Improving Requirement Traceability by Leveraging Video Game Simulations in Search-Based Software Engineering*

Javier Verón[1], Raúl Lapeña[1], Carlos Cetina[1], Óscar Pastor[2], and Francisca Pérez[1]

[1] Universidad San Jorge. SVIT Research Group, Zaragoza, Spain
{jveron, rlapena, ccetina, mfperez}@usj.es
[2] Universidad Politecnica de Valencia. PROS Research Center, Valencia, Spain
opastor@dsic.upv.es

Abstract. Video games pose different challenges during development and maintenance than classic software. For example, common and widespread assets, that are typically created as part of video game development are Non-Player Characters (NPCs). NPCs contribute to different aspects such as storytelling and user experience, and they are typically controlled by the CPU. We theorize that a reproduction of the actions of NPCs within the game (i.e., simulations) holds key information for Game Software Engineering (GSE) tasks such as Traceability Link Recovery (TLR). This paper presents our approach for supporting TLR in GSE by leveraging video game simulations. Simulation data from NPCs is used to reduce the search space. Since the reduced search space might still be too large for manual inspection, an evolutionary TLR procedure evolves a population of code fragments. As a result, a ranking of code fragments that map the requirement to the code is obtained. We evaluate our approach in Kromaia, a commercial video game released on PC and PlayStation 4. We compare our approach against a baseline that does not incorporate simulations by means of a statistical analysis. Our approach reduces the search space by 99.21% on average, and significantly outperforms the baseline with large differences in all performance indicators. A focus group with professional developers has confirmed the acceptance of our approach. Our work provides a new direction in TLR, which is an essential task in not only GSE but also in classic software engineering.

Keywords: Traceability Links Recovery · Video Games · Search-Based Software Engineering · Topic Modeling.

1 Introduction

Video games are complex products where art and software are combined during the development process to conform the final product. Due to their nature and

* Partially supported by MINECO under the Project VARIATIVA (PID2021-128695OB-I00), by Gobierno de Aragón (Research Group S05_20D), and by Centro para el Desarrollo Tecnológico y la Innovación (CDTI) (Reference: SAV-20221011).

the software artifacts that compose them, video games present challenges that differ from those of classic software, leading to important differences between Game Software Engineering and Classic Software Engineering [10,1]. Our hypothesis is that it is possible to leverage the particularities and specific domain assets of video games to provide semi-automated support and solutions for Game Software Engineering tasks.

To that extent, we have set our sight on one of the most common and widespread assets of video games: non-player characters (NPCs). NPCs are non-playable entities and elements (typically controlled by the CPU) that are created as part of the video game development process. Some examples of NPCs in video games are: the generals of the enemy troops in a Real-Time Strategy (RTS), or the rival drivers in a racing game.

It is important to note that NPCs are not created with the intention of performing Game Software Engineering (GSE) tasks on the game. Nonetheless, NPCs often have pre-programmed behaviors to allow for CPU control. We theorize that it is possible to observe NPC behaviors within the game (i.e., simulations) to reproduce and even generate knowledge that enables automated support and solutions for GSE tasks (e.g., testing the implementation of a specific requirement, or the solution to a known bug in the game). Specifically, we put the focus on Traceability Link Recovery (TLR). TLR is a software engineering task that deals with automated identification and comprehension of dependencies and relationships between software artifacts [20]. Establishing and maintaining traceability links has proven to be a time-consuming, error-prone, and labor-intensive task [20].

In this paper, we propose a novel approach that leverages the information provided by NPC simulations to provide semi-automated support for TLR between the main artifacts of GSE: the requirements and the code of a video game. To do this, the NPC's scenarios are reproduced to obtain simulation data, which is then used to reduce the search space to a subset of code that represents the scenario (and in turn contains the requirement). Since the reduced search space might still be too large for manual inspection, an evolutionary TLR procedure along with the requirement evolves a population of code fragments. The output is a ranking of code fragments that serve as candidate solutions towards the mapping of the requirement to the code.

We evaluate the performance of our approach considering an industrial case study, which belongs to a commercial video game, Kromaia. Kromaia is a video game about flying and shooting with a spaceship in a three-dimensional space. It was released on PC, PlayStation and has been translated to eight different languages. In addition, we compare our results with those of a baseline that does not incorporate simulations within the TLR process by means of a statistical analysis and an effect size measure. Finally, we carried out a focus group interview with the aim of acquiring qualitative data and feedback about the obtained results. Hence, the contributions of this paper are threefold:

– We investigate the use of NPC simulations within the TLR process in an industrial case study. To the best of our knowledge, this is the first effort in the

literature. This is relevant because TLR is an essential software maintenance and evolution task [4].
- We experimentally demonstrate that NPC simulations reduced the search space (by 99.21% on average). This enables our approach to significantly outperform the traceability results of the baseline with large differences in all performance indicators (by 23.23% in recall, 18.34% in precision, and 22.87% in F-measure).
- We provide evidence that software engineers prefer the results of our approach using simulations over the results of the baseline even though they have not used simulations for TLR before. In fact, software engineers valued very positively both the use and the usefulness of simulations to find requirements in other video game genres and more complex problems (e.g., fighting games).

In the remainder of the paper, Section 2 presents the background for our work. Section 3 presents our approach, and Section 4 describes the evaluation. Section 5 presents the results, and Section 6 discusses the outcomes of our work. Section 7 presents the threats to the validity. Section 8 reviews the related work. Finally, Section 9 concludes the paper.

2 Background

The case study we use to evaluate TLR with NPCs is the commercial video game Kromaia. Kromaia is a commercial video game programmed in C++, where the human player (i.e., the user) moves a spaceship in a three-dimensional space and fights different enemy elements through a collection of levels. In each level, the player must make the spaceship fly from an initial point to a destination point multiple times without being destroyed. The spaces where this action is carried out include: a scene or architecture with asteroids to avoid while flying, different improvements for the player's spaceship, and NPCs acting as enemies of the player (which will try to destroy the player's spaceship and have to be fought with projectiles that the player's spaceship can fire).

Video games can be defined as a series of requirements that define the multiple mechanics, dynamics, and situations in the gameplay. An example of a requirement of this nature in Kromaia is "When the human unit is damaged its armor level decreases and the interface shows the information". Developers often need to trace these requirements to fix errors or change some behaviors.

Kromaia, like most video games, has a significant amount of source code: a total of 145915 lines. Traceability between requirements and lines of code in a case like Kromaia would require approximately one hour per requirement. A game like Kromaia usually needs to trace five or more requirements every day. Thus, tracing the regular number of requirements in a regular working week would take 25 hours of work, which is impossible for a small studio to bear.

3 Our approach

Fig. 1 shows an overview of our approach. Our approach starts from a requirement that needs to be traced to the code of the game. Software engineers build a scenario for simulation that contains the requirement and use NPCs to reproduce the scenario. The simulation gathers data, registering the code that was executed to fulfill the scenario. The simulation data is used to filter the entirety of the code of the game, thus reducing the search space to a subset of code that represents the simulated scenario, which in turn contains the requirement. For the requirement "When the human unit is damaged its armor level decreases and the interface shows the information", a scenario must be built with a single NPC that damages the human unit. This action decreases the armor level and triggers the logic to display the change in the armor level information on the graphical user interface. The logic meeting this requirement is executed and registered. Thus, this registered code constitutes the reduced search space instead of the entire source code.

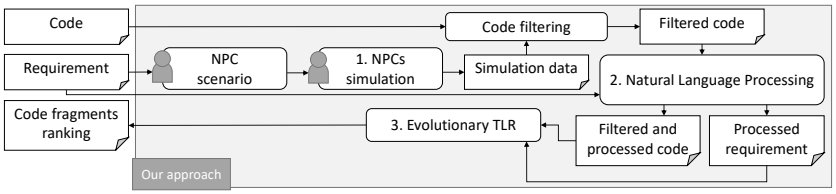

Fig. 1. Approach overview

Since the code search space for the scenario can still be too large for manual inspection, the filtered code is then processed via Natural Language Processing (NLP) techniques. Afterwards, the filtered and processed code is fed into an evolutionary TLR procedure along with the requirement, which is also processed via NLP techniques. The evolutionary TLR procedure generates, evaluates, and evolves a population of code fragments, converging into a ranking of code fragments that constitutes the final output of the approach. Each code fragment serves as a candidate solution towards the traceability of the requirement.

The ranking of code fragments is sorted according to the parameters of the TLR fitness function, which is used within the evolutionary TLR procedure. Video game software engineers can use the ranking as a starting point to determine and document the traceability, validating or enhancing the top solutions according to their knowledge of the video game.

3.1 NPC-based simulations

Non-player characters (NPCs) are video game entities that simulate human players when performing specific actions in the game through various artificial intelligence algorithms created by the developers. NPCs are developed and included

in a game and accompany the player during the game, antagonize the player, or simply populate the world recreated in the video game.

Nonetheless, NPCs and their simulation capabilities are embedded within the architecture of the game. Our rationale is that leveraging NPC-based simulations can support software engineering tasks in the game. Specifically, we aim to reproduce particular requirements with these simulations and capture the code executed to fulfill these requirements. This approach intends to identify code fragments associated with each requirement, thus reducing the search space.

To achieve this, the developers of the game set a scenario within the game, using its architecture and NPCs. The NPCs perform the actions defined in each requirement, substituting the human player when necessary. We register the executed code, creating filtered code documents for the requirements that must be traced. The filtered code documents are used as the search space for an evolutionary algorithm that performs the traceability procedure on the requirements.

3.2 Natural Language Processing

We process the input requirements and the filtered code through Natural Language Processing (NLP) techniques. This processing step serves as a means of unifying the language of the software artifacts, which facilitates the TLR process. The techniques in use are syntactical analysis, root reduction, and human NLP. Fig. 2 shows the used NLP techniques on a requirement in our approach.

1. **Syntactical Analysis:** SA techniques analyze the specific roles of each word in a sentence, determining their grammatical function. These techniques filter words that fulfill specific grammatical roles in a requirement, discarding those that do not have semantic value (such as articles or adverbs).
2. **Root Reduction:** The technique known as lemmatizing reduces words to their semantic roots (lemmas), thus avoiding verb tenses, noun plurals, and several other word forms that can negatively interfere with the TLR process.
3. **Human NLP:** Human NLP is often carried out through domain term extraction or stopword removal. Our approach searches the requirements for domain terms provided by software engineers and adds the domain terms found to the processed artifact. On the other hand, stopwords are filtered out after root reduction, using a list of stopwords that is also provided by the software engineers.

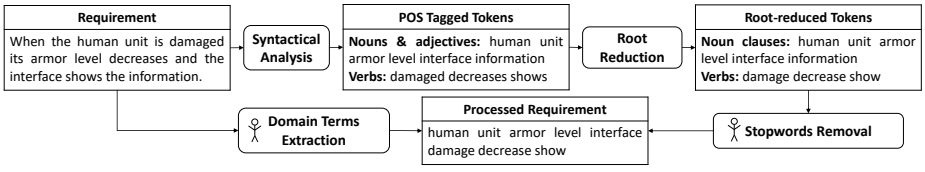

Fig. 2. Natural Language Processing Techniques

3.3 Evolutionary TLR

Fig. 3 shows an overview of the Evolutionary Algorithm that generates, evaluates, and evolves a population of code fragments, converging into a ranking of potential solutions towards the input requirement. The algorithm takes both the filtered and processed code, and the processed requirement as input, and iterates over a population of code fragments, modifying them using genetic operations until a final ranking is produced as output. More precisely, the algorithm runs in three steps:

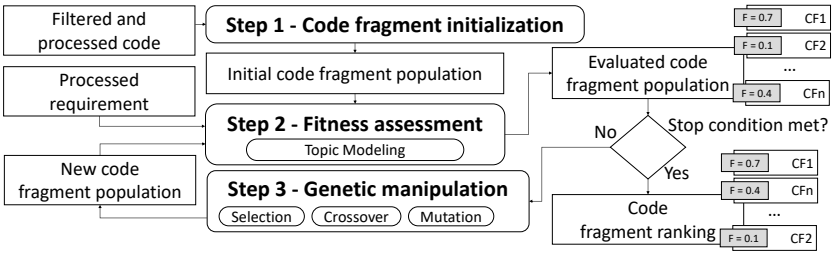

Fig. 3. Evolutionary algorithm

1. **Initialization:** The algorithm generates a population of code fragments that serves as input for the algorithm. To generate the population, parts of the code documents are extracted randomly and then added to a collection of code fragments, which are linked to the code documents they belong to.
2. **Fitness function:** This step assesses each of the candidate code fragments produced, ranking them according to a fitness function. Our approach uses Topic Modeling as fitness function because it obtained the best results when performing Feature Location, a similar Information Retrieval task to the one at hand [22].
   To do this, we utilize Latent Dirichlet Allocation (LDA) [7] since it is one of the most popular topic modeling methods [17]. LDA is an unsupervised probabilistic technique for estimating a topic distribution over a text corpus made up of a set of documents, where each document is a set of terms. As a result, a probability distribution is obtained for each document, indicating the likelihood that it expresses each topic. In addition, a probability distribution is obtained for each topic identified by LDA, indicating the likelihood of a term from the corpus being assigned to the topic.
   LDA inputs include: the documents (D), the number of topics (K), and a set of hyper-parameters. The hyper-parameters of any LDA implementation are: k, which is the number of topics that should be extracted from the data; α, which influences the topic distributions per document. A lower α value results in fewer topics per document; and β, which affects the distribution of terms per topic. A lower β value results in fewer terms per topic, which

in turn implies an increase in the number of topics needed to describe a particular document.

LDA outputs include: ϕ, which is the matrix that contains the term-topic probability distribution; and θ, which is the matrix that contains the topic-document probability distribution.

To assess the relevance of each code fragment with regard to the provided requirement (i.e., query), we use LDA. Given the terms for the query Q and the outputs of LDA (ϕ and θ), the conditional probability P of Q given a document D_i is computed as follows [5]: $Sim(Q, D_i) = P(Q|D_i) = \prod_{q_k \in Q} P(q_k|D_i)$ where q_k is the k^{th} processed term in the query, and D_i is a document (i.e., a code fragment) of the population that is made up of a set of processed terms.

Fig. 4 shows an example of the fitness assessment for each candidate code fragment (i.e., individual) from the population. Given the processed terms of the requirement (query) and each code fragment (represented as a column in the matrix of the figure), the figure depicts how a value is assigned per term in each code fragment of the population (values in the cells of the matrix) by using the LDA outputs (the ϕ and θ matrices). For example, the conditional probability for the processed term *human* given the code fragment CF_n, that is, $P(human|CF_n) = 0.39$ (as can be seen in the shaded cell in the matrix of Fig. 4). Using the values of the matrix, the fitness value is obtained for each code fragment as shown in the vector of Fig. 4. The highest fitness value in the vector of the figure is 0.41, which belongs to CF_2.

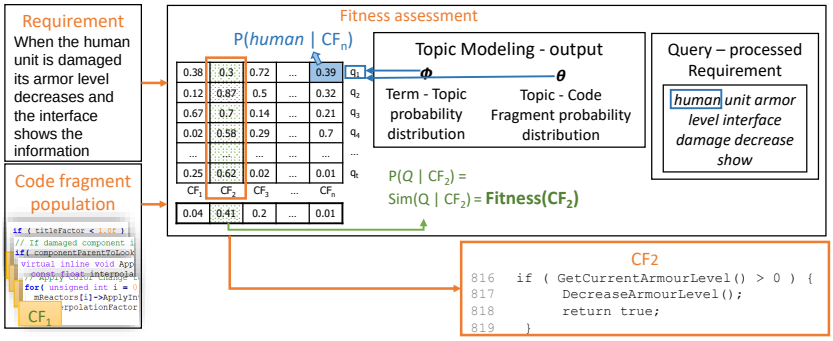

Fig. 4. Example of the fitness assessment of code fragments

3. **Genetic manipulation:** Finally, if the solution does not converge, this step generates a new population of code fragments through genetic manipulation. The generation of new code fragments, based on existing ones, is done by applying a set of three genetic operations (selection, crossover, and mutation), which are taken from the literature [6]:

The **selection operation** picks candidates from the population as input for the rest of the operations. Candidates with high fitness values have higher probabilities of being chosen as parents for the next generation. The **crossover operation** enables the creation of a new individual by combining genetic material from two parent code fragments. The **mutation operation** imitates mutations that randomly occur in nature when new individuals are born, by adding or removing code lines from the fragment.

Step 2 (fitness assessment) and Step 3 (genetic manipulation) are repeated until the solution converges to a certain stop condition (e.g., number of iterations). When this occurs, the evolutionary algorithm provides a code fragment ranking, ordered according to the values determined by the fitness function.

Similarly to other works that retrieve text from an initial query using LDA or other information retrieval techniques, the results depend on the quality of the queries [5], which is typically improved through an iterative refinement process [16]. Therefore, even when irrelevant code fragments are obtained in the ranking, the results can be considered as a starting point for the iterative refinement process. From there, software engineers can either manually modify the proposed ones or modify the requirement to automatically obtain different code fragments for the solution.

4 Evaluation

4.1 Research Questions

From our work, several research questions arise:

RQ$_1$ *Does the inclusion of simulations influence the traceability search space in Game Software Engineering?*

RQ$_2$ *What is the performance in terms of solution quality of the traceability results in Game Software Engineering for our approach using simulations and a state-of-the-art technique?*

RQ$_3$ *Are the differences in performance between our approach using simulations and the baseline significant?*

RQ$_4$ *How much do the simulations influence the quality of the solution compared to the baseline?*

4.2 Experimental Setup

The inputs to build the test cases are the requirements and the code. These test cases are then employed to perform TLR in both our approach and the baseline. The baseline is similar to our approach and it also uses topic modeling as a fitness function (as described in Section 3.3), but, unlike our approach, it does not integrate simulations into the TLR process and it does not reduce the search space. Finally, a measurements report is obtained after comparing the approved

traceability that is provided as input (used as oracle) with the results of the baseline and our approach.

The case study that we used in the evaluation is Kromaia, a commercial video game which code has been provided by our industrial partner, Kraken Empire[3]. Kromaia is made up of 145915 lines of code. In total, 18 requirements with their corresponding oracles and simulation scenarios were provided by the company. The provided requirements were selected and provided by the developers, who considered those requirements as being a representative collection in terms of maintenance. The approved traceability for each requirement (i.e., oracle) is the set of code lines (i.e., code fragment) that correspond to the full coverage for a requirement in the traceability search space. The code fragment that corresponds to each requirement has between 1 and 43 code lines.

As a result of comparing the approved traceability with the results of the baseline and our approach, a measurements report is obtained to answer each research question as follows.

Answering RQ_1: To determine whether the inclusion of simulations influences the traceability search space in GSE, the evaluation starts by taking the inputs of the case study (requirements, code, and simulation scenarios and approved traceability) to obtain the set of methods that are called during the simulations scenarios for each test case. This set of methods will conform the traceability search space for each test case. Afterwards, it is possible to assess whether the size of the traceability search space has been reduced, and if so, by how much.

In addition, the traceability search space is compared to the approved traceability to determine whether the traceability search space limits the quality of the solution by excluding code lines that are in the approved traceability.

Answering RQ_2: To determine the performance in terms of solution quality for each test case in our approach and the baseline, we compare the code fragment that achieves the highest fitness score in our approach and in the baseline against its respective oracle (i.e., ground truth) to calculate a confusion matrix.

A confusion matrix is a table that is often used to describe the performance of a classification model, comparing a set of test data (the solutions provided by the approach) against a set of known true values (the solutions from the oracle). In our case, the solutions obtained by the approach are code fragments, which are composed of a subset of code lines that are part of the original code. Since the granularity is at the level of code lines, the presence or absence of each code line is considered as a classification. The confusion matrix distinguishes between the predicted values (i.e., solution of the approach) and the real values (i.e., solution of the oracle), classifying them into four categories: (1) True Positive (TP), values that are predicted as true and are true in the real scenario; (2) False Positive (FP), values that are predicted as true but are false in the real scenario; (3) True Negative (TN), values that are predicted as false and are false in the real scenario; and (4) False Negative (FN), values that are predicted as false but are true in the real scenario.

[3] `https://www.krakenempire.com/`

From the values in the confusion matrix, it is possible to derive a series of performance measurements. Specifically, we report three performance measurements that are widely accepted by the software engineering community [25]: recall, precision, and F-measure. Recall ($\frac{TP}{TP+FN}$) measures the number of elements of the solution that are correctly retrieved by the proposed solution (how many code lines from the oracle are present in the retrieved code fragment). Precision ($\frac{TP}{TP+FP}$) measures the number of elements from the solution that are correct according to the ground truth, i.e., the oracle (how many code lines from the retrieved fragment appear in the oracle). Finally, F-measure ($2 * \frac{Precision * Recall}{Precision + Recall}$) corresponds to the harmonic mean of precision and recall.

For each requirement in both the baseline and the approach, we executed 30 independent runs (as suggested by Arcuri and Fraser [3]): 18 (requirements) x 2 (baseline and approach) x 30 repetitions, for a total of 1080 independent runs.

Answering RQ$_3$: To determine whether the differences in performance between our approach and the baseline are significant, the results must be properly compared and analyzed using statistical methods. With the aim of providing formal evidence that the differences do in fact have an impact on the performance measurements, we follow the guidelines presented in [2]. The statistical test that must be followed depends on the properties of the data, and it is accepted by the research community that a *p-value* under 0.05 implies statistical significance [2]. For each performance measure a *p-value* is recorded.

Answering RQ$_4$: To determine how much the performance is influenced by using our approach compared to the baseline, it is important to assess whether our approach is statistically better than the baseline, and if so, to measure by how much. To do this, we use Cliff's delta [11], which is an ordinal statistic that describes the frequency with which an observation from one group is higher than an observation from another group compared to the reverse situation. It can be interpreted as the degree to which two distributions overlap, with values ranging from -1 to 1. For instance, when comparing distributions of the treatment and the control, a value of 0 means no difference between the two distributions, a value of -1 means that all of the samples in the distribution of the treatment are lower than all of the samples in the distribution of the control, and a value of 1 means the opposite. In addition, threshold values can be defined [24] for the interpretation of Cliff's delta effect size as *negligible* ($|d| < 0.147$), *small* ($|d| < 0.33$), *medium* ($|d| < 0.474$), and *large* ($|d| \geq 0.474$). A Cliff's delta value is recorded for each pair-wise comparison between our approach and the baseline for each performance measure.

4.3 Implementation details

For the development of our approach and the baseline, we used Eclipse with Java. For the NLP operations used in both our approach and the baseline, we used the OpenNLP Toolkit [14].

Topic Modeling was implemented using the Collapsed Gibbs Sampling (CGS) for LDA because it requires less computational time [27], and it was previously used for locating features in source code [5].

Since the focus of this paper is not to tune the values to improve the performance of our approach when applied to a specific problem, we used default values from the literature [6,22,13]. As suggested by Arcuri and Fraser [3], default values are sufficient to measure the performance of search-based software engineering algorithms.

For the evaluation, we used an Asus ROG Strix G15 G512LW-HN038 laptop with an Intel(R) Core(TM) i7-10750H processor, 16GB RAM, an NVIDIA® GeForce® RTX 2070 graphics card, and Windows 11 (64-bit). The oracle was provided directly by the developers of the video game.

The CSV files used as input in the statistical analysis as well as an open-source implementation of the approach are available here: `https://github.com/VeronLinks/simulations-ss-tlr`

5 Results

5.1 Research Question 1

After obtaining the set of methods that are called during simulation scenarios for each test case, the traceability search space has been reduced from 145915 code lines to an average of 1157 code lines. Afterwards, the traceability search space is checked to determine whether any of the code lines that are included in the oracle have been omitted. If no code line has been omitted, the quality of the solution of our approach is not limited after reducing the search space.

RQ_1 answer: The inclusion of simulations has been shown to significantly reduce the search space for TLR in GSE by 99.21% on average. We have found that the inclusion of simulations does not limit the quality of the solution.

5.2 Research Question 2

Table 1 shows the mean values and standard deviations of recall, precision, and F-measure for our approach and the baseline. As the values show, our approach outperforms the baseline in all performance measurements.

Table 1. Mean values and standard deviations for recall, precision, and the F-measure

	Recall ± (σ)	Precision ± (σ)	F-measure ± (σ)
Our work	65.58 ± 24.32	76.36 ± 14.90	67.65 ± 15.43
Baseline	42.35 ± 25.71	58.02 ± 8.35	44.78 ± 16.84

RQ_2 answer: The results reveal that our approach outperforms the baseline in the three performance measurements. Hence, the inclusion of simulations improves the quality of the traceability results in GSE when using a state-of-the-art technique by 23.23% in recall, 18.34% in precision, and 22.87% in F-measure.

5.3 Research Question 3

The *p-values* obtained for our approach are 0.0037 for recall, 9.4^{-6} for precision, and 4.1^{-5} for F-measure.

RQ_3 answer: Since the Quade test *p-values* are smaller than the 0.05 statistical significance threshold for all performance measurements, we can state that there are significant differences in performance between our approach and the baseline for TLR in GSE.

5.4 Research Question 4

With regard to how much the simulations influence the quality of the solution compared to the baseline, the obtained Cliff's Delta values of the three reported performance measurements are large, being 0.5 for recall, 0.7623 for precision, and 0.6512 for F-measure.

RQ_4 answer: From the effect size analysis of the Cliff's Delta values that are obtained when our approach is compared to the baseline, we can conclude that there is a large influence on the quality of the solution for all performance measurements with our approach using simulations for TLR in GSE.

6 Discussion

Industrial video games feature many requirements involving multiple methods, but they also bring opportunities to TLR. Simulations, scarce in traditional software, are common in video games due to NPCs being created during the development. By analyzing the results, the noise that the simulations remove by reducing the search space helps the evolutionary TLR procedure to work significantly better than when using the complete code.

At least in the case study that was used in this work, there is no limitation in the quality of the solution (recall) when the reduced traceability search space is obtained. Nevertheless, our approach should be replicated with case studies from other domains before assuring the generalization of the results. Nevertheless, our results suggest that leveraging video game simulations for TLR is promising.

Our focus group, consisting of five software engineers in the field of GSE, included one with 15 years of video game development experience, two with six years, and two with two years of experience. This focus group dealt with the following open questions: (1) What do you think of the results of the approaches?; (2) How do you feel about locating requirements in video games using simulations?; (3) How do you imagine locating requirements using simulations in video games of other genres and in more complex video games?

The software engineers valued very positively the results of our approach using simulations, and stated that they preferred our approach using simulations because the results were far superior to the results of the baseline. Although none of the five engineers had thought of or used simulations for TLR before, they agreed on the relevance of simulations for TLR in video games. In their

opinion, the idea of using simulations for TLR is promising, and they would use simulations for TLR in video games on a daily basis.

The engineers imagined using our approach in another of their video games since they could also use simulations of NPCs to find requirements that give them trouble. In addition, the engineers confirmed that our approach using simulations can be used for other video game genres and more complex problems.

7 Threats to validity

In this section, we acknowledge the limitations of our approach using the classification of threats to validity proposed by Wohlin et. al. [26]:

Conclusion validity: To minimize this threat, we have based our work on research questions. In addition, the requirements and code used in our approach were taken from an existing commercial video game. None of the authors of this work were involved in the creation of the game or in the generation of the data, which was provided by the developers of the game.

Internal Validity: To minimize this threat, we have used the same NLP techniques on all of the software artifacts and followed the same evaluation process for all evaluated approaches. In addition, the available test cases represent a broad scope of different scenarios within the case study in an accurate manner.

Construct validity: To minimize this threat, our evaluation was performed using three measurements that are widely accepted and utilized in the state-of-the-art literature: precision, recall, and F-measure. Moreover, we have used the same software artifacts for all of the approaches, representing the same scenarios, so that cause and effect are equally represented among them.

External Validity: This threat is concerned with generalization. All of the artifacts under study in our work (requirements and code) are frequently used in video games development. In addition, TLR is a common practice in all kinds of video game development scenarios and the real-world case study used in our research represents the industrial scene of video game developments well. Moreover, we defined our approach independently of the case study, and then applied it to the case study. Hence, our approach can potentially work in any video game scenario where requirements and code are available. Nevertheless, our results should be replicated with other case studies before ensuring their external validity.

8 Related work

After reviewing a traceability survey that contains works from 1999 to 2011 [8], we used the same query presented in the survey, and we found 11 more works between 2012 and November 2023. For example, Dekhtyar et al. [12] assess the accuracy of provided Requirements Traceability Matrix (RTM) by means of an assessment committee composed of three to five different trace recovery methods, such as Probabilistic Information Retrieval (ProbIR). Parvathy et al. [21]

propose automating the computation of traceability by looking at the correlation between documents. Gethers et al. [15] propose the the Relational Topic Model for TLR and combine different orthogonal IR-based methods for improving recovery accuracy. These three approaches require additional information sources beyond source code, such as RTM, design documents, and domain models that support traceability. However, in the development of commercial video games, creating or synchronizing these artifacts is often neglected. Therefore, approaches like this work that focus on searching for solutions in terms of code fragments in the source code are necessary in this context. Lapeña et al. [19,18] leverage an already existing feature of the software artifacts to improve TLR using BPMN models and leveraging on execution traces. None of these works use simulations to reduce the search space.

We also performed a manual search and we only found three studies on locating artifacts in video games for GSE. Two of the three studies focus on bug location using simulations, but neither reduces the search space as we propose. Blasco et al. [6] focus on TLR for video games and use an evolutionary algorithm for obtaining a code fragment from the source code that realizes a requirement specified in a natural language. Casamayor et al. [9] use simulations in video games and leverage the existence of NPCs in video games to remove any additional cost for the developers, as we do, but they do not use these simulations to reduce the search space. Prasetya et al. [23] present an agent-based automated testing framework for locating bugs in video games.

The rationale behind using simulations is that video game bugs are not always related to code malfunction, making traditional testing and bug location inefficient. Video games bugs include other aspects related to design, such as balancing certain values. Testing and bug location in video games need to be reactive and able to interpret their environment to determine what actions to take. This rationale differs from the rationale behind using simulations in this work. We leverage these simulations for performing automated and precise executions of the program, and registering the executed code. Thus, the search space for TLR can be reduced with the goal of improving the quality of the results.

9 Conclusion

In this work, we have focused on GSE and TLR. GSE is a very novel field of work that deals with the challenge of developing and maintaining the software of video games, whereas TLR is a key support activity for the development, management, and maintenance of software. Video games not only bring challenges to TLR, but they also provide opportunities such as the information that can be extracted from the simulations of NPCs, which are created as part of the video game.

This information had not yet been exploited for either reducing the search space or for TLR. We have filled this gap, and we have shown that the search space can be reduced and that TLR can be significantly improved by leveraging video game simulations in a commercial video game. Specifically, compared to the baseline, the search space was reduced by 99.21% on average, and our ap-

proach improved the results with large differences (by 23.23% in recall, 18.34% in precision, and 22.87% in F-measure). A focus group has demonstrated that software engineers prefer the results of our approach using simulations over the results of the baseline. The software engineers have also highlighted the usefulness of simulations to find requirements in other video game genres and more complex problems (e.g., fighting games).

Our work opens a new direction in TLR, which is an essential task in both GSE and in classic software engineering. Other essential maintenance tasks such as bug location (our future work) can also benefit from the use of simulations.

References

1. Ampatzoglou, A., Stamelos, I.: Software engineering research for computer games: A systematic review. Information and Software Technology **52**(9), 888–901 (2010)
2. Arcuri, A., Briand, L.: A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Softw. Test. Verif. Reliab. **24**(3), 219–250 (May 2014). https://doi.org/10.1002/stvr.1486
3. Arcuri, A., Fraser, G.: Parameter tuning or default values? an empirical investigation in search-based software engineering. Empirical Software Engineering **18**(3), 594–623 (2013). https://doi.org/10.1007/s10664-013-9249-9
4. Ballarín, M., Arcega, L., Pelechano, V., Cetina, C.: On the influence of architectural languages on requirements traceability. Software: Practice and Experience **53**(3), 704–728 (2023). https://doi.org/https://doi.org/10.1002/spe.3166, https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3166
5. Biggers, L.R., Bocovich, C., Capshaw, R., Eddy, B.P., Etzkorn, L.H., Kraft, N.A.: Configuring latent dirichlet allocation based feature location. Empirical Softw. Engg. **19**(3), 465–500 (Jun 2014)
6. Blasco, D., Cetina, C., Pastor, Ó.: A fine-grained requirement traceability evolutionary algorithm: Kromaia, a commercial video game case study. Information and Software Technology **119**, 106235 (2020)
7. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. Mach. Learn. Res. **3** (2003). https://doi.org/http://dx.doi.org/10.1162/jmlr.2003.3.4-5.993
8. Borg, M., Runeson, P., Ardö, A.: Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering **19**, 1565–1616 (2014)
9. Casamayor, R., Arcega, L., Pérez, F., Cetina, C.: Bug localization in game software engineering: evolving simulations to locate bugs in software models of video games. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems. pp. 356–366 (2022)
10. Chueca, J., Verón, J., Font, J., Pérez, F., Cetina, C.: The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games. Information and Software Technology (2023)
11. Cliff, N.: Ordinal methods for behavioral data analysis. Lawrence Erlbaum Associates, Inc (1996)
12. Dekhtyar, A., Hayes, J.H., Sundaram, S., Holbrook, A., Dekhtyar, O.: Technique integration for requirements assessment. In: 15th IEEE International Requirements Engineering Conference (RE 2007). pp. 141–150. IEEE (2007)

13. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Achieving feature location in families of models through the use of search-based software engineering. IEEE Transactions on Evolutionary Computation **PP**(99), 1–1 (2017)
14. Foundation, T.A.S.: Apache opennlp: Toolkit for the processing of natural language text. `https://opennlp.apache.org/` (2020)
15. Gethers, M., Oliveto, R., Poshyvanyk, D., Lucia, A.D.: On integrating orthogonal information retrieval methods to improve traceability recovery. In: IEEE 27th International Conference on Software Maintenance, ICSM 2011, Williamsburg, VA, USA, September 25-30, 2011. pp. 133–142 (2011)
16. Hill, E., Pollock, L., Vijay-Shanker, K.: Automatically capturing source code context of nl-queries for software maintenance and reuse. In: Proceedings of the 31st International Conference on Software Engineering. pp. 232–242. ICSE '09 (2009)
17. Jelodar, H., Wang, Y., Yuan, C., Feng, X.: Latent dirichlet allocation (LDA) and topic modeling: models, applications, a survey. Multimedia Tools and Applications **78**, 15169–15211 (2019). `https://doi.org/10.1007/s11042-018-6894-4`
18. Lapeña, R., Pérez, F., Cetina, C., Pastor, Ó.: Leveraging bpmn particularities to improve traceability links recovery among requirements and bpmn models. Requirements Engineering pp. 1–26 (2022)
19. Lapeña, R., Pérez, F., Pastor, Ó., Cetina, C.: Leveraging execution traces to enhance traceability links recovery in bpmn models. Information and Software Technology **146**, 106873 (2022)
20. Oliveto, R., Gethers, M., Poshyvanyk, D., De Lucia, A.: On the equivalence of information retrieval methods for automated traceability link recovery. In: IEEE 18th International Conference on Program Comprehension. pp. 68–71 (2010)
21. Parvathy, A.G., Vasudevan, B.G., Balakrishnan, R.: A comparative study of document correlation techniques for traceability analysis. In: International Conference on Enterprise Information Systems. vol. 3, pp. 64–69. SCITEPRESS (2008)
22. Pérez, F., Lapeña, R., Marcén, A.C., Cetina, C.: Topic modeling for feature location in software models: Studying both code generation and interpreted models. Information and Software Technology **140**, 106676 (2021)
23. Prasetya, I., Pastor Ricós, F., Kifetew, F.M., Prandi, D., et al.: An agent-based approach to automated game testing: an experience report. In: Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation. pp. 1–8 (2022)
24. Romano, J., Kromrey, J.D., Coraggio, J., Skowronek, J.: Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In: annual meeting of the Florida Association of Institutional Research. pp. 1–33 (2006)
25. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill, Inc., New York, NY, USA (1986)
26. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science & Business Media (2012)
27. Xuan-Hieu Phan, C.T.N.: Jgibblda. a java implementation of latent dirichlet allocation (LDA) using gibbs sampling for parameter estimation and inference (2020)