MASTER IN COMPUTER SCIENCE PROFESSIONAL FOCUS IN DATA SCIENCE

The Coding of Isaac

Leveraging elements of video games to convey information about software quality and technical debt in the context of software visualization

GRABSI, Aniss

Award date:
2022

Awarding institution:
University of Namur

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

The Coding of Isaac: Leveraging elements of video games to convey information about software quality and technical debt in the context of software visualization

Anthony BAYET
Aniss GRABSI

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Abstract

Software quality is an important element to manage in a development project in order to avoid ever increasing refactoring costs. Visualizing the current quality state of their code is one common way to help developers in this task. We believe that a video game-based visualization tool can help developers become more engaged and get a better overall picture of their code quality in a novel way. Which brings us to these two questions: How intuitive and accurate is the video game medium to visualize the software technical debt for industry practitioners and academics? How useful is the video game in visualizing the technical debt of software in the field? These questions are explored in this work through the development of a video game prototype and its evaluation through 12 semi-structured interviews and surveys. Players easily understood the relationship between the game and the metrics. The visual elements we designed and the overall structure of the game were intuitive to users. There is still room to improve the game to make it more engaging and useful to developers, it needs more gamification elements. The video game medium is an intuitive way to convey information about software quality and technical debt, although more gamification would improve it.

Keywords: Technical debt, Video game, Software visualization, Software quality

Acknowledgment

We would like to thank our supervisor Prof. Benoît Vanderose from the university of Namur, our co-supervisor Prof. Fabian Gilson from the Canterbury Christ Church University, Te Whare Wānanga o Waitaha, and our internship supervisor, Dr. Zadia Codabux from the university of Saskatchewan for their guidance throughout this project.

We would also like to thank all the people who participated in the pilot tests and interviews conducted as part of this research.

Contents

1 Introduction

Khomh et al. [2009] showed that classes with code smells, i.e., classes showing signs of poor code quality, are more prone to change than classes of better quality. Fontana et al. [2013] and Zazworka et al. [2011] then confirmed that software quality is correlated with the number of code smells a software presents, forcing developers to devote time to changing, modifying, and fixing code more often when it contains code smells than when it does not. Zazworka et al. [2011] states "god classes", which are classes that do to many functions, have been confirmed to have the technical debt property of incurring interest since they “cost” the organization more in terms of changes and defects as the software development lifecycle progresses". These previous works have shown and demonstrated that it is important to manage this technical debt to avoid increasing interest payments(cost to fix the technical debt later). One of the first activities to be undertaken to manage debt is to let developers know it exists (Codabux et al. [2014]). So it is important to visualize technical debt and have tools that can track and provide insight into it. This led this study to the world of code quality visualization, and more precisely to the possible innovations in this field using video games.

Games are not just entertainment, Dörner et al. [2016] explains that games can have a more serious purpose in addition to the entertainment part. This causes a compromise between entertainment and serious purpose, but it is manageable. Doherty et al. [2018] states that games can then be used for eight different purposes, namely: education, entertainment, exercise, meditation, party and social. Our research explores a new goal which is the visualization of code quality in software projects. Tekinbas and Zimmerman [2003] discusses that games can be representations by themselves, which is the reason for the existence of our research.

Software is a product that can become incredibly complex. In this case, it can be very difficult to estimate the cost of the quality of the software accurately. Software quality is a broad topic, so our research focuses on one specific element: technical debt and code level quality, ignoring other elements such as dependencies. To help practitioners in this task, many code quality tools are available, such as Embold[1] or SonarQube[2]. However, while these tools can provide detailed measurements of bugs, code smells, and other code quality metrics(e.g. number of line of codes, coupling), they have trouble solving directly the problem of visualizing the software quality. Developers are overwhelmed by the amount of information about the quality of their code and must try to make sense of it on their own.

One way to overcome this issue is to use software visualization. This field of computer science focuses on the visualization of computer programs and their behavior. It is an important part of computer science, as it plays a significant role in the development and understanding of software.

The problem is that most research in this area still focuses on an overly grounded approach. Most visualizations are analytical and precise, but none have attempted to translate the analytical data into a more intuitive instrument. This general impression would allow developers to quickly identify the most relevant attributes of a program and easily assess its quality without having to read detailed tables of statistics and numbers on their code.

We propose to use a video game as a medium to depict software quality and use the game mechanics to visualize software code-level quality. To do this, a game based visualization called The Coding of Isaac has been created. It allows the player to navigate the code quality of a codebase to get a general idea of the amount of technical debt present in the codebase. The prototype was evaluated using surveys and interviews to answer two main research questions(RQ):

[1] `https://embold.io/`
[2] `https://www.sonarqube.org/`

RQ1: How intuitive and accurate is the video game medium to visualize the software technical debt for industry practitioners and academics?

RQ2: How useful is the video game in visualizing the technical debt of software in the field?

Following our evaluation, our results are promising. The players played the game without any issues and were able to understand the relationship of the game and the metrics. The visual elements (e.g. monsters, walls, floors), and overall structure of the game were intuitive to users. However, there is still room to improve the game to make it more engaging and useful to developers in their daily workflow by incorporating more gamification aspects.

To summarize the contribution of this thesis:

Contribution 1: A description of the Dungeon Crawler game genre by breaking down each of the elements that make up this type of game in detail.
Contribution 2: A proposed mapping between code quality metrics and video game components based on the application of the theory of measurement by Allen [1979] to metrics of code quality and to the components of video games.
Contribution 3: A prototype game[3] designed to visualize code quality in software development projects. Along with detailed information about how the game was designed and implemented.
Contribution 4: An evaluation of this prototype with field practitioners and students to determine if software professionals are interested in incorporating our tool into their daily workflow?

[3]Source code available on `https://github.com/snail-unamur/Yo-kai-watch`

2 Background

In this chapter we will define terminologies, present general design aspects considered for a video game, describe the dungeon crawler genre being the one chosen for this study and its main game components. Then, we look at what is code quality and the main code metrics. Finally, this chapter ends with a description of measurement theory and its role in this type of mapping task. It is important to note that the items presented in the following sections are not an exhaustive list of items in terms of code quality metrics or game components. The design elements included in the prototype developed during this study will be taken from these lists.

2.1 Video Game

2.1.1 Game elements

Werbach and Hunter [2012], define the categories of elements that make up a video game. They are either a component, a mechanic, or a dynamic. Game dynamics are the overall picture of the game (e.g., emotion, progression, narrative). Game mechanics are the processes that make the player move forward (e.g., chance, cooperation, competition, feedback). Finally, game components are the specific objects that bring the mechanics and dynamics to life (e.g. quests, levels, leaderboards, visual elements, game rules).

2.1.2 Gamification

According to Deterding et al. [2011], gamification is the practice of taking methodologies or game elements and applying them to non-game applications and processes to improve the user experience. Indeed, as mentioned in the introduction, we are taking game components and applying them to the world of technical debt management and visualization.

2.1.3 Interactions

As mentioned earlier, the centerpiece of this study is the video game that aims to help developers visualize code quality metrics. The product has two different aspects: a visualization tool and a video game. Therefore, users will interact with both parts of the product. The interactions can be of two types, visualization-related or game-related interactions. For the first type, according to Dimara and Perin [2020] when interacting with a visualization tool, the interaction will fall into one of these seven categories: Input data (e.g., adding, creating, modifying, correcting data), Processing data (e.g., compiling, filtering, selecting a subset, deriving from raw data), Mapping data (e.g., changing the layout of data, adding visual markers mapped to variables), Presentation data (e.g., navigating, styling, highlighting, and so on), and Output data (e.g., creating, editing, and so on). navigate, style, highlight), Meta (e.g., undo, redo, save actions), Social (e.g., communicate with peers, share results), Interface (modify the user interface, e.g., add notes to data items, open, close, or modify interface panels). For the second type, the game-related interactions, a multitude of interactions belong to this category, e.g. fighting enemies, navigating in the game environment, interacting with non-player characters, etc. Some interactions will only fit into one category, but sometimes an interaction will impact both the visualization and the gameplay aspect of the product.

2.1.4 Active or passive

A major aspect in the design of a video game for this research is whether it is an *active* or a *passive* game. An active game is defined in this study as a game in which the player performs at least some game-related interactions, the player is an actor in the game. On the contrary, in a passive game, the player only performs visualization-related interactions, the player is an observer of the game. In both cases, the game represents the quality of the code and the instance of the game will be different based on that, the difference being that in the passive game, the user interacts with the product as a tool rather than a game, while in the active game, the player must perform game-related interactions to get complete information about the quality of the code.

All game genres are by default active games, as one of the main characteristics of video games is to make players take action. However, at first glance, some genres seem easier to turn into passive games than others. For example, there are "idle" games: games of this kind ask the player to interact just a little bit from time to time, and then wait to see the results. Since the number of interactions is very limited, it is easy to imagine that they could be replaced or generated procedurally. Nevertheless, any kind of game can be turned into a passive game. To do so, it is necessary only to have the players' interactions replaced and generated by some artificial intelligence.

2.1.5 Offline or real-time

Visualization can have two types of rendering mode, offline and real-time. Offline means that the visualization is not immediately updated, the user must wait for the data to be processed for a certain period of time before accessing the latest status. In the case of real time, every time the underlying data is updated, the visualization is updated at the same time (e.g. a developer has pushed a new commit).

2.1.6 Dungeon crawler genre

The game developed in this study is a dungeon crawler. The choice of the game genre will guide the subsequent steps. According to Brewer [2016] The dungeon crawler genre was born with the famous board game "Dungeons and Dragons" (D&D) by Gary Gygax and Dave Arneson in 1970. Later, with the progression of computer technology, some video games began to offer dungeon crawler elements similar to those of D&D, such as combat and character evolution. It was with pedit5, also known as The Dungeon, written in 1975 by Rusty Rutherford, that the Dungeon crawler genre really began to take shape, it allowed you to play and explore a dungeon with your character and its D&D-like statics such as hit points, strength, dexterity, etc. Video games and board games that feature primarily dungeon crawling elements are considered to be belonging to the Dungeon crawler.

In this work, we will sometimes refer to the game as the "dungeon crawler". In addition to this definition, this genre is often used combined with the "roguelike" genre which is still according to Brewer [2016] a category of games using random generation of mazes, monsters, and loots. Furthermore, the game sessions of a dungeon crawler game can be independent and short, in opposition to scenario-based game genres where the player follows a story well-defined. There exist some dungeon crawler scenario-based games (e.g. *Path of Exile*[4])that do not rely on level

[4] `https://www.pathofexile.com/`

generation and independent game session. However, games like *The Binding of Isaac*[5] and *Hadès*[6] show the potential of the dungeon crawler genre with level generation and short, mostly independent game sessions.

A deeper dive into the finer elements that make up a dungeon crawler is made in the next section, so that we can map them later to the SonarCloud code metrics.

2.1.7 Game components

A multitude of "dungeon crawler" type games have been developed over the years. For example, on *Steam*[7], the popular video game distribution platform, there are 2,158 games with labelled "Dungeon Crawler" in the best-seller category[8]. And some of them have unique gameplay elements, for example, in the popular game *The Binding of Isaac*, the player encounters two types of rooms, *Angel rooms* and *Devil rooms*, which are very specific gameplay elements of this game, another example is the gods encounter in *Hadès*, from time to time the player interact with a Greek god and talk with them. These two examples illustrate why it would be complicated to create an exhaustive list of dungeon crawler game components. Instead, in the following sections of this study, only the most common dungeon crawler game components will be presented.

Appearance The appearance of the elements that the player can see is a variable composed of multiple visual variables. These elements are typically the player character, monsters, weapons, projectiles, power-ups, obstacles, rooms, walls, doors, and user interface elements. The visual variables of these elements are hue, saturation, luminance, opacity, size and the sprite used.

Moreover, the interactions between the visual elements are very important, in this context we will talk about contrast, for example if two elements have the same color and are overlapping, they will not be distinguishable.

Hit box The size of the elements is visual, but it can also be "physical" in the game using a hit box. "A hit box is an invisible shape commonly used in video games for real-time collision detection; it is a type of collision box[9]". In most cases, the visual and physical sizes are related because the collision box is invisible, so if the game wants to be fair to the player, it tries to match the graphics with the collision box, so that the players can understand what's in the collision box, and they don't have to guess it.

In the game, the hit box can prevent the player from going through a small door, making a large monster easier to hit or a small monster harder to hit. Much of the gameplay uses the hit box of different objects in the game.

Movement speed It is the speed of the elements (player, monsters, projectile). It can be discrete with categories, like walk and run modes. Or, the speed can be continuous, like a specific amount N of pixels per second.

[5] `https://store.steampowered.com/app/113200/The_Binding_of_Isaac/`
[6] `https://www.supergiantgames.com/games/hades`
[7] `https://store.steampowered.com/`
[8] `https://store.steampowered.com/tags/en/Dungeon+Crawler#p=0&tab=TopSellers` accessed 16 May 2022.
[9] `https://en.wikipedia.org/wiki/Collision_detection#Hitbox` accessed 9 May 2022.

In the game, it may be necessary to reach a certain threshold of movement speed to perform certain actions. For example, running fast enough may allow you to "run on water" and reach new places. Or you may need a certain speed to be able to dodge certain enemy attacks.

However, elements that are too fast would be impossible for the user to perceive or react to. Conversely, an element moving too slowly would make the movement impossible to perceive, and if it is the speed of the player character that is too slow or too fast, it could make the game unplayable.

Friction force As in real life, when an object is in motion, it does not stop immediately. On the contrary, it will keep a certain speed which will decrease with time because of the friction force. This variable is very specific, it applies to any moving element and is composed of two elements. First, the force that determines at what speed the element will stop moving. Secondly, the mathematical formula slowing down the element which can be a linear deceleration of N pixels/seconds or any other formula.

Damage Damage in a game can be inflicted or taken by the player, monsters, traps or scenery. It can be displayed in several ways, for example as a floating number indicating the amount of damage taken. Also, the affected element can have an animation to show it, possibly with different animations depending on the amount of damage taken. Additionally, there can be a visual effect of the "hit", like a small explosion. Also, the game object taking the damage can make a sound when hit.

Attack speed The attack speed determines how many attacks per second an entity can perform.

Trajectory The projectiles follow a specific trajectory in a game, it can be linear, curved, spiral, etc. The trajectory is composed of a type of trajectory and depending on this type, a quantitative value can be attached to it to describe the angle for example.

Health The health points of the player or the monsters. As with damage, health information can be displayed in several ways. First, health points can be displayed as a simple number or as symbols indicating the exact number of health points. Second, it can be a health bar that can be filled up more or less to represent the amount of health left. Third, the appearance of the entity can describe its health level, for example it can bleed or look dirty. Finally, the monster's skills or behavior may depend on its health. This concerns especially the bosses, when the fight can be long, the boss will have different phases, in each phase it will use different attacks (e.g. in the first phase it will use one sword and in the second phase two swords). Also, some games like *Minecraft*[10] do not even indicate the health of some monsters. In these games, players have to keep in mind how much life the monster has left depending on the damage it has taken.

If the health is represented by elements like numbers, symbols or a health bar it can be positioned at multiple places. The player's health can be in the user interface or floating near the player's character. For monsters, most of the time it is displayed near to them floating around.

Finally, multiple ways to represent health of an entity can be used together to have some redundancy.

[10] `https://www.minecraft.net`

Environment In a dungeon crawler, the environment is the location of the player character, monsters, traps, obstacles and access to other rooms. The location and number of these elements, as well as the layout and size of the rooms, are very important to the gameplay.

In addition, the environment can have special effects, for example, there can be fog or more or less light to annoy the player. The "fog of war" is also an element that can be used to modify the environment, reducing what the player can see outside a certain distance from their position. Finally, the environment can positively or negatively affect the characteristics of the player or the monsters, such as increasing the damage dealt or reducing the movement speed of the monsters.

Link between rooms Dungeon Crawler games are usually divided into several separate rooms that are connected by doors or teleporters. Access to other rooms from a specific room is crucial in these games. For example, if there is a boss fight, the exits may be locked if the boss is alive, or the player may have to pay with some in-game currency to open a door.

Others As explained in the introduction of this section, it is not possible nor very interesting to try to create an exhaustive list of gameplay elements that can be found in a dungeon crawler because there are already enough with the list presented here. Nevertheless, a list is presented below of some additional elements of dungeon controls considered not relevant enough to be exploited as a visualization variable for the rest of this work.

- Attack range.
- Defense.
- Height of elements like obstacles and jump height.
- Resources (e.g. bomb, keys, coins in "The Binding of Isaac").
- Critical chance and damage multiplier.
- Items to pick up.
- Type of damage (ice, fire, wind, slash, piercing, etc.).
- Knock back inflicted and resistance.
- Number of ammo remaining.
- Projectiles bounce number.
- Teleporters to skip rooms.
- Number of life (you are allowed to die and resurrect on place a number of time).
- Percentage of the map discovered.
- Combination of items giving super-powers.
- Distance in which monsters can sense the player.
- Game specific elements (e.g. probability of devil or angel rooms in *The Binding of Isaac*, artifacts and identity of the god giving the power ups in *Hadès*, etc.)

2.1.8 Technologies used

For the prototype, Phaser 3[11], a javascript game engine is used to create a web browser game. Express[12], a framework built on top of Node.JS[13] is also used to create a backend server. The frontend, built with Phaser 3, also needs to be served by a server, which is usually done by software such as Nginx[14] or Apache httpd[15]. However, in this study, we tested the prototype

[11] `https://phaser.io/phaser3`
[12] `https://expressjs.com/`
[13] `https://nodejs.org/`
[14] `https://www.nginx.com/`
[15] `https://httpd.apache.org/`

using the development web server provided with Phaser 3. The game is then built and served using Docker[16], a tool that permits an easier building and deployment of apps. The rationale behind these choices is described in detail in Section 7 Implementation.

2.2 Code Quality and Technical Debt

This work is focused on visualization of code quality and technical debt. Behutiye et al. [2017] define technical debt (TD) as "a metaphor used to communicate the consequences of poor software development practices to non-technical stakeholders." This means that the accumulation of all issues raised by low quality eventually makes the software unstable, prone to vulnerabilities and bugs, and less maintainable. Developers will eventually have to improve the software by fixing the issues or refactoring the code, therefore paying off the debt.

Code quality is the quality of the software at the code level. It is possible to assess the quality of a code base from the information provided by code metrics. Code metrics are measures that give developers insight into the code they are reading. These metrics can range from the simple number of lines of code to the coupling between classes to the depth of the inheritance tree. There is virtually no limit to the number of metrics one can collect on a source code. In their systematic mapping study, Nuñez-Varela et al. [2017] found a total of 190 different code metrics for object-oriented programs. And after classifying them into 27 categories, we can see that metrics can support a wide variety of tasks such as quality, failures, complexity, cohesion, change, maintainability and many more.

As specified in more details in Section 2.2.1 SonarCloud, this study uses SonarCloud, let's dive into the metrics they use for their product. The tool offers a complete documentation[17] presenting the different metrics used to characterize a project. These are listed in Table 1

Sonarcloud is based on three elements that define the quality of a code: bugs and vulnerabilities, which are simple in the sense that the more bugs or vulnerabilities and the more severe they are, the worse the code is in terms of quality, because bugs make the software unstable and vulnerabilities make it insecure and prone to attacks. The last element is code smell. A code smell is the consequence of a bad piece of code. Usually, when a code metric finds a problem in the code, that problem is called a code smell. These three elements combined are used to evaluate the quality of the code.

Metric label	Description
Cyclomatic Complexity	Complexity of the flow of the program e.g. keywords incrementing the complexity in Java: if, for, while, case, catch, throw, &&, ||, ? from 1 to infinity e.g. 18 - 924 - 1092
Cognitive Complexity	How hard it is to understand the code's control flow[18].
Duplications	Number of duplicated blocks of lines.
Issues	Number of issues raised. Can be bugs, code smells or vulnerabilities.

[16] `https://www.docker.com/`

[17] `https://docs.sonarcloud.io/digging-deeper/metric-definitions/` accessed 10 February 2022.

[18] Additional details on the calculation method can be found on `https://www.sonarsource.com/resources/white-papers/cognitive-complexity/`

Maintainability	Metric related to the technical debt of a project based on the code smells identified. This metric calculates the technical debt using the SQALE index.
Reliability	Metric related to the reliability of the code base based on the number of bugs and their severity.
Security	Metric related to degree of security of the code base based on the number of vulnerabilities and their severity
Size	Can be different kind of size-related metrics such as: the number of line of codes, comments, classes, files, directories, etc.
Tests	Test-related metrics such as the number of failed unit tests, code coverage, integration tests, etc.

Table 1: SonarCloud metrics.

2.2.1 SonarCloud

In this work, the focus is not on finding a new method for collecting code quality metrics, but only on innovating their visualization. Thus, a quality analysis tool was needed as a basis for collecting software quality metrics to feed the visualization that was developed. To this end, SonarCloud was chosen. This is a cloud-based software quality management tool developed by SonarSource SA. This company also owns SonarQube, which is the self-hosted version. The two tools share, according to Cameron [2020], "the same underlying static analysis engine to detect bugs, vulnerabilities and code smells and generate valuable code quality metrics." They both provide a comprehensive interactive dashboard to explore a specific code base and gain insight into its quality. In this thesis, SonarCloud is used exclusively, as it provides a way to explore open source repositories and their quality data through an API, which was convenient for conducting our experiment.

2.3 Measurement theory

The previous sections have listed general software quality metrics and dungeon crawler's game variables. Mapping them will be one of the next steps, but before we dive in, we will build the theoretical foundations in order to constrain the possible links between the two sets and to make sure that they are rigorously chosen.

2.3.1 Theory

This section is based on the work of Allen [1979] for whom this is a more specific subfield called "Scaling theory" whose "main goals [are] to produce good scales". This reference will be used to find the scale of the metrics and game variables we have. We will then be able to characterize the scales and know which ones can be mapped together.

In the measurement theory there are five scales: nominal, ordinal, interval, ratio, and absolute. The **nominal** scale is used to classify items into distinct categories. This scale has no relationship between two distinct items, other than equality and category belonging, unlike the **ordinal** scale. It differs from the nominal scale by the ordering of the items, which makes it possible to calculate the median of a set of items. The **interval** scale has the same characteristics as the previous ones, in addition it has a distance between two elements which allows to calculate many more statistical values like the average, the correlation, etc. Next, the **ratio** scale has an an additional element, the absolute zero which means the absence of the measured phenomenon. Finally, we get the **absolute** scale which refers to any direct count of things (for example, the number of lines of code).

In this work, the absolute scale will be referred to as the "highest level" of scale and the nominal scale as the "lowest level" of scale. With respect to scale transformation, a higher scale can always be converted to a lower scale, the side effect being loss of information. But, it is not possible to do the reverse without refining the information and adding relationships between elements. For example, to transform a number of lines of code (absolute scale) into an ordinal scale, one can set thresholds at a specific quantity of lines of code and define three categories as follows:

- If the software product has more than 50 millions lines of code, then it will be put in a category labelled "Big projects".
- If the product has less than 50 millions lines of code and more than 10 millions, then it will be put in the category "Medium projects".
- Finally, if it has less than 10 millions lines of code, then it will be put in the category "Small projects".

Now that the theoretical framework has been described, it will be applied on the previously elicited elements to find their scale type. First for the SonarCloud code quality metrics and then for the game components.

2.3.2 Application to quality metrics

The software quality metrics proposed by SonarCloud and presented in Section 2.2 Code Quality and Technical Debt are analyzed using measurement theory. The type of scale associated with

each metric is summarized in the "Scale" column of the Table 2. We observe that many of them are on an absolute scale, this is understandable because in this context most metrics are pure counts of elements.

Metric label	(Details)	Values	Scale
	Cyclomatic Complexity	Number	Absolute
	Cognitive Complexity	Number	Absolute
Duplications	Density	Ratio in % of duplicated lines	Absolute
	Duplicated lines	Number	Absolute
	Duplicated blocks	Number	Absolute
	Duplicated files	Number per directory	Absolute
Security Review	Review Category	Category name	Nominal
	Priority	Low, Medium, High	Ordinal
	Assignee	Username	Nominal
	Status	To review, Reviewed as safe, Reviewed as fixed	Ordinal
	Rating	A, B, C, D	Ordinal
	Reviewed	%	Absolute
Size	New Lines	Number	Absolute
	Lines of Code	Number	Absolute
	Lines	Number	Absolute
	Statements	Number	Absolute
	Functions	Number	Absolute
	Classes	Number	Absolute
	Files	Number	Absolute
	Comment Lines	Number	Absolute
	Comments	%	Absolute
Tests	Coverage	%	Absolute
	Lines to Cover	Number	Absolute
	Uncovered Lines	Number	Absolute
	Line Coverage	%	Absolute
	Conditions to Cover	Number	Absolute
	Uncovered Conditions	Number	Absolute
	Condition Coverage	%	Absolute

Issues	Number of issues	Number	Absolute
	Type	Vulnerability, Bug, Code smell	Nominal
	Severity	INFO, MINOR, MAJOR, CRITICAL, BLOCKER	Ordinal
	Status	Open, Confirmed, Resolved, Reopened, Closed	Nominal
	Resolution	When an issue is closed it can be: Fixed or Removed	Nominal
	Date	Date	Absolute
	Time to fix	Number	Absolute
	Assignee	Username	Nominal
	Tags	Tags label	Nominal
Maintainability	Number of code smells (per file and directory)	Number	Absolute
	Technical debt	Number of mins, hours, ...	Absolute
	Debt ratio	%	Absolute
	Rating	A, B, C, D, E	Ordinal
	Effort to reach an A rating	Number of mins, hours, ...	Absolute
Reliability	Number of bugs (per file and directory)	Number	Absolute
	Rating	A, B, C, D, E	Ordinal
	Remediation effort	Number of mins, hours, ...	Absolute
Security	Number of vulnerabilities (per file and directory)	Number	Absolute
	Rating	A, B, C, D, E	Ordinal
	Remediation Effort	Number of mins, hours, ...	Absolute

Table 2: Quality metrics framed in measurement theory.

2.3.3 Application to game variables

To analyze the game variables, we cannot use the same methodology as for the quality measures. This is because SonarCloud measurements as described formally in the documentation[19] are purely numerical or textual. For game variables, most of the time they will be more visual and uncountable even if they are sometimes presented redundantly with a more traditional numerical value. Users will usually have to interpret the data without knowing their exact amount.

Thus, it is considered that users will measure the game variables at their nominal or ordinal value. For example, a player cannot know the exact amount of health left with a health bar only with their eyes, they could measure it with a ruler, but this use case is not considered plausible. As opposed to a clear specific amount of health such as 89/100.

[19] `https://docs.sonarcloud.io/`

Furthermore, depending on the details of the game design, some scales may vary. For example, the size of the environment could be on an absolute scale instead of an ordinal scale if the player can discriminate the exact size of the room without additional tools, if the tiles are large enough, they can count them.

Finally, Table 3 summarizes the result of applying measurement theory to game variables. As discussed, people cannot accurately measure most game variables just with their eyes. Therefore, in our context, many of the game variables will be measured by players on ordinal scales. Although they can be transformed into measurements on absolute scales if their exact values are displayed on the screen.

Elements	(Details)	Value	Scale
Appearance	Color hue	#000000 - #ffffff	ordinal
	Color saturation	%	ordinal
	Color luminance	%	ordinal
	Opacity	%	ordinal
	Skin	Dragon, Demon, Goblin, ...	nominal
Size (Hitbox and appearance)		Small, Medium, Large	ordinal
Movement speed		Slow (swimming, crawling), Normal (walking), Fast (flying, running)	ordinal
Friction force	Strength	Low (slippery), Normal, High (glue)	ordinal
	Mathematical Formula	Linear, Exponential, ...	nominal
Damage	Floating number	Number	absolute
	Colored number	Text color	ordinal
	Sized number	Text Size	ordinal
	Animation	"Ouch!", "took a lot of damage Aw", "took little damage"	ordinal
Health	Numbers	Number	absolute
	Symbols	Number of symbols	absolute
	Bar fulfillment	Filling level of the bar	ordinal
	Appearance	cf. Appearance	ordinal
	Skill set, behavior	e.g. Boss phases	ordinal
Attack speed		Slow, Normal, Fast	ordinal
Environment	Size	Small, Medium, Big	ordinal
	Type	Snow, Desert, Jungle, ...	nominal
	Number of elements	Number of monsters, obstacles, doors	absolute
	Layout	Simple, Complex, Square or L shape, ...	nominal
	Hide/conceal	Fog of war, lighting (more or less)	ordinal

Link between rooms		Doors (locked or not), teleporters	absolute
Trajectory	Type	Linear, Curvy, Spiral	nominal
	Strength	Angle, Curve strength (e.g. steep, gentle)	ordinal

Table 3: Game components framed in measurement theory.

2.4 Conclusion

The purpose of this step was to define and describe each term or concept used in this paper, as well as to define the theoretical basis used to construct the experiment. Based on measurement theory, we associated each item with the type of scale of their measurement. This shows that quality measures are primarily measured on absolute scales and game variables on ordinal scales. As explained in the introduction, this means that quality measures will be representable by game variables because absolute scales are higher level scales than ordinal scales. Thus, the transformation is allowed, but it involves a loss of information. Since mapping is theoretically possible and these foundations are established, after presenting the context and goal of the research, the Section 6 Design will discuss a mapping between code quality measures and game variables.

3 Related Work

The state of research on technical debt (TD) is quite broad, according to Behutiye et al. [2017], they identified five research areas of interest related to the TD literature in agile software development (ASD). Of these areas, "technical debt management in ASD" was found to draw the most attention, followed by "architecture in ASD and its relationship to technical debt." Furthermore, eight categories addressing the causes and five categories addressing the consequences of TD in ASD were identified. "Emphasis on speed of delivery" and "architecture and design issues" were among the top causes of TD in ASD. "Reduced productivity", "system degradation", and "increased maintenance costs" are major consequences of time loss in ASD. In addition, they found 12 TD management strategies in the ASD context, of which "redesign" and "improving TD visibility" were the most important.

In the area of software visualization, two of the most popular tools are Gource by Caudwell [2010] and Code City by Wettel and Lanza [2007]. Gource is a visualization tool that allows you to see the evolution of a software project as it unfolds in a video animation; you can see the developers interacting with the files represented by nodes. Code City, on the other hand, revolves around the metaphor of the city. Classes are blocks in the city and can grow larger or smaller depending on the number of attributes and methods they contain.

In addition, many dashboards, as shown in Figure 1, exist on the market such as SonarQube[20] and its cloud-based version SonarCloud[21], Embold[22], Duecode[23], etc. These dashboards give a lot of information about the quality of a project's code, such as the number of bugs, code smells or vulnerabilities, but can be quite tedious to use. Interpreting the information they provide can also be quite difficult.

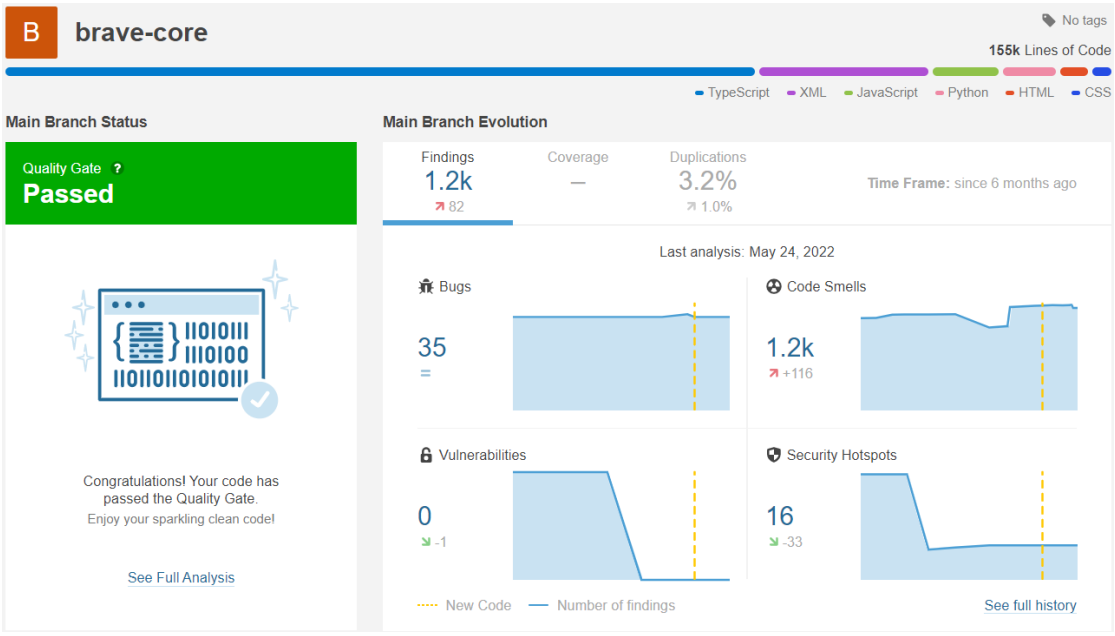

Figure 1: SonarCloud's dashboard of *Brave* (web browser).

[20] https://www.sonarqube.org/
[21] https://sonarcloud.io/
[22] https://embold.io/
[23] https://duecode.io/blog/code-quality-dashboard/

The use of a game related tools and elements for visualization is not a common practice but not new either. Indeed, Friese et al. [2008] had already discussed the pros and cons of using game engines for scientific purposes through three tools. Pérez et al. [2015] presents another tool, "SweetUnityMol: A video game-based computer graphic software", and Boeykens [2011] discusses the use of game engines for architectural historical reconstruction. In addition, Reina et al. [2020] discusses multiple elements of the video game industry that could benefit the visualization field, the paper shows that there are ready-made building blocks for visualization purposes built directly into Unity[24], one of the most famous game engines.

Moreover, some research focuses on the use of games for software visualization. Khaloo et al. [2017] presented "Code Park", a tool that places the player in a 3D game-like environment to visualize a project's code base in a more engaging and intuitive way. In addition, Balogh and Beszedes [2013] conducted a study on using an existing game to visualize the code base of a software project instead of developing a new one from scratch. They exploited the features of the game *Minecraft* to create a Code City-like tool in the game. In particular, the game graphics to create the visualization and potentially the multiplayer functionality to add a way to collaborate within the visualization.

Merino et al. [2017] takes it a step further and turns the Code city 3D environment into "City VR", a similar tool that also depicts classes but uses virtual reality (VR), allowing the user to interact with the visualization in novel ways. Their goal is to "maximize user engagement" using this device as they state it being "an interface analogous to computer games". Romano et al. [2019] compared the effectiveness of a 3D game versus a virtual reality game for software visualization and concluded that virtual reality significantly improved the accuracy of solving program comprehension tasks. Furthermore, when performing these tasks, participants using the city metaphor displayed in an immersive virtual reality were found to be significantly faster than those viewing with the city metaphor on a regular computer screen. Finally, Oberhauser and Lecon [2017] also worked on software visualization using VR. They built multiple metaphors exploiting VR, such as a universe where planets represent classes, bubble terrestrial cities where each city is a java package. However, the emphasis here is on education through entertainment, which is called edutainment. Their game "DepEnd" aims to motivate players to become familiar with source code dependency structures. And their other game "BLong" helps them to get familiar with the modularization of a code project and remember its structural modularization.

But, most of the work on game-based tools is not for software visualization purpose, instead they are focused on edutainment tools aimed to help the learning process of computer science concepts, by taking advantage of the engaging effect of video games, like Yohannis and Prabowo [2015], Oberhauser [2016], Oberhauser and Lecon [2017] and Raab [2012].

Finally, most of the research presented so far seems to be content with using video game production tools such as 3D game engines, virtual reality game engines to create software visualizations. Or, when they do implement video game elements, it is to maximize user engagement and increase their motivation to learn computing concepts. After scouring the literature, Balogh and Beszedes [2013] is the closest to the goal of the study presented here, but they primarily use the Minecraft environment to replicate Code City instead of diving into all the gameplay elements the game offers. To our knowledge, no work has looked at using video games to convey information about code quality in a software development project. The only source found referring to the use of game elements for this purpose is a blog post by an SAP employee Wulff [2020] that presents the idea of a software visualization tool that would be a video game where the game elements are actually mapped to the software in question, allowing the player to understand the quality status of their project while playing. To conclude this section, Table 4 summarize the

[24] `https://unity.com/`

main differences between the solution proposed in this thesis and the current state of the art.

Our solution	**Classic software visualization tools**	**Dashboards**	**Gamified software visualization tools**
Allows for an in-depth use of the game components offered by the gamified approach as a conveyor of information.	/	/	Uses video game to maximize user engagement and increase their motivation.
Allows the user to visualize the software along with its current state of quality	Allows the user to visualize the software without any insight on the quality	Gives insights on the software and its quality without a visualization	Doesn't offer any insight on the quality or is focused on edutainment
Goal: Offer an overview of the code quality of a project using the video game medium	Goal: Represent the software through a visualization	Goal: Giving detailed data on the quality of a software	Goal: Offer a novel way of visualizing the software or educate through entertainment

Table 4: Our solution compared to the current state of the art.

4 Research questions

A thorough analysis of the state of the art leads to the conclusion that there are currently no research focused on leveraging video game components to convey information in software visualization tool for software development projects. Outside of the literature, in a blog post, an SAP consultant and developer posted by Wulff [2020], talks about his "dream" of a game that would make reading code metrics easier. This game would belong to the city builder genre where the player must create and manage a city. The metaphor of a city in the context of software visualization is clearly inspired from Code City from Wettel and Lanza [2008], where each building is a class or a file, and its size can be the number of lines of code or its complexity. In addition, in their version of the game the author imagines game-related features such as the state of the buildings that could represent the overall quality of a class. This shows that there might be an interest in a visualization tool leveraging game components, and also that there is room for research on this topic in the literature. Therefore, the objective of this thesis is to determine if a video game can be a medium for conveying information about code quality and technical debt in the context of a code quality visualization tool. Our objective led to the following research questions.

RQ1: How intuitive and accurate is the video game medium to visualize the software technical debt for industry practitioners and academics?

Research sub-question 1.1 clarifies what is meant by "accurate," and question 1.2 defines "intuitive." Combining both sub-answers yields the answer for **RQ1**.

- **RQ1.1**: How accurately can the developers distinguish between different aspects of technical debt in the game? For example, if a file contains many vulnerabilities but no bugs, does the player understand that the code base is vulnerable but not buggy? How precise is this understanding?
- **RQ1.2**: What game mechanics can be used to allow the player to naturally relate to the underlying code quality metrics? The objective is to find game mechanics and code quality measures that, put together, allow the player to understand the connection with little or no explanation.

Hypothesis 1: We expect that it is both possible to find a good mapping between the game and the software code metrics and that this mapping is made very intuitive as defined in **RQ1.2** so that developers understand it quickly.

RQ2: How useful is the video game in visualizing the technical debt of software in the field?

Similar to research question 1, the usefulness is defined and will be answered by the combination of the four sub-questions below.

- **RQ2.1**: How much does it encourage developers to repay some debt?
- **RQ2.2**: To what extent can the tool be used by other types of users than those for whom it was originally designed, namely developers?
- **RQ2.3**: How often would this tool be used in the workflow of a target user?
- **RQ2.4**: In which context would this tool be used?

Hypothesis 2: We expect this tool to be very useful for developers to get quick overall insight on the quality of their code.

5 Methodology

To answer the research questions and make a relevant contribution to the field of software visualization, the following approach is taken:

First, an exploratory study is conducted on the vast literature of software visualization, gamification, gamified software visualization, technical debt and code quality. A thorough analysis of the state of the art leads to the conclusion that there is no research focused on the use of video games as software visualization for development projects.

Secondly, the expected contribution to the current literature is defined. The general idea is to test if the idea of having a video game as a medium to better understand the technical debt and code quality of a software development project is relevant.

Third, research questions are defined with the objective of evaluating the quality, usefulness and relevance of such a contribution. As described in more detail in the Section 4 Research questions.

To effectively answer the research questions, a prototype game visualizing the code quality of software projects is developed. It is designed by creating a mapping between code quality metrics and game components. The prototype could have been developed using two opposing approaches.

On the one hand, the game could use an existing game as a base and modify it by adjusting its components based on a mapping to code quality metrics. The main advantages of this approach are that the experience of the selected game would be proven based on its success in the market, and development would be accelerated based on the ease with which the game can be modified. Conversely, there are two major drawbacks. First, the potential limits of game modification, in fact some games are built to be modified by the community but still there are always inherent limits to a software product based on its architecture and design. Second, the rights to use a proprietary commercial game may be limited.

On the other hand, the prototype can be developed from scratch. This involves more development work and less certainty about the quality of the game, but it gives more possibilities to implement any desired functionality and avoid limitations of the right to use. Thus, this research follows the second approach to have the potential to develop any functionality and counterbalance the development burden and quality risk of the game by keeping the game design as simple as possible.

The game will be classified as an active dungeon crawler according to the definitions mentioned in the Section 2.1 Video Game. Initially, two game ideas were considered.

The first idea is a passive city building game which is a real-time visualization in which the city, buildings, roads, residents are generated based on the quality metrics of the software project code. This game would be used by developers while they are coding, since it is a passive game, the player does not need to play it to get the information, so they can keep it on the side, take a look at the information when they need it and visualize the impact of their changes in real time. If they need more clarification or other information about the visualization, they can interact with the visualization aspect of the tool since it is a passive game that only allows for visualization-related interactions.

The second idea is an active offline visualization game of the "dungeon crawler" kind in which the different components of the game are generated based on code quality metrics. This game concept would be used by developers as an independent work activity to visualize the quality of the code base.

For this research, the second game concept was chosen. The choice of an active game was made because it is closer to the common definition of a video game than its passive opposite. It allows experimenting with the use of game-related interactions to convey information in a visualization tool. In addition, its active aspect aims to engage the user more in its use. Moreover, this aspect is in good synergy with the offline aspect. Indeed, since the game is intended to be used as a standalone activity and not while doing something else, such as coding, the visualization can be offline, which reduces the performance constraints of the real-time alternative. But on the other hand, it does not support any real-time functionality.

Finally, when it comes to the game genre, there are many possibilities. The dungeon crawler is particularly appropriate for two main reasons. First, as presented in the Section 2.1 Video Game, some dungeon crawler games have already shown the potential of procedurally generated levels. Since the goal of this research is to generate a visualization that is a video game based on code quality metrics, choosing a game genre that is suitable for procedural level generation is very convenient. On the other hand, since this tool is supposed to be used to visualize the quality of a code base from time to time, it is interesting to have a game that can be played in short independent sessions. This would not be possible with a scenario-based game where the sessions would be linked by a story that would evolve over time.

In short, the prototype developed in this work is a brand new offline active dungeon crawler game.

Coming back to the research methodology, once the prototype is developed and functional, it is evaluated through 12 interviews and questionnaires. The results are then presented and discussed to conclude on possible future research and the threats to the validity.

6 Design

The goal of this research is to determine the usefulness of a video game for visualizing the technical debt and code quality of a software program, as well as the intuitive and accurate nature of this medium. The first steps were to explore the literature, state the exact research questions, enumerate the various components of a dungeon crawler game and the code quality metrics of interest, and dive into the measurement theory to derive general guidelines in order to frame the mapping of the two parts. This section is devoted to practice, mapping the game components to the code quality metrics and designing a dungeon crawler game that will then be evaluated to answer the research questions.

The game design phase in this work is divided into two parts. The first step is a preliminary ideation phase aimed at generating a first draft design. Then, this is used by the authors to further develop the overall game and its components. In both parts, the authors need to match some code metrics to game components. As mentioned earlier, this research uses measurement theory to support this activity. It is used as a constraint when mapping elements that states that any code quality metric can only be mapped to a game component of lower or equivalent scale. This constraint is taken into account by the authors throughout the design of the game.

First, for the preliminary ideation phase, the two authors work separately for 15 minutes. Both of them, equipped with paper and pens, have to make a first drawing of the game as complete as possible. The quality of the drawing does not matter, as long as the designer can use it to formulate his or her own idea of the game. When the time is up, the authors meet, present their own ideas and discuss them. Then they work together to select the best elements of each concept to create a common preliminary design that will serve as the basis for the final design. So, once this basic foundation is defined, the authors imagine and discuss additional concepts to add until the game design feels *complete*. It is complicated to specifically and objectively define what a *complete* game design is. In this study, the authors stop adding elements to the game when the number of mapped metrics is satisfactory and the game is complete enough to be playable. The minimum playable dungeon crawler consists of rooms, ways for the player to move between them, enemies that appear in those rooms, and a way for the player to fight them.

The preliminary results of the ideation phase are not discussed here. Instead, this section will directly present the complete game design. Before doing so, a few general points must be considered.

First, the aesthetic design of the game will be limited by the tile set selected by the authors. A tile set is an image file containing the different elements used in a 2D video game, the elements are framed by an atomic unit, which are squares of a defined size. In the context of this research, since the authors are not designers, they cannot create a tile set from scratch or customize it extensively. The one selected here was created by Robert, alias 0x72, who offers it for free use on `itch.io` [25]. This tile set uses tiles with 16 pixels per side.

Second, the overall structure of the game screen is composed of two elements. A heads-up display (HUD), also called status bar[26], which is fixed on the screen and displays information to the player. The other element is called the playing field in this research, it is the actual view of the game that is central and follows the player character.

Finally, several code quality metrics were presented in the Section 2.2 Code Quality and Technical Debt but only some of them will be included in the visualization to avoid overloading it. The

[25] `https://0x72.itch.io/dungeontileset-ii` accessed 10 February 2022
[26] `https://en.wikipedia.org/wiki/HUD_(video_gaming)` accessed 26 May 2022

metrics were selected during the game design when an interesting mapping was found by the authors. The Table 5 summarizes the selection.

Metric label	(Details)	Values	Scale
Size	Files and Folders	Number	Absolute
Issues	Number of issues	Number	Absolute
	Type	Vulnerability, Bug, Code smell	Nominal
	Severity	INFO, MINOR, MAJOR, CRITICAL, BLOCKER	Ordinal
	Time to fix	Number	Absolute
Maintainability	Number of code smells (per file and directory)	Number	Absolute
	Rating	A, B, C, D, E	Ordinal
	Technical debt	Number of mins, hours, ...	Absolute
Reliability	Number of bugs (per file and directory)	Number	Absolute
	Rating	A, B, C, D, E	Ordinal
	Remediation effort	Number of mins, hours, ...	Absolute
Security	Number of vulnerabilities (per file and directory)	Number	Absolute
	Rating	A, B, C, D, E	Ordinal
	Remediation Effort	Number of mins, hours, ...	Absolute

Table 5: Quality metrics used in The Coding of Isaac.

The goal of the design phase is twofold: to map the selected code quality metrics to the elements of the dungeon crawler game so that the latter represent the former, and to have a playable dungeon crawler game. This means that in order to have a complete dungeon crawler game, some elements will be added to the game even if they do not represent any quality metrics of the code. The goal is to develop a minimum playable dungeon crawler game in order to have the smallest possible core. Then, only the minimal number of elements required to consider a game as a dungeon crawler is considered. As defined above, a minimal playable dungeon crawler is a game consisting of rooms, ways for the player to move between them, enemies that appear in those rooms, and a way for the player to fight them.

6.1 Rooms

The first link imagined is between the directory structure of a software project and the rooms of a dungeon crawler. The directory structure can be visualized as a tree where folders are nodes, files are leaves and folders can contain both files and other folders, Figure 2 shows an example of such structure. Navigation in a directory structure is based on a hierarchical structure where a folder is called the parent and contains items called children. From a specific element, it is possible to go to the parent, unless it is the root item, which is the directory containing all the others and has no parent, or to go to a child item, unless it is a leaf. In The Coding of Isaac,

directories and files are rooms in which a player can be. A room is a flat horizontal surface where the player can walk and navigate. The player can also move from one room to another by navigating vertically, climbing a ladder to *go up* to the parent element or *digging* a hole to access the child element. The elements that make up the structure of a software project are the rooms of the game, and the links between the rooms are defined by the hierarchical structure of the directory.

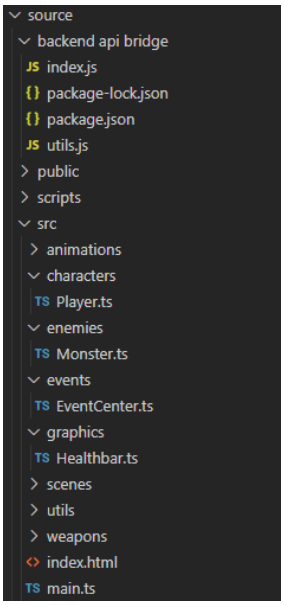

Figure 2: Software project directory structure example.

In a video game, the most basic element composing a room is the floor, which can be more or less detailed, have a texture more or less fancy. It can also be bounded or unbounded, in the first case the room has to be bounded by some kind of borders which are also detailed to some extent, and have a specific texture. These boundaries will determine the shape of the room. The textures of these two elements will give the overall look of the room, which seems perfectly adequate to represent some general measures of quality. Second, the texture of the walls that bound a room represents the level of security of the folder or file represented by the room. The wall metaphor is used to represent security using the firewall concept to help users understand this relationship. The floor texture represents the reliability rating, which is a function of the number of bugs present in the code. This rating affects the ground texture, referring to its effect on the overall stability of the software product. Finally, above the ground texture, to represent the maintainability score, more or less cracks of varying size are displayed. The maintainability rating is represented by a less visible element than the other two, because a bad maintainability rating does not imply visible problems, but more general implicit problems that will be troublesome in the long run.

All three types of ratings use a five-value ordinal scale ranging from A (best rating) to E (worst rating). As mentioned earlier, the association of a quality metric with a game element will only be allowed if the scale of the quality metric is lower than or equivalent to that of the game element. In addition, the more similar the scales, the more data will be preserved. The security and reliability metrics are mapped to the wall and ground texture, respectively. For this first design iteration, the texture values chosen are those of the element skin. In Section 2.1 Video Game, the skin is considered to be placed at a nominal scale, which would not be consistent with the previously cited constraint. In order to transform this nominal scale into an ordinal scale,

the authors decided to use the metaphor of material to order the different textures according to their preciousness, resulting in the following order: wood, stone, iron, gold and diamond. This scale is inspired by the one used in the game *Minecraft*, very popular in the gaming community. In practice, the use of this scale generates heterogeneous looking rooms with very dissimilar tiles, as shown in the Figure 3.

Figure 3: Room example using the materials tile set.

Concerning the maintainability rating, it is mapped to the percentage of slight and deep cracks displayed on the ground. A percentage is a value on an ratio scale which is an higher scale then the rating's one. Thus, similarly than in the Section 2.3 Measurement theory, thresholds are defined in the ratio scale to create categories that can be ordered from no cracks to lot of cracks. The percentage categories are summarized in the Table 6.

Maintainability rating	Ground cracks percentage
A	No cracks
B	20% Cracks 60% Slight cracks
C	40% Cracks 20% Slight cracks
D	60% Cracks
E	80% Cracks

Table 6: Percentage of cracks according to maintainability rating values.

Therefore, a room will represent either a file or a folder in the directory structure of a software project. Its walls and floor will reflect the overall quality of that part of the project. As explained earlier, the rooms are linked together according to the hierarchical structure of the directory structure. Then, when a player is in a room, they must have access to the parent directory and

the child elements. As each directory can have only one parent, it is not necessary to select it, at any time the player can press the E key to go to the parent directory of the current one. Conversely, if the player wants to go to a child item, they will have to select which one. To do this, separate squares are displayed on the floor of each room, each being an access to a child item. Like the rooms, the squares have boundaries and a defined floor area. They will therefore have a texture corresponding to the quality of the generic code of the file or folder they refer to. In this way, the player can see the general quality of a sub-room without having to navigate through it, and if they wants to navigate further into a child element, they have to walk over it and press Q.

The squares are arranged in a grid. They are all the same size, three tiles per side, and are separated by two tiles. A room is therefore always a square, when it represents a folder, the size of the room varies according to the number of children according to the formula below:

$$S_{room} = \sqrt{N_{children}} * S_{square} + (\sqrt{N_{children}} + 1) * 2$$

Where:

S_{room} = Size of the room
$N_{children}$ = Number of children of folder
S_{square} = Size of a square

The room size is limited by a minimum value of 11 tiles in order to maintain sufficient space in the room under all circumstances, even if a folder contains only one child. In the case of rooms representing files, it is different, since a file cannot have any child, the size of the room is constant. Moreover, on the floor there is only one square which represents the folder itself, this square is an exception because it is not an access to another room but it is present here just to keep homogeneity between rooms. Figure 4 shows an example of this kind of room.

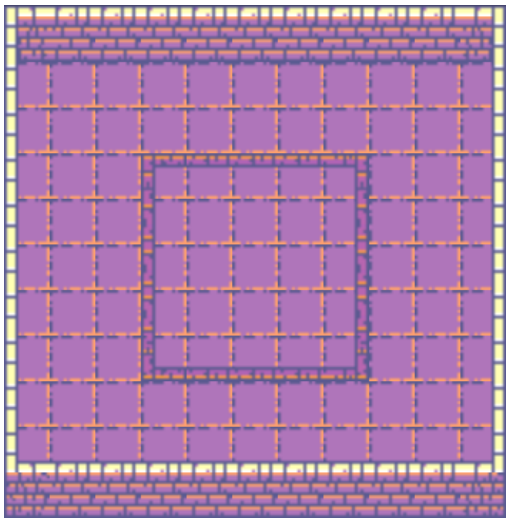

Figure 4: File room example.

This design choice keeps a homogeneous design in all rooms, regardless of the number of child elements. Moreover, an alternative design solution would have been to match the size of the

files and folders to the size of the rooms and squares. This would make the rooms less similar and redundant, and add the information about the size of the elements to the visualization, but the main problem with this alternative is that it would make the generation of the rooms much more complex, which would take too much time to develop. Regarding the chosen design, its disadvantage is the repetitiveness between rooms which is slightly compensated by the variety of textures due to the difference in general quality metrics between the parts of the software project, and the variation of the enemies.

6.2 Enemies

Enemies are one of the main elements of a dungeon crawler game. They are the threats that force the player to fight and explore the dungeon further. In parallel, in software engineering, people are threatened by issues in the code. Then, in The Coding of Isaac, each code issue is associated with a monster in the game. One of the goals is to make the player feel overwhelmed when there are too many issues in the code and then too many monsters. Moreover, in this kind of game, monsters can behave in many ways and have more or less complex list of actions they can perform in the game, which is called a skill set. The approach of this work is always to create a functional prototype in order to evaluate it later, instead of having a deep and complex gameplay. Then, the enemy part of the gameplay is inspired by the recently released game Vampire Survivors [27] which showed, with more than 98% of the 98 000 extremely positive reviews on steam, that the gameplay does not need to be complex to have an interesting game. In The Coding of Isaac, the monsters run straight at the player, if they reach him and their hitboxes collide, they do damage. On the other hand, the player's gameplay can take different forms, the traditional one is to play as a character and fight monsters, which can be done with weapons, spells, etc. As an example of an original form of combat in a dungeon crawler : Sandusky [2018] has created a peaceful dungeon crawler where the player fight monsters in a non-violent way. The traditional approach to combat was chosen in this work, in which the player plays a knight and fights monsters using a sword. The remainder of this section will focus on the enemies in The Coding of Isaac, detailing other aspects of the monsters and issues that are significant to the game and its visualization role.

6.2.1 Type

Code quality issues can be of different types, they can be bugs, code smells or vulnerabilities. The value of this type is expressed on a nominal scale, no order relationship is strictly defined between the values. In the game presented here, an issue is represented by a monster and the type of issue is mapped to the monster type represented by different skins. There are three types of monsters: goblin, demon and zombie, which, like the issue types, are on a nominal scale. This means that there is no order relationship between any two types. In practice, since these elements are not only theoretical, but also visible, some relationships exist due to the difference in color, height, width, etc. These minor differences are largely erased by the difference in skin design between the monster types, which is a major and highly discriminating difference. In the end, vulnerabilities are represented by demons, bugs by goblins, and code smells by zombies.

[27] `https://store.steampowered.com/app/1794680/Vampire_Survivors/` accessed 25 May 2022.

6.2.2 Size

In addition, the issues have different levels of severity. In fact, SonarCloud uses five different values to define the severity of issues, which are placed on an ordinal scale as follows, from least severe to most severe: INFO, MINOR, MAJOR, CRITICAL, BLOCKING. The severity level is mapped to the size of the monsters. As discussed in the introduction to this section, the authors, lacking sufficient design expertise, had to rely exclusively on resources they found online. The tile set they chose designed, for each monster category, three size levels that we will refer to here as tiny, medium, and large, each with a different skin. As a result, this study maps small monsters to INFO and MINOR severity issues, medium monsters to MAJOR issues, and large monsters to CRITICAL and BLOCKER issues. This mapping was chosen because the size of a monster represents its threat level, which reflects the severity of the issues. However, it is important to know that using size to represent severity can be misleading because users may understand a large monster as an issue that is widespread in the code, impacting a large number of lines of code. Figure 5 summarizes the mapping between monsters and issues.

Figure 5: Mapping of monsters and issues.

6.2.3 Spawn

In dungeon crawler games, monsters can spawn in a variety of ways. For example, all the monsters in a room may be directly there when the player enters the room, they may also arrive in waves depending on timing or when enough enemies in a wave are killed. The way monsters spawn in dungeon crawler games has a significant impact on the gameplay experience. Furthermore, in the context of this research, monsters represent issues in the code, these are an important part of the visualization, so the characteristics defining how they spawn will have a large impact on how users perceive it. This research seeks to give the user the feeling of being overwhelmed by monsters when there are a lot of issues in a specific folder or file, so the pace of spawn will be very crucial to this goal. Also, the number of items in a room at the same time is a common performance bottleneck in video games, so this should be taken into account when designing the spawn of monsters.

In this game, the monsters in a specific room are the sum of the monsters of each child in the folder, if it is a file the number of monsters is equal to the number of issues. Thus, the chosen approach is to spawn the monsters from the accesses to the child elements, represented by the

squares on the floor of the room. In addition, on each square, there is an indicator of the number of monsters that have yet to spawn from that location and the total number of monsters, dead or alive, related to that child. This indicator is illustrated by the elements a) in Figure 6. Using this, the player is able to quickly find which directory or file has more or less issues. The disadvantage is that the number of monsters displayed does not give any details about the type and severity of the issues that will spawn.

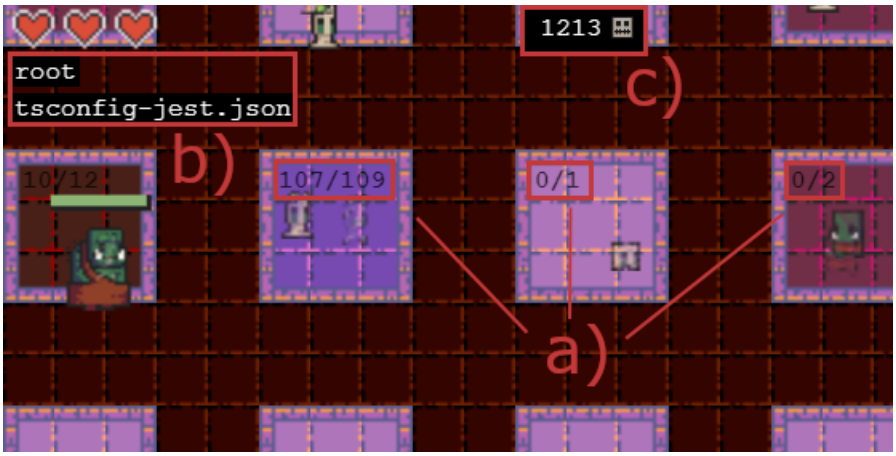

Figure 6: Labels in the main game screen: a) the numbers displayed on the tiles, b) the file and folder names displayed in the upper left corner, and c) the total number of monsters alive and going to spawn.

When spawning, monsters from the same directory or file do not spawn in exactly the same place. Instead, nine spawn points are defined and distributed in the area of each square and monsters spawn randomly on one of these spawn points, this is illustrated in Figure 7. This way, if a lot of monsters have to spawn in a short period of time on the same square, it will look more organic and visible than on a single static point.

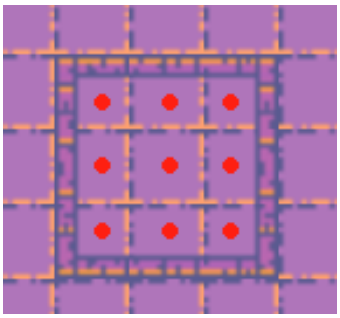

Figure 7: Locations of monster spawn points in one square. Each spawning point is represented by a red dot.

One of the major aspects of monster spawning is the timing and context of the spawning: **when** will the monsters spawn and **what event** will trigger their spawning. Several approaches can be taken, since this is an important part of dungeon crawler gameplay, they will be discussed in detail here.

First, monsters can spawn on all squares at once when the player enters a room. This approach gives an idea at a glance of how many issues a folder contains. It can work if the number of

monsters per tile is low enough. For example, if there is more than one monster per tile that spawns as soon as the player enters the room, the game would not be playable. Thus, this approach is not suitable for the context of this research, as a folder may have more than a thousand associated issues, which will overload the room and also cause performance problems. Nevertheless, this approach would be interesting if the mapping of issues and monsters were different. For example, if instead of associating an issue with a single monster, several issues were grouped together and represented by a single large monster, then this spawn approach would be usable because the number of monsters would be quite limited.

Second, monsters can spawn in waves. Several elements define this approach: **which monsters** spawn from **which squares**, **when**.

The selection of monsters that pop up in each wave greatly influences the game experience and the information the user will understand from the tool. In some games, different types of monsters have different attack patterns, some only do close combat, others can throw projectiles. In such a context, it is interesting to mix the different types of monsters to have a varied fighting experience. But in The Coding of Isaac, all monster types have the same basic pattern of walking towards the player. However, each monster represents a specific issue whose characteristics are reflected by the skin and size of the monster. Therefore, users will understand different information depending on the selection of monsters that spawn. For example, two squares both have ten monsters that will spawn on them, if for one square the first two monsters to spawn are medium and for the other it is two small ones, the player might think that the first square is more critical than the second one but in the end the first square might have ten medium monsters and the other one might have two small ones and eight big ones. The example is illustrated in Figure 8. Nevertheless, in this research, the texture of the room elements is designed to give general information about the quality of the code in a folder or file, the important information about the monsters is their number and the player's ability to kill them all. The spawning order of the monsters is the same as the issue order offered by default by the SonarCloud API.

Figure 8: Illustration of the problem of the spawning order of monsters.

The second element of the spawn of monsters in The Coding of Isaac is their original square. A first approach is to spawn monsters from a sub-room until it is exhausted, then choose another one to continue until all are exhausted. It may be interesting to focus on specific squares of the parent one by one, but it defeats the purpose of the tool, which is to give a general idea of the quality of the room. Besides, if the player wants to focus on a specific square, they can always dig in it. A second approach is to spawn a certain number of monsters in each square that still contains monsters at the same time. The main drawback of this approach is that the number of monsters that spawn depends on the number of files and folders in a room. This approach, where the number of monsters to spawn is set to infinity and the triggering event is the player's entry into the room, is identical to the first approach presented where all monsters spawn at once. The final approach to selecting squares is to consider the squares in a circular list, spawn a quantity of monsters in one square and move on to the next, once a square is exhausted it can be removed from the list. This approach allows monsters to be spawned from each square and gives an overall idea of the stakes of the folder. Obviously, if the player is in a room representing

a file, there is no square selection since there is only one square in the room.

The last component is the spawn trigger event, it can be periodic after a defined period of time or depend on a specific action of the player, for example when the player enters a room or has killed some monsters and the number of living monsters is below a defined threshold. The advantage of the latter approach is that it can help solve the potential performance issue if the threshold is low enough. The issue with this approach is that the game waits for the player and gives them some control over the appearance of monsters, which erases the feeling of being overwhelmed. The periodic trigger event has the exact opposite disadvantage and advantage. Another variable in the periodic trigger is the time between waves, which can be fixed for all rooms (e.g., one wave every two seconds), or variable depending on the number of monsters to be spawned (e.g., all monsters must spawn within three minutes). The latter solution is interesting for having a variable spawn rate, so that a room with few issues will have a low spawn rate and seem more manageable than a room with many issues. But the disadvantage of this strategy is that it can be difficult to find the right length of time to be appreciable, either for rooms with few issues or for rooms with many issues. On the other hand, the strategy using a fixed amount of time will be less scalable, for a large amount of issues it will take a long time to spawn them all.

Finally, as mentioned in the introduction to this section, one of the purposes of monsters in The Coding of Isaac is to make the player feel overwhelmed when a folder has a lot of issues. Thus, the spawn of monsters in this game is based on a periodic triggering event with a fixed time, at each triggering event a monster spawns from a specific square, then the next monster will spawn from the next square and so on. This way, the player will have an overview of the issues in the folder and is expected to feel overwhelmed if there are too many monsters. In addition, the HUD displays information about how many monsters are alive and still have to spawn, illustrated by the element c) in Figure 6.

6.2.4 Health

The final component of monsters that we discuss here is their health. As presented in the Section 2.1.7 Health, the health of a monster can be visualized in several ways. In this work, a monster's health is mapped to the estimated number of minutes it would take to solve the associated issue. The concept is that by attacking the monsters, the player will feel if they as an individual are able to tackle the issues. This metric is provided by SonarCloud as an estimate and its accuracy obviously depends on many factors such as the developer's skills, experience with the project, etc. It is used in a similar way here, as a way to estimate whether an issue can be solved by an individual. Thus, the information about the health of a monster does not have to be very precise either. It is then represented only by a health bar without any numerical values displayed, just to let players know how successful they are in solving the issues.

6.3 Player

The player character in The Coding of Isaac is not intended to convey much information about the quality of the project code. Rather, it is a means by which the player can interact with the game to gather information by exploring and fighting monsters.

6.3.1 Navigation

As presented earlier in the Section 6.1 Rooms, the player can navigate in two perpendicular directions.

On the one hand, the player can navigate horizontally, which refers to exploring a single room. To do this, the player can use four keys: W, A, S and D which represent the actions to move the character up, left, down, and right, respectively. All key assignment decisions are made according to the American keyboard layout. They are chosen instead of the traditional arrow keys so that users use their left hand to move and leaves their right hand free to use the mouse. The players are supposed to keep their left hand near these keys during the game, as these should be used often. Specifically, players are supposed to use these controls with their index, middle and ring fingers and leave their little finger and thumb free for the other controls.

On the other hand, the player can navigate vertically. As we saw earlier, this involves moving from a specific folder or file to its parent by moving up or to one of its children by moving down. The player can use the E key to go up from anywhere in a room, and the Q key to dig into a specific child item that the player character is standing on, if they are standing on any. The animations of these actions are shown in Figure 9 and Figure 10 respectively. These keys were chosen because they are meant to provide a very common interaction for the player and are very close to the horizontal move keys (W, A, S, D). When the player enters a new room, their character is positioned near the center of the room, at the bottom right of the center square instead of the center where monsters spawn to prevent a monster from spawning directly on the player.

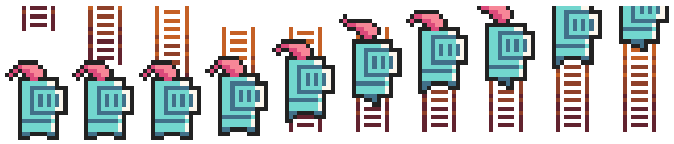

Figure 9: Go up animation.

Figure 10: Dig animation.

To help them find their way around the vertical structure of the game, two labels are added to the upper left corner of the HUD, illustrated by the element b) in Figure 6. The first is the name of the current file or directory. The second is the name of the file or folder represented by the square that the player character is currently on. If they are not on any square, the message "Nothing here" is displayed instead.

Also, since monsters spawn on squares that are a representation of the sub-directories and files, it may be difficult or impossible for the player to reach a specific square if too many monsters spawn there. Thus, the player can also navigate using the mouse, the right click is similar to climbing in the parent element, and the left click corresponds to digging in the square on which the mouse cursor is located. Also, when digging the player character is moved quickly over the square instead of an instant teleportation in order to keep a consistent flow for the user.

These two navigation modes force the player to go through each room one by one. If they want

to reach a specific square, this would not be convenient. Therefore, a map function has been added to the game to help the player in this case.

6.3.2 Map

The map is designed to give the player a more holistic view of the project structure, and to help them navigate more quickly between two distinct rooms. This screen consists of two main elements, also called panels, the project directory structure in the center and a search bar with a suggestion list in the upper left corner, illustrated in Figure 11.

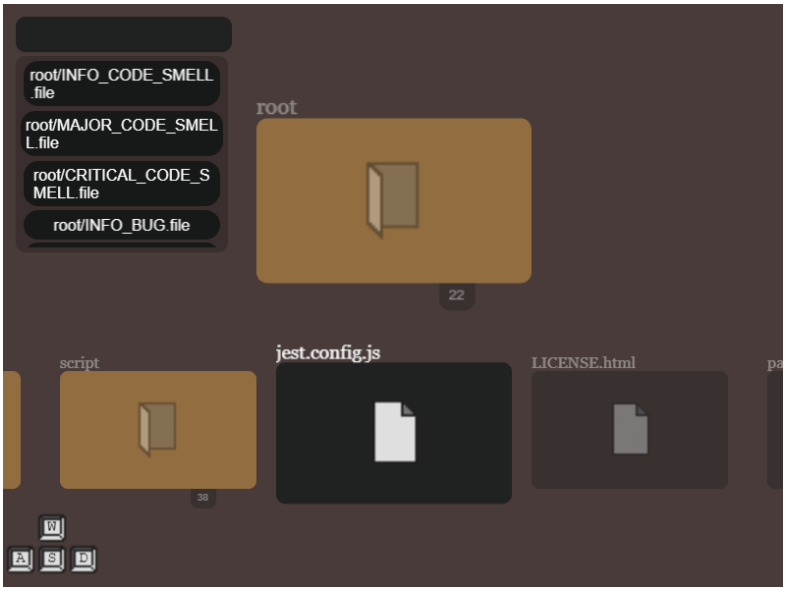

Figure 11: Map screen.

The tab key is chosen to open the map because it can be pressed by the little finger of the left hand which is supposed to be free as mentioned earlier. The map can be closed using the same key, which corresponds to going to the currently selected file or folder. Clicking on an item in the suggestion list will also close the map and send the player into the room. The suggestion list is generated based on the search string typed into the search bar. The search is designed to find all items whose path begins with the search string. This is not the most usable search engine as some users might want to find the item from its name directly, but it is the best compromise to have sufficient functionalities for the time allocated for development.

This screen of the game is more detached from the game aspect and closer to the real structure of the project, only the information about the size metrics of the project as presented in Section 2.2 Code Quality and Technical Debt is displayed. Folders and files are represented by rectangles with the name of the item at the top and the number of linked children just below. There are two different designs, one for files with a brown color and a file icon, and one for folders with a gold color and a folder icon as shown in Figure 12. Players navigate the map with the same character movement controls: W, A, S and D. When the player opens the map, the focus is on the file or folder represented by the room they were in. The sibling items are displayed horizontally as a non looping carousel. By moving left or right, the player moves the focus to the associated item. This display method is chosen to ensure a partial scalability in case a folder contains many items. The parent of the current item is displayed at the top of this panel, and the user can

access it by moving up and moving the focus towards it. They can also move down, which is like digging into a folder and focusing on its first child. The order of the children is taken from the SonarCloud API, in fact they are sorted by type, first folders then files, and alphabetically. As far as controls go, players can hold down a key to quickly repeat it conveniently, and when they reach the end of a list and try to go further, the screen shakes slightly to notice them.

Figure 12: Folder map rectangle on the left with the number of children at the bottom, and file map rectangle on the right. The the folder and file name is displayed at top of the rectangle.

6.3.3 Combat

The second major part of the interaction for the player character is the combat. Similar to the enemies, the player interaction is designed to be as simple as possible.

In video games, attacks can have many different designs, it can be a close contact attacks or projectiles, single target or multiple targets, consuming resources or not, etc. In this project, the player will only have one attack, which will be a close contact, multi-target directional sword strike in front of the character, with a delay between attacks. The close contact aspect will force the player to get closer to the monsters and add some challenge to the game. The delay and the directional aspect serve the same purpose of balancing the game so that it is not too easy.

The directional aspect can be designed in several ways for a top-down game, where the playing field is seen from above, like The Coding of Isaac. First, the attack can be in the direction of the mouse cursor. In this project, the mouse is already used to select squares and navigate quickly, and binding an additional control to the mouse would not be usable. Second, the attack can be launched in the direction of the nearest enemy or according to the direction of the player's walk. The former reduces the challenge for the player since it doesn't require them to aim manually, while the latter is chosen to try and keep the difficulty level decent.

Also, the multi-target aspect is chosen instead of a single target attack. Indeed, since this game is designed for hordes of monsters to come at the player, a multi-target attack is chosen to give players a sense of satisfaction when slicing through tons of enemies.

In addition, attacks in video games can be launched manually or automatically. For this research, the manual method is chosen because it involves the player more in the fight. They are expected to be more aware of the level of difficulty in killing the monsters, which reflects the difficulty in solving the problems. The attack is triggered by pressing the Space key, because according to other key bindings, the thumb of the left hand is always free and can press this key conveniently.

In addition, the attack inflicts a certain amount of damage. In this project, the amount is set to 2.5, as explained in the Section 6.2 Enemies, the health of monsters is the estimated time required to fix the issue, so the damage inflicted by the player represents 2.5 minutes of work on

this issue. This value was determined empirically by the authors by trying it out on a full-size SonarCloud project called BRAVE CORE.

An important point of this game is to make it interesting and challenging enough to intrigue the players but not so much that they forget the main goal which is code quality visualization. To provide a sort of challenge, the player has three hearts, which are displayed in the HUD in the upper left corner. Each time they are hit by a monster, they lose a heart, and if they reach zero, they die. Death should not be too frustrating in this game because the main goal is not the fighting challenge. The player will then be moved to the middle of the room and it starts over.

In addition, the difficulty of the rooms should be experienced independently, for example the player should not have more difficulty in a room depending on the previous one. For this reason, the player's heart counter is reset to zero when they move from one room to another. This means that the player cannot die by moving to another room, but they will still lose their progress because when they enter a room, it starts over.

Finally, knockback and invincible frame mechanics have been added to the game to improve the gameplay and make it more fluid. These are common mechanics in video games. The knockback mechanic means that when an object is hit, it is pushed back in the opposite direction of the attacker, enhancing the feeling of hitting or being hit. The invincibility frame consists of a short period of time after being hit, during which the player or the monster cannot be hit again, as a result, something cannot be hit multiple times too quickly.

6.4 Sound

There are two categories of sounds in a video game. First, the sound effects used to support and reinforce the events that occur in a game, such as footsteps when a character walks, sword strikes, falls, etc. Secondly, the music for the general atmosphere, it can vary during the game depending on the atmosphere of the room, whether the player is fighting or not, etc.

In the game presented here, several sound effects are used for the different types of events that can occur in the game. As far as music is concerned, it is interesting to have an adaptive one in this context to match the quality metrics of the code and to enhance the visualization. In fact, in The Coding of Isaac, five pieces of music were selected to reflect the five values of the reliability assessment. This ranges from the quietest music for the highest reliability rating to very nervous music for the lowest rating. The choice to duplicate the reliability index coding was made empirically as it provides an indicator of the overall quality of the room while varying sufficiently between rooms.

6.5 Visualization tool features

Three features are included in the game solely to support its visualization aspect. First, the player can click on a monster to access the full details of the issue on the SonarCloud website. This can pose a usability problem because monsters and squares can overlap and both are clickable. To address this issue, a second feature is added. It is called here the *freeze mode*, when it is activated, the monsters stop spawning up and moving, and the player cannot move the character, instead the motion controls the camera. Thus, the player can look around, hover objects, and continue to navigate from room to room with Q, E, and the left and right clicks. Navigating between rooms keeps the freeze mode on. Moreover, in this mode, the character animations are canceled to speed up the navigation as much as possible. Finally, when freeze mode is activated,

the user can hover monsters and squares with the mouse cursor to get more details about them with a floating tooltip. For monsters, this is the path of the element they come from, their type and size. For squares, the information is the path to the file or folder in the project's directory structure and the number of numbers linked to it as seen in Figure 13.

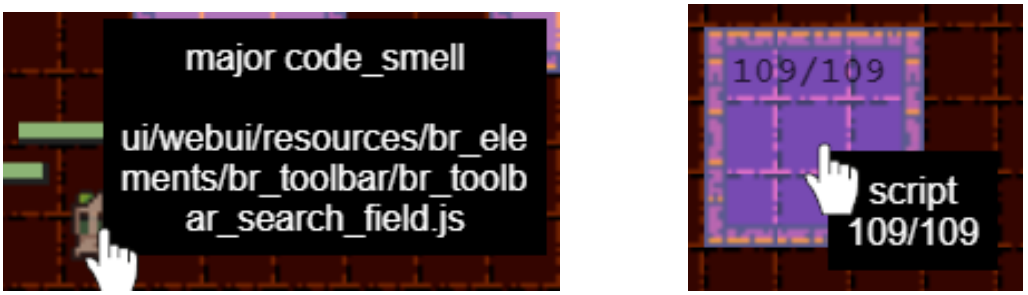

Figure 13: Tool tips. The left image is the monster tool tip showing the type of issue and absolute path to it in the project file structure. And the right image is for squares on the ground which are subdirectories or files, in the tool tip there is the name of the file/directory and the number of monsters that will spawn from this place out of the total number of issues in it.

6.6 Tutorial

When someone enters The Coding of Isaac, they first encounter a loading screen used to keep them informed of the game's loading progress, and then they land in the tutorial room. This is designed to resemble a room in the game to directly immerse the player. It is a playable tutorial, the player have access to all the commands of the game and is free to test them in context. The key bindings are displayed by icons representing the keys, and the mouse bindings by icons with the corresponding side blinking. In addition, the tutorial attempts to introduce the player to the game concepts with some examples of monsters and floor and wall textures. In addition, the floor texture in this room is designed to be neutral among the scale of textures used in the game. Finally, when the player wants to exit the tutorial, they can do so by clicking on the central square named "exit_tutorial". The Figure 14 shows that room.

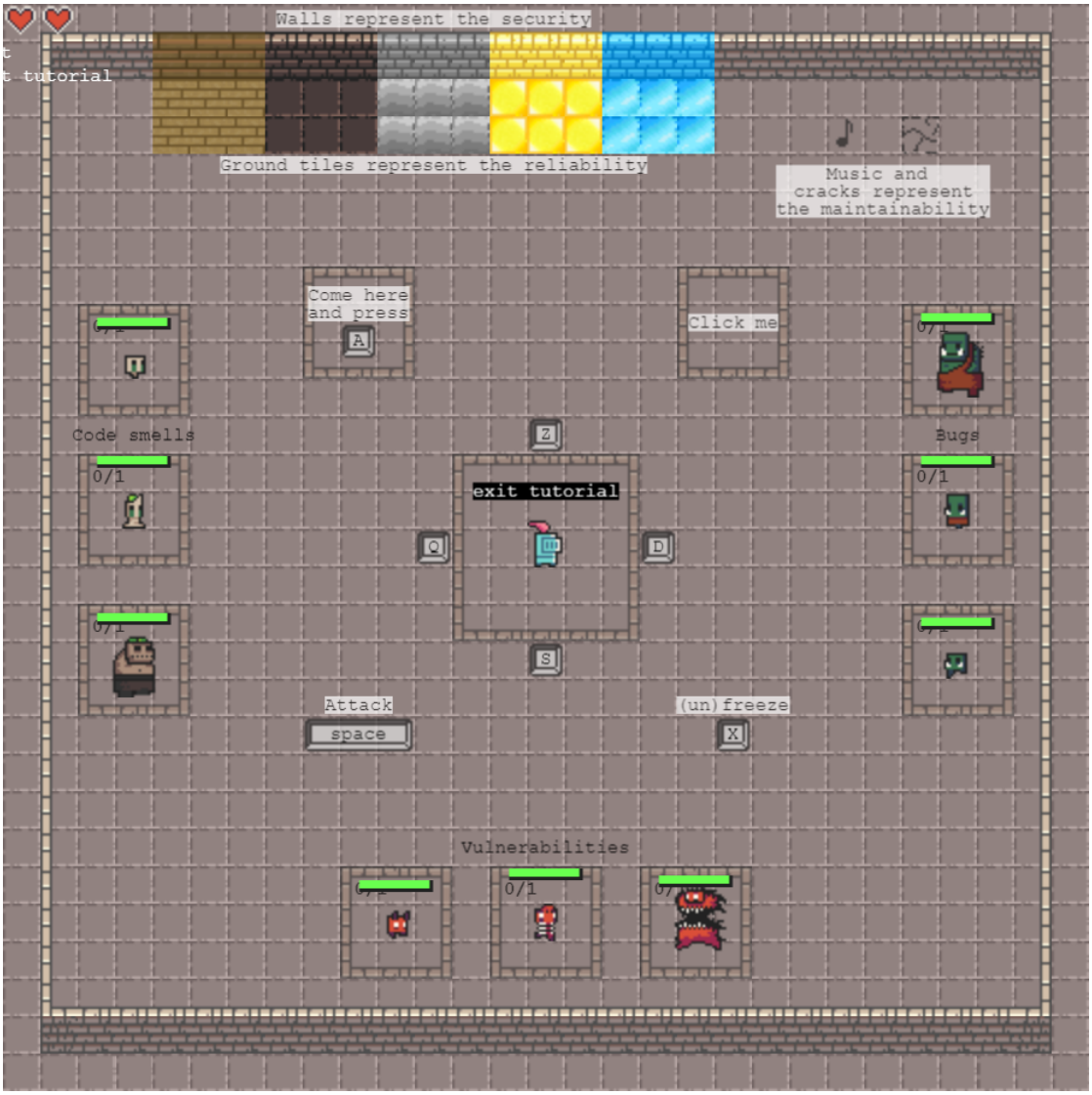

Figure 14: Tutorial room of the first design.

6.7 Project selection

The project selection screen is the bridge between the tutorial room, which is the same for all users, and the actual game which represent a specific software project. This screen consists of three panels, illustrated in Figure 15 : a search bar, a recommended project, and a list of suggestions.

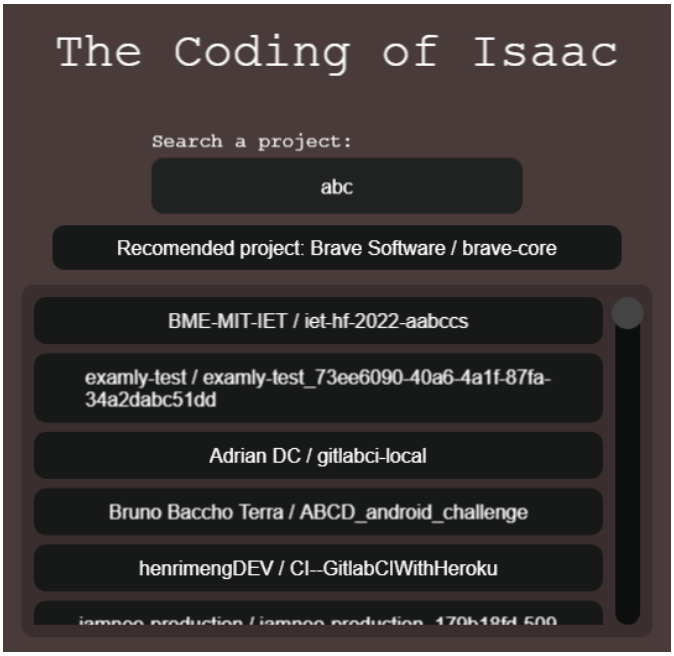

Figure 15: Project selection screen.

The user can type in the search bar a string to search for a specific project, and select it in the suggestion panel which can be scrolled with the mouse wheel. They can also choose the "recommended project", this panel is specifically there for this search so that later, during the evaluation, participants will use the same project to have a similar basis for the data. Once the user clicks on a project name, a loading screen appears to keep the user informed of the project's loading progress before actually entering the game.

6.8 Conclusion

The design phase of The Coding of Isaac has led to map the selected code quality metrics presented in the introduction with game components. The Table 7 presents this mapping. In addition to the mapping, some features have been added to support specifically the usability of the tool and the game. Now that the design phase is complete, the following step is the implementation of the tool which is discussed in the next chapter.

Code quality metric name	Metric value	Game element value	Game element name
Issue type	Code smell	Zombie	Monster type
	Bug	Goblin	
	Vulnerability	Demon	
Issue severity	INFO and MINOR	Tiny	Monster size
	MAJOR	Medium	
	CRITICAL and BLOCKER	Big	
Time to fix issue	Number of minutes	Number of health point	Monster health
Maintainability rating	A	No cracks	Ground cracks
	B	20% Cracks 60% Slight cracks	
	C	40% Cracks 20% Slight cracks	
	D	60% Cracks	
	E	80% Cracks	
Reliability|Security rating	A	Diamond	Ground|Walls texture
	B	Gold	
	C	Iron	
	D	Stone	
	E	Wood	
Directory structure	Number of children	Number of tiles	Room structure
	Children	Squares	

Table 7: Quality metrics and game components mapping.

7 Implementation

This section presents the technical concerns and implementation details encountered in the development of The Coding of Isaac as designed in the previous section. The first part discusses the overall architecture of the game and the infrastructure components. The second part presents the interaction path that users will go through when using the tool. The last part explains how Sonarcloud is included in the solution.

7.1 Architecture

The Coding of Isaac relies on the following three components, also presented in the Figure 16 with their different connection points:

- The Frontend, which is the game
- The SonarCloud application programming interface (API)
- The Backend, which is the bridge between the frontend and the SonarCloud API

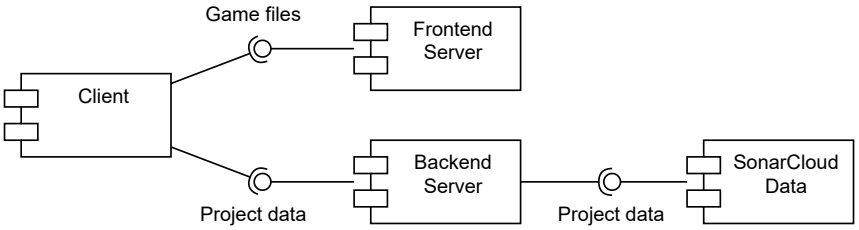

Figure 16: Game architecture.

Before choosing the technologies, the first choice to make was between making a standalone game or a plugin for an existing tool. A standalone game requires developing it from scratch but allows for greater flexibility and control of the product. On the other hand, a plugin requires learning the basics of the API or software development kit (SDK) of the tool it is developed for. However, this can be very convenient for developers as they can continue to use their usual tool with extended functionality. Choosing to develop a browser-based game combines the advantages of both approaches. Indeed, opening a website requires nothing more than a browser. A browser is a tool that developers usually already use, and since we are not locked into any tool, we keep control over the evolution of the game.

As mentioned in the background, the front-end is powered by the Phaser 3 framework. This javascript game engine allows to create a game fully playable on a web browser without any installation required.

The front-end is served by a front-end server integrated to Phaser 3. The backend uses Node.js and Express, creating a simple REST API to link the SonarCloud API and the front-end. The backend is necessary because the SonarCloud team refuses to allow cross-origin requests for security reasons. Therefore, the client cannot access SonarCloud data directly through the SonarCloud API, the only way to access it the way we want is to use the backend as a bridge between

the client and the SonarCloud servers. We chose Express JS because it is a very minimalist, stable and popular framework to build a simple backend server very quickly. Also, the authors are both familiar with Javascript, and Phaser 3 is a JavaScript framework. It was logical to keep the same technology.

When we chose the Phaser 3 framework and web browser as the platform for developing the game, the goal was to make it as accessible as possible, to avoid installation problems due to specific hardware or missing drivers, and to minimize the game's footprint on users' computers. Phaser 3 appeared to be the best choice to develop this prototype. Indeed, in this study, the objective is to create a proof of concept to prove that code quality visualization through a video game is possible. The different frameworks available on the market were compared using the following criteria considered important for this study.

- Large and active community
- Complete documentation
- Stability
- Still maintained
- Popularity
- Availability of tutorials
- Allows the making of 2D Games
- Reasonable learning curve
- Avoid compatibility issues (All-in-one)
- Flexible enough to allow the incorporation of external data

A few frameworks have been considered, Table 13 in appendix A compares them according to the above criteria. As a result, the framework that best matched the previously defined criteria was selected. The game was tested on Chrome the most popular web browser on the market[28]. Moreover, Phaser 3 makes compatibility easier because it does most of the work. Indeed, it uses WebGL which now has a good compatibility with Internet Explorer, Safari, Opera, Chrome and Firefox[29]. Finally, in the next steps, it will make the interview process more convenient since the interviewees will not have to install anything. The main disadvantage of browser-based games is the limitation of resources, but in the context of this research this does not matter because the game, as designed, does not require many resources. As far as requirements are concerned, this game uses about 750 MB of RAM (on Chrome), an Internet connection and 50 MB of transferred data every time the game is loaded.

If someone wants to host the game, they will need three elements. First a server to host the Nodejs/Express back-end, a web server to serve the front-end to clients and a domain name to make the game accessible to the world. In our case, in order to quickly test the prototype, everything was hosted on the same home server using Docker[30]. Once configured, Docker allows us to do quick builds and makes deployment much simpler. It is also useful to go back to previous builds in case something goes wrong. However, it is important to note that we did not use a production-ready web server to serve the frontend, but the development server included with Phaser3, which should not be done in a real production environment, as it can raise security and performance issues, as the development server is not built to handle many clients or with security in mind, unlike a production web server like Nginx[31] or Apache[32]. But since it is only used here for testing purposes on few users, there was no need to use anything else.

[28] `https://kinsta.com/browser-market-share/`
[29] `https://caniuse.com/webgl`
[30] The Docker image is available on the project repository on `https://github.com/snail-unamur/Yo-kai-watch`
[31] `https://www.nginx.com/`
[32] `https://httpd.apache.org/`

7.2 Game flow

A user session on The Coding of Isaac always begins with the game downloading from the front-end web server. The client essentially downloads all the files required to run the game, which is only necessary once at the beginning of the session. Then, the player enters the tutorial where they have time to get used to the game controls just before going to the project selection screen shown in Figure 17.

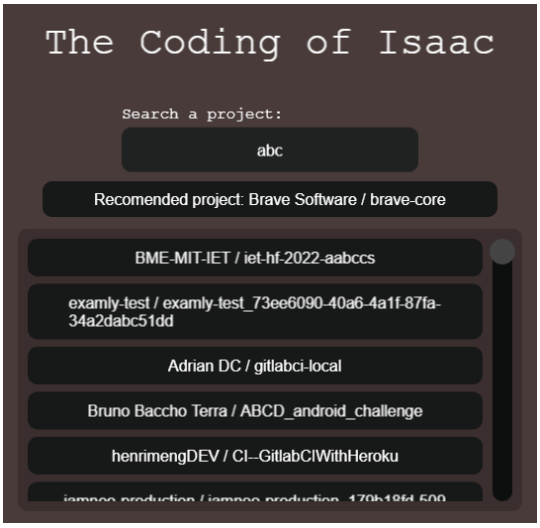

Figure 17: Project selection screen.

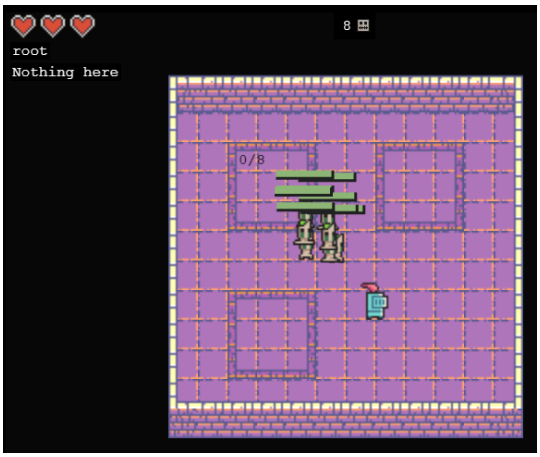

Figure 18: In-game representation of the root directory of a public project on SonarCloud.

When selecting the project, the player can search for any public project available on SonarCloud, at the time of writing, it is not possible to connect to a private Sonarcloud repository or any other Sonarqube repository because the prototype test did not require such functionality. However, in the case of a final product, it would probably be an important feature to consider. Each time the player types in text to find a project, the frontend sends a request to the backend server, which forwards it to the SonarCloud API to get a list of the first 25 projects containing the entered text, sorted by most recent analysis date on SonarCloud.

Once the player has selected the project, the front-end will similarly query the backend web server, passing the request to get the data from the latest project analysis to the SonarCloud API. Finally, the player will enter the game starting in the root file shown in Figure 18. The workflow textually presented here is summarized in the sequence diagram in Figure 19.

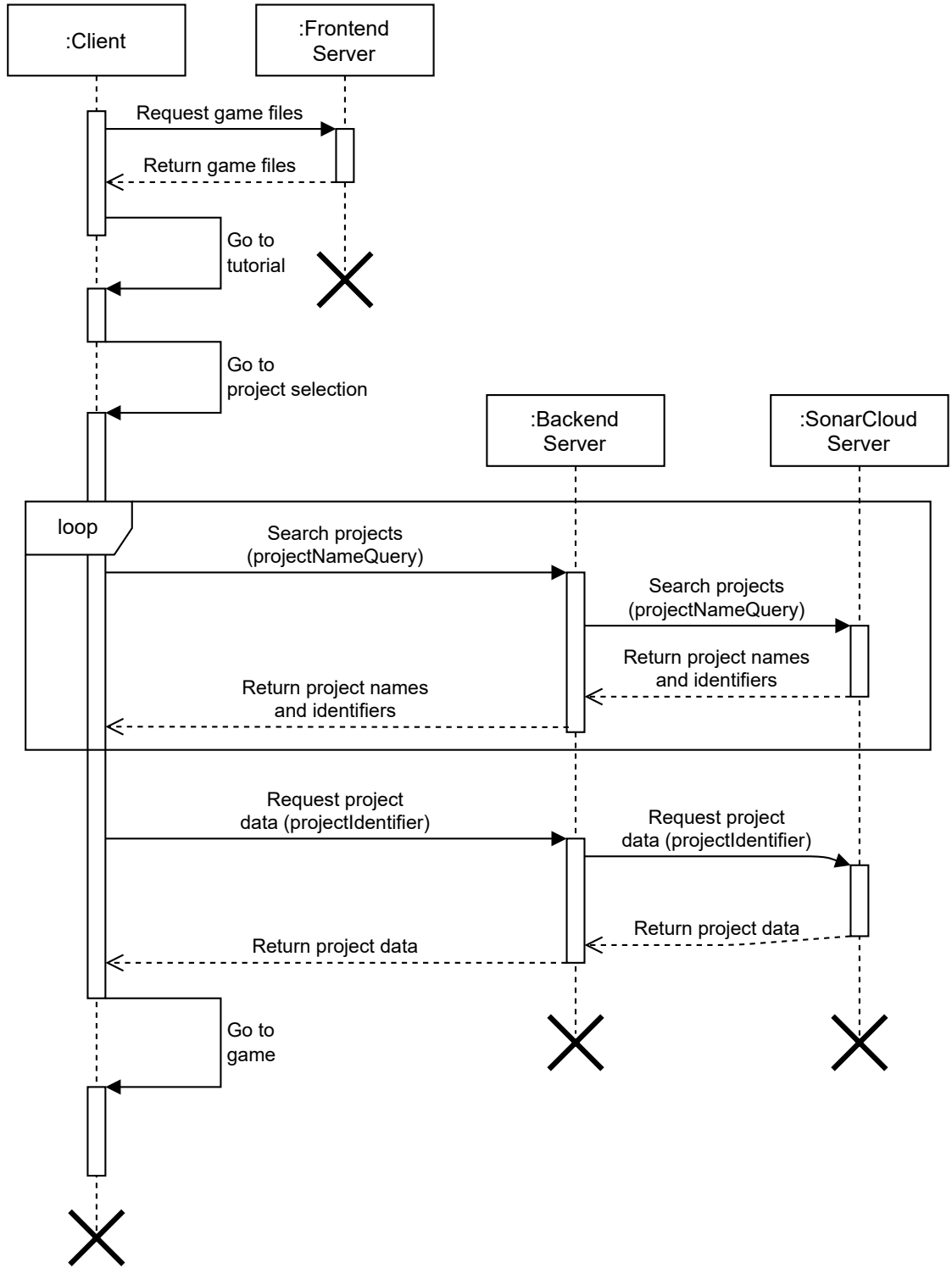

Figure 19: Sequence diagram of the player workflow.

7.3 SonarCloud

The choice of SonarCloud is a matter of multiple factors. First, it provides free access to quality analysis from a large number of open source projects that use the tool for their quality management.

Second, they provide an API to access the analysis data from these projects. In the context of this work, it is very convenient to be able to test the tool on multiple software without having to install anything or perform our own analysis. Also, it requires less infrastructure than SonarQube because it is cloud-based and hosted by SonarSource SA.

Third, SonarCloud provides both high-level and low-level information. High-level information is general assessments of a project's reliability, maintainability and security. Low-level information is specific code smells detected in a line of code, the number of lines of code, etc. High-level information will be particularly important for this project, as we expect it to easily provide insight into the overall quality of the project.

Finally, the work done in this thesis could be replicated for any other quality management tool, as long as the tool gives you a way to extract measurements.

The SonarCloud API is very comprehensive and allows for the collection of all types of measurements needed here. However, its documentation is very lacking. Therefore, some reverse engineering and deduction was required to find some unmentioned but important paths in the API. For example, the API endpoints for the SonarCloud's search capabilities in the explore page are not mentioned in the documentation because it is not a normal use case as stated by Schmitt [2020]. But for this project, it is necessary to be able to easily search any project. So we inspected the network tab using the browser's debugging tools to determine what API calls were made when searching and filtering results on this page. The relevant API endpoint found is:

```
https://sonarcloud.io/api/components/search_projects
```

The following end point is also used to retrieve the code metrics of a specific project once selected.

```
https://sonarcloud.io/api/measures/component_tree
```

And finally, this last endpoit is used to retrieve issues of a specific project from most recent to oldest.

```
https://sonarcloud.io/api/issues/search
```

8 Evaluation Methodology

The tool is evaluated iteratively in two rounds. Starting with the initial design presented in Section 6 Design, it will be refined after the pilot interviews to make sure it is stable enough for the actual interviews.

The first round of interviews consists of light, informal pilot interviews to identify the major problems with the initial design and the evaluation protocol for subsequent interviews. The objectives of these interviews are first to gather specific qualitative data on the game design. Second, to provide answers to the research questions through surveys, interview questions and experiments on the intuitiveness of the scales.

To test the prototype, we decided to use a method that combined allowing the target users to test the game while conducting a semi-structured interview with them. Indeed, this exploratory research is in a new and specific area of visualization, so it is possible that the authors may not think of certain questions and answers that would be relevant to the field. For this reason, it is preferable to use a type of interview with some open-ended questions that allow for unexpected answers. In order to have some quantitative data as well, at the end of the interview, the interviewees are asked to fill out a questionnaire to collect their impressions of the game they had just played.

The game session allows the players to discover the game for the first time and us to observe how players interact with the game and adapt the questions based on what happens to them in the game. The questionnaire at the end of the game session helps to cross-check our interpretation of the qualitative data collected through the interviews with a more quantitative approach.

8.1 Participants selection

To properly assess the quality level of the prototype and answer all the research questions stated above. We need to select the participants into two groups: industry practitioners and students. Based on these requirements, 15 volunteers were invited or showed interest to test the game under our supervision. This actually led to 12 interviews, three volunteers canceled or could not participate anymore. In order to minimize bias in interviews three rules were followed.

1. No interviews with friends or family.
2. No interviews with acquaintances even if we talked to them for less than five minutes.
3. All interviews are conducted in English if the interviewee fully understands it.

Refraining from taking people we know, we avoid much of the bias that would result from the person's opinion of us. We therefore avoid letting this opinion affect the object of the study.

Also, the wording of the questions is very important in interviews because it can influence the responses. To avoid word choice affecting the interview results, all interviews use exactly the same questions and are all conducted in English, the language spoken by all interviewees. To determine this, the preliminary email discussions with the interviewees were in English. Therefore, if they agree to the meeting, we consider them comfortable doing it in English.

8.2 Interview objectives

To conduct the semi-structured interviews. The questions were developed to cover all research questions and extract as much relevant data as possible. The Table 8 lists all questions and the expected data related to the research questions.

Id	Question	Expected data
1	Do you feel overwhelmed by the information in the game? e.g. too much information, too little information, information too complex, etc.	**RQ1.2**: Check if the information is on demand for the user or if one kind of information is being forced on them
2	What do you think of the severity of the issues this room contains?	**RQ1.1** + **RQ1.2**: Check participant understanding of this file tech debt (code smells, bugs, vulnerabilities). Compare their answer with the actual report from SonarCloud
3	What does playing in this room indicate about the code quality?	**RQ1.1**: Check participant understanding of this file code quality. Compare their answer with the actual report from SonarCloud
4	What do you like and don't like about combat in the game?	**RQ1.2**: Get data on combat intuitiveness and ease of use. What can be improved, etc
5	Do you feel motivated to fix TD when playing the game?	**RQ2.1**: Check how much does it encourage developers to repay some debt.
6	How would you see this tool used in your everyday life?	**RQ2.4**: See how participants think they will use this tool if it was a real product.
7	When would you use this tool?	**RQ2.3**: See when participants think they will use this tool if it was a real product.
8	Who would you see playing this game?	**RQ2.2**: See who participants think will use this tool if it was a real product.
9	What did you like about the game?	Potentially uncaught information from the other questions. Maybe if the participant has something the authors did not think about to say.
10	What did you dislike about the game?	Potentially uncaught information from the other questions. Maybe if the participant has something the authors did think about to say.

Table 8: Interview questions with their corresponding expected data.

8.3 Protocol

The recruitment of the participants was done with the help of the three thesis directors and a friend. We wanted to have a mix of students and professionals in the software development field. By combining the connections of our contacts, we managed to get the contact information of eight employees and four students for the interviews. Each of them was emailed a formal invitation to participate in the study, along with a comprehensive information sheet providing more information about the study and how we will collect and protect the data. Finally, the interviews

were scheduled according to the participant's availability. Immediately after an interview was scheduled, the interviewee was given a demographic form to fill out in order to collect more specific data about their profile. The information sheet and demographic form are available in the appendix section.

In this document, interviewer and investigator are used as interchangeable names. In addition, to limit small variations due to the personalities of the researchers, all interviews were conducted by the same investigator. With the exception of two student interviews that were conducted face-to-face, the second author acted as an assistant to the interviewer and observed and took additional notes on the interview process. The structure of each interview is as follows:

1. Presenting who the interviewer is and what this interview is for.
2. Explain once again how the data will be collected.
3. Ask the participant to sign the consent form to allow the interview to start. This document is in the appendix section.
4. Remind the participant that they can ask any questions they would like at any point.
5. Proceed with the textures and musics ordering tests.
6. Start the game and let the player explore freely while asking the interview questions of Table 8.
7. Finish the meeting by asking the interviewee to fill out a last questionnaire, which is in the appendix.

As mentioned in the fifth point, during the interviews, a test of the intuitiveness of the texture scales is conducted. It is divided into two parts: first five screenshots of the tiles of the game are showed, each with a color level from the funky color scale with a label attached to it, and the participant is asked to order it in a way that makes sense to themselves, they do not have to explain it. Then, the operation is repeated with the rainbow color set. Figure 20 shows the screenshots used in these experiments.

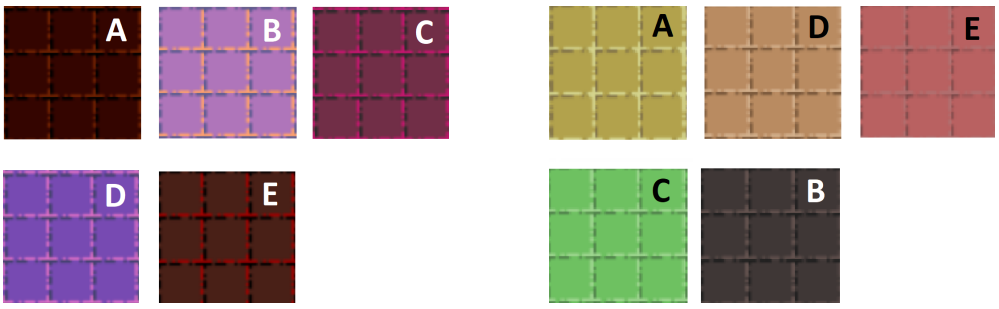

Figure 20: Images used to test the color scales intuitiveness. Left image is for the funky scale and the right image is for the rainbow scale.

After the color scales ordering intuitiveness experiments, the subject is asked to repeat a similar task but with the music. This time, the investigator sends a link to a SoundCloud playlist containing the five pieces of music, with the title replaced by an identifier from 1 to 5, the artist's name removed, and placed in a random order. As with the color scale experiments, the same random order was kept for all interviews.

Moreover, at any point in the interview, the investigator would let the participant try each feature for a period of time to gather data on the intuitiveness of the mechanics. If the volunteer began to say or show signs of annoyance, the researcher would reveal the solution to unblock the

participant to avoid loosing too much time. If the investigator had to intervene, the feature in question was considered poor in terms of intuitiveness.

In addition, to maximize the quality of the interviews, we followed the recommendations of Turner [2010]. The article proposed a list of practical elements to pay attention to before and during the interview, such as where the interview should take place or the most optimal way to ask questions without inducing an answer.

Finally, after the interview, the participant were invited to fill out a survey to gather their final thoughts about the game. This allows us to have meaningful quantitative data and confirm their thoughts. The survey questions are taken from the Intrinsic Motivation Inventory[33], "a multi-dimensional measurement device intended to assess participants' subjective experience related to a target activity in laboratory experiments". This measurement device was built by Ryan and Deci [2000]. This tool offers a certain number of variables that can be assessed such as Interest/Enjoyment, Perceived Competence, Effort/Importance, etc. For our case the relevant variables that covers our research questions are:

- Interest/Enjoyment
- Perceived Competence
- Value/Usefulness

Interest/enjoyment measures the fun and engaging part of the game, value/usefulness checks if the game has any value besides being fun and finally perceived competence helps us understand if the game is too difficult and intuitive or not.

This tool offers a set of questions that can be used to assess each variable. Three and five questions were selected per variable to have enough redundancy. The authors of this tool have written that "past research suggests that order effects of item presentation appear to be negligible". The questions used in this research are summarized in Table 9

Scoring calculation: (R) questions means that the scoring scale is reversed, so the question score must be calculated as "8 - value of answer". For normal questions it is the value of the answer. For each of the three categories detailed above, the mean is calculated.

8.4 Tools used for coding and analysis

A certain number of third-party and home-made tools were used to code the data from interviews and to analyze it.

Once the interview was conducted, it was transcribed. Using Descript[34] first, this automatic transcription tool takes an audio or video file and turns it into text. It can automatically generate speaker tags in the text to identify the person speaking. In addition, it adds timestamps to the text which are very useful to be able to go back to the audio file when it is necessary to listen to the clip for more context or to correct the automatic transcription.

In practice, each time someone had to correct the transcription generated by the tool because it is not perfect. In this study, there were different data sources. In the face-to-face interviews, the voices were recorded on a mobile microphone, the screen was recorded via OBS[35] on the

[33] `https://selfdeterminationtheory.org/intrinsic-motivation-inventory/` accessed 19 March 2022
[34] `https://www.descript.com/`
[35] `https://obsproject.com/`

Interest/Enjoyment	Perceived Competence	Value/Usefulness
While I was playing this game, I was thinking about how much I enjoyed it.	I believe I was skilled at this game.	I think this is important to play this game because it can help a developer understand the code quality of a project.
This game did not hold my attention at all. (R)	I am satisfied with my performance at this game.	I believe playing this game could be beneficial to me.
I enjoyed playing this game very much	After playing this game for a while, I felt competent.	I think playing this game could help me understand the issues with the code.
This game was fun to play.		I would be willing to play again because it has some value to me.
		I think that playing this game is useful for a developer.

Table 9: Post interview survey questions.

computer, and the field notes of the observations were taken independently. Thus, in order to ensure temporal consistency of the different data sources, the editing function of Descript was not used, but the transcript was extracted into a text file and the corrections were made in Google Doc.

To support the data coding process, the authors used an internal tool named the "label picker"[36] to easily explore and reuse existing labels and sub-labels. The tool represents the structure as explained earlier, from themes to labels to sub-labels. In the tool, labels are considered the parents of sub-labels and the children of themes. All of these tags are manually added by the authors once; then, two types of interactions are possible. First, a colored filled circle means that it has children and clicking on it will display them. Then, a transparent circle is a leaf that has no children, and by clicking on it, the correctly formatted tag will be placed in the clipboard, ready to be pasted into the text file. Finally, a click on the central circle will return to the parent. The tool is illustrated by Figures 21 and 22.

The topic selector, also known as the "topic guide", allows us to analyze all the interview questions in one place instead of spreading them across multiple documents. The tool displays bubbles with topics that are based on the coding done in the previous step, as shown in Figure 23. This tool is a modified version of the label picker tool presented earlier.

A number indicating how many interviews mentioned the specific item has been added to each bubble. Clicking on the bubble shows the corresponding interview ids. For example, if there were three mentions of "gamification", only those three interviews will be displayed under the gamification topic. This way, we can easily see which theme our respondents consider relevant or not without having to go through each document multiple times. The topic selector has also been refined by adding relevant information from the demographic survey. For example, if an accessibility specialist says they do not find the game useful, a bubble for their position is added indicating a negative opinion, even if they did not mention it directly in their responses during

[36]Source code of the tool available on `https://github.com/snail-unamur/Yo-kai-watch/tree/main/evaluation/interview_data_visualization_tool`

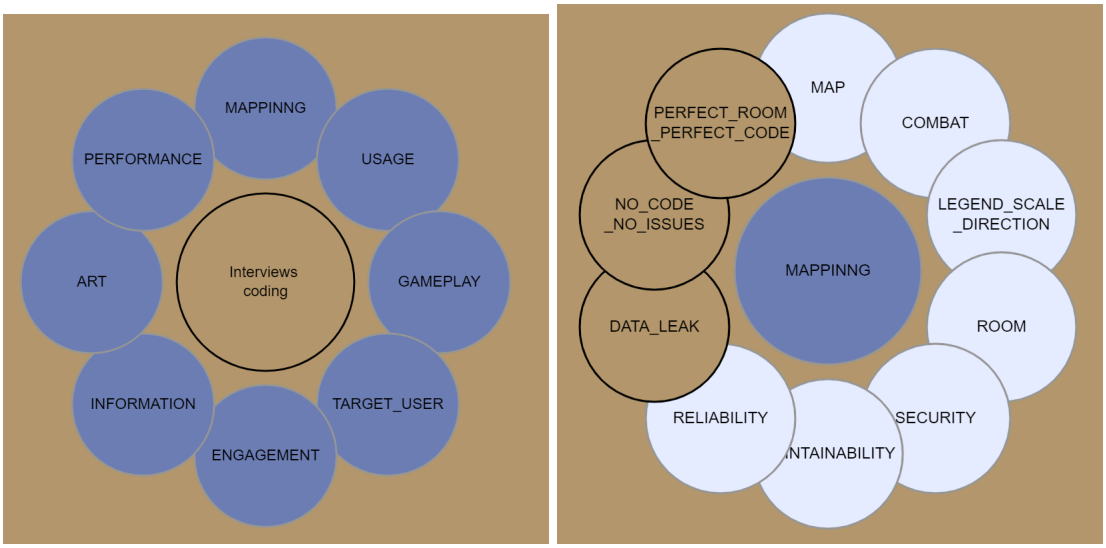

Figure 21: Label picker in-house tool screenshots (1).

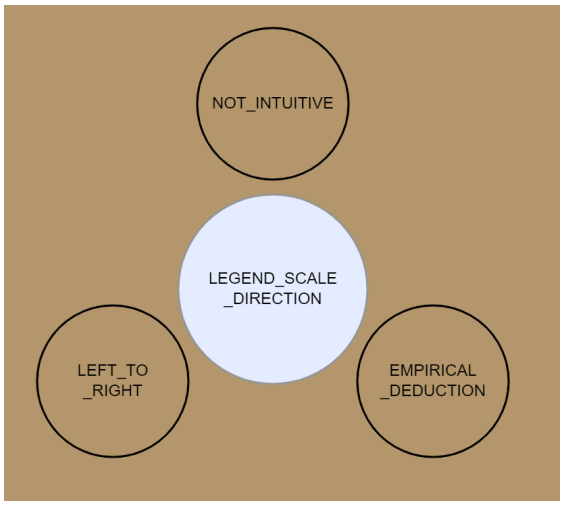

Figure 22: Label picker in-house tool screenshots (2).

the interview.

Another tool used is a script developed to extract the relevant paragraph or quote from an interview transcript using the tags assigned to it. Here, timestamps related to the tags are also extracted, so we know exactly where the tag appeared in the interviews. At any time, the information given by the quote can be cross-checked with what was actually said in the conversation, ensuring that everything matches up correctly.

This collection of tools proved extremely useful during the coding and data analysis phase, as it allowed us to make the coding phase more efficient and quicker and to get a quick overview of all the different topics discussed in all the interviews at once, as well as to quickly locate specific pieces of information in the transcripts. All tools were manually tested using a small sample of example data to verify that the tool's behavior was expected.

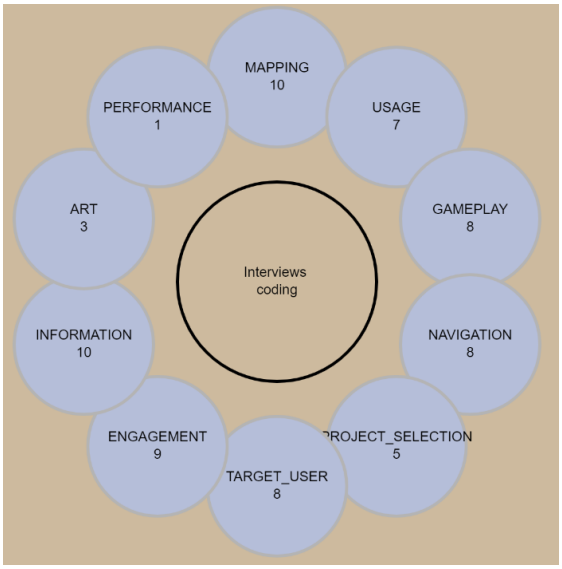

Figure 23: Topic selector indicating the number of interviews in which each theme appeared.

8.5 Data coding process

The process of coding began with corrections to the transcript consisting of checking if the content generated by the tool is correct, but also removing unnecessary content such as greetings and small talk, dividing the text into paragraphs, and merging the different data sources into the text file by adding relevant contextual information based on the video and the observation field notes. These output files are the basic artifacts used to analyze the content.

Miles et al. [2018] open coding approach was followed to obtain more intelligible data from the files generated from the multiple interview data sources. Both authors participated in the coding of the data to generate the maximum amount of insight and ensure the least amount of information loss.

The open coding approach is an iterative process: an interview is conducted and transcribed, then the data is coded, resulting in labels and sub-labels. Next, the labels are categorized into broader themes. Then, the process is repeated with a new interview, but this time the labels and sub-labels already generated are used as the basis for coding the data, while allowing new labels to emerge from the data.

Each thing said by the participant that the authors consider interesting is tagged during the first round of coding. This tag contains two parts: the first one, called *label*, is a descriptive word of the content and the second part, called *sub-label*, specifies the first descriptive word with a value, for example: "STUDENT-POSITIVE", the first part "STUDENT" means that the respondent is talking about a student using the tool and the second part after the hyphen "POSITIVE" means that they think that this category of user could use the tool. According to the open coding approach, the labels and sub-labels are not predefined, they emerge from the data. This coding is done in the previously generated text files, the person who codes, writes in front of the paragraphs the tag they find relevant.

The usual workflow began once a text file was fully generated and populated with contextual information from the other data sources mentioned before. The authors separately code the file

using the Label Picker tool and create new tags if necessary. Once both authors have coded the file once, they discuss each other's coding to agree on a common coding and create a merged coded text file that is exported in TXT format so it can be processed automatically. Then, a small python script is used to extract the tags from the file and generate the list of new tags for this iteration by comparing it with the list of previous tags. The new tags are then classified in the different themes. In addition, the script calculates the proportion of new tags generated to the number of previously generated tags used in the coding. The numerical value of the proportion is an indicator of the evolution of the completeness of the coding, as well as the absolute number of new tags generated. The evolution of these values over the course of the interviews is presented in the Figure 24. It is interesting to note that after six interviews, the values begin to stabilize around 10-20 new tags per additional interview, and 10-20% of new tags.

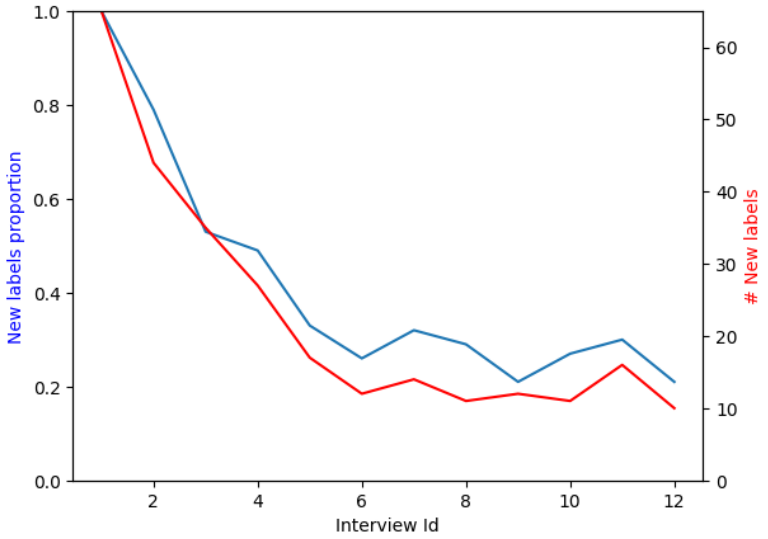

Figure 24: Evolution of the proportion and number of new tags through the different interviews.

8.6 Data analysis process

This section presents explanations about how the end-of-interview questionnaires were analyzed and finally an overview of the methodology followed for the analysis of the interviews is given.

Regarding the questionnaires given to the respondents at the end. They were asked to rate several sentences on a 7-point Likert scale ranging from 1 (strongly disagree) to 7 (strongly agree). This standardized scoring system then allowed us to follow Ryan and Deci [2000] and easily track the responses of different respondents to generate clear statistical results for the analyses.

As for the qualitative data, it comes essentially from the interviews themselves. For each topic that emerged, the different opinions expressed by the interviewees are summarized. These summaries are then presented in themes in Section 10 Results. To increase the quality and accuracy of our interpretations, we made sure to cross-check the information with the questionnaires and interview transcripts before drawing conclusions. In addition, whenever possible and relevant, we included quotes that were particularly interesting or that helped to illustrate an opinion better than a rephrased text would.

9 Pilot test

Prior to conducting the 12 interviews that will be described in Section 10 Results. Pilot tests were conducted. The purpose was twofold: to find major problems in the game design and in the evaluation process. It was important to ensure that the interview questions, protocol, and game were rigorously vetted and validated prior to implementation, as time and the number of people available to participate are limited. Pilot interviews would reduce the risk of interview failure.

These pilot interviews were conducted the same way as the actual interviews thus they followed the same protocol and recommendations mentioned in Section 8.3 Protocol.

Three pilot interviews were conducted with three computer science students at the University of Saskatchewan. For this pre-assessment phase, participants were selected based on their accessibility and availability, regardless of their position, experience, or relationships.

9.1 Resulting changes

Numerous insights were obtained from the pilots that led to some changes related to the end of interview questionnaire, the choice of game colors, combat, tutorial and finally bugs. The following sections detail for each of these elements **what** has been changed and **why**.

9.1.1 Questionnaire

With regard to the questionnaire given at the end of the interview, some questions were confusing for the pilot study participants. We chose to delete these questions because there were enough remaining redundant questions to cross validate each variable.

9.1.2 Colors

Furthermore, a significant design problem was identified. The materials metaphor, illustrated in Figure 25, used for the texture of the ground and walls was not intuitive enough for the testers. None of them made the connection to the preciousness of the material (wood, stone, iron, gold, diamond). Players simply thought it was about colors, not materials. This was too confusing for them, so we decided to rework this design and use the color and color scale design literature to select the colors that best fit our purpose.

First of all, in this project the objective of the colors is to induce a sense of order with the chosen colors so that players can clearly differentiate a room with few problems from a room with many problems by going through three gradual intermediate rooms. Therefore, five colors must be mapped to the quality grades A, B, C, D and E of the SonarCloud code.

To do this, after a dive into the color and color scale designing literature, we chose to go for the theory of graphic semiology of BertinBertin [1967]. Although it seems very oriented in the realization of legends and maps. The theory of colors that the author proposes corresponds perfectly to what we need in order to overcome the problem of the intuitiveness of our colors. It is combined with Merwin and Wickens [1993] and Silva et al. [2007] which present three types of scale: the gray scale, the saturation scale and the rainbow scale (also called spectrum scale).

Figure 25: Room example using the materials tile set.

They found that the gray scale is the most effective for ordering elements next to each other, and that the rainbow scale is better for absolutely estimating the order of elements taken separately. The main weakness of the gray scale and the saturation scale is that each of them "provides only a limited number of distinct levels"Silva et al. [2007].

In the context of this study, only one variable needs to be mapped to color. This implies the use of a "univariate color scale" as discussed in Silva et al. [2007] which could use any of the three scales described above. In addition, the inclusion of redundancy by combining multiple visual elements such as color, saturation, or brightness would have some advantages, like overcoming visual impairments or even improving the clarity of the scale by taking advantage of the strength of each element.

Thus, considering the task we expect users to perform with the ordinal nature of the data we have. We decided to go with a **modified version of the rainbow scale** (1) to help users have an absolute understanding of room quality.

However, the purple color is removed because it is too close to red, resulting in the following range: brown, orange, yellow, green and blue. But, in our context, the colors can't just fill the wall and floor tiles, there are some color variations in these elements to make them look like walls and a floor. Also, fully saturated colors are not well suited for a video game where aesthetics are important. Therefore, a tile set using derived versions of the colors was designed by modifying manually the colors of the original tile set. An example of a room using this tile set is shown in Figure 26.

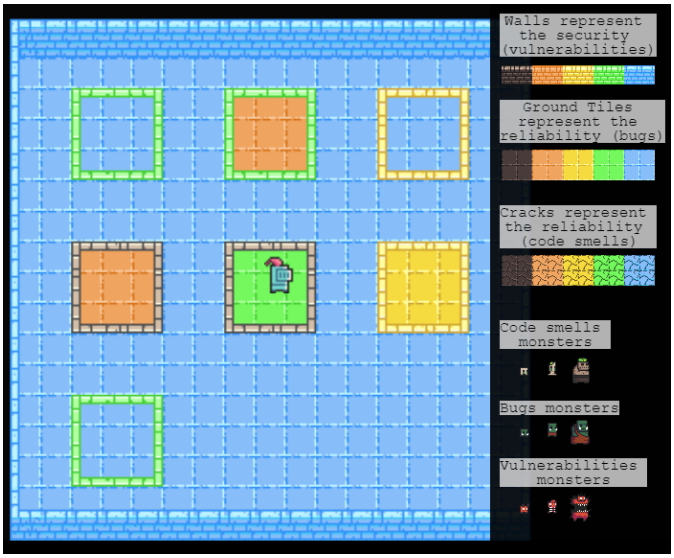

Figure 26: Room example using the rainbow scale tile set.

In practice, this scale is very differentiable. However, it is too different from the classic dungeon crawler that follows a rather dark and more homogeneous color palette to remain consistent with the universe that takes place in a dungeon atmosphere, the Figure 27 illustrates this with screenshots from the 10 best-seller games categorized as "Dungeon Crawler" on Steam[37]. To counteract the latter problems, another set of tiles was considered. Instead of modifying manually the color of the walls and ground textures, one texture is selected, and the others are derived from it using color transformation tools to try to keep as much consistency as possible.

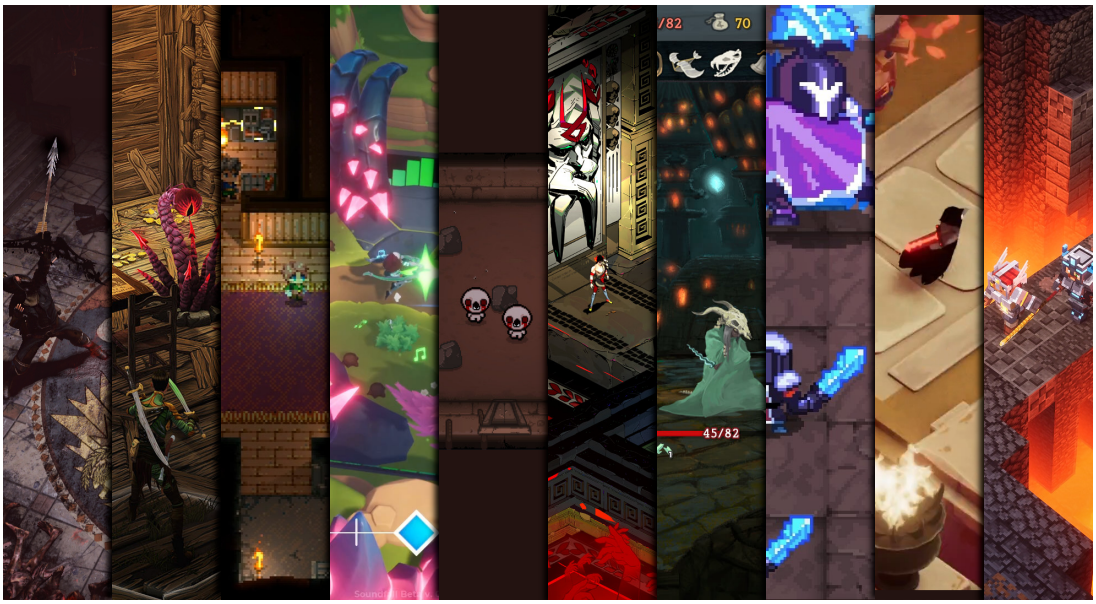

Figure 27: 10 best-seller games categorized as "Dungeon Crawler" on Steam.

We then considered the option of **grayscale** (2) visible in Figure 28, ranging from the lightest

[37] `https://store.steampowered.com/tags/en/Dungeon+Crawler#p=0&tab=TopSellers` accessed 17 May 2022

gray possible for the game to the darkest gray possible. The problem here is that in order for the game to continue to have distinct textures, we can't go too dark or too light. Otherwise, the tile textures disappear. This greatly limits us in a grayscale approach and increases the difficulty for users to differentiate them.

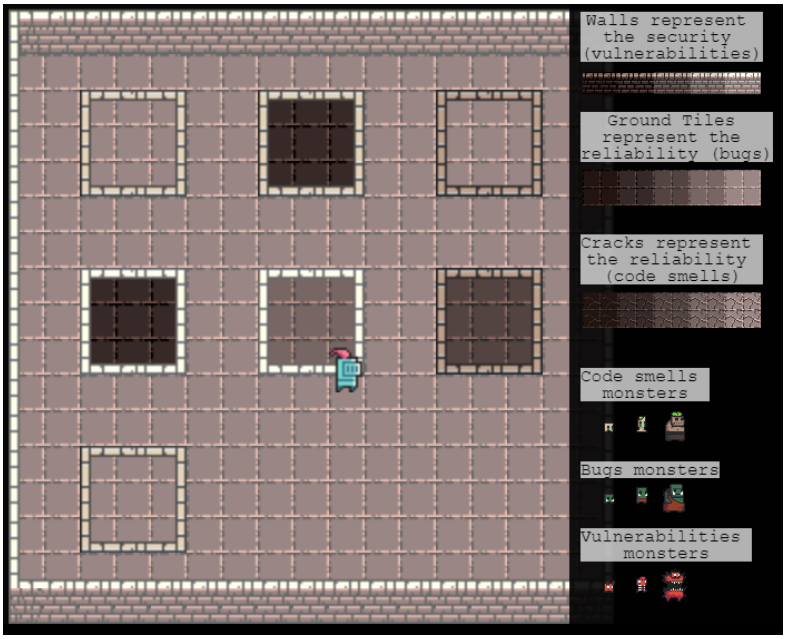

Figure 28: Room example using the gray scale tile set.

Based on the conclusions drawn earlier and Silva et al. [2007]'s recommendation regarding the use of redundancy, a new scale was designed by exploiting the hue and brightness of the colors; in this way, redundancy is added without changing the basic look of the textures and retain the dungeon-like feel of the game, it is a mix of the rainbow and the grayscale. As recommended in the same article, it is a **grayscale with two additional colors** (3). Going from black to white but passing through red and blue. The goal is to have a better level of comparison and distinction at the same time.

As shown in Figure 29, the difference in level is not easily distinguished. The saturation level has been increased to create an original and distinguishable look to the final version shown in Figure 30.

Figure 29: Room example using the rainbow & gray scale tile set.

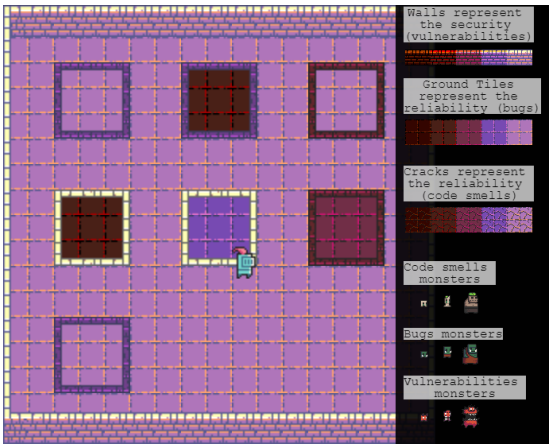

Figure 30: Room example using the funky tile set.

9.1.3 Game

The tutorial room has also undergone a major overhaul. Most of the text guidance has been removed, because pilot tests showed that people did not read it at all. The Figure 31 shows the evolution between the first version to the refined one, with the information being lighter and more digestible for the players.

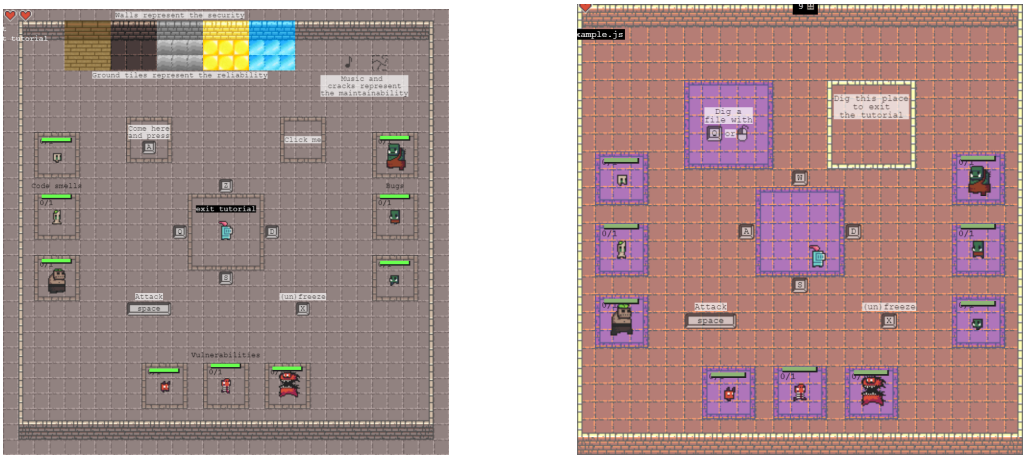

Figure 31: Tutorial room evolution (enlarged versions in appendix Figure 46 and Figure 47).

Also, to solve the problem of not understanding the game once the player leaves the tutorial. Information has been added when opening the freeze mode. In this way, if a user needs information, they can access it at any time during the game by pressing the corresponding key to freeze the game and display the overlay as shown in the screenshot in Figure 32.

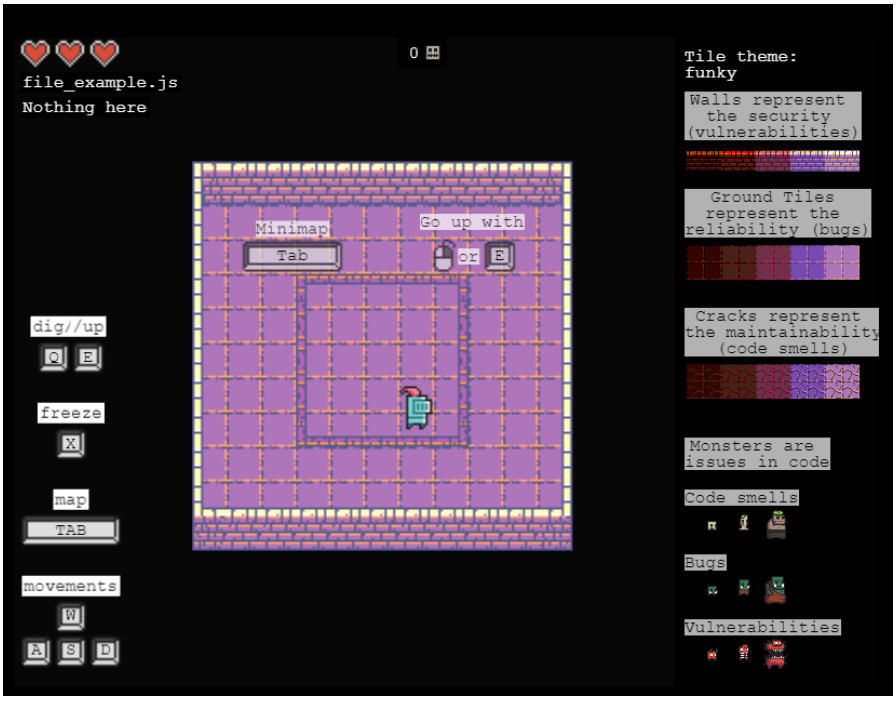

Figure 32: Freeze mode information panel.

Next, the combat system was redesigned to make it more usable and intuitive for players, as we noticed that most of the participants in our pilot test had difficulties with the system. Initially, the attack hitbox was a rectangle in front of the player that corresponded to the direction in which the player was walking as illustrated by the Figure 33. This mechanism made combat more difficult because in this game the player has to run away from the monsters. So we decided to switch to an attack that goes around the character instead of an attack that just faces it to make it easier to fight in the game as seen in Figure 34.

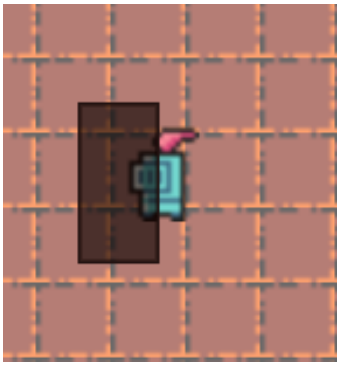

Figure 33: Front attack. The black box represents the hitbox.

Figure 34: Circular attack. The black box represents the hitbox.

Finally, we tackled all the game breaking bugs identified by the pilot testers to avoid any issues during the subsequent interviews.

10 Results

In this section, the themes identified through the coding of the interview data are presented. The results are presented as well as the percentage of interviewees who mentioned the same thing. In parentheses is the actual number of interviews that mentioned the theme in question. When it is relevant, the data will be illustrated with quotes from the interviews' transcripts. Also, some items are not discussed, because they were only mentioned by few person and concerned an irrelevant topic for this study (e.g. "I like the animation of the character"). Ten themes emerged from the interview data: Mapping, Usage, Gameplay, Navigation, Target user, Engagement, Information, Art, Project selection and Performance. The themes "usage" and "art" are not discussed in separated specific sections as they are under-discussed and less relevant, instead they are embed in other appropriate sections. In addition, four sections will address major aspects of this research that were not themes on their own in the data analysis: Music, Missing features, Freeze feature and Color scales.

10.1 Mapping

The main core of this study is the mapping between the code metrics and the game components, each interviewee commented at least once on this topic.

One of the key relationships between the measurements and the game is the correspondence between SonarCloud ratings and the texture of the floor, walls, and the cracks. 75%(9) of participants understood that color represented overall quality. Four respondents expressed isolated opinions, one said that the correlation between color and music was useful (two other participants said that music represented danger), another suggested having different colors for the walls and floor. Furthermore, 83%(10) of the participants, when they entered a perfect room, i.e., a room with only A-tiles and no monsters spawning as shown in Figures 35 and 36, stated that the code represented by that room must be perfectly clean.

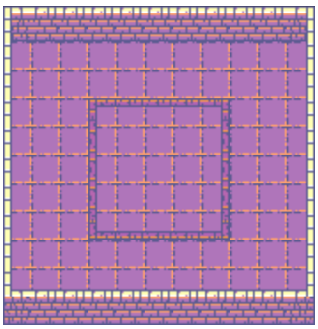

Figure 35: Perfect room (funky tile set).

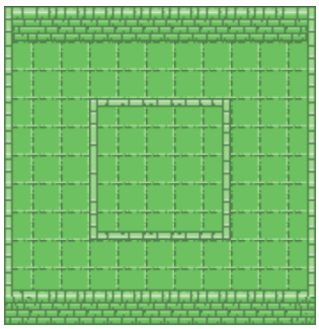

Figure 36: Perfect room (rainbow tile set).

Two interviewees struggled to understand the three main general metrics (reliability, safety, and maintainability). Also, overall, the maintainability metric was more problematic than the others. 50%(6) of the participants expressed that they could not find cracks and four of them were confused and tried to find them, in fact they expressed various theories like:

Interviewee #3: So I get the cracks must be the perimeter around the sub-directories.

Concerning the monsters in the game, of the eight interviewees who discussed code smell monsters, only one was confused because they did not use the same word for it in their company. Vulnerability monsters were not a problem for participants, except for one who thought the wall itself represented the vulnerability. Finally, bugs were discussed by seven participants, two of which thought the floor itself represented the bug instances. Overall, 67%(8) of the participants explicitly said that they understood that each monster represents a SonarCloud issue. Similarly, no participants were confused by the number of monsters appearing in a room, those who did mention it said it was intuitive or that the monsters in a room simply came from the subdirectories. Another monster characteristic discussed in the interviews was the meaning of the size, in fact two participants did not understand the connection between monster size and code metrics.

Regarding the combat aspect of the game, 58%(7) of the participants felt like not getting any information from it. In addition, eight participants stated that when they felt overwhelmed by the monsters, it meant that the code was bad. Only two participants found out that a room in which they were able to kill all the monsters meant that the debt of this file was repayable by an individual.

In the end, 75%(9) of the participants found that the room structure was mapped to the project structure where a room was either a file or a folder and the squares on the floor are sub-elements contained in the folder. 50%(6) of participants understood that the map also represented the directory structure of the project, it is important to note that two participants did not use the map at all.

10.2 Information

In The Coding of Isaac, the information is displayed in several ways. The interviews provided a lot of feedback on this topic, how it is introduced in the tutorial, the general feeling about information, and the labels used.

Regarding the information in general, 33%(4) of the participants said that there is some learning curve to go through to fully understand the game, since there are several metaphors used that have to be learned like the monsters types, the ground and wall textures. In addition, two participants found the amount of information too large, and one participant stated that the information was complex. On the contrary, only two participants found the information intuitive.

The game also includes a tutorial that is supposed to introduce players to the basic concepts of the game. During the interviews, 75%(9) of the interviewees commented on the tutorial. Most of the comments were personal opinions or experiences that were not shared by anyone else as: controls not noticed, trying to click on a monster, needing more help from the interviewer, finding that there was the right amount of information, and missing information about what the squares on the floor are. But two observations were made in multiple interviews. First, three different participants stated that a contextual introduction to what the game actually is was missing. Second, interviewees #4, #7, #9, and #12 left the tutorial before exploring all the features presented there, the controls for the movements, attack, freeze mode, maps, navigation with dig and go up. Also, some of these participants effectively missed some of the commands as noted in other sections.

Additionally, several dynamic labels are used in the game to display data. The most discussed labels are a) the numbers displayed on the tiles, b) the file and folder names displayed in the upper left corner, and c) the total number of monsters alive and going to spawn. These labels are highlighted in the screenshot in the Figure 37.

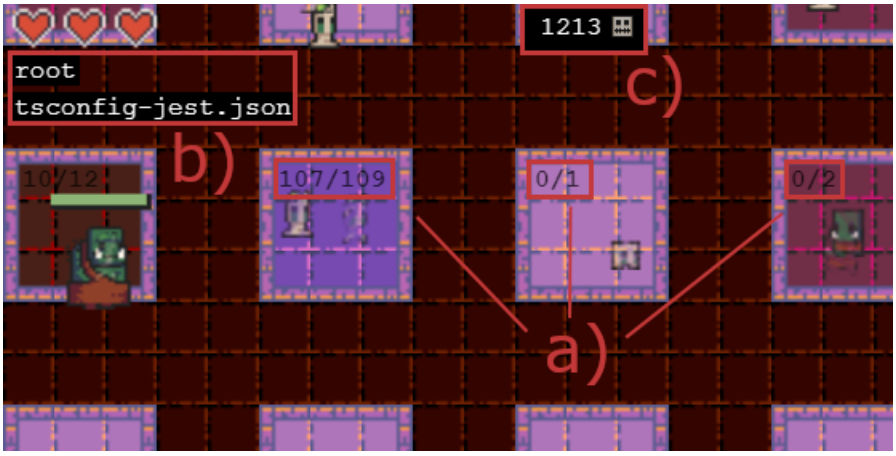

Figure 37: Labels in the main game screen: a) the numbers displayed on the tiles, b) the file and folder names displayed in the upper left corner, and c) the total number of monsters alive and going to spawn.

The number displayed on the tiles was discussed in nice different interviews. Three participants understood directly that the number corresponded to the number of monsters that will appear at that location. Three of them had to deduce the meaning empirically by moving from room to room, killing some monsters, and observing the impact on that label. The last three interviewees did not fully understand the label. Also, three participants who understood the meaning of the label said that there was a problem for them, because the number is just the number of monsters, they do not know the type or size of the enemies, so comparing the rooms on that basis could be misleading.

The file and folder name indicator was noticed and understood correctly by seven different interviewees. On the contrary, two participants did not notice it and said that it would be interesting to have this piece of information.

The last label discussed was the total number of monsters. It was only noticed by 25%(3) of the participants, but they all understood it correctly.

10.3 Freeze feature

Several people commented on the freeze feature. There are two main use cases for this feature: reminding the player of essential game information (monster types, color meanings, and commands) through the information panel, and stopping time to let the player explore the room at ease, additionally unlocking a tool tip providing details about monsters and subdirectory squares that appears when these items are hovered over with the mouse.

Regarding the informational aspect of the freeze frame feature, ten participants reported that it was useful or that they had to use it at least once to remember some information. For example, in the interview #12, after leaving the tutorial, when the player entered the first room of the *Brave* project, they immediately froze the game to find out what the monsters were, discovering that there were some bugs and a lot of code smells in this specific project. On the other hand, two interviewees had troubles remembering the freeze frame button, in this case; they stated that they were looking after the key for the freeze mode. Three others missed it in the tutorial, they did not try to press that key in the tutorial. In these situations, the interviewer helped

the participant not to waste too much time by reminding them of this feature. In the case they missed the key, the interviewer told them about the key when they faced a problem addressed by this feature, for example, knowing what the types of monsters or the textures stand for, or having troubles to fight the monsters and observe the elements at the same time.

In addition, through the interviews, some specific isolated opinions were collected on the freeze feature. In interview #1, the participant thought the information panel was unattractive, with too much information, and that it was "devy". By "devy" the interviewee might means that it looks like something for developers, an expert system that would be less attractive but more focused on the efficiency. Interviewee #6 had trouble understanding that the squares on the floor are subdirectories and tried to find them in the information panel. Finally, participant #10 who is an accessibility expert commented on this panel. First, they stated that the icon for the "Q" key was not clear. Second, while playing, they sometimes used the arrow keys, so they suggested to add it to the keybinding list. Third, when reading the monster information, as illustrated in Figure 38, they were confused by the label "monsters", in fact they expected there to be four categories of enemies: Vulnerabilities, Bugs, Code Smells and *Monsters*. And they suggested adding images for this fourth category as the following quote indicates

Interviewee #10: Because just by these images [of code smells, bugs and vulnerabilities], I know; what is a code smell, what is a bug. But I don't see any monster pictures. So that would help.

Figure 38: Monsters information in the freeze panel.

Regarding the second purpose of this function, which is to allow the player to explore the rooms freely, 25%(3) of the participants did not use the freeze feature to observe or navigate with the mouse. Three other participants did not use it to observe the room but only to navigate. This leaves 50%(6) of the interviewees who did use it, two of whom found it on their own, the others were informed by the interviewer. Similarly, five participants, or 83% of the participants who used this feature, found it useful. In addition, freezing the game unlocks two additional interactions. First, hovering over a file or monster displays a tool tip with information as seen in Figure 39. During the interview, two participants did not use it at all and five used it directly on their own, the others were mixed. Second, the player can click on any monster to access the details of the issue on the SonarCloud website. This feature was not used much during the interviews, indeed six interviewees did not use it at all, four were told by the interviewer about it, and only one respondent used it intuitively while the last one found it randomly when trying to click on a tile under a monster. Furthermore, of the six participants who tried this feature, four of them found it useful.

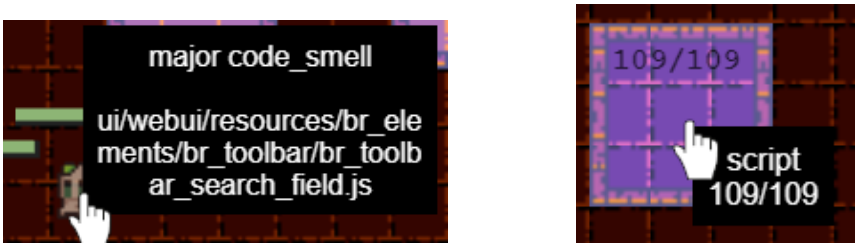

Figure 39: Tool tips. The left image is the monster tool tip showing the type of issue and absolute path to it in the project file structure. And the right image is for squares on the ground which are subdirectories or files, in the tool tip there is the name of the file/directory and the number of monsters that will spawn from this place out of the total number of issues in it.

10.4 Music

In the interviews, 17%(2) of the participants complained about the volume of the music being too loud, and three participants wanted to adjust the volume themselves, which is not yet possible in the game. For the interviews conducted online, it was not possible for the participants to reduce the volume without changing the volume of the call. Only two were able to reduce the volume because they were in-person interviews, but they did not complain about the volume.

Furthermore, regarding the music itself, participant #11 did not like it, #4 found it too nervous, and during interview #8, the participant suggested that the music should be customizable to have their own choice of music as quoted below.

> *Interviewee #8: Yeah, because, I'm coding. So it would be good if it was my choice of music.*

In addition, as mentioned in Section 8.3 Protocol, before the beginning of the interviews, the intuitiveness of the music ranking is tested. To do this, participants are asked to listen to the music and rank it. Then, the Kendall's tau-b correlation coefficient is calculated, it is done using a tool from Wessa [2022], of each ranking made by the participants to assess the accuracy with which they ranked the music in comparison to the one defined by the authors. In this test, the absolute value of the coefficient is taken into account because whether the participant orders the music in the same way as the authors or in the opposite order, it is still the right order. As a result, 50% of the participants found the same order (coefficient = 1), and the others had a value of 0.4 or less, the results are summarized in Figure 40.

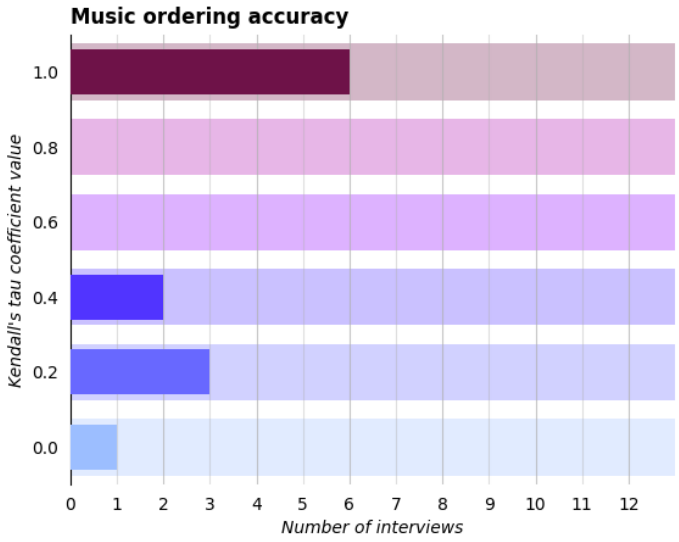

Figure 40: Kendall's tau-b correlation coefficient of the **music** scale ordering experiment.

10.5 Color scales

The color scales intuitiveness was tested in the same way as the music presented in the previous section. The protocol used is extensively detailed in the Section 8.3 Protocol. The results of the experiments on the funky and rainbow color scales are summarized in Figures 41 and 42 respectively.

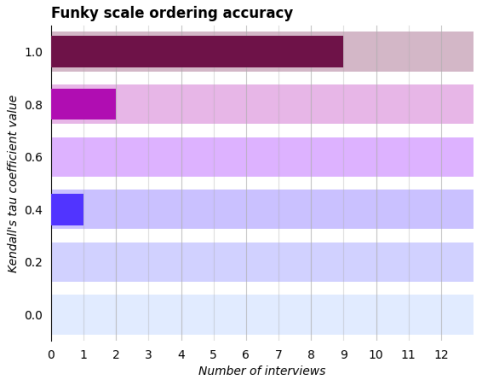

Figure 41: Kendall's tau-b correlation coefficient of the **funky** color scale ordering experiment.

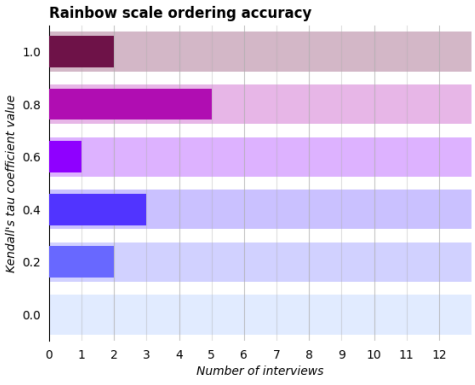

Figure 42: Kendall's tau-b correlation coefficient of the **rainbow** color scale ordering experiment.

During the interviews, 83%(10) of the interviewees found that the scales were ordered left (dark) to right (light) (11 & 12 didn't say it). The funky scale has been criticized by three different participants, one said that the different colors could not be discriminated enough or that the inverse order would be more logical, even two participants had trouble to get the order from left to right and had to deduce it by exploring rooms and thinking about it. On the contrary,

nobody had trouble understanding the order of the rainbow scale except one participant which explained it because they are colorblind and could not see the difference between some of the chosen colors as they rely only on the hue difference to be discriminated. However, as the funky scale relies on more characteristics of colors (e.g. brightness, saturation) it is more usable for people with colorblind disability.

10.6 Navigation

Several participants said they felt compelled to navigate and move from room to room. Interviewees #5 and #10 said it was necessary in order to use the tool properly and interviewees #9 and #10 stated they felt compelled to leave the rooms because of monsters spawning pace.

The most commonly used navigation way by the participants is the dig and go up controls. As mentioned in the Section 6 Design, this has been design to be used as the first way to navigation. During the interviews, almost all participants used it on their own, only #12 missed it completely in the tutorial because they clicked on the exit tile before digging into the example files. Also, #7 and #8 missed the GO UP key, in fact #8 dug into an example file and left it using the map and #7 directly clicked on the exit tile without exploring every part of the tutorial. In these three situations, the investigator reminded the participant what the control was when they appeared to be stuck in a file or walking around the first room without digging into any files for a few minutes. This occurs even with the reminder of the controls in the lower left corner of the freeze menu, but the label DIG and GO UP can be confusing, as indicated by participant #10 who read the label, as shown in Figure 43, as "dig up".

Figure 43: "dig" and "go up" labels in the freeze panel.

Navigating the game is a key feature of this tool and monsters can make it more difficult. Indeed, 58%(7) of the participants found navigation difficult when there were monsters. The navigation problems were anticipated by the authors, and to address them, an alternative navigation method is implemented in the game. Using the mouse, the player can left-click on subdirectories to dig in and right-click anywhere to go back to the parent folder.

During the interviews, despite the fact that the feature is in the tutorial via blinking mouse icons in the navigation explanation, only player #10 found it on their own, and after some time during the game session, they had difficulty remembering this command. Also, participant #1 did not find the control but also did not complain about the navigation problems, and did not seem to have any difficulty navigating, so the interviewer did not have to tell them about it. The last special case was #2 who complained about navigation problems but due to time constraints the interviewer was unable to show them the control.

The other 75%(9) participants were told by the interviewer how to navigate with the mouse. Five of them because they explicitly complained about the difficulty of navigating, and the others because at some point the context of the interview was conducive to introducing this new interaction. Finally, four people stated that this feature is useful, all of them were introduced to it by the interviewer, three of them (#3, #7, #9) because they explicitly expressed the navigation

problem and the last one (#12) was introduced to it because the timing of the interview allowed it.

In addition, participant #8 had particular difficulty with click navigation because they did not have a mouse but only a track pad, which is less convenient than a mouse for this specific interaction as designed in this study.

This game is designed to be played with one hand on the keyboard and the other on the mouse to exploit its full potential. But this may not be very clear at first glance. For example, at the beginning of interview #10, the player starts with exclusively the keyboard, then exclusively the mouse, and only after navigating for several seconds does they begin to fully use both hands to navigate effectively. In addition, this may partially explain why 11 out of 12 participants did not find the mouse controls. The following quote illustrates the problem perfectly.

Interviewee #4: I was just using the keyboard. I didn't know I could use the mouse.

The last way to navigate the game is to use the map illustrated in Figure 44. All participants opened it at least once. 42%(5) of them were invited by the interviewer to try it, in order to test this part of the game as well. Regarding the usability of this feature, four players said and showed that the map is usable, five showed that it is rather bad and the last three players remained neutral about the usability of this feature. In addition, during the first interview, the user suggested that a mini-map in the map screen to quickly navigate with the mouse from one location in the file system to another would be a great addition to the product. Despite the fact that there is a search bar in the menu that is already supposed to help users in this specific task, but as observed, only one interviewee noticed it and tried to use it, but they tried to type only the name of the file they were looking for when instead they should be writing the full path to it.

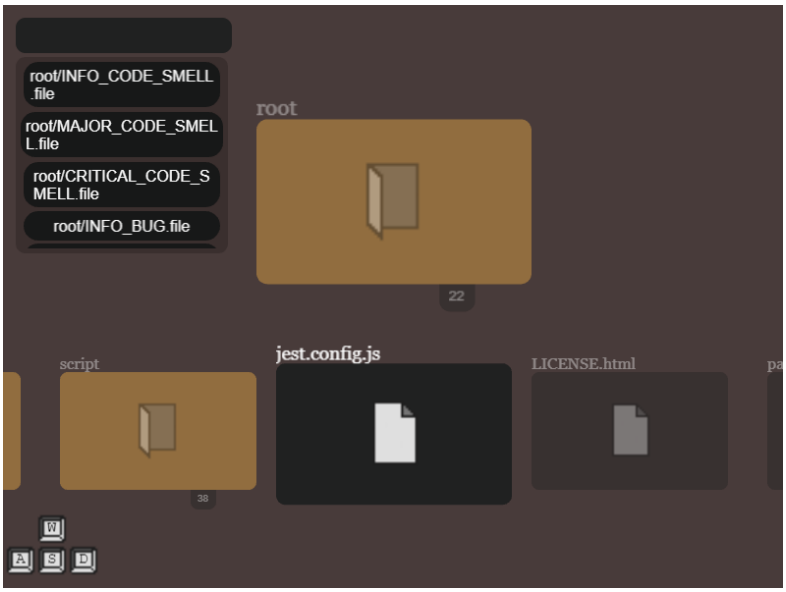

Figure 44: Map screen.

Two participants found navigation more difficult than it could be. They explain this by the fact that they are not familiar with the code base and file structure of the project, so in a game where

the world is the file structure of the project, it is difficult to navigate.

10.7 Gameplay

50%(6) of the participants commented on the gameplay of the product presented here. During the interviews, 33%(4) of the participants complained about an annoying delay between character attacks. Similarly, two participants complained about the lack of damage inflicted by the character they were playing.

In addition, three respondents felt that the combat was not diverse enough, because there is only one action available to fight and the monsters only run straight at the player. Conversely, 67% (8) of the participants found the pace of the combat overwhelming with all the monsters coming towards them. Interestingly, one participant felt that the combat was not diverse enough and that the pace was too fast.

In addition, 33%(4) of the participants found the fact that the monsters of a specific room were always spawning when entering it, whether or not they had been killed before, to be confusing.

Finally, the participant #11 stated that they did not find the meaning of having hearts (the player health bar) in the game.

10.8 Goal of the game

A very popular theme mentioned by 11 of the 12 respondents was the purpose of the game. 50%(6) of the participants, said that they did not find any well-defined goal in the game, but in the end they thought that it is to kill all the monsters. Another participant just considered that there was no clear goal of the game, and three respondents stated that the goal was clearly to kill all the monsters. The participants missing a clear goal for the game mostly asked verbally what it was.

> *Interviewee #6: What I am still wondering is what is the objective of playing the game, but maybe this is just a game with no objective. And so there is no reason to think about that. But yeah, this is the main remaining question I have. Why am I playing the game.*

In addition, seven participants expressed that they did not feel they had any agency in the game, i.e., they did not feel like their actions had an impact on the game. Among these, it is interesting to note the opinions they expressed regarding the goal of the game. Indeed, one of them thought that the goal of the game was to find the worst file in the code base and kill all the monsters in it, another one did not talk about the goal of the game at all during the interview. Finally, the remaining five interviewees expressed both a feeling of not having agency in the game and that the game has no goal.

44%(4) thought that the objective was to find the worst file to fix. Only interviewee #3 understood the actual goal as it has been designed, as shown in the following quote.

> *Interviewee #3: It provides that implicit feedback and it provides a nice summary of all the numerical metrics. Everything kind of works together to provide that implicit feedback.*

Additionally, in two interviews, the participant stated that the goal of this tool would be to be fun. In particular, interviewee #3 articulated this by talking about wanting to have a more cathartic game, in which one could obliterate the enemies and use the game as an outlet.

> *Interviewee #3: [..] it's not as cathartic as if you just go around killing them all and have your vengeance.*

In addition, interviewees were asked if the tool would motivate them to correct the technical debt of a project they were working on, six responded positively, two said that they were not sure because they would have to try it with one of their projects to find out, and four did not think it would motivate them and gave some reasons why. First, according to #2, the tool lacks a prioritization feature, such as a way to filter problems by type or severity to avoid having a huge heterogeneous horde of enemies attacking you. Second, according to #6, #11, and #12, this would be due to the lack of gamification features which will be more detailed in the next section. Conversely, two participants explained that the tool can be motivating because it makes you realize how bad the code is when you see the overwhelming number of monsters.

10.9 Target users

This work is interested in who would be the users of the tool presented here, if only the developers would be interested. Thus, the interviewer asked the same question to each participant.

> *Who would you see using this tool?*

The interviews identified 12 categories of users. Six of them were mentioned more than once. The category that was most discussed in all interviews was developers in general. Although according to the opinions of the participants, it is not sure whether the tool would be interesting for them or not, as two interviewees said it would not be useful for them, four were positive and one was not sure if it would be the case for all developers, while it certainly would be interesting for game developers. The second most cited category was managers, here only one participant thought it would not be interesting for "strong individuals" like managers, another participant was not sure, but four of them were positive about the interest of this tool for managers. Then we had four positive opinions about adopting a tool like this for students and three for teachers. In addition, two people thought that a UX designer would enjoy the tool and two thought that the team as a whole would be a good user of the game. Another role category that we obtained evidence on was data scientists. Two data scientists participated in the study and both stated that the tool would not be useful to them. Finally, three specific roles were cited as potential users by various participants, they are: Functional Analyst, Solution Architect, and Tester.

In the end, the answers created a list of ten categories of potential users that are summarized in the Table 10.

When discussing the potential users of this tool, three participants said that traditional dashboards, as presented in the Section 3 Related Work, help to get the information more easily. Specifically, the interviewees, #2 and #11, working in the field of data science said that they do not use visualization tools at the moment. Conversely, interviewee #9 who is a developer uses visualization tools (e.g., Jira issue tracker, Github issues with the repository). They stated that traditional visualization tools are more comprehensive than the one presented here. Additionally, it is interesting to note that participants #4, #6, #8, and #12 were already using SonarCloud

Category	Positive	Negative
Developer	4	2
Manager	4	1
Student	4	0
Teacher	3	0
UX designer	2	0
Team	2	0
Data scientist	0	2
Functional analyst	1	0
Solution architect	1	0
Tester	1	0

Table 10: Potential users mentioned during the interviews.

or SonarQube and did not comment on this matter. All others had never used a visualization tool to manage code quality or technical debt.

10.10 Potential usage

As for the target users, the interviewer asked the same question to each participant about the potential use cases of The Coding of Isaac.

How would you see this tool being used?

92%(11) of the participants responded, only one did not respond and talked about something else at that time before running out of time for the interview. In the end, 10 different scenarios were collected across the interviews that are summarized in the Table 11.

Category	Occurrences
Debt fight engagement	7
Communication	4
Monitor debt	4
Track people performance	3
Complement classical visualization tools	2
Introduce technical debt in a teaching context	2
Management	1
Task list	1
, Assignment marking	1
Pair programming	1

Table 11: Use case scenarios mentioned during the interviews.

The most cited usage scenario discussed by seven participants is to engage people in the fight against debt and motivate them to pay off technical debt. Followed by communication, four participants imagine this tool being used in meetings to present the current state of the program code. In addition, one of them and three others imagine it more for tracking technical debt over time. From a management perspective, three interviewees thought that this tool could be used by managers to track each person's performance and the status of their code, another user said

that managers could also use it to assign people to pay off technical debt for certain parts of the code. The role of this tool as a complement to traditional software visualization tools and not as a stand-alone tool was cited by two different participants. Two others suggested using the tool for educational purposes to introduce the concept of technical debt to students in a more intuitive and tangible way. Subsequently, some scenarios were cited separately by different participants: to use it as a task list to select issues to fix or part of project to work on in order to pay off technical debt, but only with additional functionalities to support it; to use it to get an overview of the quality of students' work in order to grade it later; and finally to use it in the context of pair programming.

One interviewee pointed out that this game is also a tool that creates a kind of dilemma between having enough information and being entertaining and fun. In fact, two people explicitly said that it is a tool, but one said that the game aspect was then distracting and the other said that as a stand-alone tool, it was missing information. Two other participants stated that the product lacks information, and that it is particularly difficult to access detailed information. One went into more detail, indicating that information on evolution is missing.

Participants also discussed some requirements for using the tool. #3 and #9 agreed that the tool would be more suitable for large projects than small ones. On the other hand, #11 cautioned that it would not be much usable for very large projects because of the number of files and folders.

> *Interviewee #11: Like for big projects, with many directories, it could be somehow complicated, like moving around different rooms.*

Interviewee #3 spoke of a possible threat from users who would seek the challenge.

> *Interviewee #3: [..] if somebody's enjoying the game too much, then they would [..] purposely make bad code to have a harder challenge [..]*

But right after that, the participant completed their sentence by saying that:

> *Interviewee #3: [..] but I feel that's a minor side effect.*

Player #1 found that a major barrier to using this tool would be that managers would not like to let employees play a video game during work hours.

> *Interviewee #1: I feel like in the business aspect, people are kind of like, go, go, go, just test it. Like, you don't need to fool around with playing it and seeing how it works.*

On the opposite, #6 thinks that in order to be engaging, this tool has to be chosen by the user and not imposed by a hierarchical superior.

> *Interviewee #6: Maybe some people will not want to play this game and they will not want to use this tool. So yeah, I think that for some it would work but I would pay attention that it doesn't create more frustration for the people that don't want to play the game.*

The final aspect regarding the actual use of this tool that was covered in the interviews is the frequency of use. 9 of the 12 participants mentioned it. The results are summarized in Table 12.

Category	Occurrences
Agile sprint review	3
Release	2
Each week	2
Try by curiosity	2
Each day	1

Table 12: Frequency of use of the tool mentioned during the interviews.

Indeed, three participants imagine that this tool could be used at the end of each sprint in an Agile development life cycle during the sprint review in order to communicate and track technical debt. Two participants think that this tool could be used on a regular weekly basis, and one on a daily basis. In addition, two interviewees stated that the tool could be used to clean up code before releasing it to production. On the contrary, one participant stated that the tool could not be used in the later stages of development but only in the early stages, when there is still time to improve it. Finally, two participants see this tool as a novelty that people would try just to satisfy their curiosity.

10.11 Missing features

As mentioned in the Section 10.8 Goal of the game, three participants stated that the game lacked some form of gamification. One of them made this clearer by giving as an example a scoring system to further motivate the player to use the tool.

In addition, other participants suggested some features that could be added to the game. Respondent #1 talked about adding a mini-map in the upper right corner of the map menu, which would be clickable to help the player quickly navigate the entire file system. The same participant said that the information panel in freeze mode reminds them of "lore books" in other games. Indeed, in some other dungeon crawler games, the player can find information about the different elements of the game in a book, such as the monsters, their weaknesses, strengths, and a short description. The interviewee thought that this could be an interesting way to make the tool feel more like a game. Additionally, the same participant suggested having a message to inform the user when they have killed all the monsters in a room. This echoes the lack of gamification in the tool, and this feature would provide metrics on the player's game session to compare it to their peers, but the interviewee rethought this thinking that a good player is not necessarily a good developer, and rewarding them based on their game performance might not be relevant.

Participant #12 who said that the game lacked appropriate gamification features suggested to add a death recap feature. This idea is detailed in the following quote from that interview.

> *Interviewee #12: Oh, in that case, what would be great? I just think it would be really fun if, when you die, you see what code smell or what bug killed you. And it would give the developers a personal reason to go and eliminate it. It makes it personal.*

Also, one participant expressed a need for instant feedback in the game when they fix something in the code. Another respondent suggested associating the file type with a game element. For

example, a python file could have snakes crawling on the ground. Two other participants said it would be interesting to associate the file size with a game element.

10.12 Project selection

When the player enters the project selection screen illustrated in Figure 45 just after leaving the tutorial room, the investigator gives them a brief introduction about it:

> *The projects here are all open-source and publicly available on SonarCloud. For the sake of this interview, I'll ask you to select the recommended one, the Brave one.*

This moment of project selection was not long in any interview and the interviewee did not talk much during this time. Nevertheless, some interesting facts were observed through observation of the players' interactions. The first action of 42%(5) of the participants was to navigate through the list of projects, one of them tried to scroll by dragging the "scroll indicator", the circle on the right of the screen highlighted in the Figure 45, with the mouse, which is not implemented yet, and then they scrolled with the mouse wheel like the other four participants. 25%(3) of the participants did not notice the recommended section at first sight, the investigator had to give a little more explanation.

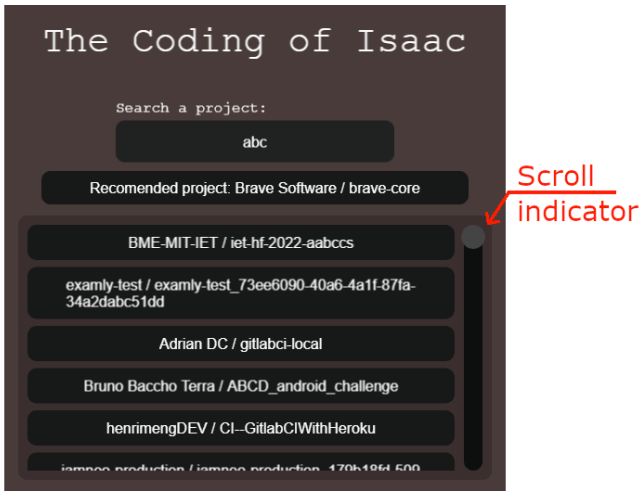

Figure 45: Project selection screen.

10.13 Performance

The topic of performance was not addressed often by the participants, in fact only three of them did, each regarding a different topic. Participant #9 found that it took a long time to load a specific issue page on SonarCloud when trying to access it from the game. Therefore, they suggested integrating SonarCloud directly into the tool on the side where there is free space, so that users can access SonarCloud details without having to open a new tab each time. Participant #8 triggered an out-of-bounds bug where the character goes outside the boundaries of the play area, in our context the play area is supposed to be limited by the walls of the room but we knew that sometimes walking in the upper left corner could trigger this bug. Finally, one user noticed

a drop in the frame rate because it was letting a lot of monsters appear. So, they suggested limiting the number of monsters that can appear to avoid this issue. This solution has already been discussed in Section 6.2.3 Spawn.

10.14 Others

In addition, as mentioned in Section 8.3 Protocol, questionnaires were completed by the interviewees at the end of the interview, which allowed us to verify and compare what each of them said during and after the play session.

As the Intrinsic Motivation Inventory scale requires, we calculated for each participant the average for each of the three previously defined variable using the score of the questions. Then a final average for each variable was made usign the average of each participant. The final results for each variable is as follow:

- Interest/Enjoyment: 5.17 out of 7.
- Perceived competence mean: 4.39 out of 7.
- Value/Usefulness: 5.17 out of 7.

Each variable has a mean of over 3.5. We can say that, on average, the volunteers found the tool interesting and enjoyed trying it, found some value in it, or found it useful. Finally, they also felt that their own skills in using the tool were correct, although we infer a little less positivity from this because it is a little below the other variables.

11 Discussions

Many themes were discussed during the interviews. Based on the data collected from these, this section will discuss the answers to the research questions. As a reminder, the overall goal of this research is to determine if a video game can be a medium to convey information about code quality and technical debt in the context of a code quality visualization tool. This objective is divided into two main research questions.

> *RQ1: How intuitive and accurate is the video game medium to visualize the software technical debt for industry practitioners and students?*

This first research question is too broad to be answered directly. It is therefore divided into three more specific questions and by answering these sub-questions, it will become possible to answer the general research question.

> *RQ1.1: How accurately can the developers distinguish between different aspects of technical debt in the game? For example, if a file contains many vulnerabilities but no bugs, does the player understand that the code base is vulnerable but not buggy? How precise is this understanding?*

Using The Coding of Isaac, users are able to distinguish a part of a software project with or without significant technical debt. They are particularly accurate in detecting the perfect part of the project that contains no problems and is correctly rated by SonarCloud on the three aspects of maintainability, reliability and security. Due to the absence of poorly rated parts of the project for maintainability, users have a hard time fully understanding it. For reliability and safety, they can observe the different textures of the walls and floor as they explore the rooms, which helps them understand which rooms are better than others on these criteria. Finally, when it comes to the type of problems related to a case, users can easily understand whether it is homogeneous or not using the different monster skins that reflect it and are easily differentiated.

Also, as expected, the numerical values of the squares representing subdirectories and files can be misleading because the type and severity of problems are hidden in the value. Furthermore, even though the general code quality metrics are represented by the components of the squares, they are not always sufficient to compare two subdirectories. This led some participants to consider that to fully understand the state of a project's technical debt, it is necessary to go through each room one by one.

Summary **RQ1.1**: In The Coding of Isaac, the different aspects of technical debt visualized in the game can be accurately understood within a specific room if the player has had the opportunity to explore and become accustomed to the different mappings. With respect to the overall state of a project's technical debt, due to misleading comparison criteria, users might have to explore the game at length in order to fully understand it.

> *RQ1.2: What game mechanics can be used to allow the player to naturally relate to the underlying code quality metrics? The objective is to find game mechanics and code quality measures that, put together, allow the player to understand the connection with little or no explanation.*

Several game components are used in this research to convey information about the quality of a software project's code. One of the main categories of components exploited is the visual element. It includes the texture of the floor and walls, the cracks on the floor, the monsters with their different types and sizes. These elements are generally understood by the participants, with the exception of the cracks, which confuses most of them because they do not find them in the game. Similarly, interviewees often use the freeze information panel to understand or remember their meaning, with the exception of monster size which seems to be more intuitive.

Additionally, there were two types of textures for the floor and walls. At first glance, the funky textures are more intuitive for users to order than the rainbow textures. But in the end, all players were able to guess the correct order after exploring the rooms and taking a look at the freeze information panel.

Moreover, the most intuitive mapping of the game components is the monsters to the issues. Participants understand that they are a threat and that they must eliminate them, at least in the game as a player. When they start to feel overwhelmed by these elements, they understand that something is wrong with this part of the software project. Paradoxically, some of them feel that the combat does not give them any information. In fact, the metaphor of the monsters' life as the time estimated to solve the issues was not clear to the users.

Finally, the parallel between the structure of dungeon rooms and the structure of directories is one of the best understood metaphors. It is important to note that there are many elements that support this metaphor such as the file and folder names in the HUD and the map screen that is more like a file structure exploration tool than a game map.

Summary **RQ1.2**: Multiple game components in the game are intuitive in conveying information about code quality metrics. First, the visual elements, especially when made explicit by a document like the freeze information panel. Second, the metaphor of monsters as code-threatening elements like problems. Finally, the mapping between the structure of the environment and the directory structure of the software project.

> *RQ2: How useful is the video game in visualizing the technical debt of software in the field?*

As with the first research question, the second is too general to be answered directly and is divided into four specific questions.

> *RQ2.1: How much does it encourage developers to repay some debt?*

In general, participants had mixed views on this question. Half of them responded positively, thinking that it would motivate them to correct the technical debt. Conversely, three participants explicitly stated the opposite and that this was due to the lack of gamification in the tool. In fact, as discussed earlier in this work, the tool was designed as a game, but without an emphasis on engagement. Instead, the authors considered the game components only as a way to visualize code quality metrics. Thus, the lack of gamification was intended, but since the tool is a game, it is understandable that participants expected to find some gamification that would engage them in using the tool. Additionally, because it is a game, aspects of gamification can be easily incorporated into this product in the future.

In terms of motivation to fix the technical debt, one participant felt that the tool lacks a filtering feature to avoid having a huge heterogeneous horde of monsters rushing at you. This was one

of the features the authors had in mind that could not be implemented due to time constraints. Kienle and Muller [2007] consider it a mandatory feature for a complete visualization tool. This creates the belief that the requirements for the traditional visualization tool and gamified tools are similar, or at least overlap. In addition, one participant cautioned about potential scalability issues for very large software projects, which is a common pitfall in software visualization tools as indicated by Bedu et al. [2019] and presented as a non-functional requirement by Kienle and Muller [2007].

It is also interesting to note that the feeling of being overwhelmed by monsters is part of the visualization. The game was designed to give players this feeling when the code presents many problems. In fact, the majority of participants said that they felt overwhelmed by monsters at some point and that this meant that the code related to the room was bad. Thus, if a filtering feature is implemented it should be done carefully so as not to break this feeling.

In addition, two participants felt that they should try the tool on their own project to answer this question. During the evaluation, each participant tested the tool on the same project to gather usable data for the research. However, the tool is designed to be used on the project the user is currently working on.

Summary **RQ2.1**: The tool encourages users to correct technical debt. It could be more engaging by incorporating more gamification aspects into the tool. Also, the common requirements and pitfalls of this type of tool could be similar to those of traditional visualization tools.

RQ2.2: Who would be the target user of such a tool?

RQ2.3: How often would this tool be used in the workflow of a target user?

RQ2.4: In which context would this tool be used?

The other three sub-questions of RQ2 are closely related, and are therefore addressed together because they require related answers about: **who** will use the tool **when** and **for what**. During the interviews, two broad categories of users were mentioned by the participants: practitioners and academics.

First, according to the participants, in industry, developers and managers might be interested in a tool like this. Developers might want to use it to track and visualize their technical debt in an attractive way, or to communicate the current status of technical debt in the code base to managers and stakeholders less in the code. Managers might appreciate this tool to track technical debt across the project, or to track the performance of individuals, or even to find parts of the project that need work and assign people to those tasks. These use cases involve regular use of the tool. But, it can also be used at each sprint review in a scrum development cycle as a communication tool between all stakeholders, or just before releases to assess the quality of the code. Similarly, it can be used as a task list or as a way to assign people to a part of the project, on a daily or weekly basis.

Second, some participants discussed the use of the tool by students and teachers. They indicated that teachers could use it as a fun way to introduce the concept of technical debt to students, or to get a sense of the quality of the code produced by students to help teachers grade the work. In these contexts, the tool would be used on an ad hoc basis to introduce the concept of technical debt or at specific times to grade student work. In terms of students use of the tool, participants imagined that they would use it to produce good quality code for their work.

In addition, a few participants discussed that because of the playfulness of the tool, "strong individuals" and "people in the business aspect" would not understand or accept developers "to fool around playing it". Some also criticized the tool in general, comparing it to traditional visualization software, claiming that the latter would be more comprehensive and simpler than the one presented in this research.

Summary **RQ2.2**, **RQ2.3** and **RQ2.4**: The Coding of Isaac could be used by developers, managers, teachers and students. They could use it to monitor technical debt, track the performance of individuals, introduce the concept of technical debt, and engage people in the fight against technical debt. The tool could be used at key moments, such as the end of a sprint or the release of a new version, or on a weekly or daily basis. Finally, the fun aspect could be a barrier to its adoption by some people.

The results of the evaluation led to three additional findings that are not directly related to the research questions, but because they were major topics of discussion during the interviews, they are discussed here.

First, several participants stated that they lacked an explicit purpose for the game. Since it is an active game requiring the player to interact with it, they were looking for an objective. Some said this could be related to the lack of an introduction when the game launches. Similarly, some interviewees said that they did not feel like they were doing anything, as if their actions were insignificant. In fact, the goal as stated in this research is to have a sort of contemplative game where the player visualizes the information. However, explaining more about the purpose of the game could increase immersion and user engagement.

Second, since the game requires users to play it in order to obtain the information, it must be usable. Some data on the usability of the game was collected during the interviews, showing that this tool is not ready to be used without the help of a system expert. This is a major point of interest for this type of tool, because if it is not sufficiently usable, users will not be able to understand the information represented in the game, thus completely missing its original purpose as a visualization tool.

Third, the usability of a video game is supported by the way it presents the game to the player. This means that the tutorial plays a major role in this task and makes it an essential component of the usability of the tool. In The Coding of Isaac, the tutorial did not cover all aspects of the game, such as accessing the details of problems on SonarCloud by clicking on a monster. So this is a major potential point of improvement for the tool.

Summary **additional conclusions**: Having an explicit goal for the game might be interesting for increasing the immersion and user engagement. In addition, the usability of the tool is a critical point in order to support its main goal of providing access to information. Therefore, the game tutorial is crucial as it introduces the player to the concepts and controls of the game.

12 Threats to validity

Despite the fact that everything was done to minimize bias, there are still several threats to validity in the evaluation process.

In the color scale test, where we tested participants' ability to rank different textures, according to Vegas et al. [2015], a better methodology, such as a cross-over design, might have reduced some of the threats to validity that we might have now. Because we did not have a huge amount of participants, we could not simply show one randomly selected set of textures to each participant, as the risk that the results would only be the opinion of a few respondents was too great. Instead, all 12 participants ordered both sets. The following is a list of potential threats to validity that our methodology raises according to the guidelines of Easterbrook et al. [2008]:

Construct validity Since the goal of this research is to determine if a game-based visualization tool could be useful to developers, asking them directly to give a trial-based evaluation of such a prototype is a good method for obtaining meaningful data.

Internal validity The test did not take order into account; the two sets of textures were shown in the same order to each participant, funky then rainbow. What should have been done was to randomly select the order of the two tests for each participant; in this way, we would have avoided a potential bias for the second test or, at least, gained insight into the extent to which the results are affected by a test taken just before. Moreover, the interviewer who conducted the interviews is a novice in interviewing. Some biases were likely introduced unintentionally or errors made due to his lack of experience. The interviews may also have been influenced by the interviewer's personality. However, the pilot tests served as training for the investigator. Therefore, this threat has been at least partially mitigated.

About **RQ 2.2**: The question used in the interviews directly asks participants who they would see using this tool, we have induced a positive bias because they are less likely to discuss who they would not see using this tool at all. This may have affected the results of that research question.

In addition, the results may not be generalizable because this research uses a specific prototype video game (with certain characteristics such as: offline, active, dungeon crawler). Thus, the results on game components as a relevant information vector might not work for other game genres. Or there could be another type or genre that would have yielded better results. Also, with respect to the game design, it includes only certain game components selected and mapped by the authors, other game components might produce different results.

For the sake of time, and to ensure that the same evaluation protocol was applied. Only one project was used for all interviews the *Brave* browser. This game and visualization may not yield the same results on other SonarCloud projects.

As for the coding and analysis of the data, this was done by humans and therefore there may be some bias. Although we have tried to counteract this by having the two authors work separately and then discussing their coding to reduce the impact of their individual biases.

External validity We believe that generalization of the results is possible at least to the variety of professional and student profiles that responded to our interviews. Apart from the

fact that all of them are people known to our supervisors or to friends. Hence, there might be some bias because they might just want to please their friends and be less likely to be completely honest. Another threat is the fact that only five (42%) of the 12 participants used a tool to manage technical debt which might suggest that it is either not a priority for them or that they are not very comfortable with it making potentially harming the results of this study.

The results of the evaluation might depend on the usability of the game, while this is not the main focus of the work. The quality of the UI/UX as well as the way the interviewees perceived the solution probably plays a role in the results.

The participants' backgrounds and cultures are not diversified enough, making generalization to all other cultures a potential challenge. The metaphors used by the authors may be rooted in their own cultural norms, which greatly threatens generalization.

Respondents provide an evaluation of the game, but this evaluation may be biased for four reasons: (1) The project, *Brave* browser, they tested is unfamiliar to them, (2) The interview environment may not be their workplace; (3) They are distracted by the interviewer asking questions while they are learning about the game; (4) They are learning about the tool at the same time they are asked to give feedback.

13 Future works

The results showed that game components have potential as information vectors in visualization tools and what their use cases might be in real life. In addition, this research discusses the strengths and weaknesses of The Coding of Isaac. Thus, there are two main possible approaches for future work. Expand the field by continuing to explore other aspects or conduct more in-depth research on specific elements presented here.

In fact, in order to explore the interest of the field, this research develops a specific instance of a dungeon crawler game called The Coding of Isaac for code quality visualization. Thus, future work on tools focusing on visualizing other types of topics would be interesting to see if the effectiveness of the game components as information vectors depends on it. Furthermore, the prototype is a specific game instance of a video game genre incorporating only some of its possible game components. It would therefore be interesting to have similar work on other types of games of different genres, or with other game components to try to find the best ones.

Moreover, this work reveals that some requirements of classical visualization tools could also be applicable for gamified tools. It would then be interesting to investigate to what extent the common good practices and pitfalls of traditional visualization are applicable to this new domain.

Furthermore, this research has drawn specific conclusions regarding the prototype presented here. Future work could be conducted on these by exploring the addition of gamification as requested. Similarly, future research could look at the multiple features requested by the interviewees. Additionally, the best way to test the potential use of a tool in real life is to release it as a real business tool and measure its success with real users. Therefore, it would be interesting to conduct research focused on evaluating The Coding of Isaac in this way to further test its value to practitioners.

Finally, regarding the evaluation of the prototype, it has only been conducted on a single software project. Further research on the generalization of the results of this work would be interesting. This can be done by conducting similar evaluations on other projects. Or, by analyzing the data from the SonarCloud projects and trying to determine if the projects are similar enough to generalize the results or if additional evaluations on other projects are needed.

14 Conclusion

In the literature on the visualization of code quality metrics, no research had been conducted to date on the potential of the video game as a medium for conveying information. This thesis explored this part of the field that could bring new possibilities regarding the design of visualization tools. Its two main objectives are to understand the potential for intuitiveness and precision of the video game medium in this use, and to explore the potential for use of a tool of this type. To find answers, a prototype of a brand new offline active game of the dungeon crawler type was developed. It was then evaluated through interviews with practitioners and academics. Due to the lack of previous work on this specific topic, this research tends to explore the topic in a general way in order to determine if further research would be of interest. Additionally, conclusions are drawn based on a single prototype that is a specific example of a video game genre that is not broadly representative of video game design possibilities, so the conclusions cannot be generalized. However, this work has drawn multiple conclusions about the potential uses of a brand new active offline dungeon crawler game as a tool for visualizing code quality.

This research revealed that multiple components of a video game can be used to visualize information. Indeed, as with traditional visualization tools, visual elements can be used. Furthermore, the concept of enemies in a video game can be effectively exploited as a metaphor for visualizing threatening elements to users, and the structure of the game environment can be used to represent the structure of the actual element being visualized, such as the directory structure of a software project.

In terms of adoption, a tool like The Coding of Isaac can be used by multiple roles (developers, managers, teachers, students) for multiple purposes (technical debt, tracking individuals' performance, introducing the concept of technical debt, engaging people in the fight against technical debt). It would be used on a regular basis, either at certain milestones or on a weekly or daily basis. Moreover, even if the playful aspect is interesting for visualization purposes as mentioned before, this aspect could be a barrier to its adoption by some people.

Finally, the design of such a tool could correspond to the design of a visualization tool, so all the research on the visualization domain would be applicable here. Proving this point would help define the best practices and pitfalls for the specific domain explored in this study.

We hope that the positive results concluded by this research will pave the way for further research on this topic, as it is a potential gateway to a multitude of highly innovative design possibilities for the visualization domain.

15 Bibliography

References

Mary J Allen. *Introduction to measurement theory*. Waveland Press, January 1979.

Gergo Balogh and Arpad Beszedes. CodeMetropolis — a minecraft based collaboration tool for developers. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, September 2013.

Laure Bedu, Olivier Tinh, and Fabio Petrillo. A tertiary systematic literature review on software visualization. In *2019 Working Conference on Software Visualization (VISSOFT)*, pages 33–44. IEEE, September 2019.

Woubshet Nema Behutiye, Pilar Rodríguez, Markku Oivo, and Ayşe Tosun. Analyzing the concept of technical debt in the context of agile software development: A systematic literature review. *Inf. Softw. Technol.*, 82:139–158, February 2017.

J. Bertin. *Sémiologie graphique*, volume 8. EHESS, Paris, 1967.

S Boeykens. Using 3D design software, BIM and game engines for architectural historical reconstruction. *Designing Together-CAADfutures*, pages 493–509, 2011.

Nathan Brewer. Going rogue: A brief history of the computerized dungeon crawl, 2016.

C. Cameron. Sonarcloud or sonarqube? - guidance on choosing one for your team, apr 2020. URL `https://blog.sonarsource.com/sq-sc_guidance`. Accessed on 2022-03-21.

A H Caudwell. Gource: visualizing software version control history. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 73–74. ACM, 2010.

Zadia Codabux, Byron J Williams, and Nan Niu. A quality assurance approach to technical debt. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2014.

S Deterding, D Dixon, R Khaled, and L Nacke. From game design elements to gamefulness: defining“ gamification”. In *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*, pages 9–15. 2011.

Evanthia Dimara and Charles Perin. What is interaction for data visualization? *IEEE Trans. Vis. Comput. Graph.*, 26(1):119–129, January 2020.

Shawn M Doherty, Joseph R Keebler, Shayn S Davidson, Evan M Palmer, and Christina M Frederick. Recategorization of video game genres. *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, 62(1):2099–2103, September 2018.

Ralf Dörner, Stefan Göbel, Wolfgang Effelsberg, and Josef Wiemeyer. *Serious games*. Springer, 2016.

Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

Francesca Arcelli Fontana, Vincenzo Ferme, Alessandro Marino, Bartosz Walter, and Pawel Martenka. Investigating the impact of code smells on system's quality: An empirical study on systems of different application domains. In *2013 IEEE International Conference on Software Maintenance*, pages 260–269. IEEE, 2013.

Karl-Ingo Friese, Marc Herrlich, and Franz-Erich Wolter. Using game engines for visualization in scientific applications. In *IFIP International Federation for Information Processing*, IFIP International Federation for Information Processing, pages 11–22. Springer US, Boston, MA, 2008.

Pooya Khaloo, Mehran Maghoumi, Eugene Taranta, David Bettner, and Joseph Laviola. Code park: A new 3D code visualization tool. In *2017 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, September 2017.

Foutse Khomh, Massimiliano Di Penta, and Yann-Gael Gueheneuc. An exploratory study of the impact of code smells on software change-proneness. In *2009 16th Working Conference on Reverse Engineering*, pages 75–84. IEEE, 2009.

H M Kienle and H A Muller. Requirements of software visualization tools: A literature survey. *4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 2–9, 2007.

Leonel Merino, Mohammad Ghafari, Craig Anslow, and Oscar Nierstrasz. CityVR: Gameful software visualization. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, September 2017.

David H Merwin and Christopher D Wickens. Comparison of eight color and gray scales for displaying continuous 2D data. *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, 37(19):1330–1334, October 1993.

Matthew B Miles, A Michael Huberman, and Johnny Saldaña. *Qualitative data analysis: A methods sourcebook*. Sage publications, 2018.

Alberto S Nuñez-Varela, Héctor G Pérez-Gonzalez, Francisco E Martínez-Perez, and Carlos Soubervielle-Montalvo. Source code metrics: A systematic mapping study. *Journal of Systems and Software*, 128:164–197, 2017.

R Oberhauser. An ontological perspective on the digital gamification of software engineering concepts. *International Journal on Advances in Software*, 9:207–221, 2016.

Roy Oberhauser and Carsten Lecon. Gamified virtual reality for program code structure comprehension. *Int. J. Virtual Real.*, 17(2):79–88, January 2017.

S Pérez, T Tubiana, A Imberty, and M Baaden. Three-dimensional representations of complex carbohydrates and polysaccharides-SweetUnityMol: A video game-based computer graphic software. *Glycobiology*, 25(5):483–491, 2015.

F Raab. CodeSmellExplorer: Tangible exploration of code smells and refactorings. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, September 2012.

Guido Reina, Hank Childs, Krešimir Matković, Katja Bühler, Manuela Waldner, David Pugmire, Barbora Kozlíková, Timo Ropinski, Patric Ljung, Takayuki Itoh, Eduard Gröller, and Michael Krone. The moving target of visualization software for an increasingly complex world. *Comput. Graph.*, 87:12–29, April 2020.

Simone Romano, Nicola Capece, Ugo Erra, Giuseppe Scanniello, and Michele Lanza. On the use of virtual reality in software visualization: The case of the city metaphor. *Information and Software Technology*, 114:92–106, October 2019.

R M Ryan and E L Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *Am. Psychol.*, 55(1):68–78, January 2000.

T Sandusky. *Plato's Evil Closet: Using a Nonviolent Dungeon Crawler to Address White Heteropatriarchy in Video Games (Doctoral dissertation)*. Mills College, 2018.

O. Schmitt. List of all public projects on sonarcloud using api, oct 2020. URL `https://community.sonarsource.com/t/list-of-all-public-projects-on-sonarcloud-using-api/33551/4`.

Samuel Silva, Joaquim Madeira, and Beatriz Sousa Santos. There is more to color scales than meets the eye: A review on the use of color in visualization. In *2007 11th International Conference Information Visualization (IV '07)*. IEEE, July 2007.

K S Tekinbas and E Zimmerman. *Rules of play: Game design fundamentals*. MIT press, 2003.

Daniel Turner, III. Qualitative interview design: A practical guide for novice investigators. *The Qualitative Report*, May 2010.

Sira Vegas, Cecilia Apa, and Natalia Juristo. Crossover designs in software engineering experiments: Benefits and perils. *IEEE Transactions on Software Engineering*, 42(2):120–135, 2015.

K Werbach and D Hunter. *For the win: how game thinking can revolutionize your business*. Wharton Digital Press, 2012.

P Wessa. Free statistics software, office for research development and education, 2022. URL `https://www.wessa.net/`.

Richard Wettel and Michele Lanza. Visualizing software systems as cities. In *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, June 2007.

Richard Wettel and Michele Lanza. Codecity: 3d visualization of large-scale software. In *Companion of the 30th international conference on Software engineering*, pages 921–922, 2008.

Enno Wulff. I have a dream: Code visualization, Dec 2020. URL `https://blogs.sap.com/2020/12/29/i-have-a-dream-code-visualization/`.

Alfa Yohannis and Yulius Prabowo. Sort attack: Visualization and gamification of sorting algorithm learning. In *2015 7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games)*. IEEE, September 2015.

Nico Zazworka, Michele A Shaw, Forrest Shull, and Carolyn Seaman. Investigating the impact of design debt on software quality. In *Proceedings of the 2nd Workshop on Managing Technical Debt*, pages 17–23, 2011.

A Appendix

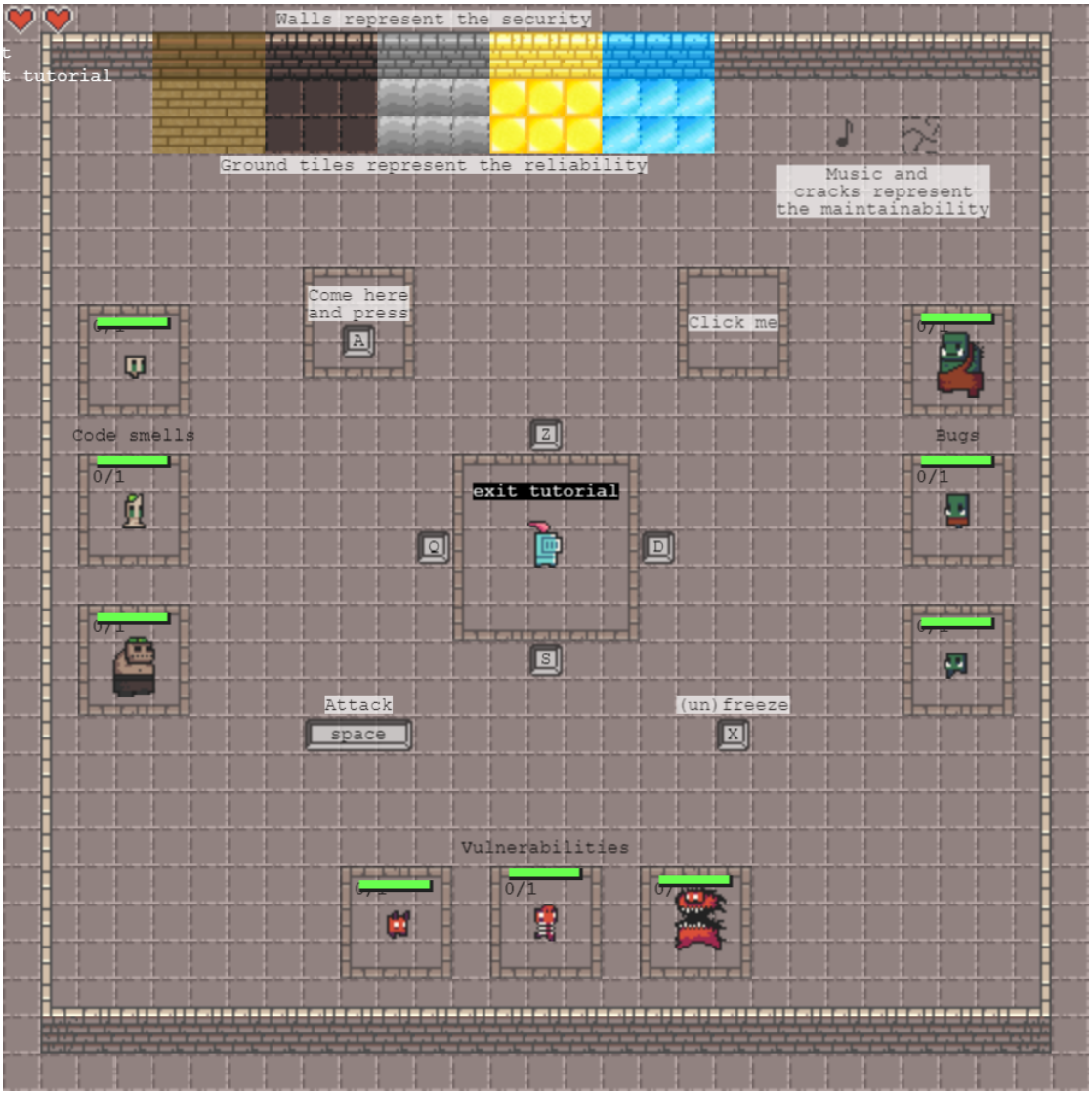

Figure 46: Tutorial room of the first design.

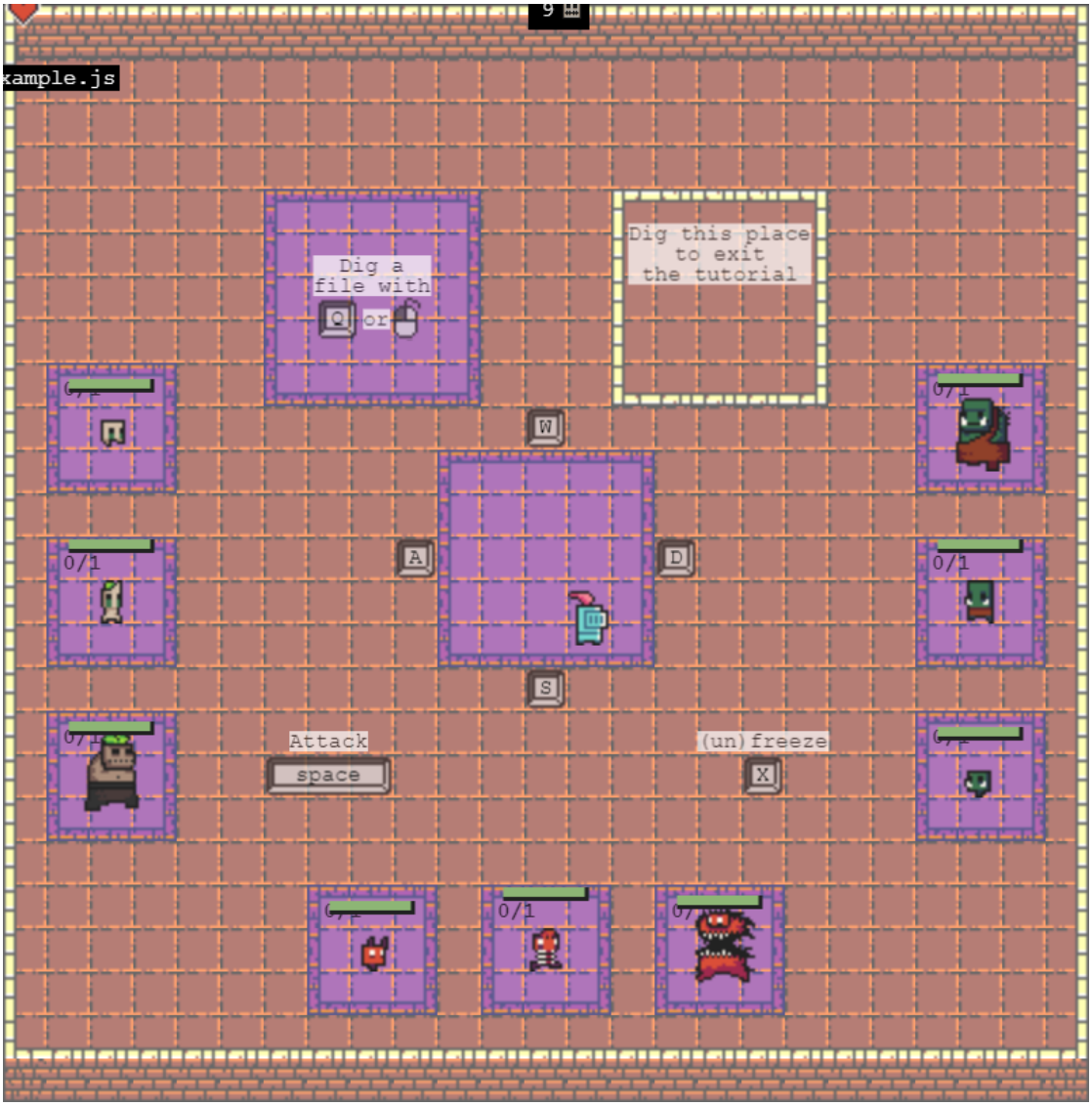

Figure 47: Tutorial room after the pilot interviews.

Criteria|Framework	Phaser 3	MelonJS	ImpactJS	KaboomJS
Community	Github discussions, Discord, forums. All community channels seem quite active with new posts and topics every day.	Forums not very active, Github discussions neither, Discord, Gitter	Abandoned Forums	Not very active Github Discussions, Abandoned Forums, Discord.
Documentation	Detailed documentation available with >1770 code examples	Detailed documentation available	Not very complete documentation, paid online courses available to get started with it.	Very recent and incomplete.
Stability	the 3.0 version of Phaser came out in Feb 2018. Since then the latest stable version is 3.55.2 which came out in May 2021.	The framework is quite mature. latest version is 10.9.0 (May 2022). Development started in 2011.	Last commit on Github was made in February 2022.	Very new framework that came out in 2021. Mostly unstable.
Maintenance status	The github repo is updated with new commits every day. A beta version 3.60 is currently in preparation. And also the next big iteration of the framework called Phaser 4.	Active development: New commits nearly everyday but only one person is working on it.	The project seems to have been abandoned since 2018 when it got open sourced.	Active development: commits multiple times per week.
Popularity	31.8k Github stars	4k Github stars	1.8k Github stars	1.6k Github stars
Tutorials	Lots of tutorials available on YouTube even one walk-though detailing how to build a basic dungeon crawler.	Two official step by step tutorial but Very few tutorials on YouTube. Moreover, they are older than two years old.	Old tutorials available on YouTube (3-9 years old).	Quite a few tutorials available on YouTube for such a recent project.
Learning curve	Made easy by the amount of resources on the framework	Quite steep because of the lack of tutorials and community resources.	Steep because of the lack of tutorials, community resources and support.	Very steep because it is quite new.
Compatibility	All-in-one framework with community plugins for missing features. No other libraries would be required.	Seems to be self sufficient, but no community plugins available.	Seems to be all-in-one	Lacks a lot of features yet because it is quite new. More a library than a framework.
Flexibility	Can get quite low level meaning there is no problem in connecting external data easily.	Very low level, quite flexible.	Very low level, quite flexible.	Very low level, very flexible.

Table 13: Comparison of some JavaScript game frameworks for web browser game development.

INFORMED CONSENT FORM

Full Title of Project: **Yokai Watch - The Coding of Isaac**

Name of Principal Investigator: Aniss Grabsi

Please initial box

1. I confirm that I have read and understood the subject information sheet dated 2022-03-08 for the above study and have had the opportunity to ask questions which have been answered fully ☐

2. I understand that my participation is voluntary and I am free to withdraw at any time, without giving any reason, without my medical care or legal rights being affected. ☐

3. I give permission for these individuals to collect and process the data as explained in the information sheet ☐

4. I agree to take part in the above study. ☐

Name of the participant signature date

Name of the investigator signature date

The Coding Of Isaac

Have you ever dreamed of literally killing the bugs in your code? 🔥

Let us introduce you to The Coding of Isaac.

This game aims to give you an overview of the quality of your code base by playing against it.

We would like you to fill out this short form so we can learn a little more about you before the interview.

Thank you for your participation!

If you have any questions you can contact us here:
Anthony Bayet | anthony.bayet@student.unamur.be or lpf713@mail.usask.ca
Aniss Grabsi | aniss.grabsi@student.unamur.be or qcr947@mail.usask.ca

*Obligatoire

1. What is your current occupation? *

Une seule réponse possible.

- ◯ Student *Passer à la question 2*
- ◯ Employee *Passer à la question 4*

Student information

2. What year of study are you in? *

Une seule réponse possible.

- ◯ 1st year of Bachelor
- ◯ 2nd year of Bachelor
- ◯ 3rd year of Bachelor
- ◯ 1st year of Master
- ◯ 2nd year of Master
- ◯ Autre : _____

3. Do you have some experience working in the industry? *

Une seule réponse possible.

- ◯ I have never worked in the industry.
- ◯ less than 2 year
- ◯ 2 - 5 years
- ◯ 6 - 10 years
- ◯ 11 - 20 years
- ◯ > 20 years

Passer à la question 8

Employee information

4. Number of employees in your company *

Une seule réponse possible.

- ◯ 1 - 99
- ◯ 100 - 999
- ◯ 1000 - 4999
- ◯ > 5000
- ◯ Don't know
- ◯ Autre : _____

5. How many years of experience do you have in software development? *

Une seule réponse possible.

- ◯ 0 - 2 years
- ◯ 3 - 5 years
- ◯ 6 - 10 years
- ◯ 11 - 20 years
- ◯ > 20 years

6. What is your role in your department/division? *

Une seule réponse possible.

- ◯ Software developer
- ◯ Software architect (Design)
- ◯ Requirements Analyst
- ◯ Quality Analyst / Testing
- ◯ Project Manager
- ◯ User Interface Development
- ◯ Autre : _____

7. My company takes technical debt and quality very seriously. *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

About yourself

8. How much do you play video games? *

Une seule réponse possible.

- ◯ Everyday
- ◯ Once or twice a week
- ◯ Once or twice a month
- ◯ Once or twice a year
- ◯ I never play video games

9. How would you evaluate your knowledge of technical debt (code smells, bugs, vulnerabilities)? *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Very little knowledge	◯	◯	◯	◯	◯	◯	◯	Very strong knowledge

10. What tools do you generally use to manage/visualize technical debt *

Plusieurs réponses possibles.

- [ ] Code city
- [ ] SonarQube / SonarCloud
- [ ] Embold
- [ ] I don't use any.
- [ ] Autre : _____

End of interview questionnaire

Thank your for participating in this study.

Before you go, please fill out this questionnaire to give us some global insights on how the interview went.

*Obligatoire

1. I think this is important to play this game because it can help a developer understand the code quality of a project. *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

2. I believe playing this game could be beneficial to me. *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

3. While I was playing this game, I was thinking about how much I enjoyed it. *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

4. This game did not hold my attention at all. *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

5. I believe I was skilled at this game. *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

6. I think playing this game could help me understand the issues with the code *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

7. I am satisfied with my performance at this game *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

8. I enjoyed playing this game very much. *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

9. I would be willing to play again because it has some value to me. *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

10. This game was fun to play *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

11. After playing this game for a while, I felt competent. *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

12. I think that playing this game is useful for a developer. *

Une seule réponse possible.

	1	2	3	4	5	6	7	
Strongly disagree	◯	◯	◯	◯	◯	◯	◯	Strongly agree

The Coding Of Isaac

Interview information sheet

You are invited to participate in a research study that explores how effective a video game would be in the context of software visualization. After reading this information sheet, you will have the opportunity to ask any questions you may have. You will be separately requested to provide consent for participating in the study.

Purpose of the research The goal of this study is to determine if the video game medium is an effective way to represent the state of the code and technical debt of an IT development project.

Description of the process Subjects will be requested to provide their opinion on the game they'll play during the interview. The interview is expected to take approximately 30-40 minutes to be completed.

Procedures for collecting research data During the interview, the participant will be asked to describe their thoughts during the game and answer a few questions. At the end of the interview, the participant will be asked to complete a questionnaire. A microphone will be used to record the interview and the computer screen will also be recorded.

Potential risks and benefits of participation You will not receive any direct benefit from participating in the study. The procedures and methods used during this study do not involve health risks, social risks, financial risks and risks relating to personal data breaches.

Data confidentiality, processing and storage All personal data collected during the study will be processed in compliance with the EU's General Data Protection Regulation (GDPR) and the data protection laws of Canada. Data about individual participants will not be disclosed to external persons. The data used in this study will be obtained through an interview, a questionnaire, a microphone and a screen capture. The data will be accessible only to the researchers on a Google Drive account (owned by Aniss Grabsi) and the processed data will be stored on the same Google Drive account for a maximum of 2 years.

Protecting the participants' privacy in research papers/publications The research materials and data collected during the study will be retained by Aniss Grabsi for a maximum of 2 years, after which they will be securely destroyed. The data will not include personal information on the participant. Prior to analysis and dissemination, all the data will be completely anonymized. Raw data will not be disclosed to any third parties.

Funding sources None

Voluntary participation Participation in this study is entirely voluntary, and you have the right to withdraw from the study at any time, either permanently or for a temporary period. Refusal to participate or discontinuing participation at any time will not affect any treatment you may later receive. The data collected on the subjects to the point of withdrawal will also be removed from the database.

Privacy protection in the context of research papers and communicating about the study No confidential information or any type of information that might enable the identification of the

subjects will be collected. The questionnaires are completely anonymous. Data will be processed only on researcher's computers and published only in aggregated form. Participants will receive a summary of the findings after the study has been completed. The results can be reported in the form of a scientific publication.

Inquiries Please direct all inquiries about the study to one of the following researchers

Researchers' contact details

Name: Aniss Grabsi
Unit: Department of Computer Science, University of Namur (UNamur)
Email address: aniss.grabsi@student.unamur.be or qcr947@mail.usask.ca

Name: Anthony Bayet
Unit: Department of Computer Science, University of Namur (UNamur)
Email address: anthony.bayet@student.unamur.be or lpf713@mail.usask.ca