasy console', meaning it has been designed as a game engine with artificial limitations that evoke an earlier era of game development. These limitations



(a) *Celeste Classic.* (b) *Celeste.*

Figure 4: Celeste Classic (left) was developed into Celeste (right) in 2018, which also has a very large speedrunning community - its Any% category has 6,233 verified runs at the time of writing.

include a fixed palette size, a maximum length for the game's code, and a fixed memory for sprites, game maps, music, and sound.

We chose PICO-8 as the target for our case study for several reasons. First, all games written with PICO-8 are open source, which gave us a lot of options for exploring approaches to instrumentation, and reading input and output from the games. PICO-8 is a great platform for AI researchers to target in general for this reason. Second, PICO-8 includes games with a significant speedrunning community, particularly *Celeste*, now known as *Celeste Classic*, a 2015 platformer made by Maddy Thorson and Noel Berry [1]. Celeste Classic's Any% category on Speedrun.com has over 750 verified runs at the time of writing, and has spawned a lot of custom content related to speedrunning the game, including an interactive PICO-8 tutorial game called *Celeste Tech Training* (CTT) designed to teach players important speedrunning techniques. There are also games designed for PICO-8 by speedrunners, specifically with speed challenges in mind [19].

To better understand some of the challenges and research questions involved in working with speedrunning techniques and games, we explored three different approaches for implementing an AI agent framework capable of interfacing with PICO-8 games. Our motivating challenge was to build an agent capable of performing a *spike jump* in Celeste Tech Training. A spike jump is a technique used in all versions of Celeste that takes advantage of the fact that spikes only kill the player if they are moving towards them. The player can jump 'through' spikes from below, allowing them to touch the floor, refresh their air-dash capabilities, and then jump again into the air. Figure 1 shows three stages of a spike jump in Celeste Tech Training.

4.1. Instrumenting PICO-8

One reason that leads researchers to use specially-made environments for AI research is that most games, even open-sourced games, are not designed for supporting AI agents. For example, many search-based AI algorithms require a forward model, or some way of rapidly resetting the game state. We began first by exploring whether such an implementation would be possible within a PICO-8 game itself. Writing an agent within the same project as the game it targets allows for speedrunning agents or related tech to be implemented and transferred around more easily, and allows more direct access to game features.

We modified a version of Celeste Tech Training (CTT) with the features needed to implement a search-based AI player, to see how feasible this is to do within a PICO-8 game. PICO-8's size restrictions means that any new code we add is competing for space with the game code itself. For some games, including Celeste, developers have pushed the limits of the platform's size restrictions so much that almost no space remains to add our custom code in. PICO-8 does support combining multiple 'carts'

together (a 'cart' or cartridge is a self-contained program with its own spriteset, map, code, and so on) however carts remain self-contained and so we cannot load code from an AI agent cart while still executing code on the main game cart. For this reason, any custom API for an AI player would need to be as minimal as possible.

We first added in a virtual controller which bypassed the normal inputs for PICO-8. This was relatively simple – the cart maintains a map of PICO-8's six buttons, and we implement functions for pressing, holding, and releasing buttons. During the update loop, these virtual button presses are converted into real button presses in the game's logic. This allowed us to hardcode a spike jump using virtual inputs, shown in Figure 1.

We attempted to add in save states, which are useful for rapidly resetting the game during search. PICO-8 has no native support for save states, although its memory is entirely accessible and editable. PICO-8 does have support for copying large chunks of memory out to other locations – in particular, we can use a command called `cstore()` to save the contents of the game's memory to another cartridge file, or `memcpy()` to store segments of a game's memory into unused higher memory addresses. However, we cannot access the Lua program stack or other kinds of memory data relating to the execution of PICO-8 itself, which means although we can store a lot of information about the game state, we are unable to inject the state back into the game from within PICO-8. This makes full memory save states infeasible from within PICO-8, as far as we understand; although many other platforms exist where it may be feasible to write a speedrunning agent within the game itself, especially older platforms for which strong emulation exists [20].

4.2. Glitch Replication with Celia

If we cannot reliably implement AI agents within a PICO-8 game, then we will need an external tool to interact with and change the PICO-8 game state directly. Celia is a tool built in Lua for creating TAS runs of PICO-8 games, designed by PICO-8 user gonengazit[11], which includes a number of useful features including automatically sending inputs from a TAS file. This allowed us to write an external script in Lua to generate and execute TAS files, using a modified version of CTT which could be instantly reset to the spike jump test level. After resetting the cartridge, our system would use Celia to play a randomly-generated TAS, then evaluate the result using an objective function. To simplify the search, our TAS generator only added inputs on every fifth frame, although for a full-scale system it would be simple to expand this to every frame.

This approach used purely random generation for the TAS files as a proof of concept, but it was able to successfully rediscover the spike jump mechanic after a short series of runs. Writing an AI agent in Lua using Celia has the advantage of sharing a platform/language with PICO-8, which means that while it isn't as self-contained as an agent written into the cartridge itself, it is nonetheless quite portable and uses only a single language. However, constraining ourselves to Lua means we miss out on the opportunity to support a wider range of agents and users, and limit access to certain tools and libraries.

It is also worth noting that rediscovering an existing glitch is not the same as glitch *hunting*. Suppose we suspected that there were other ways to cross a spike pit in 1. In order to force a search-based agent to discover these, we would need to forbid or discourage existing techniques like the spike jump. It is hard to specifically exclude a technique as it may be possible to perform the technique in many ways, with different timings, or with other junk inputs mixed in. One way to search for new glitches might simply be to change the objectives. For example, if our objective function incorporates time, we might only be interested in TAS files that reach the end faster than the spike jump TAS does. Another would be to design situations that are not known to be beatable, an approach used in automated game design research [21].

[11] https://www.lexaloffle.com/bbs/?uid=70985

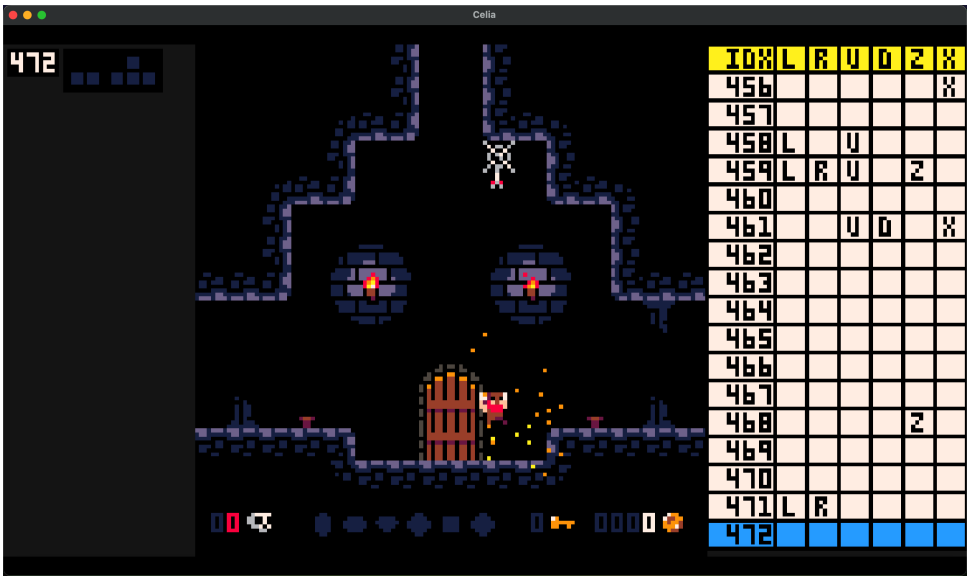


Figure 5: Screenshot taken from the speedrunning tool Celia [22] displaying random keypress inputs for the game *GET OUT of this Dungeon*.

4.3. Python Interfaces for General Play

Our third system took the usage of Celia and built it into a fully external system, written in Python. Python is a very popular language, particularly with AI researchers, and so this is also a good approach for building a widely-accessible framework for running events such as competitions. Our goal here was to investigate how a completely external program could use Celia to perform a search within a PICO-8 game.

Celia supports advancing and rewinding by a single frame, which we can use to more fully search the state space in a more granular way. This allowed us to build a system with a forward simulation of the game which precisely executes the game code and rolls it back using Celia's frame skip. This enables a greedy search of the state space to look for inputs that make progress in the game, find new states, or avoid death. More importantly, because this system manipulates individual frames rather than loading a full TAS, it does not need a reset point to be added in as it does not reset the cartridge repeatedly to test new routes. This makes it a lot more efficient and closer to existing AI frameworks that can test an action and explore different parts of the state space. Celia can also output screenshots of the game state, which could be used for vision-based evaluations as well. Our implementation did not use this; instead, it was searching based on novelty. The source code can be found here: https://anonymous.4open.science/r/pico-speedrunning/README.md

This approach was tested on three separate, open-sourced PICO-8 games retrieved from the PICO-8 BBS, the 'bulletin board system' that serves as the platform's community and game-sharing network[12]: *GET OUT of this Dungeon*[13], *treeboi_test*[14], and *Witch Loves Bullets*[15]. With all three games, randomly made TAS files were successfully generated, loaded, and played in the Celia software. The original code of each game was not modified – thus, this system can allow for speedrunning, glitch discovery, and agent replay on any PICO-8 game.

Two out of three of the games tested currently have an extremely small community of speedrunners on the Speedrun.com leaderboards. The third game, treeboi_test, was the most recently made game and chosen at random from the BBS page. For future work, we would like to develop an artificial agent to search for novel game states via screenshots using keypresses simulated with Celia. Ideally, this

[12]https://www.lexaloffle.com/bbs/?cat=7
[13]https://www.lexaloffle.com/bbs/?tid=31538
[14]https://www.lexaloffle.com/bbs/?tid=142653
[15]https://www.lexaloffle.com/bbs/?tid=35109

system can identify new tricks for speedrunning or optimize agent movements by discovering glitches to reach quicker game state times. The completion times of the agent can be evaluated against existing leaderboard times or even pave the way for a new speedrunning community for a PICO-8 game.

5. Discussion and Future Work

5.1. Responsible Research

The authors of this paper include speedrunners, game developers and fans of speedrunning, and this paper represents our attempt to earnestly engage with an area of games culture that we believe is full of interesting potential and fun challenges, and that means a lot to us personally. However, particularly in the current climate, it is important for us as AI researchers to engage respectfully and openly with communities we wish to work alongside. Therefore, it is an important point of future work for all of us interested in this space to build connections and dialogue with communities of speedrunners and understand what interests them, what worries them, and how we can conduct our research responsibly. This is particularly relevant given that 'AI' is a charged term currently, and so being clear that research in the area does not need to involve LLMs and can be done locally, with no data, and little energy use, for example, may be crucial in initial dialogues.

In particular, the initial response to the concept of 'AI for Speedrunning' might be one of concern, as it is seeking to automate something that is fundamentally about the joy of human skill and performance. However, we see AI's contribution to this area as being closer to TAS or 'R&D' (what some speedrunners call people who look for glitches). This is a way to support the curiosity of the speedrunning community with new tools and techniques that they can use and modify themselves. In the same way that speedrun randomisers are closely aligned with the goals of the procedural generation community, we hope that the common ground between game-playing AI research and speedrunning will yield exciting new ways to explore, break, and possibly even play, games of all kinds.

5.2. Routing

One topic not touched in our case studies is the challenge of *routing* a speedrun. Routing is the decision-making process concerned with which tasks a player should complete to finish a speedrun, how they should be completed, and in what order. Sometimes this is simply a question of ordering: for example the *Dark Souls* 'All Bosses' category requires that all bosses are defeated, so routing involves simply choosing which is the fastest order to fight them in[16]. However, routing may also involve deciding whether some tasks should be performed at all. For example, in the *Final Fantasy XII* Any% speedrun, the speedrunner must acquire a certain amount of *gil* (the game's currency) to spend on particular items during the run[17]. How they acquire this gil is completely open to the runner – they could fight monsters, do quests, or sell items. Here, routing is a task selection problem: given a set of possible tasks, what is the most efficient combination of tasks that achieves a particular goal (such as having an amount of money)?

Over time, the most optimal routes will be adopted as standard by speedrunners. However, these can change if new routes are tested and found to be faster, or – more commonly – something else changes in the speedrun that affects the rest of the route. For example, in the *Bioshock* speedrun, changes to the route were made after a skip was found that resulted in the player not acquiring a certain skill. Although the skip saved time overall, it required a new route and approach to certain levels to take into account the fact that the player no longer had the same skills to use.

Routing is a very complex problem in speedrunning, particularly for large games with many different tasks to perform and no fixed order they must be completed in. This problem maps naturally onto problems in computer science related to scheduling, planning, and search – many of which are already studied by game AI researchers, as we cited earlier in this paper. There remains a lot of unexplored space

[16] https://www.speedrun.com/darksouls
[17] https://www.speedrun.com/ff13

here in terms of identifying and formalising routing problems, solving them efficiently, and explaining and visualising solutions for users.

5.3. Competitions

Traditionally, AI competitions have been a powerful way to attract researchers to work collaboratively on a problem, pool resources and create open-source example work for education and further research [15, 23, 24]. We believe that speedrunning is a perfect subject for an AI competition, particularly one based around PICO-8 thanks to its easily-modifiable nature, and the progress we made with our exploratory work. The most important component needed for this would be real games to test on, with the consent and buy-in from the developers to allow us to use their source code.

A straightforward competition format would be an Any% speedrun of a game, although some kind of incentive or constraint would be needed to dissuade researchers from submitting an agent that simply runs the equivalent of a TAS (a randomiser would be one example of this, as this keeps most aspects of the game consistent but changes the specific challenge encountered). Alternatively, small perturbations could be introduced that are enough to derail pre-written TAS routes, but not so significant that they change the nature of the game. A final alternative, which would also be a positive way to engage with game development and speedrun communities, would be to center the competition around a base game such as Celeste Classic, but have the final competition test be on an unknown mod or variant of the game (many fangames have been made using Celeste Classic's mechanics, but with new levels). Commissioning developers directly to produce new levels would also be a nice way to combine our communities.

Glitch discovery is another potential competition format. This could work as a bounty-type format (discovering new glitches to bypass a certain problem or area in a game, with no specific competition phase) or a more traditional competition where a set number of known glitches exist and the agent is given an amount of time to find as many as they can. Since glitch hunting can benefit from generality of approach, it may not be necessary to specify the game in advance (perhaps the genre – such as platformer – or the basic controls). Glitch hunting is a counterintuitive challenge where, by definition, setting challenges based on existing glitches may not adequately test agents' ability to find entirely new classes of glitch that are not known to exist yet.

6. Conclusions

Speedrunning is a vibrant community of both research and practice, and has been in existence for longer than any videogame AI conference or journal. It is much more than simply playing games quickly - it's a space where unusual forms of play are celebrated and discovered, where game design is pushed beyond its breaking point, and where incredible interdisciplinary work takes place. AI researchers are yet to properly engage with this space, and it holds myriad exciting new challenges and problems for them to tackle, including a very different approach to game-playing than game AI has historically focused on.

In this paper, we discussed some of the basic properties of modern speedrunning and how this differentiates itself from most of the problems that game AI researchers have tackled in the past. We reported on our work investigating PICO-8 as a platform for speedrunning, and used this to highlight some future work that could potentially be carried out in this area, particularly in organising future events that researchers can focus on together, such as competitions. Above all, we want to ensure that researchers approach this community with an open mind and a respectful manner, and build relationships with the community who already know this space so well, and who have worked so hard to build it into what it is today. We hope this paper encourages others to take a look in this area, and join us in exploring the future potential of research in this area.

7. Declaration on Generative AI

No Generative AI tools were used in the preparation of this paper.

8. Acknowledgements

The authors wish to thank the speedrunning community who have influenced our thinking in so many different ways and led us to writing this paper. Thanks to the reviewers for their kind feedback and suggestions. This work began at *Dagstuhl Seminar 24261: Computational Creativity for Game Development*. Thanks to the seminar organisers and to Schloss Dagstuhl for facilitating this.

Filippo Carnovalini is funded by the European Union, HORIZON- MSCA-2022-PF-01 Project ID 101108690 (CALIOPE). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Executive Agency (REA). Neither the European Union nor the REA can be held responsible for them.

References

[1] M. S. Thorson, N. Berry, Celeste classic, 2015. URL: https://maddymakesgamesinc.itch.io/celesteclassic.
[2] Raptite, Celeste tech training, 2023. URL: https://www.lexaloffle.com/bbs/?tid=53386.
[3] E. Koziel, Speedrun Science: A Long Guide to Short Playthroughs, Fangamer, 2019.
[4] Naughty Dog, The Last of Us Part II, 2020.
[5] R. Mawhorter, P. A. Mawhorter, A. Smith, The randomizer community does procedural content generation research, in: Proceedings of the 17th International Conference on the Foundations of Digital Games, 2022, pp. 1–3.
[6] Act_, Speedrun.com terminology, 2022. URL: https://www.speedrun.com/forums/the_site/ll4tb, also archived on archive.org.
[7] TASVideos.org, https://tasvideos.org/, 2003.
[8] J. Togelius, S. Karakovskiy, R. Baumgarten, The 2009 mario ai competition, in: IEEE Congress on Evolutionary Computation, 2010, pp. 1–8.
[9] S. M. Lucas, Ms pac-man competition, SIGEVOlution 2 (2008).
[10] N. Froleyks, T. Balyo, Using an algorithm portfolio to solve sokoban, Proceedings of the International Symposium on Combinatorial Search 8 (2021) 165–166. URL: https://ojs.aaai.org/index.php/SOCS/article/view/18416.
[11] M. Groß, D. Zühlke, B. Naujoks, Automating speedrun routing: Overview and vision, in: Applications of Evolutionary Computation, 2022, p. 471–486.
[12] E. S. dos Santos Junior, A. d. O. Machado, M. C. F. Macedo, A. C. dos Santos Souza, Reinforcement learning para treino do pac-man em speedrun / reinforcement learning for pac-man speedrun training, Brazilian Journal of Development 5 (2019). URL: https://ojs.brazilianjournals.com.br/ojs/index.php/BRJD/article/view/4755. doi:10.34117/bjdv5n11-242.
[13] D. Ford, Speedrunning: Transgressive play in digital space, in: Proceedings of the 2018 DiGRA International Conference, July, 2018, pp. 25–28.
[14] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, O. G. Younis, Gymnasium: A standard interface for reinforcement learning environments, 2024. URL: https://arxiv.org/abs/2407.17032. arXiv:2407.17032.
[15] S. Ontanón, The combinatorial multi-armed bandit problem and its application to real-time strategy games, in: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, volume 9, 2013, pp. 58–64.
[16] S. Roohi, C. Guckelsberger, A. Relas, H. Heiskanen, J. Takatalo, P. Hämäläinen, Predicting game

difficulty and engagement using ai players, Proceedings of the ACM on Human-Computer Interaction 5 (2021).
[17] C. Muratori, Walk monster, 2012. URL: https://caseymuratori.com/blog_0005.
[18] H. Kemps, Super mario bros. speedrun record shattered by a matter of milliseconds, 2016. URL: https://www.vice.com/en/article/super-mario-speedrun-record-shattered-by-milliseconds/.
[19] rarelikeaunicorn, Astra and the new constellation, 2023. URL: https://rarelikeaunicorn.itch.io/astra.
[20] J. Franušić, K. Tuite, A. Smith, Playable quotes for game boy games, in: Proceedings of the 18th International Conference on the Foundations of Digital Games, 2023.
[21] M. Cook, S. Colton, A. Raad, J. Gow, Mechanic miner: reflection-driven game mechanic discovery and level design, in: Proceedings of the 16th European Conference on Applications of Evolutionary Computation, 2013.
[22] gonegazit, Celia - a PICO-8 TAS framework, 2022. URL: https://github.com/gonengazit/celia.
[23] C. Salge, M. C. Green, R. Canaan, J. Togelius, Generative design in minecraft (gdmc): settlement generation competition, in: Proceedings of the 13th International Conference on the Foundations of Digital Games, 2018.
[24] M. Čertický, D. Churchill, K.-J. Kim, M. Čertický, R. Kelly, Starcraft ai competitions, bots, and tournament manager software, IEEE Transactions on Games 11 (2019).