Crawling, Indexing, and Retrieving Moments in Videogames

Xiaoxuan Zhang
Department of Computational Media
University of California, Santa Cruz
xzhan209@ucsc.edu

Zeping Zhan
Department of Computational Media
University of California, Santa Cruz
zzha50@ucsc.edu

Misha Holtz
Department of Computer Science
University of California, Santa Cruz
mnholtz@ucsc.edu

Adam M. Smith
Department of Computer Science
University of California, Santa Cruz
amsmith@ucsc.edu

ABSTRACT

We introduce the problem of content-based retrieval for moments in videogames. This new area for artificial intelligence in games exercises automated gameplay and visual understanding while making connections to information retrieval. We propose a number of techniques to discover the interesting moments in a game (crawling), show how to compress moments into an efficiently searchable structure (indexing), and recall those moments most relevant to a user-provided query (retrieving). We combine these ideas in a prototype visual search engine and compare it with commercial visual search engines. Searching within a corpus of moments from Super Nintendo Entertainment System games using query images extracted from YouTube videos, our prototype is able to identify moments that web-oriented search engines rarely see.

CCS CONCEPTS

• **Information systems** → *Information retrieval*; • **Computing methodologies** → *Artificial intelligence*; • **Applied computing** → *Computer games*;

KEYWORDS

Visual Search, Deep Learning, Relevance Feedback, Super Nintendo Entertainment System, Speedrunning

ACM Reference Format:
Xiaoxuan Zhang, Zeping Zhan, Misha Holtz, and Adam M. Smith. 2018. Crawling, Indexing, and Retrieving Moments in Videogames. In *Foundations of Digital Games 2018 (FDG18), August 7–10, 2018, Malmö, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3235765.3235786

1 INTRODUCTION

Artificial intelligence (AI) and information retrieval (IR) have a long and entangled past [8, 19]. AI powers multiple facets of commercial web search engines like Google, Baidu, and Yandex. Although these services are primarily designed to retrieve hypertext documents based on textual queries, they are increasingly growing into the domains of visual search (using images as queries). Visual search can involve sophisticated automated understanding of image and video content. However, these search engines have no direct ability to understand the content of interactive media such as videogames. Modern web crawlers, for example, would not discover and crawl a webpage that is only mentioned in a credits screen only visible after playing through a videogame.

We envision a new class of search engines operating in the domain of content-based retrieval for moments in videogames. This is a new area for AI in games, and it expands technical games research by making connections to IR. These search engines could be used by game scholars to find moments in games that support their arguments. They could augment tools such as the Game and Interactive Software Scholarship Toolkit (GISST) [9] by allowing scholars to locate moments in a game they do not have the time or skill to reach via interacting directly with a game's control inputs. At the same time, this new class of search engines could be used by other AI systems that aim to develop expert gameplay strategies, map out the reachable spaces in a design (e.g. Mappy [16]), or enable new tools to provide feedback in the design process [15]. Speedrunners, designers, and educators could find, create, and share bookmarks in a game's state space that could be used to improve the precision of their technical communication.

This paper takes a first step in realizing this vision by showing how to crawl, index, and retrieve moments in a collection of over 750 videogames for the Super Nintendo Entertainment System (SNES). We show how algorithms can be combined with public data sources to uncover the contents of a game: moments in interactive play. We show how existing techniques for visual search can be adapted for use on videogames, capturing them as discrete state machines deterministically advancing on input from players. We show how to match indexed moments with user-provided queries, judge their relevance, and compute search quality metrics for moment retrieval systems. Finally, we describe a prototype visual search engine capable of interactively retrieving moments from SNES games given game screenshots from the wild as input, and compare the results with commercial visual search engines like Google Image Search and TinEye.

2 BACKGROUND

To our knowledge, there is not yet any academic work on *content-based* retrieval for videogames or other inherently interactive media.

FDG18, August 7–10, 2018, Malmö, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6571-0/18/08…$15.00
https://doi.org/10.1145/3235765.3235786

As such, we need to import ideas, vocabulary, and technical methods from disparate fields.

2.1 Web Search

Web search is the canonical large-scale application of IR ideas [5, Chap. 1]. In web search, it is typical for the user to provide their query in the form of a brief text phrase and expect results sorted from a constantly growing collection of on the order of trillions of webpages [10].

To automatically ingest information from the Web, major search engines deploy crawlers (agents) that systematically explore and download partial copies of the Web graph. In this graph, nodes are individual pages, and edges represent hyperlinks from one page to another. Because the users of a search engine are intelligent explorers themselves, it is not critical that absolutely every node is visited by the crawler. So long as a large enough fraction of the the most important nodes are visited, users may answer their queries with only a little more browsing. In the videogame domain, we would be happy to crawl just enough of a game's vast state space in order to provide useful results for user queries.

The datasets resulting from crawling processes are immense. To answer a user's query, it is not feasible to scan through the downloaded copy of the Web to harvest relevant results. Instead, IR systems employ an indexing process to produce a compact and efficiently traversable representation of the crawl results. Given a query, an index structure helps to quickly identify the very few results that the user might see on the results page. Preview snippets for only this tiny result set can be computed by consulting the full contents of the matched result documents. In the videogame domain, we associate the content of a moment with the full state snapshot of a game platform emulator in between frames of animation. In IR terms, these moments are the documents, and the index should give us enough clues to reconstruct the full game state data on demand.

The IR literature offers many methods for retrieving relevant documents from a collection (called a corpus) given a query. The vector space retrieval model [13, Chap. 6] (originally a generalization of keyword counts) offers an intuitive setting for retrieval. Here, an embedding function maps a document (or a query) to a point in space. Good embeddings will place similar documents closer together in space and unrelated documents further apart. The estimated relevance of a document to a query can then be approximated by a distance calculation. Retrieval in this model reduces to a kind of nearest-neighbor lookup. In sophisticated web search engine designs, multiple layers of index-accelerated matching, filtering, ranking, and re-ranking systems are applied to compute a manageably small result set for the user to browse.

2.2 Book Scanning

In 2002, Google initiated "Project Ocean" to scan every book in the world with the intent to make searchable a vast collection of content not already represented on the Web [20]. At the time, digital library services already offered book search based on traditional IR methods, but the results of a search worked at the level of whole books. A user might find that one book is likely relevant to their query phrase based on previously recorded metadata about that book. However, in order to find the specific location in that book (at the level of a page number), one would need to consult the index at the end of the physical book, if it even included one. Scanning the world's books promised the benefits of content-based search (which drove the previous Web search revolution) for pre-Web and off-Web documents.

The way we search for videogames now is analogous to this situation with books. We can search for game titles in Steam[1] by term or tag, but are left hopeless if we want search within a given game. For sufficiently popular games (as with sufficiently popular books), the clues can be found in content of blogs or news articles when they are indexed by major web search engines. Perhaps we are lucky enough to find a walkthrough page with textual instructions (or gameplay video with an informative text transcript) for how to reach an interesting point in a game, but this is not enough. Until 2017, we did not even have a notion equivalent to page numbers, a way to cite a specific moment in a game in one of our papers [9].

Discovering the content of a physical book requires physical effort to turn and align pages. In the early stages of the book scanning project, this was done entirely by human experts, but now such efforts are assisted by robotics.[2] We envision a similar trajectory towards automation to effectively scan the contents of the vast spaces of play for videogames. Although it is easy to understand when a book has been scanned to completion (after each page is finished), we need some other form of measurement in the domain of videogames to judge the fraction of interesting content covered by a given crawling procedure.

2.3 Audio and Visual Search

Most commercial Web search engines have simple homepages featuring a large input box which invites the user to enter their search query in text. Increasingly, these same services are allowing users to search using images as the query: by dragging an existing image onto the browser or capturing a new photo with a mobile device.

Audio search engines like SoundHound[3] and Shazam[4] allow users to identify the music playing around them by recording a few seconds of audio. Signal processing techniques are used to compute a fingerprint of the sound that can be matched against an indexed database of known musical works [4]. We would like to recreate this experience for videogames. Because a videogame might contain many hours of experientially significant gameplay compared to the few minutes of a pop song, it is important that the search results for videogames return specific moments (analogous to specific book pages) rather than just game titles.

Early-generation image search engines, such as the initial release of Google Image Search,[5] allowed users to search for images using textual queries, fitting into the existing paradigm of web search. These systems used text features from webpages that embedded the images as metadata about the images rather than analyzing the pixels in the images themselves. The "reverse image search" engine TinEye[6] popularized the interaction of searching by an image. This

[1]http://store.steampowered.com/search/
[2]https://youtu.be/RdLcrNeWjIs?t=141
[3]https://soundhound.com/soundhound
[4]https://www.shazam.com/
[5]https://images.google.com/
[6]https://www.tineye.com/

method of image searching allows users to find the original webpage context of images they have found posted elsewhere. One use of this interaction model is helping readers identify hoaxes by uncovering when eye-catching photos are misused out of their original context.[7] We envision image-based investigations being a frequent use case for game scholars interacting with a videogame moment search engine.

Current-generation visual search engines such as GrokStyle[8] apply deep analysis to image pixel data to recognize objects (such as furniture items from a catalog) even when they are re-photographed in different viewpoints, lighting, and surrounding contexts. Rather than engineering such systems from scratch, deep learning techniques are used to train a neural network that can map raw images into an embedding space of moderately high-dimensionality (hundreds of dimensions rather than the much higher dimensionality of raw image pixels). Embedding models are optimized to place different images of the same object nearby and place images of similar-looking but different objects at least some distance apart. Additionally, proxy prediction tasks (such as correctly guessing the category of an object from just its location in an embedding space) can provide additional guidance to these learning systems when related data is available [2]. In the videogame domain, we train screenshot embedding networks on a proxy task: reconstructing the contents of game platform memory from the embedding vector. More details are given in Section 4.1.

2.4 Playing Games Automatically

Several general purpose algorithms are known which can play some videogames surprisingly well. Even more surprising, these algorithms are often able to achieve super-human playing performance based on analysis of raw screenshot pixels alone (rather than carefully designed game state abstractions). Deep reinforcement learning (RL), in particular, has shown excellent results on a variety of videogames for the Atari 2600 platform [14].

RL requires that an environment, such as a videogame, provide some kind of reward signal. This reward signal is typically derived from the game's score, which is itself derived from inspection of the game's memory state. As with Web crawling, crawling the space of play for videogames requires the ability to usefully explore in the absence of a predefined reward signal. For example, we want to crawl a game's credits screen and options menus despite there being no score benefit for doing so. Work on intrinsically-motivated RL suggests that agents generate their own reward signals, often based on models of novelty and curiosity. Interestingly, augmenting existing deep RL methods with an exploration bonus has allowed them to achieve better score performance than algorithms that try to optimize the score directly [3].

Seen this way, efficiently exploring the space of play for classic videogames might be seen as just a niche technique for optimizing performance in a testbed for AI algorithms. By contrast, we see these algorithms potential starting points for automated crawling procedures that would be run to harvest interesting moments from culturally impactful interactive media.

[7]https://www.youtube.com/watch?v=lwHkIrGhhFg
[8]https://www.grokstyle.com/

3 CRAWLING

Before launching into our own kind of book scanning project, we need to answer some more basic questions. Which game platform should we start with? From where will we get the games? How will we explore those games (how will we turn the pages or prioritize a crawl queue)?

3.1 SNES Platform

Our project is initially focused on the Super Nintendo Entertainment System (SNES). This platform is known as a 16-bit home videogame console. We consider this platform a useful midpoint between the 8-bit Atari 2600 platform used in the latest deep reinforcement learning research and the 32-bit Android mobile platform. While the processor executing Atari games has access to just 128 bytes of working memory to define game state, the SNES has 128 kilobytes, and modern Android apps use between 16 and 128 megabytes of memory.

We are also attracted to the diversity of games produced for the SNES platform. The launch title *Super Mario World* features linear platformer gameplay that is well exercised in AI competitions [21]. Beyond this, there is interaction driven by textual dialogs and menus in *Chrono Trigger*. There are three-dimensional graphics and collision logic in *Star Fox*. There is open-ended gameplay in *Sim City*. There is even mouse-driven interaction with audio-visual authoring tools in the arguably not-game *Mario Paint*.

Rather than work with the physical hardware of the SNES platform, we utilized software emulators. In particular, we use a version of the BizHawk emulator (building on the cycle-accurate BSNES[9] emulator core) created by the tool-assisted speedrun community.[10] This allows us to step through the moments in a game, frame by frame (at nominally 60 frames per second), algorithmically selecting which combination of the twelve buttons to press (4096 possible actions). From BizHawk, we can extract color screenshots (at a resolution of 224 by 256 pixels) and platform state snapshots (approximately 3 megabytes in size) which capture the contents of (CPU accessible) working memory, video memory, and other subsystem state. By replaying the same inputs over time from the canonical boot state (when the platform starts with initially empty memory), we can reproduce a past state exactly. Unfortunately, platform snapshots are not portable between different emulator implementations nor are emulators perfectly faithful to the hardware implementation of the SNES platform.

3.2 Game Sources

Our project utilizes a collection of over 750 games derived from multiple sources. Many commercial SNES games are available in public archives.[11] Interestingly, these archives seem to have better coverage for non-SNES game platforms. Homebrew SNES games or fan-made modifications of commercial games can be found in ROM hacking communities.[12] New games for the SNES platform are still being released on these sites as recently as 2017.

[9]http://www.bannister.org/software/bsnes.htm
[10]http://tasvideos.org/Bizhawk/SNES.html
[11]https://archive.org/details/Snes_Roms
[12]http://www.romhacking.net/

Unfortunately, not all games are compatible with all emulators, so our integration with this version of the BizHawk emulator does not give us full coverage for the SNES platform. On the other hand, BizHawk can emulate many different game platforms ranging from the Atari 2600 to the Nintendo 64 (based on a 32-bit processor connected to approximately 4 megabytes of working memory). As a result, our approach puts us within reach of platforms similar to those targeted by the latest Android games.

3.3 Exploration Methods

Crawling a game requires a strategy for how to proceed from one moment to the next. Here, we consider some basic human, machine, and hybrid approaches. Each run of a method produces a collection of moments that we call a corpus.

3.3.1 Speedruns. Sites like TASvideos[13] offer up hundreds of input sequence recordings (called “movies”) for expert gameplay of SNES games. Some of these exploit glitches and other atypical gameplay techniques to reach a game’s end as fast as humanly (or computer-assisted humanly) possible. Others offer a clean (no mistakes, but also no special tricks) run through “100%” of a game’s content. The intersection of games with many kinds of speedruns and speedruns compatible with BizHawk is not as large as we originally hoped.

The few compatible “100%” speedruns on this site offer a very valuable external definition of what it means to have seen *all* of a game’s content. A one-hour movie (even at 60 frames per second), represents a very small fraction of a game’s truly reachable state space. Nevertheless, it is one representation of what a dedicated community believes the game has to offer. These speedruns will anchor our future attempts to quantify the effectiveness of automated crawling algorithms. Already, they have allowed us to bootstrap our indexing and retrieval systems in the absence of a robust crawling process.

3.3.2 Speedrun Branches. With only a small amount of code, we can begin to explore states off the main path indicated by the “100%” speedruns. Consider a speedrun of n moments cut into k equally-sized spans. One way to uncover new and potentially useful moments is to seek to the start of span i and begin playing back the original speedrun’s input sequence for a random span j. In a fighting game for which the player must execute special combination moves with precise timing, this exploration strategy will occasionally try those special combinations in new contexts.

We invented this strategy as a simplistic method to get states with a similar distribution throughout a game like *Super Metroid* without having any of them be byte-for-byte identical to those in the original speedrun.

3.3.3 15-minute Student Sessions. For many games, getting past the main menu and through the first introductory cutscene can be difficult for our automatic exploration algorithms. Game screens that ask the player to enter their name using an on-screen keyboard can be particularly challenging.

To get wider coverage than speedruns and to hopefully get to the first moment of core gameplay, we enlisted students in playing one game after another for 15 minutes at a time, recording their inputs in a standard BizHawk file format that we could process similar to a speedrun movie later. We have collected these human gameplay sessions for over 100 games so far.

In the future, we hope to use transfer learning techniques to port a human-like mode of exploration (demonstrated in these sessions) to games no one on our team has yet tried. If nothing else, it should be easy to use the distribution of input configurations (which buttons were pressed in which combinations for how long) to give a more human-like bias to automated exploration algorithms.

3.3.4 Attract Mode Animations. With 650 games remaining untouched by the methods above, we sought a very simple strategy that could be quickly applied to the remainder of the games. Our next strategy leans on the fact that many SNES games, inspired by previously successful arcade games, feature an attract mode. If the game is left at the main screen with no input for sufficient time, many games go into a self-demonstration mode that features snippets of actual gameplay. While nothing is happening with the control state during this time (each frame we submit to BizHawk that no buttons are pressed), the memory states are still moving through patterns similar to those that would be seen during live human gameplay. Indeed, many attract mode animations are simply playing back previous recordings of human play by the game’s original developers.

Not all of our attract mode datasets are interesting, however. Some games simply sit at the main screen forever. Others show story teaser cinematics without revealing gameplay. The existence of these teaser cinematics presents an interesting challenge for automatic exploration algorithms: to reach them, the algorithm must account for explicitly pressing no buttons for many thousands of frames in a row with no visual (and very little in-memory) feedback that progress is being made towards the new scene.

3.3.5 Rapidly Exploring Random Trees. Can we do better than pressing no buttons at all? The rapidly exploring random trees (RRT) algorithm [11] provides one conceptually simple solution. During an execution of the RRT algorithm, the algorithm repeatedly samples a random goal location (a point in some space yet to be defined), selects the previously-achieved state with a spatial projection closest to that goal location, and then attempts to make progress towards to goal by applying a primitive action in selected state. An edge in the tree is created from the selected state to the state just reached. Bauer and Popović introduced this technique to the technical games research community as part of a level design feedback tool [1].

When developing and debugging RRT algorithms, it is desirable for the goal space to be a low dimensional (easy to visualize) space. In initial experiments with *Super Metroid*, we devised a projection of game states onto a fan-created 2D map of the first area within the game[14] by using expert knowledge of the memory layout for this game.[15] Unsatisfied with how often the player character got stuck on the right side of the screen and did not finish jumping moves (a reasonable outcome for starting in the top-left of the goal space), we investigated alternate vector space representations. We

[13]http://tasvideos.org/Movies-SNES.html

[14]http://www.vgmaps.com/Atlas/SuperNES/SuperMetroid-SpaceColony.png

[15]https://drewseph.zophar.net/Kejardon/RAMMap.txt

got more promising results using the 256-dimensional embeddings derived from speedrun data discussed in Section 4.1.

Initially, we expanded the selected node in the tree by applying one of the 4096 control input configurations selected uniformly. Although this offered a chance to try every possible configuration, the actions selected were not relevant for making progress towards the goal (nor, in most cases, relevant to the game at all). In response, we trained a neural network to guess the relevant action to make progress by comparing the 256-dimensional embedding of the current state and the goal state. The labels for the supervised learning setup model for this network were taken from the expert speedrun. We operated under the assumption that the speedrunner was continuously demonstrating how to get from one state to whatever state they would be in a few seconds later. This at least got the exploration to focus on only using the button configurations with a distribution similar to that found in the speedrun data.

3.3.6 RRT Branches. We have considered but not yet applied the strategy of initializing the RRT algorithm with the moment tree from an expert speedrun, speedrun branches, or the 15-minute student sessions. We hope that this will bypass the disappointingly slow start of vanilla RRT as well as provide more useful branches than our naive uniform branching strategy that plays back inputs from the wrong context.

4 INDEXING

Having temporarily stored the full platform state snapshots (about 3 megabytes each) during the crawling process, we now need to produce a compact index structure which will enable efficient retrieval without the need to re-scan the bulk dataset.

4.1 Vector Space Embeddings

Operating in the vector space retrieval model [13, Chap. 6], we are inspired by systems such as ProductNet [2] that map detailed images to vectors using deep neural networks. In previous work, we considered a number of different vector representations of screenshot images and memory state, some using techniques as simple as principal components analysis (PCA) to perform the required dimensionality reduction [22]. In all cases, we were interested in vector representations of only moderately high dimensionality (usually 256).

Training deep neural networks for embedding images to vectors requires some indirection, as we do not have a dataset of the ideal vector representations. Our first strategy considers setting up a supervised learning task in which good prediction performance will be considered a proxy for good retrieval performance. In particular, we ask that a relatively simple neural network be able to predict the contents of the first four kilobytes of memory for a given moment given only the embedding of screenshot pixels as input. This very simple strategy yields surprisingly good retrieval results [22].

Fig. 1 describes the architecture of the neural network we used in the pixels-to-memory proxy task. Embedding vectors are associated with the bottleneck layer in this network. Despite being trained only as an intermediate representation on the proxy task, this moment vector representation manifests peculiar properties usually associated with learned word vector representation in natural language processing. In particular, the manifest support for reasoning by analogy. Fig. 2 shows two sets of four images (varying in main character power-up state and location within a level). We selected these moments to represent the analogy that A is to B as C is to D. Starting with the vector for moment C, we can add a scaled difference of vectors for B and A to get a vector $Q = C + \alpha(B - A)$. In both instances, Q is more similar (by the cosine similarity metric we use for retrieval) to D than it is to the base image C or the others. A visual search engine user seeking moment D could search by vector-algebra analogy with screenshots A, B, and C. The parameter α controls the strength of the influence of the distinction between B and A.

We also consider manifold learning techniques that attempt to learn embedding models that smoothly map images of adjacent points in gameplay time to nearby points in space. Using a triplet loss model, we simultaneously apply the same embedding model to three images Q, A, and B (Q representing a query image while A and B represent potential retrieval results). We add a penalty term to the learning problem's optimization so that the cosine similarity between Q and A is higher than the cosine similarity between Q and B. For each moment Q in our training corpora, we pair it with a moment A randomly sampled from within a few seconds of gameplay (in a speedrun) while B is randomly sampled from the rest of the corpus.

Speedruns provide us with one more kind of data not used in the above techniques: control input data. We also consider models where the inputs associated with a moment must be reconstructable given the embedding of the current moment's screenshot image and the embedding from a moment a few seconds later. We conjecture that control information may reveal useful visual structure related to play affordances. We leave comparative evaluation of these strategies and their combinations to future work. Unless otherwise specified, our work proceeds using the embeddings trained on the simple pixels-to-memory proxy task.

A typical method for visualizing data in high-dimensional spaces such as ours is the t-distributed stochastic neighbor embedding (tSNE) algorithm [12]. A tSNE visualization of three of our corpora is visible in Figure 3. In this visualization, we found that screenshots taken from the same room or level in the game tended to be part of the same cluster while structure within clusters sometimes echoed the structure of gameplay possibilities (such as when the player has multiple distinct routes to achieve a goal).

4.2 Moment Trees

Because very few crawled moments will ever be requested by users, we would like to optimize storage of moment data outside of the index. We would not like to pay the full 3 megabyte cost for each moment in a corpus. By exploiting the deterministic nature of our selected emulator and the availability of the control inputs used in the crawl, we can achieve significant compression of a corpus. Figure 4 shows an example of our compressed tree representation.

The key idea is to explicitly represent the full platform snapshot data for just a single moment in the corpus. We call this the root state (and typically it is equivalent to the platform's clean boot state). All other moments are represented by the sequence of inputs needed to apply each frame to reach that state. An integer value from 0 to 4095 represents (in binary) the state of the primary controller's 12

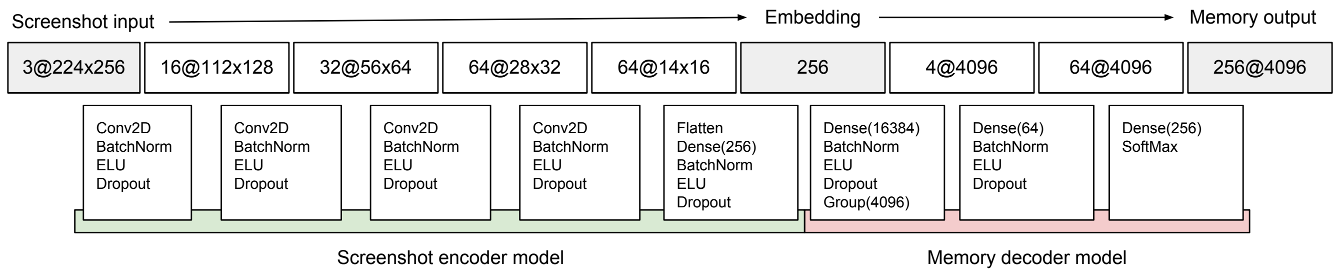

Screenshot input
Embedding
Memory output
3@224x256
16@112x128
32@56x64
64@28x32
64@14x16
256
4@4096
64@4096
256@4096
Conv2D BatchNorm ELU Dropout
Conv2D BatchNorm ELU Dropout
Conv2D BatchNorm ELU Dropout
Conv2D BatchNorm ELU Dropout
Flatten Dense(256) BatchNorm ELU Dropout
Dense(16384) BatchNorm ELU Dropout Group(4096)
Dense(64) BatchNorm ELU Dropout
Dense(256) SoftMax
Screenshot encoder model
Memory decoder model

Figure 1: Architecture of our deep neural network for predicting memory contents from screenshot pixels. The top row illustrates data representations (by tensor shape) while the bottom row represents data transformations (by layer type). All `Conv2D` layers apply 3x3 filter kernels in 2x2 strided convolution. `Dropout` layers replace 20% of outputs with zeros during training only to improve robustness. After training, the memory decoder model is discarded and the screenshot encoder model is kept for future use.

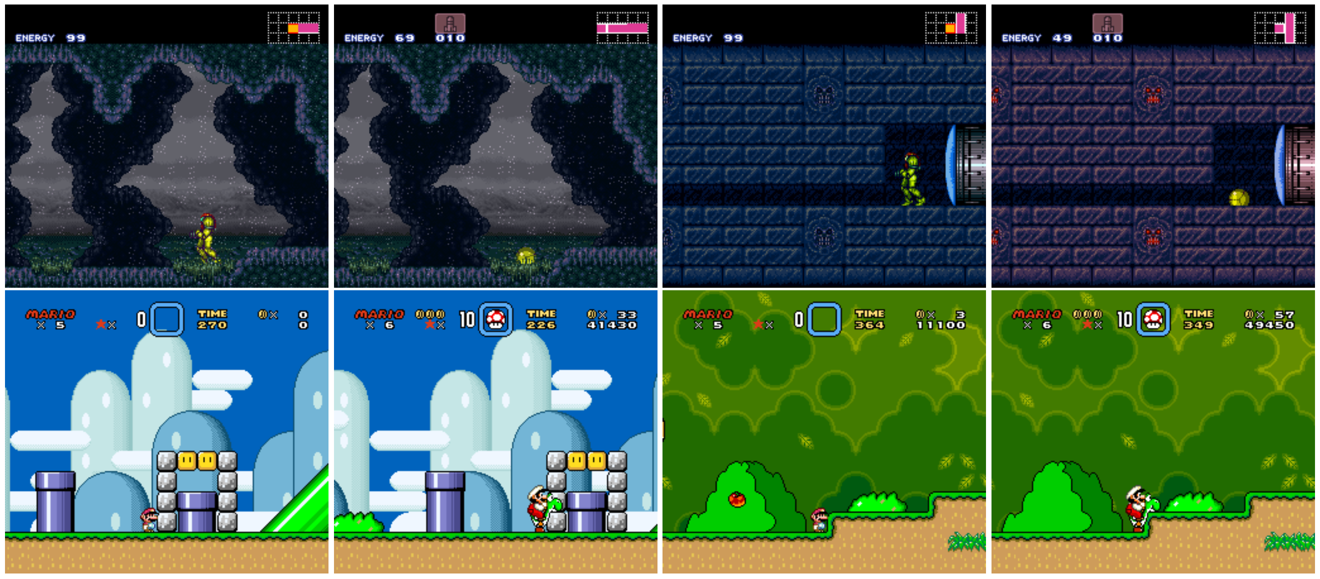


Figure 2: Left-to-right, images A, B, C, and D from *Super Metroid* and *Super Mario World* were selected to express the analogy that A is to B as C is to D. Surprisingly, linear operations on the vector representations of these moments is sufficient to construct a query vector $Q = C + \alpha(B - A)$ that is closer to D than it is to C (verified for $\alpha = 1$).

buttons during that animation frame. If we think of a graph formed by the nodes discovered in our various crawling approaches, that graph always forms a tree. From a given parent moment, we apply just a few frames worth of input over time to reach a child moment. Speedruns form long chains, speedrun branches form spindly trees consisting of chain segments, and RRT produces very bushy trees in which some moments have many many children.

For a typical corpus (consisting of a few thousand moments), the amortized storage cost per moment is approximately one kilobyte. To facilitate visual inspection of a moment before trying to reconstruct the full platform state, we also store a losslessly compressed (PNG) representation of the screen at the time of each moment. Because of repeating pixel patterns resulting from the SNES's sprite-driven graphics system, these images compress quite well (usually to low tens of kilobytes each).

5 RETRIEVING

Having collected and indexed a large number of videogame moment corpora, how can we retrieve those moments that are best matched to a user's query?

5.1 Queries

We imagine that users of future videogame moment search engines will most often search by image, using screenshots of gameplay. One source of these images is by direct interactive play of the game of interest. If the user does not have access to the game, does not have the time or skill to play it, or simply does not know how to proceed in the game, they may use existing search engines to find gameplay videos from other players. These videos can be skimmed to find examples of the moment of interest. Meanwhile, screenshots might be found in existing documents (e.g. research papers, webpages,

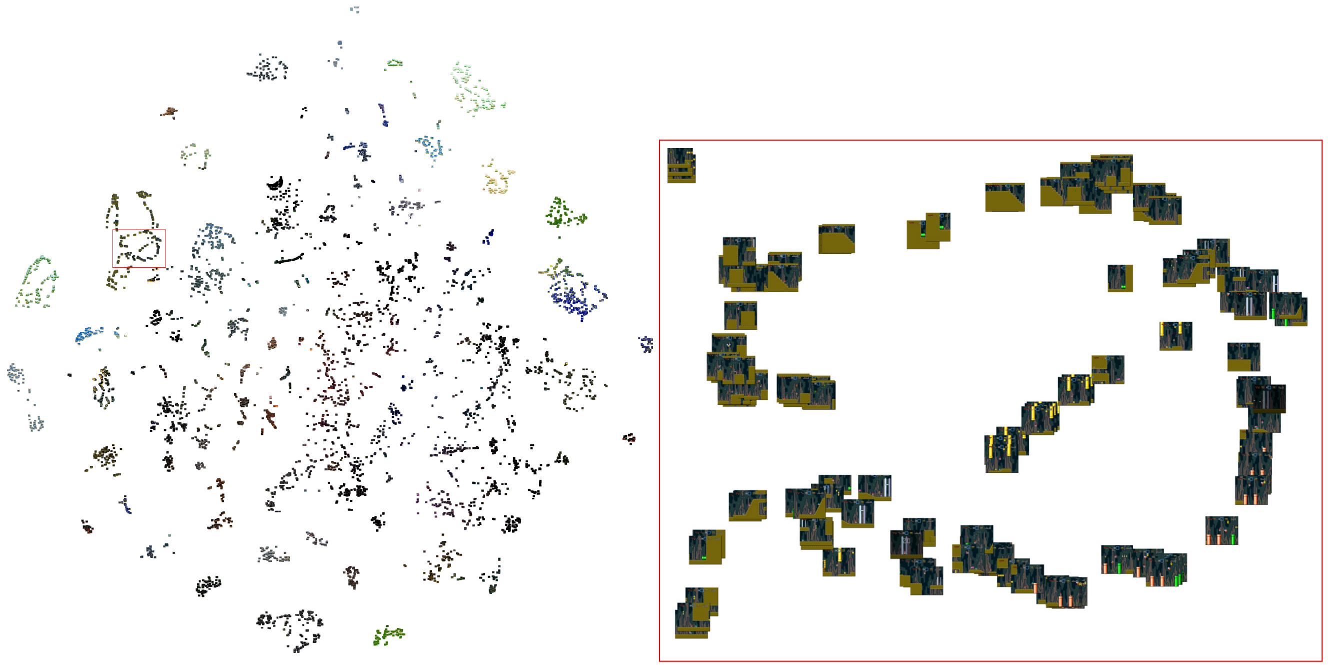
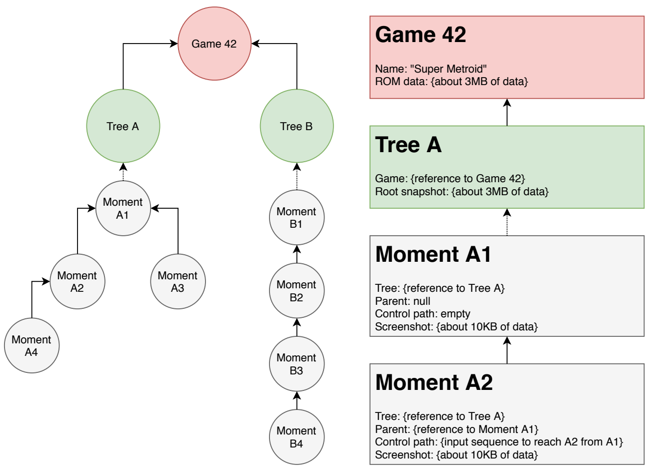


Figure 3: A tSNE visualization of embeddings for approximately 10,000 moments from speedruns for *Super Mario World, Super Metroid*, and *ActRaiser* (left). Detail for a cluster of *Mario* moments where different paths in a level are visible (right).



Game 42
Tree A
Tree B
Moment A1
Moment A2
Moment A3
Moment A4
Moment B1
Moment B2
Moment B3
Moment B4
Game 42
Name: "Super Metroid"
ROM data: {about 3MB of data}
Tree A
Game: {reference to Game 42}
Root snapshot: {about 3MB of data}
Moment A1
Tree: {reference to Tree A}
Parent: null
Control path: empty
Screenshot: {about 10KB of data}
Moment A2
Tree: {reference to Tree A}
Parent: {reference to Moment A1}
Control path: {input sequence to reach A2 from A1}
Screenshot: {about 10KB of data}

Figure 4: Two moment trees for a single game. Tree A is a branchy tree as might result from an automatic exploration algorithm. Tree B is a linear chain tree as might result from playing back an expert speedrun input sequence.

etc.). When trying to locate moments in a fan-created modification of an existing game, screenshots from the original game might also prove useful queries.

After the user has selected a query image, we can apply the same embedding model used to index the outcome of a crawl. This will result in a query vector that lives in the same space as those used in the indexes. If the user selects multiple images to use as a query (or selects a video snippet from which we can sample a representative set of individual frames), one simple strategy is to average the vectors associated with each individual query image. Although we expect few users to search by memory state (they must have a platform snapshot in hand), it is still useful to think of memory embedding vectors as possible (components of) query vectors. See Section 6 for the treatment of multiple query images and memory embedding vectors in our example search engine.

5.2 Relevance

Given a query vector and archives of moments indexed by their vectors, how do we estimate how relevant a moment is to the query? Many different distance metrics are reasonable to apply, but we have focused our initial experiments on using the cosine similarity metric (related to the dot product between normalized versions of the document and query vectors).

Much past research has gone into developing algorithms and systems that accelerate nearest neighbor lookup (including in the cosine-similarity sense). A recently released open source library from Facebook uses distributed processing on multiple GPUs to quickly answer queries over billions of images [7]. At the current scale of our datasets, it takes more time to apply the deep embedding model to a single query image on the GPU than it does to perform a naïve linear scan on the CPU for the thousands of moment vectors in each of our corpora.

In our search engine, we simply rank (sort) all of the moments in a given corpus by their cosine similarity to a query. In more sophisticated search engine designs (such as the Maguro system in Microsoft's Bing search engine [17]), multiple layers of re-ranking systems are applied to more carefully sort and prune successively smaller lists of documents by more and more complex criteria. Reranking is an excellent technique for addressing both scalability and search quality in Web-scale IR systems, but we leave exploration of this in our domain to future researchers.

5.3 Evaluation

How can we say if one version of a search engine is better than another or if one's model of relevance better captures the sense of relevance that a user intends when they form their query? Precision and recall are well established metrics for evaluating IR systems [13, Chap. 9]. These metrics (and others derived from them) require a sense of ground truth: a signal from outside the system being evaluated that says whether (or how much) a document is truly relevant to a query. Human evaluation is one source, but proxy measures that make use of extra information are also useful.

To evaluate our videogame moment retrieval systems, we have applied the mean average precision (mAP) metric [13, Chap. 9]. Informally, mAP computes the density of truly relevant results near the top of the result list, averaged over a set of test queries and the results that the search engine yields for them. To judge the ground truth relevance of a moment to a query, we have constructed experiments based on speedrun data from games with a known memory layout and scene structure (*Super Mario World* and *Super Metroid*). In one time-oriented proxy for ground truth relevance, we say that one moment in a speedrun is relevant to all others within a fixed time threshold (e.g. 60 seconds). In a space-oriented proxy, we say that one moment in a speedrun is relevant to all others that happen in the same room or level (using memory layout knowledge for specific games to decode the location from the memory state). Despite the time and space-based proxies capturing very different notions of relevance, our screenshot and memory state embeddings based on deep neural networks yield surprisingly consistent performance. In our previous work [22] we found that embeddings based on the pixels-to-memory proxy task yield a typical mAP score near 0.5 for a variety games and relevance definitions.[16]

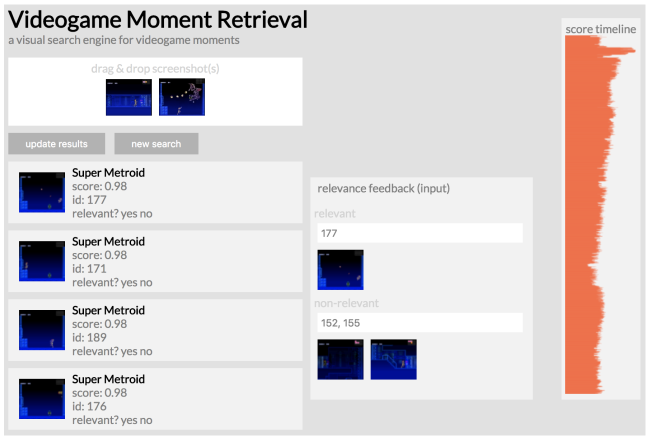
Future work, ideally in contact with game scholars and other potential users, should attempt to identify senses of moments and moment relevance that are useful for answering realistically posted queries. For example, is it important that the system provides intuitive results for black-and-white or low-contrast screenshots? Do users want to search by hand-drawn sketches?

6 A VISUAL SEARCH ENGINE

Combining all of the above techniques for crawling, indexing, and retrieving moments from over 750 SNES games, this section describes our prototype visual search engine.

6.1 Technical Design

Our initial prototype was implemented as a static webpage (no server-side logic) leveraging the KerasJS[17] library for applying pretrained deep neural networks in the browser environment. Our image embedding models and the corresponding indexes resulting


Videogame Moment Retrieval
a visual search engine for videogame moments
drag & drop screenshot(s)
update results
new search
score timeline
relevance feedback (input)
relevant
non-relevant

Super Metroid
score: 0.98
id: 177
relevant? yes no

Super Metroid
score: 0.98
id: 171
relevant? yes no

Super Metroid
score: 0.98
id: 189
relevant? yes no

Super Metroid
score: 0.98
id: 176
relevant? yes no

177

152, 155

Figure 5: Prototype visual search engine. Searching within a *Metroid* speedrun corpus, two screenshots were used as initial input and three images were tagged from the initial results list as positive and negative examples of relevant moments. The red graph on the right indicates retrieval score (cosine similarity) as a function of time within the input recording. The timeline indicate that this query most resembles moments in a specific brief scene that appeared near the start of the speedrun (the game's tutorial level).

from applying them to our corpora were implemented in the Python (with Jupyter Notebook).

From a fixed list, the user first selects a specific corpus of moments they want to search within. This causes the system to load both the bulk data of corresponding index of the corpus and the Keras image embedding model that was used to generate the index.

Next, when a user drags one or more images onto our page, the system applies the embedding model to the each image to yield an averaged query vector. The cosine similarity between this vector and the indexed vectors is used to compute a top-list of search results. For each result, we display a thumbnail of the moment's screenshot, an estimated relevance score (simply the cosine similarity in the prototype), and an identification number. This number is sufficient for separately recreating the full game state using the moment tree representation.

Figure 5 shows a screenshot of our initial prototype system in action. A video of this prototype is available online.[18]

In the next iteration of our prototype, we are adding significant server-side logic that should reduce the amount of data that needs to be loaded into the user's browser (making it much more like the visual search offering in Google Image Search) and supporting multi-corpus search.

6.2 Relevance Feedback

In a textual search engine, it is relatively easy for a user to edit their query string in response to patterns they see in initial search results. Perhaps they fix their spelling or add a disambiguating keyword. For visual search, editing one's query image is not so easy. Finding related or disambiguating screenshots could require

[16]Searching with access to complete memory state information yields a mAP of about 0.7 while a random baseline scores <0.1 on the same games/tasks.
[17]https://github.com/transcranial/keras-js

[18]https://drive.google.com/open?id=1eGkx1mh_Nry1hHP2S4j0p4HVL2pMCzTy

interaction with a different visual search engine or even additional skillful game playing effort. In an effort to support more fluent interaction with our search engine, we implemented a relevance feedback mechanism based on the classic Rocchio algorithm [18].

In relevance feedback, users can browse the initial results of a search to mine positive and negative examples of relevant results. A classic example of this interaction would be in a text-driven image search engine where the user has searched for the term “bike.” Upon seeing various visual results, the user might select a motorcycle image a negative example if their true goal is to find images of bicycles.

In our search engine, the user can re-submit their original query augmented with any number of positive and negative examples selected from the previous results. In the Rocchio algorithm (operating within the vector space retrieval model), a modified query vector is formed by a weighted average of the original query vector, the vectors associated with documents (moments) form the positive and negative example sets. Negative results are intuitively weighted negatively. This can be interpreted as exploiting vector analogies in the embedding space [6].

Because individual screenshots can be highly ambiguous, relevance feedback offers a way for the system to leverage its understanding of memory states. Imagine forming the vector representation of a moment by concatenating the 256-dimensional embedding of a screenshot with the 256-dimensional embedding of its memory state. In initial query vectors computed from user-submitted screenshots, we can fill in all-zero values for the memory components of this vector. However, upon tagging positive and negative examples from initial results, the Rocchio algorithm will produce a modified query vector with non-zero disambiguating values in the memory components of the vector. In a game like *Super Metroid* where which powerups a player has already collected is not easily discerned by inspecting an individual screenshot, this ability to reason about unobserved game state is important.

6.3 Experiment: Chozo Statue Battle

To demonstrate our search engine and qualitatively compare its behavior to commercial visual search offerings, consider two images in Figure 6. These two images were extracted from a YouTube video[19] of gameplay for *Super Metroid*. The images are blurry, contain motion and video compression artifacts, and are imprecisely cropped. The two images depict moments related to a surprising battle with a hostile Chozo statue. Chozo statues are typically inanimate, usually holding beneficial items for the player to collect.

Using the current (March 2018) public version of Google Image Search on the first image yields impressive results. It offers “super metroid chozo statue boss” as a text approximation of the query, and three of the top ten “visually similar” images appear to come from the statue battle scene, four show Chozo statues from elsewhere in the game, and the remaining three show Chozo statues in other *Metroid* titles (for platforms released later than the SNES). How can Google do so well? This iconic view of the Chozo statue battle is very similar to screencaptures created by fans of the game to post on game tutorial websites or manually selected as the thumbnail image for videos of this battle scene. This moment in the game is culturally

[19]https://www.youtube.com/watch?v=yB317FOcU0Y

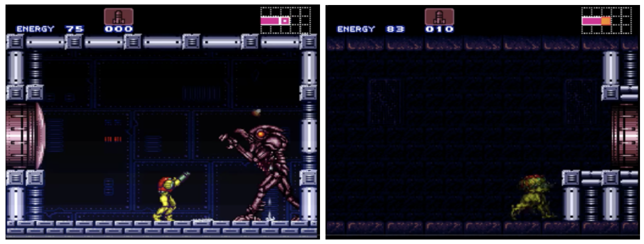


Figure 6: Two frames extracted from a YouTube video showing a battle with a Chozo statue (known as the Bomb Torizo). The first frame is an iconic depiction of this battle while the second, less distinctive view comes from just before the Chozo chamber is first entered.

important enough for it to have crept out of the interactive media of videogames into the text and image formats that search engines like Google understand. Google’s emphasis on text explains why related scenes from other *Metroid* titles appear in the results. By contrast, all of the top ten results from TinEye (which does not consider text) are from the desired battle scene in *Super Metroid* (perfect precision).

Using Google Image Search on the second image—from a moment just before the player enters the room where the Chozo statue battle occurs—yields much less impressive results. It offers “action-adventure game” as the text approximation of the query, and none of the top ten “visually similar” results are from any *Metroid* game title. TinEye manages to produce just one result for this less interesting moment, however it is a very close match. The result image is an automatically generated thumbnail for a video posted by the @WatchSpeedruns Twitter account. For either search engine, precision is very low.

Using our search engine yields the results in Fig. 7. Just a single image is used here and no examples are provided to the relevance feedback mechanism. Querying with the first image, nine of the top ten results show the desired Chozo statue battle. Querying with the second image, three of the top ten show the door entering into the Chozo chamber (as in the query image) while three more depict the room that would contain the battle itself (which the search engine as deemed to be a similar moment despite the iconic Chozo not visible in the query image). Upon careful manual inspection of this particular moment corpus for *Super Metroid* (derived from a different playthrough than the one in the YouTube video) we found that the search engine had successfully returned all three representations of the door outside the Chozo chamber (perfect recall). The search engine returns no results showing the statue actually attacking because the speedrun data we used interestingly demonstrated the difficult “Bomb Torizo skip” technique[20] (in which the sleeping statue is never wakened). Considering that the query images show moments that are only similar to but not exact matches for any in the indexed corpus (from a single speedrun), our search engine is producing a dense collection of reasonably relevant results.

[20]http://deanyd.net/sm/index.php?title=Bomb_Torizo#Bomb_Torizo_Skip

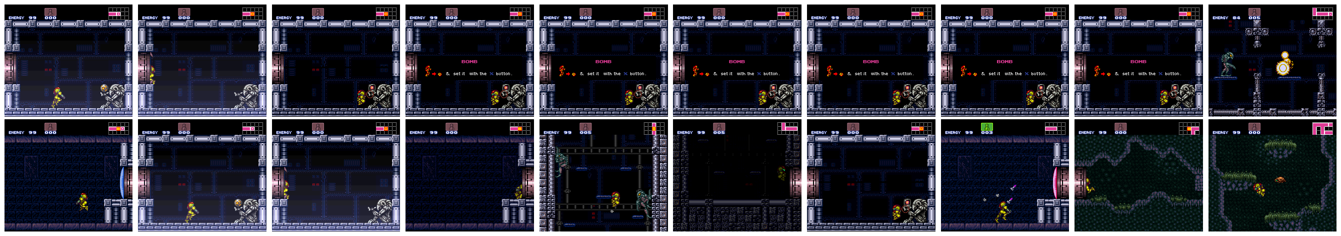


Figure 7: Top ten results from our prototype visual search engine for the iconic (top) and less distinctive (bottom) images from Figure 6. Results are taken from a speedrun corpus distinct from the gameplay session that yielded the query images.

7 CONCLUSION

In this paper, we have introduced the problem of content-based retrieval for moments in videogames, and we have shared a collection of techniques for crawling, indexing, and retrieving those moments. We combined these ideas into a prototype visual search engine and tested it on in-the-wild query images and games. We invite a new generation of AI and IR researchers to develop algorithms that can efficiently harvest the most interesting moments in games by automatically playing them, ideally taking the opportunity to transfer play and design knowledge between previously indexed games. We invite technical game researchers to refine visual understanding models from computer vision and extend them with notions of agency that result from relating how a player can act to what they can observe during play. We also invite game scholars to demand better tooling that will assist them in carrying out digital humanities research.

ACKNOWLEDGMENTS

We would like to thank Yash Dua and Zijie Zhang for their assistance in producing 15-minute recordings of gameplay for over 100 SNES games. Playing games on a schedule is harder than it sounds. The idea of quickly getting attract-mode data for the full 750 game collection is due to Joël Franusic. Verification of the vector analogy property for our moment vectors was performed by Barrett Anderson.

We would also like to acknowledge the essential contributions of publicly available interaction data from the tool-assisted speedrunning community (TASvideos.org) and memory layout information from the ROM hacking community (RomHacking.net).

REFERENCES

[1] Aaron Bauer and Zoran Popović. 2012. RRT-Based Game Level Analysis, Visualization, and Visual Refinement. In *Proc. of the AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment.*

[2] Sean Bell and Kavita Bala. 2015. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 98.

[3] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. 2016. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems.* 1471–1479.

[4] Vijay Chandrasekhar, Matt Sharifi, and David A Ross. 2011. Survey and Evaluation of Audio Fingerprinting Schemes for Mobile Query-by-Example Applications.. In *ISMIR*, Vol. 20. 801–806.

[5] W Bruce Croft, Donald Metzler, and Trevor Strohman. 2010. *Search engines: Information retrieval in practice.* Vol. 283. Addison-Wesley Reading.

[6] Gregory Finley, Stephanie Farmer, and Serguei Pakhomov. 2017. What analogies reveal about word vectors and their compositionality. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017).* 1–11.

[7] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[8] Karen Sparck Jones. 1999. Information retrieval and artificial intelligence. *Artificial Intelligence* 114, 1-2 (1999), 257–281.

[9] Eric Kaltman, Joseph Osborn, Noah Wardrip-Fruin, and Michael Mateas. 2017. Game and Interactive Software Scholarship Toolkit (GISST). (2017).

[10] John Koetsier. 2013. How Google searches 30 trillion web pages, 100 billion times a month. *Venture Beat* (March 2013). https://venturebeat.com/2013/03/01/how-google-searches-30-trillion-web-pages-100-billion-times-a-month/

[11] Steven M LaValle. 1998. Rapidly-exploring random trees: A new tool for path planning. (1998).

[12] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[13] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[15] Mark J Nelson. 2011. Game Metrics Without Players: Strategies for Understanding Game Artifacts.. In *Artificial Intelligence in the Game Design Process.*

[16] Joseph Osborn, Adam Summerville, and Michael Mateas. 2017. Automatic mapping of NES games with mappy. In *Proceedings of the 12th International Conference on the Foundations of Digital Games.* ACM, 78.

[17] Knut Magne Risvik, Trishul Chilimbi, Henry Tan, Karthik Kalyanaraman, and Chris Anderson. 2013. Maguro, a system for indexing and searching over very large text collections. In *Proceedings of the sixth ACM international conference on Web search and data mining.* ACM, 727–736.

[18] Joseph John Rocchio. 1971. Relevance feedback in information retrieval. *The SMART retrieval system: experiments in automatic document processing* (1971), 313–323.

[19] Linda C Smith. 1976. Artificial intelligence in information retrieval systems. *Information Processing & Management* 12, 3 (1976), 189–222.

[20] James Somers. 2017. Torching the Modern-Day Library of Alexandria. *The Atlantic* (April 2017). https://www.theatlantic.com/technology/archive/2017/04/the-tragedy-of-google-books/523320/

[21] Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N Yannakakis. 2013. The mario ai championship 2009-2012. *AI Magazine* 34, 3 (2013), 89–92.

[22] Zeping Zhan and Adam M Smith. 2015. Retrieving Game States with Moment Vectors. (2015).