CLIP meets GamePhysics: Towards bug identification in gameplay videos using zero-shot transfer learning

Mohammad Reza Taesiri
taesiri@ualberta.ca
University of Alberta
Edmonton, AB, Canada

Finlay Macklon
macklon@ualberta.ca
University of Alberta
Edmonton, AB, Canada

Cor-Paul Bezemer
bezemer@ualberta.ca
University of Alberta
Edmonton, AB, Canada

ABSTRACT

Gameplay videos contain rich information about how players interact with the game and how the game responds. Sharing gameplay videos on social media platforms, such as Reddit, has become a common practice for many players. Often, players will share gameplay videos that showcase video game bugs. Such gameplay videos are software artifacts that can be utilized for game testing, as they provide insight for bug analysis. Although large repositories of gameplay videos exist, parsing and mining them in an effective and structured fashion has still remained a big challenge. In this paper, we propose a search method that accepts any English text query as input to retrieve relevant videos from large repositories of gameplay videos. Our approach does not rely on any external information (such as video metadata); it works solely based on the content of the video. By leveraging the zero-shot transfer capabilities of the Contrastive Language-Image Pre-Training (CLIP) model, our approach does not require any data labeling or training. To evaluate our approach, we present the `GamePhysics` dataset consisting of 26,954 videos from 1,873 games, that were collected from the GamePhysics section on the Reddit website. Our approach shows promising results in our extensive analysis of simple queries, compound queries, and bug queries, indicating that our approach is useful for object and event detection in gameplay videos. An example application of our approach is as a gameplay video search engine to aid in reproducing video game bugs. Please visit the following link for the code and the data: https://asgaardlab.github.io/CLIPxGamePhysics/

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

video mining, bug reports, software testing, video games

1 INTRODUCTION

Video game development is a highly complex process. There are many unique challenges when applying general software engineering practices in video game development [32, 37, 40, 43, 48], including challenges in game testing. Manual testing is a widely accepted approach to game testing [39, 42, 52], however this manual process is slow and error-prone, and most importantly, expensive. On the other hand, it is challenging to automate game testing [31, 39, 47] due to the unpredictable outputs of video games. Despite progress towards automated game testing methods [13, 34, 51, 52] and game testing tools [5, 24, 41, 58], new approaches to game testing must be researched.

Figure 1: Video identified by our approach with the bug query *'A horse in the air'* for Red Dead Redemption 2.

The difficulty of game testing due to the unique nature of games calls for unique testing methodologies as well. For example, we could leverage the visual aspect of games in the testing process. Having a gameplay video is very helpful when trying to reproduce a bug in the development environment for further analysis, as bug reports often contain incomplete information [7]. The ability to search a large repository of gameplay videos with a natural language query would be useful to help reproduce such bug reports. For example, in the game development domain, a bug report might state 'my car is stuck on the rooftop' without any screenshot or video to show what is actually happening. A gameplay video search would allow game developers to find an example instance of a specific bug in the pile of gameplay videos from their playtesting sessions or the internet (e.g., YouTube, Twitch).

Despite containing rich information, the challenges related to video parsing and understanding mean that gameplay videos are difficult to utilize. Manually identifying bug instances is time consuming, and there is limited prior research on automatic methods for mining large repositories of gameplay videos [33, 35].

In this paper, we address the challenges of extracting useful information from large repositories of gameplay videos. We propose an approach for mining gameplay videos using natural language queries by leveraging the Contrastive Language-Image Pre-Training (CLIP) model [45] to identify similar text-image pairs without any additional training (i.e., zero-shot prediction). We leverage CLIP for videos by pre-processing the video frame embeddings and use Faiss [23] to perform a fast similarity search for the pairs of text queries and video frames. In our approach, we present two methods to aggregate across the similarity scores of each text-image pair to identify relevant videos. To evaluate our approach, we collected and prepared the `GamePhysics` dataset, consisting of 26,954 gameplay

videos that predominantly contain game physics bugs. We evaluate our approach with sets of simple queries, compound queries, and bug queries, and show that our approach can identify objects and (bug-related) events in large repositories of gameplay videos. Figure 1 shows an example of a video that was identified by our approach when searching videos from the Red Dead Redemption 2 game using the bug query *'A horse in the air'*. The primary application of our approach is as a gameplay video search engine to aid in reproducing game bugs. With further work, e.g. setting thresholds to limit false positives, our approach could also be used as a bug detection system for video games.

The main contributions of our paper are as follows:

- We propose an approach to search for objects and events in gameplay videos using a natural language text query.
- We collect and prepare the `GamePhysics` dataset, consisting of 26,954 gameplay videos from 1,873 games.
- We report results that demonstrate the promising performance of our approach in identifying game physics bugs through 3 experiments with simple, compound, and bug queries.

The remainder of our paper is structured as follows. In Section 2, we motivate our study by providing relevant background information. In Section 3, we discuss related work. In Section 4, we present our approach to mining large repositories of gameplay videos. In Section 5, we discuss collecting and pre-processing the `GamePhysics` dataset. In Section 6 we detail our experiment setup, and in Section 7 we present our results. In Section 8 we provide discussion and insights on the performance of our approach. In Section 9 we outline limitations of our approach. In Section 10 we address threats to validity. We conclude our paper in Section 11.

2 MOTIVATION AND BACKGROUND

2.1 Video game (physics) bugs

In this paper, we are interested in a specific category of bugs in video games that we call 'game physics' bugs. Game physics bugs are not necessarily related to an inaccurate physics simulation. Many of these bugs are related to the faulty representation of game objects due to an error in the internal state of that object. A few sample instances of game physics bugs can be seen in Figure 2. In Figure 2a, a bug from Grand Theft Auto V related to object collisions is shown. Figure 2b shows a bug from The Elder Scrolls V: Skyrim, related to object clipping. In Figure 2c, a bug from Red Dead Redemption 2 related to ragdoll poses can be seen. Figure 2d shows a bug from Cyberpunk 2077, related to object collisions. Identifying game physics bugs is challenging because we need to be able to extract specific, high-level (abstract) events from the gameplay videos, that are often similar to correct behaviour.

2.2 Challenges in mining gameplay videos

Until now, it has been challenging to extract valuable information from large repositories of gameplay videos. Identifying bug instances by manually checking the contents of gameplay videos is time-consuming [33]. Therefore, automatic methods for mining gameplay videos are required. The only existing approach for automatically extracting events from gameplay videos requires manual data labelling (and the training of new models) [35], which itself is time-consuming. Therefore, an effective method for extracting valuable information from gameplay videos should be able to automatically analyze the video contents without requiring manual data labelling.

2.3 Contrastive learning and zero-shot transfer

While there are many approaches towards zero-shot learning, we are interested in assessing the zero-shot performance of pre-trained contrastive models. Contrastive learning is a machine learning technique in which the goal is to learn a representation of inputs such that similar items stay close to each other in the learned space, while the dissimilar items are far away [3, 9]. In recent years, contrastive learning has been one of the key drivers in the success of self-supervised learning methods and has been used for zero-shot transfer learning [11, 18, 28, 45]. Zero-shot learning is a family of problems in machine learning, in which an algorithm is required to solve a task without having a training set for that specific task [29, 30]. To illustrate this idea, suppose that a person has never seen a zebra before. If we give a detailed description of a zebra to them (e.g., an animal similar to a horse, but with black-and-white stripes all over their bodies), that person can identify a zebra when they see one.

2.4 The Contrastive Language-Image Pre-Training (CLIP) model

One contrastive model that has proven zero-shot transfer capabilities is the Contrastive Language-Image Pre-Training (CLIP) model [45], which can leverage both text and image inputs together. We decided to use CLIP because of its multimodal capabilities and the size of its training dataset. The CLIP model consists of two parts: a text encoder, and an image encoder. These two parts work individually, and they can accept any English text and image as input. When an encoder of this model receives an input, it will transform it into an embedding vector. These embedding vectors are high-level features that are extracted by the network, representing the input. More specifically, these embedding vectors are how the neural network represents, distinguishes, and reasons about different inputs. Both encoders of this model will produce vectors of the same dimension for image and text inputs. Not only do these vectors have the same dimension, but they are also in the same high-dimensional feature space, and are therefore compatible with each other. For example, the embedding vector of the text 'an apple' and the embedding vector of an image of an apple are very close to each other in this learned space. The CLIP model was pre-trained on over 400 million pairs of images and text descriptions that were scraped from the internet, and has six different backbone architectures: 'RN50', 'RN101', 'RN50x4', 'RN50x16', 'ViT-B/32', 'ViT-B/16'. The models with 'RN' in their name are ResNet-based [21] models using traditional convolutional layers, while the 'ViT' models are based on vision transformers [14].

3 RELATED WORK

Event extraction from video content is of special importance for various data mining tasks [36, 44]. Only two prior studies have explicitly explored automatic approaches for mining gameplay videos,

(a) Bug in Grand Theft Auto V. Car stuck in a tree after colliding.

(b) Bug in The Elder Scrolls V: Skyrim. Dragon stuck in the ground.

(c) Bug in Red Dead Redemption 2. Incorrect sitting animation.

(d) Bug in Cyberpunk 2077. Cars stuck together after colliding.

Figure 2: Sample instances of game physics bugs.

with varying success. Lin et al. showed that using metadata (such as keywords) to identify YouTube videos that contain video game bugs is feasible [33], but our approach looks at the video contents, which Lin et al. do not take into account. Our approach is more useful for game developers, as we can identify objects and (bug-related) events within gameplay videos. Luo et al. propose an approach for automatic event retrieval in e-sport gameplay videos that requires manual data labelling, a fixed set of classes (events), and the training of new models [35]. Our approach is more robust and easier to set-up, as we can search gameplay videos with any English text query to identify specific objects and events without performing manual data-labelling.

Although there is limited prior work on mining large repositories of gameplay videos, there are several studies that propose approaches to automatically detect graphics defects in video games. One of the earliest approaches for automated detection of graphics defects was published in 2008, in which a semi-automated framework was proposed to detect shadow glitches in a video game using traditional computer vision techniques [38]. Recent studies have utilized convolutional neural networks in their approach to automatically detect a range of graphics defects [10, 13, 34, 49]. Instead of detecting graphics defects, our work is concerned with the automatic identification of game physics bugs in gameplay videos.

Tuovenen et al. leverage the visual aspect of games through an image matching approach to create a record-and-replay tool for mobile game testing [51]. Our approach leverages the visual aspect of games in a different way; instead of recording tests through gameplay, we automatically identify bugs in gameplay videos.

Some studies have proposed approaches for automated detection of video game bugs through static or dynamic analysis of source code. Varvaressos et al. propose an approach for runtime monitoring of video games, in which they instrument the source code of games to extract game events and detect injected bugs [52]. Borrelli et al. propose an approach to detect several types of video game-specific bad smells, which they formalize into a tool for code linting [8]. Our approach differs as we do not require access to the source code of games; instead we identify video game bugs based solely on the contents of gameplay videos.

In addition to related work on automatic bug detection for video games, there exists a wide range of work that leverages recent advancements in deep learning to provide new tools and techniques that address problems faced by game developers. Several studies have sought to make AI methods accessible in the video game development and testing cycle, either through the game's internal state, raw pixels, or through a high-level neural network-based representation [27, 50]. Some studies have proposed approaches to accompany a game designer through the creation process of a game by providing suggestions and explanations to the designer [19, 20, 26]. Other studies have incorporated reinforcement learning and evolutionary methods to create AI agents that can automatically

play games [6, 25, 55]. These AI agents can be further employed to perform automated game testing sessions [4, 16, 17, 46, 58]. Our work is different from those listed above, as we focus on assisting game developers by providing an approach to efficiently search large repositories of gameplay videos to find bug instances.

4 OUR APPROACH

To assist with detection and analysis of game bugs, we propose an approach that quickly and effectively searches a large repository of gameplay videos to find a specific object or event in a particular game. For creating such a powerful search system, one could utilize a traditional supervised classification technique. However, any supervised classification method requires a training dataset, a test dataset, and a fixed number of classes. Maintaining these two datasets and labeling each sample is demanding and labour-intensive. On the other hand, the CLIP model provides zero-shot transfer learning capabilities that allow us to develop an approach to automatically mine gameplay videos while avoiding the aforementioned issues. Figure 3 shows an overview of our approach.

4.1 Encoding video frames and the text query

Our approach accepts a set of videos and any English text query as inputs. We first extract all frames from each video, and then use the CLIP model to transform our input text query and input video frames into the embedding vector representations described in Section 2.4. We selected the CLIP model because it is flexible enough to accept any arbitrary English text as a query and compare it with a video frame, without any additional training.

4.2 Calculating the similarity of embeddings

As well as avoiding manual data labelling, our approach avoids depending upon any extra information, such as metadata, to search gameplay videos. Instead, we are able to calculate similarity scores solely based on the contents of the video frames and the text query. The similarity score in our problem is a distance between an embedding vector representing a text query and another embedding vector representing a video frame. To calculate similarity scores for the pairs of embedding vectors, we opted for cosine similarity, a widely-used similarity metric [15, 53, 54, 57]. We require an exhaustive search to calculate the similarity score of the text query with each individual frame in each input video. The performance of an exhaustive search will decrease inversely with an increasing number of videos in a repository. To combat this, we use Faiss [23] to conduct an efficient similarity search.

4.3 Aggregating frame scores per video

Although CLIP is designed to accept text and images as inputs, we can leverage CLIP for videos by treating each video as a collection of video frames (i.e. a collection of images). To identify specific events that could occur at any moment in a gameplay video, we cannot subsample the video frames as suggested in the original CLIP, because due to the richness of events in a single gameplay video, skipping any part of the video may lead to information loss and inaccurate results. Therefore, we perform a similarity search on all frames of all videos by comparing each individual video frame with the target query text, and we subsequently aggregate

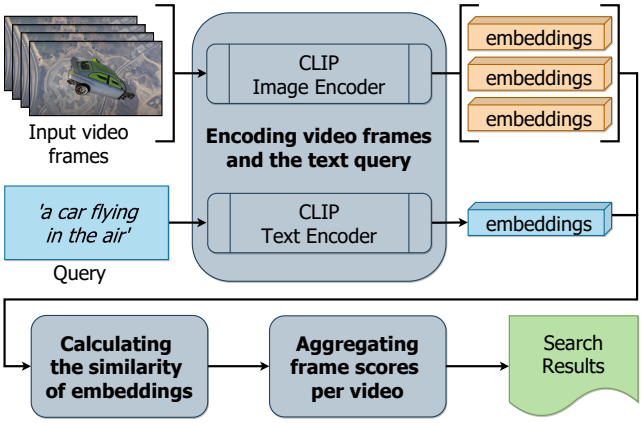


Figure 3: Overview of our gameplay video search approach.

the similarity scores across each video. Below we detail the design of two different methods for aggregating the video frame similarity scores for each gameplay video. Our approach supports the two aggregation methods without the need to re-calculate the similarity scores.

Aggregating frame scores using the maximum score. Our first score aggregation method ranks videos based on the maximum similarity score across all frames belonging to each video. This method is highly sensitive, as a single frame with high similarity can lead to an entire video being identified as relevant to the query.

Aggregating frame scores using the similar frame count. In the second score aggregation method, we begin by ranking all frames of the input videos based on their similarity scores with the text query. Then, we select a predefined number (the *pool size* hyperparameter) of highest-ranked frames across all videos. Finally, we count the number of frames per video within this pool of highest-ranked frames. This method is less sensitive than our first aggregation method, as identified videos must have multiple frames that are among the most similar to the input text query. We selected 1,000 as the default pool size value in our study.

5 PREPARING THE GAMEPHYSICS DATASET

5.1 Collecting the `GamePhysics` dataset

Developing and testing a new machine learning system requires a dataset. Unfortunately, there is no such dataset for gameplay bugs. To this end, we present the `GamePhysics` dataset, which consists of **26,954** gameplay videos collected from the GamePhysics subreddit. An overview of our data collection process can be seen in Figure 4.

Extracting post metadata and downloading videos. To collect the data, we created a custom crawler that uses both the official Reddit API and the popular `PushShift.io` API [2]. In our crawler, we use the `PushShift.io` API to get high-level information about each submission in the GamePhysics subreddit. After obtaining high-level data, we use Reddit's official API to update the scores and other metadata of each submission. For downloading the actual

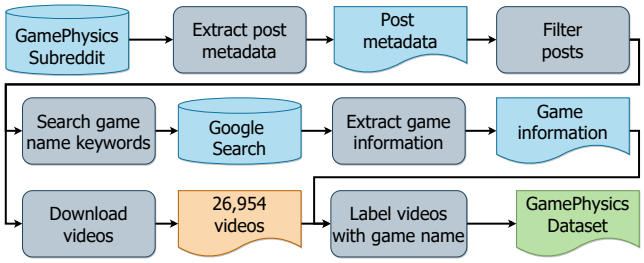


Figure 4: Overview of our data collection process.

video files, we use a combination of `youtube-dl` and `aria2c` to extract video links and download them.

Filtering posts. We applied several filters to our dataset during the data collecting process to remove spam posts, low-quality content, and outliers. There are several spam posts in the GamePhysics subreddit, and these posts are marked explicitly as spam by the subreddit's moderators. Furthermore, we treat post scores as a quality signal as this score captures up/down votes from Reddit users, and consider any post with a score of less than one as low-quality content. The lengths of the video files vary from a few seconds to multiple hours. We avoid long videos in our dataset, because they can contain multiple events of different kinds and are very hard to process. We only keep videos that are longer than 2 seconds and shorter than 60 seconds. After applying our filters, our final dataset contains **26,954** video files from **1,873** different games.

Labelling videos with the game name. In order to simulate the realistic scenario in which a game developer would search a repository of gameplay videos for a specific game, we extract the game name for each gameplay video from the title of its respective post. Detecting the game's name from a GamePhysics submission is not straightforward. While there is a community guideline that suggests including the name of the game in the submission's title, people often forget to include the game name or use several aliases for the game name, meaning the task of detecting the game name can be hard. For example, 'GTA V' is a widely-used alias that refers to the 'Grand Theft Auto V' game. To address this issue, we created a second custom crawler to search game name keywords in Google and subsequently map them to the full game name. Google search results provide a specific section called the Knowledge Panel that contains the game name, as well as other relevant game information such initial release date, genre, development studio(s), and publisher.

5.2 Pre-processing the videos

As discussed in Section 4.2, our approach can search a large repository of gameplay videos more efficiently by pre-processing the embedding vectors of every frame for each video in the repository before inputting any text queries. Therefore, for our dataset to be suitable for our approach, we pre-process all videos in the `GamePhysics` dataset before proceeding with any experiments. We pre-processed all 26,954 videos using a machine with two NVIDIA Titan RTX graphics cards, but it is certainly possible to perform this step with less powerful graphics cards too. It is worth noting that this is by far the most computationally expensive step in our approach.

6 EXPERIMENT SETUP

In this section, we describe an extensive analysis of our approach on the `GamePhysics` dataset through a diverse set of experiments. To assess the performance of our video search method, we performed several experiments with varying levels of difficulty. The main obstacle to evaluating our search system is the lack of a benchmark dataset. To this end, we designed three experiments with three corresponding sets of queries to shed light on the capabilities of our proposed method.

6.1 Experiment overview

In the first two experiments, we evaluate the accuracy of our approach when retrieving videos with certain objects in them. The results for this step indicate the generalization capability of the model for the third experiment. In the third experiment, we evaluate the accuracy of our approach when retrieving videos with specific events related to bugs.

6.2 Selecting CLIP architectures

To understand the relative performance of the available ResNet-based and vision transformer-based CLIP models, we opted to try two different backbone architectures in our system, namely 'RN101' and 'ViT-B/32'. We chose these backbones as fair baseline comparisons because they are the largest backbone architectures in their respective families, assuming we stipulate equivalent input image sizes (224×224). For comparison, the 'ViT-B/32' backbone architecture contains 151,277,313 total parameters, while the 'RN101' backbone architecture contains 119,688,033 total parameters. We selected the largest architectures as we are performing inference with these models, not training them.

6.3 Selecting video games

Our dataset contains videos from 1,873 different video games, and the differences in their high-level characteristics, such as genre, visual style, game mechanics, and camera view, can be vast. Therefore, we performed a comprehensive evaluation in all three experiments with 8 popular video games that differ in their high-level characteristics. The only uniting characteristic for our selected games is that they have open-world mechanics, because developers of open-world games would find particular benefit from an effective video search for bug reproduction. Open-world games allow a player to freely explore the game world, providing a larger set of potential interactions between the player and game environment. Open-world games are therefore more likely to suffer from game physics bugs that are difficult to reproduce. Table 1 shows each game we selected for our experiments, as well as some game characteristics and the reason for inclusion. In total, 23% of videos in the `GamePhysics` dataset are from these 8 video games (6,192 videos).

6.4 Query formulation

To come up with a set of relevant search queries in the experiments, we randomly picked 10 videos from each of the 8 selected games.

Table 1: Games selected for evaluation of our approach. All selected games are open-world.

Game	Key	Genre	Visual style	Reason for inclusion	Videos
Grand Theft Auto V	GTA	Action-adventure	Realism	Variety of vehicles	2,230
Red Dead Redemption 2	RDR	Action-adventure	Realism	Historical style	754
Just Cause 3	JC3	Action-adventure	Realism	Physical interactions	680
Fallout 4	F4	Action role-playing-game	Fantasy realism (Retro-futuristic)	Unique look and feel	614
Far Cry 5	FC5	First-person shooter	Realism	First-person camera	527
Cyberpunk 2077	C77	Action-adventure	Fantasy realism (Futuristic)	High-quality lighting	511
The Elder Scrolls V: Skyrim	ESV	Action role-playing-game	Fantasy realism	Magical effects	489
The Witcher 3: Wild Hunt	W3	Action role-playing-game	Fantasy realism	Mythical beasts	387

The first author manually reviewed each of the 80 samples to understand what was happening and how we could describe events in gameplay videos that contain bugs. This sampling process helped us pick relevant objects and events to use in our queries.

6.5 Experiment 1: Simple Queries

In this experiment, we searched for specific objects in videos, e.g. a car. Our main objective in this experiment is to demonstrate the capability of our system for effective zero-shot object identification. As a reminder, we never trained or fine-tuned our neural network model for any of these experiments or any video game. We created 22 distinct queries for Experiment 1, including transportation vehicles, animals, and special words describing the weather or environment. For this experiment we wanted our approach to operate with very high sensitivity, and so we selected our first aggregation method, i.e. using maximum frame score per video (Section 4.3).

6.6 Experiment 2: Compound Queries

Continuing our evaluation, we search for compound queries, i.e. queries in which an object is paired with some descriptor. Similar to Experiment 1, we only use compound queries that are relevant to each video game. For example, in the previous experiment, we searched for videos in the Grand Theft Auto V game that contained a car, but in this experiment we evaluate the performance of our approach when searching for objects with a specific condition, like a car with a particular color. For this second experiment, we created a set of 22 compound queries, and again selected our first aggregation method (using maximum frame score per video).

6.7 Experiment 3: Bug Queries

In the third experiment, we search for bug queries, i.e. phrases that describe an event in the game which is related to a bug. We manually created specific textual descriptions for different bug behaviors in the game and searched our dataset to see if we could detect bugs in gameplay videos. Similar to the previous experiments, our bug queries are game-specific. For this experiment, we created a set of 44 unique bug queries, with each query describing an event. Given the complex nature of the bug queries in Experiment 3, we decided to use our less sensitive aggregation method, based on the number of highly similar frames per video (as described in Section 4.3).

Table 2: Average top-k accuracy (%) per game for simple queries (Experiment 1).

		GTA	RDR	JC3	F4	FC5	C77	ESV	W3
ViT-B/32	**Top**-1	74	71	61	65	50	55	54	54
	Top-5	89	86	67	71	88	73	62	62
RN101	**Top**-1	84	50	61	59	59	43	62	62
	Top-5	89	79	83	82	94	71	92	85

6.8 Evaluating the experiments

Evaluating Experiment 1 and Experiment 2. In the first and second experiments, we assess the sensitivity of our approach by measuring top-1 and top-5 accuracy. This is because for our approach to be useful to a game developer, the search system should be able to reliably identify objects specified in the text queries. Top-k accuracy is a binary measure; if there is a correct result in the top-k results, the accuracy is 100%, otherwise the accuracy is 0% – there are no possible values in between.

Evaluating Experiment 3. In the third experiment, we measured the accuracy of our approach using recall @5. The reason for this choice is that we want to see what proportion of videos are relevant to the bug query, and how susceptible our system is to false positives when searching with bug queries. It is possible to report recall at higher levels, but the problem is that we cannot know how many videos in the dataset exactly match the search query without manually checking every video. Recall @5 is 100% when all five out of five retrieved videos match the bug query, 80% when four out of five retrieved videos match, etc. until 0% when there are no matching videos.

7 RESULTS

In this section, we present the results of the three experiments we designed to examine the ability of our proposed search system.

Results for simple queries (Experiment 1). In the first experiment we measured the top-1 and top-5 accuracy of our system with simple queries. The average accuracy for experiment 1 per game can be seen in Table 2, and per query in Table 4. The overall average top-1 accuracy and average top-5 accuracy for 'ViT-B/32' is 60% and 76% respectively, and for 'RN101' we have 64% and 86% respectively.

Table 3: Average top-k accuracy (%) per game for compound queries (Experiment 2).

		GTA	RDR	JC3	F4	FC5	C77	ESV	W3
ViT-B/32	**Top**-1	68	88	56	43	31	50	56	56
	Top-5	100	100	81	64	69	75	89	67
RN101	**Top**-1	84	88	31	36	56	67	33	44
	Top-5	95	100	75	79	94	83	78	56

Table 4: Average top-k accuracy (%) per query for simple queries (Experiment 1). N is the number of games searched.

		ViT-B/32		RN101	
Query	N	**Top**-1	**Top**-5	**Top**-1	**Top**-5
Airplane	4	75	100	100	100
Bear	5	80	100	60	100
Bike	4	50	75	50	100
Bridge	8	88	88	50	100
Car	5	80	100	80	100
Carriage	4	50	50	75	100
Cat	6	33	50	33	67
Cow	8	63	75	25	75
Deer	7	57	71	75	100
Dog	8	25	38	38	63
Fire	8	88	100	100	100
Helicopter	5	60	60	60	100
Horse	3	67	100	100	100
Mountain	7	100	100	100	100
Parachute	2	0	67	67	100
Ship	8	50	63	38	75
Snow	6	67	83	33	50
Tank	3	67	67	100	100
Traffic Light	5	40	40	20	20
Train	5	80	100	17	67
Truck	4	75	100	100	100
Wolf	6	17	50	86	86
Average	**5.5**	**60**	**76**	**64**	**86**

These results show that our system can identify a majority of objects in the game without fine-tuning or re-training.

Results for compound queries (Experiment 2). In the second experiment we measure the top-1 and top-5 accuracy of our approach with compound queries. The average accuracy for experiment 2 per game can be seen in Table 3, and per query in Table 5. For the second experiment, we find that our approach shows particularly high performance for all of our selected games, except for The Witcher 3: Wild Hunt. Our approach achieves an overall average top-5 accuracy of **78%** using 'ViT-B/32' and **82%** using the 'RN101' model. These results show that our approach is flexible enough to effectively search gameplay videos with compound queries.

Results for bug queries (Experiment 3). In the final experiment, we measure recall @5 of our approach with bug queries. Table 6 shows

Table 5: Average top-k accuracy (%) per query for compound queries (Experiment 2). N is the number of games searched.

		ViT-B/32		RN101	
Query	N	**Top**-1	**Top**-5	**Top**-1	**Top**-5
A bald person	8	75	88	88	88
A bike on a mountain	4	25	75	50	75
A black car	5	80	100	80	100
A blue airplane	4	25	75	50	75
A blue car	5	80	80	40	100
A brown cow	7	29	71	57	71
A brown horse	3	100	100	100	100
A car on a mountain	4	75	75	75	100
A golden dragon	2	0	50	0	50
A gray tank	3	33	67	33	33
A man on top of a tank	4	25	50	0	0
A person in a jungle	7	57	100	57	100
A person on a mountain	7	71	100	57	100
A person wearing gold	8	50	88	50	100
A person wearing purple	8	50	88	25	63
A person with a pig mask	1	100	100	100	100
A police car	3	33	67	67	100
A police officer	3	33	33	67	100
A red car	5	80	100	80	100
A white airplane	4	75	75	50	100
A white horse	3	33	67	33	67
A white truck	5	40	60	60	80
Average	**4.7**	**53**	**78**	**55**	**82**

the results for Experiment 3 for each query with each game. Our approach shows particularly high performance for Grand Theft Auto V, Just Cause 3, and Far Cry 5. The average accuracy for Experiment 3 across all 44 unique queries is 66.12% and 66.35% using 'ViT-B/32' and 'RN101' respectively. These numbers suggest that, in most cases, our approach can reliably retrieve relevant videos based on an English text query containing a description of an event. Moreover, we can conclude that contrastive pre-training methods are powerful enough to be used in the video game domain, especially for bug detection in gameplay videos.

8 DISCUSSION

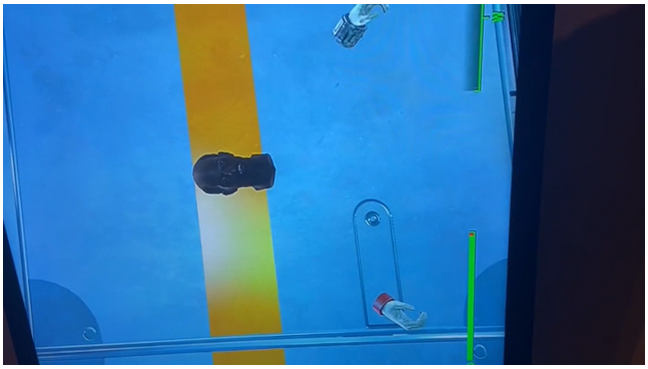
In this section, we discuss the strengths and weaknesses of our approach, based on the results of our experiments. Figure 5 shows several example video frames from videos identified when searching gameplay videos with text queries using our approach. These examples help to illustrate the promising potential of our approach. Given that our method does not require any training on gameplay videos, our zero-shot object and event identification results are promising. During our experiments, the first author manually analyzed each video returned by our search approach, including false positives. Below, the causes of false positives in our search results are detailed.

Adversarial poses. One important category of problems is the unusual pose of familiar objects. As extensively tested and reported by

Table 6: Recall @5 (%) for bug queries (Experiment 3). Queries that were not used per game are shown with values of '-'.

	ViT-B/32								RN101							
Query	GTA	RDR	JC3	F4	ESV	W3	C77	FC5	GTA	RDR	JC3	F4	ESV	W3	C77	FC5
A bike inside a car	40	-	-	-	-	-	-	-	20	-	-	-	-	-	-	-
A bike on a wall	100	-	-	-	-	-	-	-	100	-	-	-	-	-	-	-
A car flying in the air	100	-	100	40	-	-	60	80	100	-	100	40	-	-	100	80
A car on fire	60	-	80	-	-	-	60	80	60	-	100	-	-	-	80	100
A car in vertical position	100	-	100	-	-	-	80	100	100	-	100	-	-	-	80	60
A car stuck in a rock	-	-	-	-	-	-	40	-	-	-	-	-	-	-	20	-
A car stuck in a tree	60	-	40	-	-	-	-	60	100	-	60	-	-	-	-	40
A car under ground	-	-	-	-	-	-	60	-	-	-	-	-	-	-	20	-
A carriage running over a person	-	-	-	-	20	-	-	-	-	-	-	-	40	-	-	-
A dragon inside the floor	-	-	-	-	20	-	-	-	-	-	-	-	60	-	-	-
A head without a body	-	-	-	20	-	-	-	-	-	-	-	0	-	-	-	-
A headless person	-	20	-	-	-	-	-	-	-	20	-	-	-	-	-	-
A helicopter inside a car	-	-	-	-	-	-	-	20	-	-	-	-	-	-	-	40
A horse floating the air	-	-	-	-	100	-	-	-	-	-	-	-	100	-	-	-
A horse in the air	-	80	-	-	-	100	-	-	-	100	-	-	-	100	-	-
A horse in the fire	-	40	-	-	-	20	-	-	-	20	-	-	-	20	-	-
A horse on fire	-	-	-	-	20	-	-	-	-	-	-	-	20	-	-	-
A horse on top of a building	-	60	-	-	-	-	-	-	-	20	-	-	-	-	-	-
A horse to stand on its legs	-	-	-	-	-	60	-	-	-	-	-	-	-	100	-	-
A person falling inside the ground	-	-	-	20	-	-	-	-	-	-	-	40	-	-	-	-
A person flying in the air	80	100	100	60	40	100	80	100	100	100	80	100	60	100	80	100
A person goes through the ground	-	40	-	-	-	-	-	-	-	0	-	-	-	-	-	-
A person in fire	-	100	-	60	60	100	-	-	-	100	-	80	60	80	-	-
A person inside a chair	-	-	-	100	40	-	-	-	-	-	-	40	40	-	-	-
A person inside a rock	-	-	-	-	-	80	-	-	-	-	-	-	-	40	-	-
A person on the house wall	-	-	-	-	-	60	-	-	-	-	-	-	-	40	-	-
A person stuck in a barrel	-	-	-	-	60	-	-	-	-	-	-	-	40	-	-	-
A person stuck in a tree	80	-	-	-	-	40	-	-	80	-	-	-	-	40	-	-
A person stuck inside a wall	-	-	-	20	-	-	-	-	-	-	-	40	-	-	-	-
A person stuck on the ceiling	-	-	-	-	40	-	-	-	-	-	-	-	40	-	-	-
A person under a vehicle	80	-	-	60	-	-	20	-	60	-	-	60	-	-	0	-
A person under a car	60	-	-	-	-	-	-	-	60	-	-	-	-	-	-	-
A person under a vehicle	-	-	-	-	-	-	-	60	-	-	-	-	-	-	-	80
A person under the carriage	-	40	-	-	-	-	-	-	-	40	-	-	-	-	-	-
A person without head	-	-	-	20	-	-	-	-	-	-	-	20	-	-	-	-
A ship under water	-	-	-	-	-	40	-	-	-	-	-	-	-	80	-	-
A tank in the air	80	-	100	-	-	-	-	-	80	-	80	-	-	-	-	-
A vehicle inside the water	80	-	80	40	-	-	80	100	80	-	100	40	-	-	40	80
A vehicle on top of building	100	-	100	-	-	-	100	-	100	-	100	-	-	-	100	-
A vehicle on top of rooftop	60	-	80	-	-	-	-	-	100	-	80	-	-	-	-	-
An airplane in a tree	-	-	-	-	-	-	-	100	-	-	-	-	-	-	-	80
An airplane in the water	20	-	60	-	-	-	-	60	40	-	80	-	-	-	-	80
Cars in accident	100	-	60	-	-	-	100	100	80	-	80	-	-	-	100	100
Two cars on top of each other	-	-	-	-	-	-	60	-	-	-	-	-	-	-	40	-
Average	**77**	**60**	**82**	**44**	**44**	**67**	**67**	**78**	**83**	**50**	**87**	**46**	**51**	**67**	**60**	**76**

Alcorn et al. [1], neural networks occasionally misclassify objects when they have different poses than what they used to have in the training set. For example, consider a neural network that can detect a 'car' in an image. It is possible to find a particular camera angle for which the neural network can not detect the 'car' in that image. In a dataset of natural images, we might have lots of cars, but the camera angle and the position of the car relative to the camera do not vary a lot. A neural network trained on these datasets will struggle to detect a car when it sees it from a very unusual angle (e.g., when it is positioned vertically)



(a) Video of *'A head without a body'* from Fallout 4.

(b) Video of *'A person stuck in a barrel'* from The Elder Scrolls V: Skyrim.

(c) Video of *'A car in a vertical position'* from Grand Theft Auto V.

(d) Video of *'A person stuck in a horse'* from The Witcher 3: Wild Hunt.

Figure 5: Relevant gameplay videos identified using our approach with bug queries.

Confusing textures and patterns in the images. The textures and patterns can pose influential distractions and confusion for the neural network model in our approach. Sometimes a part of a game environment has a texture similar to another object. For example, our model confuses a striped colored wall in the Grand Theft Auto V game with a 'parachute.' This category of problems is hard to encounter globally because each game has a diverse look and feel and creative artistic directions.

Confusion about different types of vehicles. During analysis of the videos that contain bugs related to cars, we noticed that sometimes some of the results partially match the textual description, except we see a bike instead of a car. Through our manual evaluation of the search results, we found that the model in our approach sometimes confused cars, bikes, airplanes, tanks, and other military vehicles. An instance of this misclassification is when we search for 'A car on fire'. In some of the retrieved videos, we saw an airplane on fire instead of a car.

Confusion about different four-legged animals. After reviewing several results for queries related to animals, we found out that the model in our approach struggles to distinguish different animals. More specifically, for gameplay videos the CLIP model will confuse 'dogs', 'cats', 'deer', 'wolves', and sometimes 'cows' and 'horses' with each other. A possible remedy for this problem is getting help from a separate pre-trained animal detection model to verify the CLIP model's prediction.

9 LIMITATIONS

9.1 Adversarial samples

Every machine learning method suffers from a group of adversarial attacks and out-of-distribution samples. As described extensively in previous work [1], any data point outside the training distribution is problematic for machine learning algorithms. Similarly, we observe some cases in which the neural network model makes an incorrect prediction. In particular, our model has some difficulty making a correct guess if it saw an object in an unfamiliar or adversarial pose. Due to physical simulations in video games, these adversarial poses are prevalent.

Another observation we had is about text patches inside the games. The CLIP model has the ability to 'read' the text inside an image as well. This feature is not something that the model was explicitly trained for, but rather some emergent behavior of pre-training in a contrastive setting. Sometimes searching a particular text query will result in retrieving the video that ignores the meaning of the text query, but the image contains that text. For example, if any video frames include a text field containing 'a blue

car', searching for the query 'a blue car', will retrieve that video. Obviously, depending on the use case, this can be treated as both a feature and bug.

9.2 Improvements on search speed

In our proposed method, we calculate the embeddings of all frames in advance. With this pre-processing step, our system answers an arbitrary text query in just a few seconds. It might not be possible to perform this step in advance for some exceptional use cases. For handling such cases, there are some performance improvement techniques to run each neural network faster in inference mode, at the cost of sacrificing the model's accuracy. For example, it is possible to reduce the floating point precision of a model [12] or even binarize the entire model [22]. One simple but effective way to achieve faster runtime is to cut the last layers of the neural network gradually to reach an optimal performance vs. accuracy trade-off [56]. Using these techniques, or similar speed-up approaches, improving the presented system is possible.

10 THREATS TO VALIDITY

Threats to internal validity. Due to a lack of a benchmark dataset, we designed a set of custom queries for searching the gameplay videos in our `GamePhysics` dataset. To address potential bias when generating these queries, the first author performed a pilot analysis of 80 gameplay videos across the 8 selected games to determine relevant objects and events before we designed the queries.

In each of our experiments, we assumed that an accuracy measurement of 0% indicated that our approach failed to correctly identify any relevant videos. For example, in Experiment 3 we assumed that a recall @5 of 0% in our search results indicated that our approach failed to identify that bug query in that game. However, it could instead be the case that our dataset does not contain any videos that match the query. Without a benchmark dataset, we do not have the ground truth for whether a repository of gameplay videos contains any matches for a given arbitrary text query. This means that the reported performance values are possibly lower estimates of the actual performance.

In Experiment 3, we used our second aggregation method (Section 4.3), which involved the selection of a pool size hyperparameter. Although we selected the default value of 1,000 based on manual trial and error, different selections of this hyperparameter could lead to different results for Experiment 3. Therefore, future research is required to understand how the selection of the pool size in our second aggregation method impacts the performance of our approach.

Threats to external validity. While our dataset predominantly consists of gameplay videos that contain game physics bugs, our approach may not be as effective with other datasets of gameplay videos. Non-curated datasets may contain many more false positives (non-buggy gameplay), for example if using gameplay streaming footage. Additionally, we excluded long (>60 seconds) videos, meaning our approach may not be effective for long videos. We also ignored all videos with scores of zero from the GamePhysics subreddit. After manually checking a random sample of low-scored posts we observed that a score of 0 almost always indicated low quality/spam/etc. This threshold might not be generalizable to other subreddits. Future research is required to evaluate the performance of our approach with long videos and non-curated datasets.

11 CONCLUSION

In this paper, we proposed a novel approach to mine large repositories of gameplay videos by leveraging the zero-shot transfer capabilities of CLIP to connect video frames with an English text query. Our approach is capable of finding objects in a large dataset of videos, using simple and compound queries. Additionally, our approach shows promising performance in finding specific (bug-related) events, indicating it has the potential to be applied in automatic bug identification for video games. Even though we did not perform any fine-tuning or re-training to adapt the CLIP model to the video game domain, our approach performs surprisingly well on the majority of video games. We evaluated our system on a dataset of **6,192** videos from eight games with different visual styles and elements. When experimenting with the bug queries, we measured recall @5 and found the average accuracy of our approach across all 44 unique bug queries is **66.24%** when averaged across both of the CLIP architectures utilized in our experiments. Furthermore, our manual analysis of the search results enabled us to discuss causes of false-positives in our approach and identify several future research directions. Our approach lays the foundation to utilizing contrastive learning models for zero-shot bug identification in video games. Future work in this line of research will provide more insights into video games bugs, and will pave the way to creating a new paradigm of automated bug detection methods for video games.

REFERENCES

[1] Michael A Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh Nguyen. 2019. Strike (with) a pose: neural networks are easily fooled by strange poses of familiar objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4845–4854.

[2] Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. 2020. The pushshift reddit dataset. In *Proceedings of the international AAAI conference on web and social media*. Volume 14, 830–839.

[3] Suzanna Becker and Geoffrey E Hinton. 1992. Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355, 6356, 161–163.

[4] Joakim Bergdahl, Camilo Gordillo, Konrad Tollmar, and Linus Gisslén. 2020. Augmenting automated game testing with deep reinforcement learning. In *2020 IEEE Conference on Games (CoG)*, 600–603.

[5] Joakim Bergdahl, Camilo Gordillo, Konrad Tollmar, and Linus Gisslén. 2020. Augmenting automated game testing with deep reinforcement learning. In *2020 IEEE Conference on Games (CoG)*. IEEE, 600–603.

[6] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

[7] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 308–318.

[8] Antonio Borrelli, Vittoria Nardone, Giuseppe A Di Lucca, Gerardo Canfora, and Massimiliano Di Penta. 2020. Detecting video game-specific bad smells in unity projects. In *Proceedings of the 17th International Conference on Mining Software Repositories*, 198–208.

[9] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7, 04, 669–688.

[10] Ke Chen, Yufei Li, Yingfeng Chen, Changjie Fan, Zhipeng Hu, and Wei Yang. 2021. Glib: towards automated test oracle for graphically-rich applications. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1093–1104.

[11] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.

[12] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*.

[13] Parmida Davarmanesh, Kuanhao Jiang, Tingting Ou, Artem Vysogorets, Stanislav Ivashkevich, Max Kiehn, Shantanu H Joshi, and Nicholas Malaya. 2020. Automating artifact detection in video games. *arXiv preprint arXiv:2011.15103*.

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

[15] MohammadAmin Fazli, Ali Owfi, and Mohammad Reza Taesiri. 2021. Under the skin of foundation nft auctions. *arXiv preprint arXiv:2109.12321*.

[16] Pablo García-Sánchez, Alberto Tonda, Antonio M. Mora, Giovanni Squillero, and Juan Julián Merelo. 2018. Automated playtesting in collectible card games using evolutionary algorithms: a case study in hearthstone. *Knowledge-Based Systems*, 153, 133–146.

[17] Camilo Gordillo, Joakim Bergdahl, Konrad Tollmar, and Linus Gisslén. 2021. Improving playtesting coverage via curiosity driven reinforcement learning agents. In *2021 IEEE Conference on Games (CoG)*, 1–8.

[18] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. 2020. Bootstrap your own latent - A new approach to self-supervised learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[19] Matthew Guzdial, Nicholas Liao, and Mark Riedl. 2018. Co-creative level design via machine learning. In *Joint Proceedings of the AIIDE 2018 Workshops co-located with 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2018), Edmonton, Canada, November 13-14, 2018* (CEUR Workshop Proceedings). Volume 2282. CEUR-WS.org.

[20] Matthew Guzdial and Mark Riedl. 2016. Game level generation from gameplay videos. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

[22] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. *Advances in neural information processing systems*, 29.

[23] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 7, 3, 535–547.

[24] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, et al. 2018. Unity: a general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

[25] Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. 2019. Deep learning for video game playing. *IEEE Transactions on Games*, 12, 1, 1–20.

[26] Faraz Khadivpour and Matthew Guzdial. 2020. Explainability via responsibility. In *Proceedings of the AIIDE Workshop on Experimental AI in Games*.

[27] Nazanin Yousefzadeh Khameneh and Matthew Guzdial. 2020. Entity embedding as game representation. In *Proceedings of the AIIDE Workshop on Experimental AI in Games*.

[28] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[29] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. 2009. Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 951–958.

[30] Hugo Larochelle, Dumitru Erhan, and Yoshua Bengio. 2008. Zero-data learning of new tasks. In *AAAI* number 2. Volume 1, 3.

[31] Chris Lewis and Jim Whitehead. 2011. The whats and the whys of games and software engineering. In *Proceedings of the 1st international workshop on games and software engineering*, 1–4.

[32] Chris Lewis, Jim Whitehead, and Noah Wardrip-Fruin. 2010. What went wrong: a taxonomy of video game bugs. In *Proceedings of the fifth international conference on the foundations of digital games*, 108–115.

[33] Dayi Lin, Cor-Paul Bezemer, and Ahmed E Hassan. 2019. Identifying gameplay videos that exhibit bugs in computer games. *Empirical Software Engineering*, 24, 6, 4006–4033.

[34] Carlos Ling, Konrad Tollmar, and Linus Gisslén. 2020. Using deep convolutional neural networks to detect rendered glitches in video games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* number 1. Volume 16, 66–73.

[35] Zijin Luo, Matthew Guzdial, and Mark Riedl. 2019. Making CNNs for video parsing accessible: event extraction from dota2 gameplay video using transfer, zero-shot, and network pruning. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 1–10.

[36] Laura MacLeod, Margaret-Anne Storey, and Andreas Bergen. 2015. Code, camera, action: how software developers document and share program knowledge using youtube. In *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 104–114.

[37] Emerson Murphy-Hill, Thomas Zimmermann, and Nachiappan Nagappan. 2014. Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development? In *Proceedings of the 36th International Conference on Software Engineering*, 1–11.

[38] Alfredo Nantes, Ross Brown, and Frederic Maire. 2008. A framework for the semi-automatic testing of video games. In *AIIDE*.

[39] Luca Pascarella, Fabio Palomba, Massimiliano Di Penta, and Alberto Bacchelli. 2018. How is video game development different from software development in open source? In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 392–402.

[40] Fábio Petrillo, Marcelo Pimenta, Francisco Trindade, and Carlos Dietrich. 2009. What went wrong? a survey of problems in game development. *Computers in Entertainment (CIE)*, 7, 1, 1–22.

[41] Johannes Pfau, Antonios Liapis, Georg Volkmar, Georgios N Yannakakis, and Rainer Malaka. 2020. Dungeons & replicants: automated game balancing via deep player behavior modeling. In *2020 IEEE Conference on Games (CoG)*. IEEE, 431–438.

[42] Cristiano Politowski, Fabio Petrillo, and Yann-Gäel Guéhéneuc. 2021. A survey of video game testing. *arXiv preprint arXiv:2103.06431*.

[43] Cristiano Politowski, Fabio Petrillo, Gabriel Cavalheiro Ullmann, Josias de Andrade Werly, and Yann-Gaël Guéhéneuc. 2020. Dataset of video game development problems. In *Proceedings of the 17th International Conference on Mining Software Repositories*, 553–557.

[44] Luca Ponzanelli, Gabriele Bavota, Andrea Mocci, Massimiliano Di Penta, Rocco Oliveto, Mir Hasan, Barbara Russo, Sonia Haiduc, and Michele Lanza. 2016. Too long; didn't watch! extracting relevant fragments from software development video tutorials. In *Proceedings of the 38th international conference on software engineering*, 261–272.

[45] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event* (Proceedings of Machine Learning Research). Volume 139. PMLR, 8748–8763.

[46] Shaghayegh Roohi, Christian Guckelsberger, Asko Relas, Henri Heiskanen, Jari Takatalo, and Perttu Hämäläinen. 2021. Predicting game difficulty and engagement using AI players. *Proc. ACM Hum. Comput. Interact.*, 5, CHI, 1–17.

[47] Ronnie ES Santos, Cleyton VC Magalhães, Luiz Fernando Capretz, Jorge S Correia-Neto, Fabio QB da Silva, and Abdelrahman Saher. 2018. Computer games are serious business and so is their quality: particularities of software testing in game development from the perspective of practitioners. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1–10.

[48] Patrick Stacey and Joe Nandhakumar. 2009. A temporal perspective of the computer game development process. *Information Systems Journal*, 19, 5, 479–497.

[49] Mohammad Reza Taesiri, Moslem Habibi, and Mohammad Amin Fazli. 2020. A video game testing method utilizing deep learning. *Iran Journal of Computer Science*, 17, 2.

[50] Chintan Trivedi, Antonios Liapis, and Georgios N. Yannakakis. 2021. Contrastive learning of generalized game representations. In *2021 IEEE Conference on Games (CoG), Copenhagen, Denmark, August 17-20, 2021*. IEEE, 1–8.

[51] J Tuovenen, Mourad Oussalah, and Panos Kostakos. 2019. Mauto: automatic mobile game testing tool using image-matching based approach. *The Computer Games Journal*, 8, 3, 215–239.

[52] Simon Varvaressos, Kim Lavoie, Sébastien Gaboury, and Sylvain Hallé. 2017. Automated bug finding in video games: a case study for runtime monitoring. *Computers in Entertainment (CIE)*, 15, 1, 1–28.

[53] Markos Viggiato, Dale Paas, Chris Buzon, and Cor-Paul Bezemer. 2021. Identifying similar test cases that are specified in natural language. *arXiv preprint arXiv:2110.07733*.

[54] Markos Viggiato, Dale Paas, Chris Buzon, and Cor-Paul Bezemer. 2022. Using natural language processing techniques to improve manual test case descriptions. In *International Conference on Software Engineering - Software Engineering in Practice (ICSE - SEIP) Track*. (May 8, 2022).

[55] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 7782, 350–354.

[56] Mehrshad Zandigohar, Deniz Erdoğmuş, and Gunar Schirner. 2021. Netcut: real-time dnn inference using layer removal. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 1845–1850.

[57] Mehrshad Zandigohar, Mo Han, Deniz Erdoğmuş, and Gunar Schirner. 2020. Towards creating a deployable grasp type probability estimator for a prosthetic hand. In *Cyber Physical Systems. Model-Based Design*. Roger Chamberlain, Martin Edin Grimheden, and Walid Taha, editors. Springer International Publishing, Cham, 44–58. ISBN: 978-3-030-41131-2.

[58] Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen, and Changjie Fan. 2019. Wuji: automatic online combat game testing using evolutionary deep reinforcement learning. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 772–784.